WDC's 65C816 MICROPROCESSOR
Facts, Myths and Why You Should Use It

In 1984, the Western Design Center (WDC), developers of the popular 65C02 eight-bit microprocessor, released a 16-bit successor called the 65C816. The 65C816 was the result of a consultation between Apple Computer and William D. Mensch, founder of WDC, in which Apple wanted a processor with new features, but also with the ability to execute the existing 6502 code base without modification. The target system for this new processor would be the Apple IIGS.

Subsequent to the adoption of the 65C816 by Apple, Nintendo designed and produced the Super Nintendo Entertainment System (SNES), powered by a core that incorporated a modified form of the 65C816. Both systems were well-received—the SNES was a best-seller, and demonstrated that the 65C816 was a capable processor.

The 65C816, officially known as the W65C816S (the 'S' referring to its static core), is in current production as of this writing (June 2021), available as a discrete part in DIP40, PLCC44 and QFP44 packages, and as a soft-core in Verilog for use in custom hardware. WDC's W65C256S is a system-on-a-chip whose core is the 65C816. Discrete 65C816s, as well as the W65C256S, are available from stock through electronics distributors.

Despite its maturity, or perhaps because of it, much misinformation has been circulated about the 65C816. As a result, many would-be builders of hobby computers have shied away from the processor. This is quite unfortunate, as a system designed around the 65C816 can offer capabilities that are out-of-reach with the 65C02. *Ergo* what follows is a list of common misconceptions concerning the 65C816 and what the facts are.

- **It's not 6502-compatible.**

  FALSE!

  At power-on or following reset, the 65C816 starts in "emulation" mode, meaning it mostly looks like a 65C02 in software. The "mostly looks like" modifier refers to the fact that unlike the eight-bit processors, the 65C816 has no undefined opcodes. Whereas undefined opcodes behave as `NOP`s in the 65C02, or do strange things in the NMOS processors (including crashing the processor), all 256 opcodes in the 65C816 are documented instructions. An interesting side-effect of the 65C816's design is the instructions that are new to the 65C816, such as `PEA` and `TSC`, are usable in emulation mode as well as native mode, although with limited utility in some cases.

  The bottom line is the 65C816 is 6502-compatible as long as your program does not use the undefined opcodes or rely on undocumented behaviors present in the eight-bit processors. In the same vein, the 65C816 is 65C02-compatible as long as the Rockwell extensions (`BBR`, `BBS`, `RMB` and `SMB`) are not used. Those opcodes have been reassigned to new, and generally more powerful, 65C816-specific instructions.

- **Native mode is all 16-bit.**

  FALSE!

  Switching the 65C816 to native mode operation does not force you to process everything 16 bits at a time. The only user-accessible registers that are fixed to 16 bits in native mode are the direct page (zero page) pointer (DP) and stack pointer (SP).

  The "width" of the accumulator and index registers is determined by clearing or setting two bits in the status register using simple assembly language instructions. The 65C816 always comes out of reset with those bits set, which means the registers are eight-bits-wide by default.

  If you are hesitant about 16-bit processing, leave the registers set to eight bits after switching to native mode and pretend your 65C816 is just a 65C02 on steroids. However, don't be afraid to try out some 16-bit code.

- **Programming in "native" mode is hard.**

  FALSE!

  This nonsense seem to constantly pop up, often promulgated by someone who has likely never seen a properly-written 65C816 assembly language program.

  65C816 assembly language mostly looks just like 65C02 assembly language, but with the availability of some new and powerful instructions. Whether in emulation or native mode, all the familiar 65C02 instructions are present (excepting the aforementioned Rockwell extensions), look the same as before, and do the same things they do in the 65C02.

  What native mode does for you is open the door to 16-bit processing, the ability to easily address many megabytes of RAM, ROM and I/O with succinct code, and the ability to fully utilize processor features that are "turned off" or "crippled" in emulation mode. For example, in native mode, SP is a 16-bit register, making it possible to assign any address in the first 64 kilobytes (KB) of memory as the top of the stack. The start of zero page, referred to as "direct page" in 65C816 parlance, may likewise be set to any address in the first 64 KB by changing the value of DP.

  In particular, the ability to relocate direct page gives rise to some very useful capabilities. For example, direct page can be relocated to the stack, giving each function (subroutine) its own direct page that vanishes once the function has finished its work. This capability is facilitated by the 65C816's ability to copy SP directly to the accumulator, perform arithmetic on it and copy the result back to SP to change the 65C816's notion of where the stack is located.

As this is not a programming manual, we will not discuss at length all of the 65C816's capabilities in native mode (a substantial and highly-recommended book has already been written about it). What we will say is this: native mode operation actually makes writing high-performance software much easier than with the eight bit processors. Often, you will be able to do things in a few steps that would take reams of 65C02 instructions to accomplish—or you will be able to accomplish tasks that are simply not practical to carry out with the 65C02.

- **Memory can only be accessed as banks.**

  FALSE!

  This nonsense also seems to constantly be repeated, usually by someone with little-to-no knowledge of how the 65C816 operates.

  Program segments are confined to a single bank of the programmer's choosing, which is an artifact of maintaining 65C02 compatibility. Even this is not as much a limitation as it would seem. Although unusual, a program's size can greatly exceed 64 kilobytes (the size of a bank) and bank boundaries may be crossed by using long jumps and long subroutine calls—it just takes a little planning and adroit programming.

  Direct page, the stack and the hardware vectors are "hard wired" to bank $00, again an artifact of 65C02 compatibility. However, since direct page and the stack are relocatable, they being confined to bank $00 is not the limitation it would seem as well.

  Data access routines can be made bank-aware or bank-agnostic at the programmer's discretion—there are arguments in favor of both programming models, and both may be used in the same program without any loss in efficiency. It is possible to build data structures that span hundreds of kilobytes of memory (or even megabytes) and access that data with succinct code, resulting in the 65C816 seeing its address space as linear.

- **A 65C816 system is difficult to design.**

  FALSE!

  The 65C816 has the same address and data bus as the 65C02, with some operating differences that give the 65C816 its additional capabilities, such as 24-bit addressing. During Ø2 low (Ø2 referring to the system clock), the 65C816 emits the A16-A23 address component—colloquially referred to as the "bank address"—on the data bus. When Ø2 goes high, the 65C816 stops emitting the bank address and the data bus becomes the data bus.

  You are not obliged to do anything with the bank address, in which case the 65C816 will confine addressing to the range $0000-$FFFF, same as the 65C02.

If you are interested in implementing extended memory, that is, memory beyond 64KB, you can add the bank-latching hardware into your project and increase the 65C816's addressing range to as much as 16 megabytes. An example circuit for latching the bank address is in the official 65C816 data sheet, and methods for generating A16-A23 have been published for use with programmable logic. The author of this paper has built a system with 128 KB of RAM, running at 16 MHz and using only discrete logic.

The 65C816 has some additional output signals that are useful in managing the system and avoiding spurious chip selects and other undesirable behaviors that were characteristic of the NMOS 6502. As well, you are not obliged to use any of these signals, although neglecting them may prove to be unwise.

With the use of modern programmable logic, a 65C816 system with megabytes of RAM is as easy to design and build as a 65C02 system with 64 kilobytes—and far more powerful!

- **A 65C816 system costs more to build.**

  FALSE!

  The same parts you would use to build a basic 65C02 system can be used to build a basic 65C816 system. If you decide to build your 65C816 system with a lot of RAM then yes, cost will go up in proportion to the amount of RAM installed. That said, the result will be a system with much more capability.

- **The 65C816 is slower in native mode than the 65C02.**

  FALSE!

  Both processors support the same official maximum Ø2 rate, which is 14 MHz. However, when processing 16 bits at a time, the 65C816 will effectively execute register-based instructions, such as TXA or ASL twice as fast as the 65C02 will at the same Ø2 rate. For example, when both the accumulator and index registers have been set to 16 bits, the 65C816 will execute TXA in the same two Ø2 cycles as will the 65C02, yet will copy two bytes in the same time required for the 65C02 to copy one byte.

  Similarly, when performing 16-bit fetches and stores, the 65C816 will substantially outperform the 65C02, with a performance gain of 60-80 percent for functionally identical code segments. For example, in order to store a 16-bit value to absolute memory, the 65C02 must do it a byte-at-a-time, requiring two, for example, STA instructions at a cost of four cycles each. In contrast, with the accumulator set to 16-bits and the 16-bit value already loaded, the 65C816 can do with a single STA instruction at a cost of five cycles total what the 65C02 does at a cost of eight cycles. In repetitive load/store sequences, such as processing two multi-byte numbers, that difference can have a major effect on system throughput.

While on the topic of loading and storing from/to memory, the 65C816 has two instructions that are able to copy raw data from one location to another at the rate of nearly 140 kilobytes per second per MHz Ø2 clock speed. Put into perspective, a 65C816 system operating at 14 MHz and using the `MVN` (block copy forward) or `MVP` (block copy backward) instruction is able to copy eight kilobytes of RAM from one place to another in approximately 0.004 seconds, using surprisingly-little code. The `MVN` and `MVP` instructions far out-perform anything that can be done with conventional programming when it comes to mass-copying data in memory.

The only area in which the 65C816 is "slower" than the 65C02 is in interrupt response when running in native mode, a byproduct of the former having an extra register to manipulate. Whatever performance disparity this may represent is not of any great importance when one considers that the 65C816 in native mode much can do more than the 65C02 in the same number of clock cycles when 16-bit fetches, stores and arithmetic/logical operations are performed. If any performance deficiencies do occur with a 65C816 system it will likely be due to inefficient programming.

As well, there are other myths in circulation about the 65C816, however the above has covered the important ones. Yes, the 65C816 is "different" than the 65C02, but that "difference" is trivial and all for the good. Please give it a try!