A look back at memory models in 16-bit MS-DOS

devblogs.microsoft.com/oldnewthing/20200728-00

July 28, 2020

Raymond Chen

In MS-DOS and 16-bit Windows programming, you had to deal with memory models. This terms does not refer to processor architecture memory models (how processors interact with memory) but rather how a program internally organizes itself. The operating system itself doesn't know anything about application memory models; it's just a convenient way of talking about how a program deals with different types of code and data.

The terms for the memory models came from the C compiler, since this informed the compiler what type of code to generate. The four basic models fit into a nice table:

		Data pointer size	
		Near	**Far**
Code pointer size	**Near**	Small	Compact
	Far	Medium	Large

The 8086 used segmented memory, which means that a pointer consists of two parts: A segment and an offset. A *far pointer* consists of both the segment and the offset. A *near pointer* consists of only the offset, with the segment implied.

Once you had more than 64KB of code or more than 64KB of data, you had to switch to far code pointers or far data pointers (respectively) in order to refer to everything you needed.

Most of my programs were *Compact*, meaning that there wasn't a lot of code, but there was a lot of data. That's because the programs I wrote tended to do a lot of data processing.

The *Medium* model was useful for programs that had a lot of code but not a lot of data. User interface code often fell into this category, because you had to write a lot of code to manage a dialog box, but the result of all that work was just a text string (from an edit box) and some flags (from some check boxes). Many computer games also fell into this category, because you had a lot of game logic, but not a lot of game state.

MS-DOS had an additional memory model known as *Tiny*, in which the code and data were all combined into a single 64KB segment. This was the memory model required by programs that ended with the `.COM` extension, and it existed for backward compatibility with CP/M. CP/M ran on the 8080 processor which supported a maximum of 64KB of memory.

Far pointers could access any memory in the 1MB address space of the 8086, but each object was still limited to 64KB because pointer arithmetic was performed only on the offset portion of the pointer. *Huge* pointers could refer to memory blocks larger than 64KB by adjusting the segment whenever the offset overflowed.[1] Pointer arithmetic with huge pointers was computationally expensive, so you didn't use them much.[2]

You weren't limited to the above memory models. You could make up your own, known as *mixed model programming*. For example, you could say that most of your program is *Small* memory model, but there's one place where you need to access memory outside the default data segment, so you declared an explicit far pointer for that purpose. Similarly, you could define an explicitly far function to move it out of the default code segment.

The memory model specified the default behavior, so if you called, say, `memcpy`, you got a version whose pointer size matched your memory model. If you had a *Small* or *Medium* model program and wanted to copy memory that was outside your default data segment, you could call the `_fmemcpy` function, which was the same as `memcpy` except that it took far pointers. (If you used the *Compact* or *Large* memory model, then `memcpy` and `_fmemcpy` were identical.)

One of my former colleagues is back in school and was talking with his (younger) advisor. Somehow, the topic turned to the 8086 processor. My colleague and his friend explained segmented addressing, near and far pointers, how pointer equality and comparison behaved,[3] the mysterious A20 gate, and how the real mode boot sector worked. "The look of horror on his face at how segment registers used to work was priceless."

I quickly corrected my colleague. "'Used to work'? They still work that way!"

Bonus chatter: You can see the remnants of 16-bit memory models in the macros `NEAR` and `FAR` defined by `windows.h`. They don't mean anything any more, but they remain for source code backward compatibility.

[1] In MS-DOS, huge pointers operated by putting the upper 16 bits of the 20-bit address in the segment and putting the remaining 4 bits in the offset. The offset of a huge pointer in MS-DOS was always less than 16.

[2] In 16-bit Windows, there was a system function called `hmemcpy`, which copied memory blocks that could be larger than 64KB. And now you know what the `h` prefix stood for.

[3] Segmented memory is a great source of counterexamples. For example, in a segmented memory model, you could have a pointer *p* to a buffer of size *N*, and another pointer *q* could satisfy the inequalities

```
p <= q && q <= p + N
```

despite not pointing into the buffer.

This is one of the reasons why the C and C++ programming languages do not specify the result of comparisons between pointers that do not point to elements of the same array.

Raymond Chen

Follow