The Intel 80386, part 9: Stack frame instructions

devblogs.microsoft.com/oldnewthing/20190130-00

January 31, 2019

Raymond Chen

There are a pair of specialized instructions for creating and tearing down stack frames.

```
    ENTER   i16, 0      ; push ebp
                        ; mov ebp, esp
                        ; sub esp, (uint16_t)i16
```

The `ENTER` instruction sets up a stack frame for a new subroutine. It combines three instructions into one, so that what used to be encoded in eight bytes (1 + 2 + 5) is now encoded in four. However, even on the 80386, the combination instruction executes more slowly than the three component instructions, so this was always a size optimization, not a speed optimization.

```
    LEAVE               ; mov esp, ebp
                        ; pop ebp
```

The `LEAVE` instruction tears down the stack frame by reversing the effects of the `ENTER` instruction. This is a one-byte instruction that replaces two instructions that together require three bytes (2 + 1), so it is a size optimization. But it also executes faster than the two instructions it replaces, so it is also a speed optimization.

Modern compilers avoid the `ENTER` instruction but keep the `LEAVE` instruction.

Bonus chatter: What's with the second operand of the `ENTER` instruction?

In C code, the second operand is always zero because C doesn't support lexically-nested procedures with inherited stack frames. So in practice, you will always see zero as the second parameter.

The second parameter can go up to 15, and it represents the number of additional values pushed onto the stack after pushing *ebp*.

```
    ENTER   i16, n      ; push ebp
                        ; sub ebp, 4   ⎱ n times
                        ; push [ebp]   ⎰
                        ; mov ebp, esp
                        ; sub esp, (uint16_t)i16
```

This means that the `ENTER` instruction can read as many as fifteen 32-bit values from memory and can write as many as sixteen 32-bit values to memory. That's a lot of memory access for a single instruction.

Next time, we'll look at atomic operations and memory alignment.

Raymond Chen

Follow