

Re: assembly language and reverse engineering

Re: assembly language and reverse engineering

Source: <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2007-12/msg00186.html>

- *From:* "cr88192" <cr88192@xxxxxxxxxxxx>
 - *Date:* Sat, 8 Dec 2007 00:17:53 +1000
-

"Herbert Kleebauer" <klee@xxxxxxxxxx> wrote in message
<news:47590E97.6DBB54C8@xxxxxxxxxxxx>

cr88192 wrote:

"Herbert Kleebauer" <klee@xxxxxxxxxx> wrote in message

Why does this myth never die? Please explain what you can learn in Windows assembly programming what you can't learn when doing assembly programming in DOS (and we are not speaking about the Windows API but about assembly programming).

the point is not what you can't learn, but what you have to learn in the process.

And you have far less to learn to get your first working DOS program than windows program. You can even use a hex editor, insert a single byte 0xc3 and save it as demo.com and you have made your first program.

conjecture, mostly.

the point is that one will be learning windows, not DOS.

to understand com files has little to do with windows, or to the task of learning how to disassemble/reverse engineer windows code.

Re: assembly language and reverse engineering

dos would force you to learn more than windows or linux (for example, one has to learn about segment registers, ...).

Why do you need something to know about segment registers when programming in DOS. There is a flat binary executable format call com format. The only restriction is the limit of 64 kbyte, but that's surely no problem for "learning" assembly programming.

as I have typically seen, usually the first few steps in a com file are setting things up, ...

none the less, to understand this, one still ends up also having to understand things like segment registers, and other concepts that cease being all that relevant in 32-bit userspace.

learning real mode programming, would take longer, and teach things that, unless one is doing OS coding, are not terribly useful...

This is nothing but a myth, the opposite is true. Please show my any proof of your claim.

what "is" of value, unless your goal is learning theory, and not doing?...

learning dos to learn asm, is good for if you want to gernerally learn asm, but not if one simply wants to disassembler or reverse engineer things.

for sake of a specific task, a more direct route is likely better.

for example, in 'windows' do we typically use registers and interface with the OS via interrupts?...

no (though possible, it is generally not done this way).

usually, we interface with the C RTL, and thus it is more prudent to know the specifics of STDCALL and CDECL, and the API, rather than knowing about an interrupt-driven interface.

Re: assembly language and reverse engineering

Assembly programming means to generate an instruction stream for the CPU and not for the OS. And the CPU is the very same whether you use Windows, Linux or DOS as an OS.

but, real mode is real mode, and pmode is pmode.
the way we use the CPU is different,

There is no difference in the add, mov, jmp and all the other user mode instructions. The only difference is, that in real mode you can use also instructions (including the ones which switch to protected mode) which a non privileged user (which an assembly programming beginner should be) isn't allowed to use.

there are differences.
the primary difference is the register sizes/names;
the secondary difference is the mod/rm behavior.

we can assume that a newb, unless they are concerned with actually learning all this stuff, it is not all that useful to expect them to correctly understand such concepts as register sizes and namings, or that memory addressing behaves differently.

and so is the mod/rm structure.

The mod/rm structure has nothing to do with DOS or Windows, it has also nothing to with the CPU being in Real or Protected Mode. There are always both forms (16 and 32 bit) available and it's completely up to the programmer which one he chooses (but in 32 bit Protected Mode you better don't choose the 16 bit addressing modes).

it does matter.

DOS is almost always 16-bit real mode (DPMI and/or manually going into PM being ignored).
modern windows is almost always used as 32-bit PM (likewise, 16 bit PM is being ignored, as is running stuff in a DOS box, which does not usefully address the task at hand...).

Re: assembly language and reverse engineering

learning RM first could leave one thinking they can use only SI, DI, BX,
or
BP in memory references...

Why? There are exactly the same addressing modes in Real as in Protected
Mode available.

no, they are different.

in RM we have:

SI+disp

DI+disp

BX+disp

BP+disp

BX+SI+disp

BX+DI+disp

BP+SI+disp

BP+DI+disp

these are, in short, our only real options (ignoring overrides, which
fundamentally change how memory operations work, namely that full 32 bit
addresses are used in calculations).

now, in 32-bit PM, the rule is this:

$\text{breg} + \text{sc} * \text{ireg} + \text{disp}$

we have all sorts of combinations, like the example given before.
we can regard this as a very different construction.

now, a person could learn in RM, and later move to PM.

they may subtly fail to realize that the way memory addresses are used is
different, making incorrect use of registers, or being faced with confusion
upon encountering code using proper 32 bit forms.

much the same as how long mode is very different from what has come before
(more regs, different register usage conventions, ...).

take something even as simple as function entry:

in RM we see something like:

Re: assembly language and reverse engineering

Re: assembly language and reverse engineering

```
_foo:  
push bp  
mov bp, sp  
push si  
  
mov si, [bp+4]  
mov ax, [si]  
....  
  
pop si  
pop bp  
ret
```

and, to do the same thing in 32 bit PM:

```
_foo:  
push ebp  
mov ebp, esp  
  
mov eax, [ebp+8]  
mov eax, [eax]  
....  
  
pop ebp  
ret
```

as can be seen, these are, different...

how much more different would it be if these functions actually did something?...

and we expect people to just gloss over these kind of differences?...

or, at least until they come to understand things like:
`mov eax, [ecx+edx*4]`

Why do you think the above instruction is extracted from a 32 bit Protected Mode program written for Windows and not from a source code for a 16 bit Real Mode com program executed in DOS.

because, this is a 32-bit memory access, and is thus not applicable to DOS. we are forced into using 32 bit registers, and not 16 bit ones. this requires special provisions (for example, because 386-era processors did not implicitly zero-extend regs, ...).

Re: assembly language and reverse engineering

we may see this kind of thing, for example, in big real mode, but big-real is its own thing...

It's nothing but a CPU instruction which is neither specific to Windows or DOS nor for Protected or Real Mode.

but, the CPU is modal, and these modes make a notable impact on the way that the CPU is used and behaves.

for example, we write very different code for real mode, protected mode, and long mode.
they are hardly the same thing.