

## Re: ASM noob – couple of questions

---

*Source:* <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2006-03/msg00333.html>

---

- *From:* o//annabee <fack@xxxxxxxxxxxxxxxx>
  - *Date:* Fri, 10 Mar 2006 19:54:44 +0100
- 

På Fri, 10 Mar 2006 04:13:15 +0100, skrev Frank Kotler <fbkotler@xxxxxxxxxxxx>:

Betov wrote:

I, for one, don't know. DirectX is the COM thingie, right? (.com = simple, COM = COMplicated) Beth gave a rundown on it some time ago. No wonder John can't figure it out! There's a kind of "linked list" from one "interface" to the next, right?

COM is a simple prinsiple, just like oop (it really is oop) that leads fast to complex and unreadable code. Plus the many strange names given to its parts that confuses one beyond believe.

But using directx, or COM is n prinsiple like this :

1)  
call some api, to get back a pointer to a socalled COMObject.

(  
Each object is identified, and obtained, either by a socalled GUID, or by calling a spesificly exported function for it. For Direct3D theres a function called "Direct3DCreate9" to create the main Direct3D object, but it can also be created via a call to "Ole32.CoCreateInstance", but then you have to spesify its GUID, which is a Global Unique/Universal Identifier DShit.  
)

The pointer you get back, points to a structure (a call table) of socalled interfaces. An interface is nothing but an offset to a "pointer to a procedure". You are not given directly access to this structure, but you refer to their functions via the Function offsets.

All Com objects implements 3 "interfaces" or functions and thus each have a calltable of at least 3 functions.

```
[QueryInterface 0  
AddRed 4  
Release 8]
```

Any decendant object, will have these 3 interfaces, at the same offsets. They are "inherited" from the base

## Re: ASM noob – couple of questions

object, Iunknown.

So, the next functions in a COM object, will have an index of 12 (OCh)

take the example of the DirectShow IEnumPins class:

```
IEnumPins.ComClass:  
[IEnumPins.Next 0c  
IEnumPins.Skip 010  
IEnumPins.Reset 014  
IEnumPins.Clone 018]
```

This object has the above IUnknown implicitly defined, and thus it really looks like this:

```
IEnumPins.ComClass:  
  
[  
IEnumPins.QueryInterface 0  
IEnumPins.Addref 04  
IEnumPins.Release 08  
IEnumPins.Next 0c  
IEnumPins.Skip 010  
IEnumPins.Reset 014  
IEnumPins.Clone 018]
```

When we have obtained a pointer to a com object, we can use this list, along with the following macro to call to one of these functions

```
[iCall | mov edx #2 | mov edx D$edx | push #L>3 | push #2|call D$edx+#1]
```

for instance :

```
iCall IEnumPins.Next D$COMObjectPointer <param1> <Param2> ...
```

which unfolds to

```
mov edx D$ComObjectPointer  
mov edx D$edx  
push <param1>  
push <param2>  
call D$edx + IEnumPins.Next
```

To do a proper call one needs the documentation.

COM as explained in C, or Delphi documentation is such an abstract evil Beast, that it is said to take like 10 years to get your head around it. And that is true enough for me to never have bothered. I am interested in it, because its needed to do DirectX/DirectMedia programming. For this, the above is really all your need, apart from the documentation, todo DirectX with RosAsm or with any assembler running windows.

You will need the documentation to determine the names of the functions to call and their parameters, and

## Re: ASM noob – couple of questions

their relation to iunknown.

As you see, there is not really any Object "Tree" involved here. Each object is just one long flat list of pointers (callable) to "abstracted" functions supposed to do all the work needed to initialize and perform the userpart of the communication with the object.

The benefits are said to be language independence. The COM objects are maybe written in C or Pascal, but can be used from either, or from assembly, or even from Visual Basic.

There are "portable graphics libraries", but John may need DirectX or a near equivalent. This is for the webcam, right? Did "fasmcam.asm" do you any good? That uses AVICAP32.DLL. I don't know what the Linux equivalent would be. I saw an interesting comment that suggested that simply piping the "camera device" into the "framebuffer device" was "interesting"...

We all agree on the point that, in Win32 Assembly, around 80% of the time is wasted at searching the OS infos.

Not limited to Win32, either. I don't think the situation in Linux is quite as bad as you seem to think, but... The "assembly language" part is a lot easier than the "interfacing with the OS" part! Doing anything with graphics in Linux is particularly bad. This is why the "just call the library" (or "you \*must\* call the library"!) is so popular. :(

Just for the hell of it, here's some Windows/Linux code Numit\_or posted (Yahoo nasm-users group). Uses "opengl" (and/or "libglut" and/or "libglu"... "Mesa" may have something to do with it...) Requires some external files, it won't assemble or anything, but "just to look at":

```
;-----  
; triangle.asm  
;  
; A very simple *Linux/Windows* opengl application using the glut library. It  
; draws a nicely colored triangle in a top-level application window.  
; Compile with:  
; make linux (in Linux)  
; make windows (in Windows)  
;  
; you can find nasm in:  
; http://nasm.sourceforge.net/  
;  
; Mingw is in:  
; http://www.mingw.org/  
;  
; author: numit_or (numit_or at cantv.net)  
;  
;-----  
;  
; libGL is in  
; libMesaGL1 pack  
; libgkut and libGLU are in:
```

Re: ASM noob – couple of questions

```
; libMesaglut pack
;
;-----

#include "nmacros.inc"

GL_COLOR_BUFFER_BIT equ 16384
GL_POLYGON equ 9

section .data
title db 'A Simple Triangle', 0
zero dd 0.0
one dd 1.0
half dd 0.5
neghalf dd -0.5

section .text
display:
invoke glClear, GL_COLOR_BUFFER_BIT ; glClear(GL_COLOR_BUFFER_BIT)
invoke glBegin, GL_POLYGON ; glBegin(GL_POLYGON)
invoke glColor3f, [one], 0, 0 ; glColor3f(1, 0, 0)
invoke glVertex3f, [neghalf], [neghalf], 0 ; glVertex(-.5, -.5, 0)
invoke glColor3f, 0, [one], 0 ; glColor3f(0, 1, 0)
invoke glVertex3f, [half], [neghalf], 0 ; glVertex(.5, -.5, 0)
invoke glColor3f, 0, 0, [one] ; glColor3f(0, 0, 1)
invoke glVertex3f, 0, [half], 0 ; glVertex(0, .5, 0)
invoke glEnd ; glEnd()
invoke glFlush ; glFlush()
ret

PROC main, argc, argv
invoke glutInit, addr [.argc], [.argv]
invoke glutInitDisplayMode, 0
invoke glutInitWindowPosition, 80, 80
invoke glutInitWindowSize, 400, 300
invoke glutCreateWindow, title
invoke glutDisplayFunc, display
invoke glutMainLoop
ret
ENDP
;-----
```

Thanks for this one. Even it uses C, it doesnt look to complicated.

A DirectX example would be much more complex to read. In the sense that all names would be 100% unreadable. The above I actually understands at first read.

Now, to me, that doesn't look a hell of a lot like "real assembly language". I can stare at it and think like a CPU, but I can't see what it "does". Reminds me a bit of why I found trying to

Re: ASM noob – couple of questions

learn C "unsatisfying" – I felt more like a data entry clerk than a programmer. The library routines are having all the fun! :)

Can't blame Linux, or libraries – I get the same reaction to the Windows APIs. James was asking for some "pure asm" – defined as "using just interrupts" for Windows. Might have a better shot at that in Linux, actually. I'm beginning to figure out the "framebuffer device" – graphics for Linux with "just ints". This isn't the popular "GUI app" that John's wanting to learn, though... :(

Maybe someone, could give us ASMERS access to the whole of the machine through some DLL? If someone wrote a DLL, which would run like a service of something, we could all get access to the whole of the machine through it? Then we could program the machine directly whenever needed? Have anyone written such a library? Is it possible to bring a client program into ring 0 with such a library? For instance, would such a service give access to program windows using the real underlying interrupts? Could one for instance take windows to Fullscreen mode, and disable most of the system, and create sort of like an embedded OS on it?

Dos was more fun!

No Frank. You silly nostalgic. :)) Dos was more trouble. When you fixed the trouble, it took longer time, thus it was more of a relief, more of a felling of having "conquered" to solve it. Thus it may felt more fun. But you had acomplished..... less :))

But, say if the above suggestion could be performed... Could windows become fun again?

Best,  
Frank

—  
Sendt med Operas revolusjonerende e-postprogram: <http://www.opera.com/mail/>