

Re: cFASM (calling FASM as a C function)

Re: cFASM (calling FASM as a C function)

Source: <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2007-03/msg00821.html>

- *From:* //\\\\o\\\\annabee <Wannabee.Wannabee.org>
 - *Date:* Wed, 21 Mar 2007 21:06:50 +0100
-

På Wed, 21 Mar 2007 19:54:12 +0100, skrev randyhyde@xxxxxxxxxxxxxx <randyhyde@xxxxxxxxxxxxxx>:

On Mar 20, 6:48 pm, "vid...@xxxxxxxxxx" <vid...@xxxxxxxxxx> wrote:

- > If you go through the FASM source code, you'll find that it's
- > written using a 386-era programming style. By that, I mean that Tomasz
- > uses CISC instructions that get as much work done per byte of object
- > code as possible. While that was an effective programming style at one
- > time, on the modern "more-RISC-like" x86 CPUs you pay a heavy penalty
- > for using that programming style. A classic example is how Tomasz uses
- > the SCASD instruction to do nothing other than increment EDI by 4.
- > Clever, if saving space is your prime concern, but certainly not the
- > fastest way to add 4 to EDI. Given how heavily string instructions
- > get used in the code, I wouldn't be surprised to find that you could
- > double the speed of FASM by replacing those instructions.

In some discussions i recall, Tomasz was against lowlevel optimizations that makes maintaining harder.

This isn't a debate I want to get into here, but if maintenance is a concern, substituting discrete instructions for string instructions is nothing compared with other issues I've found. For example, the complete overlapping of function boundaries in the code is a real maintenance nightmare. The fact that one procedure calls another, that called procedure jumps back into the calling code, sometimes without cleaning up the stack (intended or otherwise) make maintenance difficult. Another issue: the almost exclusive use of "magic numbers" in the source code (e.g., offsets into structs and the like) is a **real** maintenance nightmare.

But ignoring some existing issues, I just don't see how using "add edi, 4" is going to make the code less readable and less maintainable than employing a trick like "scasd" to do the same task.

Also, as you probably don't know, size is a concern here. Tomasz wants to keep .COM version of FASM, but size of FASM is dangerously hitting

Re: cFASM (calling FASM as a C function)

64KB. :)

Why keep a .com version? .EXE work fine in 16-bit code. Even if one insists on keeping a 16-bit version active, there is no reasonable argument for keeping a .com version in this day and age. Granted, a "tiny model" program meshes well with FLAT code and nothing special needs to be done to the source file, but that's the price to pay for being able to maintain a 16-bit version.

Cheers,
Randy Hyde

Example of HEX only assembler

```
[Randy.Hide 0  
Randy.Age 4  
Randy.Pandy 8  
Randy.IQ 12
```

```
SizeOf_Randy 16]
```

Sourcecode:

```
B8 SizeOf_Randy  
E8 GetMem  
C7 40 Randy.Age 50  
E8 FreeMem
```

binary:

```
B8 10 00 00 00  
E8 00 45 40 00  
C7 40 04 50 00 00 00  
E8 00 90 40 00
```

:)))

----* Compared to assembly (not to confuse with HLA)

main:

```
mov eax SizeOf_Randy | call GetMem | mov D$eax + Randy.Age 80
```

----* compared to Delphi

Program LIES;

Type

Randy = Record

hide, age, pandy, IQ : DWORD;

end;

begin

New(Randy);

Re: cFASM (calling FASM as a C function)

Re: cFASM (calling FASM as a C function)

```
Randy.Age := 80;  
Dispose(Randy);  
end;
```

May someone write the `_complete_` HLA version? :D :D

It should be obvious, that the DELPHI code is `_MUCH_` less readable than the HEX version and with some practise much easier to write read and maintain.

The Hex version has all the benefits this way. The record can be changed, without touching the code. It is smaller. It will compile faster, it will use less paper when printed. It will give faster loading times, running times and use less memory. It will allow you to use less webspace for publishing. It will teach you all the opcodes so it will be much easier to read other applications. I know I could learn to write HEX this way, and to make as large a program with that as I do with RosAsm. Probably I will learn more, as the asm sometimes actually hide certain information.

What do you say. Who will make me this hex-assembler ? Maybe I try myself.

I could figure it be easy to allow for each time a label is changed, at the routine point, to automatically optionally rename all calls to it.

Given that RosAsm has proved that much of the trouble with ASM is the poor tools, (MASM, TASM, HLA) I say the same thing about this hex thing. Give it a proper tool, and we can make programming in hex as easy as programming in Visual Basic.

don't you agree?

Seems fully logic to me. I think I found myself a project.

.