

Re: I'd like to learn asm...

## Re: I'd like to learn asm...

---

*Source:* <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2006-05/msg00070.html>

---

- *From:* "randyhyde@xxxxxxxxxxxxxx" <randyhyde@xxxxxxxxxxxxxx>
  - *Date:* 1 May 2006 14:36:24 -0700
- 

Herbert Kleebauer wrote:

"randyhyde@xxxxxxxxxxxxxx" wrote:

Herbert Kleebauer wrote:

Intel software developer's manual vol 2a: 744 pages  
Intel software developer's manual vol 2b: 548 pages

That's a bit more than "a few hundred pages" mind you.

Remove the documentation for FP, MMX, SSE and then recount.

While we're at it, remove all the system (protected-mode) instructions, too.

But that's exactly what trade books do. And on top of it, those trade books explain how to actually *use* the machine instructions to accomplish useful work. Something the manufacturers documentation does not.

Yes, I know that you recommend using the obsolete 386 version as a guide. But why should someone give up the 486+ instructions?

Where did I say that you "should give up the 486+ instructions"? What I say is, that the 386 instruction set is a good start for learning x86 assembly programming. And once you are familiar with the 386, you can look at the extensions in the newer processors.

Re: I'd like to learn asm...

Re: I'd like to learn asm...

Okay. So read a couple hundred pages and then 1,500 pages more?

No offense, but the reader is better off reading a trade book to get the scoop on the important instructions, and the perusing the Intel documentation to complete their knowledge once they are comfortable with the basic instruction set. From experience (with thousands of students), I can assure you that "tossing out a list of the machine instructions" is a miserable way to learn assembly language programming. I realize you don't think that people who can't digest the information in this way are unworthy of learning assembly language, but some of us are interested in teaching a large number of people assembly language programming; not just the 1% who can figure it out on their own from the manufacturer's information.

The bottom line is that the manufacturer's documentation, while invaluable, usually contains \*far\* too much information of a highly detailed nature.

What is "\*far\* too much information"? Either it is information about the behavior of an instruction (then you need this information for assembly programming) or it is just some text which has nothing to do with the instruction (then I wouldn't call "information" in the context of an instruction description).

Well, let's consider the most often-used machine instruction: MOV.

Here's some of the information included in the Intel documentation for MOV:

Opcode Instruction

64-Bit

Mode

Compat/

Leg Mode Description

88 /r MOV r/m8,r8 Valid Valid Move r8 to r/m8.

REX + 88 /r MOV r/m8\*\*\*,r8\*\*\* Valid N.E. Move r8 to r/m8.

89 /r MOV r/m16,r16 Valid Valid Move r16 to r/m16.

89 /r MOV r/m32,r32 Valid Valid Move r32 to r/m32.

REX.W + 89 /r MOV r/m64,r64 Valid N.E. Move r64 to r/m64.

(and on and on for all variants of the MOV instruction)

The information above is \*great\* if you're writing an assembler or disassembler. It's not very interesting to the person wanting to learn assembly language programming. Continuing on:

Re: I'd like to learn asm...

```
IF SS is loaded
THEN
IF segment selector is NULL
THEN #GP(0); FI;
IF segment selector index is outside descriptor table limits
or segment selector's RPL ` CPL
or segment is not a writable data segment
or DPL ` CPL
THEN #GP(selector); FI;
IF segment not marked present
THEN #SS(selector);
ELSE
SS segment selector;
SS segment descriptor; FI;
FI;
```

This kind of information is great for a OS architect, computer architect, or similar person, it's of little interest to the person wanting to learn assembly language programming. Continuing on:

#### Protected Mode Exceptions

#GP(0) If attempt is made to load SS register with NULL segment selector.

If the destination operand is in a non-writable segment.

If a memory operand effective address is outside the CS, DS, ES, FS, or GS segment limit.

If the DS, ES, FS, or GS register contains a NULL segment selector.

#GP(selector) If segment selector index is outside descriptor table limits.

If the SS register is being loaded and the segment selector's RPL and the

segment descriptor's DPL are not equal to the CPL.

If the SS register is being loaded and the segment pointed to is a non-writable data segment.

If the DS, ES, FS, or GS register is being loaded and the segment pointed

to is not a data or readable code segment.

If the DS, ES, FS, or GS register is being loaded and the segment pointed

to is a data or nonconforming code segment, but both the RPL and the CPL

are greater than the DPL.

#SS(0) If a memory operand effective address is outside the SS segment limit.

#SS(selector) If the SS register is being loaded and the segment pointed to is marked not present.

#NP If the DS, ES, FS, or GS register is being loaded and the segment pointed

to is marked not present.

Re: I'd like to learn asm...

(and continuing on and on)

This kind of detail is interesting to have if you're writing code to handle exceptional conditions, but it's a bit too much detail for someone who just wants to know how to use the MOV instruction to write an 80x86 program. After all, how many people get all concerned about the types of exceptions that can occur everytime they write a MOV instruction? Most people don't care at all until an exception occurs. THEN they look it up in the manual. There is no need to be concerned with this kind of information when first learning assembly language programming.

Indeed, for the MOV instruction, all the beginner really wants to know is that it copies data from source to destination, what's legal for the source and destination operands, and how they would use the MOV instruction to solve real-world problems. The Intel documentation doesn't really explain what you would use MOV for, and it certainly doesn't explain how you would use it to solve real-world problems. That's why trade books are so popular. They explain this kind of stuff.

This is why trade books on assembly language programming sell so well -- they skip discussing unimportant instructions and concentrate on explaining how to use common instructions to actually \*accomplish\* something.

It doesn't make any sense to teach 10% of the integer instruction, 10% of FP instructions, 10% of the MMX instructions ....

What trade book does this?

Better teach 100% of the integer instructions and ignore the rest for the moment (i.e. start with the 386 instruction set).

Really? What good are instructions like LAR and LSL going to do the beginner who is first learning assembly language? For that matter, as almost all programming today is done in flat mode, why should they learn about segments and any of the instructions that manipulate segment registers?

This is like reading a manual. It doesn't make any sense to read only the first chapters until you know enough to use

Re: I'd like to learn asm...

Re: I'd like to learn asm...

the tool for simple tasks and only read further chapters if you are unable to solve a problem with your current knowledge.

That may not make sense to you, but that's the way the world operates.

You always should read the complete manual (from the first to the last page) before you even start the tool.

Yes, you probably should. But again, that's not the way the world operates.

As you've noted yourself, you don't need to learn the entire instruction set in order to write meaningful programs (e.g., you can skip the FPU, MMX, and SSE instructions according to you). Likewise, you don't have to learn all of the integer instructions. You can start out with a reasonable subset and grow your knowledge as time progresses. Don't forget, people were writing perfectly reasonable assembly applications with the 8088, long before the 386 and later x86 came along. Even *then* they weren't using the entire instruction set.

And it doesn't matter if you only understand 20% of what you are reading. It is essential that you have an overview of all the features even if you don't understand any details at that time.

Really? It's essential for someone to read about the LAR or INVLPG instructions before they use the MOV instruction?

But it is unnecessary to read the manuals about all the existing plug-ins for the tool as long as you don't use them.

This completely contradicts what you've just said.

So, anyone who wants to learn x86 assembly programming should read the complete processor manuals (ignoring the FP, MMX, SSE extensions for the moment) before writing the first line of assembler code.

Re: I'd like to learn asm...

Re: I'd like to learn asm...

Wait a second. First you say they need to review all the features, now you're saying that they can ignore better than half the instruction set. Which is it? And if they can ignore the FP, MMX, and SSE instructions, why now allow them to ignore the system (protected mode only) instructions? And why not ignore the "obsolete" legacy instructions that Intel doesn't want you to use any more? And what about....?

Guess what, this is *exactly* what the trade books do. And that's why they're so popular (versus learning from the Intel manuals).

## 2. A book about the OS interface.

most of the information in Petzold's book (or MSDN on-line, if you prefer) is of little interest to someone learning assembly language.

Yes, it "is of little interest", but still necessary. Assembly programming means YOU, the PROCESSOR and the OS INTERFACE and nothing else.

That may be what it means to you, but I can assure you that this is not what it means to everyone else.

There  
must be no hidden information.

Nonsense. The OS interface itself hides information. Indeed, the *processor* hides stuff (Do you, for example, know exactly how memory is cached during your program's execution? Do you know how instructions are being scheduled?)

This means, you need to understand  
each byte the exe file,

Why? While this information is interesting to know, it's hardly necessary to write an assembly language program. I can write the same assembly language code to run under Windows or Linux without ever

Re: I'd like to learn asm...

Re: I'd like to learn asm...

having to worry about what form the executable file takes.

Knowing the executable (or object code) file format \*may\* be useful in a few special cases, but it's hardly required knowledge for writing assembly language programs.

how the exe file is loaded into memory

What does this have to do with assembly language? Either you'd have the same concerns when writing HLL code or you don't really care at all. Again, there are a few specialized cases where knowing this information might prove useful, but for the bulk of applications one might write with assembly, how the OS loads the program into memory is irrelevant.

and  
how the API calls works.

Or you could just use a standard library and abstract this stuff away (i.e., \*hide\* it). It doesn't change the fact that you're using assembly in any way.

And all this is much easier with the DOS  
interface (or BIOS interface and direct hardware access).

It's even easier to use a library such as the C standard library of the HLA standard library to completely abstract away the OS API. The OS API has \*nothing\* to do with learning or using assembly language. You can write assembly code for an embedded system that has no OS, for example.

After all, learning the comparable Windows or Linux API calls that do the same thing as the DOS API is no more work than learning DOS. And in the end, you've learned something that you can continue to use rather than wasting your time learning an obsolete OS' API.

It is much better to LEARN and UNDERSTAND something which is obsolete than just to USE something (without really understanding it) which is not obsolete.

Re: I'd like to learn asm...

Re: I'd like to learn asm...

Better in what way? Learning for the sake of learning, perhaps. But most people don't learn assembly language because they just want to learn something new. Most people who desire to learn assembly language (as opposed to those being forced to learn it in a required course) have a specific application in mind. And today, few of those applications involve writing DOS code. Indeed, IIRC, the OP wanted to learn Linux, but would settle for Windows. Nowhere did he mention DOS and I really doubt he would be interested in wasting his time with DOS (indeed, IIRC, he voiced his displeasure with wasting time with Windows). And while on the subject, I might point out that the Linux API isn't significantly more complex than DOS'. So if you want to go with a simplistic API approach, I'd think you'd argue that people should be learning Linux. At least that's not obsolete.

We don't need programmer who are able to somehow generate working code, we need people who exactly understand how the code is working.

You've failed to convince me that someone who learns DOS first is going to exactly understand how their Windows or Linux applications are working. And who really cares if they completely understand how some DOS demo works? Nobody uses DOS any more and that knowledge doesn't carry over to OSes that people actually care about today.

Of course, you can also bypass the OS's native interface completely and make calls to a standardized library. This could be the C standard library (e.g., portable across OSes) or something like the HLA standard library (portable across Windows and Linux).

That has nothing to do with assembly programming.

Neither does the OS API. But you keep bringing that up. The OS API is just an abstraction of the underlying hardware. So why not abstract the OS API while we're at it?

This is generating a sequence of library calls using the CPU "call" instruction.

Re: I'd like to learn asm...

Re: I'd like to learn asm...

So? What do you think an OS API \*CALL\* is?

3. A book about the assembler you use (a few dozen of pages)

If the assembler's documentation is only a few dozen pages, then it is either grossly underdocumented or has so few features that people should run, not walk, away from the product.

An assembler is a tool which converts a symbolic representation of CPU instructions, memory addresses and constants into a binary form and nothing more. Anything else is a programming language which allows you to include CPU instructions but is inappropriate for understanding the processor architecture (learning assembly programming). And for an assembler a few dozen of pages documentation should be more than enough.

Yes, I realize you've written a very "limited-feature" assembler of your own. But the fact that you find such a limited tool appropriate for all the assembly language programming you do does not imply that everyone else finds such a tool acceptable. And considering the pains you claim to go to in order to avoid working in assembly language, I'm also sure that the tool you find appropriate for your use would be unsuitable for someone else. Further, as you \*wrote\* your own assembler, I'm quite sure that you'd think a couple of dozen pages of documentation is all \*you\* need for \*that\* assembler.

Of course, the fact that you've never bothered to learn how to read anything remotely resembling 80x86 syntax suggests that you take this concept (of only a couple dozen pages) serious. Alas, most people wanting to learn x86 assembly language don't approach it by writing their own assembler. If they did, then I'm sure they'd find a couple dozen pages of documentation appropriate for their assemblers, too.

In opposite to the DOS interface, writing applications in assembler is really obsolete (in the desktop, not in the embedded market).

You're probably correct.

But that doesn't stop people from wanting to learn how to do it for one reason or another. So why compound one mistake with another (learning

Re: I'd like to learn asm...

Re: I'd like to learn asm...

DOS)?

But  
learning assembly programming (understanding the processor architecture)  
will never be obsolete.

Of course, this is why most Universities and Colleges still require students to take an assembly language programming course.

So as long as you're learning, why not learn it with a tool that makes your knowledge practical, in case you really \*need\* to use assembly once in a while? This is a major student complaint: why should I be learning this "obsolete" information involving DOS? OTOH, teach them the same exact material under Windows or Linux and they're much more motivated.

So people should run, not walk, away from a product which tries to promote HL constructs for an assembler.

Well, except for the fact that this has proven to be an especially efficient and practical way to learn assembly language; especially during the first couple of weeks of the course when the students haven't mastered enough of the basic machine instructions to do anything useful.

Either  
use an optimizing HL languages or pure CPU instructions and then combine the results (using inline assembly or a linker) but never use a mixture in one language ("High Level Assembler").

But you said that the idea was to learn the machine architecture, not write programs. So who \*cares\* whether the code is optimized or not (and I can guarantee you that student code is suboptimal). All that's important is that the students learn the machine instructions and the basics of machine architecture. High-level assemblers have proven especially valuable in this context. By the end of the course, the students have learned \*more\* machine instructions and are writing more complex \*pure\* assembly language programs than when taught with traditional (non-high-level assembler) methods.

As much as you would like everyone to have to suffer the same way you did when learning assembly language (learning it by simply reading the manufacturer's literature), most people want to make more efficient use

Re: I'd like to learn asm...

Re: I'd like to learn asm...

of their time and reject your approach.

"Salvation through suffering" hardly applies here.

Cheers,

Randy Hyde

.