

Re: which book to start with...?

Source: <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2007-12/msg00174.html>

- *From:* Frank Kotler <fbkotler@xxxxxxxxxxx>
 - *Date:* Fri, 07 Dec 2007 07:34:53 GMT
-

naunetr wrote:

....

okay thanks. luckily linux API is simple to use and resembles the c standard lib routines somewhat. the man page descriptions are written for c programming. but its easy to translate to assembler so far. but what about syscalls that take more than 5 parametres? are the others pushed onto stack like in c (but not in reverse order)?

If there are more than 5 parameters (6 in newer sys_calls), they're arranged in a structure, and the address of the structure is passed in ebx. (usually... AFAIK). This often *is* on the stack, but doesn't have to be. I guess "fastcall" can put some parameters in registers, and "extras" on the stack, and I think the 64-bit calling conventions do it, but I think 32-bit Linux is always either/or.

....

can we define the standard sections (.text, .data and .bss) several times? will they be merged when object file is created?

Yeah.

```
section .text
_start:
mov eax, 4
section .data
msg db "hello "
section .text
mov ebx, 1
section .data
db "world"
section .text
mov ecx, msg
section .data
db "!"
section .text
mov edx, 13
section .data
```

Re: which book to start with...?

```
db 10
section .text
int 80h
mov eax, 1
int 80h
```

Something like that ought to work (untested). We can also get ld to "merge" or "combine" or "overlap" sections, I think, but I don't know how that works.

also is there any chance that .data can be put in read-only memory?

Yeah, as long as we don't need to alter it. We can put "constant data" in .text, or Nasm knows .rodata. The default attributes of an arbitrarily-named section are right for "constant data" too, I think.

because i use it to define buffers also... or should i be safe and put them in .bss as nasm advises?

Buffers – not initialized – can go in .bss, where they won't add anything to the length of the file. Or we could put 'em in .data... but they *will* add to the length of the file. We need the memory at runtime either way, of course.

```
section .bss
buffer resb 1000h
```

or:

```
section .data
buffer resb 1000h
```

Nasm will complain about that, but do the right thing. The "right" way to do it – in .data – would be:

```
buffer times 1000h db 0
```

You wouldn't want a buffer in .text, since we'll be writing to it. Section .bss is nominally "uninitialized" data, but is in fact cleared to zero. An exception to this is a dos .com file, where you can write something to .bss, exit the program, restart it and still find our data there. If something happens between runs to cause us to be loaded at a different address, this won't work, of course...

....

so if we have compared two signed values with cmp we should probably use jumps like je, jg, jl, js etc. and for unsigned compares jz, ja, jb etc. am i on the right track.

Right.

Re: which book to start with...?

Re: which book to start with...?

but if we are sure that the result of a `cmp` between two signed values will give only a positive result then we can use `ja`, `jb` and similar?

I guess it depends on what you mean by "positive result". We can mix 'em a little, but be aware that a "large" unsigned number can be a "small" (less than zero) signed number.

```
mov al, 0FFh ; could mean 255 or -1
mov bl, 1
cmp al, bl
ja someplace ; taken 255 greater than 1 (unsigned)
jg someplace ; not taken -1 not greater than 1 (signed)
```

so both these jumps are "correct" depending on what type of comparison you want. is that right?

Right. We'd probably write "`mov al, -1`" for clarity, if we intended it to be treated as signed, but it's the same bit pattern.

....

even for binary files i think `read` will return 0 when the end of file is reached.

Yes. `Read` returns the number of bytes read (in `eax`). But our "`getc`" wants to return the character we read, not the number of bytes. If `read` returns zero – EOF – we haven't *got* a byte to return. We could just exit right there – *this* program is finished when that happens. But we might want to do more, so we return some value that couldn't be a valid character. If we returned zero from `getc` when `read` returned zero, we couldn't distinguish between a valid character (zero is unlikely in a text file, but...) and EOF.

....

basically i thought preopening `stdin`, `stdout` and `stderr` are only c behaviour and since we are not using the c library at all, we should do an `open` for them. in windows we have to do `GetStdHandle` before we can write to console. but `stdin`, `stdout` and `stderr` have no pathnames associated for them so `open` is not possible.

Might be able to open `"/dev/tty?"` or something...

this is the docs installed by `nasm v2` installation from source. it installed in `/usr/doc/nasm` and the html contents file (`nasmdoc0.html`) doesnt have a link to the instruction reference but i have to manually load the `nasmdocb.html` file in browser.

There shouldn't be any `nasmdocb.html` in 2.00 at all. Shouldn't be any reference to it in the contents, or the

Re: which book to start with...?

Re: which book to start with...?

index, or anywhere. I wouldn't want to guarantee that it's completely "clean", though.

<http://home.comcast.net/~fbui/intel.html>

So...I think it's "too bad" that the section was removed from the Nasm manual, but it isn't a total loss...

okay i'll use that then. i used nasm's one because intel's manuals are too big and detailed for me yet :) also i hate pdf files...

Painful Document Format? Agreed! (but I really should read that Intel manual some day...)

Best,
Frank

.