Real mode assembly I

From OSDev Wiki

In this tutorial we will assemble a small 16-bit assembly language kernel with NASM and boot it.

Difficulty level

Beginner

Contents

Overview

EDIT #2: I've put a little "series box" at the bottom of each page to allow quick access to the previous and next tutorial in the series.

You're probably going to sigh and dismiss yet another tutorial on writing operating systems in x86 assembly language, especially since this one uses real mode. But there's a catch to this one; it actually does more than printing "Hello World" to the screen and halting.

For this, you'll need:

- the latest version of NASM (2.05.01 as of November 28th, 2008)
- PARTCOPY on Windows or dd on Linux
- an emulator like QEMU, Bochs, or Microsoft Virtual PC

So what's it going to look like?

Well, there will be a single source file, the kernel. What about a bootloader? This is such a small kernel, we're not going to use a filesystem at all, we're just going to put the kernel into the first few sectors of the floppy!

The system will have a string printing call (of course), keyboard input, and a strcmp call similar to that of C, all packaged into less than a sector.

But I want a GUI and sound effects and all the Windows games to work on my OS...

Beginner Mistakes

So where's the code?

Here you go, go wild.

```
org 0x7C00   ; add 0x7C00 to label addresses
bits 16      ; tell the assembler we want 16 bit code

  mov ax, 0  ; set up segments
  mov ds, ax
  mov es, ax
  mov ss, ax     ; setup stack
  mov sp, 0x7C00 ; stack grows downwards from 0x7C00

  mov si, welcome
  call print_string

mainloop:
  mov si, prompt
  call print_string

  mov di, buffer
  call get_string

  mov si, buffer
  cmp byte [si], 0  ; blank line?
  je mainloop       ; yes, ignore it

  mov si, buffer
  mov di, cmd_hi  ; "hi" command
  call strcmp
  jc .helloworld

  mov si, buffer
  mov di, cmd_help  ; "help" command
  call strcmp
  jc .help

  mov si,badcommand
  call print_string
  jmp mainloop

.helloworld:
  mov si, msg_helloworld
  call print_string

  jmp mainloop

.help:
```

```
  mov si, msg_help
  call print_string

  jmp mainloop

welcome db 'Welcome to My OS!', 0x0D, 0x0A, 0
msg_helloworld db 'Hello OSDev World!', 0x0D, 0x0A, 0
badcommand db 'Bad command entered.', 0x0D, 0x0A, 0
prompt db '>', 0
cmd_hi db 'hi', 0
cmd_help db 'help', 0
msg_help db 'My OS: Commands: hi, help', 0x0D, 0x0A, 0
buffer times 64 db 0

; ================
; calls start here
; ================

print_string:
  lodsb        ; grab a byte from SI

  or al, al  ; logical or AL by itself
  jz .done   ; if the result is zero, get out

  mov ah, 0x0E
  int 0x10      ; otherwise, print out the character!

  jmp print_string

.done:
  ret

get_string:
  xor cl, cl

.loop:
  mov ah, 0
  int 0x16   ; wait for keypress

  cmp al, 0x08    ; backspace pressed?
  je .backspace   ; yes, handle it

  cmp al, 0x0D  ; enter pressed?
  je .done      ; yes, we're done

  cmp cl, 0x3F  ; 63 chars inputted?
  je .loop      ; yes, only let in backspace and enter

  mov ah, 0x0E
  int 0x10      ; print out character
```

```
  stosb  ; put character in buffer
  inc cl
  jmp .loop

.backspace:
  cmp cl, 0    ; beginning of string?
  je .loop     ; yes, ignore the key

  dec di
  mov byte [di], 0     ; delete character
  dec cl               ; decrement counter as well

  mov ah, 0x0E
  mov al, 0x08
  int 10h              ; backspace on the screen

  mov al, ' '
  int 10h              ; blank character out

  mov al, 0x08
  int 10h              ; backspace again

  jmp .loop    ; go to the main loop

.done:
  mov al, 0    ; null terminator
  stosb

  mov ah, 0x0E
  mov al, 0x0D
  int 0x10
  mov al, 0x0A
  int 0x10             ; newline

  ret

strcmp:
.loop:
  mov al, [si]   ; grab a byte from SI
  mov bl, [di]   ; grab a byte from DI
  cmp al, bl     ; are they equal?
  jne .notequal  ; nope, we're done.

  cmp al, 0  ; are both bytes (they were equal before) null?
  je .done   ; yes, we're done.

  inc di     ; increment DI
  inc si     ; increment SI
```

```
    jmp .loop   ; loop!

  .notequal:
    clc   ; not equal, clear the carry flag
    ret

  .done:
    stc   ; equal, set the carry flag
    ret

    times 510-($-$$) db 0
    dw 0AA55h ; some BIOSes require this signature
```

To assemble on Windows:

```
nasm kernel.asm -f bin -o kernel.bin
partcopy kernel.bin 0 200 -f0
```

Or on Linux:

```
nasm kernel.asm -f bin -o kernel.bin
dd if=kernel.bin of=/dev/fd0
```

Those commands assemble your kernel binary and write them to the first floppy disk. Go ahead and test out your operating system now!

What next?

Why, that's up to you of course! You could add more commands, expand your OS to another sector and learn to use the BIOS floppy functions, add a stack and improve the calls, etc.

Have fun with your OS, however you decide to write it!

EDIT on December 12 2008: I've written a second part to this tutorial at Real mode assembly II. This and future parts will have less code to copy and paste and more theory!


```
<- none  |  Real mode assembly II ->
```


Categories: Level 1 Tutorials | Real mode assembly
