Writing a Boot Loader

How to write ‘Hello World!’ to the screen in increasingly complicated ways

Software

- NASM Assembler (http://www.nasm.us/)
- QEMU Emulator (https://www.qemu.org/)
- = No destroyed PC hardware ;)

```
brew install nasm qemu
```

Turning on the Computer

- Press the Power Button
- BIOS Hardware Test
- Loads the first 512 bytes off the floppy disk/hard drive
- Does byte 510-512 equal 0x55AA? YES jump to 0x7C00!

Real Mode

- http://wiki.osdev.org/Real_Mode
- BIOS Interrupts https://en.wikipedia.org/wiki/BIOS_interrupt_call
- 16 bit instructions
- Only 1 MB of memory can be accessed (ignoring segments)

Register	Accumulator	Counter	Data	Base	Stack Pointer	Stack Base Pointer	Source	Destination
64-bit	RAX	RCX	RDX	RBX	RSP	RBP	RSI	RDI
32-bit	EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
16-bit	AX	CX	DX	BX	SP	BP	SI	DI
8-bit	AH AL	CH CL	DH DL	BH BL				

```
1  bits 16
2  org 0x7c00
3
4  boot:
5      mov si,hello
6      mov ah, 0x0e
7  .loop:
8      lodsb
9      or al,al
10     jz halt
11     int 0x10
12     jmp .loop
13 halt:
14     cli
15     hlt
16 hello: db "Hello world!",0
17
18 times 510 - ($-$$) db 0
19 dw 0xaa55
20
```

output 16 bit instructions

ah=0x0e int 0x10 means
'Write Character in TTY mode'

loads byte at address `ds:si` into `al`.

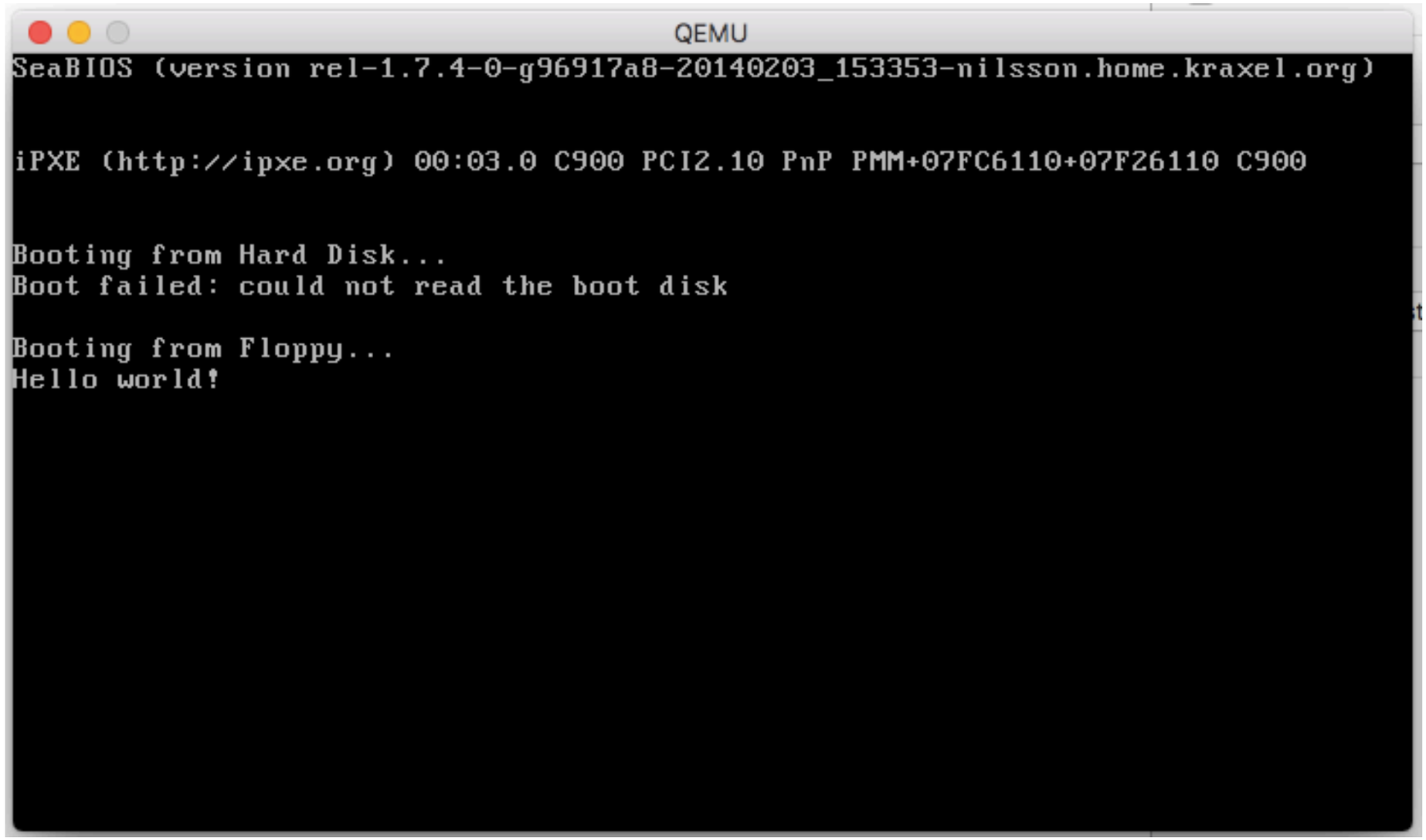
clear interrupts
halt cpu

pad remaining 510 bytes with 0

magic!

```
$ hexdump boot.bin
0000000 be 10 7c b4 0e ac 08 c0 74 04 cd 10 eb f7 fa f4
0000010 48 65 6c 6c 6f 20 77 6f 72 6c 64 21 00 00 00 00
0000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
*
00001f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa
0000200
```

```
nasm -f bin boot1.asm -o boot.bin
qemu-system-i386 -fda boot.bin
```



32 bit Mode

- Enable A20 Line (http://wiki.osdev.org/A20_Line)

```
mov ax, 0x2401
int 0x15
```

- Setup a Global Descriptor Table (http://wiki.osdev.org/GDT)
- Set Protected Mode Bit on **cr0**
- Jump to 32 bit Code!

Global Descriptor Table

- Tells the CPU what memory ranges mean
- Useful for memory protection
- Tells CPU about 64/32/16 bit modes
- Can you execute this? Can you read/write this?

- Load with special **lgdt** instruction!

```
gdt_start:
    dq 0x0
gdt_code:
    dw 0xFFFF
    dw 0x0
    db 0x0
    db 10011010b
    db 11001111b
    db 0x0
gdt_data:
    dw 0xFFFF
    dw 0x0
    db 0x0
    db 10010010b
    db 11001111b
    db 0x0
gdt_end:
gdt_pointer:
    dw gdt_end - gdt_start
    dd gdt_start
```

code segment from 0–0xFFFF with read/write/execute and 32 bits flags

data segment from 0–0xFFFF with read/write and 32 bits flags

pointer structure telling CPU how big the GDT is

0	8	12	16	20	24	28	32
limit_low	base_low		base_middle	access	flags	base_high	

access layout

0							8
present	ring level		1	executable	direction	read/write	accessed

flags layout

0				4	8
granularity	size	0	0	limit_high	

```
CODE_SEG equ gdt_code - gdt_start
DATA_SEG equ gdt_data - gdt_start
```

```
lgdt [gdt_pointer]      load gdt table
mov eax, cr0
or eax,0x1              set protected mode bit in cr0
mov cr0, eax
mov ax, DATA_SEG        set all the other segments to data
mov ds, ax
mov es, ax
mov fs, ax
mov gs, ax
mov ss, ax
jmp CODE_SEG:boot2      set code segment and jump!
```

Protected Mode VGA

```
mov ax, 0x3
int 0x10
```

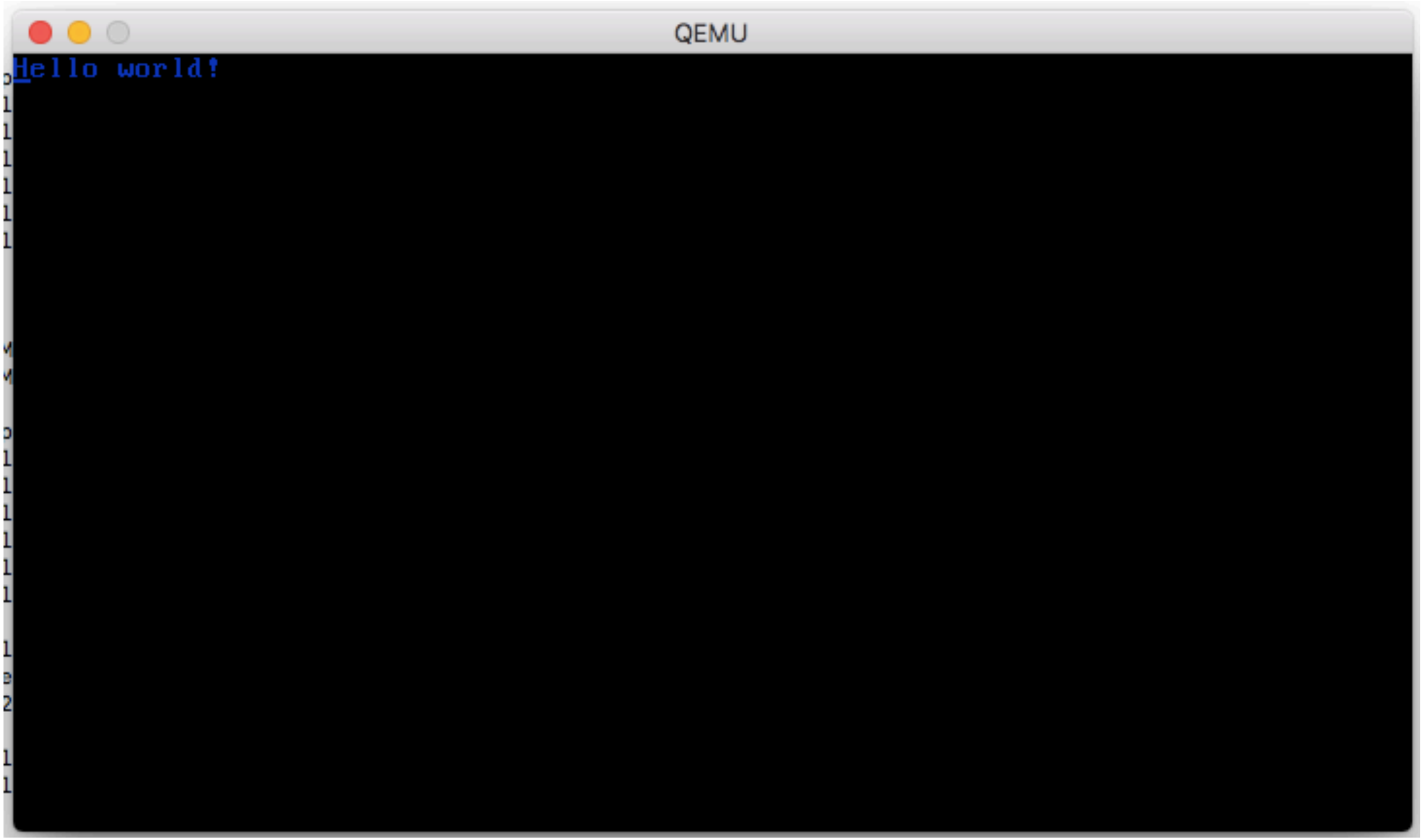
Text Mode 3
80x50 characters with
16 colours memory mapped to `0xb8000`

0		8	16
background_color	foreground_color	ascii character	

```
bits 32
boot2:
    mov esi,hello
    mov ebx,0xb8000
.loop:
    lodsb
    or al,al
    jz halt
    or eax,0x0100
    mov word [ebx], ax
    add ebx,2
    jmp .loop
halt:
    cli
    hlt
hello: db "Hello world!",0
```

loads byte at address `ds:esi` into `al`.

set foreground colour to blue (1)


QEMU
Hello world!

Beyond 512 Bytes

Disk Interrupts (https://en.wikipedia.org/wiki/INT_13H)

```
mov [disk],dl
```

bios magic dl register value

```
mov ah, 0x2      ;read sectors
mov al, 6        ;sectors to read
mov ch, 0        ;cylinder idx
mov dh, 0        ;head idx
mov cl, 2        ;sector idx
mov dl, [disk]   ;disk idx
mov bx, copy_target;target pointer
int 0x13
```

ah=0x2 int 0x13 means ‘Read Sectors From Drive’

```
times 510 - ($-$$) db 0   our boot sector zero padding
dw 0xaa55   bootsector magic value
copy_target:
bits 32
    hello: db "Hello more than 512 bytes world!!",0
boot2:
    mov esi,hello
    mov ebx,0xb8000
.loop:
    lodsb
    or al,al
    jz halt
    or eax,0x0F00   0x0F00 = white text (15)
    mov word [ebx], ax
    add ebx,2
    jmp .loop
```


QEMU
Hello more than 512 bytes world!!

Getting to C++!

avoid C++ name mangling

```
extern "C" void kmain()
{
    const short color = 0x0F00;
    const char* hello = "Hello cpp world!";
    short* vga = (short*)0xb8000;
    for (int i = 0; i<16;++i)
        vga[i+80] = color | hello[i];
}
```

+80 to write on second line (80x50 mode)

Cross Compiler

- A compiler specifically targeted to your platform
- A Complete Nightmare to compile!
- Removes weird OS hacks, optimisations and function call conventions

```
brew tap zanders3/homebrew-gcc_cross_compilers
brew install i386-elf-gcc
```

Call C++ from Assembly

```
bits 32
    mov esp,kernel_stack_top
    extern kmain
    call kmain
    cli
    hlt

section .bss
align 4
kernel_stack_bottom: equ $
    resb 16384 ; 16 KB
kernel_stack_top:
```

esp = stack pointer
it grows down!

reserve 16KB of stack

Link it all together!

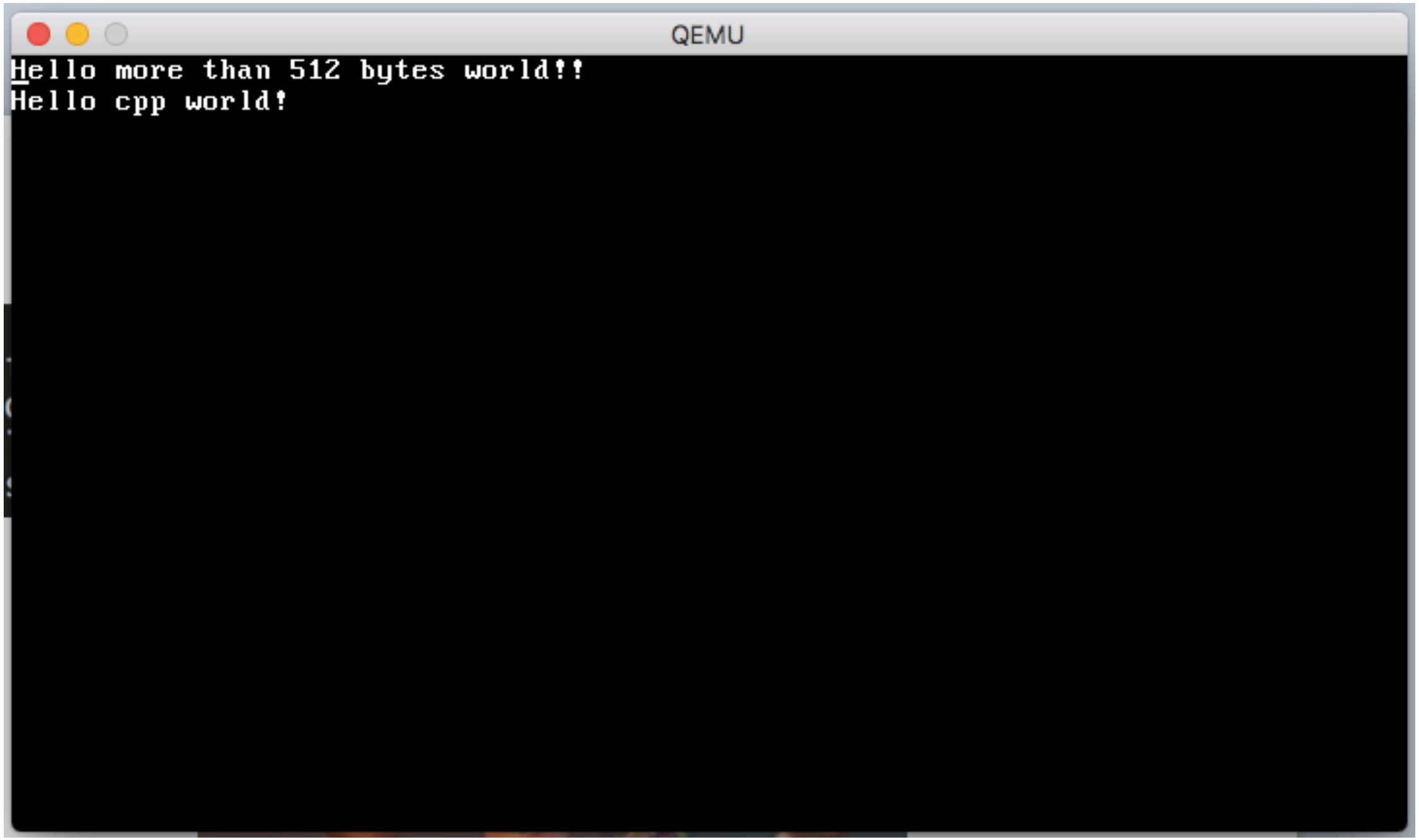
```
ENTRY(boot)
OUTPUT_FORMAT("binary")
SECTIONS {
    . = 0x7c00;
    .text :
    {
        *(.boot)
        *(.text)
    }

    .rodata :
    {
        *(.rodata)
    }

    .data :
    {
        *(.data)
    }

    .bss :
    {
        *(.bss)
    }
}
```

output asm directly in binary
not ELF or EXE, etc.

start at 0x7c00

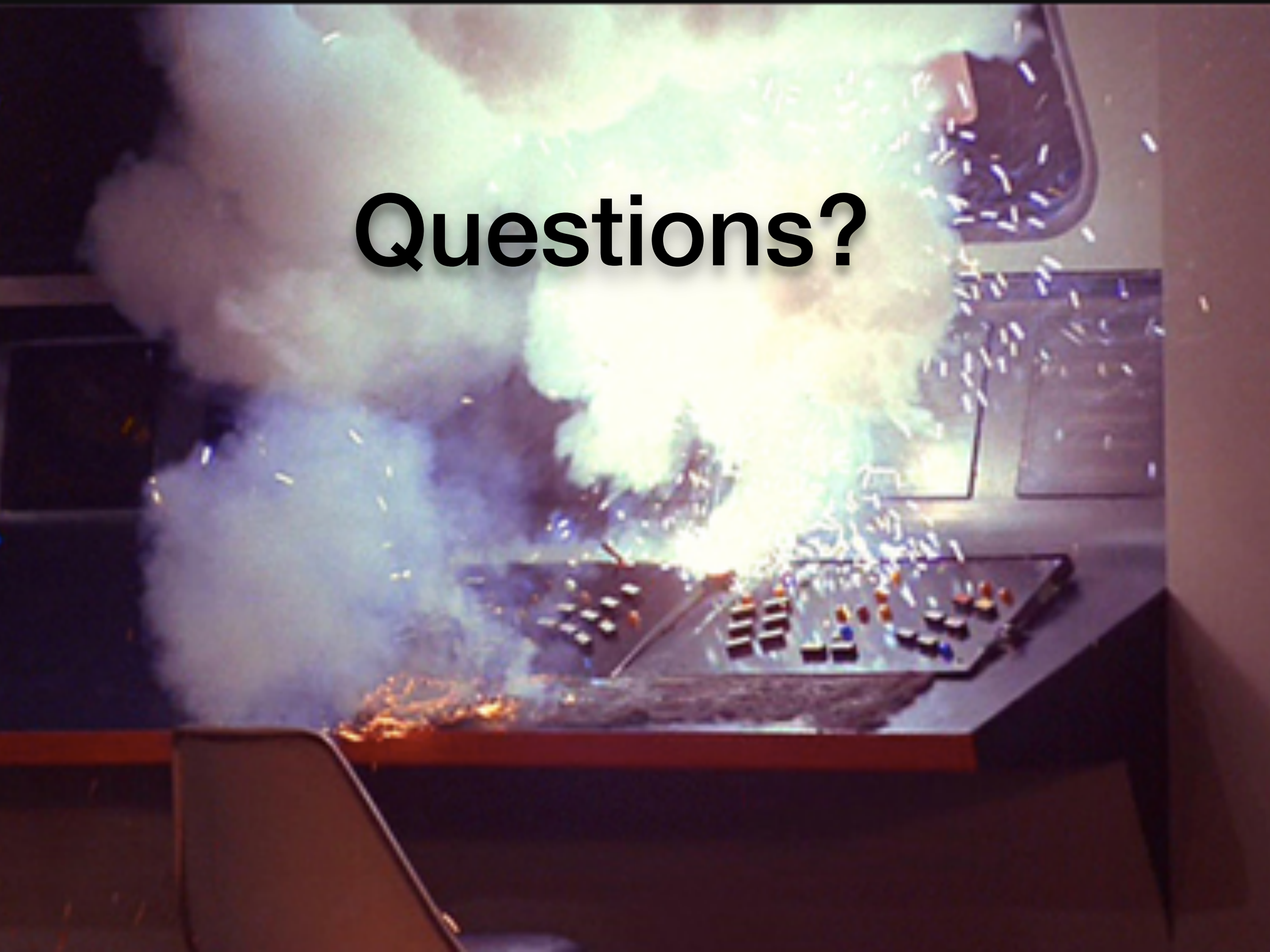
put the boot loader first
all the C++ stuff after

```
nasm -f elf32 boot4.asm -o boot4.o
i386-elf-_g++ kmain.cpp boot4.o -o kernel.bin -nostdlib -ffreestanding
-mno-red-zone -fno-exceptions -fno-rtti -Wall -Wextra -Werror -T
linker.ld
qemu-system-i386 -fda kernel.bin
```


QEMU
Hello more than 512 bytes world!!
Hello cpp world!

Resources

- http://wiki.osdev.org/Main_Page
- http://3zanders.co.uk/2017/10/13/writing-a-bootloader/
- https://os.phil-opp.com/multiboot-kernel/
- http://www.jamesmolloy.co.uk/tutorial_html/


Questions?