<https://pdos.csail.mit.edu/6.828/2017/labs/lab1/>

1. Familiarize yourself w/ assembly

2. Use GDB’s si command to trace into the ROM BIOS for a few more instructions, and try to guess what it might be doing.

* At what point does the processor start executing 32-bit code? What exactly causes the switch from 16- to 32-bit mode?

Switches from real to protected mode at 0x7c1e. It uses a bootstrap GDT and segment translation that makes virtual addresses identical to their physical addresses, so that the effective memory map does not change during the switch.

The instructions are:

lgdt gdtdesc

movl %cr0, %eax

orl $CRO\_PE\_ON, %eax

movl %eax, %cr0

# Jump to next instruction, but in 32-bit code segment.

# Switches processor into 32-bit mode.

ljmp $PROT\_MODE\_CSEG, $protcseg

* What is the *last* instruction of the boot loader executed, and what is the *first* instruction of the kernel it just loaded?

Last instruction bootloader is 0x7d6b.

First instruction of kernel is 0xf0100507 < kdb\_intr+16>

* *Where* is the first instruction of the kernel?

0x10018: 0x0010000c

* How does the boot loader decide how many sectors it must read in order to fetch the entire kernel from disk? Where does it find this information?

It looks at how many times it needs to iterate in the for loop of the main.c bootmain function. It must iterate eph times, where eph = ph + ELFHDR -> e\_phnum.

3. Trace into bootmain() in boot/main.c, and then into readsect().

Looking at obj/boot/boot.asm, I found that readsect()’s address is 0x7c7c.

4. Identify the exact assembly instructions that correspond to each of the statements in readsect().

The statements in readsect() are:

Void readsect(void \*dst, uint32\_t offset)){

// wait for disk to be ready

waitdisk();

outb(0x1F2, 1); //count = 1

outb(0x1F3, offset);

outb(0x1F4, offset >> 8);

outb(0x1F5, offset >> 16);

outb(0x1F6, offset >> 24 | 0x E0);

outb(0x1F7, 0x20); //cmd 0x20 – read sectors

// wait for disk to be ready

waitdisk();

//read s ector

insl(0x1F0, dst, SECTSIZE/4);

}

Assembly Instructions, set breakpoint to 0x7c7c in gdb, step through (si)

Void readsect(void \*dst, uint32\_t offset){

7c7c: 55 push %ebp

7c7d: 89 e5 mov %esp, %ebp

7c7f: 57 push % edi

7c80: 8b 4d 0c mov 0xc(%ebp), %ecs

//wait for disk to be ready

waitdisk();

7c83: e8 e2 ff ff ff call 7c6a <waitdisk>

}

0x7c6a calls waitdisk fcn 🡪 push %ebp

0x7c6b calls insl function 🡪 0x7c6b to 0x7c78

5. Trace through the rest of readsect() and back out into bootmain(), and identify the begin and end of the for loop that reads the remaining sectors of the kernel from the disk.

Beginning: 0x7d51

End: 7d66

6. Find out what code will run when the loop is finished, set a breakpoint there, and continue to that breakpoint. Then step through the remainder of the boot loader.

When the loop finishes, call the entry point from the ELF header (0x7d6b, call \*0x10018).