Simon Chen
Intro To OS
Homework 2
Spring 2017

1)

the $ prefix is for immidiates (constants), and the % prefix is for registers (they are *required*).

[f000:fff0] 0xffff0: ljmp $0xf000,$0xe05b

0xfe05b: cmpl $0x0,%cs:0x6574
Compare check if the value at x6574 is 0

0xfe062: jne 0xfd2b6
jump into fd2b6 if value at x6574 is not 0;
Since next memeory locaiton is still fe066, it didn't jump

0xfe066: xor %ax,%ax
Checks if %ax, %ax are different
They are the same thing, so the xor returns a false

0xfe068: mov %ax,%ss
mov %ss into %ax
Moves the value at ss (segment register) into the register ax

Skipped because the previous condition was false

0xfe06a: mov $0x7000,%esp
move %esp into $0x70000
Moves value in %esp(stack pointer) into 0x7000

0xfe070: mov $0xf3c24,%edx
move %edx into $0xf3c24
Moves value in %edx(data) into 0xf3c24

0xfe076: jmp 0xfd124
jump into fd124

Assumed unconditional control jump transfer; so the next control transfer instruction

2)
At what point does the processor start executing 32-bit code? What exactly causes the switch from 16 to 32-bit mode?

It transitions from executing 16-bit code to 32-bit when it performs mov &cr0,& eax

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

The last instruction of the boot loader is 0x7d61: call 0x10018. The first instruction from the kernet is movw $0x1234,0x472.

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 depends the sectors by reading into the kernel file. It starts at 0x10000c and reads into the kernel's first 8; so 8 sectors. The kernel has an ELF header at the boot loader uses to read.

EIP & CS
ljmp $0xf000,$0xe05b
eip 0xfff0 0xfff0
cs 0xf000 61440

0xfe05b: cmpl $0x0,%cs:0x6574
eip 0xe05b 0xe05b
cs 0xf000 61440
3)

Assume int a[4] is stored at memory location 0x40000 first

```
void
f(void)
{
   int a[4];
   int *b = malloc(16);
   int *c;
   int i;

   printf("1: a = %p, b = %p, c = %p\n", a, b, c);
        // 1: a = 0x7ffcb5cb30b0, b = 0x227f010, c = 0x40074d

   c = a;
   for (i = 0; i < 4; i++)
          a[i] = 100 + i;
   c[0] = 200;
   printf("2: a[0] = %d, a[1] = %d, a[2] = %d, a[3] = %d\n",
          a[0], a[1], a[2], a[3]);
        //2: a[0] = 200, a[1] = 101, a[2] = 102, a[3] = 103

   c[1] = 300;
   *(c + 2) = 301;
   3[c] = 302;
   printf("3: a[0] = %d, a[1] = %d, a[2] = %d, a[3] = %d\n",
          a[0], a[1], a[2], a[3]);
        //3: a[0] = 200, a[1] = 300, a[2] = 301, a[3] = 302

   c = c + 1;
```

```
    *c = 400;
    printf("4: a[0] = %d, a[1] = %d, a[2] = %d, a[3] = %d\n",
            a[0], a[1], a[2], a[3]);
          //4: a[0] = 200, a[1] = 400, a[2] = 301, a[3] = 302

    c = (int *) ((char *) c + 1);
    *c = 500;
```

Xv86 Diagram

(break 31)

rax	0x32	50
rbx	0x0	0
rcx	0x31	49
rdx	0x7ffff7dd59e0	140737351866848
rsi	0x7fffffce	2147483598
rdi	0x1	1
rbp	0x7fffffffde50	0x7fffffffde50
rsp	0x7fffffffde20	0x7fffffffde20
r8	0x7ffff7b8b9c0	140737349466560
r9	0x0	0
r10	0x7ffff7dd26a0	140737351853728
r11	0x246	582
r12	0x400490	4195472
r13	0x7fffffffdf50	140737488346960
r14	0x0	0
r15	0x0	0
rip	0x40067e	0x40067e <f+257>
eflags	0x206	[PF IF]
cs	0x33	51
ss	0x2b	43
ds	0x0	0
es	0x0	0
fs	0x0	0
gs	0x0	0

(break 34)

rax	0x7fffffffde45	140737488346693
rbx	0x0	0
rcx	0x31	49
rdx	0x7ffff7dd59e0	140737351866848
rsi	0x7fffffce	2147483598
rdi	0x1	1
rbp	0x7fffffffde50	0x7fffffffde50
rsp	0x7fffffffde20	0x7fffffffde20
r8	0x7ffff7b8b9c0	140737349466560
r9	0x0	0
r10	0x7ffff7dd26a0	140737351853728
r11	0x246	582

```
r12            0x400490  4195472
r13            0x7fffffffdf50        140737488346960
r14            0x00
r15            0x00
rip            0x40068d  0x40068d <f+272>
eflags          0x202    [ IF ]
cs             0x33      51
ss             0x2b      43
ds             0x0 0
es             0x0 0
fs             0x0 0
gs             0x0 0
```

4)
The kernel starts at 0xf0100000 which is passed to the readseg function. It turns into 0xFFFFFF, because it loads the kernel text program. The first 8 words contains the first 8 words of the kernel program, this is why it's split into 8 sectors. This means the memory address is saved into the kernel to help load it. When it enters the bootloader, there are no values, but once it enters the kernel values are saved.