CS 344 Operating Systems Laboratory Assignment 1

TEAM MEMBERS:

NAME	ROLL NUMBER
RITWIK GANGULY	180101067
PULKIT CHANGOIWALA	180101093
SAMAY VARSHNEY	180101097
UDANDARAO SAI SANDEEP	180123063

EXERCISE 1:

Modified Program ex1.c which now includes inline assembly that increments the value of x by 1.

```
#include<stdio.h>
int main(int argc, char **argv)
{
        int x=1;
        printf(“Hello x = %d\n”, x);
        asm("incl %0": "+r"(x));
        /*
        First method to increment the value of x using inline assembly language having one line
        asm("incl %0": "+r"(x));
            1) incl command adds 1 to the 32-bit contents of the variable specified
            2) %0 refers to the first variable passed (which in this case is x)
            3) the ‘+’ sign before r denotes that x acts as both input and output
        */
        /*
        Second method to increment the value of x using inline assembly language having
        multiple  lines (uncomment to use this)
                asm(“mov %1, %0 \n\t”
                “add $1, %0”
                :  “=r” (x)
                :  “r” (x));
        */
        printf(“Hello x = %d after increment\n”, x);
        if(x == 2) {
                printf(“OK\n”);
```

```
        } else {
                printf("ERROR\n");
        }
}
```

EXERCISE 2:

Explanation:

1st instruction: [f000:fff0] 0xffff0: ljmp $0x3630,$0xf000e05b

- Jump to CS = $0xf000 & IP = 0xe05b
- 0x3630 is jump to this CS (earlier in the BIOS)
- 0xf000e05b is the IP which is different from the lab because it is 32 bits rather than 16 bits and that is all the way into the top of the extended memory location but before the memory mapped PCI device location reserved by the BIOS

2nd Instruction: [f000:e05b] 0xfe05b: cmpw $0xffc8,%cs:(%esi)

- Compare content at 0xffc8 & with content at code segment offset with value at esi.
- esi:- 32-bit source index register

3rd Instruction: [f000:e062] 0xfe062: jne 0xd241d0b2

- Jump to 0xd241d0b2 if the above comparison does not set ZF

4th instruction: [f000:e066] 0xfe066: xor %edx,%edx

- ZF was set thus jump of previous instruction doesn't occur
- It set edx to zero, edx is 32-bit general-purpose register.

5th instruction: [f000:e068] 0xfe068: mov %edx,%ss

- Move content of stack segment register(ss) to edx

6th instruction: [f000:e06a] 0xfe06a: mov $0x7000,%sp

- Move content at the location pointed 16-bit stack pointer(sp) to $0x7000

```
The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp   $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
+ symbol-file kernel
warning: A handler for the OS ABI "GNU/Linux" is not built into this configurati
on
of GDB.  Attempting to continue with the default i8086 settings.

(gdb) si
[f000:e05b]    0xfe05b: cmpw   $0xffc8,%cs:(%esi)
0x0000e05b in ?? ()
(gdb) si
[f000:e062]    0xfe062: jne    0xd241d416
0x0000e062 in ?? ()
(gdb) si
[f000:e066]    0xfe066: xor    %edx,%edx
0x0000e066 in ?? ()
(gdb) si
[f000:e068]    0xfe068: mov    %edx,%ss
0x0000e068 in ?? ()
(gdb) si
[f000:e06a]    0xfe06a: mov    $0x7000,%sp
0x0000e06a in ?? ()
```

EXERCISE 3:

a) The command **ljmp $(SEG_KCODE<<3), $start32** causes the switch from 16 to 32-bit mode in the bootasm.S.

```
(gdb) b *0x7c29
Breakpoint 1 at 0x7c29
(gdb) c
Continuing.
[   0:7c29] => 0x7c29:   mov    %eax,%cr0

Thread 1 hit Breakpoint 1, 0x00007c29 in ?? ()
(gdb) si
[   0:7c2c] => 0x7c2c:   ljmp   $0xb866,$0x87c31
0x00007c2c in ?? ()
(gdb) si
The target architecture is assumed to be i386
=> 0x7c31:      mov    $0x10,%ax
0x00007c31 in ?? ()
(gdb) x/20i 0x7c29
   0x7c29:      mov    %eax,%cr0
   0x7c2c:      ljmp   $0xb866,$0x87c31
   0x7c33:      adc    %al,(%eax)
   0x7c35:      mov    %eax,%ds
   0x7c37:      mov    %eax,%es
   0x7c39:      mov    %eax,%ss
   0x7c3b:      mov    $0x0,%ax
   0x7c3f:      mov    %eax,%fs
   0x7c41:      mov    %eax,%gs
   0x7c43:      mov    $0x7c00,%esp
   0x7c48:      call   0x7d3b
   0x7c4d:      mov    $0x8a00,%ax
   0x7c51:      mov    %ax,%dx
   0x7c54:      out    %ax,(%dx)
   0x7c56:      mov    $0x8ae0,%ax
   0x7c5a:      out    %ax,(%dx)
   0x7c5c:      jmp    0x7c5c
   0x7c5e:      xchg   %ax,%ax
   0x7c60:      add    %al,(%eax)
   0x7c62:      add    %al,(%eax)
```

b) Last Instruction of boot loader executed:

in bootmain.c it is:

entry = (void(*)(void))(elf->entry);
entry();

in bootblock.asm it is:

7d87: ff 15 18 00 01 00 call *0x10018

The First Instruction of Kernel it just loaded is:

0x10000c : mov %cr4,%eax

Also the first instruction of the kernel should be at **0x10018**.

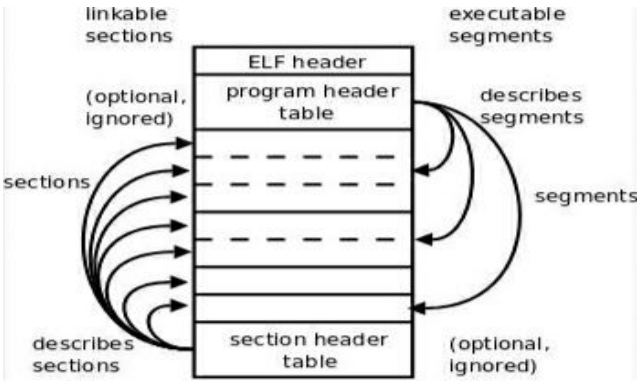
```
(gdb) b *0x7c00
Breakpoint 1 at 0x7c00
(gdb) c
Continuing.
[   0:7c00] => 0x7c00:  cli

Thread 1 hit Breakpoint 1, 0x00007c00 in ?? ()
(gdb) b *0x7d87
Breakpoint 2 at 0x7d87
(gdb) c
Continuing.
The target architecture is assumed to be i386
=> 0x7d87:      call   *0x10018

Thread 1 hit Breakpoint 2, 0x00007d87 in ?? ()
(gdb) si
=> 0x10000c:    mov    %cr4,%eax
0x0010000c in ?? ()
(gdb) x/1x 0x10018
0x10018:        0x0010000c
```

c) The boot loader reads the number the program headers in the ELF header and loads them all:

```
ph = (struct proghdr*)((uchar*)elf + elf->phoff);
eph = ph + elf->phnum;
for(; ph < eph; ph++){
        pa = (uchar*)ph->paddr;
        readseg(pa, ph->filesz, ph->off);
        if(ph->memsz > ph->filesz)
        stosb(pa + ph->filesz, 0, ph->memsz - ph->filesz);
}
```



EXERCISE 5:

When boot loader's link address is 0x7C00 then commands are running properly and transition from 16 to 32 bit was occurring at **0x7C31** address location as seen below:

```
(gdb) b *0x7c00
Breakpoint 1 at 0x7c00
(gdb) c
Continuing.
[   0:7c00] => 0x7c00:  cli

Thread 1 hit Breakpoint 1, 0x00007c00 in ?? ()
(gdb) b *0x7c31
Breakpoint 2 at 0x7c31
(gdb) c
Continuing.
The target architecture is assumed to be i386
=> 0x7c31:      mov    $0x10,%ax

Thread 1 hit Breakpoint 2, 0x00007c31 in ?? ()
```

But when the boot loader's link address is changed to any other address (we took **0x7C24** in this case), after running
make clean
make
and restarting gdb
and continuing from address location 0x7C00,
then the boot loader is restarting again and again after running some instructions in the gdb.

```
(gdb) b *0x7c00
Breakpoint 1 at 0x7c00
(gdb) c
Continuing.
[   0:7c00] => 0x7c00:  cli

Thread 1 hit Breakpoint 1, 0x00007c00 in ?? ()
(gdb) c
Continuing.
[   0:7c00] => 0x7c00:  cli

Thread 1 hit Breakpoint 1, 0x00007c00 in ?? ()
(gdb) b *0x7c55
Breakpoint 2 at 0x7c55
(gdb) c
Continuing.
[   0:7c00] => 0x7c00:  cli

Thread 1 hit Breakpoint 1, 0x00007c00 in ?? ()
(gdb) si
[   0:7c01] => 0x7c01:  xor    %eax,%eax
0x00007c01 in ?? ()
(gdb) si
[   0:7c03] => 0x7c03:  mov    %eax,%ds
0x00007c03 in ?? ()
(gdb) c
Continuing.
[   0:7c00] => 0x7c00:  cli

Thread 1 hit Breakpoint 1, 0x00007c00 in ?? ()
```

As seen in the image above, we tried to run commands after continuing from breakpoint at 0x7C00 address location and we always end up hitting the same breakpoint at 0x7C00.
Also 16 to 32 bit architecture change didn't occured as breakpoint **b *0x7C55** is not hitted which should be responsible for architecture change in this case.

ljmp $(SEG_KCODE<<3), $start32 is the first instruction that breaks.
Before changing the link address of the boot loader, from address 0x7C00, after performing 2-3 **si 10** instructions, architecture changed from 16 to 32 bit.
But after changing the link address to 0x7C24, architecture didn't change which means that the boot loader is not loaded properly at the changed link address.

EXERCISE 6:

At the point when BIOS enters the boot loader (at first breakpoint):

```
(gdb) b *0x7c00
Breakpoint 1 at 0x7c00
(gdb) c
Continuing.
[   0:7c00] => 0x7c00:  cli

Thread 1 hit Breakpoint 1, 0x00007c00 in ?? ()
(gdb) x/8x 0x00100000
0x100000:       0x00000000      0x00000000      0x00000000      0x00000000
0x100010:       0x00000000      0x00000000      0x00000000      0x00000000
(gdb) x/8i 0x00100000
   0x100000:    add     %al,(%eax)
   0x100002:    add     %al,(%eax)
   0x100004:    add     %al,(%eax)
   0x100006:    add     %al,(%eax)
   0x100008:    add     %al,(%eax)
   0x10000a:    add     %al,(%eax)
   0x10000c:    add     %al,(%eax)
   0x10000e:    add     %al,(%eax)
```

At the point when the boot loader enters the kernel (at second breakpoint):

```
(gdb) b *0x7d87
Breakpoint 2 at 0x7d87
(gdb) c
Continuing.
The target architecture is assumed to be i386
=> 0x7d87:      call   *0x10018

Thread 1 hit Breakpoint 2, 0x00007d87 in ?? ()
(gdb) x/8x 0x00100000
0x100000:       0x1badb002      0x00000000      0xe4524ffe      0x83e0200f
0x100010:       0x220f10c8      0x9000b8e0      0x220f0010      0xc0200fd8
(gdb) x/8i 0x00100000
   0x100000:    add     0x1bad(%eax),%dh
   0x100006:    add     %al,(%eax)
   0x100008:    decb    0x52(%edi)
   0x10000b:    in      $0xf,%al
   0x10000d:    and     %ah,%al
   0x10000f:    or      $0x10,%eax
   0x100012:    mov     %eax,%cr4
   0x100015:    mov     $0x109000,%eax
```

8 words of instruction at 0x00100000 at the point when BIOS enters the boot loader and 8 words of instruction at 0x00100000 at the point when the boot loader enters the kernel are different as when the BIOS enters and loads the boot loader, then it just loads it in memory location between 0x7C00 and 0x7DFF due to which all the 8 words of instructions are zero at 0x00100000. But before the boot loader enters the kernel, it already has performed the 16 to 32 bit transition and setting up of stack which leads to new instructions at address 0x00100000.