Assignment 0

Abhishek Agrahari

Roll No. = 190123066

Q1: Modified ex1.c file is present in the submitted folder. Following modification is done.

```
__asm__(
    "addl $1, %%eax;\n\t"
    : "=a" (x)
    : "a" (x)
);
```

Syntax : __asm__("assembly code": output operands : input operands);

On running ex1.c we get the following output.

```
abhishek@LAPTOP-UE6RCUDL:/mnt/c/Users/abhis/OneDrive/Desktop/5th Semester/OS/LAB$ gcc ex1.c
abhishek@LAPTOP-UE6RCUDL:/mnt/c/Users/abhis/OneDrive/Desktop/5th Semester/OS/LAB$ ./a.out
Hello x = 1
Hello x = 2 after increment
OK
abhishek@LAPTOP-UE6RCUDL:/mnt/c/Users/abhis/OneDrive/Desktop/5th Semester/OS/LAB$
```

Q2:

```
abhishek@LAPTOP-UE6RCUDL: /mnt/c/Users/abhis/OneDrive/Desktop/xv6-public
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
        info "(gdb)Auto-loading safe path"
(gdb) source .gdbinit
+ target remote localhost:26000
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
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 configuration
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    0xd241d0b2
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 ?? ()
(gdb) si
[f000:e070]    0xfe070: mov    $0x7c4,%dx
0x0000e070 in ?? ()
(gdb) si
[f000:e076]    0xfe076: jmp    0x5576cf26
0x0000e076 in ?? ()
(gdb) si
[f000:cf24]    0xfcf24: cli
0x0000cf24 in ?? ()
(gdb) si
[f000:cf25]    0xfcf25: cld
0x0000cf25 in ?? ()
(gdb) si
[f000:cf26]    0xfcf26: mov    %ax,%cx
0x0000cf26 in ?? ()
(gdb) si
[f000:cf29]    0xfcf29: mov    $0x8f,%ax
0x0000cf29 in ?? ()
```

Instruction 1: compare operands at given address.

Instruction 2: jump if values at given address are not equal.

Instruction 3: take xor of two operands, in this case sets edx = 0.

Instruction 4: loads value in register ss(stack segment) with edx which is 0.

Instruction 5: loads value 0x7000in register sp.

Instruction 6: loads value 0x7c4 in register dx.

Instruction 7: jump at a address that is stored in given address..

Instruction 8:clear interrupt flag.

Instruction 9:clear direction flag..

Instruction 10: loads value of register ax with value of register cx.

Instruction 11:loads value 0x8f in register ax.

EXERCISE 3 : Comparing instructions near 0x7C00 location :

In Bootasm.S (source code):

```
.code16                      # Assemble for 16-bit mode
.globl start
start:
  cli                        # BIOS enabled interrupts; disable

  # Zero data segment registers DS, ES, and SS.
  xorw    %ax,%ax            # Set %ax to zero
  movw    %ax,%ds            # -> Data Segment
  movw    %ax,%es            # -> Extra Segment
  movw    %ax,%ss            # -> Stack Segment

  # Physical address line A20 is tied to zero so that the first PCs
  # with 2 MB would run software that assumed 1 MB.  Undo that.
seta20.1:
  inb     $0x64,%al              # Wait for not busy
  testb   $0x2,%al
  jnz     seta20.1

  movb    $0xd1,%al              # 0xd1 -> port 0x64
  outb    %al,$0x64
```

In Bootblock.asm

```
.code16                       # Assemble for 16-bit mode
.globl start
start:
  cli                         # BIOS enabled interrupts; disable
    7c00: fa                    cli

  # Zero data segment registers DS, ES, and SS.
  xorw    %ax,%ax             # Set %ax to zero
    7c01: 31 c0                 xor    %eax,%eax
  movw    %ax,%ds             # -> Data Segment
    7c03: 8e d8                 mov    %eax,%ds
  movw    %ax,%es             # -> Extra Segment
    7c05: 8e c0                 mov    %eax,%es
  movw    %ax,%ss             # -> Stack Segment
    7c07: 8e d0                 mov    %eax,%ss

00007c09 <seta20.1>:

  # Physical address line A20 is tied to zero so that the first PCs
  # with 2 MB would run software that assumed 1 MB.  Undo that.
seta20.1:
  inb     $0x64,%al               # Wait for not busy
    7c09: e4 64                 in     $0x64,%al
  testb   $0x2,%al
    7c0b: a8 02                 test   $0x2,%al
  jnz     seta20.1
    7c0d: 75 fa                 jne    7c09 <seta20.1>

  movb    $0xd1,%al               # 0xd1 -> port 0x64
    7c0f: b0 d1                 mov    $0xd1,%al
  outb    %al,$0x64
    7c11: e6 64                 out    %al,$0x64
```

In GDB:

```
(gdb) x/10i 0x7c00
=> 0x7c00:      cli
   0x7c01:      xor    %eax,%eax
   0x7c03:      mov    %eax,%ds
   0x7c05:      mov    %eax,%es
   0x7c07:      mov    %eax,%ss
   0x7c09:      in     $0x64,%al
   0x7c0b:      test   $0x2,%al
   0x7c0d:      jne    0x7c09
   0x7c0f:      mov    $0xd1,%al
   0x7c11:      out    %al,$0x64
```

Showing first 10 instruction from location 0x7c00 onwards. On comparing the 3 images, one can see that these instruction are very similar except some minor differece in notations.

Readsect() in bootblock.asm

```
void
readsect(void *dst, uint offset)
{
    7c90: f3 0f 1e fb           endbr32
    7c94: 55                    push   %ebp
    7c95: 89 e5                 mov    %esp,%ebp
    7c97: 57                    push   %edi
    7c98: 53                    push   %ebx
    7c99: 8b 5d 0c              mov    0xc(%ebp),%ebx
  // Issue command.
  waitdisk();
    7c9c: e8 dd ff ff ff        call   7c7e <waitdisk>
}
```

Readsect in bootmain.c

```
void
readsect(void *dst, uint offset)
{
  // Issue command.
  waitdisk();
  outb(0x1F2, 1);   // count = 1
  outb(0x1F3, offset);
  outb(0x1F4, offset >> 8);
  outb(0x1F5, offset >> 16);
  outb(0x1F6, (offset >> 24) | 0xE0);
  outb(0x1F7, 0x20);  // cmd 0x20 - read sectors

  // Read data.
  waitdisk();
  insl(0x1F0, dst, SECTSIZE/4);
}
```

Reading remaining sectors of the kernel, bootblock.asm

```
315     for(; ph < eph; ph++){
316       7d8d: 39 f3                   cmp    %esi,%ebx
317       7d8f: 72 15                   jb     7da6 <bootmain+0x5d>
318     entry();
319       7d91: ff 15 18 00 01 00       call   *0x10018
320   }
321       7d97: 8d 65 f4                lea    -0xc(%ebp),%esp
322       7d9a: 5b                      pop    %ebx
323       7d9b: 5e                      pop    %esi
324       7d9c: 5f                      pop    %edi
325       7d9d: 5d                      pop    %ebp
326       7d9e: c3                      ret
327     for(; ph < eph; ph++){
328       7d9f: 83 c3 20                add    $0x20,%ebx
329       7da2: 39 de                   cmp    %ebx,%esi
330       7da4: 76 eb                   jbe    7d91 <bootmain+0x48>
331       pa = (uchar*)ph->paddr;
332       7da6: 8b 7b 0c                mov    0xc(%ebx),%edi
333       readseg(pa, ph->filesz, ph->off);
334       7da9: 83 ec 04                sub    $0x4,%esp
335       7dac: ff 73 04                pushl  0x4(%ebx)
336       7daf: ff 73 10                pushl  0x10(%ebx)
337       7db2: 57                      push   %edi
338       7db3: e8 44 ff ff ff          call   7cfc <readseg>
339       if(ph->memsz > ph->filesz)
340       7db8: 8b 4b 14                mov    0x14(%ebx),%ecx
341       7dbb: 8b 43 10                mov    0x10(%ebx),%eax
342       7dbe: 83 c4 10                add    $0x10,%esp
343       7dc1: 39 c1                   cmp    %eax,%ecx
344       7dc3: 76 da                   jbe    7d9f <bootmain+0x56>
345         stosb(pa + ph->filesz, 0, ph->memsz - ph->filesz);
346       7dc5: 01 c7                   add    %eax,%edi
347       7dc7: 29 c1                   sub    %eax,%ecx
348   }
```

The instructions from line 327 to line 348 read the remaining sectors of the kernel from the disk. When loop is finished, instruction in line 319 call *0x10018 is executed.

```
(gdb) b *0x7d91
Breakpoint 2 at 0x7d91
(gdb) c
Continuing.
The target architecture is assumed to be i386
=> 0x7d91:      call   *0x10018
```

a) In the bootasm.S file, **ljmp $(SEG_KCODE<<3), $start32** instruction is where the processor starts executing 32-bit code.

The lgdt command causes a switch from 16-32-bit mode.

```
# Switch from real to protected mode.  Use a bootstrap GDT that makes
# virtual addresses map directly to physical addresses so that the
# effective memory map doesn't change during the transition.
lgdt    gdtdesc
movl    %cr0, %eax
orl     $CR0_PE, %eax
movl    %eax, %cr0
```

b) **<u>Last instruction of the boot loader executed</u>**:

In <u>bootmain.c:</u>

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

entry();

In <u>bootblock.asm</u>

7d91: ff 15 18 00 01 00   call   *0x10018

<u>The first instruction of Kernel it just loaded is:</u>

0x10000c: mov %cr4,%eax

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

Thread 1 hit Breakpoint 1, 0x00007c00 in ?? ()
(gdb) b *0x7d91
Breakpoint 2 at 0x7d91
(gdb) c
Continuing.
The target architecture is assumed to be i386
=> 0x7d91:      call   *0x10018

Thread 1 hit Breakpoint 2, 0x00007d91 in ?? ()
(gdb) si
=> 0x10000c:    mov    %cr4,%eax
0x0010000c in ?? ()
(gdb)
```

c) The information about the number of sectors it must read to fetch the entire kernel is stored in phnum attribute of **ELF binary header**. Here ph initially points to the program header and iterate till eph which points to the last sector.
(screentshot taken from bootmain.c)

```
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 4

$ objdump -h kernel

```
abhishek@LAPTOP-UE6RCUDL:/mnt/c/Users/abhis/OneDrive/Desktop/xv6-public$ objdump -h kernel

kernel:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000713a  80100000  00100000  00001000  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .rodata       00000ff3  80107140  00107140  00008140  2**5
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .data         00002516  80109000  00109000  0000a000  2**12
                  CONTENTS, ALLOC, LOAD, DATA
  3 .bss          0000af88  8010b520  0010b520  0000c516  2**5
                  ALLOC
  4 .debug_line   00006cfd  00000000  00000000  0000c516  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  5 .debug_info   00012258  00000000  00000000  00013213  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  6 .debug_abbrev 00004007  00000000  00000000  0002546b  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  7 .debug_aranges 000003a8  00000000  00000000  00029478  2**3
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  8 .debug_str    00000ed7  00000000  00000000  00029820  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  9 .debug_loc    00006833  00000000  00000000  0002a6f7  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 10 .debug_ranges 00000d08  00000000  00000000  00030f2a  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 11 .comment      0000002a  00000000  00000000  00031c32  2**0
                  CONTENTS, READONLY
abhishek@LAPTOP-UE6RCUDL:/mnt/c/Users/abhis/OneDrive/Desktop/xv6-public$
```

VMA and LMA address at the .text section of the kernel is different.

VMA: Link address of the section

LMA: Load address of the section

The **load address** of a section is the memory address at which that section should be loaded into memory.

The **link address** of a section is the memory address from which the section expects to execute.

$ objdump -h bootblock.o

```
abhishek@LAPTOP-UE6RCUDL:/mnt/c/Users/abhis/OneDrive/Desktop/xv6-public$ objdump -h bootblock.o

bootblock.o:     file format elf32-i386

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000001d3  00007c00  00007c00  00000074  2**2
                  CONTENTS, ALLOC, LOAD, CODE
  1 .eh_frame     000000b0  00007dd4  00007dd4  00000248  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .comment      0000002a  00000000  00000000  000002f8  2**0
                  CONTENTS, READONLY
  3 .debug_aranges 00000040  00000000  00000000  00000328  2**3
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  4 .debug_info   000005d2  00000000  00000000  00000368  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  5 .debug_abbrev 0000022c  00000000  00000000  0000093a  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  6 .debug_line   0000029a  00000000  00000000  00000b66  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  7 .debug_str    00000237  00000000  00000000  00000e00  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  8 .debug_loc    000002bb  00000000  00000000  00001037  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  9 .debug_ranges 00000078  00000000  00000000  000012f2  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
abhishek@LAPTOP-UE6RCUDL:/mnt/c/Users/abhis/OneDrive/Desktop/xv6-public$
```

VMA and LMA address at the .text section of the bootloader is same.

Exercise 5-

1) Changed link address from 0x7c00 to 0x7c05 at line 106 of Makefile.

2) Then **make clean** and **make** was executed.

3) Started gdb and set breakpoint at 0x7c00. On doing continue at first breakpoint it comes back to the first breakpoint. This happen again and again in an infinite loop.

```
(gdb) b *0x7C00
Breakpoint 1 at 0x7c00
(gdb) continue
Continuing.
[   0:7c00] => 0x7c00:  xchg   %ax,%ax

Thread 1 hit Breakpoint 1, 0x00007c00 in ?? ()
(gdb) continue
Continuing.
[   0:7c00] => 0x7c00:  xchg   %ax,%ax

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

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

Thread 1 hit Breakpoint 1, 0x00007c00 in ?? ()
(gdb) continue
Continuing.
[   0:7c00] => 0x7c00:  xchg   %ax,%ax

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

$objdump -f kernel

```
abhishek@LAPTOP-UE6RCUDL:/mnt/c/Users/abhis/OneDrive/Desktop/xv6-public$ objdump -f kernel

kernel:     file format elf32-i386
architecture: i386, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x0010000c
```

Exercise 6-

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

```
(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)
(gdb)
```

At the point where the boot loader enters the kernel (at second checkpoint)

```
(gdb) b *0x7d91
Breakpoint 2 at 0x7d91
(gdb) c
Continuing.
The target architecture is assumed to be i386
=> 0x7d91:      call   *0x10018

Thread 1 hit Breakpoint 2, 0x00007d91 in ?? ()
(gdb) x/8x 0x00100000
0x100000:       0x1badb002      0x00000000      0xe4524ffe      0x83e0200f
0x100010:       0x220f10c8      0xa000b8e0      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    $0x10a000,%eax
```

They are different because when BIOS enters the boot loader at breakpoint 1, kernel is not loaded at 0x00100000 therefore shows zero while on the other hand when boot loader enters kernel at the second breakpoint kernel have been loaded at that location therefore it is showing non zero values.