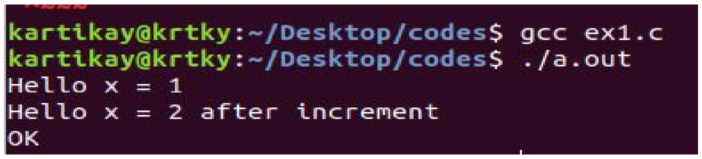
CS343-Assignment 1

Group Number - 17

Name	Roll Number
1. Samiksha Sachdeva	180123040
2. Kartikay Goel	180101033
3. Falak Chhikara	180101023
4. Rahul Choudhary	180101060

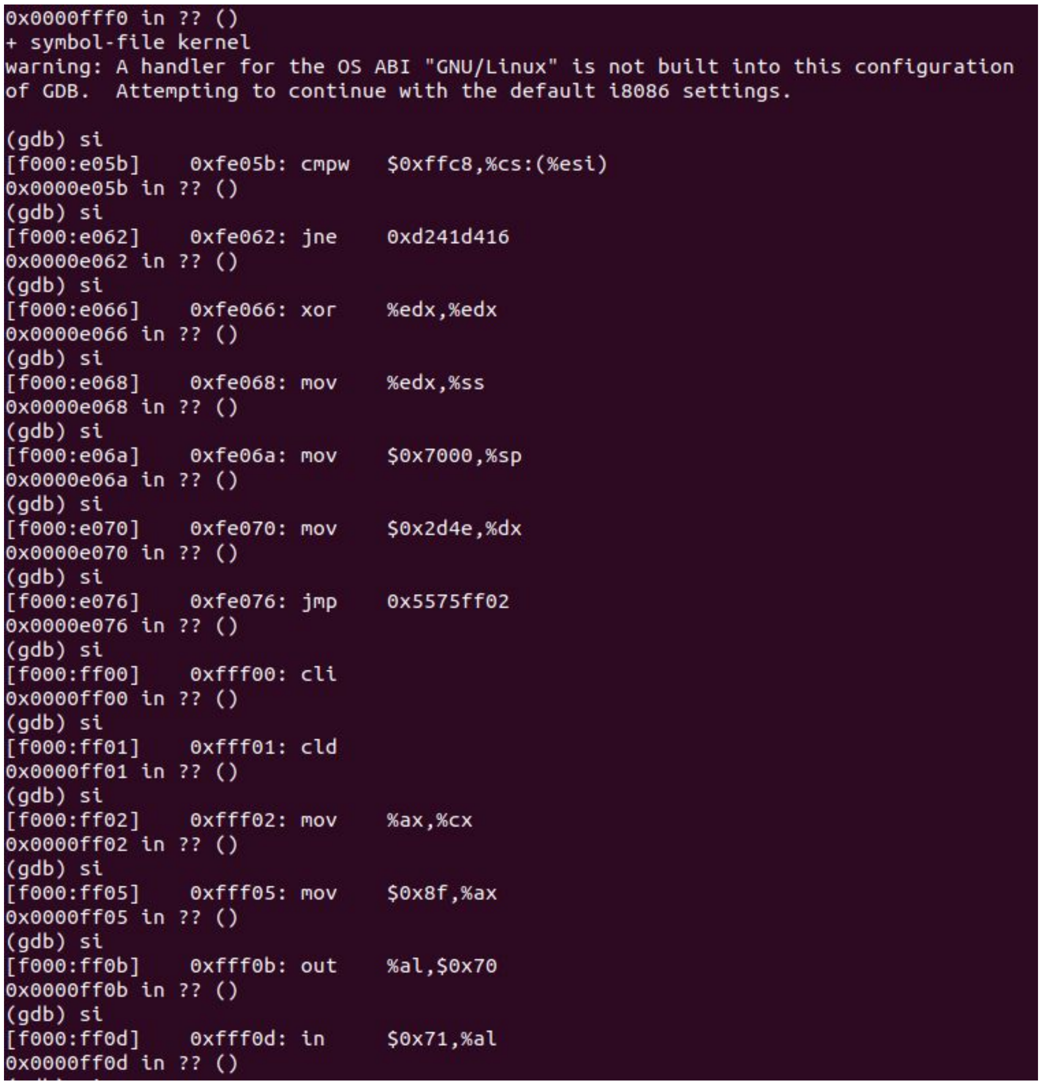
Exercise 1:

We wrote this line of code to increment x by 1 in ex1.c: **asm("inc %0": "+r"(x));**
The **inc** is used to increment the value of the provided argument. "**+r**" denotes that x is the input as well as the output. And **%0** denotes the first and the only argument, x.



```
kartikay@krtky:~/Desktop/codes$ gcc ex1.c
kartikay@krtky:~/Desktop/codes$ ./a.out
Hello x = 1
Hello x = 2 after increment
OK
```

Exercise 2:



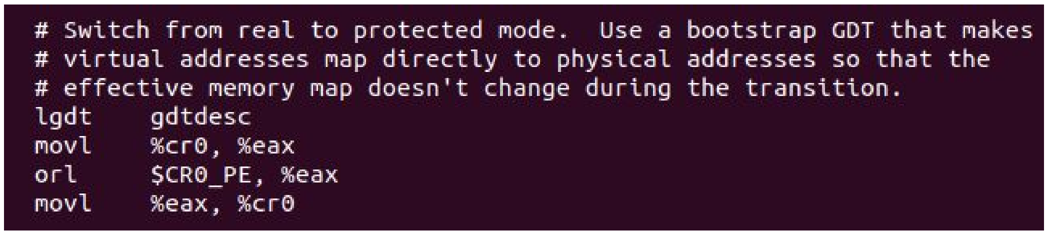
```
0x0000fff0 in ?? ()
+ symbol-file kernel
warning: A handler for the OS ABI "GNU/Linux" is not built into this configuration
of GDB.  Attempting to continue with the default i8086 settings.

(gdb) si
[f000:e05b]    0xfe05b: cmpw   $0xffc8,%cs:(%esi)
0x0000e05b in ?? ()
(gdb) si
[f000:e062]    0xfe062: jne    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 ?? ()
(gdb) si
[f000:e070]    0xfe070: mov    $0x2d4e,%dx
0x0000e070 in ?? ()
(gdb) si
[f000:e076]    0xfe076: jmp    0x5575ff02
0x0000e076 in ?? ()
(gdb) si
[f000:ff00]    0xfff00: cli
0x0000ff00 in ?? ()
(gdb) si
[f000:ff01]    0xfff01: cld
0x0000ff01 in ?? ()
(gdb) si
[f000:ff02]    0xfff02: mov    %ax,%cx
0x0000ff02 in ?? ()
(gdb) si
[f000:ff05]    0xfff05: mov    $0x8f,%ax
0x0000ff05 in ?? ()
(gdb) si
[f000:ff0b]    0xfff0b: out    %al,$0x70
0x0000ff0b in ?? ()
(gdb) si
[f000:ff0d]    0xfff0d: in     $0x71,%al
0x0000ff0d in ?? ()
```

EDX (32 bit general register) is set to 0 using the **xor instruction**.(xor of equal values is 0)
SS (Segment register) is set to 0. (using the **mov instruction**)
SP (Stack pointer) is given the hex address 0x7000.(using the next **mov instruction**). **DX** (16 bit general register) is given the address 0x2d4e.(using the next **mov instruction**).
CMPW is a comparison operator which compares the segmented address to hex address 0xffc8 and performs **jne** (jump if not equal to 0).
CLI, CLD clears the interrupt and direction flags (i.e. sets them to 0).
AX,AL register is used for the I/O port access and here we can see that **OUT instruction** sends a byte (8 bits) [8f = 10001111] to port 0x70 (CMOS RAM index register port) and RTC is configured according to the input it gets. The immediate **IN instruction** is for CMOS exclusively as it requires an immediate instruction from 0x71 port or the RTC will be left in uncertain state.
We may safely conclude that BIOS is sort of a preparation phase before beginning the main booting process from the boot sector.
It is then that BIOS backtracks and initialises all the registers and input/output devices that are accessible from the BIOS itself, until it finds the bootable disk. It then transfers the control to the bootloader.
The next step is booting from the disk. Putting a breakpoint in **0x7c00** halts the further execution of the bootloader and just starts it.

Exercise 3:

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

The **lgdt** command causes the switch from 16 to 32-bit mode. (real to protected mode.)



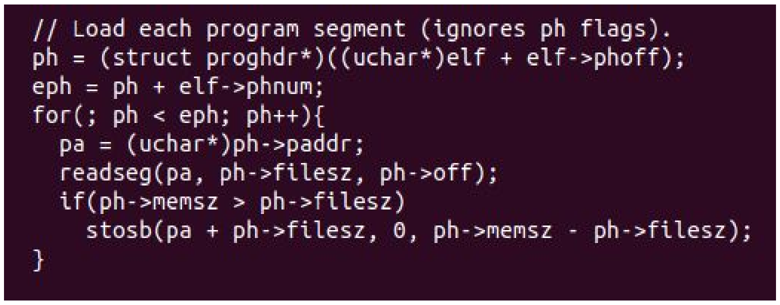
```
# Switch from real to protected mode.  Use a bootstrap GDT that makes
# virtual addresses map directly to physical addresses so that the
# effective memory map doesn't change during the transition.
lgdt    gdtdesc
movl    %cr0, %eax
orl     $CR0_PE, %eax
movl    %eax, %cr0
```

b) Last instruction that BootLoader executes:

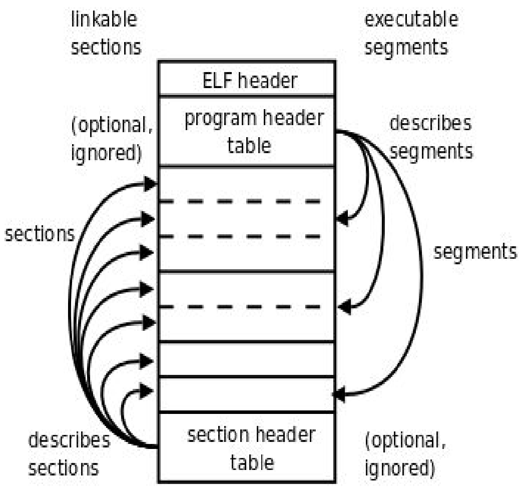
In bootmain.c it is: entry = (void(*)(void))(elf->entry);
In bootblock.asm: 7d87: ff 15 18 00 01 00 call *0x10018

First instruction of the kernel it loaded: **movl %cr4, %eax. The first instruction is present at 0x0010000c.**

c)



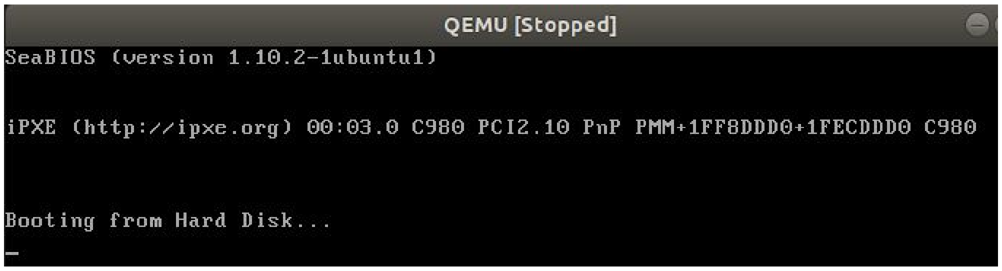
```
// Load each program segment (ignores ph flags).
ph = (struct proghdr*)((uchar*)elf + elf->phoff);
eph = ph + elf->phnum;
for(; ph < eph; ph++){
  pa = (uchar*)ph->paddr;
  readseg(pa, ph->filesz, ph->off);
  if(ph->memsz > ph->filesz)
    stosb(pa + ph->filesz, 0, ph->memsz - ph->filesz);
}
```



ph is a pointer to the program header. **elf** contains the number of sectors required to fetch the entire kernel. **eph** is the pointer to the last sector.
The for loop loads each sector from ph upto eph incrementing ph at each iteration. This information is present in elf headers.

Exercise 5:

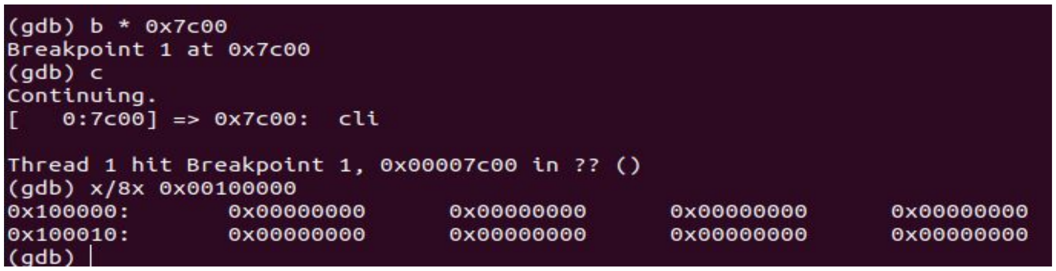
The OS is stuck at booting from the hard disk. The ip(instruction pointer) begins to loop between *6dc1* goes to *eeee* overflows and goes back to *6dc1*.



```
QEMU [Stopped]
SeaBIOS (version 1.10.2-1ubuntu1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF8DDD0+1FECDDD0 C980


Booting from Hard Disk...
_
```

Exercise 6:

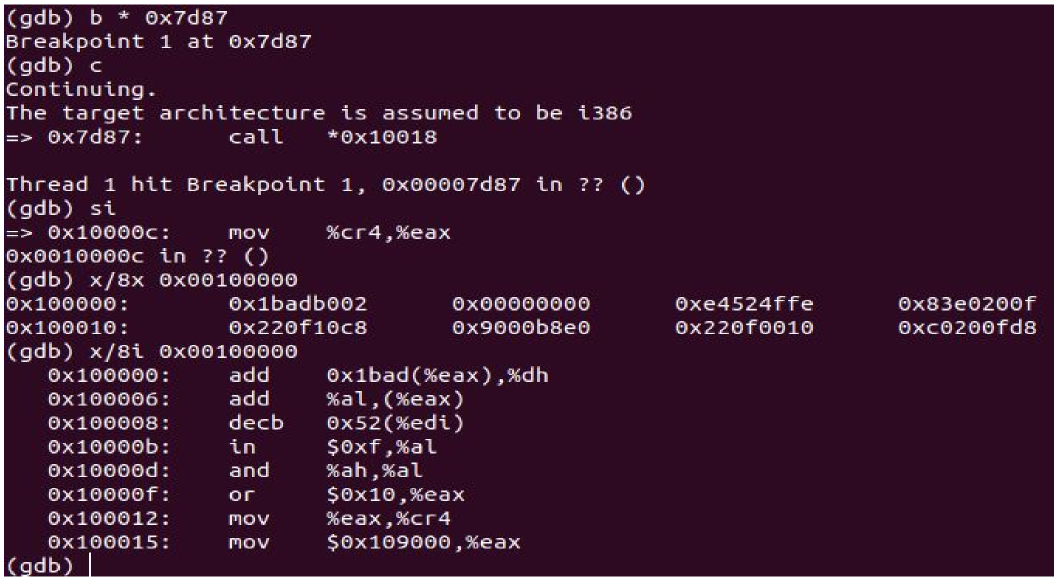
When BIOS enters the boot loader, the 8 words of memory are all zero.



```
(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) |
```

We are examining those memory blocks whose address is higher than the address at which the boot loader is loaded. And since the kernel hasn't been loaded yet, these 8 words in memory are zero.

When Bootloader enters the kernel, the first instruction of the kernel is loaded at *0x0010000c* (as shown in the below screenshot) and hence when we examine the memory near this we find instructions.



```
(gdb) b * 0x7d87
Breakpoint 1 at 0x7d87
(gdb) c
Continuing.
The target architecture is assumed to be i386
=> 0x7d87:      call   *0x10018

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

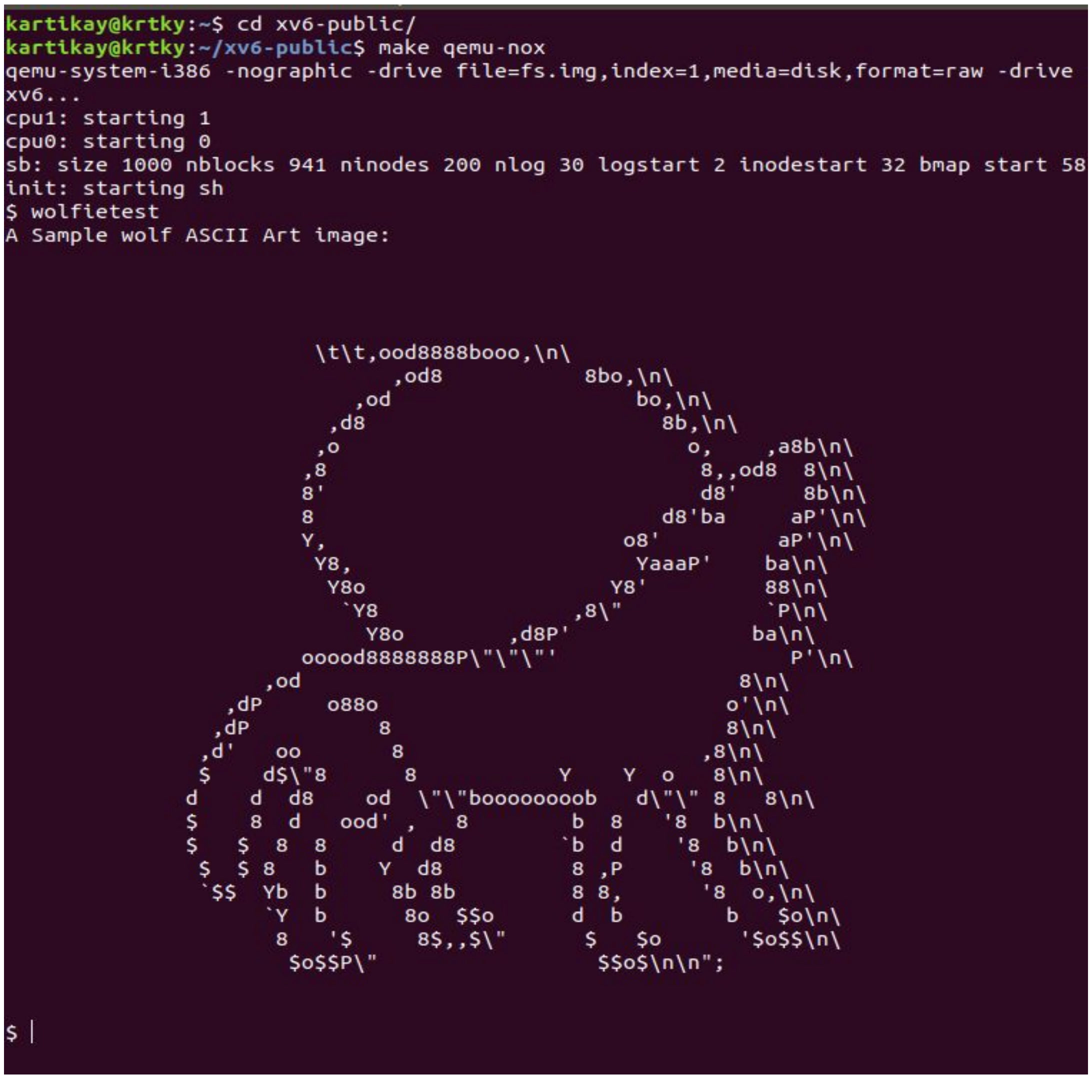
This is because the kernel is not copied till the bootloader is not executed.
The content of the second breakpoint is shown in the screenshot above. (Although in the screenshot, it is written as 'Breakpoint 1', it is actually the content of 'Breakpoint 2' as we have taken separate screenshots for the report purpose.)

Exercise 7 and 8:

Code is submitted as a separate file.

We have made changes in the following files:

1) Makefile
2) syscall.c (added [SYS_wolfie] sys_wolfie   && extern int sys_wolfie(void);)
3) syscall.h (added #define SYS_wolfie 22)
4) sysfile.c (added main function call)
5) user.h (added prototype of function int wolfie(void* ,uint);)
6) usys.S (added SYSCALL(wolfie))
7) wolfietest.c (added caller function)



```
kartikay@krtky:~$ cd xv6-public/
kartikay@krtky:~/xv6-public$ make qemu-nox
qemu-system-i386 -nographic -drive file=fs.img,index=1,media=disk,format=raw -drive
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ wolfietest
A Sample wolf ASCII Art image:



                        \t\t,ood8888booo,\n\
                              ,od8           8bo,\n\
                           ,od                   bo,\n\
                         ,d8                       8b,\n\
                        ,o                           o,    ,a8b\n\
                       ,8                             8,,od8  8\n\
                       8'                             d8'     8b\n\
                       8                           d8'ba     aP'\n\
                       Y,                       o8'         aP'\n\
                        Y8,                      YaaaP'    ba\n\
                         Y8o                   Y8'         88\n\
                          `Y8               ,8\"           `P\n\
                            Y8o        ,d8P'              ba\n\
                       ooood8888888P\"\"\"'                  P'\n\
                    ,od                                  8\n\
                 ,dP     o88o                           o'\n\
                ,dP          8                          8\n\
               ,d'   oo       8                       ,8\n\
               $    d$\"8      8           Y    Y  o   8\n\
              d    d  d8    od  \"\"booooooooob  d\"\" 8   8\n\
              $    8  d   ood' ,   8        b  8   '8  b\n\
              $   $  8  8     d  d8         `b  d    '8  b\n\
               $  $ 8   b    Y  d8          8 ,P     '8  b\n\
               `$$  Yb  b     8b 8b         8 8,      '8  o,\n\
                    `Y  b      8o  $$o      d  b        b   $o\n\
                     8   '$     8$,,$\"      $   $o      '$o$$\n\
                      $o$$P\"                 $$o$\n\n";


$ |
```