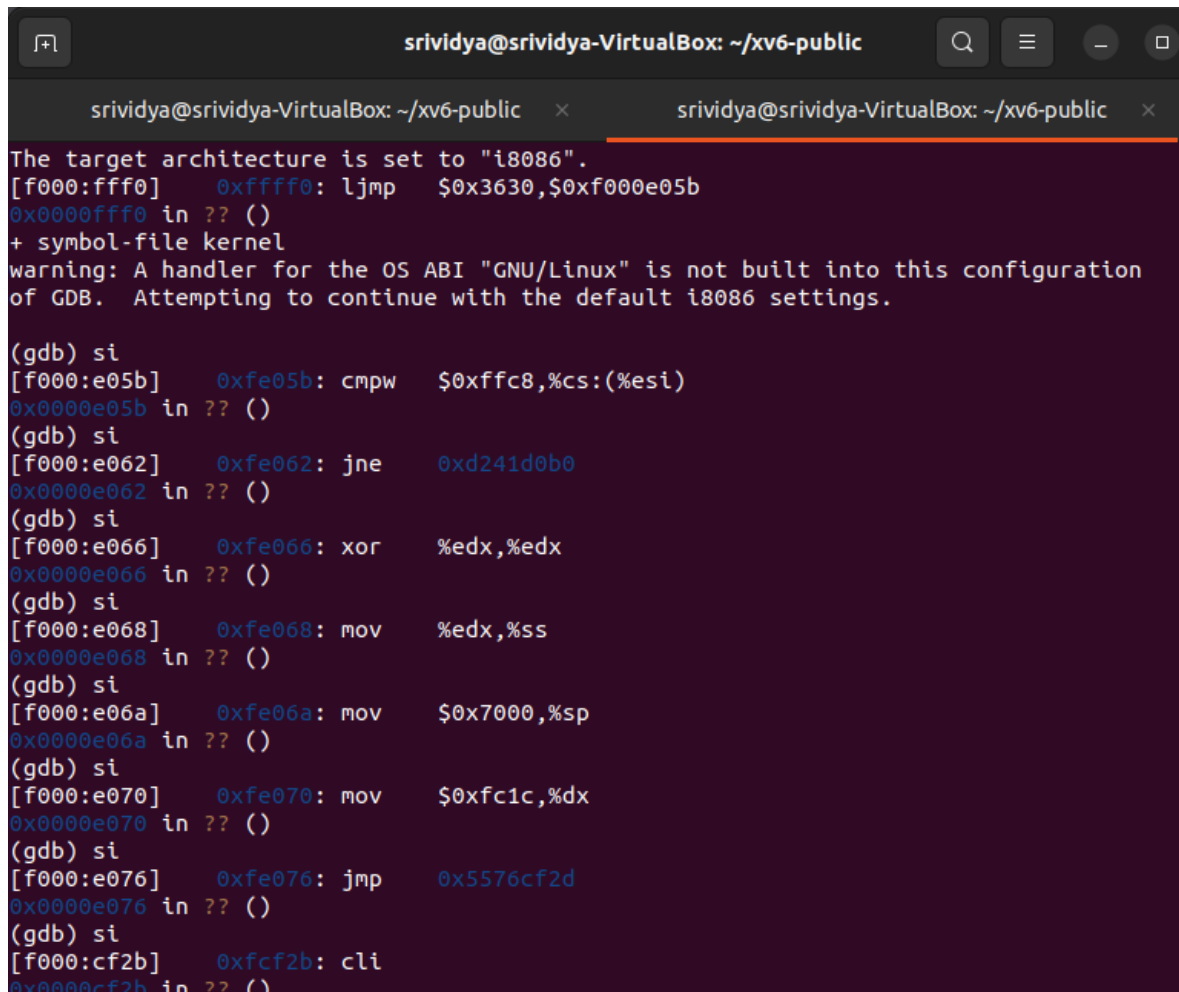


**Exercise 1** - Modified code is in ex1.c file

**Exercise 2** -



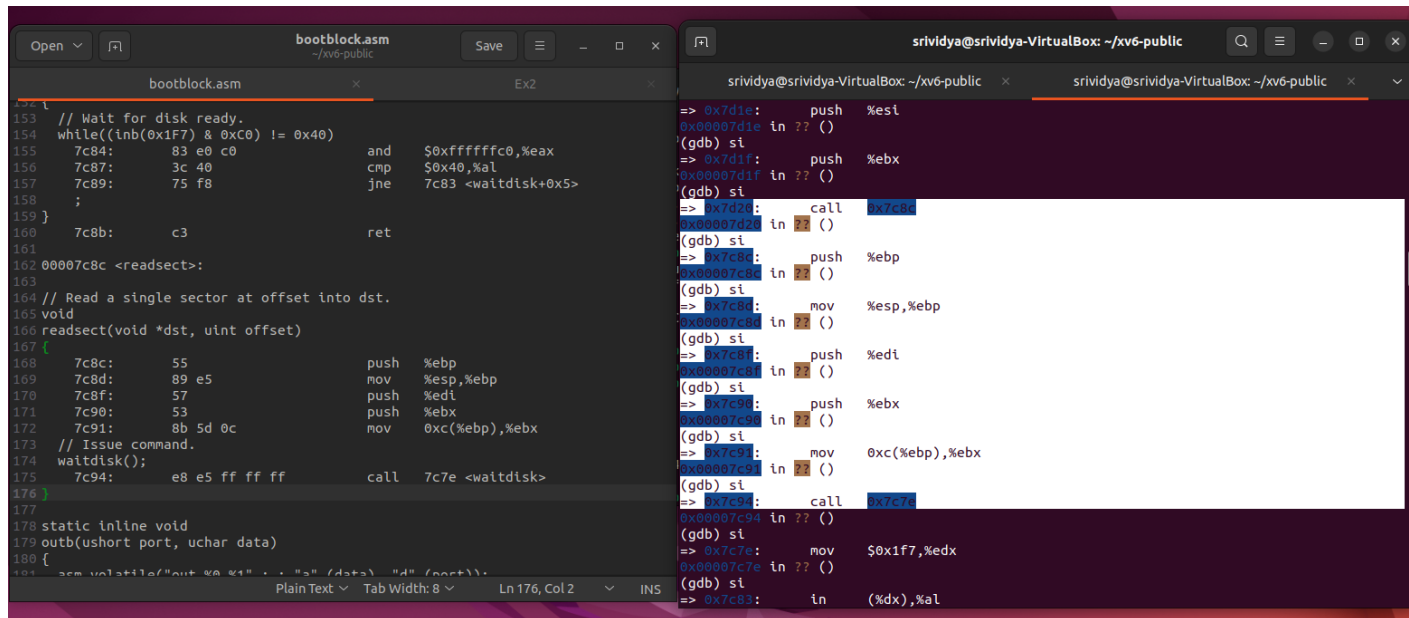
```
srividya@srividya-VirtualBox: ~/xv6-public
The target architecture is set to "i8086".
[f000:fff0] 0xfffff0: ljmp $0x3630,$0xf000e05b
0x0000fff0 in ?? ()
+ symbol-file kernel
warning: A handler for the OS ABI "GNU/Linux" is not built into this configuration
of GDB. Attempting to continue with the default i8086 settings.

(gdb) si
[f000:e05b] 0xfe05b: cmpw $0xffc8,%cs:(%esi)
0x0000e05b in ?? ()
(gdb) si
[f000:e062] 0xfe062: jne 0xd241d0b0
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 $0xfc1c,%dx
0x0000e070 in ?? ()
(gdb) si
[f000:e076] 0xfe076: jmp 0x5576cf2d
0x0000e076 in ?? ()
(gdb) si
[f000:cf2b] 0xfc2b: cli
0x0000cf2b in ?? ()
```

The BIOS when loaded jumps backwards from 0xf000:fff0 to 0xf000:0xe05b. It then set up an interrupt descriptor table and initializes various devices such as the VGA display.

### Exercise 3 -

readsect() assembly instructions:



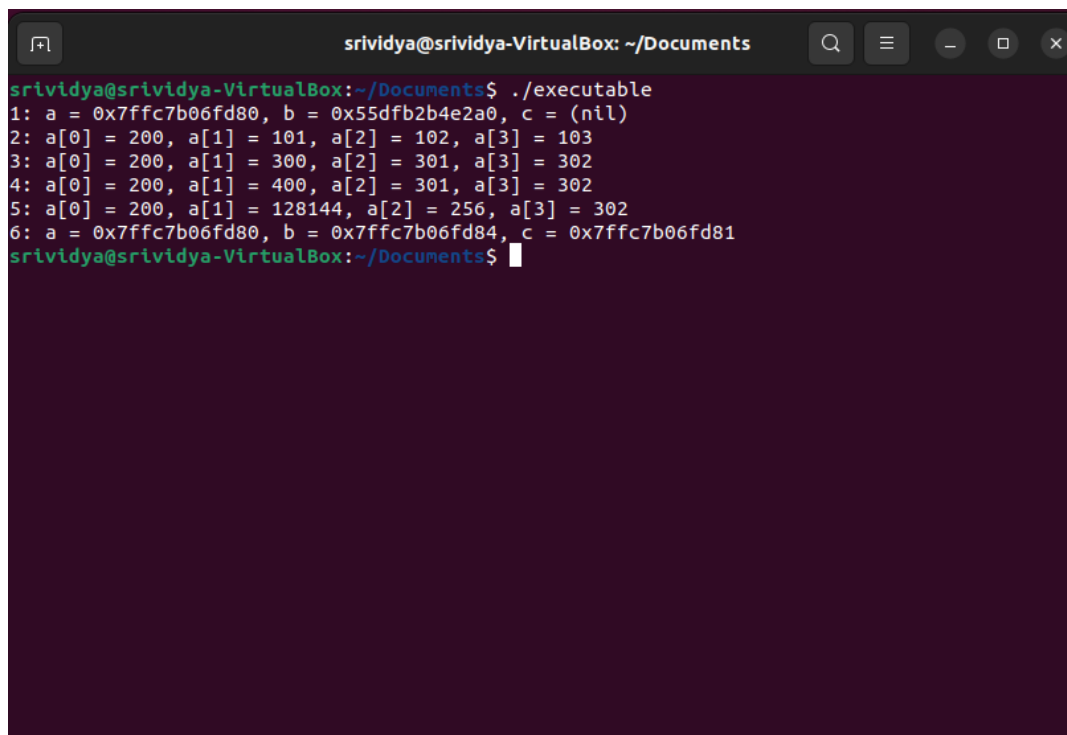
The image shows a code editor window titled 'bootblock.asm' with the following assembly instructions:

```
153 // Wait for disk ready.
154 while((inb(0x1F7) & 0xC0) != 0x40)
155     7c84:    83 e0 c0      and     $0xfffffc0,%eax
156     7c87:    3c 40        cmp     $0x40,%al
157     7c89:    75 f8        jne     7c83 <waitdisk+0x5>
158 ;
159 }
160     7c8b:    c3          ret
161
162 00007c8c <readsect>:
163
164 // Read a single sector at offset into dst.
165 void
166 readsect(void *dst, uint offset)
167 {
168     7c8c:    55          push    %ebp
169     7c8d:    89 e5      mov     %esp,%ebp
170     7c8f:    57          push    %edi
171     7c90:    53          push    %ebx
172     7c91:    8b 5d 0c    mov     0xc(%ebp),%ebx
173 // Issue command.
174 waitdisk();
175     7c94:    e8 e5 ff ff call     7c7e <waitdisk>
176 }
177
178 static inline void
179 outb(ushort port, uchar data)
180 {
181     asm volatile("out %0,%1" : "=a" (data), "=d" (port));
182 }
```

The GDB debugger window shows the following instructions:

```
=> 0x7d1e:    push    %esi
0x00007d1e  tn ?? ()
(gdb) si
=> 0x7d1f:    push    %ebx
0x00007d1f  tn ?? ()
(gdb) si
=> 0x7d20:    call    0x7c8c
0x00007d20  tn ?? ()
(gdb) si
=> 0x7c8c:    push    %ebp
0x00007c8c  tn ?? ()
(gdb) si
=> 0x7c8d:    mov     %esp,%ebp
0x00007c8d  tn ?? ()
(gdb) si
=> 0x7c8f:    push    %edi
0x00007c8f  tn ?? ()
(gdb) si
=> 0x7c90:    push    %ebx
0x00007c90  tn ?? ()
(gdb) si
=> 0x7c91:    mov     0xc(%ebp),%ebx
0x00007c91  tn ?? ()
(gdb) si
=> 0x7c94:    call    0x7c7e
0x00007c94  tn ?? ()
(gdb) si
=> 0x7c7e:    mov     $0x1f7,%edx
0x00007c7e  tn ?? ()
(gdb) si
=> 0x7c83:    in      (%dx),%al
```

### Exercise 4 -



The image shows a terminal window titled 'srividya@srividya-VirtualBox: ~/Documents' with the following output:

```
srividya@srividya-VirtualBox:~/Documents$ ./executable
1: a = 0x7ffc7b06fd80, b = 0x55dfb2b4e2a0, c = (nil)
2: a[0] = 200, a[1] = 101, a[2] = 102, a[3] = 103
3: a[0] = 200, a[1] = 300, a[2] = 301, a[3] = 302
4: a[0] = 200, a[1] = 400, a[2] = 301, a[3] = 302
5: a[0] = 200, a[1] = 128144, a[2] = 256, a[3] = 302
6: a = 0x7ffc7b06fd80, b = 0x7ffc7b06fd84, c = 0x7ffc7b06fd81
srividya@srividya-VirtualBox:~/Documents$
```

Line 1:

a, b, c are pointers to integer variables. a is allocated 16 bytes of memory on the stack. b is allocated 16 bytes of memory on the heap. Pointer c stores some junk pointer since it is declared but uninitialized.

Line 2:

In the line 15 **for loop** changes the value of integers in array to 100, 101, 102, 103. The line “c=a” makes the pointer c point to the same integer as a. So when c[0] is assigned 200 it changes the first element in array a because c is just another name for array a.

Line 3:

c[1]=300; - Changes a[1] to 300 as c is an alias for a.

\*(c+2)=301; - Another way of saying c[2]=301. a[2] is set to 301.

3[c]=302; -Another way of saying c[3]=302. a[3] is set to 302.

Line 4:

c=c+1; - This makes c point to the location of a[1]

\*c=400; - This changes a[1] to 400

Line 5:

c = (int \*) ((char \*) c + 1); - The hexadecimal value of the address stored in pointer c increases by only 1 since we typecast it to a character pointer before incrementing it. This is because the size of a character type data in C is 1 byte. The pointer c is then typecast back into an integer type. At the end of this c points to a segment of 4 bytes beginning from the second byte of a[1] and ending at the first byte of a[2].

The contents of a[2] and a[3] at this point look as follows.

a[2]=400

a[3]=301

10010000 10000000 00000000 00000000 00101101 10000000 00000000 00000000

After \*c=500 it changes to:

a[2]=128144

a[3]=256

10010000 11110100 00000001 00000000 00000000 10000000 00000000 00000000

Line 6:

b=(int\*)a+1; - Increases hexadecimal address value by 4.

c = (int \*) ((char \*) a + 1); - Increases hexadecimal address value by only 1

### Exercise 5 -

1- Changed link address from 0x7c00 to 0x7c10.

2- The make clean was then executed and the makefile was executed again.

3- The code stops working after the line at 0x7c2c and jumps to 0xfe05b. It gets into an infinite loop after execution of a few instructions.

(gdb)

[ 0:7c2c] => 0x7c2c:        ljmp    \$0xb866,\$0x87c41

0x00007c2c in ?? ()

(gdb)

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

0x0000e05b in ?? ()

### Exercise 6 -

The point BIOS enter bootloader

```
srividya@srividya-VirtualBox: ~/xv6-public
srividya@srividya-VirtualBox: ~/xv6-public
d by your 'auto-load safe-path' set to "$debugdir:$datadir/auto-load".
To enable execution of this file add
    add-auto-load-safe-path /home/srividya/xv6-public/.gdbinit
line to your configuration file "/home/srividya/.config/gdb/gdbinit".
To completely disable this security protection add
    set auto-load safe-path /
line to your configuration file "/home/srividya/.config/gdb/gdbinit".
For more information about this security protection see the
"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 set to "i8086".
[f000:fff0] 0xffff0: jmp $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) x/8x 0x00100000
0x100000: 0x00000000 0x00000000 0x00000000 0x00000000 0x00000000
0x100010: 0x00000000 0x00000000 0x00000000 0x00000000
(gdb)
```