Lab 1 – Team-6

```
anand@anand-Lenovo-G580:~$ cd Desktop/Anand/Fall-2016/Adv-OS/lab/
anand@anand-Lenovo-G580:~/Desktop/Anand/Fall-2016/Adv-OS/lab$ make qemu-gdb
***
*** Now run 'make gdb'.
***
qemu-system-i386 -hda obj/kern/kernel.img -serial mon:stdio -gdb tcp::26000 -D qemu.log  -S
WARNING: Image format was not specified for 'obj/kern/kernel.img' and p
Automatically detecting the format is dangerous for raw images
Specify the 'raw' format explicitly to remove the restrictions
```

QEMU [Paused]

Guest has not initialized the display (yet).

anand@anand-Lenovo-G580: ~/Desktop/Anand/Fall-2016/Adv-OS/lab

```
auto-load".
To enable execution of this file add
        add-auto-load-safe-path /home/anand/Desktop/Anand/Fall-2016/Adv-OS/lab/.
gdbinit
line to your configuration file "/home/anand/.gdbinit".
To completely disable this security protection add
        set auto-load safe-path /
line to your configuration file "/home/anand/.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:
---Type <return> to continue, or q <return> to quit---
        info "(gdb)Auto-loading safe path"
+ target remote localhost:26000
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.

The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp    $0xf000,$0xe05b
0x0000fff0 in ?? ()
+ symbol-file obj/kern/kernel
(gdb) b *0x7c00
Breakpoint 1 at 0x7c00
(gdb)
```

```
anand@anand-Lenovo-G580:~$ cd Desktop/Anand/Fall-2016/Adv-OS/lab/
anand@anand-Lenovo-G580:~/Desktop/Anand/Fall-2016/Adv-OS/lab$ make qemu-gdb
***
*** Now run 'make gdb'.
***
qemu-system-i386 -hda obj/kern/kernel.img -serial mon:stdio -gdb tcp::26000 -D qemu.log  -S
WARNING: Image format was not specified for 'obj/kern/kernel.img' and p
Automatically detecting the format is dangerous for raw images
Specify the 'raw' format explicitly to remove the restrictions
```

QEMU [Paused]

```
SeaBIOS (version rel-1.8.1-0-g4adadbd-20150316_085822-nilsson.home

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F93BE0+07EF3B

Booting from Hard Disk...
```

anand@anand-Lenovo-G580: ~/Desktop/Anand/Fall-2016/Adv-OS/lab

```
To completely disable this security protection add
        set auto-load safe-path /
line to your configuration file "/home/anand/.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:
---Type <return> to continue, or q <return> to quit---
        info "(gdb)Auto-loading safe path"
+ target remote localhost:26000
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.

The target architecture is assumed to be i8086
[f000:fff0]    0xffff0: ljmp    $0xf000,$0xe05b
0x0000fff0 in ?? ()
+ symbol-file obj/kern/kernel
(gdb) b *0x7c00
Breakpoint 1 at 0x7c00
(gdb) c
Continuing.
[   0:7c00] => 0x7c00:  cli

Breakpoint 1, 0x00007c00 in ?? ()
(gdb)
```
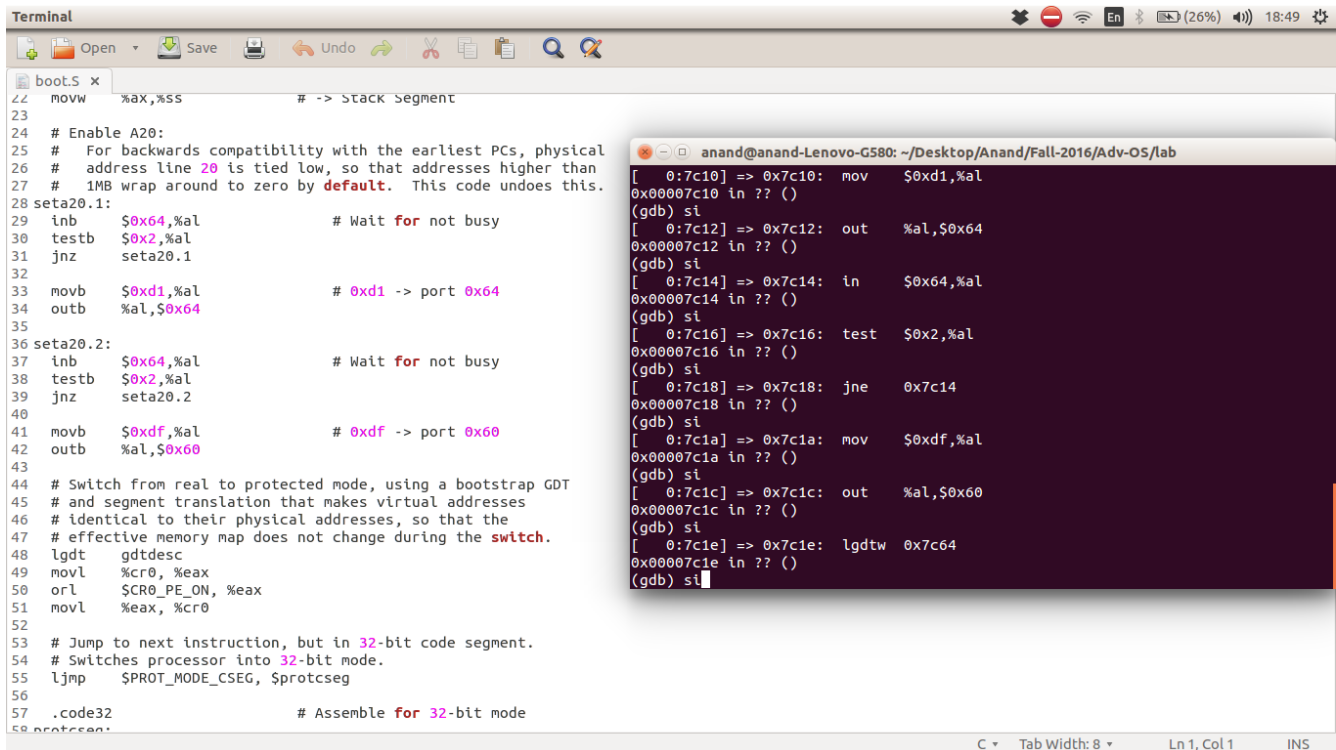
Exercise 3:   ljmp $PROT_MODE_CSEG, $protcseg

  orl $CR0_PE_ON, %eax
   7c26:   66 83 c8 01   or $0x1,%ax

  movl   %eax, %cr0
   7c2a: 0f 22 c0   mov   %eax,%cr0

```
 22   movw    %ax,%ss          # -> Stack Segment
 23
 24   # Enable A20:
 25   #   For backwards compatibility with the earliest PCs, physical
 26   #   address line 20 is tied low, so that addresses higher than
 27   #   1MB wrap around to zero by default.  This code undoes this.
 28 seta20.1:
 29   inb     $0x64,%al               # Wait for not busy
 30   testb   $0x2,%al
 31   jnz     seta20.1
 32
 33   movb    $0xd1,%al               # 0xd1 -> port 0x64
 34   outb    %al,$0x64
 35
 36 seta20.2:
 37   inb     $0x64,%al               # Wait for not busy
 38   testb   $0x2,%al
 39   jnz     seta20.2
 40
 41   movb    $0xdf,%al               # 0xdf -> port 0x60
 42   outb    %al,$0x60
 43
 44   # Switch from real to protected mode, using a bootstrap GDT
 45   # and segment translation that makes virtual addresses
 46   # identical to their physical addresses, so that the
 47   # effective memory map does not change during the switch.
 48   lgdt    gdtdesc
 49   movl    %cr0, %eax
 50   orl     $CR0_PE_ON, %eax
 51   movl    %eax, %cr0
 52
 53   # Jump to next instruction, but in 32-bit code segment.
 54   # Switches processor into 32-bit mode.
 55   ljmp    $PROT_MODE_CSEG, $protcseg
 56
 57   .code32                  # Assemble for 32-bit mode
 58 protcseg:
```

C ▾   Tab Width: 8 ▾   Ln 1, Col 1   INS

```
anand@anand-Lenovo-G580: ~/Desktop/Anand/Fall-2016/Adv-OS/lab
0x00007c1a in ?? ()
(gdb) si
[   0:7c1c] => 0x7c1c:  out    %al,$0x60
0x00007c1c in ?? ()
(gdb) si
[   0:7c1e] => 0x7c1e:  lgdtw  0x7c64
0x00007c1e in ?? ()
(gdb) si
[   0:7c23] => 0x7c23:  mov    %cr0,%eax
0x00007c23 in ?? ()
(gdb) si
[   0:7c26] => 0x7c26:  or     $0x1,%eax
0x00007c26 in ?? ()
(gdb) si
[   0:7c2a] => 0x7c2a:  mov    %eax,%cr0
0x00007c2a in ?? ()
(gdb) si
[   0:7c2d] => 0x7c2d:  ljmp   $0x8,$0x7c32
0x00007c2d in ?? ()
(gdb) si
The target architecture is assumed to be i386
=> 0x7c32:      mov    $0x10,%ax
0x00007c32 in ?? ()
(gdb)
```

((void (*)(void)) (ELFHDR->e_entry))();
f010000c: 66 c7 05 72 04 00 00   movw $0x1234,0x472

Open ▾ | Save | | Undo | | | | |

boot.S ✕ | main.c ✕

```c
24  *    hard-drive, this code takes over...
25  *
26  * * control starts in boot.S -- which sets up protected mode,
27  *   and a stack so C code then run, then calls bootmain()
28  *
29  * * bootmain() in this file takes over, reads in the kernel and jumps to it.
30  ********************************************
31
32 #define SECTSIZE        512
33 #define ELFHDR          ((struct Elf *) 0x10000) // scratch spa
34
35 void readsect(void*, uint32_t);
36 void readseg(uint32_t, uint32_t, uint32_t);
37
38 void
39 bootmain(void)
40 {
41         struct Proghdr *ph, *eph;
42
43         // read 1st page off disk
44         readseg((uint32_t) ELFHDR, SECTSIZE*8, 0);
45
46         // is this a valid ELF?
47         if (ELFHDR->e_magic != ELF_MAGIC)
48                 goto bad;
49
50         // load each program segment (ignores ph flags)
51         ph = (struct Proghdr *) ((uint8_t *) ELFHDR + ELFHDR->e
52         eph = ph + ELFHDR->e_phnum;
53         for (; ph < eph; ph++)
54                 // p_pa is the load address of this segment (as
55                 // as the physical address)
56                 readseg(ph->p_pa, ph->p_memsz, ph->p_offset);
57
58         // call the entry point from the ELF header
59         // note: does not return!
```

```
● — □  anand@anand-Lenovo-G580: ~/Desktop/Anand/Fall-2016/Adv-OS/lab
=> 0x7c32:       mov    $0x10,%ax
0x00007c32 in ?? ()
(gdb) si
=> 0x7c36:       mov    %eax,%ds
0x00007c36 in ?? ()
(gdb) si
=> 0x7c38:       mov    %eax,%es
0x00007c38 in ?? ()
(gdb) si
=> 0x7c3a:       mov    %eax,%fs
0x00007c3a in ?? ()
(gdb) si
=> 0x7c3c:       mov    %eax,%gs
0x00007c3c in ?? ()
(gdb) si
=> 0x7c3e:       mov    %eax,%ss
0x00007c3e in ?? ()
(gdb) si
=> 0x7c40:       mov    $0x7c00,%esp
0x00007c40 in ?? ()
(gdb) si
=> 0x7c45:       call   0x7d0a
0x00007c45 in ?? ()
(gdb)
```

C ▾   Tab Width: 8 ▾   Ln 1, Col 1   INS

Open ▾ | Save | | Undo | | | | |

boot.S ✕ | main.c ✕

```c
24  *    hard-drive, this code takes over...
25  *
26  * * control starts in boot.S -- which sets up protected mode,
27  *   and a stack so C code then run, then calls bootmain()
28  *
29  * * bootmain() in this file takes over, reads in the kernel and jumps to it.
30  ********************************************
31
32 #define SECTSIZE        512
33 #define ELFHDR          ((struct Elf *) 0x10000) // scratch spa
34
35 void readsect(void*, uint32_t);
36 void readseg(uint32_t, uint32_t, uint32_t);
37
38 void
39 bootmain(void)
40 {
41         struct Proghdr *ph, *eph;
42
43         // read 1st page off disk
44         readseg((uint32_t) ELFHDR, SECTSIZE*8, 0);
45
46         // is this a valid ELF?
47         if (ELFHDR->e_magic != ELF_MAGIC)
48                 goto bad;
49
50         // load each program segment (ignores ph flags)
51         ph = (struct Proghdr *) ((uint8_t *) ELFHDR + ELFHDR->e
52         eph = ph + ELFHDR->e_phnum;
53         for (; ph < eph; ph++)
54                 // p_pa is the load address of this segment (as
55                 // as the physical address)
56                 readseg(ph->p_pa, ph->p_memsz, ph->p_offset);
57
58         // call the entry point from the ELF header
59         // note: does not return!
```

```
● — □  anand@anand-Lenovo-G580: ~/Desktop/Anand/Fall-2016/Adv-OS/lab
   0x7d4b:       pushl  0x4(%ebx)
   0x7d4e:       add    $0x20,%ebx
(gdb) x/20i
   0x7d51:       pushl  -0xc(%ebx)
   0x7d54:       pushl  -0x14(%ebx)
   0x7d57:       call   0x7cd1
   0x7d5c:       add    $0xc,%esp
   0x7d5f:       jmp    0x7d47
   0x7d61:       call   *0x10018
   0x7d67:       mov    $0x8a00,%edx
   0x7d6c:       mov    $0xffff8a00,%eax
   0x7d71:       out    %ax,(%dx)
   0x7d73:       mov    $0xffff8e00,%eax
   0x7d78:       out    %ax,(%dx)
   0x7d7a:       jmp    0x7d7a
   0x7d7c:       add    %al,(%eax)
   0x7d7e:       add    %al,(%eax)
   0x7d80:       add    %al,(%eax)
   0x7d82:       add    %al,(%eax)
   0x7d84:       add    %al,(%eax)
   0x7d86:       add    %al,(%eax)
   0x7d88:       add    %al,(%eax)
   0x7d8a:       add    %al,(%eax)
(gdb) x/20i
```

C ▾   Tab Width: 8 ▾   Ln 1, Col 1   INS

```
ph = (struct Proghdr *) ((uint8_t *) ELFHDR + ELFHDR->e_phoff);
     eph = ph + ELFHDR->e_phnum;
     for (; ph < eph; ph++)
  7d4e:    83 c3 20              add    $0x20,%ebx
```

```
365     7d4b:       ff 73 04            pushl   0x4(%ebx)
366             goto bad;
367
368             // load each program segment (ignores ph flags)
369             ph = (struct Proghdr *) ((uint8_t *) ELFHDR + ELFHDR->e_phoff);
370             eph = ph + ELFHDR->e_phnum;
371             for (; ph < eph; ph++)
372     7d4e:       83 c3 20            add     $0x20,%ebx
373             // p_pa is the load address of this segment (as well
374             // as the physical address)
375             readseg(ph->p_pa, ph->p_memsz, ph->p_offset);
376     7d51:       ff 73 f4            pushl   -0xc(%ebx)
377     7d54:       ff 73 ec            pushl   -0x14(%ebx)
378     7d57:       e8 75 ff ff ff      call    7cd1 <readseg>
379             goto bad;
380
381             // load each program segment (ignores ph flags)
382             ph = (struct Proghdr *) ((uint8_t *) ELFHDR + ELFHDR->e_phoff);
383             eph = ph + ELFHDR->e_phnum;
384             for (; ph < eph; ph++)
385     7d5c:       83 c4 0c            add     $0xc,%esp
386     7d5f:       eb e6               jmp     7d47 <bootmain+0x3d>
387             // as the physical address)
388             readseg(ph->p_pa, ph->p_memsz, ph->p_offset);
389
390             // call the entry point from the ELF header
391             // note: does not return!
392             ((void (*)(void)) (ELFHDR->e_entry))();
393     7d61:       ff 15 18 00 01 00   call    *0x10018
394 }
395
396 static __inline void
397 outw(int port, uint16_t data)
398 {
399             __asm __volatile("outw %0,%w1" : : "a" (data), "d" (port));
400     7d67:       ba 00 8a 00 00      mov     $0x8a00,%edx
401     7d6c:       ...
```

LaTeX ▾   Tab Width: 8 ▾   Ln 389, Col 1   INS

```
370             eph = ph + ELFHDR->e_phnum;
371             for (; ph < eph; ph++)
372     7d4e:       83 c3 20            add     $0x20,%ebx
373             // p_pa is the load address of this segment (as well
374             // as the physical address)
375             readseg(ph->p_pa, ph->p_memsz, ph->p_offset);
376     7d51:       ff 73 f4            pushl   -0xc(%ebx)
377     7d54:       ff 73 ec            pushl   -0x14(%ebx)
378     7d57:       e8 75 ff ff ff      call    7cd1 <readseg>
379             goto bad;
380
381             // load each program segment (ignores ph flags)
382             ph = (struct Proghdr *) ((uint8_t *) ELFHDR + ELFHDR->e_phoff);
383             eph = ph + ELFHDR->e_phnum;
384             for (; ph < eph; ph++)
385     7d5c:       83 c4 0c            add     $0xc,%esp
386     7d5f:       eb e6               jmp     7d47 <bootmain+0x3d>
387             // as the physical address)
388             readseg(ph->p_pa, ph->p_memsz, ph->p_offset);
389
390             // call the entry point from the ELF header
391             // note: does not return!
392             ((void (*)(void)) (ELFHDR->e_entry))();
393     7d61:       ff 15 18 00 01 00   call    *0x10018
394 }
395
396 static __inline void
397 outw(int port, uint16_t data)
398 {
399             __asm __volatile("outw %0,%w1" : : "a" (data), "d" (port));
400     7d67:       ba 00 8a 00 00      mov     $0x8a00,%edx
401     7d6c:       b8 00 8a ff ff      mov     $0xffff8a00,%eax
402     7d71:       66 ef               out     %ax,(%dx)
403     7d73:       b8 00 8e ff ff      mov     $0xffff8e00,%eax
404     7d78:       66 ef               out     %ax,(%dx)
405     7d7a:       eb fe               jmp     7d7a <bootmain+0x70>
```

LaTeX ▾   Tab Width: 8 ▾   Ln 393, Col 56   INS

Terminal ✳ ⛔ 🛜 En ✻ 🔋(31%) 🔊 19:12 ⚙

```
anand@anand-Lenovo-G580:~$ cd Desktop/Anand/Fall-2016/Adv-OS/lab/
anand@anand-Lenovo-G580:~/Desktop/Anand/Fall-2016/Adv-OS/lab$ make qemu-gdb
***
*** Now run 'make gdb'.
***
qemu-system-i386 -hda obj/kern/kernel.img -serial mon:stdio -gdb tcp::26000 -D qemu.log  -S
WARNING: Image format was not specified for 'obj/kern/kernel.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
6828 decimal is XXX octal!
entering test_backtrace 5
entering test_backtrace 4
entering test_backtrace 3
entering test_backtrace 2
entering test_backtrace 1
entering test_backtrace 0
leaving test_backtrace 0
leaving test_backtrace 1
leaving test_backtrace 2
leaving test_backtrace 3
leaving test_backtrace 4
leaving test_backtrace 5
Welcome to the JOS kernel monitor!
Type 'help' for a list of commands.
K> anand@anand-Lenovo-G580:~/Desktop/Anand/Fall-2016/Adv-OS/lab$ mal
***
*** Now run 'make gdb'.
***
qemu-system-i386 -hda obj/kern/kernel.img -serial mon:stdio -gdb tc
WARNING: Image format was not specified for 'obj/kern/kernel.img' a
         Automatically detecting the format is dangerous for raw im
         Specify the 'raw' format explicitly to remove the restrict
```

anand@anand-Lenovo-G580: ~/Desktop/Anand/Fall-2016/Adv-OS/lab

```
    0x7d9a:      add     %al,(%eax)
    0x7d9c:      add     %al,(%eax)
    0x7d9e:      add     %al,(%eax)
    0x7da0:      add     %al,(%eax)
    0x7da2:      add     %al,(%eax)
    0x7da4:      add     %al,(%eax)
    0x7da6:      add     %al,(%eax)
    0x7da8:      add     %al,(%eax)
    0x7daa:      add     %al,(%eax)
    0x7dac:      add     %al,(%eax)
    0x7dae:      add     %al,(%eax)
    0x7db0:      add     %al,(%eax)
    0x7db2:      add     %al,(%eax)
(gdb) b *0x0010000c
Breakpoint 2 at 0x10000c
(gdb) b *0x0010000c
Note: breakpoint 2 also set at pc 0x10000c.
Breakpoint 3 at 0x10000c
(gdb) c
Continuing.
=> 0x10000c:       movw     $0x1234,0x472

Breakpoint 2, 0x0010000c in ?? ()
(gdb)
```

Terminal ✳ ⛔ 🛜 En ✻ 🔋(32%) 🔊 19:15 ⚙

```
anand@anand-Lenovo-G580:~$ cd Desktop/Anand/Fall-2016/Adv-OS/lab/
anand@anand-Lenovo-G580:~/Desktop/Anand/Fall-2016/Adv-OS/lab$ make qemu-gdb
***
*** Now run 'make gdb'.
***
qemu-system-i386 -hda obj/kern/kernel.img -serial mon:stdio -gdb tcp::26000 -D qemu.log  -S
WARNING: Image format was not specified for 'obj/kern/kernel.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
6828 decimal is XXX octal!
entering test_backtrace 5
entering test_backtrace 4
entering test_backtrace 3
entering test_backtrace 2
entering test_backtrace 1
entering test_backtrace 0
leaving test_backtrace 0
leaving test_backtrace 1
leaving test_backtrace 2
leaving test_backtrace 3
leaving test_backtrace 4
leaving test_backtrace 5
Welcome to the JOS kernel monitor!
Type 'help' for a list of commands.
K> anand@anand-Lenovo-G580:~/Desktop/Anand/Fall-2016/Adv-OS/lab$ mal
***
*** Now run 'make gdb'.
***
qemu-system-i386 -hda obj/kern/kernel.img -serial mon:stdio -gdb tc
WARNING: Image format was not specified for 'obj/kern/kernel.img' a
         Automatically detecting the format is dangerous for raw im
         Specify the 'raw' format explicitly to remove the restrict
```

anand@anand-Lenovo-G580: ~/Desktop/Anand/Fall-2016/Adv-OS/lab

```
    0x7d9a:      add     %al,(%eax)
    0x7d9c:      add     %al,(%eax)
    0x7d9e:      add     %al,(%eax)
    0x7da0:      add     %al,(%eax)
    0x7da2:      add     %al,(%eax)
    0x7da4:      add     %al,(%eax)
    0x7da6:      add     %al,(%eax)
    0x7da8:      add     %al,(%eax)
    0x7daa:      add     %al,(%eax)
    0x7dac:      add     %al,(%eax)
    0x7dae:      add     %al,(%eax)
    0x7db0:      add     %al,(%eax)
    0x7db2:      add     %al,(%eax)
(gdb) b *0x0010000c
Breakpoint 2 at 0x10000c
(gdb) b *0x0010000c
Note: breakpoint 2 also set at pc 0x10000c.
Breakpoint 3 at 0x10000c
(gdb) c
Continuing.
=> 0x10000c:       movw     $0x1234,0x472

Breakpoint 2, 0x0010000c in ?? ()
(gdb)
```

```c
45
46          // is this a valid ELF?
47          if (ELFHDR->e_magic != ELF_MAGIC)
48                  goto bad;
49
50          // load each program segment (ignores ph flags)
51          ph = (struct Proghdr *) ((uint8_t *) ELFHDR + ELFHDR->e_phoff);
52          eph = ph + ELFHDR->e_phnum;
53          for (; ph < eph; ph++)
54                  // p_pa is the load address of this segment (as well
55                  // as the physical address)
56                  readseg(ph->p_pa, ph->p_memsz, ph->p_offset);
57
58          // call the entry point from the ELF header
59          // note: does not return!
60          ((void (*)(void)) (ELFHDR->e_entry))();
61
62 bad:
63          outw(0x8A00, 0x8A00);
64          outw(0x8A00, 0x8E00);
65          while (1)
66                  /* do nothing */;
67 }
68
69 // Read 'count' bytes at 'offset' from kernel into physical address 'pa'.
70 // Might copy more than asked
71 void
72 readseg(uint32_t pa, uint32_t count, uint32_t offset)
73 {
74          uint32_t end_pa;
75
76          end_pa = pa + count;
77
78          // round down to sector boundary
79          pa &= ~(SECTSIZE - 1);
80
```

C ▾   Tab Width: 8 ▾   Ln 56, Col 62   INS

```
364          readseg(ph->p_pa, ph->p_memsz, ph->p_offset);
365  7d4b:       ff 73 04              pushl  0x4(%ebx)
366          goto bad;
367
368          // load each program segment (ignores ph flags)
369          ph = (struct Proghdr *) ((uint8_t *) ELFHDR + ELFHDR->e_phoff);
370          eph = ph + ELFHDR->e_phnum;
371          for (; ph < eph; ph++)
372  7d4e:       83 c3 20              add    $0x20,%ebx
373                  // p_pa is the load address of this segment (as well
374                  // as the physical address)
375                  readseg(ph->p_pa, ph->p_memsz, ph->p_offset);
376  7d51:       ff 73 f4              pushl  -0xc(%ebx)
377  7d54:       ff 73 ec              pushl  -0x14(%ebx)
378  7d57:       e8 75 ff ff ff        call   7cd1 <readseg>
379          goto bad;
380
381          // load each program segment (ignores ph flags)
382          ph = (struct Proghdr *) ((uint8_t *) ELFHDR + ELFHDR->e_phoff);
383          eph = ph + ELFHDR->e_phnum;
384          for (; ph < eph; ph++)
385  7d5c:       83 c4 0c              add    $0xc,%esp
386  7d5f:       eb e6                 jmp    7d47 <bootmain+0x3d>
387                  // as the physical address)
388                  readseg(ph->p_pa, ph->p_memsz, ph->p_offset);
389
390          // call the entry point from the ELF header
391          // note: does not return!
392          ((void (*)(void)) (ELFHDR->e_entry))();
393  7d61:       ff 15 18 00 01 00     call   *0x10018
394 }
395
396 static __inline void
397 outw(int port, uint16_t data)
398 {
399          __asm __volatile("outw %0,%w1" : : "a" (data), "d" (port));
400  7d67:       ba 00 8a 00 00        mov    $0x8a00,%edx
```

LaTeX ▾   Tab Width: 8 ▾   Ln 388, Col 62   INS

Exercise 6:
Bios enters the bootloader : Breakpoint at 0x7c00;
Since the kernel is yet to load the locations specified will have no value.
Bootloader enters the kernel : Breakpoint at 0x0010000c
 At this address kernel is loaded hence the change in values.

Exercise 7: At 0x00100000 is the physical space of the kernel and at 0xf0100000 holds no value since paging in not enabled. After the next instruction paging is enables and at 0xf0100000 has the same memory as 0x00100000 as the kernel requires its programs to be at the top of the virtual memory, which provides the remaining space for the user programs.

After call of the relocation function it determines that there is no paging enabled,

so the kernel displays an error. Jmp %eax fails.

Exercise : 9   # Set the stack pointer
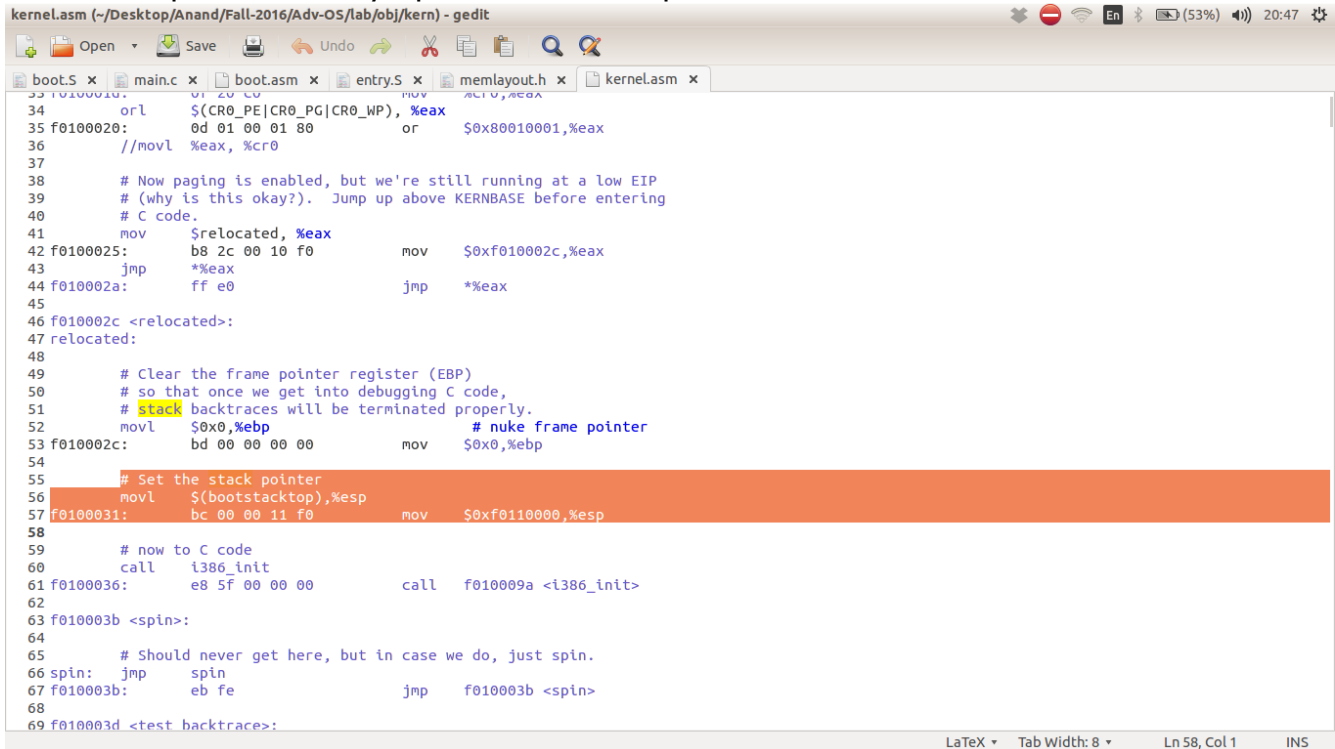   movl $(bootstacktop),%esp
f0100031: bc 00 00 11 f0   mov $0xf0110000,%esp
The kernel reserves the stack space by KSTKSIZE=8*PGSIZE.(8*4096 = 32mb)
0xf0000000- ebp ; esp-0xf0110000.
The stack pointer always points to the top of the stack towards bootstack.


```
kernel.asm (~/Desktop/Anand/Fall-2016/Adv-OS/lab/obj/kern) - gedit

boot.S  x    main.c  x    boot.asm  x    entry.S  x    memlayout.h  x    kernel.asm  x

33 f0100010:                  or  20 co           mov    %cr0,%eax
34          orl      $(CR0_PE|CR0_PG|CR0_WP), %eax
35 f0100020:      0d 01 00 01 80           or     $0x80010001,%eax
36          //movl  %eax, %cr0
37
38          # Now paging is enabled, but we're still running at a low EIP
39          # (why is this okay?).  Jump up above KERNBASE before entering
40          # C code.
41          mov      $relocated, %eax
42 f0100025:      b8 2c 00 10 f0           mov    $0xf010002c,%eax
43          jmp      *%eax
44 f010002a:      ff e0                    jmp    *%eax
45
46 f010002c <relocated>:
47 relocated:
48
49          # Clear the frame pointer register (EBP)
50          # so that once we get into debugging C code,
51          # stack backtraces will be terminated properly.
52          movl     $0x0,%ebp                # nuke frame pointer
53 f010002c:      bd 00 00 00 00           mov    $0x0,%ebp
54
55          # Set the stack pointer
56          movl    $(bootstacktop),%esp
57 f0100031:        bc 00 00 11 f0           mov    $0xf0110000,%esp
58
59          # now to C code
60          call    i386_init
61 f0100036:        e8 5f 00 00 00           call   f010009a <i386_init>
62
63 f010003b <spin>:
64
65          # Should never get here, but in case we do, just spin.
66 spin:    jmp      spin
67 f010003b:      eb fe                    jmp    f010003b <spin>
68
69 f010003d <test_backtrace>:
```

LaTeX ▾   Tab Width: 8 ▾   Ln 58, Col 1   INS