Reading. Read the following sections in the textbook.

- Ch 3: 3.1–3.4 up to and including 3.4.2

Homework Questions. Do the following problems and turn in your answers. For each part, turn in your source code and a screenshot of **gdb** positioned on the first line (instruction) of the "exit" part your program. Your screenshot should include the registers and the source code layouts. Note that to compile your assembly programs do the following steps. If your program is saved in a file called `prog.s`, from the command line run the following (where `$` denotes the command prompt) to compile your program.

```
$ gcc -c -g prog.s
$ ld prog.o -o prog
```

The result will be an executable file named `prog`. You can then run **gdb** using the command:

```
$ gdb prog
```

Once in **gdb**, create a breakpoint at the **_start** label:

```
(gdb) break _start
```

Then run the program:

```
(gdb) run
```

This will cause **gdb** to pause at the breakpoint. Open the register and source viewer:

```
(gdb) layout reg
```

Then step through the program:

```
(gdb) step
```

See the lecture notes for more information and examples.

1. Type the following program into a file named `p1.s`, compile it, and step through it using **gdb**. Explain what happens to the `%rax`, `%rbx`, and `%rip` registers as the "main" part of the program is executed (before the exit code is called)

```
        .global _start
        .text
_start:
        movb    $53, %al      # put value into low ax byte
        movb    $37, %bh      # put value into high bh byte
        # exit program
        movq    $60, %rax     # sys call 60 is exit
        xor     %rdi, %rdi    # return 0
        syscall
```

2. Type the following program into a file named `p2.s`, compile it, and step through it using `gdb`. Explain what happens to the `%rax` register *and why* as the "main" part of the program is executed (before the exit code is called).

```
        .global _start
        .text
_start:
        movw    $65535, %ax     # put value into ax
        movb    $0x01, %al      # put value into low ax byte
        # exit program
        movq    $60, %rax       # sys call 60 is exit
        xor     %rdi, %rdi      # return 0
        syscall
```

3. Write a program similar to the one above but that goes "the other way", i.e., stores a smaller value into a register and then stores a larger value into it. Explain what happens to the register and why.

4. Type the following program into a file named `p4.s`, compile it, and step through it using `gdb`. Explain what happens to the `%rax` and `eflag` registers *and why* as the "main" part of the program is executed (before the exit code is called).

```
        .global _start
        .text
_start:
        movw    $17, %ax      # put value into ax
        addw    $-17, %ax     # ax = -17 + ax
        movw    $23, %bx      # put value into bx
        subw    $23, %bx      # bx = bx - 23
        # exit program
        movq    $60, %rax     # sys call 60 is exit
        xor     %rdi, %rdi    # return 0
        syscall
```

5. Type the following program into a file named `p5.s`, compile it, and step through it using `gdb`. Explain what happens to the `%rax`, `%rbx`, and `eflag` registers *and why* as the "main" part of the program is executed (before the exit code is called).

```
        .global _start
        .text
_start:
        movw    $65535, %ax     # put value into ax
        addw    $2, %ax         # ax = 2 + ax
        movw    %ax, %bx        # move ax value to bx
        addw    $-2, %bx        # bx = bx - 2
        # exit program
        movq    $60, %rax       # sys call 60 is exit
        xor     %rdi, %rdi      # return 0
        syscall
```