ES2 EE3400.

The exercises in this set provide experience with assembly programming in ARM and related information.

You can use the cpu emulator https://cpulator.01xz.net/?sys=arm for executing code described below. It is based on the ARM-Av7 core. Its operation is described here,

1. How to store data in memory. The website above has total program memory starting from #0000 0000 and going till #FFFF FFFF. See the tab named "Memory". When you write a program in assembly and compile it, the program's opcode will be stored in memory starting from #0000 0000.
For example, write this code in the "Editor" tab:

```
.global _start
_start:
    mov r0, #4
```

Press the "Compile and Load" button and it gives you a disassembly in the other tab. Now examine "Address" and "Opcode" columns in disassembly tab, and look at the Memory tab, at the memory location of #0000 0000. (you may need to scroll, since the memory tab wraps the display after #ffff ffff is reached.
"Mov" instruction is used to move values from one register to another.
.global statements can be considered as declaration of global variables like functions and some data.

Your programs will normally take very small space in memory, the rest of it is available for data.
To store a value of "4" in memory:

```
.global _start
.equ a, 4
.equ b, 0x100
_start:
    ldr r0, =a
   add r0, r0, r0
    ldr r1, =b
    str r1, [r1]
```

Press the "Compile and Load" button and look at the disassembly and memory contents.
- Ldr and str are load and store register instructions, which we will cover in next class.
- .equ allocates space in memory for storage of that variable. =a is substituted by the address of the variable.

Above program stores value "4" in register r0, doubles its value and then stores the value 0x100 (256) in register r1. Then it stores the value in register r1 (the second argument) to the location pointed by the register r1. You can examine the memory location 0x100 after executing this program.

2. Evaluation of conditions is done by instruction called “cmp”. The results are stored in NZCV bits of the APSR (application program status register).

```
.global _start
.equ a, 4
.equ b, 0x100
_start:
  ldr r0, =a
  add r0, r0, r0
  ldr r1, =b
  cmp r1, r0
  strle r1, [r1]
```

Here we are using “conditional execution” of “str” which is the general form of the branch execution used to make conditional statements in a higher level language. A branched version of the above program is (here the reversed branch condition is used to skip the “str” instruction.

```
.global _start
.equ a, 4
.equ b, 0x100
_start:
  ldr r0, =a
  add r0, r0, r0
  ldr r1, =b
  cmp r1, r0
  bge skip
  Str r1,[r1]
skip:
```

Run the above two code fragments and verify if they execute according to your expectations. Also note the value of the PC (program counter) as the execution proceeds. What happens to PC near BGE statement?

3. Find the smallest of three given numbers. Store these numbers at location 0x1000 using the method shown above.

1	0x102C 0x7040 0x2344

Then the output should be in register r0: 0x102C.

4. You can use the branching instructions to make loops. Find the sum of squares for a series of integers. Their total number n and values are located at a given memory location, e.g.
n=5 and numbers = 0x1 0x2 0x3 0x4 0x5
Write C implementation of the same first.

5. Write a program (in python or C) to generate opcodes corresponding to all the variants of the following instructions:
A. add
B. ldr
C. b
By variants we mean: add r0, r1, r2; add r0, r1, r2, lsl #2 etc.

6. Run the following code which implements a function and uses the stack to store the link register. The program has two corrections, which you have to find out. You can also use sp, lr and pc instead of the register numbers (r13, r14 and r15 respectively).

```
.global _start
_start:
Main:
        mov sp, #1000
        stmfd r13!, {lr}
        mov r0, #1
        mov r1, #3
        mov r2, #4
        bl do_something
        mov r1 , r0
        mov r0 , #0
        ldmfd r13!, {pc}

do_something:
        stmfd r13!, {r4, lr}
        add r4, r0, r1
        mov r0 , r2
        add r0 , r4 , r0
        ldmfd r13!, {r4 , pc}
```

Also find out what this function is doing.

Above The function do_something saves data in r4 on stack since it is using that register for calculations. By convention the return values are left in register r0, which both "main" and "do_something" implement. Arguments 1 and 2 for do_something are passed to it in registers r0 and r1.