National University of Computer and Emerging Sciences, Lahore Campus

Course:	**COAL Lab**	**Code:**	**EL213**
Program:	**BS (Computer Science)**	**Semester:**	**Fall 2018**
Duration:	**150 minutes**	**T. Marks:**	**100**
Date:	**Thursday 13-Dec-2018**	**Weight**	**40**
Section:	**All**	**Page(s):**	**2**
Exam:	**Lab Final**		

Instructions/Notes:

- Use of the internet, notes, codes, lab manuals, and flash drives is strictly prohibited.
- You are only allowed to use the soft copy of book.
- Plagiarism will result in **F** grade in lab.
- Submission path: Section-X (here X will be your section A or B or C)
  \\sandata\xeon\Fall 2018\Shakeel Zafar\Final COAL\Section-X\Q1 or Q2
- Code must be **indented properly**, failure to comply will incur a penalty.

Question # 1: 50 marks

(a) Almost all operating systems, offer the utility of *scheduled tasks*. With this utility any task can be run at a scheduled time. You can think of it as an alarm clock - where the alarm sounds at a fixed time - or the countdown timer - where a reminder or a note is displayed when a defined amount of time has elapsed.
We want to add the capability of *scheduled tasks* in the multitasking kernel of example 11.2. You can use the *rotating-bar-task* from book, example 11.1.

Your kernel has to multi-task only 3 rotating bar tasks: **task one** – displays a rotating bar at [es:0]; **task two** – displays a rotating bar at [es:158]; and **task three** – a *scheduled task* which displays a rotating bar at [es:200]. Here es points to the video segment.
Let's say task three is to be scheduled by the kernel after every 5 seconds. Normally, the kernel would multi-task between task zero, task one and task two. But when a period of 5 seconds would have elapsed, the kernel would run task three.
For task three your initpcb would take an extra parameter through the stack– the number of ticks (stored in a label "sched_time") after which to run the task.

(b) The thing with *scheduled tasks* is that they run only once when the time comes. Modify your code so that only a single iteration of the *scheduled task* – task three – is executed every time it runs.
Note: Do not use the kernel from 11.1

Question # 2: 50 marks

In some science fiction stories and films, when the computers encounter an illogical situation, their response to the situation is displayed as a "NOT OKAY" message. Given the problem below, write an assembly language program which can simulate this behavior whenever it sees arithmetic statements that are incorrect.

Input
Program will take as input an arithmetic equation from the user via keyboard.
That equation will be in format: "A Operator B = C", where A is a numeric value (0<=A<=99), B is a numeric value (0<=B<=99) and C is also a numeric value (0<=C<=99). And "Operator" can be "+", "-" or "*". A, B, C and operators are single space-separated. You have to store the complete equation in memory.

Output:
After taking the equation as input, you have to process it. Processing means, determining whether the equation is correct or incorrect. Correct equation is the one where left-hand side is equal to right-hand side. Print the equation and the message "OKAY" (in case the equation is correct) and "NOT OKAY" (if it is incorrect) on video memory.

For this you can divide the task into sub-tasks (routines). So, you must write these routines:

1. **inputRoutine** - This routine takes no parameter. It uses **'int 16'** (software interrupt) to **take input of equation** from **user** and stores the equation in some memory array label. The routine **returns** the address of this label via **stack**.
2. **formNumber**: This routine takes one parameter that is **address** of label in memory where array of digits is placed. The routine reads all the digits placed on that label, until it encounters a space. It combines the digits to form a number and returns that number via stack.
3. **tokenizerRoutine**: This routine takes one parameter, which is address of a label in memory, where the complete equation is placed. Using the routine in (2) above, this tokenizer routine checks whether the left-hand side of equation is equal to right hand side and displays "OKAY" if the equation is correct and "NOT OKAY" if it is not.

Sample I/O

Correct Cases:	**Incorrect Cases:**
13 + 15 = 28 OKAY 12 - 11 = 1 OKAY 4 * 18 = 72 OKAY	65 - 65 = 1 NOT OKAY 1 + 90 = 92 NOT OKAY 2 * 4 = 19 NOT OKAY