In this project you will work in a group of 3 to 4 to create a software real time clock (RTC or simply refer to as “clock” in this document unless specified otherwise) with alarming capability using MSP430G2 Launchpad and assembly language programming. Note that some microcontrollers have hardware real time clocks as a peripheral you can use, our G2553 does not. This project is divided into three standalone parts. Each member of your group should take up one part of the project and your group will work together to combine them in the end.

Three parts of the program are described in its own sections in this document. The three parts are:
- Subroutine: modular arithmetic
- Subroutine: find time difference
- Main program and interrupts

At the very beginning of your program you should write down in a comment section who has written which part of this project.

In this project description, graded tasks that you must accomplish are highlighted in blue. All variable names are shown in bold italic to avoid confusions.
Subroutine: Modular Arithmetic (Student A)

You will be writing a subroutine called mod, which finds the integer part and the remainder of an integer division. Another student will use your subroutine to handle clock overflows.

The contract of the subroutine mod is as described below. You should follow this contract strictly in your program. Otherwise points will be taken off. Copy and paste the contract into your code right before your mod subroutine.

; -----------------------------------------------------------------------------------------
; ; Description:  
; ; Find the integer part and the remainder of a division of integer X divide by integer Y
; ; Prerequisite (before function call)
; ;    push dividend X onto stack
; ;    push divisor Y onto stack
; ;    Reserve two words on the stack for outputs to return
; ;    In total 4 words are reserved on the stack
; ; Input:
; ;    dividend X - 8(SP)
; ;    divisor Y - 6(SP)
; ; Output
; ;    Remainder of X/Y - 2(SP)
; ;    floor(X/Y) - 4(SP)
; ; Post-requisite (after function call)
; ;    store outputs
; ;    clear reserved stack (4 words in total)
; ; -----------------------------------------------------------------------------------------

Multiplication and division are not basic operations in assembly. The MSP430G2553 does not have a peripheral will do multiplication or division for you. You will need to implement division using loops to repeat subtractions.

Subroutine: Find Time Difference (Student B)

Clock time in this project varies from 00:00:00 to 23:59:59. The clock times should be stored using three byte variables in RAM with the first byte indicates hour, the second indicates minute and the third indicates second.

You will be writing a subroutine called findTimeDiff that finds the difference in seconds between two clock times. Contract of findTimeDiff is as described below. Copy and paste the contract into your code before your subroutine. Pay attention to the case in which the alarm time is in the next day. Since a 16 bit register only counts 2^16-1 before overflowing, number of seconds in a day is however 24*60*60 > (2^16-1), we would need two words to store the time difference.
Main Program and Interrupts (Student C and D)

Our clock will count hour, minute and second for a whole day from 00:00:00 to 23:59:59. The current clock time should be stored using three byte variables in RAM with the first byte indicates hour, the second indicates minute and the third indicates second. Every time your program resets, your clock will be initialized to 12:00:00.

You will be provided with an approximately 1Hz "tick" of the clock in a skeleton project provided to you. The 1Hz "tick" of the clock is implemented using a variable named tick stored in RAM. The skeleton program sets the variable tick to 1 approximately every 1 second utilizing the subsystem master clock (SMCLK) and the TimerA module on the MSP430G2553. You do not need to worry about the peripheral setups. However, you will need to call two given subroutines named init_Clock and then init_TimerA in your main program to setup the peripherals and enable the GIE after finishing all the setups you want.

In an infinite loop in your main program after enabling GIE, if the variable tick is 1, you clear the variable tick and update your clock. You need to write code to update the hour, minute and second variables when the clock "ticks", meaning:
every tick, increment **second** by 1
every 60 ticks, increment **minute** by 1 and **second** overflow to 0
every 60 minutes, increment **hour** by 1 and **minute** overflow to 0
every 24 hours, **hour** overflow to 0

There are multiple ways to achieve the overflow mechanism described above. You should use the subroutine written by student A called **mod**. Read the above sections for the contract of subroutine **mod**.

The alarm time is stored as a three byte variables in RAM just like the current clock time. A byte variable named **isAlarmSet** needs to be reserved on RAM. A set or reset on BIT0 of this variable indicates the on/off of the alarm function. BIT0 of **isAlarmSet** is toggled at the press of the pushbutton on pin1.3. You need to write code to setup this interrupt yourself. Another 2 word variable **timeTillAlarm** needs to be reserved to keep track of when the alarm will be triggered. If the alarm is switched on, update **timeTillAlarm** using subroutine **findTimeDiff** written by student B.

In the same infinite loop in your main program where you update your clock, reduce the two words variable **timeTillAlarm** by 1 if the alarm is on. If both words in **timeTillAlarm** reaches 0, turn on LEDG to indicate the alarm time is reached. LEDG should be turned off when the alarm is turned off.

A block diagram of the main project flow is as illustrated below.
fig. Block diagram of the main program of this project

*note timeTillAlarm is 2 words. Both words need to be checked against 0.
Note that in this project your microcontroller is running constantly and consuming a lot of power. In later parts of the project you will integrate low power mode and more peripherals into the project so that your CPU only wakes up when needing to trigger alarm or update clock, and then goes back to sleep after finishing its work.

**Bonus**

Below are two options to gain INDIVIDUAL bonus points for this course. The bonus points listed is **added on top of your overall score** for this course:

**Bonus Project A (5 pts) - Day time to BCD conversion**

Write a subroutine that converts time stored using variable *hour, minute* and *second* into binary coded decimals (BCD). To score 5 points you will need to provide a well written contract of your subroutines.

**Bonus Project B (10 pts) - De-bouncing of pushbutton**

Mechanical switches like pushbuttons can give false readings (therefore false triggering of interrupts) when you press on them due to bouncing of springs or other mechanical structures. De-bouncing can be implemented in software. Basic idea is to implement a very simple low pass filter by only register a successful press after a consecutive of the same value (1 or 0) is read. You should do research on de-bouncing yourself and write a subroutine with contract for this bonus. We will test your subroutine using a blinking LED project with push button interrupt. Passing the human test gives you 5 pts for this bonus. The implementation of your subroutine and contract consists of the other 5 pts.