CSC 230 - SUMMER 2017
INTRODUCTION TO COMPUTER ARCHITECTURE
ASSIGNMENT 3
UNIVERSITY OF VICTORIA

Due: Sunday, July 16th, 2017 before 11:55pm. **Late assignments will not be accepted.**

1 Overview

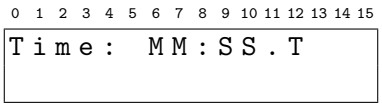
This assignment is the culmination of the assembly programming part of the course. You will implement an accurate digital stopwatch in AVR assembly, using a built-in timer and the LCD screen to display the time. As with assignment 2, you should start early, since this assignment requires you to apply virtually all of the techniques that the course has covered for embedded programming, and debugging assembly code is not easy.

The core of this assignment is a timer display which uses the LCD screen to display the number of minutes, seconds and tenths of a second since timing started. Implementing this aspect alone (with no user input functionality or other stopwatch-like features) is sufficient to obtain 13 out of 18 possible marks (and potentially up to 14 marks if you can justify that the timing is highly accurate). The remaining marks are obtained by implementing user input functionality to pause and reset the timer, along with the ability to compute lap (split) times.

Section 2 describes the basic timing functionality needed. Section 3 describes the user input functionality required for the basic stopwatch and Section 4 describes the functionality needed for the fully-featured stopwatch (which includes a lap timer)

2 Timer Display

The LCD screen has two lines, each with 16 columns, numbered from 0 to 15. The basic timer required will display the time on the LCD screen in the format below.

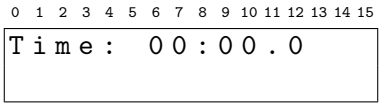


Where `MM` and `SS` are the two digit (base 10) counts of elapsed minutes and seconds (respectively) and `T` is the number of tenths of a second that have elapsed. If the text displayed, or its alignment (that is, the exact column numbers in which it appears), differs from the above, it will be considered incorrect.

When the timer is running (see below), the display will be continuously updated to reflect the current time in minutes, seconds and tenths of a second. Note that the value of `T` must always be in the range 0 through 9 (inclusive), the value `SS` must always be a two-digit base-10 number in the range 0 through 59 (inclusive) and the value `MM` must always be a two-digit base-10 number in

the range 0 through 99 (inclusive). There is no need to add support for the timer to run beyond 100 minutes.

At the `RESET` signal, the timer will be cleared to zero and display



If no user input features are implemented, the behavior of the program will be as follows. Note that this is **not** the expected behavior of the program once user input is implemented (for example, if you have correctly implement the start/stop button, the timer must not automatically start when the program starts). See the next section for details on the expected behavior when user inputs are supported.

- When the program starts (that is, at the `RESET` signal), the screen will be initialized to display the interface above, with the time initialized to `00:00.0`.
- The program will *immediately* start the timer, which will then be updated in real time as time elapses. Since no user input is implemented, there will be no facility to start or stop timing, so the timer will continue running indefinitely.
- The timer is expected to be theoretically accurate to within 1 second every ten minutes (under the assumption that the system clock is precisely 16Mhz). There is no need to handle cases where the timer runs for longer than 100 minutes (so no more than two digits will ever be needed for the minute value).

3 Basic Inputs

A basic stopwatch normally has two input features: a button to start or stop the timer (which can also be used to pause the timer) and a button to reset the clock to zero. For this functionality, we will use the `SELECT` and `LEFT` buttons. The table below gives a formal description of the expected behavior of each button. These buttons affect the primary time displayed on the top line of the LCD screen (as shown in the previous section).

Button	Behavior
SELECT	**Start/stop timer**: When support for this button has been implemented, the program will start (at the RESET signal) with the timer stopped (at 00:00.0). Pressing this button when the timer is stopped will start the timer (within 0.1 seconds of the button press) and pressing the button when the timer is running will stop the timer **immediately**. It is considered incorrect behavior for the timer to continue to tick even once after the button has been pressed to stop timing. When the timer is stopped, the button may be pressed again to restart it (without resetting the time to zero), so this button can be used as a 'pause' feature.
LEFT	**Clear timer**: When this button is pressed, the timer will immediately be cleared to 00:00.0. In cases where the start/stop button has also been implemented, the cleared timer must also enter a stopped state (such that the start/stop button can be used to start the timer). If the start/stop button has not been implemented, then the cleared timer will not be stopped (but continue counting starting from 00:00.0). In cases where lap timing support has been implemented (see Section 4), this button must **not** clear the displayed lap time or disable the display of lap times. However, it should clear the CURRENT_LAP_START variable.

Notice that the SELECT button is used to both start and stop the timer. Therefore, you will need logic similar to that used for the PAUSE feature on Assignment 2 to ensure that a single button press is not read as repeated presses.

4 Lap Timing

Digital stopwatches often support a "lap timer" (or "split timer") which allows the timing of sub-intervals with respect to the running stopwatch. For this assignment, we will use a very simple lap timing mechanism: when a lap time is requested, the program will display the 'start time' of the lap (which will either be 00:00.0 or, if the lap timer has been used before, the time at which the last lap ended) and the 'end time' of the lap (which will be a copy of the current time on the timer). After a lap time has been captured, the timer will continue running but the displayed lap time will remain static until another lap time is requested.

Tracking lap times will require keeping three extra time variables, LAST_LAP_START, LAST_LAP_END and CURRENT_LAP_START. The LAST_LAP_START and LAST_LAP_END variables track the start and end times of the most recent lap which was captured with the UP button. The CURRENT_LAP_START tracks the beginning of the 'current' lap (which will become the lap start time when the UP button is next pressed). When the timer is cleared, CURRENT_LAP_START is set to 00:00.0.

When the RESET signal occurs, the lower line of the LCD (row 1) is cleared. Lap times are only displayed after the UP button is pressed. When the UP button is pressed, the LAST_LAP_START and LAST_LAP_END times will be displayed on the bottom row of the LCD, with the start time shown on the absolute left and the end time shown on the absolute right, in the following format.

```
 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
┌──────────────────┐
│Time:  MM:SS.T    │
│MM:SS.T   MM:SS.T │
└──────────────────┘
```

When lap timing is implemented, the `UP` and `DOWN` buttons will have the following behavior.

Button	Behavior
`UP`	**Set Lap**: When this button is pressed, the `LAST_LAP_START` time is set to the `CURRENT_LAP_START` time, the `CURRENT_LAP_START` and `LAST_LAP_END` times are **both** set to the current stopwatch time, and the display of the `LAST_LAP_START` and `LAST_LAP_END` time on the bottom row of the LCD (using the format above) is enabled. The lap times will continue to be displayed until the `UP` button is pressed again (which will change the displayed values), the `RESET` signal occurs (which will restart the program and disable the display of lap times) or the `DOWN` button is pressed (which will disable the display of lap times). The `UP` button should exhibit this behavior regardless of whether the main timer is currently running or paused.
`DOWN`	**Clear Lap**: When this button is pressed, the `CURRENT_LAP_START` variable will be set to `00:00.0` and the display of lap times will be disabled (so row 1 will be blank) until the `UP` button is pressed again.

5 Implementation Suggestions

The sections below give some advice on how to implement various aspects of the timing program. You are free to ignore the advice if you have a different idea.

5.1 Storing Times

The stopwatch is expected to display times in the format `MM:SS.T` and to keep time with 1/10 second precision. It is possible to store the entire time as a single integer value equal to the number of tenths of a second elapsed, and doing so makes the process of incrementing the time comparatively easy (since it becomes a wide addition operation). However, since the time must be continuously converted to a string to be displayed on the LCD, this representation may be inconvenient, since converting a large number to a series of digits requires the implementation of division and modulus operations (as in assignment 1).

Instead, consider representing the 5-digit time value by using an array of 5 bytes, with one byte per digit. Converting such a representation to a string is relatively straightforward (since one-digit values can be converted to strings by a simple addition) and the increment operation (which advances the time by a tenth of a second) can be achieved using an algorithm like the one below (where the 5 byte array is called `A` and it is assumed that the five digits in the `MM:SS.T` representation are represented by indices $0-4$ (with 0 containing the first `M` digit and 4 containing the `T` digit).

```
//Add one to the MM:SS.T representation
void increment_time(A){
    //Make an array containing the maximum value allowed for each
    //digit in the representation. Note that the in the MM:SS.T
```

```
    //representation, the T value can be between 0 and 9, the
    //SS value can be between 00 and 59, and the MM value can
    //be between 00 and 99.
    //(There is no need to "create" this array every time the function
    // is run, since it can be placed in program memory in advance)
    //
    //                                 M  M  S  S  T
    unsigned char maximum_values[5] = {9, 9, 5, 9, 9};

    unsigned char i;
    //Add one to the last index (T)
    A[4] += 1
    //Now work backwards if any digit exceeded the limits
    //in the array above.
    for (i = 4; i > 0; i++){
        if (A[i] > maximum_values[i]){
            A[i] = 0;
            A[i-1] += 1;
        }
    }
    //If A[0] exceeded 9, then wrap around to 0.
    if (A[0] > maximum_values[0])
        A[0] = 0;
}
```

Similarly, functions can be used to copy time values (for use with saving lap times) or to set a time array to zero.

5.2 Mitigating Button Aliasing

As we noticed during the completion of assignment 2, constantly polling the buttons to detect button presses can result in unusual behavior. In particular, when buttons with low voltages (such as `RIGHT`) are pressed, polling the buttons might result in the voltage for a different button being read. As a result, you may have experienced "button aliasing" during assignment 2, where pressing one button gives the behavior for a different button. For this assignment, it is expected that your code will proactively prevent button aliasing to the greatest extent possible (although it may be impossible to prevent the aliasing in all cases). To reduce the impact of aliasing, consider modifying your button polling logic to use one of the following techniques (or a different technique of your own design):

- Add a short delay between consecutive polls to reduce the probability that an erroneous transient value will be read.
- Instead of polling 'constantly', poll the buttons at regular intervals (e.g. 10 times per second). As long as the buttons are polled more often than the timer ticks, there will be no distinguishable difference in behavior if the frequency of polling is reduced.
- Use a function to poll the ADC and return a number in the range $[0, 5]$ based on which button was detected. Instead of simply calling the function to poll the buttons, call the function twice

(or three times) and only acknowledge the button press if the return value was the same for all calls.

6 Sample Run Illustration

The table below shows the state of the LCD screen before and after various inputs during a sample run of the program. Obviously, your code should work on all possible input sequences, not just the ones below.

Event	LCD (before)	LCD (after)
Program starts	N/A	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:00.0
10 seconds elapse	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:00.0	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:00.0
`SELECT` pressed (Timer Starts)	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:00.0	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:00.0
0.1 seconds elapse	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:00.0	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:00.1
15 seconds elapse	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:00.1	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:15.1
65.5 seconds elapse	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:15.1	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 01:20.6
`LEFT` pressed (Timer Clears)	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 01:20.6	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:00.0
5 seconds elapse	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:00.0	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:00.0
`SELECT` pressed (Timer Starts)	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:00.0	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:00.0
1 second elapses	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:00.0	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:01.0
15 seconds elapse	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:01.0	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:16.0

Event	LCD (before)	LCD (after)
`SELECT` pressed (Timer Stops)	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:16.0	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:16.0
20 seconds elapse	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:16.0	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:16.0
`SELECT` pressed (Timer Starts)	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:16.0	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:16.0
1.5 seconds elapse	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:16.0	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:17.5
`UP` pressed (Set Lap)	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:17.5	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:17.5 00:00.0 00:17.5
5 seconds elapse	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:17.5 00:00.0 00:17.5	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:22.0 00:00.0 00:17.5
`UP` pressed (Set Lap)	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:22.0 00:00.0 00:17.5	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:22.0 00:17.5 00:22.0
3 seconds elapse	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:22.0 00:17.5 00:22.0	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:25.0 00:17.5 00:22.0
`DOWN` pressed (Clear Lap)	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:25.0 00:00.0 00:17.5	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:25.0
4.6 seconds elapse	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:25.0	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:29.6
`UP` pressed (Set Lap)	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:29.6	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 Time: 00:29.6 00:00.0 00:29.6

7 Evaluation

Submit all `.asm` and `.inc` files needed to assemble your assignment electronically via conneX (including all of the necessary LCD library files). Your code must assemble, upload and run correctly on the ATmega2560 boards in ECS 249 using the toolchain and methodology described in Lab 3.

If your code does not assemble as submitted, you will be allowed to spend some of your demo time attempting to fix it, but the demo will be limited to 10 minutes, and you will lose marks if you miss parts of the evaluation by spending time fixing your code. If your code cannot be assembled during the demo, you will receive a mark of at most 2.

This assignment is worth 9% of your final grade and will be marked out of 18 during an interactive demo with an instructor. Demos must be scheduled in advance (through an electronic system available on conneX). If you do not schedule a demo time, or if you do not attend your scheduled demo, you will receive a mark of zero.

The marks are distributed among the aspects of the assignment as follows.

Marks	Aspect
11	The basic, non-interactive timer described in Section 2 functions correctly with reasonably accurate timing (such that a human observer cannot notice any discrepancy).
1	Functions are used as appropriate instead of duplicated code, and all functions exhibit correct register preservation using the stack.
1	Multi-byte persistent data (such as the current time) is stored primarily in data memory instead of being held continuously in registers. It is permissible to temporarily use registers to hold values while your code is working with them.
1	The timing is theoretically accurate to within 1 second out of 10 minutes when the 16Mhz system clock is assumed to be exact.
1	The `SELECT` button correctly implements the start/stop feature described in Section 3. The code must ensure that a single press of the button be read only once (regardless of how long the button is held down).
1	The `LEFT` button correctly implements the clear feature described in Section 3.
1	The `UP` button correctly implements the 'set lap' feature described in Section 4. Note that unless the display of lap times is correct, it is impossible to evaluate this component (and the component below).
1	The `DOWN` button correctly implements the 'clear lap' feature described in Section 4.

If your code exhibits severe effects of button aliasing (making it difficult to accurately test the input features), the evaluator may deduct 1 mark from your total. If your code is not well organized, or if it is poorly documented, the evaluator may ask you explain any aspects that are unclear. If you are unable to do so, up to 2 marks may be deducted. To be clear, you are not required to have spotless, perfectly organized code, but you should be prepared to explain any hard-to-read parts of your code.

You are permitted to delete and resubmit your assignment as many times as you want before the due date, but no submissions or resubmissions will be accepted after the due date has passed. You will receive a mark of zero if you have not officially submitted your assignment (and received a confirmation email) before the due date. Ensure that each submitted file contains a comment with your name and student number.

Ensure that all code files needed to assemble, upload and run your code in ECS 249 are submitted. Only the files that you submit through conneX will be marked. The best way to make sure your submission is correct is to download it from conneX after submitting and test it. You are not permitted to revise your submission after the due date, and late submissions will not be accepted, so you should ensure that you have submitted the correct version of your code before the due date. conneX will allow you to change your submission before the due date if you notice a mistake. After submitting your assignment, conneX will automatically send you a confirmation email. **If you do**

not receive such an email, you did not submit the assignment. If you have problems with the submission process, send an email to the instructor **before** the due date.