

Homework Assignment 5.2

Due: Mon, 28 Feb 2022 17:30:30 (approximately 2 days from the time this page was loaded)

[13 points possible] [100 penalties possible] [0 penalties graded so far]

Weight = 1.00

This homework is a review of programming techniques for GPIO, Interrupt handling, and basic recursion. To complete this homework, you should create a project in System Workbench with the Standard Peripheral firmware. Write the following subroutines, and put them all in the same file named hw52.c. You should use the SystemWorkbench template file, but remove the `main()` subroutine from it. The result should look like this:

```
#include "stm32f0xx.h"
// That's it. The #include is all you need to start.
```

To test your subroutines, you can link hw52.c against the test.c file (that can be found at some point) on the homework web page. The test module will exercise these subroutines and check their results and effects.

The problems are exactly the same as hw5. You already know how to do them! You're simply going to practice doing them with the C language instead of assembly language. (Question 1 is pretty easy, yes?) You now have a deep understanding for exactly what is happening when you write C statements.

Repeat: This is just a copy-and-paste of homework 5. All the places below that talk about assembly language are what you did for homework 5. Now you will do the same thing in C. Yes, question 1 is obviously easy.

The test.c file provides many good examples of how to manipulate configuration registers using the C language. Almost everything we do in the class after the midterm lab practical exam will be done using C. We find it takes a little effort to get used to working with pointers and structure fields to do peripheral configuration. This homework assignment will get you ready for lab 6.

Q1: recur [1 point]

Translate ("hand compile") the following C function into assembly language.

```
// Just another strange recursive function.
unsigned int recur(unsigned int x) {
    if (x < 3)
        return x;
    if ((x & 0xf) == 0)
        return 1 + recur(x - 1);
    return recur(x >> 1) + 2;
}
```

Q2: enable_portb [1 point]

Write an assembly language subroutine named `enable_portb` that configures the RCC to enable the clock to GPIO Port B but leaves the other clock control bits as they were.

Q3: enable_portc [1 point]

Write an assembly language subroutine named `enable_portc` that configures the RCC to enable the clock to GPIO Port C but leaves the other clock control bits as they were.

Q4: setup_pb3 [1 point]

Write an assembly language subroutine named `setup_pb3` that configures pin PB3 for the following:

- Input
- Pull-down resistor enabled

It should not change any other configuration for any other pins.

Q5: setup_pb4 [1 point]

Write an assembly language subroutine named `setup_pb4` that configures pin PB4 for the following:

- Input
- Neither pull-down nor pull-up resistor enabled

It should not change any other configuration for any other pins.

Q6: setup_pc8 [1 point]

Write an assembly language subroutine named `setup_pc8` that configures pin PC8 for the following:

- Output
- Output Speed: High Speed

It should not change any other configuration for any other pins.

Q7: setup_pc9 [1 point]

Write an assembly language subroutine named `setup_pc9` that configures pin PC9 for the following:

- Output
- Output Speed: Medium Speed

It should not change any other configuration for any other pins.

Q8: action8 [1 point]

Write an assembly language subroutine named `action8` that reads the state of PB3 and PB4. If PB3 is high and PB4 is low, then set PC8 to 0. Otherwise, set PC8 to 1. To test this, you should wire a push buttons to PB3 and PB4 as you did in lab experiment 4. When either button is pressed, it should connect the appropriate pin to a logic high. The green LED should be illuminated except when PB3 is high and PB4 is low.

Q9: action9 [1 point]

Write an assembly language subroutine named `action9` that reads the state of PB3 and PB4. If PB3 is low and PB4 is high, then set PC9 to 1. Otherwise, set PC9 to 0. The blue LED should illuminate only when the button for PB4 is pressed and PB3 is not.

Q10: External Interrupt Handler [1 point]

Write an assembly language subroutine to act as the Interrupt Service Routine (ISR) for line 2 (pin 2 of the selected port). You should look up the name for this ISR in the startup/startup_stm32.s file and **copy and paste it** to avoid making any mistakes. It should *acknowledge* the interrupt by writing a 1 to the appropriate bit of the EXTI_PR register. It should also increment the global variable named 'counter'.

Q11: enable_exti [1 point]

Write an assembly language subroutine named `enable_exti` that does the following:

- enable the system clock to the SYSCFG subsystem.
- set up the appropriate SYSCFG external interrupt configuration register (see the FRM, page 177) to use pin PB2 for the interrupt source
- configure the EXTI_RTSR (see the FRM, page 224) to trigger on the rising edge of PB2
- set the EXTI_IMR to not ignore pin number 2
- configure the NVIC to enable the interrupt for the ISR

Q12: (the interrupt handler for Timer 3) [1 point]

Write an assembly language subroutine that acts as an Interrupt Service Routine (ISR) for the Timer 3 update interrupt. You must look up the name for this interrupt and add the proper assembler directives to indicate that it consists of Thumb instructions. It should do the following:

- Toggles PC9 (the blue LED).
- Acknowledges the interrupt by clearing the Timer 3 update interrupt flag.

Timer 3, for what you're doing here, works just like Timer 6. It has more features, but if you're not using them, you don't have to pay any attention to them. You know how to turn on the update event interrupt for Timer 6. It works the same way for Timer 3. You should be ready to use any timer in this way.

Q13: enable_tim3 [1 point]

Write an assembly language subroutine named `enable_tim3` that does the following:

- Enables the system clock to the timer 3 subsystem.
- Configures the Auto-Reload Register and Prescaler of Timer 3 so that an update event occurs exactly four times per second.
- Set the DIER of Timer 3 so that an interrupt occurs on an update event.
- Write the appropriate bit to the NVIC ISER so that the interrupt for the Timer 3 update event is enabled.
- Enable the counter for Timer 3.