# Pre-lab Assignment 3

Due: Fri, 18 Sep 2020 23:59:59 (approximately 83 days ago)
[Score: 25 / 30 points possible]

Most embedded applications use many peripherals (GPIO being one among them). Embedded software controls a peripheral by writing to a set of dedicated memory locations (often called configuration registers). It can read the state of the peripheral by reading from a set of memory locations (often called the status register). Almost all of the accesses are done through setting or clearing particular bits in these "control" and "status" registers. Together, these are generally referred to as "I/O registers". (It is also important to remember that I/O registers are distinct from CPU registers such as R0 or SP. I/O registers are referred to by accessing a location in memory using a load or store instruction.)

**Setting a bit without changing others:**

Whenever we set a bit/bits, we want do so without modifying the values of the other bits in the register. Here is an example:

Say R0 = 0x10A0 C000 and we want to set bit 16, so that the value ends up as R0 = 0x10A1 C000.

This is achieved using an OR operation which will turn on all the bits in an "ORed-in" value while leaving all the bits in the target that were already set to one.

In the above example our ORed-in value would be 0x0001 0000.

OR the value into the target with a statement like: R0 = R0 | value;

These are the most common set of instructions used in embedded software.

Armed with this information, answer the following questions:

## Academic Integrity Statement [0 ... -100 points]

By typing my name, below, I hereby certify that the work on this prelab is my own and that I have not copied the work of any other student (past or present) while completing it. I understand that if I fail to honor this agreement, I will receive a score of zero for the lab, a one letter drop in my final course grade, and be subject to possible disciplinary action.

Raghuram Selvaraj ✓

## (1) [1 point]

Given R0 = 0x8042 4321 what is the **minimal** value must you OR to it to set R0 = 0x90CB C7F3? (Specify the number as 0x____ ____.)

0x1009 84d2 ✗

## (2) [1 point]

Write an assembly instruction to perform R0 = R0 | R1.

```
orrs r0, r1
```
✓

## (3) [1 point]

Given R0 = 0xFB31 8000 what value should you OR with it to set bit 9 of R0 without affecting any of the other bits? (To be clear, bit 0 is the rightmost of least significant bit. Bit 31 is the leftmost or most significant bit.)

```
0x0000 0200
```
✓

## (4) [1 point]

Given R0 = 0xC312 3381 what is the **minimal** value you should OR with it so that R0 = 0xF356 F389? (Specify the number as 0x____ ____.)

```
0x3044 C008
```
✓

## (5) [1 point]

Given the results of questions (2) and (3), write a simple assembly language subroutine that expects two arguments (in R0 and R1) that represent, respectively, an original value and a value to OR into it. It should return the new value with the corresponding bits set. Name (label) your subroutine "set_bits". (Just the subroutine is fine. You do not need to make it .global or use the assembler directives for .cpu, .syntax, etc. We'll be grading this prelab by hand.) Remember to return from the subroutine by using either **`bx lr`** or starting the subroutine with **`push {lr}`** and returning with **`pop {pc}`**. Either way works here.

```
.text
.global set_bits
set_bits:
        push {lr}
        orrs r0, r1
        pop {pc}
```
✓

## (6) [1 point]

Now imagine your goal is to clear a the bits in R0 that are set in R1. For example, assume R1 = 0x0000 0020. In that case, you want to use it to clear bit 5 of R0. If R0 = 0x10A2 0030 you want it to become 0x10A2 0010. If R0 = 0xFFFF FFDF you want it to remain 0xFFFF FFDF.

If R1 was, instead, 0x0000 0030, then you would want to use it to clear bits 4 and 5 of R0. If R0 = 0x10A2 0030 you want it to become 0x10A2 0000. If R0 = 0xFFFF FFDF you want it to remain 0xFFFF FFCF.

To do this, you must use a BIT-CLEAR operation. What instruction should you use to make the change to R0 if the bits set in R1 are the bits you want to clear in R0?

BICS r0, r1 ✓

---

## (7) [1 point]

What **minimum** value for R1 would you use if you wanted an initial value for R0 of 0xFFFF FFFF to change to 0xFFFF EDB7? (Specify the number as 0x____ ____)

0x0000 1248 ✓

---

## (8) [1 point]

Now assume that R0 = 0x1032 FF00. What is the **minimal** value for R1 that you would use for the instruction in question (6) so that R0 would be changed to R0 = 0x1022 EE00? (Specify the number as 0x____ ____)

0x0010 1100 ✓

---

## (9) [1 point]

Write an instruction to shift the 16 most significant bits in R1 down to the least significant bits of R1. (The original least significant 16 bits are thrown away.) For instance, 0xffff 0000 should become 0x0000 ffff. 0x00ff ff00 should become 0x0000 00ff.

lsrs r1, r1, #16 ✓

---

## (10) [1 point]

Based on your answers for questions (6) and (9), write a simple subroutine in assembly language that expects two arguments, R0 and R1. R0 contains a 16-bit value. You don't have to care about the 16 most significant bits of R0. R1 contains two sets of 16-bit values. The subroutine should copy only the lower 16 bits of R1 into R2. (Do this by loading 0x0000 ffff into R2 and then ANDing R1 into R2.) Then it should shift the upper 16 bits of

R1 into the lower 16 bits of R1 and use those bits to clear R0 as you did in question (6). Finally, it should OR the value in R2 into R0. Stated succinctly, the upper 16 bits set in R1 are cleared in the lower 16 bits of R0, and the lower 16 bits set in R1 are set in R0. This is effectively what the GPIO BSRR register does.

Remember that your subroutine should have instructions to **return** properly.

```
bsrr:
    push {lr}
    ldr r2, =0xffff
    ands r2, r1
    lsrs r1, r1, #16
    bics r0, r1
    orrs r0, r2
    pop {pc}
```

✓

## (11) [1 point]

For the following questions, consult the STM32F0x1 Family Reference, Chapter 7, Reset and clock control (RCC).

GPIO ports are *peripherals* on the STM32F0 microcontroller. Almost every peripheral is disabled by default in order to avoid consuming power in subsystems that are unused. What is the common name for the I/O register within the Reset and Clock Control (RCC) subsystem that is used to power on, enable a clock for, and generally activate GPIO ports?

RCC_AHBENR ✓

## (12) [1 point]

The I/O register in question (11) has 32-bits. Which bit must be set to 1 to enable GPIO Port C? (Just write the bit position number, where the most significant bit is 31 and the least significant bit is 0.)

19 ✓

## (13) [1 point]

What the initial value (the value set at system power-up) of the I/O register you identified in question (11)? (Specify the number as 0x____ ____)

0x0000 000c ✗

## (14) [1 point]

What is the base address of the RCC I/O registers? (See Table 1 on page 46.) (Specify the number as 0x____ ____)

0x4002 1000 ✓

---

## (15) [1 point]

What is the address of the I/O register you named in question (11)? State the absolute hexadecimal address, not its offset in the RCC. (Specify the number as 0x____ ____)

0x4002 1014 ✓

---

## (16) [1 point]

Write the assembly language instructions you would need to load the value of the register you name in question (11) into R3. Just a code segment with a few instructions. This should not be a full subroutine.

```
ldr r0, =0x40021000
ldr r3, [r0, #0x14]
```

✓

---

## (17) [1 point]

Combine what you did for questions (5), (11), and (16) to write a subroutine that enables only the clocks for GPIO Ports B **and** C but does not turn off the clocks to other things that are already enabled. Name (label) your subroutine `enable_ports_bc`.

```
.equ RCC, 0x40021000
.equ AHBENR, 0x14
.equ IOPBEN, 0x40000
.equ IOPCEN, 0x80000

enable_ports_bc:
    push {lr}
    ldr r0, =RCC
```

✓

---

## (18) [1 point]

What is the base address of the GPIOC I/O registers? (See Table 1 again.) (Specify the number as 0x____ ____)

0x4800 0800 ✓

---

## (19) [1 point]

What is the address (not the offset from the GPIOC register base) of the GPIOC MODER I/O register? (You'll have to use Table 1 and click on the link to go to Section 9.4.12 on page 171.) (Specify the number as 0x____ ____)

0x4800 0800 ✓

---

## (20) [1 point]

What is the address (not the offset) of the GPIOC ODR I/O register? (Specify the number as 0x____ ____)

0x4800 0814 ✓

---

## (21) [1 point]

What is the address (not the offset) of the GPIOC IDR I/O register? (Specify the number as 0x____ ____)

0x4800 0810 ✓

---

## (22) [1 point]

What is the address (not the offset) of the GPIOB IDR I/O register? (Careful... GPIOB IDR, not GPIOC IDR.) (Specify the number as 0x____ ____)

0x4800 0410 ✓

---

## (23) [1 point]

The GPIOC (Port C) MODER uses two bits for each external I/O pin to configure the operation of the pin. To configure a pin for input, both bits must be zero. To configure a pin for output, the corresponding MODER bits must be '01'. Thereafter, any update to the ODR bit for the pin will be reflected on the output for that pin.

Write a subroutine that does the following:

- Enable the GPIO Port C peripheral as you did with enable_ports_bc, above.
- Configure the GPIO Port C MODER so that pin 7 is set for output. (i.e., set bits 14 and 15 of the MODER appropriately) (A previous version of this question incorrectly suggested setting bits 16 and 17.)
- Read from the ODR, set bit 7 of that value to '1', and write it back to the ODR.

Remember that the yellow LED on your development board is connected to pin 7 of GPIO Port C, so you will know if your program works correctly.

```
.equ RCC, 0x40021000
.equ AHBENR, 0x14
.equ IOPCEN, 0x80000
.equ GPIOC, 0x48000800
.equ MODER, 0x0
.equ ODR,   0x14

ldr r0, =RCC
ldr r1, [r0, #AHBENR]
ldr r2, =IOPCEN
orrs r1, r2
str r1, [r0, #AHBENR]

ldr r0, =GPIOC
ldr r1, [r0, #MODER]
movs r2, #3
lsls r2, r2, #14
orrs r1, r2
str r1, [r0, #MODER]

ldr r1, [r0, #ODR]
movs r2, #1
lsls r2, r2, #7
orrs r1, r2
str r1, [r0, #ODR]
```

## (24) [1 point]

For the following questions, consult the STM32F091RCT6 datasheet.

For the the general purpose I/O pins of the microcontroller listed in table 13 with a "TTa" designation, what are the voltage limits (negative and positive)? (See Table 21)

-0.3V - 4.0V

## (25) [1 point]

How much current is the microcontroller able to sink (or source) over a single I/O pin? (See Table 22)

25 sunk (-25 source) mA ✓

---

## (26) [1 point]

What is the maximum current rating for the sum of all the I/O pins combined? (Table 22 again)

-80 source, 80 sunk mA ✓

---

## (27) [1 point]

What is the maximum current rating for the entire microcontroller? (This is usually understood to be the total current going through either power or ground pins of the microcontroller.)

-120/120 mA ✓

---

## (28) [1 point]

For the following questions, consult the STM32F091 Development Board User Manual.

Which microcontroller pin is connected to the red user output LEDs on the development board?

PC6 ✓

---

## (29) [1 point]

Which microcontroller pin is connected to the user pushbutton (SW2) on the development board?

Port A, pin 0 ✓

---

## (30) [1 point]

List the pins that are available to use for Port D of the microcontroller.

Port D, pins 0-15 ✗