1. (1) The Tiva Launchpad board has a multi-colored LED that is controlled through three GPIO lines on Port F, one for red, one for green, and one for blue. The red LED is attached to line **PF8**, the green LED is attached to line **PF4**, and the blue LED is attached to line **PF0**. Write a sequence of instructions to showcase all three colors by create a loop that selects one color at a time, cycling through all three (in the cycle of **red, green, blue, blue, green, red**) by changing the value being written to the port. (Hint: Related base address 0x40000000 and offset 0x38 and **be sure to show the delay time between two lights and try to set the delay time to 3 seconds**.)

(2) Consider an interrupt being caused by a **32-bit** timer (the 16/32-Bit Timer 0A in the Tiva TM4C1233H6PM Microcontroller) counting down to zero in EXAMPLE 15.4.

(a) Write a sequence of instructions to set the match value if the timer is required to expire in **5 seconds** and the frequency of the system clock is set to 15 MHz**.** (Hint: Store the matching value in the register at base address 0x40000000 and offset 0x30.)

(b) write a programthat includes Subroutine **Get\_Match** (using **an empty descending stack**, with SP=0x40000090 initially, to avoid the side effect)**,** and calls to the subroutine. Subroutine **Get\_Match** computes the match value and put it in R2. Assume the number of seconds to expire and the system clock frequency in MHz are respectively in R0 and R1 before calling the subroutine (suppose both numbers are **integers** and **be sure to use (2a) as a test case to show the execution results**).

2. If the words at addresses 0x42000000 through 0x43FFFFFF are the bit-banded alias of the bits at addresses 0x40000000 through 0x400FFFFF (the bit-banded region), (**Note: comment** the solutions to problems (1)~(3) before the program in problem (4).)

(1) give the bit (its bit number and its word address) to be accessed in the bit-banded region if its bit-banded alias address is 0x0x43070010.

(2) set the bit in (1) **without** using bit-band operations.

(3) write a sequence of instructions to

(a) read bit 12 of a word at address 0x40038000 (with bit-banded operations)

(b) read bit 12 of a word at address 0x40038000 (without bit-banded operations)

(4) write a programthat includes 2 subroutines **Conv2Alias** and **Con2Region** (both using **a full descending stack**, with SP=0x40000080 initially, to avoid the side effect)**,** and calls to the two subroutines.

(a) Subroutine **Conv2Alias** computes the bit-banded alias address and puts it in R1. Assume the address of a word in the bit-banded region and the bit number in the word to be accessed are respectively given in R2 and R3 before calling Subroutine **Conv2Alias**. (**use (3) as a test case to show the execution results**.)

(b) Subroutine **Con2Region** computes the address of a word in the bit-banded region and the bit number in the word to be accessed, and put them respectively in R2 and R3. Assume the bit-banded alias address is in R1 before calling Subroutine **Conv2Region**. (**use (1) as a test case to show the execution results**.)

3**.** (1) Consider in a 7-bit wide interrupt priority register with priority group set to 3. If the interrupt priority register of an IRQ = 11101111, **calculate** and **comment** (a) the group (preemption) priority and (b) the subpriority of the IRQ **in the program in (2)**.

(2) Write a program that includes subroutine **Priority** (all using **a full ascending stack**, with SP=0x400000A0 initially, to avoid the side effect)and a call to the subroutine. (**Be sure to use (1) as a test case to show the execution results.**) Subroutine **Priority** computes and finally puts the group (preemption) priority and the subpriority of an IRQ respectively in R0 and R1. Assume the width of the interrupt priority register is in R2, the priority group number is in R3, and the 8-bit interrupt priority register content of the IRQ is in R4 before calling the subroutine.

(3) Write a program that includes **4 subroutines** **GW\_PL**, **GW\_PR**, **GW\_NPL** and **GW\_NPR** respectively for (a)~(d) below to get the width of an interrupt priority register (address 0x40000400) and put the width respectively in R3~R6. (All using **an empty ascending stack**, with SP=0x400000B0 initially, to avoid the side effect)and **calls to the subroutines**. (**be sure to find a way to show the result when the width is 5)**.

(a) using LSL with test pattern LSL (b) using LSR with test pattern LSR

(c) using LSL without test pattern LSL (d) using LSR without test pattern LSR

**4.** (1) Rewrite Program 15-1 to include the following 3 declarations and

**MsgY DCB “divide-by-0 did happen!”, 0**

**MsgN DCB “divide-by-0 did not happen”, 0**

**MsgH DCB “Hard fault did happen!”, 0**

1. check the usage fault status register, **write string MsgY to memory with starting address 0x20000040** if a divide-by-zero **has taken place**, and **write string MsgN to memory with starting address 0x20000080** if a divide-by-zero **has not taken place**.

(Be sure to show the related **memory change** by **setting** or **not setting** **DIV\_0\_TRP**, the **3 steps** in the **entry** sequence upon processor exception, the **2 steps** in the **exit** sequence, and give the **type of the stack** used here.)

(b) create **HARD faults** using the 2 ways mentioned in the textbook (Be sure to show the execution results of the created HARD faults by **writing string MsgH to memory with starting address 0x200000C0** in the HARD fault handler).

(2) Rewrite the **UART** Program in Sec. 16.2.5 to include the declaration of the string **“Final Exam for (ID-Name)!”** as variable **IDStr**. Use calls to subroutine **Transmit** to do the following 2 steps (Hint: show the results by **F5 (Run)**)

1. display **reversely** **the string words** in the window of **UART #1** after program execution.
2. display **the string, reversely the string**, and continuously **the string** in the window of **UART #1** after program execution.

**Note:** Please

1. put necessary **Keil Tool DEBUG window screenshots** to show your **program** and **execution results** including **highlighted necessary initial assumptions and subsequent memory, register and stack changes**,
2. **comment student ID+your English name in every screenshots**, and
3. put reports into one word file named by student\_ID+your\_name.