

**Exam 1****Date:** March 1, 2019

UT EID: \_\_\_\_\_

Printed Name: \_\_\_\_\_  

Last,
First

Your signature is your promise that you have not cheated and will not cheat on this exam, nor will you help others to cheat on this exam:

Signature: \_\_\_\_\_

**Instructions:**

- Closed book and closed notes. No books, no papers, no data sheets (other than the last two pages of this Exam)
- No devices other than pencil, pen, eraser (no calculators, no electronic devices), please turn cell phones off.
- Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space (boxes) provided. **Do Not write answers on back of pages as we will not be scanning the back of your exam sheets.**
- You have 75 minutes, so allocate your time accordingly.
- Unless otherwise stated, make all I/O accesses friendly and all subroutines AAPCS compliant
- Please read the entire exam before starting.

<b>Problem 1</b>	10	
<b>Problem 2</b>	15	
<b>Problem 3</b>	10	
<b>Problem 4</b>	20	
<b>Problem 5</b>	10	
<b>Problem 6</b>	15	
<b>Problem 7</b>	20	
<b>Total</b>	100	

(2) **Problem 1a.** The intent of the function Fun1 is to call another function Output with the values from 1 to 100. There is one bug. Fix the bug in the code by adding, changing or removing as necessary.

```
void Fun1(void){
    uint16_t i = 1;
    while (i<=100){ // add braces to group statements
        Output(i);
        i = i+1;
    }
}
```

(2) **Problem 1b.** The input is a 32-bit signed value in R0, and the return value, also 32-bit signed, is in R0. The intent of the function Fun2 is to return  $1024 \cdot \text{input} + 25$ . Choose the best answer that describes this function.  **$1024 \cdot \text{input}$  is a 42 bit number,  $1024 \cdot \text{input} + 25$  is a 43 bit number**

```
Fun2    LSL    R0,R0,#10
        ADD    R0,R0,#25
        BX     LR
```

- A) This function is not AAPCS compliant.
- B) There is a bug, should have used ASR.
- C) There is a possibility for overflow.**
- D) There is a bug, should have used ADDS.
- E) There is nothing wrong, it always works.

C

(2) **Problem 1c.** What is the value of Diff when the following function is executed?

```
uint8_t Data[4] = {1,2,4,8};
uint8_t Diff;
void Fun3(void){
    Diff = Data[1] - Data[2];
}
```

**2-4 would seem to be -2, but as an 8-bit unsigned number it is 254**

(2) **Problem 1d.** Which equation describes the **power** dissipated in a resistor?

- A)  $P = V/I$
- B)  $P = V^2 \cdot R$
- C)  $P = I^2 \cdot R$**
- D)  $P = V/R^2$
- E) None of the above

**$P=V \cdot I$   
 $V=I \cdot R$  so  
 $P=I \cdot I \cdot R$**

(2) **Problem 1e.** What is the value in the variable z after executing this C code?

```
uint16_t x = 0x1234;
uint16_t y = 0xEDCB;
uint16_t z = x&&y; // && is boolean, & is bitwise
```

**True, any nonzero number**

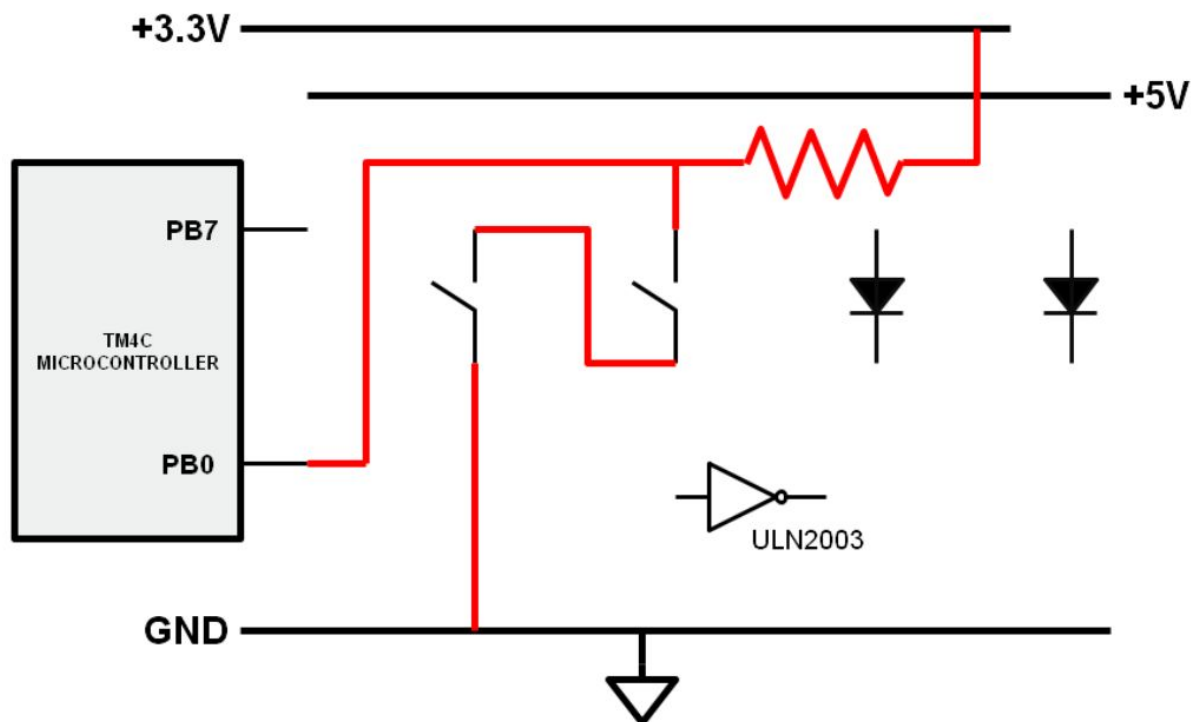
(15) **Problem 2.** Consider the following C function that finds the integer part of the square-root of a number (num). Convert the C code to assembly code so that it implements the same functionality in each box.

<pre>uint16_t root; uint32_t sqr;</pre>	<pre>AREA DATA, ALIGN=2 root space 2 sqr space 4</pre>
<pre>const uint32_t num=1000;</pre>	<pre>AREA  .text , CODE, READONLY, ALIGN=2 num DCD 1000</pre>
<pre>void Func(){     root = 1;     sqr = 1;</pre>	<pre>Func     LDR R0,=root     MOV R1,#1     STRH R1,[R0]     LDR R2,=sqr     STR R1,[R2]</pre>
<pre>while (sqr &lt;= num){     root++;     sqr = root*root; }</pre>	<pre>    LDR R3,=num     LDR R3,[R3] Loop     LDR R1,[R2]     CMP R1,R3     BHI Next     LDR R1,[R0]     ADD R1,#1     STRH R1,[R0]     MUL R1,R1,R1     STR R1,[R2]     B Loop</pre>
<pre>    root--; }</pre>	<pre>Next     LDR R1,[R0]     SUB R1,#1     STRH R1,[R0]     BX LR</pre>

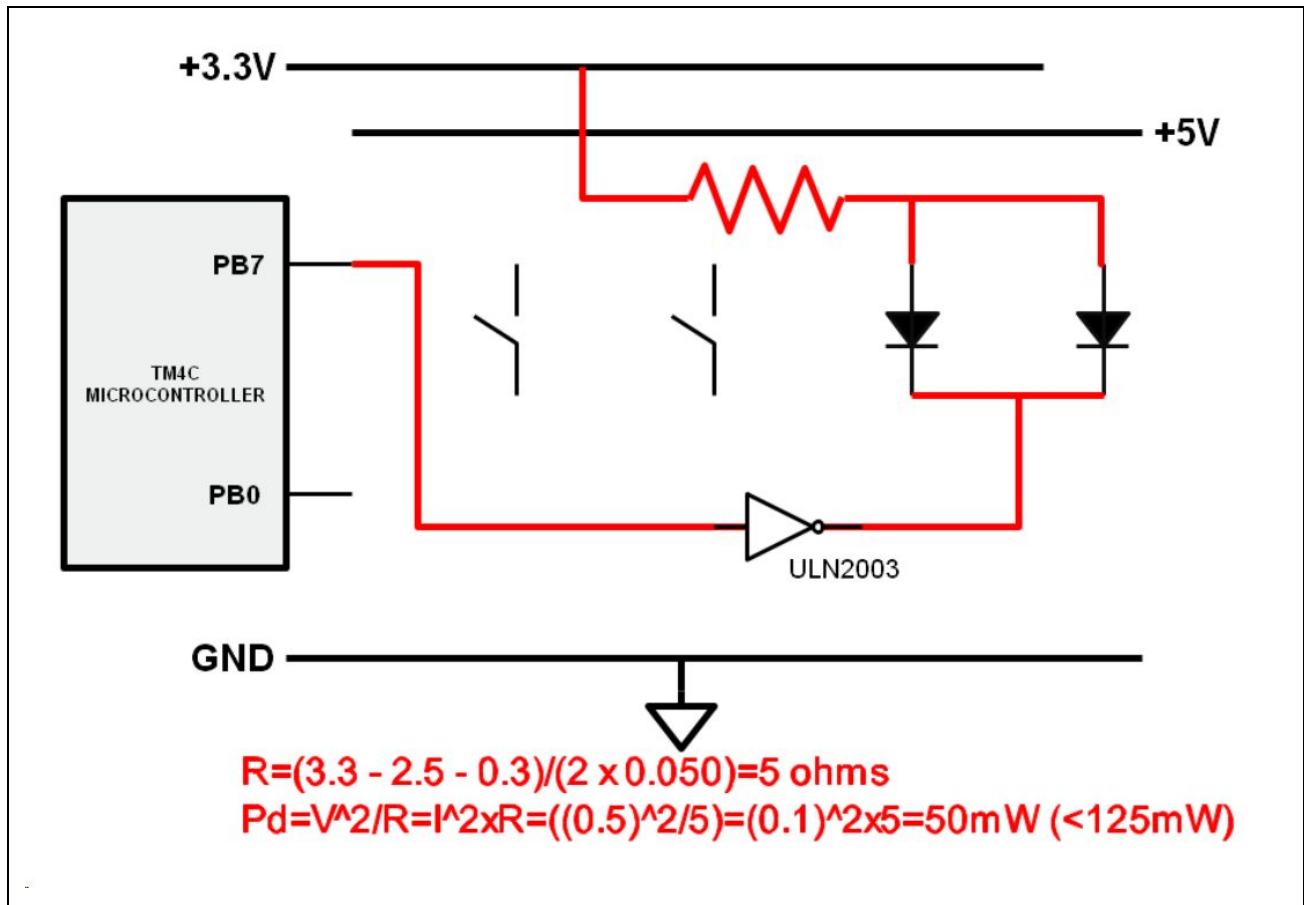
**(10) Problem 3.** Consider the following assembly function that performs input/output on Port B. You may assume Port B has been initialized so PB0-3 are inputs, and PB4 is an output. Convert the assembly function to C code so that it implements the same functionality.

<pre> OneOne   LDR R1,=GPIO_PORTB_DATA_R   LDR R0,[R1]   AND R0,R0,#0x0F   CMP R0,#0x01   BEQ One   CMP R0,#0x02   BEQ One   CMP R0,#0x04   BEQ One   CMP R0,#0x08   BEQ One   MOV R0,#0x0   B Done One  MOV R0,#10 Done STR R0,[R1]      BX LR </pre>	<pre> void OneOne(void){   uint32_t in;    in = GPIO_PORTB_DATA_R&amp;0x0F;   if ((in == 0x01)   (in == 0x02)         (in==0x04)   (in == 0x08)) {     GPIO_PORTB_DATA_R = 0x10;   }   else{     GPIO_PORTB_DATA_R = 0x00;   } } </pre>
--	---

**(10) Problem 4a:** Using only ONE 10 k $\Omega$  resistor, interface both switches to the microcontroller Port B, bit 0 such that the input voltage is LOW when both switches are closed and HIGH otherwise. The microcontroller is powered by 3.3V. Show your circuit below



**(10) Problem 4b:** Using only ONE resistor, interface both LEDs to the microcontroller Port B, bit 7 using positive logic. The LEDs are either both ON or both OFF at the same time. The operating point for each LED is 50mA at 2.5V. Assume both LEDs are identical. The microcontroller's output high/low voltages are 3.3V and 0V, respectively. The  $V_{OL}$  for the ULN2003 driver is 0.3V. You have +5V, +3.3V, and GND to which you can connect your components. Design your circuit such that the power dissipated in the resistor is less than 125mW (1/8W). Show your circuit below, compute the resistor value needed for the above operating point, and determine the power dissipated in the resistor when the LED is on. Show your calculations



Resistor Calculations

Power Calculations

**(10) Problem 5:** The objective of this question is to design a system that creates an orange light using the three-color LED of the LaunchPad. If you output a 0,0,1 to PF3,PF2,PF1, the LED will be red. If you output 1,0,1 to PF3,PF2,PF1, the LED will be yellow. To make your eyes think you see orange you can alternate red with yellow.

Address	7	6	5	4	3	2	1	0	Name
\$400F.E608			GPIOF	GPIOE	GPIOD	GPIOC	GPIOB	GPIOA	SYSCTL_RCGCGPIO_R
\$4002.53FC				DATA	DATA	DATA	DATA	DATA	GPIO_PORTF_DATA_R
\$4002.5400				DIR	DIR	DIR	DIR	DIR	GPIO_PORTF_DIR_R
\$4002.551C				DEN	DEN	DEN	DEN	DEN	GPIO_PORTF_DEN_R

```
#define SYSCTL_RCGCGPIO_R (*(volatile uint32_t *)0x400FE608)
```

```
#define GPIO_PORTF_DATA_R (*(volatile uint32_t *)0x400253FC)
```

```
#define GPIO_PORTF_DIR_R (*(volatile uint32_t *)0x40025400)
```

```
#define GPIO_PORTF_DEN_R (*(volatile uint32_t *)0x4002551C)
```

**(5) Part a)** Show the initialization function that configures PF3,PF2,PF1 as outputs. You must write friendly code. Since you are not using PF0, you need not unlock anything. You must solve this in C.

```
void Orange_Init(void){volatile uint32_t delay;
    SYSCTL_RCGCGPIO_R |= 0x20; // Port F clock
    Delay = SYSCTL_RCGCGPIO_R; // wait for stabilization
    GPIO_PORTF_DIR_R |= 0x0E; // PF3,2,1 output
    GPIO_PORTF_DEN_R |= 0x0E; // PF3,2,1 enabled
}
```

**(5) Part b)** Show the main program that calls the initialization function in part a), and executes an infinite loop that creates the orange light. You must write friendly code. You must solve this in C.

```
int main(void){
    Orange_Init();
    GPIO_PORTF_DATA_R = (GPIO_PORTF_DATA_R&~0x0E)|0x02; // friendly, red
    while(1){
        GPIO_PORTF_DATA_R ^= 0x08; // toggle, friendly
    }
}
Alternate solutions:
int main(void){ Orange_Init();
    GPIO_PORTF_DATA_R |= 0x02; // friendly, red
    while(1){
        GPIO_PORTF_DATA_R |= 0x08; // friendly, LED is yellow 0x0A
        GPIO_PORTF_DATA_R &= ~0x08; // friendly, LED is red 0x02
    }
}
int main(void){ Orange_Init();
    GPIO_PORTF_DATA_R = 0x02; // partial credit for not friendly, red
    while(1){
        GPIO_PORTF_DATA_R = 0x0A; // partial credit for not friendly,
        GPIO_PORTF_DATA_R = 0x02;
    }
}
```

**(15) Problem 6:** Assume the value of the Stack pointer (SP) is  $0x20000FFC$ . Consider, the following code sequence starts execution at label `CallR`. The initial stack contents are given on the right. When drawing the stack contents, you need only to show numbers on the stack that represents valid data.

<pre> CallE     PUSH {R4,R1}    &lt;--- A     LSR R4,R0,R1     SUB R1,R4     LSL R0,R1,R4     POP {R4,R1}     BX LR . . . CallR    &lt;--- Start here     POP {R1,R0}     MOV R4,#16     BL CallE     ADD R0,R4     SUB R1,#1     BL CallE . . .    &lt;--- B                 </pre>	<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><td style="padding: 2px;"><math>0x2000FF4</math></td><td style="padding: 2px;">64</td></tr> <tr><td style="padding: 2px;"><math>0x2000FF8</math></td><td style="padding: 2px;">6</td></tr> <tr><td style="padding: 2px;"><math>0x2000FFC</math></td><td style="padding: 2px;">32</td></tr> <tr><td style="padding: 2px;"><math>0x20001000</math></td><td style="padding: 2px;">5</td></tr> <tr><td style="padding: 2px;"><math>0x20001004</math></td><td style="padding: 2px;">16</td></tr> <tr><td style="padding: 2px;"><math>0x20001008</math></td><td style="padding: 2px;">4</td></tr> <tr><td style="padding: 2px;"><math>0x2000100C</math></td><td style="padding: 2px;">8</td></tr> </table>	$0x2000FF4$	64	$0x2000FF8$	6	$0x2000FFC$	32	$0x20001000$	5	$0x20001004$	16	$0x20001008$	4	$0x2000100C$	8
$0x2000FF4$	64														
$0x2000FF8$	6														
$0x2000FFC$	32														
$0x20001000$	5														
$0x20001004$	16														
$0x20001008$	4														
$0x2000100C$	8														

**(8) Part a)** Give the state of the stack (SP and contents) *after* the first time the PUSH instruction marked **A** is executed. Each box is 32 bits.

$0x2000FF4$	64
$0x2000FF8$	6
$0x2000FFC$	5
$0x20001000$	16
$0x20001004$	16
$0x20001008$	4
$0x2000100C$	8

SP:	$0x2000FFC$
-----	-------------

**(7) Part b)** Give the state of the stack (SP and contents) while executing the instruction marked **B**, and the values stored in R0, R1, and R4.

$0x2000FF4$	
$0x2000FF8$	
$0x2000FFC$	
$0x20001000$	
$0x20001004$	16
$0x20001008$	4
$0x2000100C$	8

SP:	$0x20001004$
R0:	6
R1:	4
R4:	16

(20) **Problem 7:** You will write two *assembly* functions. The function **Generate** is given a 31-bit value in R0 with bit 31 initially 0. If the number of bits that are set (set means the bit is a 1) in R0 is odd, then the function will set bit 31 to 1. If the number of bits that are set in R0 is even, then the function will NOT set bit 31. In this way, the return value in R0 contains the original 31 data bits, plus one more bit, such that the entire register now has an even number of bits that are 1. For example, if the input value is 0x0000E000 (there are an odd number of bits that are 1), it returns R0=0x8000E000. For example, if the input value is 0x6018C003 (there are an even number of bits that are 1), it returns R0=0x6018C003 unchanged. The second function **Check** is given a 32-bit input in R0, and returns a true (R0=1), if there are an even number of bits that are set in the input. **Check** returns a false (R0=0), if there are an odd number of bits that are set in the input. One function may call the other if you wish. Your functions must be AAPCS-compliant.

Generate	Check
<pre> PUSH {R4,LR} MOV R4,R0 ; copy BL Check CMP R0,#1 ;1 if even BEQ good ORR R4,R4,#0x80000000 good MOV R0,R4 POP {R4,LR} BX LR </pre>	<pre> MOV R1,#0 ;number of 1s MOV R2,#32 ;loop counter loop LSRS R0,R0,#1 BCC skip ADD R1,R1,#1 ;found a 1 skip SUBS R2,R2,#1 BNE loop AND R0,R1,#1 ;1 if odd EOR R0,R0,#1 ;1 if even BX LR </pre>
<pre> AlternateGenerate PUSH {R4,LR} MOV R4,R0 ; copy BL Check EOR R0,R0,#1 ;0 means even LSL R0,R0,#31 ORR R0,R0,R4 ;combine POP {R4,LR} BX LR </pre>	<pre> AlternateCheck MOV R1,#0 ;number of 1s loop LSRS R0,R0,#1 BCC skip ADD R1,R1,#1 ;found a 1 skip CMP R0,#0 ;quit when 0 BNE loop AND R0,R1,#1 ;1 if odd EOR R0,R0,#1 ;1 if even BX LR </pre>
	<pre> ExtraSpecialAlternateCheck MOV R1,#0 ;number of 1s loop LSRS R0,R0,#1 ADC R1,R1,#0 ;add carry skip BNE loop ;quit when 0 AND R0,R1,#1 ;1 if odd EOR R0,R0,#1 ;1 if even BX LR </pre>