Exam 1

Date: October 2 , 2025

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 the back of pages as we will not be scanning the back of your exam sheets.**
- You have **75+5** 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.

Problem 1	20	
Problem 2	15	
Problem 3	15	
Problem 4	8	
Problem 5	10	
Problem 6	16	
Problem 7	16	
Total	100	

(10) ***Problem 1a.*** [*Little-Endian and Sign-Extension*] Assume the contents of `R0` are `0x00004000`. Relevant 8-bit memory contents are shown below right. Assume `R1` has `1` and `R2` has `2` in it. What are the 32-bit contents (in hex) of each of the registers after the instructions below are executed:

```
LDR R3, [R0]
LDRSB R5, [R0, R1]
LDRSH R4, [R0, R2]
LDRB  R2, [R0]
```

0x00004000	0x56
	0x78
	0x9A
	0xBC

R0:
R1:
R2:
R3:
R4:
R5:

(10) ***Problem 1b.*** [*Stack: Push,Pop*] Assuming the SP is at `0x20207FF0`, after some arbitrary number of pushes and pops were done and the contents of the Stack are as shown below (left).

Before

0x20207FE8	0x55555555
0x20207FEC	0x88888888
0x20207FF0	0xFFFFFFFF
0x20207FF4	0x99999999
0x20207FF8	0xAAAAAAAA
0x20207FFC	0x22222222
0x20208000	0xEEEEEEEE

Operations

```
POP {R0-R3}
PUSH {R2, R1}
POP {R4}
MOVS R5,R3
PUSH {R0-R5}
```

After

0x20207FE8	
0x20207FEC	
0x20207FF0	
0x20207FF4	
0x20207FF8	
0x20207FFC	
0x20208000	

After the operations given above are performed in that order, what are the contents of the Stack (fill right), where is the SP (box it). What is in register R4?

R4 =

(15) ***Problem 2.*** Consider the following C code in the left column. The function **Sum2** computes the sum of all numbers that are powers of 2 between 1 and the input **N**. The last box shows a C call to this function. Convert the C code to assembly code in the right column, so that it implements the same functionality in each box (for example, the C local variables **sum** and **num** can simply be registers in assembly). AAPCS is in effect.

`uint32_t Result=0;` `uint32_t input;`	`.data` `.align 2`

`uint32_t Sum2(uint32_t N){`
`  uint32_t sum = 0;`
`  uint32_t num = 1;`
	`.text`
	`Sum2:`

`  while (num <= N){`
`  sum = sum +num;`
`  num = num<<1;`
`  }`	

`  return sum;`
 	
`}`
	`POP {PC}`

`// Assume input is set`
`// before the call below`
`Result = Sum2(input);`	

(15) ***Problem 3.*** This program takes as input a global variable **n** (unsigned 8-bit number in the range **[0, 7]**) and computes two values **L** and **U** (signed 8-bit numbers) that are based on **n**. The assembly code is shown on the left and your task is to write the equivalent in C.

(4) i. What are the values of **L** and **U** when **n** is 4?

(9) ii. Complete the missing lines of the C code to match the functionality of the assembly code. **Do not attempt a literal translation line by line, the comments give you a hint that multiple assembly lines make up one C line**.

```
    .data
L:   .space 1
U:   .space 1
    .text
n:   .byte 4
    .align 2
```

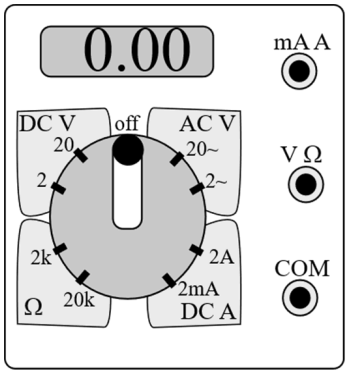
```
// Declarations
```

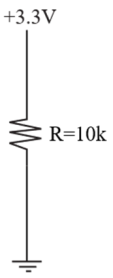
```
main:
     LDR  R0, =n
     LDRB R0, [R0]  //loads n
     SUBS R0, #1
     MOVS R1, #1
     LSLS R2, R1, R0
     SUBS R3, R2, #1 // computes U
     LDR  R4, =U
     STRB R3, [R4] // stores U
     MOVS R4, #0
     SUBS R2, R4, R2 // computes L
     LDR  R4, =L
     STRB R2, [R4]  // stores L
ForEver:
     B ForEver
```

```
// main function
int main(){




}
```

(2) iii. In 8 words or less describe what the program does for arbitrary how **L** and **U** are related to an arbitrary value of **n**.

(8) ***Problem 4.*** Show how to measure the current through a 10k resistor with a multimeter. The numbers on the meter specify the maximum value for that mode. A) Circle one of the numbers on the dial specifying the correct mode to use. B) Cut wire(s) with an X if needed. C) Add wires between the circuit and the multimeter to perform the measurement.





Do you have what it takes to be an ECE319K TA? Problems 5, and 6 are based on the following scenario:

Jane is a high school student taking an online class on Embedded systems. She is working on her Lab1 that requires her to interface two negative logic switches on PB3 and PB5 and a positive logic LED on PB4. **The operating logic of the code in assembly is to turn on the LED when either one of the two buttons is pressed but not both**. Jane is a very thorough student, she plans before she starts writing code. Specifically:

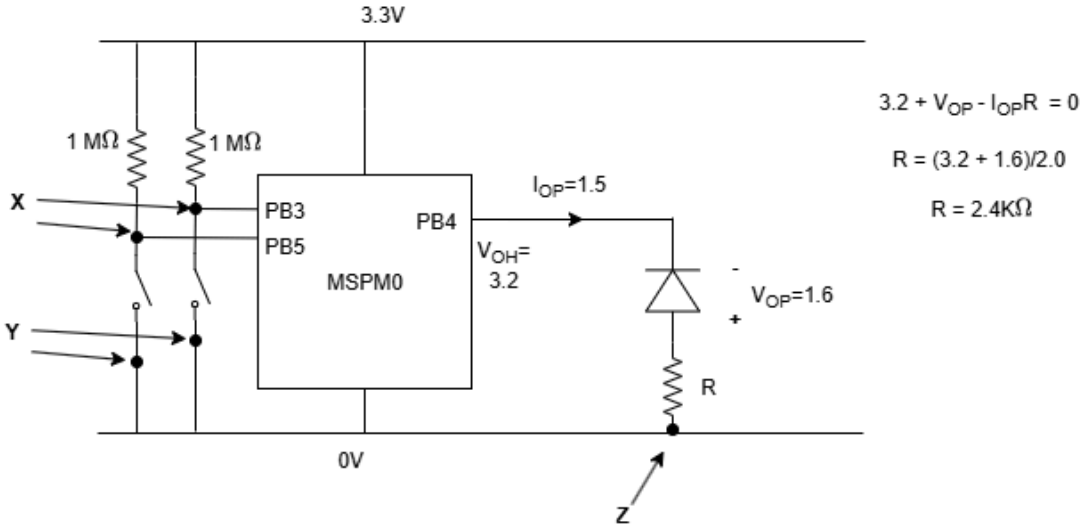
1. She draws a circuit diagram and chooses the right sized registers by looking at the data sheets for the microcontroller (MSPM0G3507), the LED and the switches.
2. She plans her software to be in parts:
    a. A subroutine (`Init`) that does the port initialization for the inputs and output
    b. The main program that calls the subroutine and then loops forever performing the intended operating logic.
    c. Before she writes the operating logic she makes a Truth-Table for how the output is determined by the inputs. This will guide her code.
3. She builds the circuit based on the circuit diagram, which she is very confident about and does not need help.

There can be errors in any of these steps. Your job is to spot them and help Jane finish the Lab. Given below is all her work:

<u>Circuit Diagram</u>

<u>Notes from Datasheets</u>:

Microcontroller: V_{OH} (Output high for a GPIO pin configured as output) = 3.2V
Microcontroller: V_{OL} (Output low for a GPIO pin configured as output) = 0.5V
LED Operating Point: (V_{OP}=1.6V, I_{OP}= 1.5mA)
Switch Resistance when Open: 1 MΩ



(10) ***Problem 5***. Circle (just the number) the mistakes that Jane made in drawing her circuit diagram from the list below:

Possible Mistake
1. The resistors used in the switches are in the wrong place - They should be below the switch. 2. The resistors used in the Switches are too large in value - They must be much lower than the 1 MΩ which is the Switch resistance when Open. 3. There is no need to connect the bottom side of the switch to the Ground. 4. The switch connections (marked **X**) are in the wrong place - They must be moved to **Y**. 5. The LED is incorrectly interfaced for a Positive Logic - The connection marked **Z** must be connected to the 3.3V rail and not the ground. 6. The LED is connected backwards - The negative end (cathode) must be connected to the resistor. 7. The calculation for resistor R is wrong - The correct equation is: $3.2 - V_{OP} - I_{OP} * R = 0$ 8. The calculation for resistor R is wrong - The correct equation is: $3.3 - V_{OP} - I_{OP} * R = 0.5$

Software Design:

Notes from Planning:

Negative Logic switches: Pressed: 0 ; Not Pressed: 1

Positive Logic LED: 1-> On ; 0->Off

Truth Table for operating logic:

PB3	PB5	PB4
0	0	0
0	1	1
1	0	1
1	1	1

Code

```
    .text
    .align 2
main:
    B PortInit  // Call PortInit Subroutine
    LDR R0, = GPIO_DIN31_0  // Device Reg for input pins PB3 and PB5
    LDR R1, = GPIO_DOUT31_0 // Device Reg for output pin PB4
    MOVS R2, #0x10 // Mask Bit for PB5
    MOVS R3, #0x04 // Mask Bit for PB3
Loop:
    LDR  R4, [R0]
    ANDS R5, R4, R2
    LSRS R5, R5, #5   // PB5 state
    ANDS R6, R4, R3
    LSRS R5, R5, #3   // PB3 state
    ORRS R5, R6
    LSLS R4, R5, #4   // To set LED state
    STR  R4, [R0]
    B Loop

// Setup - Port Initialization
PortInit:
    MOVS R0, #0x00040081
    LDR R1, =IOMUXPB3
    STR R0, [R1]    // PB3 Input: IENA(Bit18)<-1; PC(Bit7)<-1; Mode(Bits4:0)<-1
    LDR R1, =IOMUXPB5
    STR R0, [R1]    // PB5 Input: IENA(Bit18)<-1; PC(Bit7)<-1; Mode(Bits4:0)<-1
    MOVS R0, #0x81
    LDR R1, =IOMUXPB4 // Output is two steps IOMUX and GPIO_DOE31_0
    STR R0, [R1]    // PB4 Output: PC(Bit7)<-1; Mode(Bits4:0)<-1
    MOVS R0, #0x04   // Mask to identify PB4
    LDR R1 =GPIO_DOUT31_0
    LDR R2, [R1]
    ORRS R2, R0     // Friendly update to GPIO_DOE31_0 device register
    STR R2, [R1]
    POP {PC}
    .end
```

(**16**) ***Problem 6*** There are exactly 8 mistakes together in the Truth-table and Code. Circle them out of the list below:

1. The truth-table is wrong - the first row should have a 1 for PB4 and the last row should have 0 for PB4.
2. The truth-table is missing entries - It must have 8 entries, one for each combination of PB5 PB4 PB3 (000,001,010...)
3. The truth-table is wrong - the last row should have a 0 for PB4.
4. The Call to subroutine `PortInit` is wrong - `BL` must be used in place of the `B`
5. The `.end` pseudo-op is in the wrong place - It must be moved to line 18
6. The masks on lines 6 and 7 are wrong - `0x10` must be `0x20` and `0x04` must be `0x08`
7. The `MOVS` on line 21 is wrong - It must be an `LDR R0, =0x00040081`
8. The `ORRS` on line 14 is wrong - It must be an exclusive OR (`EORS`)
9. The immediate value `0x04` on line 29 is wrong - It must be `0x10` for PB4
10. There is a missing `PUSH {LR}` at the beginning of PortInit subroutine
11. The device register used on Line 30 is wrong - It must be `GPIO_DOE31_0`
12. The `LSLS` instruction of line 15 is wrong - It must be a `LSRS`
13. The immediate value 0x81 on line 26 is wrong - It must be `0x00040081`
14. The subroutine `PortInit` is not AAPCS compliant - It must save `R4-R7`
15. The subroutine `PortInit` is in the wrong place - It must be before the main

(16) ***Problem 7.*** Recall Lab1 where you were given an array of student records with two 32-bit attributes: first is a pointer (address) to an EID string and second is a Score. As part of your lab you had to write code for comparing two strings. Now, that code is available as a subroutine in C with the following prototype which you can call from the assembly function you will write:

```
uint8_t string_compare(char *string1, char *string2);
                                    // returns 1 if match, 0 otherwise.
```

Write a AAPCS compliant assembly subroutine called `Bonus`:

```
// Bonus  awards points to a specific student
// First Input:  Address of the array of records
// Second Input: Address of a student's EID  string
// Third Input:  Bonus points to increment that
//               student's score by
// Output: None
// Notes: No duplicate EIDs in the array
// Go through the student array (first input) and
//  find the student whose EID matches that in the
//  second Input and give them points (third input)
Bonus:
```