**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.

|  |  |  |
| --- | --- | --- |
| **Problem 1** | 10 |  |
| **Problem 2** | 15 |  |
| **Problem 3** | 10 |  |
| **Problem 4** | 20 |  |
| **Problem 5** | 10 |  |
| **Problem 6** | 15 |  |
| **Problem 7** | 20 |  |
| **Total** | 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\*input+25. Choose the best answer that describes this function. **1024\*input is 42 bit number, 1024\*input+input is a 43 bit number**

|  |  |  |  |
| --- | --- | --- | --- |
| Fun2: LSLS R1,R0,#10 ADDS R0,R0,R1 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/IB) P = V2 \*R**C) P = I2\*R**D) P = V/R2E) None of the above | **P=V\*I****V=I\*R so****P=I\*I\*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.

|  |  |
| --- | --- |
| uint16\_t root;uint32\_t sqr; |  **sqr: .space 4 // aligned****root: .space 2**  |
| const uint32\_t num=1000; |  **num: .long 1000** |
| void Func(){ root = 1; sqr = 1; | **Func:** **LDR R0,=root**  **MOVS R1,#1** **STRH R1,[R0]** **LDR R2,=sqr** **STR R1,[R2]** |
|  while (sqr <= num){ root++; sqr = root\*root; } |   **LDR R3,=num**  **LDR R3,[R3]****Loop:** **LDR R1,[R2]** **CMP R1,R3** **BHI Next** **LDR R1,[R0]** **ADDS R1,#1** **STRH R1,[R0]** **MULS R1,R1,R1** **STR R1,[R2]** **B Loop** |
|  root--;} | **Next:** **LDR R1,[R0]** **SUBS R1,#1** **STRH R1,[R0]** **BX LR** |

**(10) *Problem 3*.** Consider the following assembly function that performs input/output on Port B. Convert the assembly function to C code so that it implements the same functionality.

|  |  |
| --- | --- |
| OneOne: LDR R1,=GPIOB\_DIN31\_R LDR R0,[R1] MOVS R2,#0x0F ANDS R0,R0,R2 CMP R0,#0x01 BEQ One CMP R0,#0x02 BEQ One CMP R0,#0x04 BEQ One CMP R0,#0x08 BEQ One MOVS R0,#0x0  B DoneOne: MOVS R0,#10 Done: LDR R1,=GPIOB\_DOUT31\_0 STR R0,[R1] BX LR | **void OneOne(void){** **uint32\_t in;** **in = GPIOB->DIN31\_0&0x0F;** **if ((in == 0x01)||(in == 0x02)** **||(in==0x04)||(in == 0x08)) {**  **GPIOB->DOUT31\_0 = 0x10;** **}** **else{** **GPIOB->DOUT31\_0 = 0x00;** **}****}**  |

**(10) *Problem 4a***: Using only ONE 10 kΩ 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 VOL 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 CalculationsR = (3.3V-2.5V)/0.001A = 800mV/1mA = 800 ohms |
| Power CalculationsP = I2R = 0.001A\*0.001A\*800 = 0.0008W = 0.8 mW |

**(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.

|  |
| --- |
| int main(void){ **LaunchPad\_Init();** **GPIOB\_DOUTSET31\_0 = 1<<26; // friendly, red** **while(1){** **GPIOB\_DOUTTGL31\_0 = 1<<27; // toggle, friendly** **}** **}** |

**(15) *Problem 6***:Assume the value of the Stack pointer (SP) is 0x20200FFC.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CallE PUSH {R4,R1} **<--- A** LSRS R4,R0,R1  SUBS R1,R4 LSLS R0,R1,R4 POP {R4,R1} BX LR. . .CallR **<--- Start here** POP {R1,R0} MOVS R4,#16 BL CallE ADDS R0,R4 SUBS R1,#1 BL CallE . . . **<--- B** |

|  |  |  |
| --- | --- | --- |
|  | **0x20200FF4** | **64** |
|  | **0x20200FF8** | **6** |
| **SP-->** | **0x20200FFC** | **32** |
|  | **0x20201000** | **5** |
|  | **0x20201004** | **16** |
|  | **0x20201008** | **4** |
|  | **0x2020100C** | **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.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|

|  |  |
| --- | --- |
| **0x20200FF4** | **64** |
| **0x20200FF8** | **6** |
| **0x20200FFC** | **5** |
| **0x20201000** | **16** |
| **0x20201004** | **16** |
| **0x20201008** | **4** |
| **0x2020100C** | **8** |

 |

|  |  |
| --- | --- |
| SP: | **0x20200FFC** |

 |

(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.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|

|  |  |
| --- | --- |
| **0x20200FF4** |  |
| **0x20200FF8** |  |
| **0x20200FFC** |  |
| **0x20201000** |  |
| **0x20201004** | **16** |
| **0x20201008** | **4** |
| **0x2020100C** | **8** |

 |

|  |  |
| --- | --- |
| SP: | **0x20201004** |

|  |  |
| --- | --- |
| 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:**  **PUSH {R4,R5,LR}** **MOVS R4,R0 // copy** **BL Check** **CMP R0,#1 //1 if even** **BEQ good** **LDR R5,= 0x80000000** **ORRS R4,R4,R5****good: MOVS R0,R4** **POP {R4,R5,PC}****AlternateGenerate:**  **PUSH {R4,LR}** **MOV R4,R0 ; copy** **BL Check** **MOVS R3,#1** **EORS R0,R0,R3 //0 means even** **LSLS R0,R0,#31** **ORRS R0,R0,R4 //combine** **POP {R4,PC}** | **Check:** **MOVS R1,#0 //number of 1s** **MOV R2,#32 //loop counter****loop: LSRS R0,R0,#1** **BCC skip** **ADDS R1,R1,#1 //found a 1****skip: SUBS R2,R2,#1** **BNE loop** **MOVS R3,#1** **ANDS R0,R1,R3 //1 if odd** **EORS R0,R0,R31//1 if even** **BX LR****AlternateCheck:** **MOVS 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** **MOVS R3,#1** **ANDS R0,R1,R3 //1 if odd** **EORS R0,R0,R3 //1 if even** **BX LR** |