Name: _____

(10) Question 1)

(2) Part a) Assume the following memory contents, R0=0x00000102, and R1=0x20200000. What is in R2 after this one instruction executed? Show your answer in hexadecimal.

```
LDRH R2,[R0,R1]
```

0x20200100	0x80
0x20200101	0x81
0x20200102	0x82
0x20200103	0x83
0x20200104	0x84

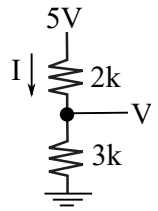
0x00008382

(2) Part b) Assume R1 has a signed 32-bit value, write assembly code that divides R1 by 16.

```
ASRS R1,R1,#4
```

(2) Part c) Write assembly code to swap the values of R2 and R5 using just PUSH and POP.

```
PUSH {R5}
PUSH {R2}
POP {R5}
POP {R2}
```



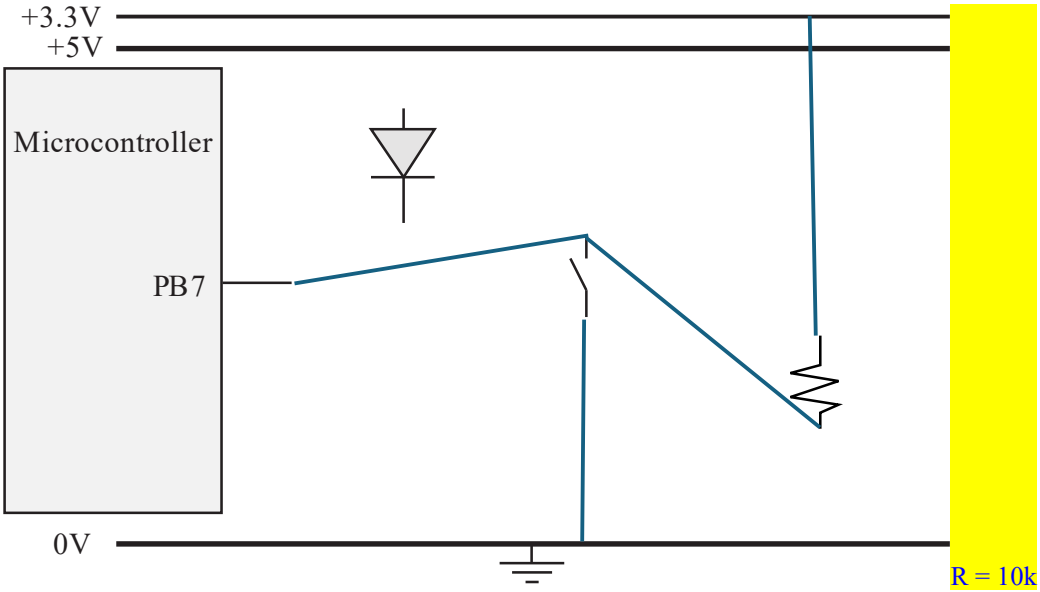
(2) Part d) What is V?

V=5*3k/(2k+3k) = 3volts

(2) Part e) What is I?

I = 5V/5k = 1mA

(12) Question 2. Interface a **switch** to PB7 using **negative logic**. Show all connections and resistor values.



(6) Question 3

(2) Part a) Assume **x, y, z** are integer variables, **x** is 3 and **y** is 2. What is **z** after this line is executed?

```
z = (x<<y)^10;
```

z = (3<<2)^10=12^10=
1100_2^1010_2 = 0110_2 = 6

(2) Part b) Assume you have a delay function that can wait any integer number of bus cycles. The period of the PWM wave must be 10,000 bus cycles. How many different duty cycles can you make?

```
while(1){
  LED_On();
  Wait(H);
  LED_Off();
  Wait(H);
}
```

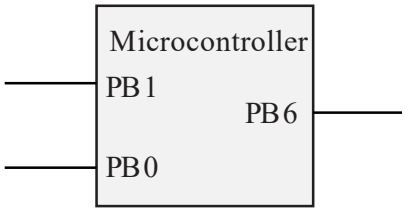
Any value from 9998 to 10001

(2) Part c) What is the range of the **uint16_t** data type in C? Give both the smallest and largest possible values.

Smallest = 0

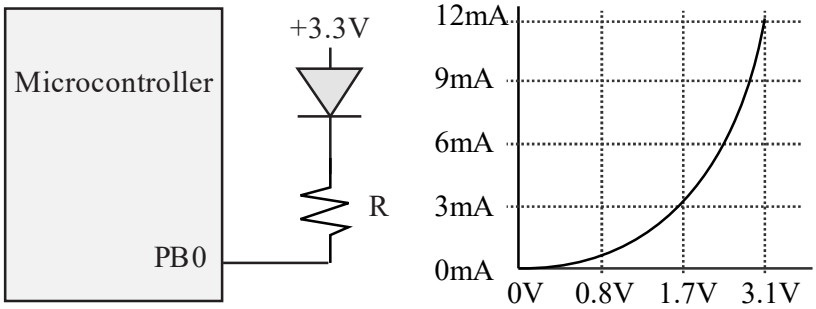
Largest = 2^{16}-1 or 65535

(15) Question 4. Write **assembly code** to do the following in a loop (i.e., run over and over without stopping). Read from two GPIO input pins (PB0 and PB1), and then write to one GPIO output pin (PB6). If the two input values are the same, output a ONE, else output a ZERO. The output to PB6 must be friendly. You do not know the configuration or status of the other bits on Port B. You can assume that Port B has been reset and powered on, the PINCM registers have been appropriately initialized, and that PB6 has been output enabled.




```
      LDR  R0,=GPIOB_DIN31_0
      LDR  R1,=GPIOB_DOUT31_0
      MOVS R2,#1    // bit0 mask
      MOVS R3,#2    // bit1 mask
      MOVS R4,#0x40 // bit6 mask
loop: LDR  R5,[R0]  // read input
      MOVS R6,R5
      ANDS R5,R5,R2 // isolate just bit 0
      LSLS R5,R5,#1 // bit 0 input positioned in bit1
      ANDS R6,R6,R3 // isolate just bit 1
      EORS R6,R6,R5 // bit 1 is 0 if equal
      EORS R6,R6,R3 // bit 1 is 1 if equal
      LSLS R6,R6,#5 // new value in bit 6 position
      LDR  R7,[R1]  // previous DOUT
      BICS R7,R4    // remove bit 6
      ORRS R7,R7,R6 // combine old and new
      STR  R7,[R1]  // output combination
      B    loop
```

(12) Question 5. Consider the LED circuit below interfaced with PB0.



(2) Part a) Is this LED interface positive or negative logic?

Negative logic

(3) Part b) Suppose the pin output is such that the LED is on. We want to choose a value for R. In no more than 5 words each, explain what would happen for the values of R below

.

1. R = 100k ohms — LED is off (or very dim)
2. R = 500 ohms — LED is on (correct brightness)
3. R = 1 ohm — LED is very bright; microcontroller is damaged; LED is damaged

(5) Part c) Suppose the desired operating point of the diode is 1.7V, 3mA. The high voltage of the microcontroller is 3.1V, and the low voltage of the microcontroller is 0.1V. Derive an equation and solve for the correct R?

R = (3.3-1.7-0.1)/3mA = 1500mA/3mA = 500 ohms

(2) Part d) If you choose a resistor with a value twice as large than the one calculated in c) will the diode be brighter or dimmer?

LED will be dimmer.

(15) Question 6) Consider the following code. What does the code do, in seven words or less?

```
uint32_t mystery(uint8_t a, uint8_t b){
  uint32_t result = 0;
  uint32_t msb = 1 << 7;
  for (int i = 0; i < 8; ++i){
    result = result << 1;
    if (a & msb) {
      result = result + b;
    }
    a = a << 1;
  }
  return(result);
}
```

Multiplies a times b.

Below is a direct translation of the mystery function, with assembly instructions and operands missing (indicated by boxes). Fill in the missing assembly instructions and operands. Your completed assembly code should produce exactly the same results as the C version. You are only allowed to fill in the blanks; you cannot add any additional instructions or change any of the instructions or arguments. It follows AAPCS.

```
Mystery:  PUSH {R4-R7,LR}
L1:   MOVS R4, #0
L2:   LDR  R2, =1<<7
L3:   MOVS R3, [#0]
L4:   CMP  R3, [#8]
L5:   [BGE]  L14
L6:   ADDS R4, [R4] , [R4]
L7:   MOVS R5, R2
L8:   ANDS [R5] , [R5] , R0
L9:   BEQ  [L11]
L10:  ADDS R4, R4, R1
L11:  [LSLS]  R0, #1
L12:  ADDS R3, #1
L13:  B    [L4]
L14:  [MOVS]  R0, R4
L15:  POP  {R4-R7,PC}
```

(8) Question 7) You are given a C function that outputs to Port A

```
void Output(uint32_t data){
  GPIOA->DOUT31_0 = data;
}
```

You write a subroutine called **MyAssemblyFunction** that calls **Output** with the **data** parameter equal to 5. Follow AAPCS.

```
 .global Output
MyAssemblyFunction:
    PUSH {LR}
    MOVS R0,#5
    BL   Output
    POP  {PC}
```

(7) Question 8: Assume the following register values:

R0	0
R1	1
R2	2
R3	3
R13(SP)	0x20201000
R14(LR)	0x000001FF

Draw the stack after these two instructions are executed.

```
PUSH {R3}
PUSH {LR,R1}
```

1	0x20200FF4
0x000001FF	0x20200FF8
3	0x20200FFC
	0x20201000
	0x20201004
	0x20201008
	0x2020100C

What is the SP after these two instructions are executed?

0x20200FF4

(15) Question 9. A variable-length character string is allocated 10 spaces and defined as follows

```
        .data
String: .space 10
```

You may not add any additional global variables. The string should be null terminated. I.e., there will be a 0x00 at the end of the string. Therefore, there is space for 9 characters. You may assume all bytes of the string are initialized to 0x00 once at the start of the system, Therefore, the string is initially empty. Write **an assembly language** function called **Append**, which appends one 8-bit character to the end of the string each time **Append** is called. The 8-bit value to store is passed in the lower 8-bits of Register R0. If the string already contains 9 characters, any call to **Append** will not store. Furthermore, if the data is 0x00, do not append. I.e., only save nonzero characters will be saved. Follow AAPCS. The following shows what happens if your function is called twice

```
  Append('h'); // String is "h"
  Append('i'); // String is "hi"
```

```
Append:
      CMP  R0,#0
      BEQ  skip        // do not store null
      LDR  R1,=String
      MOVS R2,#0       // index/length
loop: LDRB R3,[R1,R2] // find end
      CMP  R3,#0       // null?
      BEQ  done        // R2 current length
      ADDS R2,#1
      B    loop
done: CMP  R2,#9
      BEQ  skip        // full, do not store
      STRB R0,[R1,R2] // save character
skip: BX   LR
```