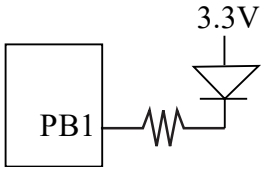
UT EID: _____ First: _____ Last: _____

Instructions:

- Closed book and closed notes. No books, no papers, no data sheets (other than the addendum)
- 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. *Anything outside the boxes/blanks will be ignored in grading*. You may use the back of the sheets for scratch work.
- You have 120 minutes, so allocate your time accordingly.
- For all questions, unless otherwise stated, find the most efficient (time, resources) solution.
- Unless otherwise stated, make all I/O accesses friendly and all subroutines AAPCS compliant.
- *Please read the entire exam before starting.*

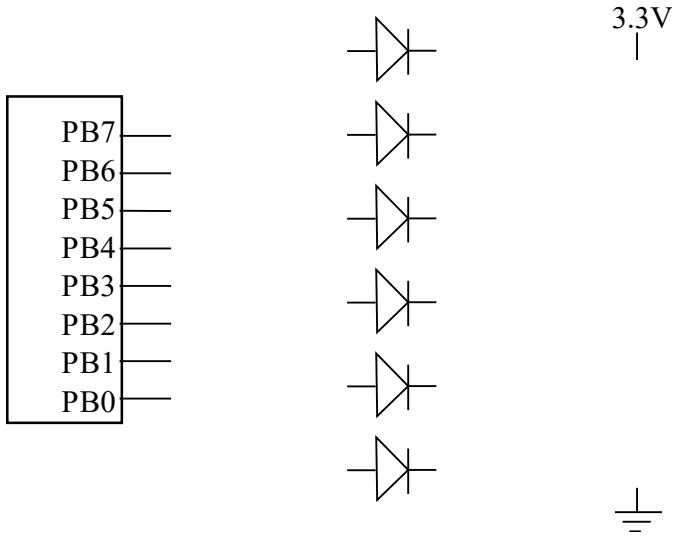
(5) Question 1) SysTick interrupts are armed and running. PB1 is initialized as an output.

```
void SysTick_Handler(void){
  GPIOB->DOUT31_0 = GPIOB->DOUT31_0^0x02;
  if(GPIOB->DOUT31_0 & 0x02){
    SysTick->LOAD = 10000;
  }else{
    SysTick->LOAD = 30000;
  }
  SysTick->VAL = 0; // causes SysTick to reload
}
```

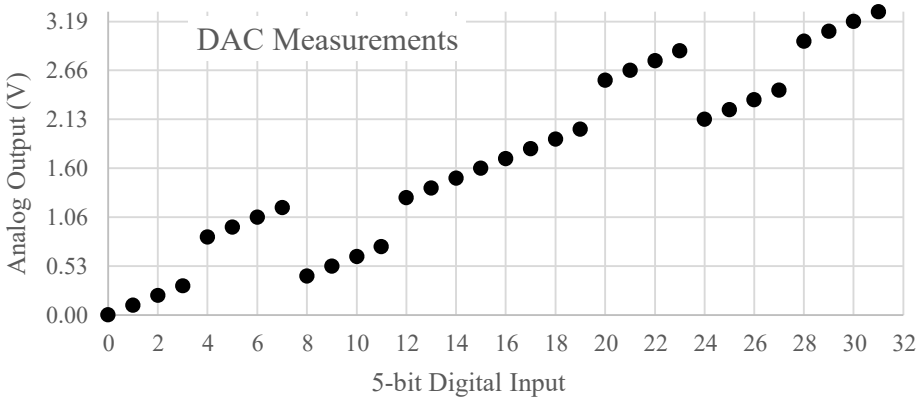


The resistor activates the LED at 100% brightness. What will be the approximate brightness of the LED? Give your answer as a percentage from 0% (off) to 100% (full).

(5) Question 2) You are given six identical LEDs. Each LED parameter is $I_d = 2.5$mA, $V_d = 1.0$V. Interface all LEDs to PB5 **without any resistors**. If PB5 is high, then all six LEDs should be on. If PB5 is low, then all six LEDs should be off. V_{OL} is 0.5V and V_{OH} is 3.0V.

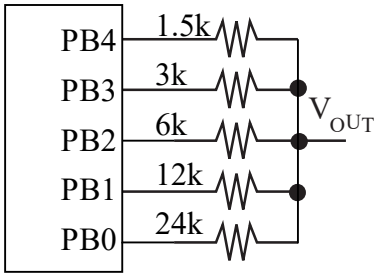


(5) Question 3) A student built a 5-bit binary weighted DAC. The circuit shows how the resistors were supposed to be connected. The student collected this calibration data. Two resistors have been swapped. Which two resistors were swapped, creating this result?



Enter your answer A-E into the box.

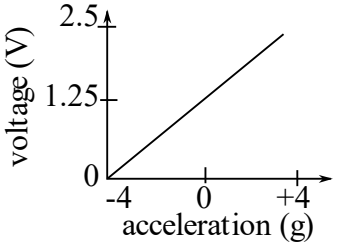
A) Swapped 12k, 24k
B) Swapped 6k, 12k
C) Swapped 3k, 6k
D) Swapped 1.5k, 3k
E) None of the above



(10) Problem 4. An analog accelerometer is connected to ADC1 channel 0. The 12-bit ADC has been initialized such that ADC1 channel 0 is configured for software triggered conversion into **MEMCTL[0]** using the 2.5V ADC reference. If the analog input is 0V, the **ADC1_In** function returns 0. If the analog input is 2.5V, the **ADC1_In** function returns 4095.

```
int32_t ADC1_In(void){
  ADC1->ULLMEM.CTL0 |= 0x00000001; // enable conversions
  ADC1->ULLMEM.CTL1 |= 0x00000100; // start ADC
  uint32_t volatile delay=ADC1->ULLMEM.STATUS; // time to let ADC start
  while((ADC1->ULLMEM.STATUS&0x01)==0x01){}; // wait for completion
  return (int32_t)ADC1->ULLMEM.MEMRES[0]; // 0 to 4095
}
```

The analog accelerometer has an input range of -4g, +4g, where g is earth's gravity. The output voltage of the accelerometer is a voltage from 0V to 2.5V, linearly related to acceleration, where -4g is 0V, and +4g is 2.5V.



TimerG12 has been configured to run periodically at 100Hz. Write C code for the TimerG12 interrupt handler. The handler must sample the ADC, sum the current ADC sample and the previous interrupt's ADC sample (forming a 13-bit sum). You should then convert the 13-bit sum to acceleration with a fixed-point resolution of 2^{-10}g storing the result in **accel**. You cannot use divide operations.

```
int32_t accel; // integer portion, resolution is (1/1024)g
void TIMG12_IRQHandler(void){
  static int32_t prev = 0; // previous ADC, 0 to 4095









}
```

(5) Problem 5. If the device from Question 4) is accelerated at 2g, answer the five questions.
Part a) What is the voltage output of the device in volts?

Part b) What is the ADC value from calling **ADC1_In**?

Part c) What is the fixed-point gravity value in the variable **accel**?

Part d) According to the **Nyquist Theorem** what are the minimum and maximum frequencies that can be reliably captured given the system defined in Question 4?

Part e) According to the **Valvano Postulate** what is the maximum frequency that can be reconstructed and still look like the original signal when plotted?

(15) Question 6) This system is written entirely in assembly and **DOES NOT** comply with AAPCS. Show the assembly implementation of **Divideby32** that modifies the 16-bit signed global in place by effectively dividing the global by 32. Do not round the result. E.g., 63/32 is 1.

```
    .data                    .text                    .text
    .align 2                 LDR  R7,=I               LDR  R6,=J
 I: .space 2 // signed       PUSH {R0,R7}             PUSH {R0,R6}
 J: .space 2 // signed       MOVS R7,#0               MOVS R6,#0
                             BL   Divideby32          BL   Divideby32
                           // I=I/32                 // J=J/32
                             ADD  SP,#8               ADD  SP,#8
```

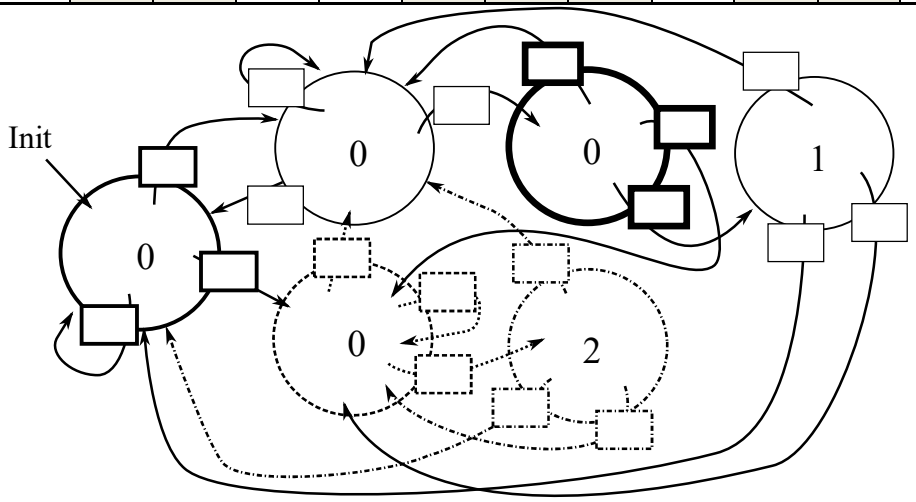
The one main program invokes **Divideby32** for **I**, and a second main invokes it for **J**. Notice the main programs balance the stack; **Divideby32** must also balance the stack. **Divideby32** has no input or output parameters in registers; rather, **Divideby32** must use the parameter passed on the stack. *Hint*: draw a stack picture and look at what is being passed.

```
Divideby32:
```

(10) Question 7) The input can be 0, 1 or 2. There are no time delays. The initial output is 0. If the input sequence is 0,1,2 the output is 1. However, if the input sequence is 1,2 the output is 2. For all other sequences the output is 0. Add input numbers for this FSM STG. For example,

Input	2	2	1	2	2	1	0	1	2	1	1	2	2
Output	0	0	0	2	0	0	0	0	1	0	0	2	0

Fill in the 18 little boxes with 0, 1, or 2

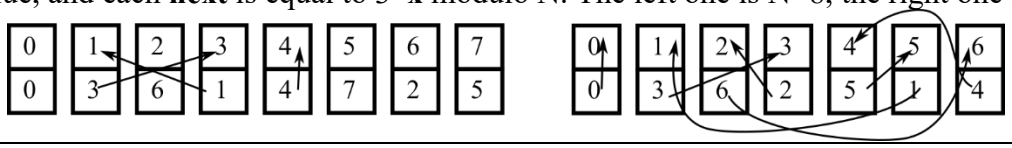


(15) Problem 8. Write functions to create and process a ring of nodes:
(3) Part a) Define a struct with two fields: An unsigned 8-bit integer **x**, and an unsigned 8-bit integer **next**, which contains the index of another node

```
#define N 8
```

(2) Part b) Create an array **Nodes** of **N node_t** structs. **N** can change if recompiled.

(5) Part c) Write code to initialize the **Nodes** array, such that the value **x** of each node equals its index value, and each **next** is equal to 3***x** modulo **N**. The left one is **N**=8, the right one is **N**=7.




```
void InitializeRings(void){
```

(5) Part d) Write code to trace a ring of nodes starting with the node whose **x** value is **start**, and then moving to the node whose **x** value is given by next. Do this repeatedly until you return to **start**. Your function should return the length of this ring. For example, if **N** = 8, and **start** = 1, length = 2 (1->3->1), If **N**=7, **start** = 4, length = 6 (see right figure)

```
uint8_t TraceRing(uint8_t start){
```

(15) Question 9. If you are in 319K solve the C problem on the left, if you are in 319H solve the C++ problem on the right. Below is code to implement a FIFO. **The code must not waste any locations to differentiate between full and empty**. Fill in the blanks in the code. You are not allowed to change any variable definitions.

```
static int *data; // ECE319K
static int head,tail,x,size;

// pt is pointer to an array
// s is size of array
int Init(int *pt, int s){
  size = s;
  data = pt;
  tail = head = 0;

  [          ] = [          ] ;

  return(1); // success
}

int Put(int value){

  if( [          ] < [          ] ){
    data[tail] = value;
    tail = (tail+1)%size;
    [                    ];
    return(1); // success
  } else {
    return(0); // fail
  }
}
int Get(void){

  if( [          ] == [          ] ){
    return(-1);
  } else {
    int ret = data[head];
    head = (head+1)%size;
    [                    ];
    return(ret);
}}
```

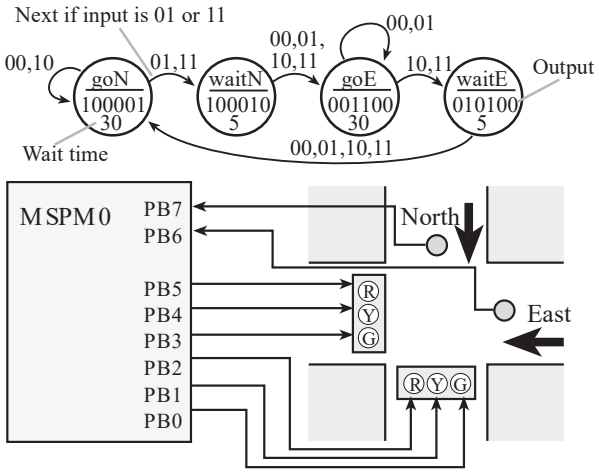
```
class fifo_t {  // ECE319H
 private:
  int *data;
  int head,tail,x,size;
 public:
  bool Init(int size);
  bool Put(int value);
  int  Get();
};
bool fifo_t::Init(int s){
  size = s;
  data=(int *)malloc(4*size);
  if(data==NULL)return(false);
  tail = head = 0;
  [          ] = [          ] ;
  return(true); // success
}
bool fifo_t::Put(int value){

  if( [          ] < [          ] {
    data[tail] = value;
    tail = (tail+1)%size;
    [                    ] ;
    return(true); // success
  } else {
    return(false); // fail
  }
}
int fifo_t::Get(){

  if( [          ] == [          ] {
    return(-1);
  } else {
    int ret = data[head];
    head = (head+1)%size;
    [                    ];
    return(ret);
}}
```

(15) Question 10) You are given a **Delay** function written in C, which follows AAPCS. It has this prototype **`void Delay(uint32_t sec)`**. The parameter **sec** is the number of seconds to wait. Complete the **main** program that runs this FSM. You need not write **Delay** or **Traffic_Init**. The initial state is **goN**. Your output must be friendly.

```
 .global Delay
goN:    .long 0x21
        .long 30  //30 sec
        .long goN,waitN,goN,waitN
waitN:  .long 0x22
        .long 5   //5 sec
        .long goE,goE,goE,goE
goE:    .long 0x0C
        .long 30  //30 sec
        .long goE,goE,waitE,waitE
waitE:  .long 0x14
        .long 5   //5 sec
        .long goN,goN,goN,goN
```



Registers R0, R2, R3, and R7 are available.

Part a) Which register should you use for **Rx**? [ ]

Part b) Which register should you use for **Ry**? [ ]

Parts c-j) Fill in the eight boxes

```
main: BL     Traffic_Init
      LDR    Ry,=goN
      LDR    R4,=GPIOB_DOUT31_0
      LDR    R5,=GPIOB_DIN31_0
loop:
   LDR  Rx,[Ry,  ]
   LDR  R1,[R4]
   MOVS R6,#0x3F
   
   R1,R1,R6
   
   Rx,Rx,R1
   
   STR  Rx,[R4]
   LDR  Rx,[Ry,  ]
   BL   Delay
```

```
LDR  Rx,[R5]
MOVS R1,#0xC0

   Rx,Rx,R1

LSRS Rx,Rx,#

ADDS Rx,Rx,#

LDR  Ry,[  ]
B    loop
```