

(14) Question 1. For each of the following components, decide where to place it within the memory map of the microcontroller. Multiple choice select: **RAM**, **ROM**, or **other**. Select other if the component is neither in RAM or ROM. There is one correct answer for each component.

(2) Part a) Registers (e.g., R0, R1, ... themselves) other
 Registers do not exist at any address in the memory map

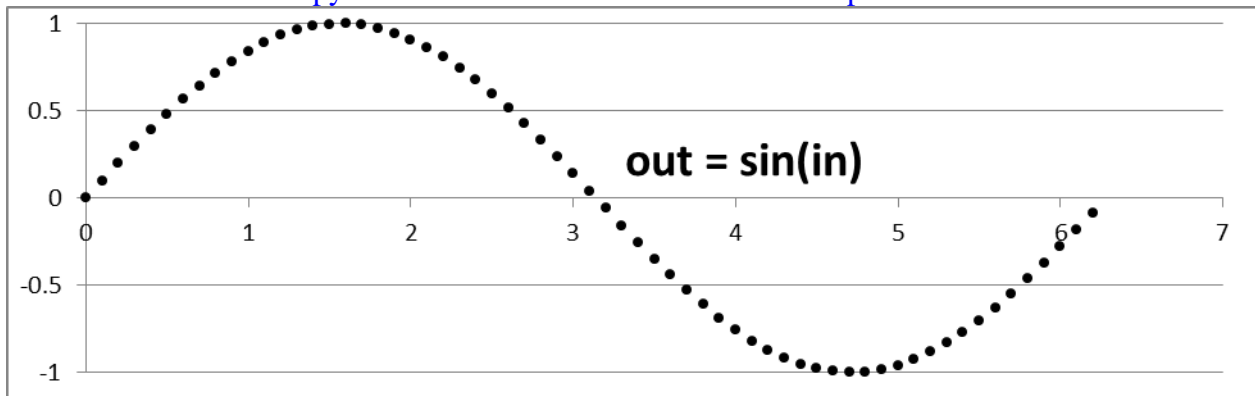
(2) Part b) Global variables RAM
 RAM allows for read/write access

(2) Part c) Reset vector (value to which the PC is initialized on power up) ROM
 ROM allows for nonvolatile storage

(2) Part d) The stack RAM
 RAM allows for read/write access

(2) Part e) Port E direction register (the register itself) other
 I/O registers have addresses, but are neither RAM nor ROM

(4) Part f) A table representing all possible outputs of a sine wave as a function of all possible inputs as illustrated in the following figure ROM
 ROM allows for large amounts of constant storage. If you were to put it in RAM, you would need a second copy in RAM to initialize the table at start up.



(8) **Question 2a.** Assume initially R4=4, R5=5, R6=6, and R7=7.

Consider the following assembly code

```

PUSH {R5,R6}
    Top-> 5,6 <-Bottom
ASR R4,R4,R7
    R4 = 4/128 = 0
EOR R7,R5,R6
    R7 = 1012 ^ 1102 = 0112 = 3
PUSH {R7,R4}
    Top-> 0,3,5,6 <-Bottom
POP {R7,R5,R4,R6}
    R4=0,R5=3,R6=5,R7=6 (stack empty)

```

R4= 0

R5= 3

R6= 5

R7= 6

After executing these five instructions, what will be the final values for these four registers?

(5) **Question 2b.** Consider this one stack instruction. Neglect AAPCS for this one question

```
POP {R0}
```

Consider the following actions

- | | |
|--|--|
| A) Read 8 bits from memory at SP into R0 | B) Read 16 bits from memory at SP into R0 |
| C) Read 32 bits from memory at SP into R0 | D) Write 8 bits from R0 into memory at SP |
| E) Write 16 bits from R0 into memory at SP | F) Write 32 bits from R0 into memory at SP |
| G) SP = SP+1 | H) SP = SP+2 |
| I) SP = SP+4 | J) SP = SP+8 |
| K) SP = SP-1 | L) SP = SP-2 |
| M) SP = SP-4 | N) SP = SP-8 |

Give the sequence of operations that occur when the **POP {R0}** instruction is executed on the Cortex M. The machine code for the instruction requires 16 bits or 2 bytes. Give your answer as an ordered sequence of the letters from A to N.

C,I
32-bit Read from stack,
Deallocate space, add 4

(3) **Question 2c.** Consider how the following function uses the stack

```

Fun PUSH {R4,R14}
;
; a bunch of assembly instructions that do not affect the stack
;
    POP {R4,R13}

```

What happens when the POP instruction is executed? Specify the one letter that best describes the effect of this POP instruction.

R14 is the link register, R13 is the stack pointer

- A) The values of R4 and R14 are swapped
 B) This function will not assemble because one cannot pop into the R13
 C) A hard fault will occur at run time because the R13 will be pointing into ROM
 D) Execution is returned to program that called the subroutine (a normal function return)
 E) The program will jump to an unknown location within the software
 F) None of the above

B
(I gave credit
for C also)

(15) Question 3. Consider the following two assembly functions, **Out** and **Func**. You may assume Port B is already initialized as an 8-bit output port.

```

Out  ORR R0,R0,#0x80    ; R0 is input called data
     LDR R1,=GPIO_PORTB_DATA_R
     STR R0,[R1]        ; output to Port B
     BX  LR              ; no return value
Func  PUSH {R4,R5,R6,LR} ; R0 is input parameter called count
     MOV R4,R0          ; R4 contains the parameter count
     MOV R5,#0
loop  CMP R5,R4
     BHS done
     MOV R0,R5          ; set data parameter for function call to Out
     BL  Out
     ADD R5,R5,#1
     B   loop
done  MOV R0,#1         ; return true=success
     POP {R4,R5,R6,LR}
     BX  LR             ; there is a return parameter=1

```

Rewrite both **Out** and **Func** functions in C (do not combine into one function)

```

#define GPIO_PORTB_DATA_R (*((volatile uint32_t *)0x400053FC)) // data register
void Out(uint32_t data){
    GPIO_PORTB_DATA_R = data|0x80; // output to Port B
}
int Func(uint32_t count){ // count is unsigned
    uint32_t i;           // R5 is local variable
    for(i=0; i<count; i++){
        Out(i);
    }
    return 1;             // success
}
// return type for Func can be any integer type

int Func(uint32_t count){ // count is unsigned
    uint32_t i=0;         // R5 is local variable
    while(i < count){
        Out(i);
        i++;
    }
    return 1;             // success
}

```

(10) Question 5. Write the C function to initialize Port B so that PB2 is an output and PB6 is an input. Make your solution friendly. You will need to access some or all of these registers.

```
#define GPIO_PORTB_DATA_R (*((volatile uint32_t *)0x400053FC)) // data register
#define GPIO_PORTB_DIR_R  (*((volatile uint32_t *)0x40005400)) // direction register
#define GPIO_PORTB_DEN_R  (*((volatile uint32_t *)0x4000551C)) // enable register
#define SYSCTL_RCGCGPIO_R (*((volatile uint32_t *)0x400FE608)) // clock register
```

```
void Init(void){
int volatile delay;
  SYSCTL_RCGCGPIO_R |= 0x02; // activate Port B
  delay = 100;
  GPIO_PORTB_DIR_R |= 0x04; // PB2 is output
  GPIO_PORTB_DIR_R &= ~0x40; // PB6 is input
  GPIO_PORTB_DEN_R |= 0x44; // enable both PB2 and PB6
}
```

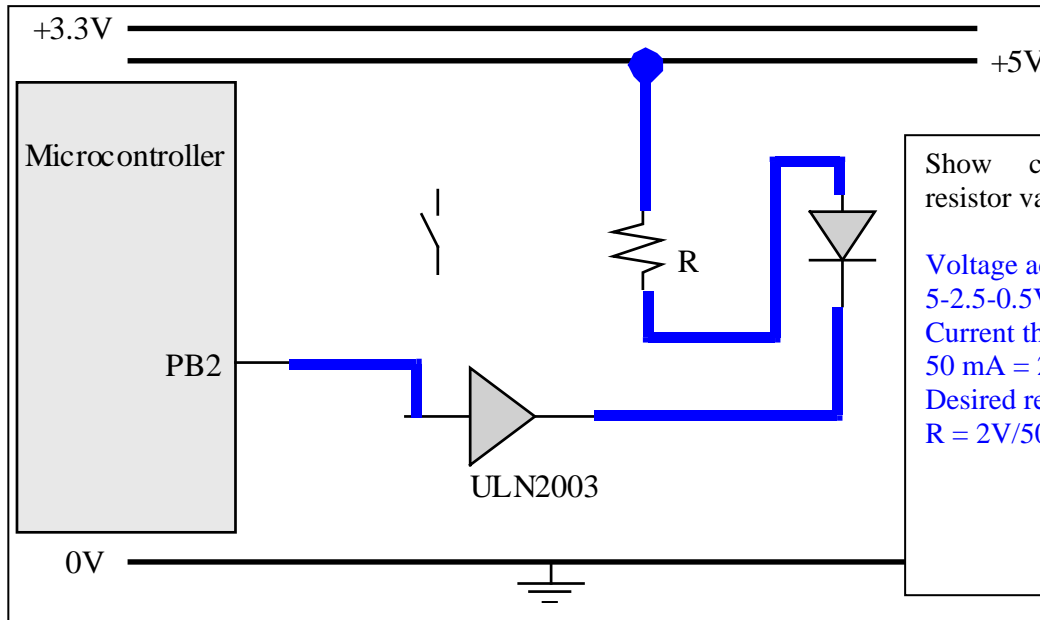
```
void Init(void){
  SYSCTL_RCGCGPIO_R |= 0x02; // activate Port B
  while((SYSCTL_PRGPIO_R&0x02)==0){};
  GPIO_PORTB_DIR_R |= 0x04; // PB2 is output
  GPIO_PORTB_DIR_R &= ~0x40; // PB6 is input
  GPIO_PORTB_DEN_R |= 0x44; // enable both PB2 and PB6
}
```

```
void Init(void){
int volatile delay;
  SYSCTL_RCGCGPIO_R |= 0x02; // activate Port B
  delay = 100;
  GPIO_PORTB_DIR_R = (GPIO_PORTB_DIR_R&~0x40)|0x04;
  // PB2 is output, PB6 is input
  GPIO_PORTB_DEN_R |= 0x44; // enable both PB2 and PB6
}
```

```
// note that ~0x40 = 0xFFFFFFFF
```

(20) Question 4. Assume the microcontroller’s output voltage high is 3.3V. Assume the microcontroller’s output voltage low is 0V. The V_{OL} for the ULN2003 driver is 0.5V. Pick resistor(s) appropriately and assume you have 5V, 3.3V, and ground to which you can connect your components. The symbols for each part are given below for your convenience – *use the minimum number of parts to construct each interface.*

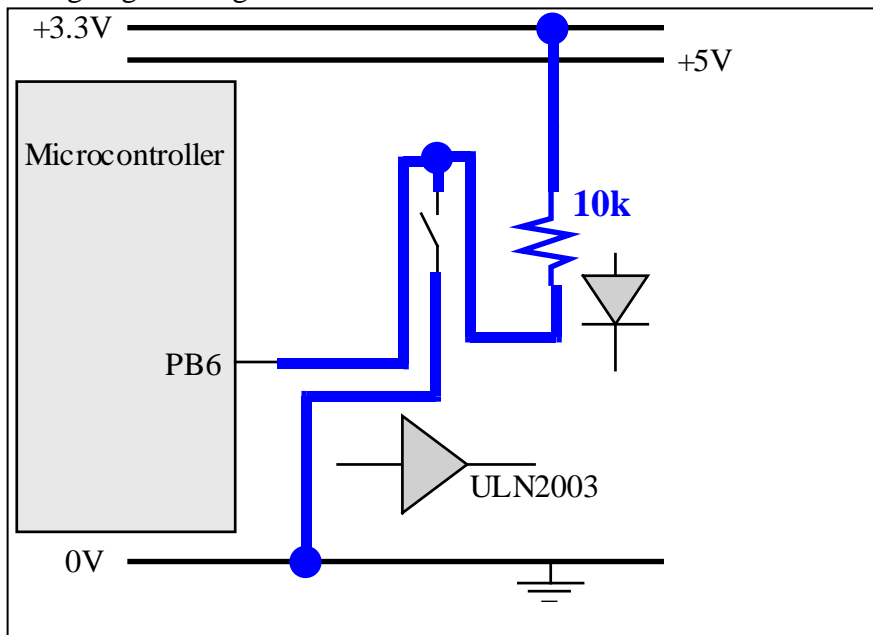
Part a) Interface the LED to Port B bit 2 using positive logic. The LED operating point is 2.5V at 50mA. Show the circuit and calculate value(s) for any resistor(s) needed.



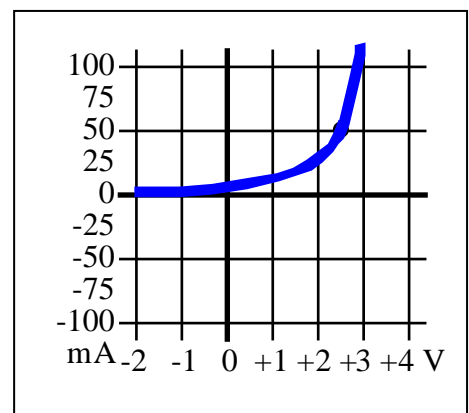
Show calculations for selecting resistor value(s)

Voltage across resistor
 $5 - 2.5 - 0.5V = 2V$
 Current through resistor
 $50\text{ mA} = 2V/R$
 Desired resistance
 $R = 2V/50\text{mA} = 40\Omega$

Part b) Interface this switch to the microcontroller Port B bit 6 using negative logic. Include a 10 kΩ external resistor.



Part c) Sketch the approximate current versus voltage response of the LED used in part a). The curve goes through 50mA at 2.5V.



Exponential shape
 Through the point
 No current for negative voltage

(25) **Question 6.** In question you will write software to control an insulin pump. You may assume the `PortB_Init` function initializes PB5-PB0 as inputs, and initializes PB7-PB6 as outputs. PB6 is an alarm output and PB7 activates the pump. You do not write `PortB_Init`. Write the assembly code that performs both these steps over and over indefinitely:

If PB5 equals 0, then set both PB6 and PB7 low;

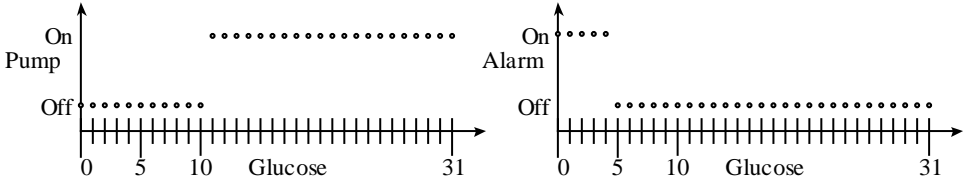
If PB5 equals 1, the device is active and you should perform these steps

Read PB4-PB0, the glucose level is unsigned 5-bit number from 0 to 31

If the glucose level is below 5, turn on PB6 alarm (otherwise turn PB6 off)

If the glucose level is above 10, turn on PB7 pump (otherwise turn PB7 off)

`GPIO_PORTB_DATA_R EQU 0x400053FC`



```

start
    BL PortB_Init

loop
    LDR R0,= GPIO_PORTB_DATA_R
    LDR R1,[R0]
    ANDS R2,R1,#0x20 ; check PB5
    BEQ off
    AND R1,R1,#0x1F ;glucose from 0 to 31
    CMP R1,#5
    BLO alarm ;unsigned branch
    CMP R1,#10
    BHI on ;unsigned branch
; pump off, alarm off
off MOV R1,#0 ;PB6=0, PB7=0
    B set
;pump off, alarm on
alarm MOV R1,#0x40 ;PB6=1, PB7=0
    B set
;pump on, alarm off
on MOV R1,#0x80 ;PB6=0, PB7=1
set STR R1,[R0]
    B loop

```

Memory access instructions

LDR Rd, [Rn] ; load 32-bit number at [Rn] to Rd
LDR Rd, [Rn,#off] ; load 32-bit number at [Rn+off] to Rd
LDR Rd, =value ; set Rd equal to any 32-bit value (PC rel)
LDRH Rd, [Rn] ; load unsigned 16-bit at [Rn] to Rd
LDRH Rd, [Rn,#off] ; load unsigned 16-bit at [Rn+off] to Rd
LDRSH Rd, [Rn] ; load signed 16-bit at [Rn] to Rd
LDRSH Rd, [Rn,#off] ; load signed 16-bit at [Rn+off] to Rd
LDRB Rd, [Rn] ; load unsigned 8-bit at [Rn] to Rd
LDRB Rd, [Rn,#off] ; load unsigned 8-bit at [Rn+off] to Rd
LDRSB Rd, [Rn] ; load signed 8-bit at [Rn] to Rd
LDRSB Rd, [Rn,#off] ; load signed 8-bit at [Rn+off] to Rd
STR Rt, [Rn] ; store 32-bit Rt to [Rn]
STR Rt, [Rn,#off] ; store 32-bit Rt to [Rn+off]
STRH Rt, [Rn] ; store least sig. 16-bit Rt to [Rn]
STRH Rt, [Rn,#off] ; store least sig. 16-bit Rt to [Rn+off]
STRB Rt, [Rn] ; store least sig. 8-bit Rt to [Rn]
STRB Rt, [Rn,#off] ; store least sig. 8-bit Rt to [Rn+off]
PUSH {Rt} ; push 32-bit Rt onto stack
POP {Rd} ; pop 32-bit number from stack into Rd
ADR Rd, label ; set Rd equal to the address at label
MOV{S} Rd, <op2> ; set Rd equal to op2
MOV Rd, #im16 ; set Rd equal to im16, im16 is 0 to 65535
MVN{S} Rd, <op2> ; set Rd equal to -op2

Branch instructions

B label ; branch to label Always
BEQ label ; branch if Z == 1 Equal
BNE label ; branch if Z == 0 Not equal
BCS label ; branch if C == 1 Higher or same, unsigned \geq
BHS label ; branch if C == 1 Higher or same, unsigned \geq
BCC label ; branch if C == 0 Lower, unsigned $<$
BLO label ; branch if C == 0 Lower, unsigned $<$
BMI label ; branch if N == 1 Negative
BPL label ; branch if N == 0 Positive or zero
BVS label ; branch if V == 1 Overflow
BVC label ; branch if V == 0 No overflow
BHI label ; branch if C==1 and Z==0 Higher, unsigned $>$
BLS label ; branch if C==0 or Z==1 Lower or same, unsigned \leq
BGE label ; branch if N == V Greater than or equal, signed \geq
BLT label ; branch if N != V Less than, signed $<$
BGT label ; branch if Z==0 and N==V Greater than, signed $>$
BLE label ; branch if Z==1 or N!=V Less than or equal, signed \leq
BX Rm ; branch indirect to location specified by Rm
BL label ; branch to subroutine at label, return address in LR
BLX Rm ; branch to subroutine indirect specified by Rm

Interrupt instructions

CPSIE I ; enable interrupts (I=0)
CPSID I ; disable interrupts (I=1)

Logical instructions

AND{S} {Rd,} Rn, <op2> ; Rd=Rn&op2 (op2 is 32 bits)
ORR{S} {Rd,} Rn, <op2> ; Rd=Rn|op2 (op2 is 32 bits)
EOR{S} {Rd,} Rn, <op2> ; Rd=Rn^op2 (op2 is 32 bits)
BIC{S} {Rd,} Rn, <op2> ; Rd=Rn&(~op2) (op2 is 32 bits)
ORN{S} {Rd,} Rn, <op2> ; Rd=Rn|(~op2) (op2 is 32 bits)
LSR{S} Rd, Rm, Rs ; logical shift right Rd=Rm>>Rs (unsigned)
LSR{S} Rd, Rm, #n ; logical shift right Rd=Rm>>n (unsigned)

```

ASR{S} Rd, Rm, Rs      ; arithmetic shift right Rd=Rm>>Rs (signed)
ASR{S} Rd, Rm, #n      ; arithmetic shift right Rd=Rm>>n (signed)
LSL{S} Rd, Rm, Rs      ; shift left Rd=Rm<<Rs (signed, unsigned)
LSL{S} Rd, Rm, #n      ; shift left Rd=Rm<<n (signed, unsigned)
    
```

Arithmetic instructions

```

ADD{S} {Rd,} Rn, <op2> ; Rd = Rn + op2
ADD{S} {Rd,} Rn, #im12 ; Rd = Rn + im12, im12 is 0 to 4095
SUB{S} {Rd,} Rn, <op2> ; Rd = Rn - op2
SUB{S} {Rd,} Rn, #im12 ; Rd = Rn - im12, im12 is 0 to 4095
RSB{S} {Rd,} Rn, <op2> ; Rd = op2 - Rn
RSB{S} {Rd,} Rn, #im12 ; Rd = im12 - Rn
CMP      Rn, <op2>      ; Rn - op2      sets the NZVC bits
CMN      Rn, <op2>      ; Rn - (-op2)   sets the NZVC bits
MUL{S} {Rd,} Rn, Rm     ; Rd = Rn * Rm   signed or unsigned
MLA      Rd, Rn, Rm, Ra ; Rd = Ra + Rn*Rm signed or unsigned
MLS      Rd, Rn, Rm, Ra ; Rd = Ra - Rn*Rm signed or unsigned
UDIV     {Rd,} Rn, Rm   ; Rd = Rn/Rm    unsigned
SDIV     {Rd,} Rn, Rm   ; Rd = Rn/Rm    signed
    
```

Notes Ra Rd Rm Rn Rt represent 32-bit registers

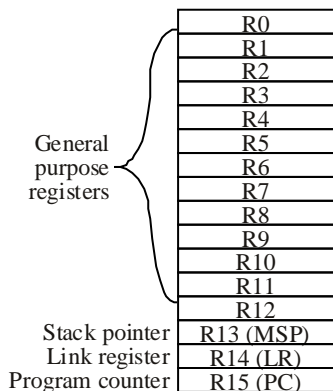
value any 32-bit value: signed, unsigned, or address
 {S} if S is present, instruction will set condition codes
 #im12 any value from 0 to 4095
 #im16 any value from 0 to 65535
 {Rd,} if Rd is present Rd is destination, otherwise Rn
 #n any value from 0 to 31
 #off any value from -255 to 4095
 label any address within the ROM of the microcontroller
 op2 the value generated by <op2>

Examples of flexible operand <op2> creating the 32-bit number. E.g., Rd = Rn+op2

```

ADD Rd, Rn, Rm          ; op2 = Rm
ADD Rd, Rn, Rm, LSL #n ; op2 = Rm<<n Rm is signed, unsigned
ADD Rd, Rn, Rm, LSR #n ; op2 = Rm>>n Rm is unsigned
ADD Rd, Rn, Rm, ASR #n ; op2 = Rm>>n Rm is signed
ADD Rd, Rn, #constant ; op2 = constant, where X and Y are hexadecimal digits:
    
```

- produced by shifting an 8-bit unsigned value left by any number of bits
- in the form 0x00XY00XY
- in the form 0xXY00XY00
- in the form 0xXYXYXYXY



Condition code bits
 N negative
 Z zero
 V signed overflow
 C carry or unsigned overflow

