Exam 1

Date: Oct 4, 2017

UT EID: ______________________ Professor: Valvano

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. *Anything outside the boxes/blanks will be ignored in grading*. You may use the back of the sheets for scratch work.
- You have 75 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.
- *Please read the entire exam before starting.*

Problem 1	12	
Problem 2	8	
Problem 3	10	
Problem 4	15	
Problem 5	10	
Problem 6	15	
Problem 7	20	
Problem 8	10	
Total	100	

(12) Question 1. Short answers.

(2) Part a) What are the two output states of open collector logic as used by the 7406 LED driver?

(2) Part b) Does the equation **power = voltage*current** apply to both resistors and LEDs? Answer yes or no.

(2) Part c) What does nonvolatile mean in context of computer memory?

(2) Part d) For what values of R0 does this code branch?

```
ORRS R0,R0,#4
BNE  FunTimes
```

(2) Part e) Considering R0 as input and R1 as output, what is the mathematical relationship between R1 and R0?

```
LSL R1,R0,#4
SUB R1,R1,R0
```

(2) Part f) If you add an *n*-bit signed number to an *m*-bit signed number, what is the maximum number of bits in the sum? Assume $n \geq m$.

(8) Question 2. Assume an 8-bit value has binary of 10010010.
What is the 8-bit value in unsigned hexadecimal format?

What is the 8-bit value in unsigned decimal format?

What is the 8-bit value in signed decimal format?

(10) Question 3 Assume `Data` is an 8-bit unsigned global variable in RAM.

```
uint8_t Data;
```

Write assembly code that performs the following C code (no function, just assembly code),

```
 if(Data >= 32){
   Data = 255;  // ceiling
 }else{
   Data = Data<<3;
 }
```

(15) Question 4. Consider the following C function with two inputs and one output.

```
uint32_t func(uint32_t in1, uint32_t in2){
  uint32_t out;
  out = 1;
  while(in1 >=  in2){
    out = out*in2;
    in2 = in2 + 1;
  }
  return out;
}
```

(5) Part a) Assume x is a 32-bit unsigned global variable. If we were to execute `x=func(6,4);` what would be the value of x?

(10) Part b) Write `func` in assembly using AAPCS

(10) Question 5. *Please read this question carefully*. Consider initialization code for a regular GPIO pin PE0. You will need to use some choices (A – E) more than once. Consider the following shorthand codes for 5 bits that need to be set during initialization

`Clk = SYSCTL_RCGCGPIO_R (bit 4)`	GPIO clock register
`Dir = GPIO_PORTE_DIR_R (bit 0)`	Direction register
`Pur = GPIO_PORTE_PUR_R (bit 0)`	Pull up register
`Pdr = GPIO_PORTE_PDR_R (bit 0)`	Pull down register
`Den = GPIO_PORTE_DEN_R (bit 0)`	Data enable register

For each of the ten cases, choose the description that best fits

A) PE0 cannot be used for input or output
B) PE0 is a regular output, where the output will be 0 or 3.3V
C) PE0 is a regular input, where input must be 0 or 3.3V
D) PE0 is an input used with a negative logic switch and no external resistor
E) PE0 is an input used with a positive logic switch and no external resistor

Clk	Dir	Pur	Pdr	Den	Place A, B, C, D, or E
0	0	0	0	1	
1	0	0	0	1	
1	0	0	0	0	
1	1	0	0	1	
1	1	0	0	0	
0	1	0	0	1	
1	0	1	0	1	
1	0	0	1	0	
1	0	0	1	1	
1	0	1	1	0	

(15) Question 6. Assume the value of the Stack pointer (SP) is **`0x20000FF8`** when the following code sequence starts execution (i.e., **`PC=0x00001000`**). The initial memory contents in and around the SP are given on the right. When drawing the stack contents, you need only to show values on the stack that represent actual valid stack data.

```
Initial PC → 0x00001000      POP {R0,R1}
             0x00001004      ADD R2,R0,R1
             0x00001008      BL  Func        ← B
             0x0000100C      ...
             ...
             0x00002000 Func PUSH {R2,LR}    ← A
             0x00002004      MOV  R2,R1
             0x00002008      MUL  R0,R2
             0x0000200C      ADD  R0,R1
             0x00002010      POP {R2,PC}
```

0x20000FF4	1
0x20000FF8	2 ← Initial SP
0x20000FFC	3
0x20001000	4
0x20001004	5
0x20001008	6
0x2000100C	7

(6) Part a) Give the SP value and stack contents after executing of the **`PUSH`** instruction, as shown by arrow A:

0x20000FF4	
0x20000FF8	
0x20000FFC	
0x20001000	
0x20001004	
0x20001008	
0x2000100C	

SP =

(10) Part b) Give the SP value and stack contents while executing the instruction at memory location 0x0000100C as shown by the arrow B. Also give the values stored in R0, R1, and R2.

0x20000FF4	
0x20000FF8	
0x20000FFC	
0x20001000	
0x20001004	
0x20001008	
0x2000100C	

SP =

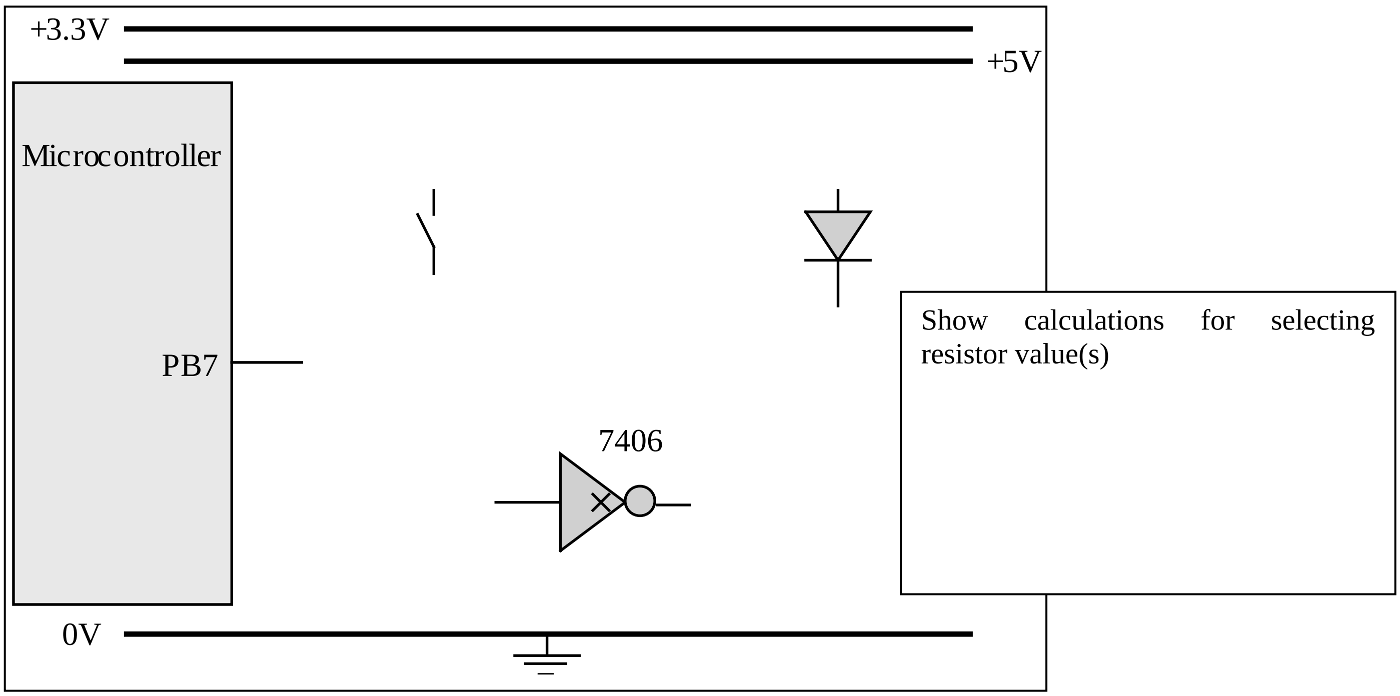
R0 =

R1 =

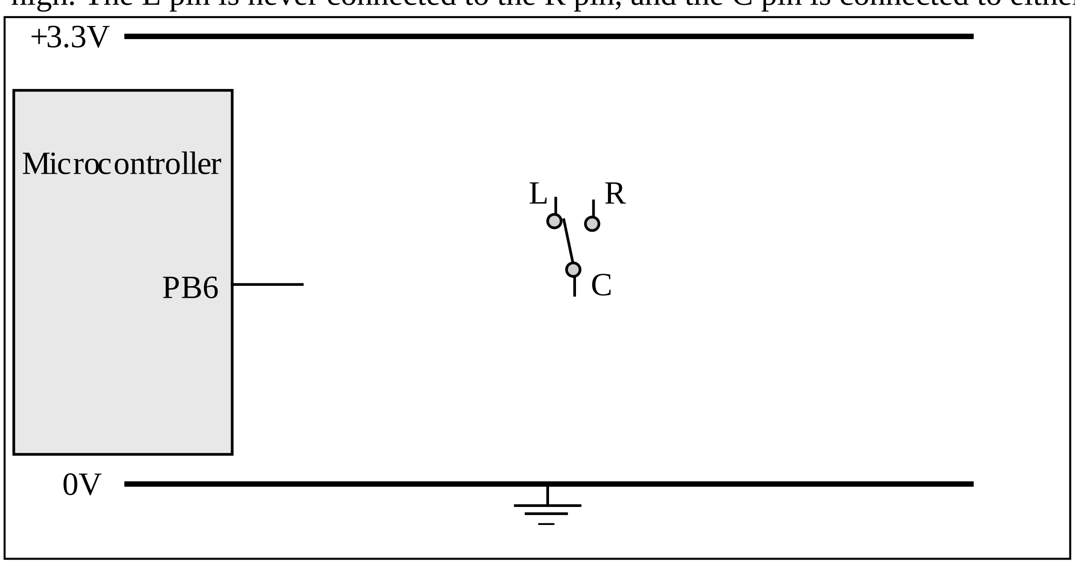
R2 =

(20) Question 7. Assume the microcontroller's output voltage high is 3.3V. Assume the microcontroller's output voltage low is 0V. The V_{OL} for the 7406 driver is 0.5V. Pick resistors 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 the interface*.

Part a) Interface the LED to Port B bit 7 (PB7) using negative logic. The LED operating point is 2.3V at 2mA.



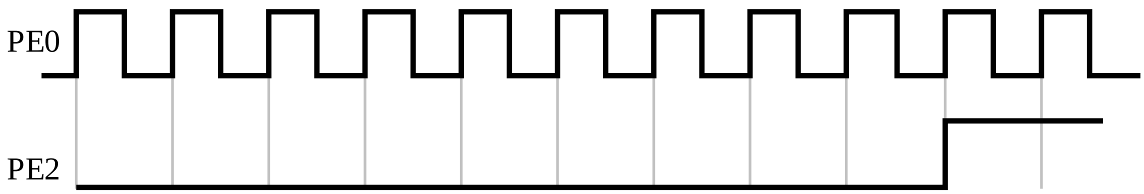
Part b) Interface this single pole double throw switch to the microcontroller PB6 input. The switch has two possibilities. The first case is the C pin is connected to the L pin. For this case, make PB6 low. The second case is the C pin is connected to the R pin. For this case, make PB6 high. The L pin is never connected to the R pin, and the C pin is connected to either L or R.



(10) Question 8. You may assume the **`PortE_Init`** function initializes PE1-PE0 as inputs, and initializes PE3-PE2 as outputs. Write the C code for this main program that performs both these two tasks over and over indefinitely:

Task 1) if PE1 equals PE0, then set PE3 high, otherwise set PE3 low;

Task 2) A trigger event is defined as the rising edge of PE0 (last time it was low, this time it is high). On the 10th trigger event, set PE2 high. Once PE2 is high it will remain high. Perform friendly output on Port E. You may assume the time PE0 is low and the time it is high are long compared to the time it takes the software to execute the loop once.



You may add local variables, and you may execute code before the **while(1)** statement. You will read and write to the Port E data register **GPIO_PORTE_DATA_R**.

```
int main(void){

  PortE_Init(); // you do not need to write this


  while(1){
```

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 ≥
BHS  label   ; branch if C == 1   Higher or same, unsigned ≥
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 ≤
BGE  label   ; branch if N == V   Greater than or equal, signed ≥
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 ≤
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`

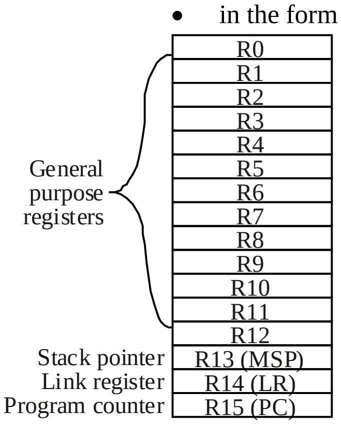
```
value   any 32-bit value: signed, unsigned, or address
{S}     if S is present, instruction will set condition codes
#im12   any value from 0 to 4095
#im16   any value from 0 to 65535
{Rd,}   if Rd is present Rd is destination, otherwise Rn
#n      any value from 0 to 31
#off    any value from -255 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