Exam 1

Date: Oct 3, 2019

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 and all subroutines AAPCS compliant.
- *Please read the entire exam before starting.*

(10) Question 1.

(2) Part a) Which equation describes the power dissipated in a resistor? .....

A) P = V/I B) $P = V^2 * R$
C) $P = I^2 * R$ D) $P = V/R^2$
E) Some of A – D F) None of A – D

D) $P=I^2R$

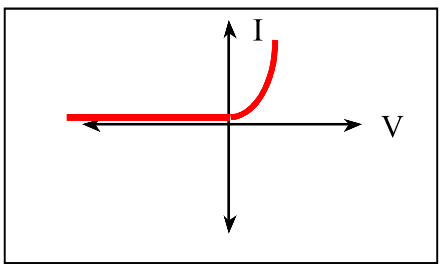
(2) Part b) If you add an *n*-bit signed number to an *m*-bit signed number, what is the maximum number of bits in the sum? Assume $n \geq m$. ....

n+1

(2) Part c) Assume you add two unsigned 32-bit numbers using the **`ADDS R2,R1,R0`** instruction. Which bit is set on overflow? ....

C-bit is set for unsigned overflow

(2) Part d) Sketch an approximate plot of LED current as a function of LED voltage:



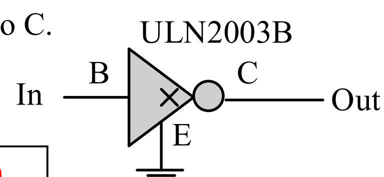
(2) Part e) What is the scope of a local variable in C? ...

A) Restricted, accessible only within the function

(2) Part f) Where are global variables allocated in C? ...

C) Permanent, they always exist, in RAM

(6) Question 2. Consider a ULN2003 driver chip, where the input **In** is connected to B, the emitter is grounded and the output **Out** is connected to C.



What is the **Out** if **In** is 3.3V? ....

Low, or 0V (any voltage from 0 to 0.5V is ok)

What is **Out** if **In** is 0.0V? ....

Floating, off, not driven, or hiZ

(12) Question 3. There is a 16-bit signed global variable called **`Stuff`**.

```
      AREA   DATA,ALIGN=2
Stuff SPACE 2
```

Write Cortex M assembly subroutine that performs the same operation as this C function

```
int16_t Stuff;
void ShiftandAdd(void){
  Stuff = (Stuff>>2)+5;
}
```

```
ShiftandAdd
  LDR   R0,=Stuff
  LDRSH R1,[R0]
  ASR   R1,R1,#2
  ADD   R1,R1,#5
  STRH  R1,[R0]
  BX    LR
```

(5) Question 4. There is a 32-bit constant called **`Thing`**.

```
      AREA    |.text|, CODE, READONLY, ALIGN=2
Thing DCD 0xF012349A
```

What is the value of R0 in hex after this assembly code is executed?

```
  LDR   R1,=Thing
  LDRSB R0,[R1]
```

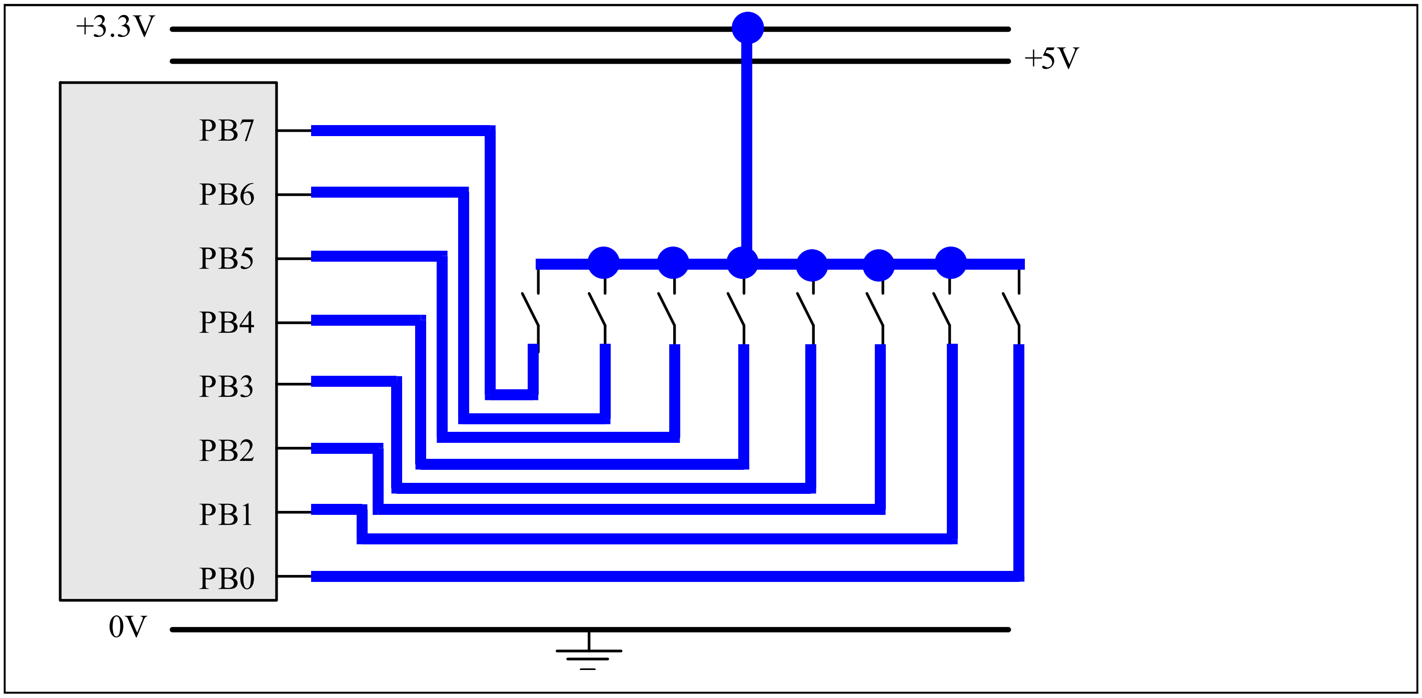
0xFFFFFF9A (little endian)

(12) Question 5. Show the declaration of a C function that finds the minimum value of an array. The length of the array is fixed at 1000. A pointer to the array is passed by reference into the function. The function returns the smallest value in the array. The function prototype is

`int8_t min(int8_t data[1000]);`

```
int8_t min(int8_t data[1000]){int8_t result;
  result = data[0];
  for(int i=1; i<1000; i++){
    if(result > data[i]){
      result = data[i];
    }
  }
  return result;
}
```

(10) Question 6. Interface eight switches to Port B using positive logic.
(5) Part a) For full credit, design the hardware interface that uses the fewest number of external components (resistors, LEDs, ULN2003B).



(5) Part b) During initialization what values should to write into the following registers? For registers you would not initialize, enter NA into the box.

GPIO_PORTB_DATA_R .... NA

GPIO_PORTB_DIR_R 0x00

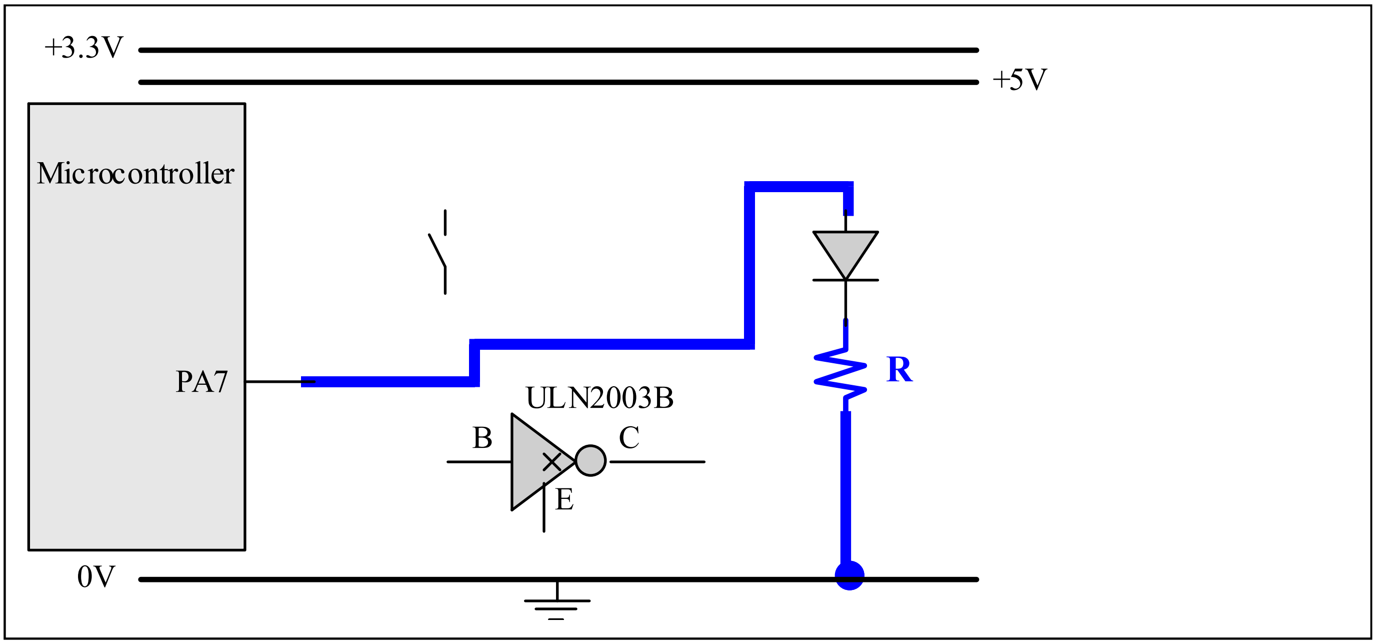
GPIO_PORTB_PUR_R .... NA or 0

GPIO_PORTB_PDR_R 0xFF

GPIO_PORTB_DEN_R 0xFF

(10) Question 7. Interface an LED to PA7 using positive logic.

(5) Part a) The desired LED operating point is 2V, 1mA. The microcontroller output high voltage is 3.2V, the microcontroller output low voltage is 0.2V. The ULN2003 output low voltage is 0.3V. For full credit, design an interface that uses the fewest number of external components. Hardware only, no software. For any resistor(s) you use, show your work for determining the resistor value(s). **R = (3.2V-2V)/1mA = 1.2kΩ**



(5) Part b) Assume Port A is initialized so PA7 is an output. Write a function in C that accepts an input parameter (0 or 0x80) and writes to Port A in a friendly manner. Include both the prototype and the declaration of the function.

```
#define GPIO_PORTA_DATA_R   (*((volatile uint32_t *)0x400043FC))
void LED_Set(uint32_t value);

void LED_Set(uint32_t value){ uint32_t old;
  old = GPIO_PORTA_DATA_R; // previous data
  old = old&(~0x80);       // clear bit 7
  GPIO_PORTA_DATA_R = old|value; // new value in bit 7
}
void LED_Set(uint32_t value){
  if(value == 0x80){
    GPIO_PORTA_DATA_R |= 0x80;  // turn on
  } else{
    GPIO_PORTA_DATA_R &= ~0x80; // turn off
  }
}
```

(15) Question 8.

```
0x00001000       POP {R0,R1}
0x00001004       ADD R2,R0,R1
0x00001008       BL  Func
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
0x20000FFC	3
0x20001000	4
0x20001004	5
0x20001008	6
0x2000100C	7

(6) Part a) Give the state of the stack (SP and contents) after executing of the **PUSH** instruction, as shown by arrow A: **pop (R0,R1}** causes R0=2, R1=3, SP = 0x20001000

Add R2,R0,R1 causes R2 = 5

BL causes LR = 0x0000100D

Push R2,LR

0x20000FF4	
0x20000FF8	5
0x20000FFC	0x0000100D
0x20001000	4
0x20001004	5
0x20001008	6
0x2000100C	7

SP = 0x20000FF8

R0 = 2

R1 = 3

R2 = 5

We give full credit for 0x0000100C. On the ARM/Thumb processors, the PC is 32 bits with bit 0 always clear. The processor uses this bit to specify if the destination code is ARM (0) or Thumb (1). For EE319K this bit will always be 1 for Thumb.

(25) Question 9. In this question there are two microcontrollers, such that the two Port B's are connected (PB7 to PB7, PB6 to PB6,…PB0 to PB0). The goal is to send a three bit value from one microcontroller to the other. The transmit software will be on the left microcontroller. At all times the transmitter must have exactly one of the Port B pins high. E.g., the 8-bit Port B data must be 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40 or 0x80. You may assume the transmitter on the left has initialized all of Port B to be output, and the receiver on the right has initialized all of Port B to be input. On the transmitter, design an assembly function that accepts a 3-bit value in R0 (let n be the value in R0, you may assume $0 \le n \le 7$) and write to Port B the value 2^n. On the receiver design an assembly function that reads Port B (knowing the value will be restricted to 2^n for some $0 \le n \le 7$) and returns in R0 the value n.

```
GPIO_PORTB_DATA_R EQU 0x400053FC
```

```
;Input: R0 is 0 to 7
Transmit
      MOV R1,#1
loop  CMP R0,#0
      BEQ done
      LSL R1,#1 ;1,2,4,… 0x80
      SUB R0,#1
      B   loop
done  LDR R0,=GPIO_PORTB_DATA_R
      STR R1,[R0]
      BX  LR
Transmit2
      LDR  R2,=Table
      LDRB R1,[R2,R0]
      LDR  R0,=GPIO_PORTB_DATA_R
      STR  R1,[R0]
      BX   LR
Table DCB 0x01,0x02,0x04,0x08
      DCB 0x10,0x20,0x40,0x80
Transmit3
      MOV R1,#1
      LSL R2,R1,R0
      LDR R3,=GPIO_PORTB_DATA_R
      STR R2,[R3]
      BX  LR
```

```
;Output: R0 is 0 to 7
Receive
      LDR  R2,=GPIO_PORTB_DATA_R
      LDR  R1,[R2]
      MOV  R0,#0 ;n
loop2 LSRS R1,#1 ;bit goes into C
      BCS  done2
      ADD  R0,#1 ;n=n+1
      B    loop2
done2 BX   LR

Receive2
      LDR  R2,=GPIO_PORTB_DATA_R
      LDR  R1,[R2]
      CLZ  R3,R1 ;count leading zeros
      MOV  R0,#32
      SUB  R0,R0,R3
      BX   LR
```

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`

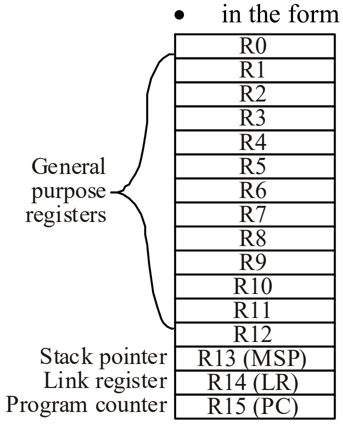
```
value   any 32-bit value: signed, unsigned, or address
{S}     if S is present, instruction will set condition codes
#im12   any value from 0 to 4095
#im16   any value from 0 to 65535
{Rd,}   if Rd is present Rd is destination, otherwise Rn
#n      any value from 0 to 31
#off    any value from -255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

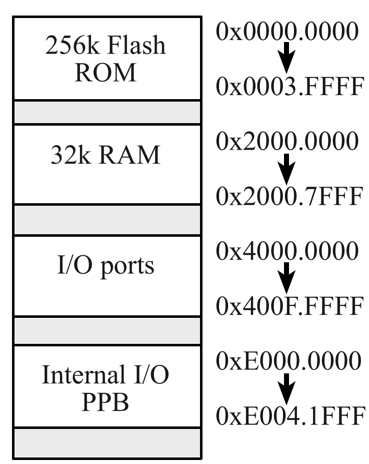
ADD Rd, Rn, #constant ; op2 = constant, where **X** and **Y** are hexadecimal digits:

- 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




```
DCB  1,2,3 ; allocates three 8-bit byte(s)
DCW  1,2,3 ; allocates three 16-bit halfwords
DCD  1,2,3 ; allocates three 32-bit words
SPACE 4    ; reserves 4 bytes
```