Exam 1 solution

Date: Feb 28, 2020

UT EID: ________________________ Professor: Cuevas, Valvano, Yerraballi

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.*

(12) Question 1. Know your Basics

(2) Part a. Which equation describes the current through an arbitrary LED? Give one letter A-F.

A) I = V/R B) I = V^2*R
C) I = R/V D) I = P/V
E) I = 8mA F) None of A – E

D) P = I*V

where I=current, V=voltage, R=resistance, P=power

(2) Part b. Some assembly instructions have a limitation that they support only 12-bit signed immediate values. What are the ranges of signed integers they support?

-2^11 to (2^11)-1
-2048 to +2047

(2) Part c. `0x20001234` equals `536,875,572` in decimal. Consider the instruction

```
LDR R0,=536875572
```

Give the one letter A-F that best explains how this instruction works.

A) This is an immediate addressing mode instruction, where the value `536,875,572` is embedded in the machine code of the instruction (not true, value placed elsewhere in ROM)

B) This has a syntax error and will not assemble, addresses must be in hex (not true). The proper syntax is to write **`LDR R0,= 0x20001234`**.

C) This is a memory access instruction. The 32-bit contents located at RAM location `0x20001234` are copied into R0 (not true addr 0x20001234 is in R0).

D) This causes a syntax error (no), the equals is not needed. The proper syntax is to write **`LDR R0,536875572`**.

E) The LDR has an offset limited to 16 bits (no), so it truncates the value `R0=0x1234`.

F) None of A – E

F

(2) Part d. There is a 32-bit constant called **`Thingy`** declared in ROM as shown below

```
        AREA    |.text|,CODE,READONLY,ALIGN=2
Thingy  DCD 0xF1F04321
```

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

```
  LDR   R1,=Thingy
  LDRSH R0,[R1]
```

Little endian gets 0x4321
Sign is 0 (positive)
Sign extend to 0x00004321

(4) Part e. Consider the following sequence of assembly code that performs some logic and shift operations. Give the values (in Hex) of the four registers executing these four instructions:

`MOV R0,#-1`	R0= 0xFFFFFFFF
`EOR R1,R0,#0xF0F0F0F0`	R1= 0x0F0F0F0F
`BIC R2,R0,#0x04`	R2= 0xFFFFFFFB (clear bit 2)
`LSL R3,R0,#4`	R3= 0xFFFFFFF0 (shift left 4 times)

(15) Question 2. Know your C

Implement a C function that takes an array (A) of size (N) and increments every Kth element of the array by a given value V. Return the number of elements updated or a 255 in case of an invalid input. Examples below cover all possible cases (*please read them before asking questions*):

N, K, V	Array A before	Indices	result	Array A after
5, 1, 2	{1,2,3,4,5}	0,1,2,3,4	5	{3,4,5,6,7}
7,2,3	{10,20,30,40,50,60,70}	0,2,4,6	4	{13,20,33,40,53,60,73}
10,6,4	{0,0,0,0,0,0,0,0,0,0}	0,6	2	{4,0,0,0,0,0,4,0,0,0}
0, 7, 6	{}	none	0	{}
8,5,1	{10,10,10,10,10}	0	1	{11,10,10,10,10}
4, 0, 2	{1,2,3,4}	none	255	{1,2,3,4}

```
uint8_t EveryK(uint16_t A[],uint8_t N,uint8_t K,uint16_t V){
uint8_t i,count=0;

  if (k==0) return 255;

  for(i=0; i<N; i=i+K){
    A[i] = A[i]+V;
    count++;
  }
  return count;
}
```

(15) Question 3. Know how to convert
There are two 8-bit global variables. Write two assembly subroutines that are literal conversions of these two C functions. Follow AAPCS.

```
uint8_t Item;
void Change(void){
  if(Item < 42){
     Item++;
  }
}
```

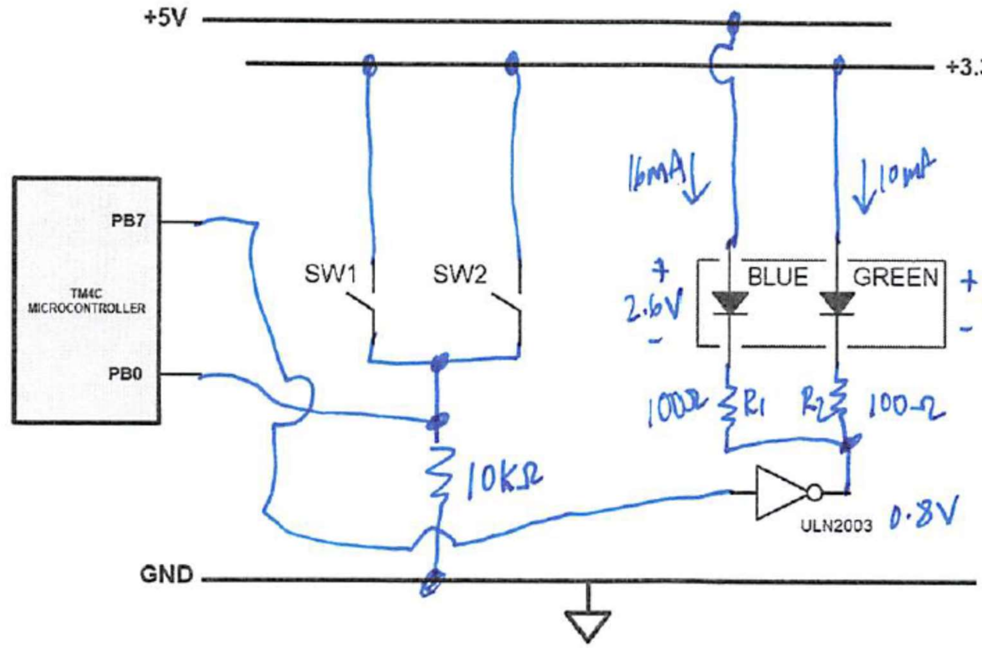
```
int8_t Item2;
void Change2(void){
  while(Item2 < 42){
     Item2++;
  }
}
```

```
    AREA  DATA,ALIGN=2
Item SPACE 1
    AREA |.text|,CODE,ALIGN=2
Change
  LDR   R0,=Item
  LDRB  R1,[R0]
  CMP   R1,#42
  BHS   done
  ADD   R1,R1,#1
  STRB  R1,[R0]
done
  BX    LR
```

```
    AREA  DATA,ALIGN=2
Item2 SPACE 1
    AREA |.text|,CODE,ALIGN=2
Change2
  LDR   R0,=Item2
loop2
  LDRSB R1,[R0]
  CMP   R1,#42
  BGE   done2
  ADD   R1,R1,#1
  STRB  R1,[R0]
  B     loop2
done2
  BX    LR
```

(20) Question 4) Know Switch/LED Interfacing

(10) Part a. Using only ONE 10kΩ resistor, interface both switches to the microcontroller Port B, bit 7 such that the input voltage is HIGH when either switch is closed and LOW otherwise. The microcontroller is powered by 3.3V. Show your circuit below.



The LED and resistor can be in either order

You can't use one resistor for two LEDs, because the currents are different. Must use two resistors for LEDs.

(10) Part b. You have a BLUE LED with an operating point of 16mA at 2.6V; and you have a GREEN LED with an operating point of 10mA at 1.5V. The two LEDs are inside the same package, so by turning them both ON at the same time, you get a CYAN color. Using, at most, only TWO resistors, interface both LEDs to the microcontroller Port B, bit 0 using positive logic. The LEDs will be either both ON or both OFF at the same time. The microcontroller's output high/low voltage is 3.3V and 0V, respectively. The V_{OL} for the ULN2003B driver is 0.8V. You have +5V, +3.3V, and GND to which you can connect your components. **Another restriction is all resistor values must be less than 250Ω.** Show your circuit above and compute the resistor value(s) needed for the above operating points. Show your calculations below.

WRONG

$R_1 = \frac{(3.3-2.6-0.8)}{0.016} = -6.25\Omega$

$R2 = \frac{(5-1.5-0.8)}{0.010} = 270\Omega$ $(>250\Omega)$ ✗

Neither can attach directly to PB7 (only 8mA available)

$R_1 = \frac{(5-2.6-0.8)}{0.016} = \frac{1.6}{0.016} = 100\Omega$ $(<250\Omega)$ ✓

$R2 = \frac{(3.3-1.5-0.8)}{0.010} = \frac{1.0}{0.010} = 100\Omega$ $(<250\Omega)$ ✓

(13) Question 5) Know your I/O

(7) Part a. Fill in the boxes so this assembly code initializes **Port A**, making **PA6 PA5** outputs and making **PA7 PA4** inputs. This code is executed once at the start of the system. **All accesses to I/O registers must be friendly.** Your *code* will set the *clock*, *direction*, and *enable* registers. You must fill in the op codes and immediate values. Each box contains exactly one assembly op code. Each oval has exactly one hex value. Do not assume DIR, DEN or DATA registers have been cleared by the reset operation. Comments are not needed.

```
GPIO_PORTA_DATA_R  EQU 0x400043FC ;data register
GPIO_PORTA_DIR_R   EQU 0x40004400 ;direction register
GPIO_PORTA_DEN_R   EQU 0x4000451C ;digital enable register
SYSCTL_RCGCGPIO_R  EQU 0x400FE608 ;GPIO clock register
Init
    LDR R3,=SYSCTL_RCGCGPIO_R
    LDR R2,[R3]
   [ORR]  R2,R2,#(0x01)
    STR R2,[R3]
    NOP
    NOP
    LDR R3,=GPIO_PORTA_DIR_R
    LDR R2,[R3]
   [ORR]  R2,R2,#(0x60)
   [BIC]  R2,R2,#(0x90)
    STR R2,[R3]
    LDR R3,=GPIO_PORTA_DEN_R
    LDR R2,[R3]
   [ORR]  R2,R2,#(0xF0)
    STR     R2,[R3]
    BX      LR
```

(6) Part b. Write assembly code that sets PA6=1 in a friendly manner

```
LDR R0,=GPIO_PORTA_DATA_R
LDR R1,[R0]    ;ok if you use LDRB
ORR R1,#0x40   ;bit 6
STR R1,[R0]    ;ok if you use STRB
```

(10) Question 6. Know your Stack

Show the contents of the stack and register values after the two marked points respectively in the execution of the following code. The initial stack pointer is `0x20001008`.

```
0x00002000      MOV   R4,#2         ; R4=2
0x00002002      MUL   R0,R4,R4      ; R0=4
0x00002004      ADD   R1,R4,#1      ; R1=3
0x00002006      SUB   R2,R4,#1      ; R2=1
0x00002008      BL    Subtract      ; LR=0x0000200A or 0x0000200B
0x0000200A      ADD   R3,R0,R4      ; 8
                                  ; <---- B
...
0x00002020   Subtract
0x00002022      PUSH {R4,R2,R1,LR}
0x00002024      ADD   R4,R0,R1 ; <---- A R4=7
0x00002026      SUB   R0,R4,R2 ;         R0=6
0x00002028      POP   {R1,R2,R4,PC} ; R1=3,R2=1,R4=2,PC=0x0000200A
```

(4) Part a. Give the state of the stack (SP and contents) after execution point A:

0x20000FF8	3	← SP
0x20000FFC	1	
0x20001000	2	
0x20001004	0x000200A	
0x20001008	---	
0x2000100C	---	
0x20001010	---	
0x20001014	---	
0x20001018	---	

(6) Part b. Give the values stored in register R0-R4 and SP after execution point B:

R0 = 6	**R3 =** 8
R1 = 3	**R4 =** 2
R2 = 1	**SP =** 0x20001008

(15) Question 7. Know how to Design

You write an assembly function called **`Expo`**, which calculates $A*B^C$, where inputs are 32-bit integers. *A* and *B* are signed, and *C* is unsigned. The result will be a signed 32-bit number. You may ignore overflow, *B* and *C* will not both be zero. Follow AAPCS. The C prototype is

`int32_t Expo(int32_t A, int32_t B, uint32_t C);`

Your solution should correctly perform these input/output examples

A, B, C	Calculation	result	comment
-4, 12, 0	-4	-4	any nonzero raised to the 0^{th} power is 1
100, 0, 13	0	0	
3, 10, 4	3*10*10*10	30000	
7, -3, 5	7*(-3)*(-3)*(-3)*(-3)*(-3)	-1701	

```
;AAPCS places inputs A,B,C in R0,R1,R2
;      places return result R in R0
A     RN    0
B     RN    1
C     RN    2
R     RN    3
Expo MOV    R,A    ;R=A
loop CMP    C,#0   ;check C
     BEQ    done   ;return R0=A if C==0
     MUL    R,R,B  ;R=R*B
     SUB    C,#1   ;loop C times
     B      loop   ;A*B*B*B...
done MOV    R0,R   ;return result
     BX     LR
Expo CMP    R2,#0 ;check C
     BEQ    done   ;return R0=A if C==0
     MUL    R0,R0,R1 ;R=R*B
     SUB    R2,#1   ;loop C times
     B      loop    ;A*B*B*B...
done                ;return result
     BX     LR
Expo MOV    R3,#1   ;product of B*B*B...
loop CMP    R2,#0   ;check C
     BEQ    done    ;return R0=A if C==0
     MUL    R3,R3,R1 ;R=R*B
     SUB    R2,#1   ;loop C times
     B      loop
done MUL    R0,R0,R3 ;A*B*B*B...
     BX     LR      ;return result
```