Exam 1

Date: February 20, 2014

UT EID: ____________________

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	10	
Problem 2	10	
Problem 3	10	
Problem 4	10	
Problem 5	10	
Problem 6	15	
Problem 7	25	
Problem 8	10	
Total	100	

(10) Question 1. Place your answers in the boxes.

Part a) The total addressable memory on the ARM Cortex M processor is these many bytes. Give the total possible, not the actual number on TM4C123.

Part b) This C operator is used to perform a bit-wise NOT operation is.

Part c) We access device registers just like we access memory. The term used for this kind of I/O is.

Part d) In conditional C expressions, a Zero value is interpreted as False. What is interpreted a True?

Part e) What LED parameter (Voltage or Current) determines whether we need a 7406 driver.

Part f) A big-endian machine will interpret a byte 0x28 as having a value of (2*16+8) = 40, whereas a little-endian machine will interpret 0x28 as (8*16+2) = 130. *True or False*.

Part g) What is the term "addressing mode" associated with, *Instructions* or *Operands*?

Part h) This data-type is the most appropriate one to create a variable in C that can take values in the range -200 to +200.

Part i) Which three registers or the ARM Cortex processor are initialized on Reset?

Part j) Give an example of a non-intrusive debugging tool.

(10) Question 2
Consider the following arithmetic operation. Assume that all registers are 8-bit.

R0 ← 100
R1 ← 227
R2 ← R0 – R1

What are the numbers in registers R1 and R2 in unsigned decimal and signed decimal?

[R1] = _________________ (unsigned decimal)
[R1] = _________________ (signed decimal)

[R2] = _________________ (unsigned decimal)
[R2] = _________________ (signed decimal)

What are the values of the condition code bits: N, Z, V, C ?
N = __________
Z = __________
V = __________
C = __________

(10) Question 3

A programmer wants to make make pins PB1, PB4, PB7 outputs and make pin PB0 an input. So he writes the below code in sequence as shown. He claims it does not work. Your job as an expert is to identify the mistake(s). First, start by commenting what is the purpose of each statement in the code. Second, after each statement, please write either OK or explain what is wrong and provide the necessary corrections to make things work as expected. You are free to add, remove or modify the sequence, as well as the code. Assume that you have access to the following correct definitions:

```
#define GPIO_PORTB_DATA_R  (*((volatile unsigned long *)0x400053FC))
#define GPIO_PORTB_DIR_R   (*((volatile unsigned long *)0x40005400))
#define GPIO_PORTB_AFSEL_R (*((volatile unsigned long *)0x40005420))
#define GPIO_PORTB_DEN_R   (*((volatile unsigned long *)0x4000551C))
#define SYSCTL_RCGCGPIO_R  (*((volatile unsigned long *)0x400FE608))
```

```
SYSCTL_RCGCGPIO_R    = 0x02;
```

```
GPIO_PORTB_DATA_R = 0x00;
```

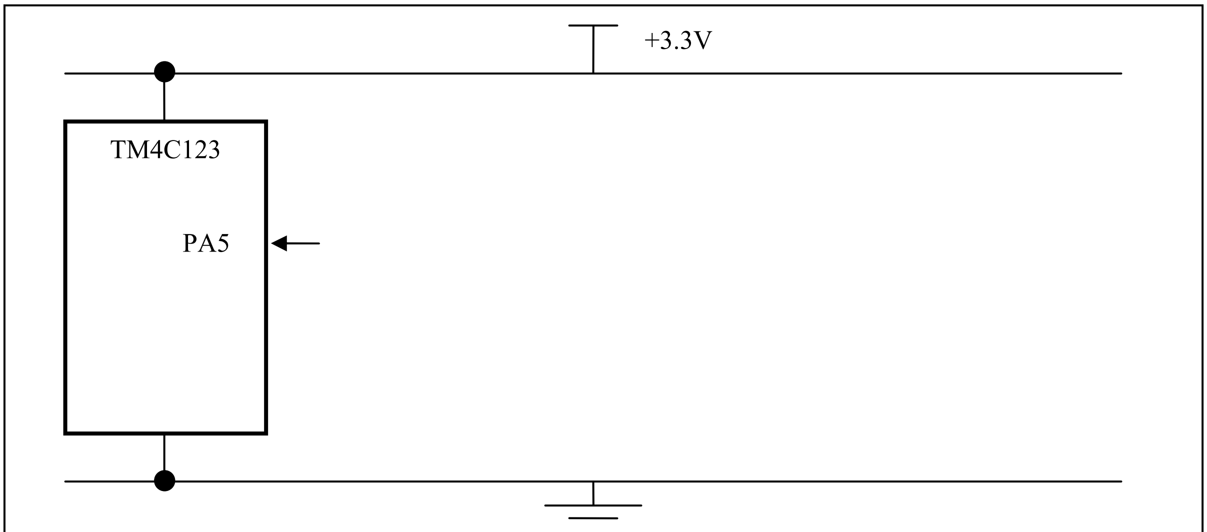
```
GPIO_PORTB_DIR_R |= 0x92;
```

```
GPIO_PORTB_DIR_R &= 0x01;
```

```
GPIO_PORTB_AFSEL_R &= ~0x93;
```

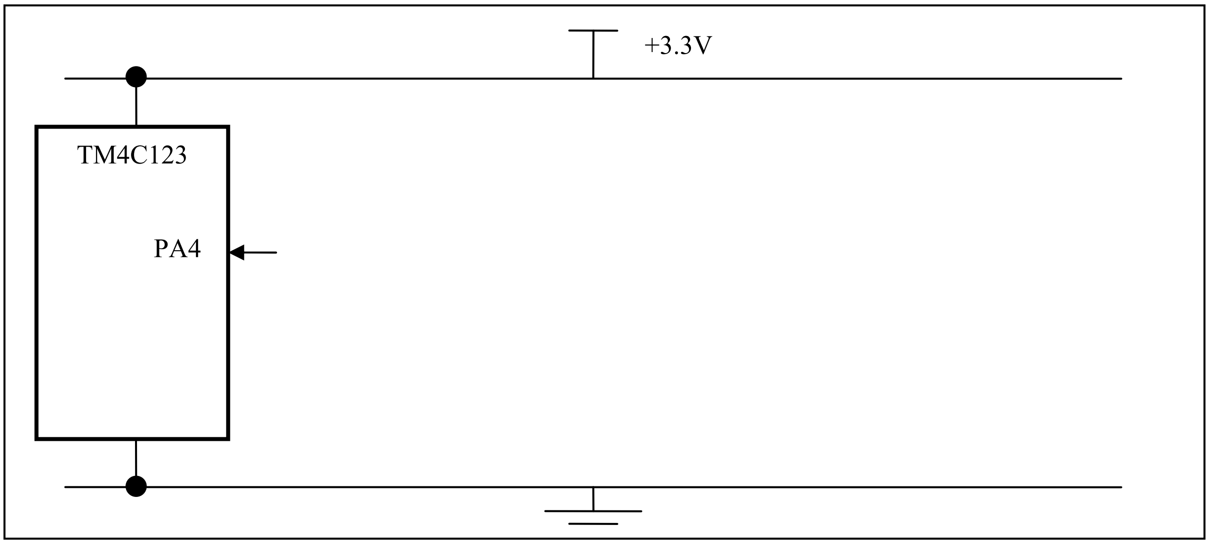
(10) Question 4.

(a) Interface a switch using a 10 kΩ resistor to port PA5 using positive logic. You may assume PA5 has been configured as an input port. Also assume that no current can flow into and out of the port pin and the switch is ideal. Find the current through the switch and the voltage across the resistor. Complete the table below the figure.



Switch configuration	Current through switch	Voltage across the resistor
Switch open		
Switch closed		

(b) Interface an LED through a resistor to port PA4 using negative logic. You may assume PA4 has been configured as an output port. The operating point of this LED is 1.5V at 1.8mA. The V_{OL} and V_{OH} of the TM4C123 is 0.3V and 3.3V resp., and the maximum current that PA4 can source or sink is 8mA. Find the value of the resistor R that needs to be connected, and show the circuit diagram. What is the current through the LED and the voltage across the resistor? Complete the table below the figure.



PA4 output	Current through LED	Voltage across the resistor
Logic level 1		
Logic level 0		

(10) Question 5. Write an *assembly* subroutine, called **`SwapLT`**, that swaps the contents of two global variables *ying* and *yang* only if *ying* is less than *yang*. Assume that variables are 16-bit signed numbers located in global RAM. You may use Registers R0-R3, or R12 as scratch registers without saving and restoring them. Don't worry about how the variables are initialized.

```
       AREA    DATA, ALIGN=2
ying   SPACE   2
yang   SPACE   2
       AREA    |.text|, CODE, READONLY, ALIGN=2
```

(15) Question 6. You are given the assembly Subroutine **`Max3`** below that takes three inputs and returns the maximum of the three inputs. **`one two three`** are input parameters, and **`max`** is the return parameter. There are no bugs in this subroutine, but it is not AAPCS compliant.

Part a) Make changes to anything inside the box so **`Max3`** becomes AAPCS compliant.

```
one   RN  1
two   RN  2
three RN  3
max   RN  5
      EXPORT Max3
Max3
      PUSH {R4}
      CMP  one,two
      BHI  C13
      CMP  two,three
      BHI  Mx2
      B    Mx3
C13   CMP  one,three
      BLO  Mx3
      MOV  max,one
      B    Done
Mx2   MOV  max,two
      B    Done
Mx3   MOV  max,three
Done
      POP  {R4}
      BX   LR
```

Assume these global variables
`uint32_t result,w,x,y,z;`

Part b) Write one line of C code that calls the **`Max3`** assembly subroutine passing the values `7000`, `1134`, and `4556` storing the result in global variable **`result`**.

Part c) Write one line of C code that calls the **`Max3`** assembly subroutine passing the contents of global variables **`x`**, **`y`**, and **`z`** storing the result in global variable **`w`**.

(25) Question 7. You are asked to write a software module that controls the child-lock feature on car door. An indicator light (connected to PortA pin 1 on the microcontroller) on the dashboard shows whether the child-lock feature is engaged or not. A switch (connected to PortA pin 0) controls whether the feature is enabled or disabled. There is some external hardware in the form of a weight sensor that sends an 8-bit input value on PortB indicating the weight of the person/object in the seat. If the child-lock feature is enabled by turning the switch (PA0) to ON, you have to read the weight from PortB and check it to see if the indicator light must be turned on. The indicator light (on PA1) must be turned on when the weight (8-bit value on PortB) is between 10 (`GPIO_PORTB_DATA_R=0x0A`) and 35 (`GPIO_PORTB_DATA_R=0x23`) and the child-lock feature is enabled (PA0 is 1). Otherwise, it must be off.

You may assume the hardware is already connected, and all initializations except setting/clearing the Direction register for Port B, are done. You have access to registers **`GPIO_PORTB_DIR_R`**, **`GPIO_PORTA_DATA_R`**, and **`GPIO_PORTB_DATA_R`** to complete your code. Write **assembly** code that manipulates the direction registers first and then continuously checks the switch and weight and updates the LED state accordingly.

Suggestion: *It might help if you visualize the solution using a flowchart*

```
…   ; Declarations already in place for GPIO registers
Start
      …; Code for GPIO initialization you are not responsible for is here
       ; *** Code you are responsible for follows ****

       ; Port B Direction register initialization




       ; Logic to check inputs and produce appropriate outputs follows



Loop
```

```
    B Loop
    ALIGN
    END
```

(10) Question 8. Show the contents of the stack after the two marked points in the execution of the following code. The initial stack pointer is **`0x20001008`**.

```
0x00002000    MOV R0,#3
0x00002002    MUL R4,R0,R0
0x00002004    ADD R5,R4,#1
0x00002006    ADD R6,R4,#2
0x00002008    BL Sub
0x0000200A    ADD  R4,R5,R6
                                 ; <---- B

Sub
   PUSH {LR,R4,R5,R6}
   MUL  R4,R4,R0 ; <---- A
   POP  {R4,R5,R6,PC}
```

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

	Contents	Address
		0x20000FF4
		0x20000FF8
		0x20000FFC
		0x20001000
		0x20001004
Initial SP →		0x20001008
		0x2000100C

b) (6 points) Give the state of the stack (SP and contents) after execution point B and the value stored in R4:

	Contents	Address
		0x20000FF4
		0x20000FF8
		0x20000FFC
		0x20001000
		0x20001004
Initial SP →		0x20001008
		0x2000100C

R4 =

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
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
```

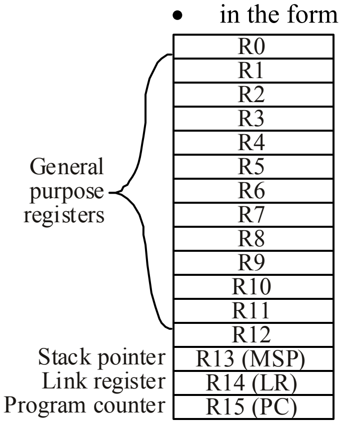
Notes `Ra Rd Rm Rn Rt represent 32-bit registers`

```
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