Exam 1

Date: July 9, 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 will be ignored in grading.
• 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  20
Problem 2  15
Problem 3  15
Problem 4  20
Problem 5  10
Problem 6  20
Total    100
(10 points) **Question 1.** Place your answers in the boxes.

**Part a)** What portion of the memory is the Stack stored on the ARM.

**Part b)** There is no NOT operation in the ARM ISA. Give the assembly instruction that will NOT bit 0 of Register R1.

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

**Part d)** How many bytes does the following array declaration take:

uint8_t Arr[] = {12,13,42,54,5};

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

**Part f)** There are two ways to pass parameters to subroutines, one is call-by-value, what is the other?

**Part g)** In keeping with AAPCS conventions, what is the maximum number of parameters one can pass (to a subroutine) using registers?

**Part h)** This data-type is the most appropriate one to create a variable in C that can take values in the range -2000 to +2000.

**Part i)** What is the first entry in the Interrupt Vector Table called?

**Part j)** Give an example of a non-intrusive debugging tool.
A programmer wants to 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:

```c
#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_RCGC2_R (*((volatile unsigned long *)0x400FE108))
```

SYSCTL_RCGC2_R = 0x02; ✓

GPIO_PORTB_DATA_R = 0x0; // OK will delay

GPIO_PORTB_DIR_R |= 0x92; // OK output pin PB7, PB4, PB1

```c
GPIO_PORTB_DIR_R &= 0x01; // OK Input on PB0
```

GPIO_PORTB_AFSEL_R &= ~0x93; ✓ // OK Disables Alt function

GPIO_PORTB_DEN_R = 0x93 // Digital Enable
(16 points) Question 3.
(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. Find the current through the switch and the voltage across the resistor. Complete the table below the figure.

<table>
<thead>
<tr>
<th>Switch configuration</th>
<th>Current through switch</th>
<th>Voltage across the resistor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch open</td>
<td>0 A</td>
<td>0 V</td>
</tr>
<tr>
<td>Switch closed</td>
<td>0 A</td>
<td>3.3 V</td>
</tr>
</tbody>
</table>

(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 respectively. Find the value of the resistor $R$ that needs to be connected, and show the circuit diagram.
(20 pts) Question 4.

Part a) (10 points) Given below is the assembly code for a subroutine called `func`. Translate the assembly code into an equivalent C function that corresponds to the given code.

<table>
<thead>
<tr>
<th>C code</th>
<th>Assembly code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>func(unsigned short x, unsigned short y)</code></td>
<td></td>
</tr>
</tbody>
</table>
| ```c
    if (x >= y) {
        x = y;
        return x;
    }
``` |
| `func CMP R0, R1
  BLO L1
  PUSH {R1}
  POP {R0}
  BX LR` |

Also briefly explain what function the code computes:

- Returns the smaller of the two inputs passed.

Part b) (10 points) Write a C function ‘g’ to perform the following functionality: read the value from global variable `input`, divide it by two, pass the result and the value 50 into the function ‘func’ from (a), and write the result of that to the global `output` variable.
(10 pts) Question 5. You are given the assembly Subroutine `Max3` below that takes three inputs in R1, R2 and R3 and returns the maximum of the three inputs R5 as 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.

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`. Assume that a global variable result has been declared as: `uint32_t result;`

```c
result = Max3(7000, 1134, 4556);
```
**Question 6.** This question tests your understanding of Arrays.

**Part a)** Assume that array of characters that is null-terminated, is passed to at function called `ChkSum`. The function computes the bit-wise XOR (exclusive or) of all the characters in the passed array and returns the result. Complete (fill blanks) the declaration and body of the `ChkSum` function:

```c
// Computes Checksum of a null-terminated character array.  
// Input: A null-terminated character array  
// Output: The Checksum

uint8_t ChkSum(char Arr[?]) {
    uint8_t res = 0, i = 0;
    while (Arr[i] != 0) {
        res = res ^ Arr[i];
        i++;
    }
    return res;
}
```

**Part b)** Declare an array called `Data` and initialize it to the five characters `H`, `O`, `W`, `D`, `Y`, providing enough space to be passed as input to the `ChkSum` function declared above.

```c
// Declare and initialize an array called Data
```

**Part c)** Call the `ChkSum` function and pass it the `Data` array and print the output it returns to the UART window (using printf). The output in the UART window must read: “Check Sum = XXX”. Where XXX is the value returned by the `ChkSum` function with `Data` as the input. You may declare any variables you need to accomplish the task.

```c
// Line or lines of code that calls ChkSum and prints
printf("Check Sum = %02hhx\n", ChkSum(Data));
```
ARM Assembly and C Operators

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
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
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 0xXYYYYYY

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

General purpose registers

Stack pointer
Link register
Program counter

256k Flash ROM
0x0000.0000
0x0003.FFFF
0x2000.0000
0x2000.7FFF
0x4000.0000
0x400F.FFFF
0xE000.0000
0xE004.1FFF
32k RAM
I/O ports
Internal I/O
PPB