

Exam 1**Date:** October 3, 2013

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	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. State the term that is best described by each definition.

Part a) A last in first out data storage system on the computer used to remember data temporarily.

Part b) Software is added to the system for the purpose of debugging, and this software has a large and significant effect on the system.

Part c) This C operator is used to perform a right shift.

Part d) The name given to describe 1,000 (10^3) bytes.

Part e) A type of logic where the voltage representing false is less than the voltage representing true.

Part f) A property of ROM such that data is not lost if power is removed and then restored.

Part g) This addressing mode is always used to access memory, shown here as the source operand of this instruction: `LDR R1, [R0]` ?

Part h) This declaration is used to create a variable in C that has a precision of 8 bits and can take negative values.

Part i) This C operator is used in if-then while-loop and do-while-loops for checking to see if two numbers are not equal.

Part j) A drawing that describes the sequence of operations of software, defining what and when software actions will occur.

Question 2 (10 points) Consider the following 8-bit subtraction (assume registers are 8 bits wide)

Load **0xC0** into **R1**

Load **0x1F** into **R2**

Subtract **R3 = R1-R2**

a. What will be the 8-bit result in Register R3 (in hex)?

b. What is 8-bit result in Register R3 (as an unsigned decimal)?

c. What is 8-bit result in Register R3 (as a signed decimal)?

d. What will be the value of the carry (C) bit?

e. What will be the value of the overflow (V) bit?

(10) Question 3. You will fill in the blanks of this C code that initializes Port B. Make pins PB6, PB4, PB1 outputs. Make the pin PB0 an input. To get full credit, this code must be friendly. Partial credit can be obtained by writing code that works, but is not friendly. Mark the box “skip” if it is not required to be executed at that spot in the initialization. You will use the following 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))
```

```
GPIO_PORTB_DATA_R =  ;
```

```
SYSCTL_RCGCGPIO_R |=  ;
```

```
delay = SYSCTL_RCGCGPIO_R;    // allow time for clock to settle
```

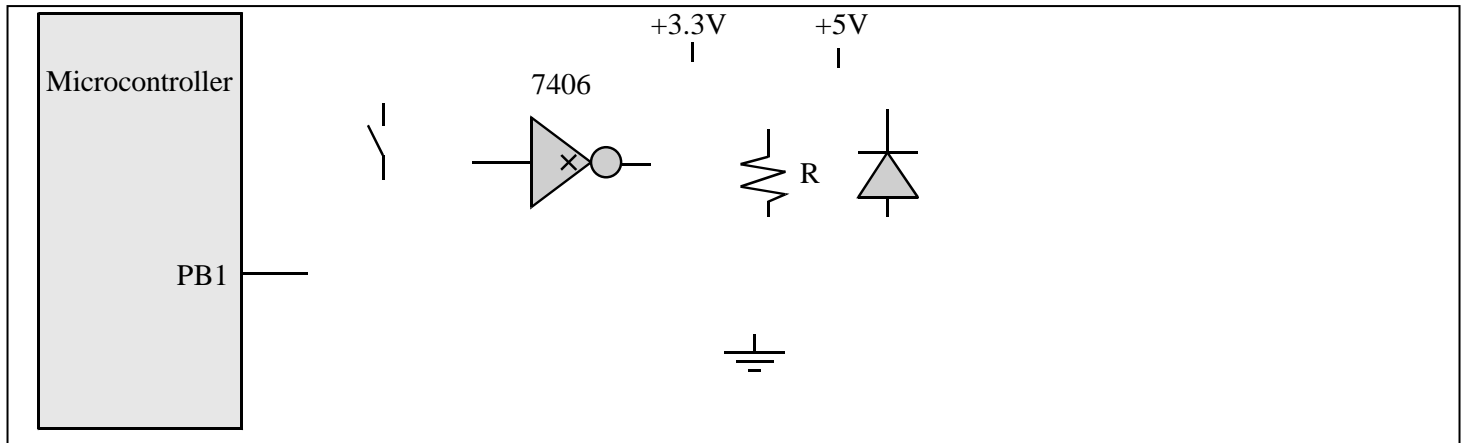
```
GPIO_PORTB_DIR_R &=  ;
```

```
GPIO_PORTB_DIR_R |=  ;
```

```
GPIO_PORTB_AFSEL_R &=  ;
```

```
GPIO_PORTB_DEN_R |=  ;
```

(10) Question 4. Interface the LED to PB1 such that if PB1 is high, the LED is on, and if PB1 is low the LED is off. The desired LED operating point is 3.0V at 20 mA. The V_{OH} of the microcontroller is 3.1 V. The V_{OL} of the microcontroller is 0.3 V. The maximum current that the microcontroller can source or sink is 8 mA. The V_{OL} of the 7406 is 0.5 V. The maximum current that the 7406 can sink is 40 mA. Your bag of parts includes the switch, the 7406, the LED, and a resistor (you specify the resistor value). Pick the fewest components to use. You will not need them all. You may also use 3.3V, 5V power, and/or ground. Show the equations used to calculate the resistor value.



(10) Question 5. Write an assembly subroutine, called **Calc**, that calculates $Output = (Input/8)-5$. The *Input* and *Output* parameters are 8-bit signed numbers located in global RAM. You may use Registers R0-R3, or R12 as scratch registers without saving and restoring them. Full credit will be given to the fastest solution. Don't worry about how *Input* is initialized, just read from *Input* and write to *Output*.

```

AREA    DATA, ALIGN=2
Input  SPACE  1
Output SPACE  1
AREA    |.text|, CODE, READONLY, ALIGN=2
    
```

(15) **Question 6.** Answer the following questions with reference to the C and assembly code below. You may assume that all linkages have been done to be able to call the assembly code from C.

Hint: Recall AAPCS

<pre> ; C code calling assembly int32_t Param1; int32_t Param2; int32_t Output; int main() { Param1 = 2; Param2 = 7; Output = Sub(Param1, Param2); } </pre>	<pre> ; Assembly code Bs RN 0 Res RN 1 Ex RN 4 Prod RN 5 Sub PUSH {R4,R5,LR} MOV Ex,#0 MOV Prod,Bs More CMP Prod,Res BGT Done MUL Prod,Prod,Bs ADD Ex,Ex,#1 B More Done MOV R0,Ex POP {R4,R5,LR} BX LR </pre>
--	--

(2) **Part a)** What is the numerical value in register R0 at the start of the assembly subroutine **Sub**?

R0 =

(2) **Part b)** What is the numerical value in register R1 at the start of the assembly subroutine **Sub**?

R1 =

(4) **Part c)** What is the numerical value of the C variable **Output** after the assignment statement, **Output = Sub(Param1, Param2);** is executed?

output =

(2) **Part d)** Why did the subroutine **Sub**, save the registers R4 and R5 on the stack?

- I. The input parameters are on the stack.
- II. The output parameter is returned on the stack
- III. In order to save the return address
- IV. Follows AAPCS convention
- V. None of the above.

(5) **Part e)** Which of the following statements describes what **Sub** does accurately?

- I. Sub returns the product of the two inputs using successive addition
- II. Sub returns the largest power to which the first input can be raised and still have it less than or equal to the second
- III. Sub returns the exponent of the first input raised to the second input
- IV. Sub returns the smallest power to which the first input needs to be raised so that it is greater than or equal to the second
- V. Sub returns the power to which the first input has to be raised to be equal to the second

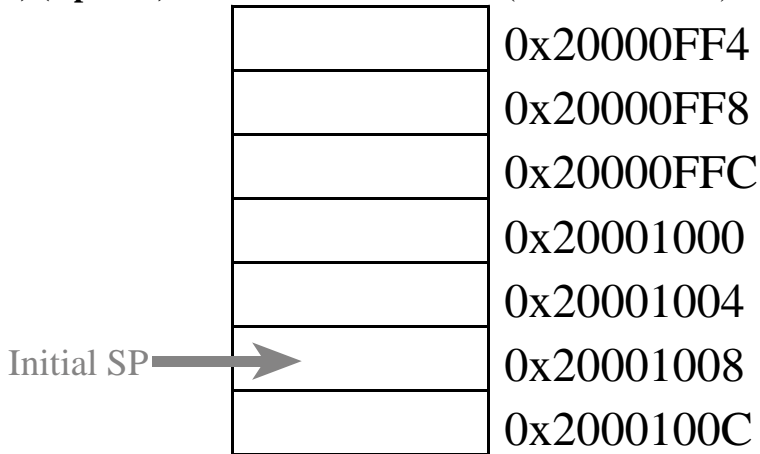
(20) Question 7. You are asked to develop the software for a control panel of a home automation system. *You can write software in either assembly or C.* Make sure that all of your software is friendly and follows the AAPCS. You may assume the hardware is already connected, and Port B is already initialized so PB5 is an output and PB4–0 are inputs. Please use the port definition `GPIO_PORTB_DATA_R` to access Port B. You are not allowed to use bit-specific port addressing. The system has five door/window switches (sensors) connected to pins PB4, PB3, PB2, PB1, PB0. Door/window signals are high if OK, and low if there is danger. There is an LED connected to pin PB5, which signifies an alarm. The LED interface is negative logic. Write the main program of the control panel that continuously checks the sensors and turns the warning LED connected if, and only if, two or more door/window switches indicate there is danger.

(10) Question 8. Show the contents of the stack after the two marked points in the execution of the following code. Assume R0=0, R1=10, R2=20, R3=30, R4=40, R5=50, and R6=60. The initial stack pointer is 0x20001008.

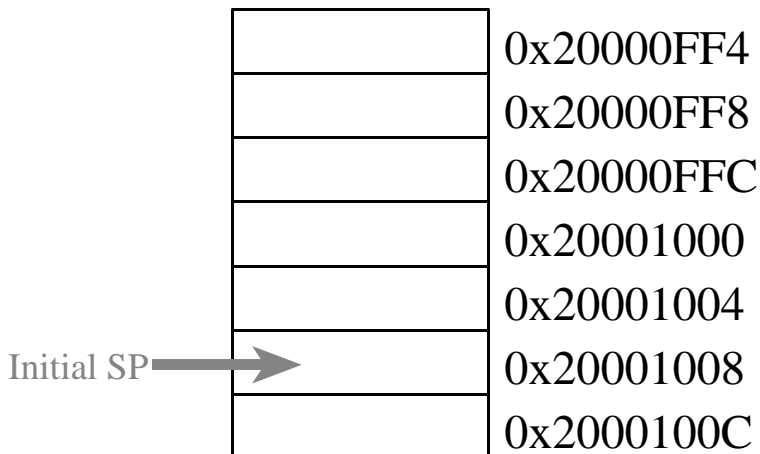
```

PUSH {R2,R3}
ADD R4,R1,R0 ; <---- A
POP {R5,R6}
ADD R5,R5,R4
ADD R6,R6,R5
PUSH {R0,R4-R6};
<---- B
    
```

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



b) (6 points) The contents of the stack (SP and contents) after execution point B:



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

