Exam 1

Date: Feb 25, 2016

UT EID: Solution Professor (circle): Janapa Reddi, Tiwari, 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.
- *Please read the entire exam before starting.*

Problem 1	10	
Problem 2	6	
Problem 3	16	
Problem 4	15	
Problem 5	10	
Problem 6	16	
Problem 7	12	
Problem 8	15	
Total	100	

(10) Question 1. State the term, symbol, instruction or expression that best answers the question.

(1) Part a) What bit gets set during the execution of the ADDS instruction to signify unsigned overflow?

C bit

(1) Part b) Mathematical relationship between the voltage across, the current through, and the dissipated power for an LED.

P = V*I

(1) Part c) This data-type is the most appropriate one to create a variable in C that can take values in the range -40,000 to +40,000.

`long` or `int32_t`

(1) Part d) According to ARM Architecture Procedure Call Standard, which registers can the callee function modify (without saving and restoring)?

R0, R1, R2, R3,R12

(1) Part e) A type of circuit that has two output states, low and off.

Open collector or 7406

(1) Part f) The name given to describe 1,024 (2^{10}) bytes.

kibibyte

(1) Part g) Addressing mode used in this instruction:

`LDR R0,=GPIO_PORTA_DATA_R`

indexed or PC relative

(1) Part h) What is the difference between these two instructions?

`PUSH {R1,R2,R3}` and `PUSH {R3,R2,R1}`

Nothing, they are the same

(1) Part i) If you multiply an n-bit signed number by an m-bit signed number, what is the maximum number of bits in the product? Assume $n \geq m$.

$n+m$

(1) Part j) What is the C operator that performs a Boolean AND. In other words it takes true/false inputs and generates a true/false output?

&&

(3) Part j) Write assembly code to create a 32-bit global variable called **`count`** in RAM, and a 8-bit constant called Max in ROM with a value of 255.

```
        AREA    |.text|,CODE,READONLY,ALIGN=2
Max     DCB   255

        AREA    DATA,ALIGN=2
count   SPACE 4
```

(6) Question 2. Assume the value is 8 bits. The binary is 11000001.
What is the value as unsigned hexadecimal?

0xC1

What is the value as unsigned decimal?

128+64+1 = 193

What is the value as signed decimal?

-128+64+1 = -63

(8) Question 3a Assume **`Data`** is an 8-bit signed variable in RAM. Write assembly code that divides the value of this variable by 8 using the shift operation, storing the result back in Data.

```
  LDR   R0,=Data   ; pointer to Data
  LDRSB R1,[R0]    ; value of Data, promoted to 32 bits
  ASR   R1,R1,#3   ; signed shift right
  STRB  R1,[R0]    ; store back
```

(8) Question 3b Write an assembly subroutine called **`Decr`**, which has one input and one output. Pass parameters using AAPCS. Assume the input is a 32-bit signed number. The function should decrement the input value with the exception that it will not decrement if the input is already at the smallest possible negative number, -2,147,483,648. This exception prevents the error where decrementing a negative value would have resulted in a positive number. The function returns the 32-bit signed result.

```
Decr CMP  R0,#-2147483648
     BEQ  skip
     SUB  R0,R0,#1
skip BX   LR

Decr SUBS R0,R0,#1
     BVC  ok
     ADD  R0,R0,#1
ok   BX   LR
```

(15) Question 4. Consider the following C function with one input and one output.

```
int32_t x;
int32_t func(int32_t in){
  int32_t out=0;
  while(in >=  0){
    out = out + in;
    in = in - 2;
  }
  return out;
}
```

(5) Part a) If we were to execute `x=func(5);` what would be the value of x?

in	out (at the out += in)
5	5
3	8
1	9

x = 9

(10) Part b) Write `func` in assembly using AAPCS

```
;Common mistakes:
; 1) AAPCS input parameter in R0, output parameter in R0
; 2) this is a while loop (must check first then do body)
;    if input were 0 or -2, then output should have been 0

func MOV R1,#0      ; R1= out=0
loop CMP R0,#0      ; R0= in
     BLT done       ; must be signed branch
     ADD R1,R1,R0   ; out = out+in
     SUB R0,R0,#2   ; in = in-2
     B   loop
done MOV R0,R1      ; AAPCS return in R0
     BX  LR

func MOV R1,R0      ; R1=in
     MOV R0,#0      ; R0=out=0
loop CMP R1,#0
     BLT done       ; must be signed branch
     ADD R0,R0,R1   ; out = out+in
     SUB R1,R1,#2   ; in = in-2
     B   loop
done BX  LR         ; AAPCS return in R0
```

(10) Question 5. You are to write a friendly port initialization subroutine in assembly or C, for an embedded system that uses all pins of Port A. Pins 0-3 of Port A are interfaced to negative logic input switches and pins 4-7 are interfaced to positive logic output LEDs. The device registers that are given to you are (you may not need all):

```
SYSCTL_RCGCGPIO_R       EQU   0x400FE608
GPIO_PORTA_DATA_R       EQU   0x400043FC
GPIO_PORTA_DIR_R        EQU   0x40004400
GPIO_PORTA_AFSEL_R      EQU   0x40004420
GPIO_PORTA_PUR_R        EQU   0x40004510
GPIO_PORTA_PDR_R        EQU   0x40004514
GPIO_PORTA_DEN_R        EQU   0x4000451C
```

Note that you are given four appropriately sized external resistors for the LEDs but no external resistors for the switches. You do not have to set **AMSEL** or **PCTL**.

```
void PortA_Init(){
  volatile uint32_t delay;
  SYSCTL_RCGCGPIO_R |= 0x01;
  delay = 10;
  GPIO_PORTA_DIR_R = 0xF0;
  GPIO_PORTA_AFSEL_R = 0x00;
  GPIO_PORTA_PUR_R = 0x0F;
  GPIO_PORTA_DEN_R= 0xFF;
}
```

(16) Question 6. Assume the value of the Stack pointer (SP) is **`0x20001000`** when the following code sequence starts execution (i.e., **`PC=0x00001000`**). The initial stack contents are given on the right.

```
0x00001000       POP {R0-R2}
0x00001004       ADD R4,R0,R1
0x00001008       BL  Func          <- B
0x0000100C       ...
...
0x00002000 Func  PUSH {LR,R4}      <- A
0x00002004       MOV  R4,R2
0x00002008       MUL  R0,R1
0x0000200C       ADD  R0,R4
0x00002010       POP {R4,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:

0x20000FF4	1
0x20000FF8	2
0x20000FFC	3
0x20001000	4
0x20001004	9
0x20001008	0x0000100C
0x2000100C	7

SP = 0x20001004

(10) Part b) Give the state of the stack (SP and contents) while executing the instruction at memory location 0x0000100C as shown by the arrow B and the values stored in R0, R1, R2 and R4.

0x20000FF4	1
0x20000FF8	2
0x20000FFC	3
0x20001000	4
0x20001004	9
0x20001008	0x0000100C
0x2000100C	7

SP = 0x2000100C

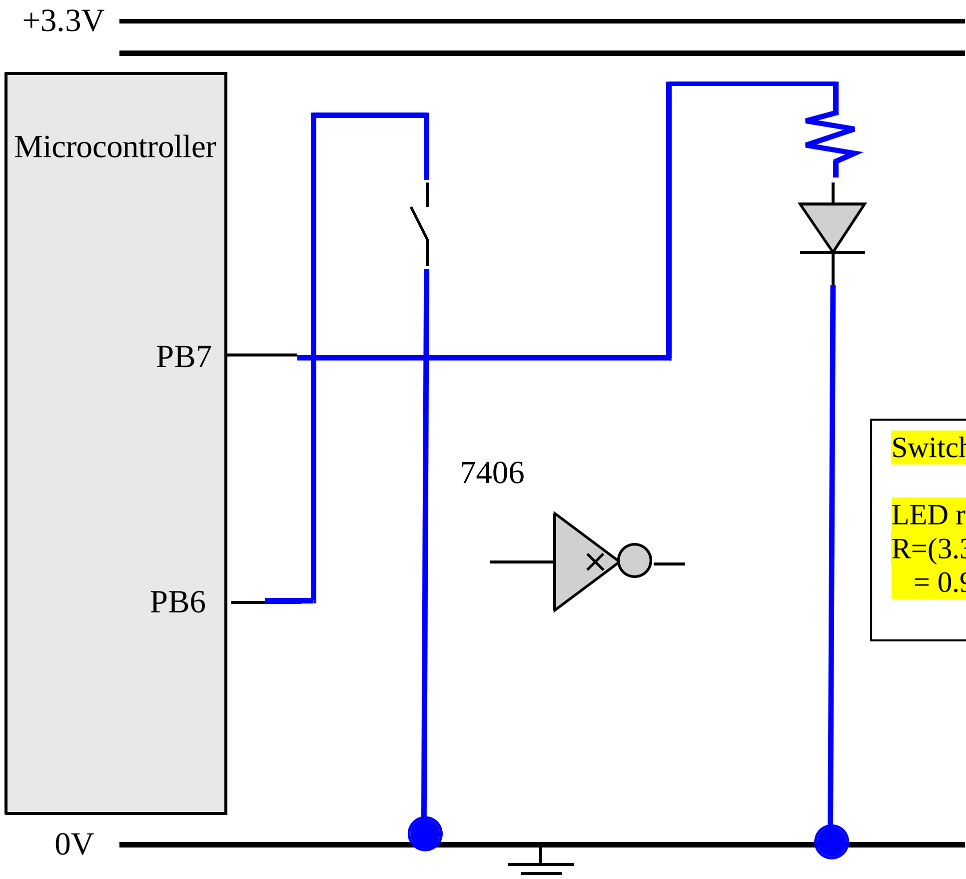
R0 = 26

R1 = 5

R2 = 6

R4 = 9

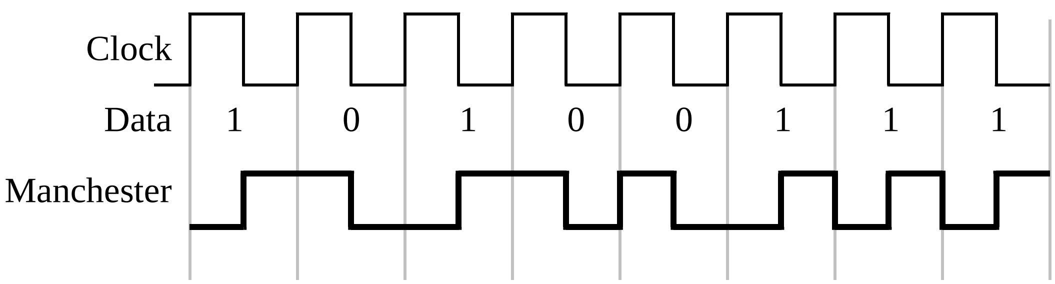
(12) Question 7. Interface the LED to Port B bit 7 (PB7) using positive logic. Connect a switch to PB6 using negative logic. The microcontroller's output voltage high is 3.3V. The LED is operating point is 2.4V at 6mA. The V_{OL} for the 7406 driver is 0.6V. Pick resistors appropriately and assume you have 5V, 3.3V, and ground to which you can connect your components. The symbols for each part are given below for your convenience – *use the minimum number of parts to construct the interfaced system.*



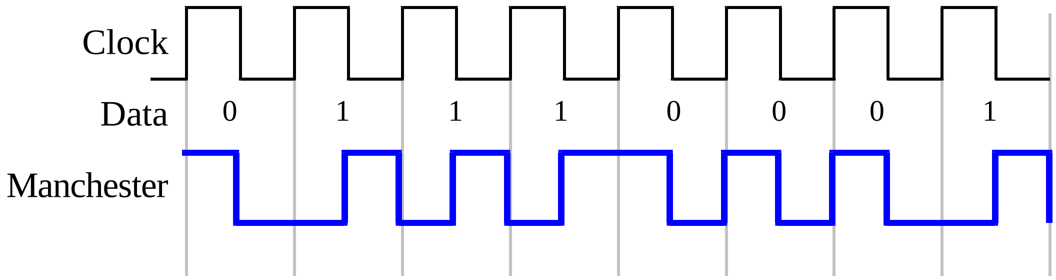
Switch uses internal resistor,

LED resistor is
R=(3.3-2.4V)/6mA
= 0.9V/6mA = 150 ohms

(15) Question 8. You are hired to design communication software for an embedded system. Your job is to implement the software logic for transmitting data using "Manchester encoding," a method to transmit bits between sender and receiver systems using edge transitions. You are given a "transmission *Clock*" (an input to the controller, separate from the CPU clock) and a *Data* value (e.g., 10100111). You have to generate a *Manchester Output* waveform on a port pin. In general Manchester encoding follows *Clock* XOR *Data* = *Manchester Output*



(3) Part a) Assuming you want to transmit the 8-bit data sequence 01110001, draw a similar diagram to the above showing the corresponding *Manchester output*.



(12) Part b) You will write a routine that transmits 8 bits of data in C. The input to this function is an 8-bit unsigned byte containing the data to be transmitted. The *Clock* input is connected to **PA1,** and the *Manchester output* is connected to **PA0**. Assume software has already initialized **PA1** and **PA0** as input and output respectively. To send one bit, wait for the *Clock* to go from low to high, set the **PA0** output to be (*Data* XOR *Clock*), wait for the *Clock* to go low, and then set the **PA0** output to be (*Data* XOR *Clock*). To send one byte repeat this procedure 8 times, once for each bit. Output the most significant bit first. Your code need not be friendly.

```
void Manchester(uint8_t data){
  int i; uint32_t bit;
  for(i=7; i>=0; i--){    // 8 bits

    while((GPIO_PORTA_DATA_R&0x02)==0){};   // wait for rise
    bit = (data>>i)&0x01; // most significant first
    GPIO_PORTA_DATA_R = bit^0x01;
    while((GPIO_PORTA_DATA_R&0x02)==0x02){};  // wait for fall
    GPIO_PORTA_DATA_R = bit;


  }

}
```

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`

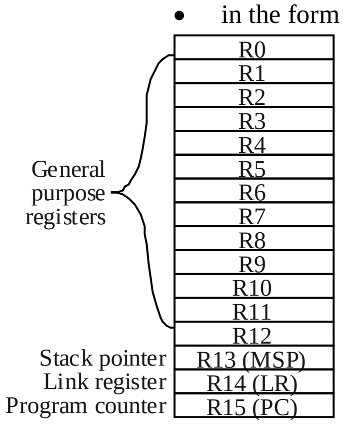
```
     value   any 32-bit value: signed, unsigned, or address
     {S}     if S is present, instruction will set condition codes
     #im12   any value from 0 to 4095
     #im16   any value from 0 to 65535
     {Rd,}   if Rd is present Rd is destination, otherwise Rn
     #n      any value from 0 to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