First:_________________ Last:_____________________

This is a closed book exam. You must put your answers on pages 1,2,3,4 only. You have 50 minutes, so allocate your time accordingly. Show your work, and put your answers in the boxes. ***Please read the entire quiz before starting.***

(4) Question 1. Digital logic currently uses binary because it is fast, low power, and very small. In the future, an EE319K student invents ternary logic that is faster, smaller and lower power than binary. This means each ternary bit can be 0, 1, or 2. Ternary means base 3 in the same way binary means base 2. What is the value of the unsigned four-digit ternary number 1201? Give your answer as a decimal number. ---

(3) Question 2. Answer true/false for each of the following three statements

Part a) Flash EEPROM memory on the TM4C123 is volatile. --------------------

Part b) I add three 32-bit numbers by executing **`ADD`** twice. The order in which I add the numbers affects the final value of the carry bit. ------------------

Part c) Dropout error can occur on a logical right shift (e.g., **`LSR`**). ------------

(4) Question 3. Consider the following 8-bit subtraction (assume registers are 8 bits wide)

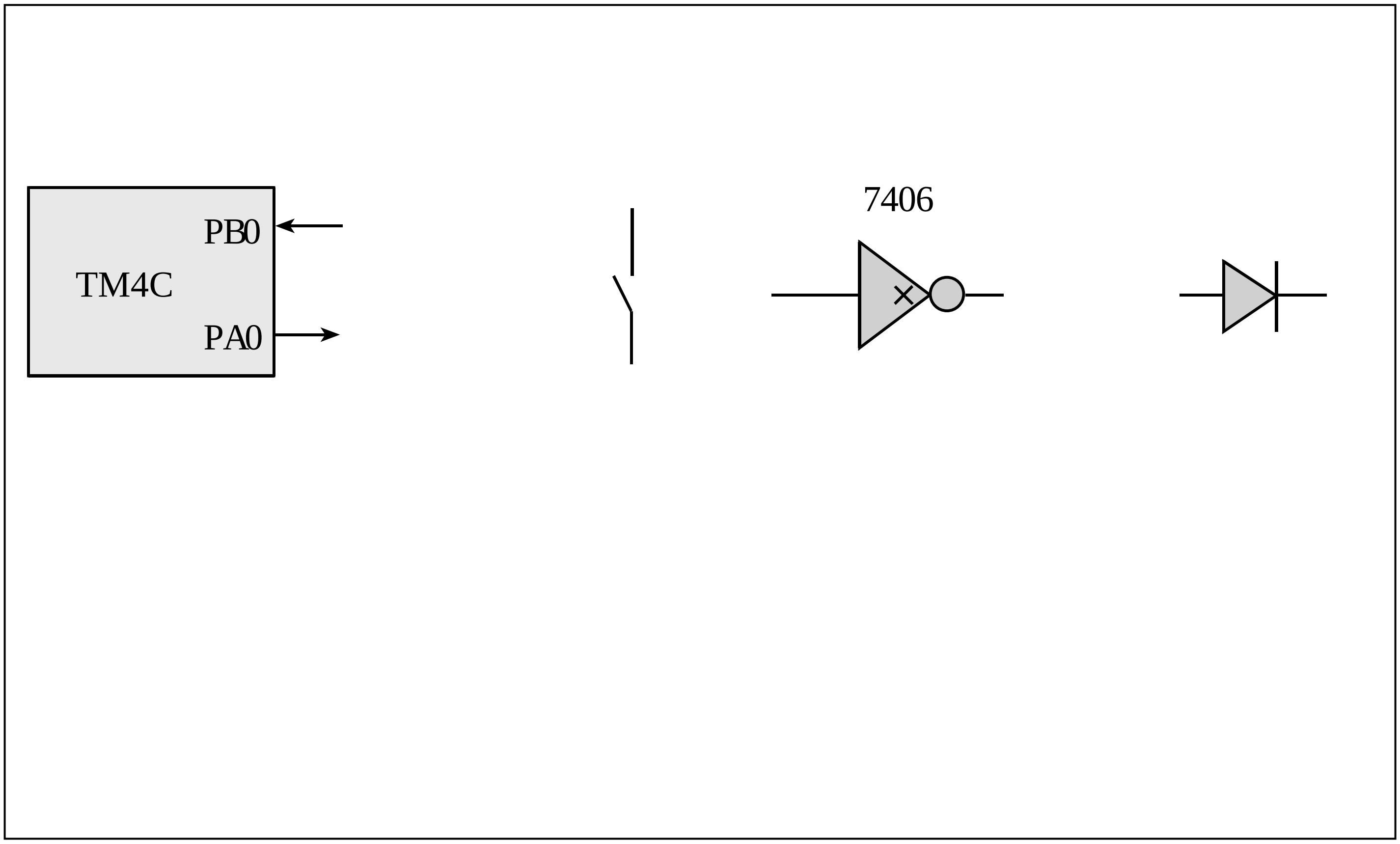
```
Load -100 into R1
Load +50  into R2
Subtract  R3 = R1-R2
```

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

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

(4) Question 4. What is the binary representation of 8-bit signed number -10?

(20) Question 5. Interface the LED to PA0. The desired LED operating point is 2.0V at 25 mA. At 25 mA you can assume the V_{OL} of the 7406 will be 0.5 V. Interface the switch to PB0 using positive logic. No software is required in this question, and you may assume PA0 is an output and PB0 an input. Your bag of parts includes the switch, the 7406, the LED, and one resistor each of the values {1Ω, 10Ω, 100Ω, 1kΩ, 10kΩ, 100kΩ and 1MΩ}. Pick the best resistors to use (you will not need them all.)



For questions 6 7, and 8, don't worry about establishing the reset vector, creating a main program, or initializing the stack pointer. You may use RAM-based global variables. Include comments. You may use the following definitions

```
GPIO_PORTA_DATA_R  EQU 0x40004080
GPIO_PORTA_DIR_R   EQU 0x40004400
GPIO_PORTA_AFSEL_R EQU 0x40004420
GPIO_PORTA_DEN_R   EQU 0x4000451C
SYSCTL_RCGCGPIO_R     EQU 0x400FE608
SYSCTL_RCGCGPIO_GPIOA EQU 0x00000001   ; port A Clock Gating Control
```

(20) Question 6. Assume seven positive logic switches are connected to PA6-PA0, and one LED is connected to PA7. Assume the direction register is properly initialized. Write an assembly language subroutine that sets PA7=1, if PA0=1, PA2=0, and PA6=0, regardless of the other 4 switches. For all other patterns of input switches, do not change the PA7 output.

; You should be able to write this in both C and assembly

(20) Question 7. Write an assembly language subroutine that adds two unsigned 32-bit numbers. The two inputs are passed in Register R0 and Register R1, and the result is returned in Register R0. Implement ceiling, such that if the sum is too big for 32 bits, return 0xFFFFFFFF.

; You should be able to write this in both C and assembly

(20) Question 8. Write an assembly language subroutine that counts the number of binary bits that are zero in a 32-bit number. The 32-bit input parameter is passed in Register R0 and the result is returned in Register R1. For example, if Register R0 = 0x00000001, return Register R1=31 because there are 31 binary zeros. If Register R0 = 0xFF0F0FFF, return Register R1 =8 because there are 8 binary zeros.

; You should be able to write this in both C and assembly

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`

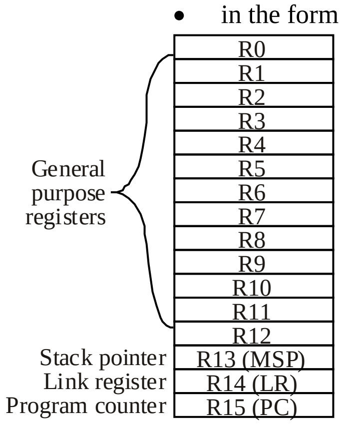
```
     value   any 32-bit value: signed, unsigned, or address
     {S}     if S is present, instruction will set condition codes
     #im12   any value from 0 to 4095
     #im16   any value from 0 to 65535
     {Rd,}   if Rd is present Rd is destination, otherwise Rn
     #n      any value from 0 to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