Quiz 1 *Fun Times*

Date: October 5, 2012

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 calculators or any electronic devices (turn cell phones off).
- You must put your answers on pages 2-6 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.*

(3) Question 1. Which of the following statements is most true? If we wished to reduce the power consumption used by our microcontroller
A) we could decrease the operating voltage?
B) we could decrease the frequency of the bus clock?
C) we could clear bits in the DEN register for unused pins?
D) none of A B or C is correct
E) A B and C are all correct

(4) Question 2. Digital logic currently uses binary because it is fast, low power, and very small. In the future, an EE319K student invents quinary logic that is faster, smaller and lower power than binary. This means each quinary digit can be 0, 1, 2, 3, or 4. Quinary means base 5 in the same way binary means base 2. What are the three **basis** elements of unsigned three-digit quinary number? Give your answers as a decimal numbers.

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

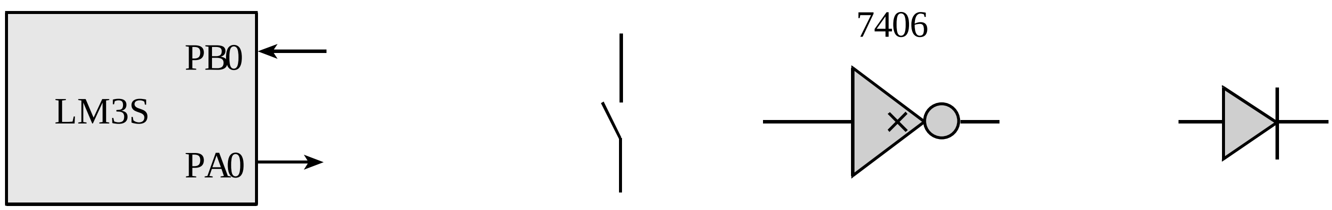
```
Load 0x9C into R1
Load 0x32 into R2
Subtract  R3 = R1-R2
```

What will be the 8-bit result in Register R3? ---

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

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

(20) Question 4. Interface the LED to PA0 in positive logic. The desired LED operating point is 1.0V at 20 mA. At 20 mA you can assume the V_{OL} of the 7406 will be 0.5 V. Assume the output high voltage of the microcontroller is 3.2 V and the output low voltage is 0.1V. Interface the switch to PB0 using negative logic. No software is required in this question, and you may assume PA0 is an output and PB0 an input. Assume the pull-up feature of PB0 will be activated by software. Your bag of parts includes the switch, the 7406, the LED, and resistors (you specify the values). Pick the fewest components to use (you may or may not need them all.) You may also use 3.3V, 5V power and ground.



(15) Question 5. Write an assembly subroutine that selects bit 9. The input to the subroutine is a 32-bit number in R0. The output in R0 is 0 if the input bit 9 is 0, and the output is 1 if the input bit 9 is 1.

(5) Question 6. Write C function that selects bit 9. The input to the function is an unsigned 32-bit number. The output of the function is 0 if the input bit 9 is 0, and the output is 1 if the input bit 9 is 1.

For questions 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. Bit-specific addressing is allowed but not required. You may use the following definitions

```
GPIO_PORTB_DATA_R  EQU 0x400053FC
GPIO_PORTB_DIR_R   EQU 0x40005400
GPIO_PORTB_AFSEL_R EQU 0x40005420
GPIO_PORTB_DEN_R   EQU 0x4000551C
SYSCTL_RCGC2_R     EQU 0x400FE108
SYSCTL_RCGC2_GPIOB EQU 0x00000002   ; port B Clock Gating Control
```

(10) Question 7. Fill in the boxes with hexadecimal numbers that initialize Port B. Bits 0, 1, and 2 are input. Bits 4 and 6 are output.

```
PortB_Init
    LDR R1, =SYSCTL_RCGC2_R
    LDR R0, [R1]
    ORR R0, R0, #--------------------------------------------------
    STR R0, [R1]
    NOP
    NOP
    LDR R1, =GPIO_PORTB_DIR_R
    LDR R0, [R1]                                               /--
    ORR R0, R0, #---------------------------------------------/
    BIC R0, R0, #-------------------------------------------\
    STR R0, [R1]                                             \---
    LDR R1, =GPIO_PORTB_AFSEL_R
    LDR R0, [R1]
    BIC R0, R0, #--------------------------------------------------
    STR R0, [R1]
    LDR R1, =GPIO_PORTB_DEN_R
    LDR R0, [R1]
    ORR R0, R0, #--------------------------------------------------
    STR R0, [R1]
    BX  LR
```

(30) Question 8. Write an assembly language main program that first calls the initialization and then performs steps 2, 3, and 4 over and over infinitely.

1) execute **`PortB_Init`** defined in Question 7.
2) read the inputs;
3) if all three inputs are equal to each other (inputs are 000 or 111) then toggle output bit 4,
4) otherwise (inputs are 001, 010, 011, 100, 101, or 110) toggle output bit 6.

Write friendly code. Comments are allowed but not needed.

(10) Question 9. Write a C language main program that first calls the initialization and then performs steps 2, 3, and 4 over and over infinitely.

1) execute **`PortB_Init();`** defined in Question 7.
2) read the inputs;
3) if all three inputs are equal to each other (inputs are 000 or 111) then toggle output bit 4,
4) otherwise (inputs are 001, 010, 011, 100, 101, or 110) toggle output bit 6.

Write friendly code. Comments are allowed but not needed. With this definition

```
#define PORTB   (*((volatile unsigned long *)0x400053FC))
```

You will be able to read and write to Port B. For example

```
     n = PORTB;  // reads all 8 bits of Port B into variable n
     PORTB = m;  // write all 8 bits of Port B with data from m
```

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 and 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`

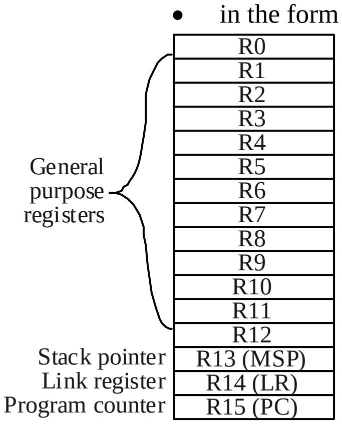
```
value   any 32-bit value: signed, unsigned, or address
{S}     if S is present, instruction will set condition codes
#im12   any value from 0 to 4095
#im16   any value from 0 to 65535
{Rd,}   if Rd is present Rd is destination, otherwise Rn
#n      any value from 0 to 31
#off    any value from -255 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