# Quiz 1 *Fun Size*

**Date:** February 23, 2012

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 calculators or any electronic devices (turn cell phones off).
- You must put your answers on pages 2-6 only.
- You have 75 minutes, so allocate your time accordingly.
- Show your work, and put your answers in the boxes.
- *Please read the entire quiz before starting.*

**(5) Question 1.** What is the value of the unsigned four-digit hexadecimal number 0x1210? Give your answer as a decimal number. ----------------------------------------------------------------

**(6) Question 2.** For each of the following statements fill in the word or phase that matches best

**Part a)** A drawing with circles (programs) and rectangles (hardware) where the arrows illustrate the type, direction and amount of data being transferred. ---------------

**Part b)** The subset of elements from which the entire set can be created.--------

**Part c)** A computer system where the I/O devices are accessed in a similar way as memory is accessed (i.e., using the same instructions). ----------

**(6) Question 3.** Consider the following instruction

```
ADD R0,R1,R2
```

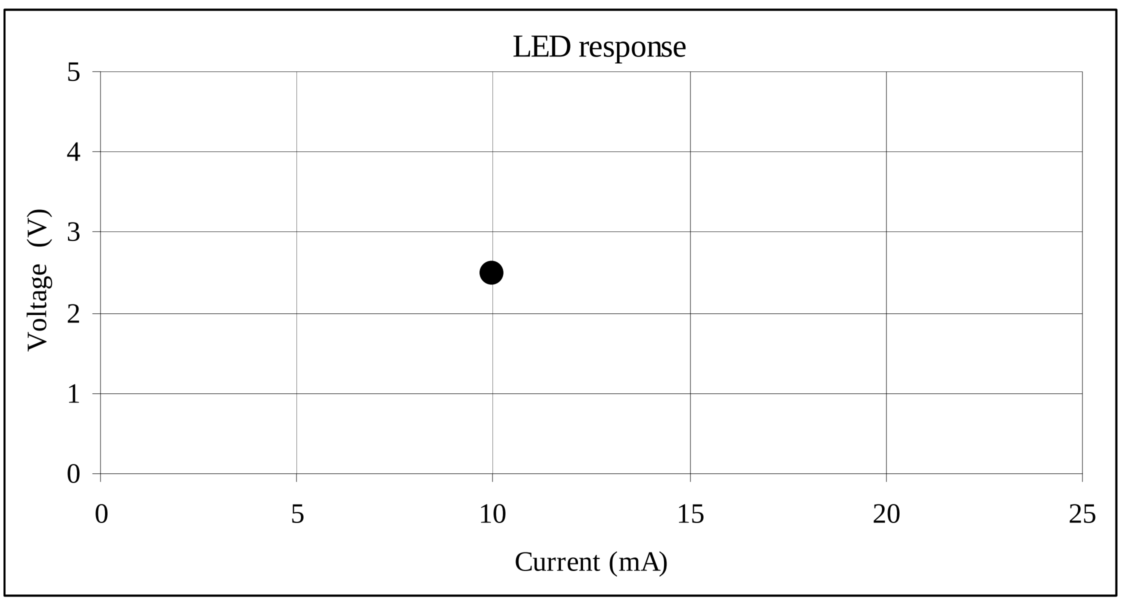
What does it mean if the overflow (V) bit is set?

What does it mean if the carry (C) bit is set?

**(5) Question 4.** A 30-bit number is approximately how many decimal digits?

**(10) Question 5.** Interface the switch to PA0 using positive logic (pressed is high, not pressed is low). No software is required in this question, and you may assume PA0 is an input. Your bag of parts includes the switch, the 7406, 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.) Use the 7406 only if it is absolutely needed. Assume $V_{OL}$ of the 7406 is 0.5 V.


**(5) Question 6.** You are given an LED with a desired operating point of 2.5V at 10 mA. Sketch the approximate voltage versus current relationship for this diode.



~~**(5) Question 7.** Assume PC is $5000, and Register B is $34. You may assume location $0001 contains $12. Show the simplified bus cycles occurring when the `subb` instruction is executed. In the "**changes**" column, specify which registers get modified during that cycle, and the corresponding new values. Do not worry about changes to the CCR. *Just show the one instruction.*~~

~~`$5000 D001      subb $0001`~~

| ~~R/W~~ | ~~Addr~~ | ~~Data~~ | ~~Changes to A,B,X,Y,S,PC,IR,EAR~~ |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**(4) Question 8.** Consider the following piece of code. Assume the PC is initially 0x00000134, and the stack pointer is initially 0x20000408.

```
0x00000134 F04F0001  Start  MOV  r0,#0x01
0x00000138 F000F807         BL   Test     ;0x0000014A
0x0000013C    ;next instructions

0x0000014A B500      Test   PUSH {lr}
0x0000014C 4400             ADD  r0,r0,r0
0x0000014E BD00             POP  {pc}
```

Think about how this program executes up to and including the execution of **`ADD`**

Fill in specific hexadecimal bytes that are pushed on the stack.

Using an arrow, label to which box the SP points.

What is the value of PC, LR, R0, and SP after the ADD instruction is executed.

**(4) Question 9.** Show the C code to create a variable named **`Position`** with range -128 to +127?

**(10) Question 10.** Write assembly code to swap R0, R1, and R2 (R0 goes to R1, R1 goes to R2, and R2 goes to R0). You must use the stack and cannot use any global variables or other registers. You do not need to set the reset vector or initialize the stack in this question.

**(20) Question 11.** Assume two positive logic switches are connected to PA2 and PA0, and one positive logic LED is connected to PA5. Write an assembly language program (start, initialization, loop) that turns on the LED if exactly one of the two switches is on. Turn off the LED if neither or both switches are pressed. After initializing the port, the input from switches and output to LED will be performed over and over continuously. Your code must have comments and be written in a **friendly** manner. 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_RCGC2_R     EQU 0x400FE108
SYSCTL_RCGC2_GPIOA EQU 0x00000001   ; port A Clock Gating Control
```

**(20) Question 12.** Write a C program that controls a kidney dialysis pump. Port G is an 8-bit output that adjusts power to the pump. The range is 0 (no power) to 255 (full power). Port H is an 8-bit input

that contains the measured blood flow in ml/min. The range is 0 (no flow) to 255 ml/min. The goal is to pump blood at 150 ml/min. If the measured flow is less than 150 ml/min, increase the power by 1 unit. If the measured flow is more than 150 ml/min, decrease the power by 1. Implement ceiling and floor (do not let the power go above 255 or below 0). First initialize Port G and Port H, then run the pump controller over and over continuously. You may use the symbols
GPIO_PORTG_DATA_R, GPIO_PORTG_DIR_R, GPIO_PORTG_AFSEL_R,
GPIO_PORTG_DEN_R, GPIO_PORTH_DATA_R, GPIO_PORTH_DIR_R,
GPIO_PORTH_AFSEL_R, GPIO_PORTH_DEN_R, SYSCTL_RCGC2_R (set bits 6 and 7).
To adjust power to the pump, write 8 bits to Port G. To measure the flow, read 8 bits from Port H.

**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`

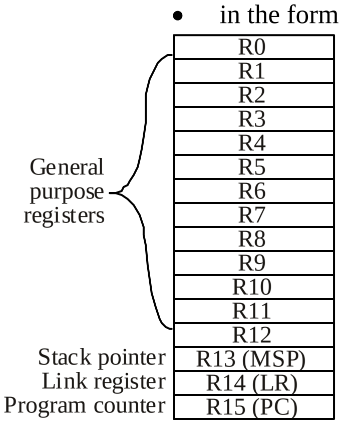
```
     value   any 32-bit value: signed, unsigned, or address
     {S}     if S is present, instruction will set condition codes
     #im12   any value from 0 to 4095
     #im16   any value from 0 to 65535
     {Rd,}   if Rd is present Rd is destination, otherwise Rn
     #n      any value from 0 to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