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 phrase 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 \{1\Omega, 10\Omega, 100\Omega, 1k\Omega, 10k\Omega, 100k\Omega and 1M\Omega\}. 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
```

<table>
<thead>
<tr>
<th>R/W</th>
<th>Addr</th>
<th>Data</th>
<th>Changes to A, B, X, Y, S, PC, IR, EAR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(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:

```c
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
```

### 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,

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
- **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

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 \quad \text{; logical shift right } Rd=Rm\ll n \quad \text{(unsigned)}

ASR\{S\} Rd, Rm, Rs \quad \text{; arithmetic shift right } Rd=Rm\ll Rs \quad \text{(signed)}

ASR\{S\} Rd, Rm, \#n \quad \text{; arithmetic shift right } Rd=Rm\ll n \quad \text{(signed)}

LSL\{S\} Rd, Rm, Rs \quad \text{; shift left } Rd=Rm\ll Rs \quad \text{(signed, unsigned)}

LSL\{S\} Rd, Rm, \#n \quad \text{; shift left } Rd=Rm\ll n \quad \text{(signed, unsigned)}

Arithmetic instructions

\text{ADD}\{S\} \{Rd,\} Rn, <op2> \quad \text{; } Rd = Rn + \text{op2}

\text{ADD}\{S\} \{Rd,\} Rn, \#im12 \quad \text{; } Rd = Rn + \text{im12}, \text{im12 is } 0 \text{ to } 4095

\text{SUB}\{S\} \{Rd,\} Rn, <op2> \quad \text{; } Rd = Rn - \text{op2}

\text{SUB}\{S\} \{Rd,\} Rn, \#im12 \quad \text{; } Rd = Rn - \text{im12}, \text{im12 is } 0 \text{ to } 4095

\text{RSB}\{S\} \{Rd,\} Rn, <op2> \quad \text{; } Rd = \text{op2} - Rn

\text{RSB}\{S\} \{Rd,\} Rn, \#im12 \quad \text{; } Rd = \text{im12} - Rn

\text{CMP} \quad Rn, <op2> \quad \text{; } Rn - \text{op2} \quad \text{sets the NZVC bits}

\text{CMN} \quad Rn, <op2> \quad \text{; } Rn - (-\text{op2}) \quad \text{sets the NZVC bits}

\text{MUL}\{S\} \{Rd,\} Rn, Rm \quad \text{; } Rd = Rn \times Rm \quad \text{signed or unsigned}

\text{MLA} \quad Rd, Rn, Rm, Ra \quad \text{; } Rd = Ra + Ra \times Rm \quad \text{signed or unsigned}

\text{MLS} \quad Rd, Rn, Rm, Ra \quad \text{; } Rd = Ra - Rn \times Rm \quad \text{signed or unsigned}

\text{UDIV}\{S\} \{Rd,\} Rn, Rm \quad \text{; } Rd = Rn/Rm \quad \text{unsigned}

\text{SDIV}\{S,\} \{Rd,\} Rn, Rm \quad \text{; } Rd = Rn/Rm \quad \text{signed}

Notes:

Ra Rd Rm Rn Rt represent 32-bit registers

\{S\} \quad \text{if } S \text{ is present, instruction will set condition codes}

\#im12 \quad \text{any value from } 0 \text{ to } 4095

\#im16 \quad \text{any value from } 0 \text{ to } 65535

\{Rd,\} \quad \text{if } Rd \text{ is present } Rd \text{ is destination, otherwise } Rn

\#n \quad \text{any value from } 0 \text{ to } 31

\#off \quad \text{any value from } -255 \text{ to } 4095

label \quad \text{any address within the ROM of the microcontroller}

op2 \quad \text{the value generated by } <op2>

Examples of flexible operand <op2> creating the 32-bit number. E.g., Rd = Rn+op2

\text{ADD} \quad Rd, Rn, Rm \quad \text{; } op2 = Rm

\text{ADD} \quad Rn, Rm, Rm, LSL \#n \quad \text{; } op2 = Rm\ll n \quad Rm \text{ is signed, unsigned}

\text{ADD} \quad Rn, Rm, Rm, LSR \#n \quad \text{; } op2 = Rm\ll n \quad Rm \text{ is unsigned}

\text{ADD} \quad Rn, Rm, Rm, ASR \#n \quad \text{; } op2 = Rm\ll n \quad Rm \text{ is signed}

\text{ADD} \quad Rn, Rm, \#constant \quad \text{; } op2 = \text{constant} \text{, where } X \text{ and } Y \text{ are hexadecimal digits:}

- \text{produced by shifting an 8-bit unsigned value left by any number of bits}
- \text{in the form } 0x00XY00XY
- \text{in the form } 0xXY00XY00
- \text{in the form } 0xXYXYXYXY

<table>
<thead>
<tr>
<th>Register</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>0x0000.0000</td>
</tr>
<tr>
<td>R1</td>
<td>0x0003.FFFF</td>
</tr>
<tr>
<td>R2</td>
<td>0x0000.0000</td>
</tr>
<tr>
<td>R3</td>
<td>0x2000.0000</td>
</tr>
<tr>
<td>R4</td>
<td>0x2000.FFFF</td>
</tr>
<tr>
<td>R5</td>
<td>0x0000.0000</td>
</tr>
<tr>
<td>R6</td>
<td>0x4000.0000</td>
</tr>
<tr>
<td>R7</td>
<td>0x41FF.FFFF</td>
</tr>
<tr>
<td>R8</td>
<td>0xE000.0000</td>
</tr>
<tr>
<td>R9</td>
<td>0xE004.0FFF</td>
</tr>
<tr>
<td>R10</td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td></td>
</tr>
<tr>
<td>R12</td>
<td></td>
</tr>
<tr>
<td>R13</td>
<td></td>
</tr>
<tr>
<td>R14</td>
<td></td>
</tr>
<tr>
<td>R15</td>
<td></td>
</tr>
</tbody>
</table>

Condition code bits

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>negative</td>
</tr>
<tr>
<td>Z</td>
<td>zero</td>
</tr>
<tr>
<td>V</td>
<td>signed overflow</td>
</tr>
<tr>
<td>C</td>
<td>carry or unsigned overflow</td>
</tr>
</tbody>
</table>