Exam 1

Date: February 22, 2013

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.
(15) **Question 1.** Select A, B, C, D, or E that best answers the question. Put answers in the boxes.

**What type of computer is the ARM Cortex M?**
A) Von Neumann
B) big endian
C) Harvard
D) CISC
E) none of the above

**Which of the following is a rule for the proper use of the stack?**
A) To pop, we first read at SP then increment the SP by 4
B) To push, first write at SP then decrement the SP by 4
C) It is ok to access memory locations above the SP (e.g., SP-8)
D) The software must execute something like LDR SP,=InitialSP to initialize the stack pointer
E) none of the above

**Why does the Cortex M have multiple buses?**
A) To reduce the number of mistakes software can make
B) To allow the microcontroller to access I/O ports
C) To speed up execution by allowing multiple actions to run in parallel
D) To reduce power saving energy
E) none of the above

**What is a data flow graph?**
A) A drawing with circles and rectangles. The circles are software modules and the rectangles are hardware. If module A invokes an operation in module B there is an arrow from A to B.
B) A drawing that describes the sequence of operations of software, defining what and when software actions will occur.
C) To describe how data are stored in a computer, we draw a picture or graph of how the data are organized.
D) A drawing with circles and rectangles. The circles are software modules and the rectangles are hardware. If a module A passes data to module B, then an arrow is drawn from A to B.
E) none of the above

**What is open collector logic?**
A) It is logic used when the microcontroller creates an input port.
B) Logic that has two states, high and low.
C) Logic that has two states, low and off.
D) Logic that has three states, high, low, and off.
E) none of the above
(15) **Question 2.** Interface the LED to PG0 such that if PG0 is low, the LED is on, and if PG0 is high the LED is off. The desired LED operating point is 1.0V at 2 mA. The $V_{OH}$ of the microcontroller is 3.1 V. The $V_{OL}$ of the microcontroller is 0.3 V. The maximum current that the microcontroller can source or sink is 8mA. The $V_{OL}$ of the 7406 is 0.5 V. The maximum current that the 7406 can sink is 40mA. Your bag of parts includes the switch, the 7406, the LED, and resistors (you specify the resistor values). Pick the fewest components to use. You will not need them all. You may also use 3.3V, 5V power and ground. Show the equations used to calculate the resistor value.
(10) Question 3. Write an assembly subroutine, called Mul7, that multiplies by 7 using just shifts, adds, and subtracts (not the MUL instruction). The input is passed by value in Register R0, and the output is returned in Register R0. You may use Registers R1, R2, R3, or R12 as scratch registers without saving and restoring them. Notice that if \( x \) is a variable, then \( 7x = 8x-x \), and also \( 7x = 4x+2x+x \).

(15) Question 4. State the term that is best described by each definition.

Part a) A property of ROM such that data is not lost if power is removed and then restored.

Part b) The error that occurs after an addition, subtraction, or multiplication such that the result does not fit back into the register.

Part c) A quality of a software system defined by how much memory is required.

Part d) The name given to describe 1,048,576 \( (2^{20}) \) bytes.

Part e) A type of logic where the voltage representing false is less than the voltage representing true.
(30) Question 5. Assume that Port G is already initialized such that PG0 is an output and PG2 is an input. You do not have to write the initialization code. Write a C function that toggles PG0 16 times. However, before each time your function toggles PG0, it should read the PG2 input. If PG2 is low, your function stops toggling and waits until PG2 becomes high. After PG0 has been toggled 16 times, your function should return. You may define additional variables. Bit-specific addressing is allowed but not required. You may use the following definition

```
#define GPIO_PORTG_DATA_R (*((volatile unsigned long *)0x400263FC))
```

This figure illustrates one possible case. PG0 toggles only while PG2 is high.

```
PG2 in _______ _______ _______
PG0 out _______ _______ _______
```

In this second possible case, the PG2 input is always high. In this case your function toggles PG2 16 times and then returns.

```
PG2 in __________________________ high
PG0 out _______ _______ _______ _______
```

In this third possible case, the PG2 input remains low forever. For this case, your function never toggles PG0 and never returns.

---

Jonathan Valvano      February 22, 2013       2:00pm-2:50pm
(15) Question 6. Consider the following assembly language system. The subroutine Circ calculates the circumference of a circle. The input is passed in using R0 in cm, and the output is returned in R0, also in cm. $2\pi$ is approximated by $6283185/1000000$. The system begins execution at Start. The assembly listing is shown.

```
0x000006CC B502      Circ      PUSH  {R1,LR}
0x000006CE 49FA                LDR   R1,=6283185
0x000006D0 F000F801            BL    Multiply
0x000006D4 BD02                POP   {R1,PC}
0x000006D6 FB00F001  Multiply  MUL   R0,R0,R1 ;R0*R1
0x000006DA 49F8                LDR   R1,=1000000
0x000006DC FBB0F0F1            UDIV  R0,R0,R1 ;return R0*R1/1000000
0x000006E0 4770                BX    LR
0x000006E2 F04F007B  Start     MOV   R0,#123  ;radius in cm
0x000006E6 F7FFFFF1            BL    Circ     ;returns 772 cm in R0
0x000006EA E7FE      Loop      B     Loop
```

Part a) In the first line of Circ, what does 0x000006CC represent?  

Part b) In the first line of Circ, what does B502 represent?  

Part c) What value is in the link register immediately after the BL Multiply instruction is executed?  

Part d) What addressing mode does the LDR R1,=6283185 instruction use?  

Part e) The system begins execution at Start and eventually calls the Multiply subroutine. Assume the stack is initially empty at the beginning of Start. In particular the SP has an initial value of 0x20001000. Assume initial value of R1 is 0 at the beginning of Start. During the execution of the Multiply subroutine, draw a stack picture illustrating all values on the stack, and the exact location of the SP at this time.

```
Initial SP
0x20000FF4
0x20000FF8
0x20000FFC
0x20001000
0x20001004
0x20001008
0x2000100C
```
Memory access instructions

LDR Rd, [Rn] ; load 32-bit number at [Rn] to Rd
LDR Rd, [Rn,#off] ; load 32-bit number at [Rn+offset] 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+offset] to Rd
LDRSH Rd, [Rn] ; load signed 16-bit at [Rn] to Rd
LDRSH Rd, [Rn,#off] ; load signed 16-bit at [Rn+offset] to Rd
LDRB Rd, [Rn] ; load unsigned 8-bit at [Rn] to Rd
LDRB Rd, [Rn,#off] ; load unsigned 8-bit at [Rn+offset] to Rd
STR Rt, [Rn] ; store 32-bit Rt to [Rn]
STR Rt, [Rn,#off] ; store 32-bit Rt to [Rn+offset]
STRH Rt, [Rn] ; store least sig. 16-bit Rt to [Rn]
STRH Rt, [Rn,#off] ; store least sig. 16-bit Rt to [Rn+offset]
STRB Rt, [Rn] ; store least sig. 8-bit Rt to [Rn]
STRB Rt, [Rn,#off] ; store least sig. 8-bit Rt to [Rn+offset]
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
MOVS Rd, <op2> ; set Rd equal to op2
MOV Rd, #im16 ; set Rd equal to im16, im16 is 0 to 65535
MVNS 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=Rs>>n (unsigned)

ASR(S) Rd, Rm, Rs ; arithmetic shift right Rd=Rs>>Rs (signed)

ASR(S) Rd, Rm, #n ; arithmetic shift right Rd=Rs>>n (signed)

LSL(S) Rd, Rm, Rs ; shift left Rd=Rs<<Rs (signed, unsigned)

LSL(S) Rd, Rm, #n ; shift left Rd=Rs<<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, #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

<table>
<thead>
<tr>
<th>General purpose registers</th>
<th>Stack pointer</th>
<th>Link register</th>
<th>Program counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R13 (MSP)</td>
<td>R14 (LR)</td>
<td>R15 (PC)</td>
</tr>
<tr>
<td>R1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Condition code bits

N negative
Z zero
V signed overflow
C carry or unsigned overflow

<table>
<thead>
<tr>
<th>256k Flash ROM</th>
<th>64k RAM</th>
<th>I/O ports</th>
<th>Internal I/O PPB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000.0000</td>
<td>0x0003.FFFF</td>
<td>0x2000.0000</td>
<td>0xE000.0000</td>
</tr>
<tr>
<td>0x0000.0000</td>
<td>0x0003.FFFF</td>
<td>0x2000.0000</td>
<td>0xE000.0000</td>
</tr>
<tr>
<td>0x2000.0000</td>
<td>0x2000.FFFF</td>
<td>0x4000.0000</td>
<td>0x41FF.FFFF</td>
</tr>
<tr>
<td>0x2000.0000</td>
<td>0x2000.FFFF</td>
<td>0x4000.0000</td>
<td>0x41FF.FFFF</td>
</tr>
<tr>
<td>0xE000.0000</td>
<td>0xE004.0FFF</td>
<td>0xE000.0000</td>
<td>0xE004.0FFF</td>
</tr>
</tbody>
</table>