Exam 1

Date: Feb 26, 2015

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 books, no papers, no data sheets (other than the last two pages of this Exam)
• No devices other than pencil, pen, eraser (no calculators, no electronic devices), please turn cell phones off.
• Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space (boxes) provided. Anything outside the boxes/blanks will be ignored in grading. You may use the back of the sheets for scratch work.
• You have 75 minutes, so allocate your time accordingly.
• For all questions, unless otherwise stated, find the most efficient (time, resources) solution.
• Unless otherwise stated, make all I/O accesses friendly.
• Please read the entire exam before starting.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem 1</td>
<td>10</td>
</tr>
<tr>
<td>Problem 2</td>
<td>6</td>
</tr>
<tr>
<td>Problem 3</td>
<td>14</td>
</tr>
<tr>
<td>Problem 4</td>
<td>10</td>
</tr>
<tr>
<td>Problem 5</td>
<td>20</td>
</tr>
<tr>
<td>Problem 6</td>
<td>5</td>
</tr>
<tr>
<td>Problem 7</td>
<td>5</td>
</tr>
<tr>
<td>Problem 8</td>
<td>15</td>
</tr>
<tr>
<td>Problem 9</td>
<td>15</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
</tr>
</tbody>
</table>
(10) Question 1. State the term, symbol, instruction or expression that best answers the question.

(1) Part a) The drawing with circles representing software modules. An arrow from circle A to circle B means software in A invokes a function in module B.

(1) Part b) This declaration is used to create a variable in C that can take on the values from 20 to +40000. Pick the most efficient format.

(1) Part c) You are writing a function with exactly three input parameters. According to ARM Architecture Procedure Call Standard, how should you pass the three parameters?

(1) Part d) According to ARM Architecture Procedure Call Standard, which registers must be preserved?

(1) Part e) The term used to define the amount of work that can be done. Units are Joules.

(1) Part f) The term that defines the subset of a number system from which all elements of that set can be derived.

(1) Part g) This C operator will perform the logic or of two Booleans (the inputs and outputs are True or False).

(3) Part h) Write the assembly code to create a 16-bit signed variable called Num. Include the details that will place the variable in RAM.
(6) **Question 2.** Complete the following table. Each row in the table contains an equal value expressed in binary, hexadecimal, unsigned decimal, and signed decimal. Assume each value is 8 bits.

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hexadecimal</th>
<th>Unsigned Decimal</th>
<th>Signed Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000001</td>
<td>0xFE</td>
<td>212</td>
<td>64</td>
</tr>
</tbody>
</table>

(14) **Question 3**

a. (4) Consider the following 8-bit addition (assume registers are 8 bits wide, and assume the condition code bits are set in a way similar to the Cortex M4). What are the condition code bits?

```
Load   0x88 into R1
Load   0xC8 into R2
Adds   R3 = R1+R2, setting the condition codes
```

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

b. (10) Complete the table below by marking with an X which branches will be taken or not taken as a result of instructions below. Assume registers are 8 bits wide, and assume the branch instructions and condition code bits are similar to the Cortex M4.

```
Load   #100 into R1
Load   #200 into R2
Subs   R3 = R1-R2 ; setting the condition codes
```

<table>
<thead>
<tr>
<th>Branch Instruction</th>
<th>Taken</th>
<th>Not Taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>B target</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BL target</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BEQ target</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNE target</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCS target</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCC target</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BVS target</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BVC target</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLO target</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BLT target</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(10) Question 4. Complete the assembly subroutine that initializes Port B. You should make PB7 PB0 outputs, and make PB4 PB1 inputs. This subroutine is called once at the start of execution of the system. All accesses to I/O registers must be friendly. Your subroutine will set the clock, direction, and enable registers. In this question do not worry about AFSEL, PUR, PDR, AMSEL, or PCTL. You must fill in the instruction or instructions for the following four boxes. Each box may contain 0, 1, or 2 instructions. Do not assume DIR, DEN or DATA registers have been cleared by the reset operation. Comments are not needed.

GPIO_PORTB_DATA_R   EQU 0x400053FC
GPIO_PORTB_DIR_R    EQU 0x40005400
GPIO_PORTB_DEN_R    EQU 0x4000551C
SYSCTL_RCGCGPIO_R   EQU 0x400FE608
PortB_Init
    PUSH {R4,R5}
    LDR R5, =SYSCTL_RCGCGPIO_R
    LDR R4, [R5]

    STR R4, [R5]
    NOP
    NOP
    LDR R5, =GPIO_PORTB_DIR_R
    LDR R4, [R5]

    STR R4, [R5]
    LDR R5, =GPIO_PORTB_DEN_R
    LDR R4, [R5]

    STR R4, [R5]
(20) **Question 5.** You have been hired to build a Morse code distress signal detector. The detector monitors for a special SOS bit pattern "000111000" on Port B, Pin 7 (i.e., PB7). Upon detecting the pattern you must activate an LED using negative logic on PB2. PB7 is input and PB2 is output. Assume from Question 4 that the port has been activated, and that PB7 and PB2 are configured to be input and output, respectively. You may assume port registers

```c
GPIO_PORTB_DATA_R EQU 0x400053FC
#define GPIO_PORTB_DATA_R (*((volatile uint32_t *)0x400053FC))
```

**Part a)** Write a subroutine in C or assembly called SOS_Detector that first reads PB7 nine times, and then if the nine consecutive inputs match the pattern "000111000", the subroutine should return a 1, otherwise it should return a 0. Subroutine SOS_Detector **must be AAPCS compliant.**

**Part b)** Integrate the SOS_Detector from above into a loop that turns on the LED (using the logic specified above) whenever the SOS pattern is detected, **waits two cycles** and turns off the LED and continues to check for the SOS signal. Execute these steps over and over. Be friendly.
(5) Question 6. You are to interface an external LED on Port A pin 0 that operates using positive logic. You have an LED whose desired brightness requires an operating point of $(V_d, I_d) = (3V, 15mA)$. Given the TM4C microcontroller output low $V_{OL}$ ranges between $(0V, 0.5V)$ and output high $V_{OH}$ ranges between $(2.4V, 3.3V)$. The 7406 driver’s $V_{OL}$ is $0.5V$. Show the calculation used to find the resistor value needed and draw the circuit below by connecting the needed elements:

R =

(5) Question 7. You are to interface an external Switch on Port A pin 0 that operates using negative logic by using the needed elements in the following figure. Given the TM4C microcontroller limits the current flow into it to $2 \mu A$ calculate the voltage at Port A pin 0 when the switch is open. Choose a value for $R$ and specify its value.

$V_{PA0} =$
(15) **Question 8.** Given below is the assembly code for an AAPCS compliant subroutine called `func`.

(8) **Part a)** Give an equivalent C function that achieves the same purpose as the given assembly code in `func`.

*Hint: Do not try to translate line-by-line, you have no access to the stack in C*

<table>
<thead>
<tr>
<th>Assembly code</th>
<th>C code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>func CMP R0, R1</code></td>
<td></td>
</tr>
<tr>
<td><code>    BLT L1</code></td>
<td></td>
</tr>
<tr>
<td><code>    PUSH {R1,LR}</code></td>
<td></td>
</tr>
<tr>
<td><code>    POP {R0,LR}</code></td>
<td></td>
</tr>
<tr>
<td><code>L1 BX LR</code></td>
<td></td>
</tr>
</tbody>
</table>

(2) **Part b)** Briefly explain what the `func` subroutine does

(5) **Part c)** Write a C function ‘g’ to perform the following functionality:

**Step1:** Read the value from global variable `input`, divide it by two;

**Step2:** Call the function ‘`func`’ from (a) and pass it two inputs: the result of step 1 and the constant value 50;

**Step3:** Update the value of the global variable `output` by adding the value returned by the call in step 2 to it;

```c
int32_t input;
ing32_t output;
void g(void) {
    // Hint: Can be done in 1 line with no extra variables,
    // however extra variables are allowed if implemented properly
}
```
(15) **Question 9.** This question deals with the stack.

**Part a)** In AAPCS, the fifth parameter is located on top of the stack. Please write the following function in assembly using AAPCS. Your subroutine must balance the stack.

```c
int32_t add_5(int32_t n1, int32_t n2, int32_t n3, int32_t n4, int32_t n5){
    return n1 + n2 + n3 + n4 + n5;
}
```

**Part b)** Assume the stack pointer (SP) is initially equal to 0x20000200, and registers R0, R1, R2, R3 are 7, 11, 15, and 19 respectively. Draw the contents of the stack and the values in registers R0, R1, R2, R3 after these two instructions are executed. Also, label the new SP on the figure.

```
PUSH {R0,R1,R2}
POP  {R3,R1}
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x200001F0</td>
<td>R0 7</td>
</tr>
<tr>
<td>0x200001F4</td>
<td>R1 11</td>
</tr>
<tr>
<td>0x200001F8</td>
<td>R2 15</td>
</tr>
<tr>
<td>0x200001FC</td>
<td>R3 19</td>
</tr>
<tr>
<td>0x20000200</td>
<td></td>
</tr>
<tr>
<td>0x20000204</td>
<td></td>
</tr>
<tr>
<td>0x20000208</td>
<td></td>
</tr>
<tr>
<td>0x2000020C</td>
<td></td>
</tr>
<tr>
<td>0x20000210</td>
<td></td>
</tr>
</tbody>
</table>

Initial SP
Memory access instructions

\begin{verbatim}
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
LDRSB Rd, [Rn]      ; load signed 8-bit at [Rn] to Rd
LDRSB Rd, [Rn,#off] ; load signed 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
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
\end{verbatim}

Branch instructions

\begin{verbatim}
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, return address in LR
BLX  Rm     ; branch to subroutine indirect specified by Rm
\end{verbatim}

Interrupt instructions

\begin{verbatim}
CPSIE I       ; enable interrupts (I=0)
CPSID I       ; disable interrupts (I=1)
\end{verbatim}

Logical instructions

\begin{verbatim}
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)
\end{verbatim}
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

General purpose
registers

R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11
R12 R13(MSP) R14(LR) R15(PC)

Condition code bits
N negative
Z zero
V signed overflow
C carry or
unsigned overflow

256k Flash
ROM
0x0000.0000
0x0003.FFFF
0x0200.0000
0x0200.7FFF
32k RAM
0x4000.0000
0x4000.FFFF
I/O ports
Internal I/O
PPB
0xE000.0000
0xE004.1FFF

Janapa Reddi, Telang, Tiwari, Valvano, Yerraballi  Feb 26, 2015  7:00pm-8:15pm