EE 3381: Microcontrollers and Embedded Systems

SOLUTIONS to Spring 2013 Second Exam (April 18, 2013)

Instructions: Read carefully before beginning.

- The time for this exam is 1 hour and 20 minutes.
- You may only refer to the ARM Instruction Set Quick Reference Card. You may NOT use your textbooks, lecture notes, lab assignments, lab solutions, a computer, or phone.
- During the exam, you may not contact TA Matthew Tonnemacher, even for clarifications: Interpret problems as best as you can and explain your interpretations/assumptions.
- Write your answers on the exam pages provided, using front and back if needed.
- This exam is covered by the SMU Honor Code. Please sign acknowledging the following pledge: On my honor, I have neither given nor received any unauthorized aid on this examination.
- Good luck!

Signature: ______________________________

Name (Printed): ______________________________

Problem	Score	Total Possible
1		12
2		10
3		6
4		11
5		21
6		20
7		20
Total		100

1. (12 pts) Translate each of the following conditions into a single ARM instruction (a condition flag state table is on the last page for your reference for b and c):

   a. Push a block of registers from r0 to r8 and the link register onto the stack using a full descending stack convention. *You may not use FD* in the instruction, but should use the appropriate choice out of IA, IB, DA, or DB, representing increment(I)/decrement(D) after(A)/before(B). You may assume that the SP has been initialized properly.

   ```
   STMDB SP!, {r0-r8,lr}
   ```

   b. Branch and link to a label called *Factorial* that branches if condition code N is clear.

   ```
   BLPL Factorial
   ```

   c. Subtract r4 from r5 and put the result in r2 only if condition code V is clear. This subtraction should *not* update the condition codes according to the result stored in r2.

   ```
   SUBVC r2, r5, r4
   ```

2. (10 pts) Assume that memory and registers r0 through r3 appear as follows (Note: I have changed the order of the addresses in this problem to reflect how we have been picturing memory all semester):

Address	Value	Register	Value
0x7FFC	0xCAFE1234	r0	0x13
0x8000	0x00000001	r1	0xFFFFFFFF
0x8004	0xFEEDDEAF	r2	0xEEEEEEEE
0x8008	0x00008888	r3	0x8008
0x800C	0x12340000	-	-
0x8010	0xBABE0000	-	-

Describe the memory and register contents after executing the instruction. *Remember that this is a block copy and the lowest register number takes the first word read from memory whereas the highest register number takes the last word read from memory.*

```
LDMDA    r3!, {r0, r1, r2}
```

r0 = 0x1, r1 = 0xFEEDDEAF, r2 = 0x00008888, r3 = 0x7FFC

3. (6 pts) Write three lines of code representing a do-while loop that will execute 10 times. (Hint: the only way it can be 3 lines is if you count down.)

```
          MOV   r0, #10
loop      SUBS  r0, r0, #1
          BNE   loop
```

4. (11 pts) Read the following code and describe what it does. Do not simply interpret line for line, but rather come up with the big picture of the operation the code performs.

```
       AREA   Prob4, CODE, READONLY
       ENTRY

start  MOV    r5, #21
       LDR    r4, =Values
oloop  LDR    r0, [r4, r5, LSL #2]
       MOV    r1, #22
       MOV    r2, #0
iloop  AND    r3, r0, #1
       ADD    r2, r3, r2, LSL #1
       MOV    r0, r0, LSR #1
       SUBS   r1, r1, #1
       BNE    iloop
       SUBS   r5, r5, #1
       BNE    oloop

Done   B      Done
Values DCD    1, 2, 4, 5, 6, 7, -8, 11, 23, 5
       DCD    4, 5, 6, 2, 3, 4, 5, 13, 45, 7
       DCD    3, 4
       END
```

What does this code do?

The code will go through the list of Values at the bottom and reverse the bits of each of the 22 words of data. However, the program will not store the result back to these Values. Therefore, in the end, nothing is changed in memory after this code is executed.

5. (21 pts) **Tables.** Complete the following code that creates a jump table containing the addresses of three functions: sin(x), cos(x), and tan(x). The arguments to these functions will be between 0 and 45 degrees. Use a test case of a cosine lookup of 27 degrees (each blank: 1 point).

```
          AREA    JumpTable, CODE, READONLY
num       EQU     __3_____
          ENTRY
start     MOV     r0, _#1____          ; test data: signify cosine to be used
          MOV     r1, _#27___          ; test data: signify 27 degrees to be used
          BL      trigfun              ; call the function
stop      B       stop
trigfun   CMP     r0, #num             ; if r0>=num, then no entry in jump table
          MOVHS   pc, lr
__________ADR     r3, JumpTable__      ; ***OR***
__________LDR     r3, =JumpTable_      ; load address of jump table
__________LDR     pc, [r3,r0,LSL #2]_ ; jump to appropriate routine
JumpTable
__________DCD     DoSin__________      ; build the jump table
__________DCD     DoCos__________
__________DCD     DoTan__________
DoSin     MOV     r7, r1               ; make a copy
          LDR     r2, =270             ; constant won’t fit in rotation scheme
          ADR     r4, _sin_data__      ; load address of sin table
          CMP     r1, #90              ; determine quadrant
          BLE     retvalue             ; first quadrant?
          CMP     r1, #180
          RSBLE   r1, r1, #180         ; second quadrant?
          BLE     retvalue
          CMP     r1, r2
          SUBLE   r1, r1, #180         ; third quadrant?
          BLE     retvalue
          RSB     r1, r1, #360         ; otherwise, fourth
retvalue  LDR     r0, [r4,r1,LSL #2]_ ; get sin value from table
          CMP     r7, #180             ; do we return a neg value?
          RSBGT   r0, r0, #0           ; negate the value
          MOV     pc, lr               ; return

sin_data  DCD     0x00000000, 0x023be164, ... , 0x7fffffff

DoCos     MOV     r7, r1               ; make a copy
          LDR     r2, =270             ; constant won’t fit in rotation scheme
          ADR     r4, _cosine_data_    ; load address of cosin table
          CMP     r1, #90              ; determine quadrant
          BLE     retvalue1            ; first quadrant?
          CMP     r1, #180
          RSBLE   r1, r1, #180        ; second quadrant?
          BLE     retvalue1
          CMP     r1, r2
          SUBLE   r1, r1, #180        ; third quadrant?
          BLE     retvalue1
          RSB     r1, r1, #360        ; otherwise, fourth
retvalue1 LDR     r0, [r4,r1,LSL #2]_ ; get sin value from table
          CMP     r7, #90
          BLT     done1
          CMP     r7, r2              ; do we return a neg value?
          BGT     done1
          RSBGT   r0, r0, #0          ; negate the value
done1     MOV     pc, lr

cos_data  DCD     0x7fffffff, 0x7ffb0280, ... , 0x00000000

DoTan     ADR     r4, _tan_data__     ; load address of tan table
__________LDR     r0, [r4,r1,LSL #2]_ ; get tan value from table
          MOV     pc, lr

tan_data  DCD     0x00000000, 0x023bf7b4, ... , 0x7fffffff

          END
```

Fill in the blanks above to complete the code.

6. (20 pts) **Subroutines.** Complete the following code that takes the factorial program discussed earlier in the semester (Chapter 3) into a subroutine, using empty ascending stacks. As before, *you may not use EA* to push values to the stack and pull values from the stack. You should use the appropriate choice from the following options: IA, IB, DA, DB. Notice that there are two different subroutines that do the same thing, but pass by register in the first and pass by reference in the second (each blank: 1 point).

```
          AREA    FactorialPassRegAndRef, CODE, READONLY
SRAM_BASE EQU     0x40000000
          ENTRY
          LDR     sp, =SRAM_BASE
          LDR     r3, =SRAM_BASE+100
__________MOV     r0, #10________     ; pass test value by register using r0
__________BL      FactReg________     ; call appropriate function below
                                      ; result now in r1
__________STR     r0, [r3]_______     ; save parameter in memory
                                      ;     (at SRAM_BASE+100)
__________BL      FactRef________     ; call appropriate function,
                                      ;     passing r3 as reference
                                      ; result now in [r3]
stop      B       stop
FactReg   STMIA   sp!, {r4,lr}        ; save appropriate registers
                                      ;     on stack for reentrancy
__________MOV     r4, r0_________     ; copy n from register passed in to r4
loop      SUBS    r4, r4, #1          ; decrement next multiplier
          MULNE   r1, r0, r4          ; perform multiply, result in r1
          MOV     r0, r1
          BNE     loop                ; go again if not complete
__________LDMDB   sp!, {r4,pc}        ; restore appropriate registers
                                      ;     from stack for reentrancy
FactRef   STMIA   sp!, {r1,r2,r4,lr}  ; save appropriate registers
                                      ;     on stack for reentrancy
__________LDR     r2, [r3]_______     ; get parameter by reference into r2
          MOV     r4, r2              ; copy n into a temp register
loop1     SUBS    r4, r4, #1          ; decrement next multiplier
          MULNE   r1, r2, r4          ; perform multiply
          MOV     r2, r1
          BNE     loop1               ; go again if not complete
__________STR     r1, [r3]_______     ; store result (r1) at [r3]
__________LDMDB   sp!, {r1,r2,r4,pc}  ; restore appropriate registers
                                      ;     from stack for reentrancy
          END
```

Fill in the blanks above to complete the code.

7. (20 pts) **Exceptions.** Build an Undefined Exception Handler that tests for and handles a new instruction called LargeSub which will perform a 128-bit subtraction, placing the result in r0, r1, r2, and r3. The first operand should be placed in r4, r5, r6, and r7, and the second operand should be in registers r8, r9, r10, and r11. The most significant words are located in the highest register values (r3, r7, and r11) and the least significant words are located in the lowest register values (r0, r4, and r8). Write the code as if your are testing the routine with two 128-bit sample values (2 points). Format for the code:

   - Any EQU statements to initialize RAM, mode, or I/F bits (1 point). Relevant bits from the program status register (PSR) are 7 (IRQ), 6 (FIQ), and 4:0 (mode). The mode numbers are on your reference card.
   - Start the system from a reset event (i.e., go through Reset Handler to initialize system - 1 point).
   - Set up the stack pointer in Undefined mode (1 point).
   - In handler, store off relevant registers and SPSR to stack (1 point).
   - Get the opcode of the instruction and check to see if it equals that for your instruction (1 point).
   - If so, call the subroutine to perform the 128-bit subtraction (1 point).
   - Implement the LargeSub functionality (1 point).
   - When subroutine completes, it returns to Reset Handler's main body, restoring all state (1 point).
   - Restore SPSR and registers (1 point).

```
RAMSTART     EQU   0x40000000     ; Start of RAM
Mode_UND     EQU   0x1B           ; Bits for Undefined Mode
Mode_SVC     EQU   0x13           ; Bits for Supervisor Mode
I_Bit        EQU   0x80           ; When I bit is set, IRQ is disabled
F_Bit        EQU   0x40           ; When F bit is set, FIQ is disabled
             AREA  LargeSubProb, CODE
     ENTRY

Vectors      LDR   PC, Reset_Addr
             LDR   PC, Undef_Addr
Reset_Addr   DCD   ResetHandler
Undef_Addr   DCD   UndefHandler
ResetHandler MSR   CPSR_c, #Mode_UND:OR:I_Bit:OR:F_Bit
                                        ; enter Undefined Mode
             LDR   sp, =RAMSTART        ; Setup Undefined stack pointer
             MSR   CPSR_c, #Mode_SVC:OR:I_Bit:OR:F_Bit
             MOV   r4, #0x4             ; Put test data in r4
             MOV   r5, #0x5             ; Put test data in r5
             MOV   r6, #0x6             ; Put test data in r6
             MOV   r7, #0x7             ; Put test data in r7
             MOV   r8, #0x8             ; Put test data in r8
             MOV   r9, #0x9             ; Put test data in r9
             MOV   r10, #0xA            ; Put test data in r10
             MOV   r11, #0xB            ; Put test data in r11
LargeSubDcd  DCD   0x77F00FF0           ; {r0-r3} = {r4-r7} - {r8-r11}
Stop         B     Stop
UndefHandler STMFD sp!, {r12,lr}        ; Store off stack registers
             MRS   r12, SPSR            ; Get SPSR
             STMFD sp!, {r12}           ; Store it on the stack
             LDR   r12, [lr, #-4]       ; Get Undef instruction
             BIC   r12, r12, #0xF00FFFFF; Extract potential 0xACE
             TEQ   r12, #0x07F00000
             BLEQ  LargeSub
             LDR   r12, [SP], #4        ; Restore SPSR from stack
             MSR   SPSR_csxf, r12
             LDMFD sp!, {r12,pc}^       ; Restore registers and return
LargeSub     SUBS  r0, r4, r8
             SBCS  r1, r5, r9
             SBCS  r2, r6, r10
             SBC   r3, r7, r11
             MOV   pc, lr               ; Return from subroutine
             END
```