EE 3381: Microcontrollers and Embedded Systems

Spring 2016 Second Exam (April 14, 2016)

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 Eric Johnson, 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. There is an additional page at the end of the exam for reference that may be used for scratch work, if necessary.
- This exam is covered by the SMU Honor Code. Please sign 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		15
2		10
3		5
4		5
5		15
6		5
7		20
8		15
9		10
Total		100

1. (15 pts) **Conditional Execution.** The following code is inefficient:

```
alg     CMP   r0, r1
        BEQ   end
        BLT   less
        SUB   r0, r0, r1
        B     alg
less    SUB   r1, r1, r0
        B     alg
end     B     end
```

   a. (5 points) What does this algorithm do (at a high level, not necessarily line by line)?

   b. (5 points) Why is it inefficient?

   c. (5 points) Rewrite the code using conditional execution to make it efficient.

2. (10 pts) **Block Store.** Assume that memory and registers r0 through r3 appear as follows:

Address	Value	Register	Value
0x9000	0x______	r5	0x11223344
0x9004	0x______	r6	0x55667788
0x9008	0x______	r7	0x99AABBCC
0x900C	0x______	SP	0x900C
0x9010	0x______		
0x9014	0x______		
0x9018	0x______		

Change any relevant memory and register contents after executing the following instruction:

```
STMDA    SP!, {r5, r7, r6}
```

3. (5 pts) What stack convention is used in the previous question?

4. (5 pts) **Loops and Branches.** Implement a for-do loop structure in ARM assembly where the initial value of the loop control variable is 1 and counts up to 10, incrementing by 1.

5. (15 pts) **Jump Tables.** Consider a video game that has 100 possible vertical locations from top to bottom where 1 is the top of the screen and 100 is the bottom and **held in r1**. Similarly, there are 100 possible horizontal locations from left to right where 1 is the left-most location, and 100 is the right-most location and **held in r2**.

When the code starts, the initial position of the vertical and horizontal are in the middle of the screen (50 vertical and 50 horizontal). The register r0 holds which direction is pressed 0=left, 1=up, 2=right, 3=down and the position should be updated by 1 unit on the screen in that direction. Fill in the blanks below to complete the functionality using a jump table. *You do not have to consider boundary conditions of the screen (less than 1 or greater than 100).*

```
       AREA   Zelda, CODE, READONLY
num    _____________________  ; Number of entries in jump table
       ENTRY
          ; assume r0 holds input from directional pad
start  MOV    r1, #50         ; represents middle vertical location (1,100)
       MOV    r2, #50         ; represents middle horizontal location (1,100)
       MOV    r3, #1          ; r3 represents increment of movement
loop   ; there would be a number of calls to subgo as buttons are pressed
       ; r0 would get a new value each time a button is pressed
       ; left = 0, up = 1, right = 2, down = 3
       BL     subgo           ; this is one of the calls to subgo
       B      loop
subgo  _____________________  ; make sure functionality supported
       _____________________  ; If r0 value not supported, return from subgo
       _____________________  ; Load address of jump table
       _____________________  ; Jump to appropriate routine
jmptab DCD    Do____________  ; Fill in appropriate direction
       DCD    Do____________  ; Fill in appropriate direction
       DCD    Do____________  ; Fill in appropriate direction
       DCD    Do____________  ; Fill in appropriate direction
Do__________________________  ; Operation when r0 = 0
       MOV    pc, lr          ; Return
Do__________________________  ; Operation when r0 = 1
       MOV    pc, lr          ; Return
Do__________________________  ; Operation when r0 = 2
       MOV    pc, lr          ; Return
Do__________________________  ; Operation when r0 = 3
       MOV    pc, lr          ; Return
       END
```

6. (5 pts) Through what mechanism are values passed in and out of subroutine *subgo* above?

7. (20 pts) **Pass By Stack.** Start by pushing 3 arbitrary 32-bit values on the stack. You will then reorder these 3 values and change them on the stack by placing the least value in the least memory address, middle value in between, and greatest value in the highest memory address. In summary, you will pass 3 arbitrary values into a subroutine *Reorder*, sort them, and pass those 3 sorted values back on the stack.

   Note that you can use any stack convention, but you must compute the offsets to and from the parameters begin passed, and while you need to use a consistent stack policy, you may not use ED,EA,FD,FA for empty descending, empty ascending, full descending, or full ascending stack policies. Rather, for full credit, you need to use the following four options instead: IA, IB, DA, DB for incremement after, increment before, decrement after, or decrement before.

   The following is the original code. Fill in the blanks to pass the parameters by stack.

```
           AREA    PassByStack, CODE, READONLY
STACKBASE  EQU     0x40000000         ; EQU for initial location of stack
           ENTRY
start      LDR     SP, =STACKBASE     ; Set up stack pointer
____________________________________  ; Set value in temp register
____________________________________  ; Set value in temp register
____________________________________  ; Set value in temp register
____________________________________  ; Put all 3 on stack with block copy
____________________________________  ; Call Reorder subroutine
 ; some code here would use ordered list
 ; ...
 ;
Done       B       Done               ; End program
Reorder    _________________________  ; Save off scratch/link registers




                                      ; blank space for reordering on stack




           _________________________  ; load back scratch/LR to return
           END
```

8. (15 pts) **Exceptions.** Build an Undefined Exception Handler that tests for and handles a new instruction called LargeAdd which will perform a 128-bit addition. The first operand is already located in r0, r1, r2, and r3, and the second operand is already located in registers r4, r5, r6, and r7. Finally, the result should be placed in r8, r9, r10, and r11. The most significant words are located in the lowest register values (r0, r4, and r8) and the least significant words are located in the highest register values (r3, r7, and r11). Write the code as if your are testing the routine with two 128-bit sample values. You may assume this test data is already in r0-r7.

```
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  LargeAddProb, 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 Undef Mode
             ________________________________________ ; Setup Undef stack pointer
             MSR   CPSR_c, #Mode_SVC:OR:I_Bit:OR:F_Bit
         ; Assume test data is already in r0-r7
LargeAdd     DCD   0x77F00FF0                ; {r8:r11} = {r0:r3} + {r4:r7}
Stop         B     Stop
UndefHandler ________________________________ ; Push relevant registers to stack
             ________________________________ ; Get Undefined instruction
             ________________________________ ; Extract the opcode in 0x0xx00000
             ________________________________ ; Test if those nibbles equal 0x7F
             ________________________________ ; If so, call LargeAdd subroutine
             ________________________________ ; Restore registers and return
LargeAdd                                      ; Perform 128-bit addition
                                              ; {r8:r11} = {r0:r3} + {r4:r7}
                                              ; ...
                                              ; ...
                                              ; ... extra space to leave room
                                              ; return from subroutine
             END
```

9. (10 pts) **Interfacing.** Fill in the following code that will configure a digital to analog (D/A) converter. Essentially, a digital value goes into a register and results in a different analog voltage level coming out of a pin based on that value. The highest digital value of 1024 corresponds to the highest possible voltage level (V_{REF}). Conversely, the lowest digital value of 0 corresponds to the lowest possible voltage level of 0. Use the D/A to output the following set of room temperatures for different parts of the day on the AOUT pin: 80, 78, 76, 78, 80.

```
           AREA     DtoA, CODE, READONLY
PINSEL1    EQU      0xE002C004
DACREG     EQU      0xE006C000
           ENTRY

main       ____________________________  ; point to pin select 1 address (0xE002C004)
_______________________________________  ; read, modify, write the value so that:
_______________________________________  ; change bits 19:18 to be the value of 01
_______________________________________  ; keeping other bits must remain unchanged
_______________________________________  ;
_______________________________________  ; point to the temperature values
_______________________________________  ; loop: look up the temperature values
_______________________________________  ; bits 15:6 of DACREG for digital value
_______________________________________  ; left shift the looked up temp to 15:6
_______________________________________  ; point to the DAC register (0xE006C000)
_______________________________________  ; write that temperature to the DAC
     ; assume sufficient waiting until the time of the next temperature is relevant
_______________________________________  ; loop until done
_______________________________________  ; exact number of lines may differ

done       B        done
temps      ____     ___, ___, ___, ___, ___
```

Bit	Symbol	Value	Description	Reset Value
5:0	–		Reserved, user software should not write ones to reserved bits. Value for reading is undefined.	NA
15:6	VALUE		After selected settling time after this field is written with a new VALUE, the voltage on A_{OUT} pin is VALUE/1024$*V_{REF}$	0
16	BIAS	0	The settling time of the DAC is 1 μs max, and the max current is 700 μA.	0
		1	The settling time of the DAC is 2.5 μs max, and the max current is 350 μA.	
31:17	–		Reserved, user software should not write ones to reserved bits. Value for reading is undefined.	NA

Table 1: DAC Register Bit Description