EE 3381: Microprocessors and Embedded Systems

Spring 2015 Second Exam (April 16, 2015)

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 Professor Camp, 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		10
2		6
3		10
4		14
5		20
6		20
7		20
Total		100

1. (10 pts) **Block Store.** Assume that memory and registers r0 through r3 appear as follows:

Address	Value	Register	Value
0x9000	0x_____	r0	0x13
0x9004	0x_____	r1	0xFFFFFFFF
0x9008	0x_____	r2	0xEEEEEEEE
0x900C	0x_____	r3	0x9010
0x9010	0x_____	SP	0x9000

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

```
STMIB    SP!, {r3, r2, r1, r0}
```

2. (6 pts) What stack convention is used in the previous question?

3. (10 pts) **Saving/Restoring State.** Describe how the interrupting mode (e.g., IRQ or FIQ) saves and restores the current program state before the interrupt occurs. For full credit, use the following terms (CPSR, SPSR, stack, push, pop, LR, and PC).

4. (14 pts) **Interpret the Code.** Read the following code and change the data regions (*data1* and *data2*) according to how the code will modify those regions' data members after the code executes.

```
        AREA    Prob4, CODE, READONLY
num     EQU     22
        ENTRY

start   LDR     r0, =data1
        LDR     r1, =data2
        MOV     r2, #num
sub1    LDR     r3, [r0], #4
        MOV     r3, r3, LSL #1
        STR     r3, [r1], #4
        SUBS    r2, r2, #1
        BNE     sub1

        LDR     r0, =data1
        MOV     r2, #num
        SUB     r1, r1, #4

sub2    LDR     r3, [r1], #-4
        MOV     r3, r3, LSL #1
        STR     r3, [r0], #4
        SUBS    r2, r2, #1
        BNE     sub2

Done    B       Done

        AREA    Block, DATA, READWRITE
data1   DCD     2, 3, 5, 7, 1, 3, 7, 9, 3, 9
        DCD     1, 7, 1, 3, 7, 3, 9, 1, 7, 1
        DCD     3, 9
data2   DCD     0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        DCD     0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        DCD     0, 0

        END
```

5. (20 pts) **Pass By Stack.** Write the division routine below as a subroutine that uses empty descending stacks. Pass the subroutine arguments to and from the subroutine using the stack. Use a dividend of 35 and divisor of 5. For full credit, you must compute the offsets to and from the parameters begin passed and you may not use ED for empty descending but need to use the following four options instead: IA, IB, DA, DB.

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

```
            AREA    Division, CODE, READONLY
STACKBASE   EQU     0x40000000          ; EQU for initial location of stack
RESULTHERE  EQU     0x99999900          ; EQU for result location after subroutine
Rcnt        RN      0                   ; assign r0 to Rcnt
Ra          RN      1                   ; assign r1 to Ra (dividend)
Rb          RN      2                   ; assign r2 to Rb (divisor)
Rc          RN      3                   ; assign r3 to Rc (result)
Rd          RN      4                   ; assign r4 to Rd (remainder)
            ENTRY
            LDR     SP, =STACKBASE      ; Set up stack pointer
___________________________________     ; Set value of 35 for dividend (Ra)
___________________________________     ; Set value of 5 for divisor (Rb)
___________________________________     ; Put both on stack
___________________________________     ; Call Division subroutine
___________________________________     ; Load result and remainder from stack
___________________________________     ; Point to RESULTHERE
___________________________________     ; Store result, then remainder in RESULTHERE
Done        B       Done                ; End program
Division    ________________________    ; Save off scratch/link registers
___________________________________     ; Load dividend from stack
___________________________________     ; Load divisor from stack
            MOV     Rcnt, #1            ; Bit to control the division
Div1        CMP     Rb, #0x80000000     ; move Rb until greater than Ra
            CMPCC   Rb, Ra
            MOVCC   Rb, Rb, LSL #1
            MOVCC   Rcnt, Rcnt, LSL #1
            BCC     Div1
            MOV     Rc, #0
Div2        CMP     Ra, Rb              ; Test for possible subtraction
            SUBCS   Ra, Ra, Rb          ; Subtract if OK
            ADDCS   Rc, Rc, Rcnt        ; Put relevant bit into result
            MOVS    Rcnt, Rcnt, LSR #1  ; Shift control bit
            MOVNE   Rb, Rb, LSR #1      ; Halve unless finished
            BNE     Div2                ; Result into Rc, remainder in Ra
            MOV     Rd, Ra              ; Now remainder is in Rd
___________________________________     ; Store result (Rc) at bottom of stack-4
___________________________________     ; Store remainder (Rd) at bottom of stack
___________________________________     ; Restore scratch registers and PC
            END
```

6. (20 pts) **Exception Handling.** Consider the following reset and undefined handler that was used in your book to define an instruction called ADDSHIFT. The ADDSHIFT instruction added r0 to Rm (a register between r0 and r7 described in the instruction) and left shifts the result by 5. The current behavior can be described as r0 = (r0 + Rm) << 5.

```
; Area Definition and Entry Point
                AREA        AddShiftDefined, CODE
SRAM_BASE       EQU         0x4000              ; start of RAM
Mode_UND        EQU         0x1B
Mode_SVC        EQU         0x13
I_Bit           EQU         0x80
F_Bit           EQU         0x40
                ENTRY
; Exception Vectors
Vectors         LDR         PC, Reset_Addr
                LDR         PC, Undef_Addr
                LDR         PC, SWI_Addr
                LDR         PC, PAbt_Addr
                LDR         PC, DAbt_Addr
                NOP                             ; Reserved Vector
                LDR         PC, IRQ_Addr
FIQHandler      ; do something important
                SUBS        PC, LR, #4          ; return to main program
Reset_Addr      DCD         ResetHandler
Undef_Addr      DCD         UndefHandler
SWI_Addr        DCD         SWIHandler
PAbt_Addr       DCD         PAbtHandler
DAbt_Addr       DCD         DAbtHandler
                DCD         0
IRQ_Addr        DCD         IRQHandler
SWIHandler      B           SWIHandler
PAbtHandler     B           PAbtHandler
DAbtHandler     B           DAbtHandler
IRQHandler      B           IRQHandler
ResetHandler
; Undefined Instruction test
; 31 30 29 28|27 26 25 24|23 22 21 20|19 18 17 16|15 14 13 12|11 10 9 8|7 6 5 4|3 2 1 0
; 0  1  1  1 |0  1  1  1 |1  1  1  1 |0  0  0  0 |0  0  0  0 |1  1  1 1|1 1 1 1|Rm
; CC = AL    |          OP           | Rn = 0    | Rd = 0    |Not Used         |Rm
; At the beginning of time (after reboot), set up a stack pointer in UNDEF mode
; since we know our simulation will hit an undefined instruction
                MSR         CPSR_c, #Mode_UND:OR:I_Bit:OR:F_Bit
                LDR         SP, =SRAM_BASE+80   ; Initialize stack pointer
                MSR         CPSR_c, #Mode_SVC:OR:I_Bit:OR:F_Bit
                MOV         r0, #124            ; Put test data in r0
                MOV         r4, #0x8B           ; Put test data in r4
ADDSHFTr0r0r4   DCD         0x77F00FF4          ; r0 = (r0 + r4) LSL #5
                MOV         r0, #124            ; Put test data in r0
                MOV         r5, #0x9F           ; Put test data in r5
ADDSHFTr0r0r5   DCD         0x77F00FF5          ; r0 = (r0 + r5) LSL #5
Stop            B           Stop
```

```
UndefHandler
                STMFD       SP!, {r0-r12,LR}     ; Save workspace & LR to stack
                MRS         r0, SPSR             ; Copy SPSR to r0
                STR         r0, [SP, #-4]!       ; Save SPSP to stack
                LDR         r0, [LR, #-4]        ; r0 = 32-bit undefined instruction
                BIC         r2, r0, #0xF00FFFFF  ; clear out all but opcode bits
                TEQ         r2, #0x07F00000      ; r2 = opcode for LargeSub
                BLEQ        ADDSHIFTInstruction  ; if a valid opcode, handle it
                   ; otherwise, look for other undefined instructions here...
                LDR         r1, [sp], #4         ; Restore SPSR to r1
                MSR         SPSR_cxsf, r1        ; Restore SPSR
                LDMFD       SP!, {r0-r12,PC}^    ; Return after undefined instructions
ADDSHIFTInstruction
                BIC         r3, r0, #0xFFFFFFF0  ; mask out all bits except Rm
                ADD         r3, r3, #1           ; bump past the SPSR on the stack
                LDR         r0, [sp, #4]         ; grab r0 from the stack
                LDR         r3, [sp, r3, LSL #2] ; use the Rm field as an offset
                ADD         r0, r0, r3           ; calculate r0+Rm
                MOV         r0, r0, LSL #5       ; r0 = (r0+Rm)<<5
                STR         r0, [sp, #4]         ; store r0 back on the stack
                MOV         pc, lr
                END
```

Make inline changes to the code to reflect the following new definition for SUBSHIFT instruction. Bits 19-16 of the instruction will represent the first operand Rn (instead of always being r0). Bits 15-12 of the instruction will represent the destination Rd (instead of always being r0). Finally, r12 will represent the amount of shifts (instead of always being 5) and the shifts will be *towards the right* instead of towards the left. Thus, the SUBSHIFT instruction would have the following more general behavior Rd = (Rn-Rm) >> r12.

7. (20 pts) **Interfacing.** Fill in the following code that will configure the D/A converter that we discussed in class. Also, use the D/A to output the following set of oven temperatures on the AOUT pin: 325, 450, 500, 350, 375.

```
            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:
____________________________________      ; bits 19:18 are 1
____________________________________      ; other bits must remain unchanged
____________________________________      ;
____________________________________      ; point to the temperature values
____________________________________      ; loop: look up the temperature values
____________________________________      ; recall that the DAC register uses bits 15:6
____________________________________      ; left shift the looked up temp to 15:6
____________________________________      ; point to the DAC register (0xE006C000)
____________________________________      ; write each temperature to the DAC
____________________________________      ; loop until done
____________________________________      ; exact number of lines may differ

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