EECS 373 - Homework #1 Solutions

Name: ______________________________ **unique name:** ________________

Due 19 January via Gradescope. Please use the answer boxes provided. Submit a PDF of your completed assignment to Gradescope. Typed answers and neat handwritten answers are both acceptable.

Question 1

Short answer questions: **[10 points, 2 each]**

A) What type of memory is executable code typically stored in, when it must survive power loss?

Non- volatile flash memory

B) What is the memory range for the peripheral devices, based on Slide 22 of Lecture 2?

0x40000000 to 0x5fffffff

C) Is the ARM ISA and hardware capable of supporting Big Endian addressing?

Yes

D) Using at most one sentence, indicate the main difference between the ARM sub and subs instructions.

SUBS instruction updates the APSR while SUB does not.

E) Is an ABI part of an ISA?

No, but the ABI builds upon and depends on the ISA.

Question 2

Part A:
Using the ARMv7-M Architecture Reference Manual describe in a straightforward manner what the ROR (immediate) instruction does. **[3 points]**

Rotate Right (immediate) provides the value of the contents of a register rotated by a constant value. The bits that are rotated off the right end are inserted into the vacated bit positions on the left. It can optionally update the condition flags based on the result.

Part B:
Write the hexadecimal for the machine code you would expect to get for the following instructions. **[9 points, 3 each]**

1) LSL R1, R4, #4

0x0121

2) LSR R1, R2, #24

0xE11

3) ASR R12, R3, #1

EA4F0C63

Question 3

For each of the following program segments, assume you start with all memory locations equal to zero. Indicate the values found in *these* memory locations when the programs end. Write all answers in hex. **[16 points, 8 for each part]**

Part A)

```
BASE_EMC = 0x74000000;
uint32_t *a = (uint32_t*)BASE_EMC;
*a = 0x01234567;
*(a-1) = 0xfedcba98;
*(uint32_t*)((uint32_t)a+2)=0x01234567;
```

Either solution is acceptable

Address	Value
0x73FFFFFD	**0xBA**
0x73FFFFFE	**0xDC**
0x73FFFFFF	**0xFE**
0x74000000	**0x67**
0x74000001	**0x45**
0x74000002	**0x23**
0x74000003	**0x01**
0x74000004	**0x00**

Address	Value
0x73FFFFFD	**0xBA**
0x73FFFFFE	**0xDC**
0x73FFFFFF	**0xFE**
0x74000000	**0x67**
0x74000001	**0x45**
0x74000002	**0x67**
0x74000003	**0x45**
0x74000004	**0x23**

Part B)

```
mov r2, #100
movw r1, #85
movt r1, #85
strh r1, [r2, #3]
str r1, [r2], #2
strb r1,[r2, #2]!
strb r2,[r2, #-3]
```

Address	Value
100	0x55
101	0x68
102	0x55
103	0x00
104	0x55
105	0x00
106	0x00

Hint: Page A6-15 of the ARMv7-M Architecture Reference Manual may be useful here.

Question 4

Write an ABI compliant assembly function that checks if an unsigned integer has a square root that is an unsigned integer and return the square root. For example if the function is given 25, it should return 5 (since 25=5^2). If no such unsigned integer exists, the function should return -1. **[20 points]**

```
.global _start
_start:

main:
mov R0, #65
bl intRoot
done:    B    done


intRoot:
mov    R1, #1
cmp    R1, R0   // Special case 1*1=1, there are other ways to do
                // this but this is easiest modification to existing
beq    isSquare
mov R3, R0

doOver:
    add    R1, R1, #1
    mul    R2, R1, R1
    cmp    R2, R3
    beq    isSquare
    bgt    itsnot
    blt    doOver

isSquare:
     mov    R0, R1
    BX lr

itsnot:
    mov    R0, #0
    sub    R0, R0, #1
    BX lr

_end:
```

NOTE: Probably no penalty for not including a sample main. Question only asks for the abi function.

Question 5

Given the C code below, write an equivalent program in assembly. You can assume that “print” is an ABI compliant function which takes an integer argument. Have the function return to the program that called it. **[17 points]**

```
void main(void)
{
      int i,a=401;
      for(i=0;i<8;i++)
      {
            a=a-i;
            print(a);
      }
}
```

```
main:
        push {r4, r5, lr}
        mov r4, #0
        mov r5, #401

loop:
        cmp r4, #8
        bge exit
        sub r5, r4
        mov r0, r5
        bl print
        add r4, #1
        b loop

exit:
pop {r4, r5, pc}
```

Note: There are a lot of ways to do #3. One could use registers above r3, rather than the stack, to keep “a” and “I” safe, though then you’d have to save (say) r4 and r5 to the stack and restore them (though only once). Net effect: there are a lot of ways to do this.

Question 6

The following tables represent two sections of memory in a Cortex-M processor. For convenience the memory values in the address 0x08000XXX block have been decoded into instructions. Assume that the Program Counter (PC) is 0x08000104 and the Stack Pointer is 0x20000000. Determine the value of R0 and the values of the memory addresses in the right table (i.e. 0x20000000 to 0x1FFFFFD0) when the PC = 0x0800010C. Leave any unknown memory values blank. **[25 points]**

R0 = 120

Address	Instruction	
0x08000104		MOV R0, #5
0x08000108		BL *func*
0x0800010C	*done*	B *done*
0x08000110	*func*	PUSH {R4, LR}
0x08000114		MOV R4, R0
0x08000118		CMP R4, #1
0x0800011C		BNE *else*
0x08000120		MOV R0, #1
0x08000124	*loop*	POP {R4, PC}
0x08000128	*else*	SUB R0, R4, #1
0x0800012C		BL *func*
0x08000130		MUL R0, R4, R0
0x08000134		B *loop*

Address	Value
0x20000000	
0x1FFFFFFC	0x0800010C
0x1FFFFFF8	0x????????
0x1FFFFFF4	0x08000130
0x1FFFFFF0	0x00000005
0x1FFFFFEC	0x08000130
0x1FFFFFE8	0x00000004
0x1FFFFFE4	0x08000130
0x1FFFFFE0	0x00000003
0x1FFFFFDC	0x08000130
0x1FFFFFD8	0x00000002
0x1FFFFFD4	
0x1FFFFFD0	