CSE2312-001 (Fall 2020)
Homework #4

Notes:

- All numbers are in base-10 unless otherwise noted.
- If part of a problem is not solvable, explain why in the answer area.
- Print out the form and handwrite your answers in the spaces below.
- Place the hw4.s file and the scanned answers to problems 1-6 in a single zip file with name lastname_hw4.zip, where lastname is your last name as listed in MyMav.
- Submit the single zip file to Canvas before 11:59:00pm on November 19, 2020.
- Make sure that the code follows the procedure call standards for ARM architecture (see IHI0042F section 5.1), with emphasis on this requirement: "A subroutine must preserve the contents of the registers r4-r8, r10, r11 and SP (and r9 in PCS variants that designate r9 as v6)." (in other words, push and pop R4-11 if you need to use them, as shown in the vector.s examples in class)

1. Encode the following numbers as single-precision floating point numbers:

a. -8192

s = ___, e = ____________, m = ____

32b hex value = ____

b. 1.0625

s = ___, e = ____________, m = ____

32b hex value = ____

c. 0.4

s = ___, e = ____________, m = ____

32b hex value = ____

d. 256 + 1/512

s = ___, e = ____________, m = ____

32b hex value = ____

e. 2.000976562

s = ___, e = ____________, m = ____

32b hex value = ____

2. Assume float x = 4194304.

a. Calculate the smallest positive number that can be added to x that will not be lost in the mantissa.

b. In general, what is the ratio of the large to the smallest single-precision floating point number that can be added together without a loss of accuracy?

3. For the following code, calculate the number of instruction cycles required to execute the following code, using the simplified pipeline timing rules in class, including the time to call this function with BL bro32 and the time to return from the function with BX LR. You can assume that the pipeline is full before the BL bro32 instruction is executed. Before you ask, note that the value of R0 does not matter.

```
// uint32_t bro32(uint32_t x)
// value in R0, return bro(value) in R0

bro32:

    MOV R1, R0 // move original value to R1

    MOV R0, #0 // zero result

    MOV R2, #0x80000000 // test mask with bit 31 set

    MOV R3, #0x00000001 // apply mask with bit 0 set

bro_loop:

    TST R1, R2 // return 0 if bit not set, non-zero if bit set

    ORRNE R0, R0, R3 // if bit in R1 set, set bit in R0

    MOVS R2, R2, LSR #1 // move mask bit to the right

    MOV R3, R3, LSL #1  // move mask bit to the left

    BNE bro_loop // loop back for remaining 31 bits

    BX LR
```

Clock cycles: ______________________

If the clock rate is 4 GHz, what is the execution time in nanoseconds? _____________

4. Assume SP = 0x2000102C before the following instructions are executed:

Address Instruction

```
10000000:       BL fn
            fn:
10001000:       MOV R0, #3072
10001008:       MOV R2, #0x7890
10001010:       PUSH {R0, R2, LR}
            loop:
10001014:       B loop
```

After this program enters the endless loop:

What is the value of the SP? ________________

Assuming the processor uses big-endian convention, what is the value of the following memory locations (place X in the blank if there is not enough information):

Address	8-bit Data
0x2000103A	____________
0x20001039	____________
0x20001038	____________
0x20001037	____________
0x20001036	____________
0x20001035	____________
0x20001034	____________
0x20001033	____________
0x20001032	____________
0x20001031	____________
0x20001030	____________
0x2000102F	____________
0x2000102E	____________
0x2000102D	____________
0x2000102C	____________
0x2000102B	____________
0x2000102A	____________
0x20001029	____________
0x20001028	____________
0x20001027	____________
0x20001026	____________
0x20001025	____________
0x20001024	____________

5. Explain the concept of memory virtualization, including the concept of paging and fragmentation. Also explain the role of virtualization in memory protection between running processes (“programs”).

6. Explain the concept of cache, including the principle of locality. Explain how this can speed up memory accesses.

7. Write assembly functions that implement the following C functions:

a. void prodF64(double z[], const double x[], const double y[], uint32_t count)
   // compute the product of each element in the arrays x and y containing
   // count entries; z[i] = x[i] * y[i], for i = 0, 1, … count-1
   // If x = {10, 20, 30}, y = {5, 10, 5}, and count = 3, then z = {50, 200, 150}

b. float int32ToFloat(int32_t x)
   // converts the signed integer to a single-precision floating point number

c. float dotpF32(const float x[], const float y[], uint32_t count)
   // returns the dot product of two arrays (x and y) containing count entries

d. double minF64(const double x[], uint32_t count)
   // returns the minimum value in the array (x) containing count entries
   // if x = {-1.1, 20, -3} and count = 3, then the function returns -3.

   Write the solution of all of these functions in a single file hw4.s with the functions being callable from a C program. You do not need to send the .c file used to test these functions.