CSE2312-002, 004 (Fall 2021) Homework #5

Notes:
With this homework, we start writing assembly functions for the RPi 3b/3b+.

All numbers are in base-10 unless otherwise noted.

If part of a problem is not solvable, explain why in the answer area.

The target date to complete this homework set is November 9, 2021.

This homework set will not be graded, but please solve all of the problems to prepare for the quizzes and exams.

1. Suppose that BUSINESS2 structure is defined as:

```
typedef struct _BUSINESS2
{
uint32_t taxId;
char name[27];
char direction;
char street[35];
uint32_t addNo;
char city[30];
char state[3];
uint32_t zip; }
BUSINESS2;
```

Show the relative offset of each field in the structure from the beginning of the structure for the unpacked (default alignment) case:

For Packed, take proposed starting location, divide by size of variable type.
if no remainder, no padding is needed, otherwise add padding bytes
equal to the remainder

uint32_t taxId;	is 0 mod 4 = 0, yes => valid start	0 -> 3
char name[27];	is 4 mod 1 = 0, yes => valid start	4 -> 30
char direction;	is 31 mod 1 = 0, yes => valid start	31 -> 31
char street[35]	is 32 mod 1 = 0, yes => valid start	32 -> 66
uint32_t addNo;	is 67 mod 4 = 0, no => need padding +1	68 -> 71
char city[30];	is 72 mod 1 = 0, yes => valid start	72 -> 102
char state[3];	is 102 mod 1 = 0, yes => valid start	102 -> 104
uint32_t zip;	is 105 mod 4 = 0, no => need padding +3	108 -> 111

since we started at zero, we add 1 so total = 112
Un-packed (STANDARD!)
total = 112

Show the relative offset of each field in the structure from the beginning of the structure for the packed case:

uint32_t taxId;	4	0 -> 3
char name[27];	27	4 -> 30
char direction;	1	31 -> 31
char street[35];	35	32 -> 66
uint32_t addNo;	4	67 -> 70
char city[30];	30	71 -> 100
char state[3];	3	101 -> 103
uint32_t zip;	4	104 -> 107

total = 108 byte

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

a. int32_t sumS32(const int32_t x[], uint32_t count)
   // returns sum of the values in the array (x) containing count entries.

b. int32_t dotpS32(const int32_t x[], const int32_t y[], uint32_t count) // returns the dot product of the values in the arrays (x and y) containing count entries.

c. uint32_t countAboveLimit(const int32_t x[], int32_t limit, uint32_t count) // returns number of values in the array (x) containing count entries that are > limit

d. int32_t findCityAligned (const char city[], const BUSINESS2 business[], uint32_t count)
   // returns the index of the first entry in the array (business) containing count entries which matches the requested city. If the city is not found, return a value of -1. You can assume that C default alignment is used for this problem.

e. int32_t findCityPacked (const char city[], const BUSINESS2 business[], uint32_t count)
   // returns the index of the first entry in the array (business) containing count entries which matches the requested city. If the city is not found, return a value of -1. You can assume that C packing is used for this problem.

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

a. 0.125 s = _0__, e = _00111110_, m = _00000000000000000000000__________________

32b hex value = __0x3E000000____

b. -15.5625 s = __1_, e = 10000010___________, m = _11110010000000000000000___________

32b hex value = _0xC1790000____

c. 0.33… s = _0__, e = _01111101__________, m = _01010101010101010101010___________

32b hex value = _0x3EAAAAAA____

d. 251.2578125 s = _0_, e = _10000110__________, m = _11110110100001000000000___

32b hex value = _0x437B4200____

e. 2048.0000001 s = _0_, e = _10001010__________, m =_00000000000000000000000____

32b hex value = _0x45000000____

4. Assume float x = 1048576.

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 floatingpoint number

that can be added together without a loss of accuracy? Consider the cases you add a number to x =

1048576 or x = 2097151.

a)

Since 104857's (2^{20}) mantissa is all zeros, the smallest we could add would set the least significant bit to 1.

```
                                      |
S   Exp       Mantissa                v
0 10010011 00000000000000000000000
```

changing to:

0 10010011 00000000000000000000001

With a mantissa of 23 digits we have $2^{(20-23)}$ so 2^{-3}

$2^{-3} = 1/8$ or 0.125

b)

For any given number expressed as 2^n, the smallest value we could add would be 2^{n-23} without losing accuracy.

So, as a ratio, of $2^n:2^{n-23}$ then substituting n with 23, we get $2^{23}:2^0$ or $2^{23}:1$