CS2100 Computer Organization
Tutorial 1: C and Number Systems
(Week 3: 26 Aug – 30 Aug 2024)

1. In 2's complement representation, "sign extension" is used when we want to represent an *n-bit* signed integer as an *m-bit* signed integer, where $m > n$. We do this by copying the MSB (most significant bit) of the *n-bit* number $m - n$ times to the left of the *n-bit* number to create the *m*-bit number.

   For example, we want to sign-extend 0110_{2s} to an 8-bit number. Here $n = 4$, $m = 8$, and thus we copy the MSB bit 0 four (8 – 4) times, giving 00000110_{2s}.

   Similarly, if we want to sign-extend 1010_{2s} to an 8-bit number, we would get 11111010_{2s}.

   Show that IN GENERAL, sign extension is value-preserving. For example, $00000110_{2s} = 0110_{2s}$ and $11111010_{2s} = 1010_{2s}$.

2. We generalize (r – 1)'s-complement (also called *radix diminished complement*) to include fraction as follows:

   $$\textbf{(}r - 1\textbf{)'s complement of } N = r^n - r^{-m} - N$$

   where n is the number of integer digits and m the number of fractional digits. (If there are no fractional digits, then $m = 0$ and the formula becomes $r^n - 1 - N$ as given in class.)

   For example, the 1's complement of 011.01 is $(2^3 - 2^{-2})$ – 011.01 = (1000 – 0.01) – 011.01 = 111.11 – 011.01 = 100.10. (Since 011.01 represents the decimal value 3.25 in 1's complement, this means that -3.25 is represented as 100.10 in 1's complement.)

   Perform the following binary subtractions of values represented in 1's complement representation <u>by using addition</u> instead. (Note: Recall that when dealing with complement representations, the two operands must have the same number of digits.)

   (a) 0101.11 – 010.0101
   (b) 010111.101 – 0111010.11

   Is sign extension used in your working? If so, highlight it.

   Check your answers by converting the operands and answers to their actual decimal values.

3. Convert the following numbers to fixed-point binary in 2's complement, with 4 bits for the integer portion and 3 bits for the fraction portion.

   (a) 1.75 (b) -2.5 (c) 3.876 (d) 2.1

   Using the binary representations you have derived, convert them back into decimal. Comment on the compromise between range and accuracy of the fixed-point binary system.

4. How would you represent the decimal value –0.078125 in the IEEE 754 single-precision representation? Express your answer in hexadecimal. Show your working.

5. Given the partial C program shown below, complete the two functions: **readArray()** to read data into an integer array (with at most 10 elements) and **reverseArray()** to reverse the array. For **reverseArray()**, you are to provide two versions: an iterative version and a recursive version. For the recursive version, you may write an auxiliary/driver function to call the recursive function.

```
#include <stdio.h>
#define MAX 10

int readArray(int [], int);
void printArray(int [], int);
void reverseArray(int [], int);

int main(void) {
   int array[MAX], numElements;

   numElements = readArray(array, MAX);
   reverseArray(array, numElements);
   printArray(array, numElements);

   return 0;
}

int readArray(int arr[], int limit) {

   // ...
   printf("Enter up to %d integers, terminating with a negative
integer.\n", limit);
   // ...
}

void reverseArray(int arr[], int size) {

   // ...
}

void printArray(int arr[], int size) {
   int i;

   for (i=0; i<size; i++) {
      printf("%d ", arr[i]);
   }
   printf("\n");
}
```

6. Trace the following program manually (do not run it on a computer) and write out its output. When you present your solution, draw diagrams to explain.

```
#include <stdio.h>

int main(void) {
   int a = 3, *b, c, *d, e, *f;

   b = &a;
   *b = 5;
   c = *b * 3;
   d = b;
   e = *b + c;
   *d = c + e;
   f = &e;
   a = *f + *b;
   *f = *d - *b;

   printf("a = %d, c = %d, e = %d\n", a, c, e);
   printf("*b = %d, *d = %d, *f = %d\n", *b, *d, *f);

   return 0;
}
```

Remember to post on the Canvas forum or QnA if you have any queries.