1 Homework 2

Due: Oct 4 **Max. Points**: 100

Please justify all answers.

To be submitted **on paper, during class**. **No late submissions**

Important Reminder: As per the course *Academic Honesty Statement*, cheating of any kind will minimally result in receiving an F letter grade for the entire course.

Note that using a computer can help solve some of these problems. However, to ensure that you understand the concepts fully, you should not make use of one. Recollect that you will not be allowed to use a computer during the exams.

In the answers which follow, underscores are used within number literals to aid readability.

1. One way of negating a 2's-complement integer N is to compute $1 + (\sim N)$. Prove that this procedure works. *10-points*

   Hint: Recall that the value of an n-bit number $b_{n-1}b_{n-2}\ldots b_1b_0$ is:

   $$-b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \ldots + b_1 \times 2^1 + b_0 \times 2^0$$

2. Given a 10-bit floating-point number with a 4-bit exponent field which follows IEEE floating point rules except for the above sizes:

   (a) What is the hexadecimal representation of the bit-pattern representing minus infinity?

   (b) Give the hexadecimal representation of the bit-pattern and the decimal value of the smallest positive number.

   (c) Give the hexadecimal representation of the bit-pattern and the decimal value of the smallest positive normalized number.

   (d) Give the hexadecimal representation of the bit-pattern and the decimal value for the most negative number which is not minus infinity.

   (e) Give the smallest and largest ULP values possible in this representation.

   You should use a calculator to provide exact answers for the decimal values. *15-points*

3. The floating point formats popular on most present-day computers use binary and hence cannot represent decimal numbers exactly. For example, the number `0.1` cannot be represented exactly in these formats. This can be a problem in situations like financial applications where exact decimal calculations are required.

There are various alternatives to binary floating point formats. This problem explores one alternative: the use of a Binary-Coded Decimal BCD floating point format.

The BCD representation of integers is straight-forward: each decimal digit is represented using its normal 4-bit binary encoding. For example, the decimal number `1234` is represented using 4 nybbles: `0001 0010 0011 0100`. Since there are two nybbles to a byte, the hex representation of the BCD representation of decimal `1234` is simply `0x1234`.

This problem deals with a 64-bit floating point BCD format `BcdFloat` where the decimal point can float within the least significant digits. Specifically, the 15 least significant nybbles of the 16 nybbles available within a 64-bit word are used to hold the 15 BCD-encoded digits of the number (with least-significant digit in the least-significant nybble). The MSB of the most significant nybble is used as a sign bit, with `1` representing a negative number. The remaining 3 bits of the most significant nybble give the position of the decimal point; specifically they are interpreted as an unsigned binary number giving the number of digits to the right of the decimal point.

Here are some examples of numbers in this representation (underscores are used for readability):

```
                    1:  0x00_00_00_00_00_00_00_01
                 1234:  0x00_00_00_00_00_00_12_34
                -1234:  0x80_00_00_00_00_00_12_34
                12.34:  0x20_00_00_00_00_00_12_34
               -12.34:  0xA0_00_00_00_00_00_12_34
        1.234:  0x30_00_00_00_00_00_12_34
       -1.234:  0xB0_00_00_00_00_00_12_34
  123456789012345:  0x01_23_45_67_89_01_23_45
 -123456789012345:  0x81_23_45_67_89_01_23_45
-12345678.9012345:  0xF1_23_45_67_89_01_23_45
```

(a) What is the ratio of the magnitude of the largest non-infinite IEEE double-precision float to the magnitude of the largest `BcdFloat`?

(b) What is the ratio of the magnitude of the smallest non-zero IEEE double-precision float to the magnitude of the smallest non-zero `BcdFloat`?

(c) The ratio of the magnitude of the smallest ULP magnitude for the IEEE double-precision float to the smallest ULP magnitude for the `BcdFloat`.

You should use a calculator to give a specific value for these ratios. *15-points*

4. Given the following declaration of a variable `s`, show the memory address

and contents of each individual byte of memory occupied by `s` in hexadecimal.

```
struct S {
  int val;
  int name[4];
  struct S *next;
} s = {
  0xdeadbeef,
  "42",
  &s
};
```

You should assume the following:

- `s` is allocated at `0x8000`.
- Fields in a `struct` are packed together without any wasted space.
- Characters are encoded in UTF-8.
- Pointers and `int`'s occupy 4 bytes.
- Multi-byte quantities use a little-endian byte ordering.

If the contents of a byte are undefined, indicate it by `uu`. *10-points*

5. Given the following C function:

```
void f(int *p1, int *p2) {
  *p1 = 5;
  *p2 = 6;
  printf("%d %d\n", *p1, *p2);
}
```

show a call to `f()` such that something **other** than a line containing `5 6` is printed. *10-points*

6. The solution to problem 7 for *Homework 1*, packed multiple integer values with limited range into a single `int` using explicit bit-twiddling. C provides an alternate way of achieving something similar using `struct` bit-fields.

   Use the web or any C books to research C `struct` bit-fields and then contrast the packing of multiple integral values into a single integer using explicit bit-twiddling with that done using C `struct` bit-fields. *10-points*

7. Find any bugs or inadequacies in the following program which purports to be a filter which copies its standard input to standard output while transforming any linefeed character `'\n'` in the input to a carriage-return `'\r'`, line-feed `'\n'` pair.

```
int main() {
  char c;
```

```
  do {
    c = fgetc(stdin);
    fputc(stdout, c);
    if (c == '\n') fputc(stdout, '\r');
  } while (c != EOF);
}
```

You should assume that all required header files have been included. You should consult the manual pages for the library functions used above. *15-points*

8. Discuss the validity of the following statements. What is more important than whether you ultimately classify the statement as **true** or **false** is your justification for arriving at your conclusion. *15-points*

   (a) An IEEE single-precision floating point number has multiple representations for zero.

   (b) C does not allow taking the address of a variable which points to an `int` as such an expression does not make any sense.

   (c) C does not allow adding 2 pointers as such an expression does not make any sense.

   (d) C does not allow adding a `int` to a pointer as such an expression does not make any sense.

   (e) The use of an array name in any expression (not a declaration) is transformed into a pointer to its first element.