Dalhousie University Faculty of Computer Science
CSCI 2122 Systems Programming
Assignment 1 Due: 28 Jan 2022, 11:30pm

Total [**29**]
Note: Please provide computations/justifications/explanations for each answer. Simply stating an answer will earn no credit.

(1) In a computer system, memory operations currently take up 30% of execution time. A new gadget called a *cache* (*i.e.* an L1 cache) speeds-up 80% of the memory operations by a factor of 4.
   (a) [**2**] What is the speed-up due to the cache?
   (b) [**3**] A second new gadget called an L2 cache speeds-up *half the remaining 20% of the memory operations* by a factor of 2. What is the total speed-up with the L1 and L2 cache together?
   Note: It is useful to draw pictures to solve this part of the problem

(2) (a) [**4**] In class, we examined an algorithm to generate the 2'sC of an integer (flip all bits; add 1). We also saw how any number in binary can be directly represented in hex. Putting the above two together, work out an algorithm that takes a number in hex and directly obtains its 2'sC representation in hex (*e.g.* $0xBEEF \rightarrow 0x4111$). Justify your answer - *i.e.* give reason(s) why your algorithm works correctly.
   (b) [**1+1+2**] The following bit pattern, $0xC0D$ was found in a 12-bit register. What is the decimal number represented if it is interpreted as: *(i)* unsigned binary; *(ii)* sign-magnitude; *(iii)* Two's complement?

contd. $\rightarrow$

(3) [**8**] This exercise is about the bit-wise operators in C. Complete each function skeleton using only straight-line code (i.e., no loops, conditionals, or function calls) and limited of C arithmetic and logical C operators. Specifically, you are only allowed to use the following eight operators: ! $\sim$, &, $\hat{|}$, + <<>>. For more details on the *Bit-Level Integer Coding Rules* on p. 128/129 of the text. A specific problem may restrict the list further: For example, to write a function to compute the bitwise `xor` of x and y, *only* using `& |, ~`

```
int bitXor(int x, int y)
{ return ((x&~y) | (~x & y));}
```

(a)
```
/*
copyLSbit:Set all bits of result to least significant bit of x
* Example:  copyLSB(5) = 0xFFFFFFFF, copyLSB(6) = 0x00000000
* Legal ops:  !  ~ &  ^  | + <<>>
*/
int copyLSbit(int x) {
return 2;}
```

(b)
```
/* negate - return -x
* Example:  negate(1) = -1.
* Legal ops:!  ~ &  ^  | + <<>>
*/ int negate(int x) { return 2; }
```

(c)
```
/* isEqual - return 1 if x == y, and 0 otherwise
* Examples:  isEqual(5,5) = 1, isEqual(4,5) = 0
* Legal ops:  !  ~ &  ^  | + <<>>
*/ int isEqual(int x, int y) { return 2; }
```

(d)
```
/* twoCmax:  return maximum two's complement integer
Legal ops:  !  ~ &  ^  | + <<>>
int twoCmax(void)
{ return 2; }
```

(4) [**8**] Suppose we number the bytes in a w-bit word from 0 (least significant) to $w/8?1$ (most significant).

Write code for the following C function, which will return an unsigned value in which byte i of argument x has been replaced by byte b:

```
unsigned replace_byte (unsigned x, int i, unsigned char b);
```

Examples:

`replace_byte(0x12345678, 2, 0xAB)` $\rightarrow$ `0x12AB5678`

`replace_byte(0x12345678, 0, 0xAB)` $\rightarrow$ `0x123456AB`