Homework 1

1、 Number Conversion

Please perform conversion on the following numbers

Octal	Binary	Decimal	Hexadecimal
2527	10101010111	1367	557
753	111101011	491	1eb
3746	11111100110	2022	7e6
177776	1111111111111110	65534	fffe

2、 Operations

(1) Now we have A: 0xF4, B: 0x11. Please compute A & B, A | B, A ^ B, ~A | ~B, A && B and A || B

Assume that both A and B are 8-bit integers, so that ~A and ~B could be determined.

```
A & B = 0x10, A | B = 0xf5, A ^ B = 0xe5, ~A | ~B = 0x0f,
A && B = true, A || B = true.
```

(2) Given two numbers x and y, I want to get a number that has the first half of x and the second half of y (such as x = 0x1111 1111, y = 0x0000 0000, result = 0x1111 0000). Please design a C program to achieve it.

```c
/**
 * @brief Get a number that has the first half of x and the second half of y.
 * Assume that both x and y are 32-bit integers.
 *
 * @param x
 * @param y
 * @return unsigned
 */
unsigned mix(unsigned x, unsigned y){
    return x & 0xffff0000u | y & 0x0000ffffu;
}
```

(3) Shift operations.

x		x << 5		x >> 3(logic)		x >> 3(arithmetic)	
Hex	Binary	Binary	Hex	Binary	Hex	Binary	Hex
0xD1	11010001	1101000100000	0x1a20	00011010	0x1a	11111010	0xfa
0x92	10010010	1001001000000	0x1240	00010010	0x12	11110010	0xf2
0x4F	01001111	0100111100000	0x840	00001001	0x09	00001001	0x09
0x36	00110110	0011011000000	0x6c0	00000110	0x06	00000110	0x06

If the type of x is “int8_t”, or rather, “signed char”, then x << 5 should be:

x \ x << 5	Binary	Hex
0xD1	00100000	0x20
0x92	01000000	0x40
0x4F	11100000	0x40
0x36	11000000	0xc0

3、 Two’s Complement Encodings

Assume we have an integer type of 8 bits, fill in the table below.

Value	Two’s complement
66	01000010
-21	11101011
127	01111111
-49	11001111

4、 Two’s Complement Multiplication

Fill in the following table showing the results of multiplying different 4-bit numbers under two’s complement.

x	y	x · y	Truncated x · y
[1000]	[0001]	1000	1000
[0100]	[0101]	10100	0100
[1101]	[0010]	11010	1010
[1110]	[1110]	11000100	0100

5、 Two's Complement

Determine the validity of the following expressions. Provide a mathematical explanation for expressions that are always true, and provide counterexamples for expressions that are not always true (ux, uy equals unsigned x, unsigned y).

- (x < y) == (-x > -y)

  Invalid.
  For instance, assume that both x and y are 32-bit signed integers.
  When x is -2147483648 and y is 0, x < y is true.
  However, -x is also -2147483648 and -y is 0, -x > -y is false.

- ((x + y) << 4) + y - x == 17 * y + 15 * x

  Valid.
  ((x + y) << 4) + y - x
  = (x << 4) + (y << 4) + y - x
  = x * 16 + y * 16 + y - x
  = 17 * y + 15 * x

- $\sim x + \sim y + 1 == \sim(x + y)$

  Valid.

  $\sim x + \sim y + 1$

  $= (\sim x + 1) + (\sim y + 1) - 1$

  $= -x - y - 1$

  $= -(x + y) - 1$

  $= \sim(x + y) + 1 - 1$

  $= \sim(x + y)$

- (ux - uy) == -(unsigned)(y - x)

  Valid.

  Signed or not, subtraction has the same form under two's complement.

  So, the following equations holds.

  (unsigned)x - (unsigned)y

  = (unsigned)(x - y)

  = (unsigned)(-(y - x))

  = -(unsigned)(y - x)

- ((x >> 2) << 2) <= x

  Valid.

  No matter the right shift operation is arithmetic or logical, (x >> 2) << 2 equals to x & 0xfffffffc.

  1) When $x \geqslant 0$, for any integer y, obviously x & y $\leqslant$ x holds, so ((x >> 2) << 2) $\leqslant$ x holds.
  2) When $x < 0$, declare ux as (unsigned) x. We have proved that ((ux >> 2) << 2) $\leqslant$ ux holds.

     Note that ux has the same binary expression as x. In other words, let y be ((x >> 2) << 2), then (unsigned)y =((ux >> 2) << 2).

     Consider the meaning of negative number x in two's complement, we have (real value of) ux = (real value of) x + 2^{32}. Now it is easy to prove that, if x, y < 0 and uy $\leqslant$ ux holds, then y $\leqslant$ x.

  Under both circumstances ((x >> 2) << 2) $\leqslant$ x always holds.