LAB01-B

question 1

We are running programs on a machine where values of type int are 32 bits. They are represented in two's complement, and they are right shifted arithmetically. Values of type unsigned are also 32 bits.

We generate arbitrary values x and y, and convert them to unsigned values as follows:

```
*/\* Creat**e some arbitr**ary values \*/*

*int x = random();*

*int y = random();*

*/\* Convert to unsigned \*/*

*unsigned ux = (unsigned) x;*

*unsigned uy = (unsigned) y;*
```

For each of the following C expressions, you are to indicate whether or not the expression always yields 1. If it always yields 1, describe the underlying mathematical principles. Otherwise, give an example of arguments that make it yield 0.

A. (x > 0) || (x - 1 < 0)

B. (x & 7) != 7 || (x<< 29 <0)

C. (x * x) >= 0

D. x < 0 || -x <= 0

E. x > 0 || -x >= 0

F. x + y == uy + ux

G. x*~y + ux * uy == -x

question 2

Floating point encoding. Consider the following 9-bit floating point representation based on the IEEE floating point format. This format does have a sign bit “s”– it can represent both negative and nonnegative numbers. • There are k = 4 exponent bits. • There are n = 4 fraction bits. Recall that numeric values are encoded as a value of the form

$$V = (-1)^s * M * 2^E$$

, where E is the exponent after biasing, and M is the significant value. （1）According to the floating point representation based on the IEEE floating point format, what is the value of the exponent bias in this question? （Please give your calculation process and answer. If there is no calculation process, you will get “zero”.） （2）Below, you are given some decimal values, and your task is to encode them in floating point format. In addition, you should give the rounded value of the encoded floating point number. To get credit, you must give these as whole numbers (e.g., 17) or as fractions in reduced form (e.g., 3/4). Any rounding of the significand is based on round-to-even, which rounds an unrepresentable value that lies halfway between two representable values to the nearest even representable value. If the representable value is enough big or enough small so that you cannot encode, you can give “+∞” or “−∞”to its rounded value. （Please give your calculation process, necessary text description and your final answers. If there is no calculation process, you will get “zero”.）

Value	Floating point bits	Rounded value
-53/4	(1)	(2)
1/16	(3)	(4)
-256	(5)	(6)
31/2048	(7)	(8)

question 3

Write C expressions, in terms of variable x, for the following values. Your code should work for any word size w ≥ 8. For reference, we show the result of evaluating the expressions for x = 0x87654321, with w = 32.

A. The least significant byte of x, with all other bits set to 0. [0x00000021].

B. All but the least significant byte of x complemented, with the least significant byte left unchanged. [0x789ABC21].

C. The least significant byte set to all 1s, and all other bytes of x left unchanged. [0x876543FF].

question 4

You have been assigned the task of writing a C function to compute a floating-point representation of 2^x You decide that the best way to do this is to directly construct the IEEE single-precision representation of the result. When x is too small, your routine will return 0.0. When x is too large, it will return +∞. Fill in the blank portion s of the code that follows to compute the correct result. Assume the function u2f returns a floating-point value having an identical bit representation as its unsigned argument.

```
float fpwr2(int x) {
/* Result exponent and fraction */
unsigned exp, frac; unsigned u;
if (x <___(1)__ ){
/* Too small. Return 0.0 */
exp = _(2)____ ; frac = ___(3)__ ;
} else if (x <___(4)__ ){
/* Denormalized result */
exp = __(5)___ ; frac = __(6)___;
} else if (x <__(7)___){
/* Normalized result. */
exp = __(8)___ ; frac = __(9)___;
} else {
 /* Too big. Return +∞ */
exp = ___(10)__ ; frac = __(11)___;
}
/* Pack exp and frac into 32 bits */
u = exp << 23 | frac;
/* Return as float */
return u2f(u);
}
```

question 5

Assume the following values are stored at the indicated memory addresses and registers: Address Value Register Value

Address	Value
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x11

Register	Value
%eax	0x100
%ecx	0x1
%edx	0x3

Fill in the following table showing the values for the indicated operands: Operand Value

Operand	Value
%eax	
0x104	
$0x108	
(%eax)	
4(%eax)	
9(%eax,%edx)	
260(%ecx,%edx)	
0xFC(,%ecx,4)	
(%eax,%edx,4)	

question 6

Completed the C code based on the Assembly code

```
#x at %ebp+8,y at %ebp+12,z at %ebp+16
mov 16(%esp),%eax
leal (%eax,%eax,2), %eax
sall $4,%eax
movl 12(%ebp),%edx
addl 8(%ebp),%edx
andl $65535,%edx
imull %edx,%eax
```

```
int arith(int x,  int y,  int z) {
    int t1 = (1);
    int t2 = (2);
    int t3 = (3);
    int t4 = t2 * t3;
    return t4;
}
```

question 7

Completed the Assembly code based on the C code

```
int shift_left2_rightn(int x, int n) {
    x <<= 4;
    x >>= n;
    return x;
}
```

```
(1) #get x
(2) #x<<=4
(3) #get n
(4) #x>>=n
```

question 8

Assume variables v and p declared with types

```
src_t v;
dest_t *p;
```

where src_t and dest_t are data types declared with typedef. We wish to use the appropriate data movement instruction to implement the operation

```
*p = (dest_t) v;
```

where v is stored in the appropriately named portion of register %eax (i.e., %eax, %ax, or%al), while pointer p is stored in register %edx. For the following combinations of src_t and dest_t,write a line of assembly code that does the appropriate transfer. Recall that when performing a cast that involves both a size change and a change of “signedness” in C, the operation should change the signedness first

src_t	**dest_t**	**Instruction**
int	int	movl %eax,(%edx)
char	int	(1)
char	unsigned	(2)
unsigned char	int	(3)
int	char	(4)
unsigned	unsigned char	(5)
unsigned	int	(6)