一.选择

1.（13 期中）对 x = $1\frac{1}{8}$ 和 y = $1\frac{3}{8}$ 进行小数点后两位取整(rounding to nearest even)， 结果正确的是 （ ）

A. $1\frac{1}{4}$, $1\frac{1}{4}$ B. 1, $1\frac{1}{4}$ C. $1\frac{1}{4}$, $1\frac{1}{2}$ D. 1, $1\frac{1}{2}$

2.（13 期中）已知函数 int x(int n) { return n*____; } 对应的汇编代码如下：

```
lea (%rdi, %rdi, 4), %rdi
lea (%rdi, %rdi, 1), %eax
retq
```

请问横线上的数字应该是 （ ）

A. 4 B. 5 C. 2 D. 10

3.（14 期中）下面说法正确的是：

A. 数 0 的反码表示是唯一的

B. 数 0 的补码表示不是唯一的

C. 1000, 1111, 1110, 1111, 1100, 0000, 0000, 0000 表示唯一的 整数是 0x8FEFC000

D. 1000, 1111, 1110, 1111, 1100, 0000, 0000, 0000 如果是单精 度浮点表示，则表示的是- $(1.\ 110111111)_2*2^{31-127}$

答：（ ）

4.（15 期中） 给定一个实数， 会因为该实数表示成单精度浮点数而发生误差。不考虑 NaN 和 Inf 的情况， 该绝对误差的最大值为

A. 2^{103}

B. 2^{104}

C. 2^{230}

D. 2^{231}

参考信息：单精度浮点数阶码 8 位，尾数 23 位

5.（15 期中）关于浮点数，以下说法正确的是

A. 给定任意浮点数 a，b 和 x，如果 a>b 成立(求值为 1)，则一定 a+x>b+x 成立

B. 不考虑结果为 NaN、Inf 或运算过程发生溢出的情况，高精度浮点数一定 得到比低精度浮点数更精确或相同的结果

C. 不考虑输入为 NaN、Inf 的情况，高精度浮点数一定得到比低精度浮点数 更精确或相同的结果

D. 给定任意浮点数 a, b 和 x，如果 a>b 不成立(求值为 0)，则一定 a+x>b+x 不成立。

6.（17期中） 若我们采用基于 IEEE浮点格式的浮点数表示方法，阶码字段 (exp)占据 k 位， 小数字段 (frac) 占据 n 位，则最小的规格化(normalized) 的正数是 _____ (结果用含有 n,k 的表达式表示)

A. $(1 - 2^{-n}) * 2^{-2^{k-1}+2}$ B. $1 * 2^{-2^{k-1}+2}$

C. $2^{-n} * 2^{-2k-1+2}$　　D. $(1-2^{-n}) * 2^{-2k+1}$

7.（17期中） 在下列的 x86-64 汇编代码中，错误的是：

A. movq %rax, (%rsp)

B. movl $0xFF, (%ebx)

C. movsbl (%rdi), %eax

D. leaq (%rdx, 1), %rdx

8.（18期中）下列哪种类型转换既可能导致溢出、又可能导致舍入？

A) int 转 float　B) float 转 int　C) int 转 double　D) float 转 double

9.（18期中）在 x86-64 下，以下哪个选项的说法是错误的？

A) movl 指令以寄存器作为目的时，会将该寄存器的高位 4 字节设置为 0

B) cltq 指令的作用是将%eax 符号扩展到%rax

C) movabsq 指令只能以寄存器作为目的

D) movswq 指令的作用是将零扩展的字传送到四字节目的

二．填空与解答

1.　（13期中）

请按 IEEE 浮点标准的单精度浮点数表示下表中的数值，首先写出形如$(-1)^s \times M \times 2^E$ 的表达式，然后给出十六进制的表示。

注：单精度浮点数的字段划分如下：

符号位 (s)：1-bit；阶码字段 (exp)：8-bit；小数字段 (frac)：23-bit；偏置值 (bias)： 127。

Value	$(-1)^s \times M \times 2^E$ ，1<=M<2	Hex representation
$-1\frac{1}{2}$		
2^{-149}		

2.（14 期中）

请按 IEEE 浮点标准的单精度浮点数表示下表中的数值， 首先写出形如 $(-1)^s \times M \times 2^E$ 的表达式，然后给出十六进制的表示。

注：单精度浮点数的字段划分如下：

符号位(s)： 1-bit；阶码字段(exp)： 8-bit；小数字段(frac)： 23-bit； 偏置值 (bias)： 127。

Value	$(-1)^s \times M \times 2^E$，1<=M<2	Hex representation
0.375		
-12.5		

3.（15 期中）

2.考虑一种 12-bit 长的浮点数，此浮点数遵循 IEEE 浮点数格式， 请回答下列问题。

注：浮点数的字段划分如下：

符号位(s)： 1-bit；阶码字段(exp)：4-bit；小数字段(frac)：7-bit。
1) 请写出在下列区间中包含多少个用上面规则精确表示的浮点数(每空 2 分， 共 4 分)
A： [1,2)
B: [2,3)
2) 请写出下面浮点数的二进制表示（每空 1 分，共 6 分）

数字	二进制表示
最小的正规格化数	
最大的非规格化数	
$17\frac{1}{16}$	
-1/8192	
$20\ \frac{3}{8}$	
$-\infty$	

4. （17 期中）

考虑有一种基于 IEEE 浮点格式的 9 位浮点表示格式 A。格式 A 有 1 个符号位， k 个阶码位， n 个小数位。现在已知$\frac{-9}{16}$的位模式可以表示为“101100010”，请回答以下问题：(注：阶码偏移量为 $2^{k-1}-1$)
1. 求 k 和 n 的值。(1 分)

2. 基于格式 A，请填写下表。值的表示可以写成整数(如 16)，或者写成分数(如 17/64)。 (注:每格 2 分)

描述	二进制位表示	值
最大的非规格化数		
最小的正规格化数		
最大的规格化数		

3. 假设格式 A 变为 1 个符号位， k+1 个阶码位， n-1 个小数位， 那么能表示的实数数量会怎样变化，数值的精度会怎样变化？(回答增加、 降低或不变即 可) (2 分)

5. （18 期中）

请补全函数 nlz，使之可以用来计算 unsigned 类型数据的二进制 表示中**前导零**的个数(提示：填写数字)：

```
// TODO : Complete the `nlz` function to
//              count the number of leading zeros
int nlz(unsigned x) {
  double w = 0.5;
  double y = (double)x + w;
  long long z;
  memcpy(&z, &y, sizeof(y));
  return_____- (z >>_____);
}
```

上式中 w 的取值并不唯一。请判断以下哪个/哪些数字也符合w 的要求：
(a) 0.2 (b) 0.3 (c) 0.9 (d) 1.2

下面所示的 strlen_x 是字符串求长度的函数在**大端序**电脑中的 一种实现，nlz 函数的意义如前所述。

```
int func_x(unsigned x) {
  unsigned u = 0x7F7F7F7F;
  unsigned y = (x & u) + u;
  y = ~(y | x | u);
  return nlz(y) >> 3;
}

size_t strlen_x(const char *s) {
  size_t t, n = 0;
  const unsigned *si = (const unsigned *)s;
  while ((t = func_x(*si++)) == 4) { n += t; }
  return n + t;
}
```

不改变 strlen_x 的实现，请修改 func_x 以使得 strlen_x 适用于**小端序**的电脑(提示：仅使用常数、算术/逻辑运算符和已定义变量)：

```
int func_x(unsigned x) {
  unsigned u = 0x7F7F7F7F;
  unsigned y = (x & u) + u;
  y = ~(y | x | u);
  return (___- nlz((___) & (y-1))) >> 3;
}
```

参考答案：

一. 选择

1. D

x= 1.00100　　half way and down --> 1.00

y = 1.01100　　half way and up-->1. 10

2.D

第一条指令将%rdi 中的值*4 再和自己相加，等同于*5 放在%rdi 里，然后再自加一次返回，也就是*10

3. D

考察反码、补码

A. 错误。若用反码表示，则可表示为 00000000 或 11111111。

B. 错误。数 0 的补码表示是唯一的： +0 的补码=+0 的反码=+0 的原码=00000000；0 的原码和反码有两种， 补码只有一种。

C. 错误。如果是无符号数表示，则表示的是 0x8FEFC000；如果是补码表示，则表示的是-0x70104000（这个解释并不很 convincing）

D. 正确。

4.A。

计算过程：浮点数阶码 8 位，即最大指数为 127，尾数为 23 位，即最低位表示的数为 2^{-23} * 2^{127} = 2^{104}。那么误差在刚好落在最低位的一半的时候最大， 即 2^{103}。其他选项为干扰项， 如果求最大指数的时候忘记非规格化数或者对偶数舍入理解不深会算出 b，如果以为指数为非规格化数会算出 c 或 d。

5. D。

如果不出现 NaN 和 Inf，浮点数是满足单调性的，这时 a 和 d 都成立。 如果出现 NaN 和 Inf，那么任何时候比较运算的操作数有 NaN 和 Inf 就为 0，就 只有 d 成立。关于 b 和 c，由于运算的偶然性，高精度不一定比低精度精确。

6. B

阶码字段 exp：00...01；小数字段 frac：00...00；偏置值 bias:2k 一 1 一 1

7.B/D 都对

解析： B:地址寄存器必须是 64 位寄存器;D:括号里少了一个逗号

8.b，本题考查浮点数类型转换，参见中文版 86 页。

9. D，movswq 应该是符号扩展

二. 填空与解答

1.

Value	$(-1)^s$ x M x 2^E 1<=M<2	Hex representation
$-1\frac{1}{2}$	(-1) x 1.1 x 2^0	0xBFC00000
2^{-149}	1.0 x 2^{-149}	0x00000001

2.

Value	$(-1)^s$ × M × 2^E 1<=M<2	Hex representation
0.375	1.1 x 2^{-2}	0x3EC00000

-12.5	$(-1) * 1.1001*2^3$	0xC1480000

3.

1) A.128　　B.64

2)

数字	二进制表示
最小的正规格化数	0 0001 0000000
最大的非规格化数	0 0000 1111111
$17\frac{1}{16}$	0 1011 0001000
-1/8192	1 0000 0000001
$10\frac{3}{8}$	0 1010 0100110
$-\infty$	1 1111 0000000

4.

1）答案： k = 4，n = 4

2）

描述	二进制位表示	值
最大的非规格化数	0 0000 1111	15/1024
最小的正规格化数	0 0001 0000	1/64
最大的规格化数	0 1110 1111	248

3）

答案：小数位变少后， NaN 的数量减少了，所以实数数量增加，数值精度降低

5.

1）

memcpy 将整个内存中的值直接移动了过去，也就是我们直接把一个数的浮点表示解释成了 long long。

我们考察一个一般的 unsigned，其形式可以写成$(00...0001xx...xx)_2$，假设其有 k 个前导 0，那么其的浮点表示应为$1.xx...xx*2^{31-k}$，那么我们想计算的 k 会出现在浮点数的指数部分，也就是要考察阶码段，阶码部分应该满足 e-bias=31-k，其中对 double 而言 bias=1023，于是 k=31+bias-e=1054-e，我们只需求出 e 的值

而求出 e 的值我们只需右移整个浮点数，让阶码部分出现在最低位即可（因为符号位一定为 0），我们右移去掉所有尾数得到的就是阶码，而 double 的尾数部分共有 52 位，因此答案为 1054,52

2）

那我们来考虑为什么要+0.5？原因在于考虑全 0 的情况，此时其浮点表示是非规格化的，无法用上述的方法求出结果。加上 0.5 以后实际的形式是$(00...00.1)_2$，也就是$1.0*2^{-1}$，此时格式是正确的，因为此时 k=32，-1=31-k，这样我们可以正常如上操作。

那么也就是说，我们希望加上的小数满足如下两个条件：第一，不能影响原数据的大小，即不能大于等于一；第二，必须形如$(.1xxxx)_2$的形式，这样才能正常操作

否则，比如我们选取 0.25，那么形式就变成了$(00...00.01)_2$，也就是$1.0*2^{-2}$，这样不成立我们上面说的 e-bias=31-k，无法求出正确结果，于是 w 的取值范围为[0.5, 1)，因此只有 c 符合要求。

3）

func_x 在干什么？我们取一个字节来看：对于 u1=0x7f=(01111111)$_2$,若 x1=0，那么(x1&u1)+u1=u1=(01111111)，而 y=~(y|x|u)=~u=(10000000)；若 x1≠0 且 x1 不等于(10000000)，那么(x1&u1)≠0，那么(x1&u1)+u1=(1xxxxxxx)，这样(y|u|x)=(11111111)，于是 y=~(y|u|x)=(00000000)，同样，若 x1=(10000000)，我们有(y|u|x)=(11111111),仍然有 y=~(y|u|x)=(00000000)

这样我们说明了：对于某个字节，若这个字节不是 0，func_x 的计算会产生全 0，否则会产生全 1

那么这个功能如何用来处理字符串长度呢？

一个字符串从低地址到高地址的字节序列是这样组织的：≠0，≠0，≠0...≠0，≠0，=0（字符串结尾的 NULL）

于是为了测量其长度，我们将每四个连续的字节转化为一个 unsigned，那么就会出现以下两种情况：

≠0，≠0，≠0，≠0（四个字节均不为零），此时操作后会生成全 0，那么计算的前导零个数是 32，除以 8 即为 4

...，=0，...（前面的字节不为 0，后面某个字节为 0，再后面不知道），此时操作后前面的字节会生成全 0，而为 0 的字节则会生成全 1，这样同样前导 0 的个数除以 8 即为前面非 0 的数的个数，同时这个数字不为 4 说明我们已经遇到了字符串的结尾，应当结束了。

但这里的问题是，只有满足大端法的机器才能这样操作，而在小端法机器上，最后一组会正好反过来，这样其长度就不对了。

也就是说，这时我们需要考察的是后导 0 的个数。那如果我们把后导 0 改写成 1，前面的都写成 0，那我们要求的就是 32-前导 0 的个数。

怎么做呢？观察题中出现的 y-1,我们知道经过上面操作后的 y 实际上就变成了(xx...xx1111111100...00)，而 y-1 就是(xx...xx1111111011...11)，那么我们只想要后面的 1 就很容易了——我们把整个 y 取反再和 y-1 按位与即可，这样前面的都会被消成 0，而后导 0 会变成 1，按位与之后还是 1，然后计算出前导 0 的个数，用 32 一减就是后导 0 的个数了。

值得说明的是，*p++的语义是*(p++)，而*(p++)的语义是*p,p++