chapter 2 Prictice Problems

*CSAPP 2ed*

# Prictice Problems

### 2.1

本题要做进制转换。进制是表示数字概念的机制。同一个数字，可以用不同的进制表示。

对于某一表示，比如“34”，我们可以从不同进制角度理解该符号的含义。所谓进制转换，是指某一进制理解特定符号，然后根据另一进制机制，生成新的符号。

以下是具体的转换结果：

A. 0x39A7F8 -> 0011 1001 1010 0111 1111 1000

B. 1100 1001 0111 1011 -> 0xC97B

C. 0xD5E4C -> 1101 0101 1110 0100 1100

D. 10 0110 1110 0111 1011 0101 -> 0x26E7B5

### 2.2

二进制数最简单的形式，是由一个1和多个0组成的序列。本题是在探讨这种简单形式在不同进制下的等价表示。

|  |  |  |
| --- | --- | --- |
| n | (Decimal) | (Hexadecimal) |
| 9 | 512 | 0x200 |
| 19 | 524,288 | 0x80000 |
| 14 | 16,384 | 0x4000 |
| 16 | 65,536 | 0x10000 |
| 17 | 131,072 | 0x20000 |
| 5 | 32 | 0x20 |
| 7 | 128 | 0x80 |

### 2.3

Byte是包含8个bit的序列。bit是有两种可能值的任意对象。在Binary notation中，我们用0或1区分这两种值。那么，我们可以用8个二进制数（0或1）构成的序列表示byte。

我们可以将二进制序列转换成其他进制表示。

|  |  |  |
| --- | --- | --- |
| Decimal | Binary | Hexadecimal |
| 0 | 0000 0000 | 0x00 |
| 167 | 1010 0111 | 0xA7 |
| 62 | 0011 1110 | 0x3E |
| 188 | 1011 1100 | 0xBC |
| 55 | 0011 0111 | 0x37 |
| 136 | 1000 1000 | 0x88 |
| 243 | 1111 0011 | 0xF3 |
| 82 | 0101 0010 | 0x52 |
| 172 | 1010 1100 | 0xAC |
| 231 | 1110 0111 | 0xE7 |

### 2.4

Hexadecimal是数字的一种表示方式。

A. 0x503c + 0x8 = ~~0x5045~~ 0x5044

B. 0x503c - 0x40 = ~~0x4eec~~ 0x4ffc

C. 0x503c + 64 = 0x503c + 0x40 = 0x507c

D. 0x50ea – 0x503c = ~~0xac~~ 0xae

进位理解错误：正确的理解是0x10 = 0xf + 0x1，而我错误的理解为0x10 = 0xf。因此导致计算出错。

### 2.5

Int类型size是4bytes。第一行定义一个int类型数字val，值是0x87654321，正好是4bytes。本题要理解Little endian和Big endian。以下分别用L和B表示

Little endian

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| … | 21 | 43 | 65 | 87 | … |

Big endian

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| … | 87 | 65 | 43 | 21 | … |

具体答案如下：

A. L: 21 B: 87

B. L: 21 43 B: 87 65

C. L: 21 43 65 B: 87 65 43

### 2.6

本题比较Int和float两种类型数字在byte层面的表示。

A. 0x00359141 -> 00000000001101011001000101000001,

0x4A564504 -> 01001010010101100100010100000100

B. 0x00359141 -> 00000000001101011001000101000001,

0x4A564504 -> 01001010010101100100010100000100

黄色部分为两个二进制序列的重合部分。该部分的包含21个二进制数。

C. 0x00359141 -> 00000000001101011001000101000001,

0x4A564504 -> 01001010010101100100010100000100

绿色部分是两个序列的非重合部分。

为什么不同呢？int和float机制不同。

### 2.7

本题要理解“编码”的概念。这里我们要用ASCII编码将字符串序列转换为十六进制数序列。

Output = 61 62 63 64 65 66

### 2.8

定义运算是在符号之间建立关系。

|  |  |
| --- | --- |
| Operation | Result |
| A | [01101001] |
| B | [01010101] |
| ~a | [10010110] |
| ~b | [10101010] |
| a&b | [01000001] |
| a|b | [01111101] |
| a^b | [00111100] |

### 2.9

通过本题，我们知道，可以用符号表示现实世界的概念。

A.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| R | G | B | Color | Complement RGB | Complement Color |
| 0 | 0 | 0 | Black | 111 | White |
| 0 | 0 | 1 | Blue | 110 | Yellow |
| 0 | 1 | 0 | Green | 101 | Magenta |
| 0 | 1 | 1 | Cyan | 100 | Red |
| 1 | 0 | 0 | Red | 011 | Cyan |
| 1 | 0 | 1 | Magenta | 010 | Green |
| 1 | 1 | 0 | Yellow | 001 | Blue |
| 1 | 1 | 1 | White | 000 | Black |

B.

Blue | Green = Red ~ 001 | 010 = 011

Yellow & Cyan = Green ~ 110 & 011 = 010

Red ^ Magenta = Blue ~ 100 ^ 101 = 001

### 2.10

根据题干，我们知道inplace\_swap与数据的location有关。Bit-level operation是从bit序列角度操作运算对象。如果x与y的值相等，会有什么情况？

参考p83，我们要用到XOR运算的三个性质：

|  |  |  |
| --- | --- | --- |
| Step | \*x | \*y |
| Initially | *a* | *b* |
| Step 1 | *a* | *a^b* |
| Step 2 | *a^(a^b)*  *=(a^a)^b*  *=0^b*  *=b* | *a^b* |
| Step 3 | *b* | *(a^(a^b))^( a^b)*  *=b^a^b*  *=a* |

### 2.11

本题用inplace\_swap交换整数数组中的元素值。题中指出在数组长度为奇数时，会出问题。

如果两个元素保存在同一个location，那么，在调用inplace\_swap过程中，改变一个元素的时候，会影响另一个元素的值。

A. k+1.

B. 当first和second指向同一个元素时，我们却将它们作为不同元素对待。我们行分析inplace\_swap。在执行第一行后，由于x与y指向同一元素，那么它们的值相等，因此\*y=0。同时，因为x与y指向同一元素，所以\*x也等于0。在执行第二行时，相当于\*x=0^0。执行该行后，\*x与\*y都为0。同样的，在执行第三行时，等价于\*y=0^0。此时，\*x与\*y都为0。

C. 通过以上分析，我们知道不能让first和second指向同一元素。我们将for-loop的判断条件修改为first <= cnt/2-1。Cnt表示数组的长度，cnt/2表示整除。整除后减一得到first应当指向的最后一个元素的下标值。

### 2.12

本题是要应用masking operations。写出C表达式，意味着可以使用算术运算。

A. 0x87654321->0x00000021。根据p86的论述，我们得到下述表达式

1 int x=0x87654321;

2 x = x & 0xFF

最终，x=0x00000021。

B. 0x87654321->0x789ABC21。

1 int x=0x87654321;

2 int y = x & 0xFF;

3 x = ~x + 2\*y – 0xFF;

在第一个表示中x被赋值为0x87654321。根据A，第二个表达式中，y定义为0x00000021。第三个表达式等价为:

x = ~ 0x87654321 + 2 \* 0x00000021 – 0xFF

x = 0x789ABCDE + 0x00000021 – 0xFF + 0x00000021

x = 0x789ABCFF – 0xFF + 0x00000021

x = 0x789ABC00 + 0x00000021

x = 0x789ABC21

C. 0x87654321->0x876543FF。从B中，我们可以看出，只要获得0x789ABC00后，再用~运算就可得到0x876543FF。具体表达式如下：

1 int x=0x87654321;

2 int y = x & 0xFF;

3 x = ~(~x + y – 0xFF);

第1、2行表达式与B中相同。第3个表达式的执行过程如下：

x = ~(~ 0x87654321 + 0x00000021 – 0xFF)

x = ~(0x789ABCDE + 0x00000021 – 0xFF)

x = ~(0x789ABCFF – 0xFF)

x = ~0x789ABC00

x = 0x876543FF

### 2.13

本题用两个基础函数实现or和xor运算。我只能是通过观察运算的效果来建立联系。

本题是要分析语义。

A. 我们假设一个序列z=0，然后设计下述表达式：

1 int z = 0x0;

2 bis(z, x);

3 bis(z, y);

第2、3表达式的语义是将x或y的为1的bit位，设置z中对应bit位的值为1。该语义等价于x|y。同时由于x=bis(z,x)，上述表达式简化为:

1 bis(x,y)

B. ~~xor的含义是要区分两个序列的不同处。那么根据题干，可以从bic角度思考实现方式。另外，在A中，我们实现了or语义。我们可以思考or与xor的联系。~~

~~bis(bic(y,x), bic(x,y))~~

~~我们从masking角度理解bis和bic。Bis和bic都是masking operations。~~

~~本题是探讨masking operations和bool operations的关系。~~

从例子角度思考，以下例子包含所有模式

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | bis(Y,X) | bis(X,Y) | bic(Y, X) | bic(X, Y) |
| X = 0110  Y = 0101 | 0111 | 0111 | 0001 | 0010 |

通过观察，我们可以发现bic(Y, X)和bic(X, Y)都包含了部分结果。通过bis，我们将不同部分组合成最终结果：bis(0001, 0010)=0011。该结果等价于xor(X,Y)。

因此，结果为下述表达式：

bis(bic(X,Y),bic(Y,X))

### 2.14

注意区分两者的概念。

bit level ops = {~, &, |, ^}, logical ops = {!, &&, ||}

x = 0x69 = 01100110, ~x = 10011001, !x = 0

y = 0x39 = 00111001, ~y = 11000110, !y = 0

|  |  |  |  |
| --- | --- | --- | --- |
| Expr | Value | Expr | Value |
| x & y | 00100000  0x20 | x && y | 1  0x01 |
| x | y | 01111111  0x7F | x || y | 1  0x01 |
| ~x | ~y | 11011111  0xDF | !x || !y | 0  0x00 |
| x & !y | 0  0x00 | x && !y | 1  0x01 |

### 2.15

从Bit sequence角度思考，本题是在比较两个bit seq是否一致。在bit level上，我们可以用xor运算比较bit序列。如果两个序列一致，那么^运算的结果应是0；否则，结果不为0。然后，我们使用NOT operation，将结果转换为1或0。以下是表达式：

!(x^y)

### 2.16

本题题干很有启发意义。题干中区分了bit sequence与binary notation。至于题目本身，重点是理解shift operations。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | | Logical | | Arithmetic | |
| x | | x<<3 | | x>>2 | | x>>2 | |
| Hex | Binary | Binary | Hex | Binary | Hex | Binary | Hex |
| 0xC3 | 11000011 | 00011000 | 0x18 | 00110000 | 0x30 | 11110000 | 0xF0 |
| 0x75 | 01110101 | 10101000 | 0xA8 | 00011101 | 0x1D | 00011101 | 0x1D |
| 0x87 | 10000111 | 00111000 | 0x38 | 01000011 | 0x43 | 11100001 | 0xE1 |
| 0x66 | 01100110 | 00110000 | 0x30 | 00011001 | 0x19 | 00011001 | 0x19 |

### 2.17

二进制序列、十进制序列与十六进制序列都是bit sequence的表示方式。含义encoding机制决定。做为encoding机制，Unsigned和two’s complement将bit sequences赋予数学意义。

本题在二进制基础上，为十六进制序列定义数学意义。

### 2.18

不理解题目的意思

### 2.19

利用2.17解答本题。

### 2.20

### 2.21

本题涉及运算符的优先级、underflow、类型转换。

在Two’s Complement机制中，减一与加负一是等价操作吗？

在bit level，-2147483647-1与-2147483647-1U产生相同的bit patterns。但两者的含义不同。

前者用two’s complement理解，后用unsigned理解。

A. -2147483647-1 == 2147483648U

当我们从bit level观察时，要注意-1的bit pattern。

从bit层面观察，(-2147483647-1)的结果为2147483648，类型为signed。在执行==时，2147483648的类型转换为unsigned，值不变。最终结果为true。

B. -2147483647-1 < 2147483647

从bit层面观察，(-2147483647-1)的结果为2147483648，类型为signed。在执行==时，2147483648的类型转换为unsigned，值不变。最终结果为false。

C. -2147483647-1U < 2147483647

我们可以从不同角度(-2147483647-1U)表达式。

~~在bit level，该表达式的结果与A、B中一样。从unsigned角度观察，该序列的值是2147483648U。最终结果为false。~~

先应用T2U，将-2147483647转换成对应的unsigned值，为4294967295。上述表达式等价于：4294967295U-1U< 2147483647U，结果为false。

D. -2147483647-1 < -2147483647

从bit层面观察，(-2147483647-1)的结果为2147483648，类型为signed。最终结果为false。

E. -2147483647-1U < -2147483647

先应用T2U，将-2147483647转换成对应的unsigned值，为4294967295。上述表达式等价为：

4294967295U-1U< 4294967295U。结果为false。

### 2.22

本题应用B2T转换公式。目的是促进对expanding策略的理解。Expanding策略基于这一性质。

A.

B.

C.

### 2.23

本题关于shift operations和expanding operations。

Func1和func2差异在于int操作的顺序。在func2中，执行int类型变换后，right shift operation是arithmetic shift。

A.

|  |  |  |
| --- | --- | --- |
| w | func1(w) | func2(w) |
| 0x00000076 | 1. <<: 0x00076000 2. >>: 0x00000076 3. int: 0x00000076 | 1. int: 0x00000076，序列第一个元素是0 2. >>: 0x00076000，序列第一个元素是0 3. <<: 0x00000076，arithmetic shift |
| 0x87654321 | 1. <<: 0x54321000，序列第一个元素是0 2. >>: 0x00054321，logical shift 3. int: 0x00054321 | 1. int: 0x87654321，序列第一个元素是1 2. <<: 0x54321000，序列第一个元素是0 3. >>: 0x00054321，arithmetic shift |
| 0x000000C9 | 1. <<: 0x000C9000 2. >>: 0x000000C9 3. int: 0x000000C9 | 1. int: 0x000000C9，序列第一个元素是0 2. <<: 0x000C9000，序列第一个元素是0 3. >>: 0x000000C9，arithmetic shift |
| 0xEDCBA987 | 1. <<: 0xBA987000，序列第一个元素是1 2. >>: 0x000BA987，logical shift 3. int: 0x000BA987 | 1. int: 0xEDCBA987，序列第一个元素是1 2. <<: 0xBA987000，序列第一个元素是1 3. >>: 0xFFFBA987，arithmetic shift |

B. 描述每个computation的执行效果。

在func1和func2函数中，分别有三个运算，差异在于运算的执行顺序。

下面分别阐述每个运算的执行效果。在func1中，表达式等价为：

unsigned x = word << 24;

unsigned y = x >> 24;

int z = y;

在func2中的运算等价为：

int x = word;

int y = x << 24;

int z = y >> 24;

（补充）

### 2.25

本题要在bit level思考。

|  |  |  |  |
| --- | --- | --- | --- |
| Unsigned | | Two’s complement | |
| Original | Truncated | Original | Truncated |
| 0 | 0 | 0 | 0 |
| 2 | 2 | 2 | 2 |
| 9 | 1 | -7 | 1 |
| 11 | 2 | -5 | 3 |
| 15 | 7 | -1 | -1 |

### 2.26

本题思考由类型导致的程序BUG。

A. 第一个参数指向的字符串长度小于第一个参数指向的字符串长度时，该程序返回错误结果。

B. strlonger程序主体等价于下述表达式：

1 unsigned len\_s = strlen(s);

2 unsigned len\_t = strlen(t);

3 return len\_s – len\_t > 0;

当len\_s小于len\_t时，从数学意义上，（len\_s – len\_t）结果小于0。但是，其结果是unsigned类型，因此值将underflow。最终大于0。

C. 程序主体修改为：return strlen(s) > strlen(t);

### 2.27

本题利用overflow。

int uadd\_ok(unsigned x, unsigned y) {

return (x+y)>=x && (x+y>=y;

}

### 2.28

本题重点在理解概念的含义。

|  |  |  |  |
| --- | --- | --- | --- |
| x | |  | |
| Hex | Decimal | Decimal | Hex |
| 0 | 0 | 0 | 0 |
| 5 | 5 | 11 | B |
| 8 | 8 | 8 | 8 |
| D | 13 | 3 | 3 |
| F | 15 | 1 | 1 |

### 2.29

本题考察对Two’s complement规则的理解，参考Figure 2.24(p119)。特别注意边界情况。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | y | x+y |  | Case |
| -12  [10100] | -15  [10001] | -27  [100101] | 5  [00101] | 1 |
| -8  [11000] | -8  [11000] | -16  [110000] | -16  [10000] | 2 |
| -9  [10111] | 8  [01000] | -1  [111111] | -1  [11111] | 2 |
| 2  [00010] | 5  [00101] | 7  [00111] | 7  [00111] | 3 |
| 12  [01100] | 4  [00100] | 16  [10000] | 0  [0000] | 4 |

### 2.30

只有向上overflow，既可能向上也可能向下，因此本题与2.27不同。我们可以发现 x与y异号时，不会overflow。只有当x与y均为正数或负数时，才能overflow。我们分两种情况判断，设计以下函数

int tadd\_ok(int x, int y) {

// case 4

if (x > 0 && y > 0 && (x+y) < x)

return 0

// case 1

else if (x < 0 && y < 0 && (x+y) > x)

return 0

return 1

### 2.31

本题没有处理overflow。

如果sum = x + y overflow，那么sum – x 会 underflow。结果，sum-x仍等于y。

### 2.32

考虑边界

与negation有什么关系？

### 2.33

本题要理解Two’s complement negation。本题与2.28有联系。

下表中，第一列hex表示bit pattern；第二列decimal是two’s complement interpretation。

|  |  |  |  |
| --- | --- | --- | --- |
| x | |  | |
| Hex | Decimal | Decimal | Hex |
| 0 | 0 | 0 | 0 |
| 5 | 5 | -5 | B |
| 8 | -8 | -8 | 8 |
| D | -3 | 3 | 3 |
| F | -1 | 1 | 1 |

本题的内涵是什么？

与2.28比较后可以发现，the bit patterns generated by two’s complement and unsigned negation are the same.

### 2.34

本题参考figure 2.26(p123)。关键是理解公式2.17(p123)的内涵。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Mode | x | | y | | x\*y | | Truncated x\*y | |
| Unsigned | 4 | [100] | 5 | [101] | 20 | [010100] | 4 | [100] |
| Two’s Comp. | -4 | [100] | -3 | [101] | 12 | [001100] | -4 | [100] |
| Unsigned | 2 | [010] | 7 | [111] | 14 | [001110] | 6 | [110] |
| Two’s Comp. | 2 | [010] | -1 | [111] | -2 | [111110] | -2 | [110] |
| Unsigned | 6 | [110] | 6 | [110] | 36 | [100100] | 4 | [100] |
| Two’s Comp. | -2 | [110] | -2 | [110] | 4 | [000100] | -4 | [100] |

### 2.35

根据p116所述，modular addition forms an abelian group。

第一种情况：x=0。在第一行中，p等于0，没有overflow。在第二行中，||运算符的第一个operand值为true。根据logical operaor shortcircuit性质，不会继续求值第二个operand。最终返回true。

第二种情况：。

1. 我们基于公式2.17思考。

2.

Let *r =* ，我们得到

3.

只有当r=t=0时，y=q。

### 2.36

本题与2.32类似。

本题不能像2.31一样。

这里，我们从64位机器中long long类型的上下界出发分析是否会越界。

LL\_MAX\_64

LL\_MIN\_64

// case 1

x > 0

LL\_MIN\_64 / x <= y <= LL\_MAX\_64 / x

// case 2

x < 0

LL\_MAX\_64 / x <= y <= LL\_MIN\_64 / x

// case 3

x = 0

### 2.37

我应当从参数值的范围入手。

A. 当ele\_cnt值是负数时，修改的程序行为与原来的不符。

B. 根据int和unsigned int类型在32-bit机器上的上下界，计算出ele\_cnt\*ele\_size的取值范围。再选择合适的数据类型。

### 2.38

|  |  |  |
| --- | --- | --- |
|  | b=0 | b=a |
| k=0 | (a<<0)+0 = a\*1 | (a<<0)+a = a\*2 |
| k=1 | (a<<1)+0 = a\*2 | (a<<1)+a = a\*3 |
| k=2 | (a<<2)+0 = a\*4 | (a<<2)+a = a\*5 |
| k=3 | (a<<3)+0 = a\*8 | (a<<3)+a = a\*9 |

### 2.39

根据2.40，我认为是要考虑Two’s comp.。

通过观察Figure 2.26(p123)，我猜测要将Form B修改成：-(x<<n+1)+(x<<m)。

如果n是the most significant bit，那么k是负数。

Form A: -(x<<n)+ (x<<n-))+ ……+ (x<<m)

Form A经简单变换后得到Form B：-(x<<n+1)+(x<<m)

### 2.40

本题应从bit patterns和left shifts观察k。

### 2.42

本题要将正负两种情况结合起来。因此，我们在以下基础上做改进

(x < 0 ? x+(1<<k)-1 : x) >> k

根据题意，k = 4。x的正负性决定是否要加bias。而x的第1个bit决定x是否为负数。

如果x是负数，那么(x>>31)是-1；如果x是非负数，那么(x>>31)是0；当x是负数时，要加bias；当x是非负数时，不加bias。我们把条件组合成以下表示：

x+((1<<16)-1)\*(0 – (x >> 31)) >> 16

等价于以下

input : x, k

int is\_negative = 0 – (x >> (int\_size-1));

int bias = (1<<k) - 1;

x + bias\*is\_negative >> k

### 2.43

利用p131和p127的知识解决本题。

x = -(x<<5) -> x \* -32

(y < 0 ? y+7 : y ) >> 3 -> y / 8

M = -32，N = 8

### 2.44

本题涉及int和unsigned类型范围、类型转换、logical operations、算术运算。

A. ~~x取值可分为3个区间：。当时，，该表达式值为true；当时，，该表达式值为true；当时，，该表达式值为true。综上，该表达式值为true。~~

~~另一方面，我们可以证明该表达式的逆否命题为true。~~

~~没有考虑overflow。。。~~

如果x=TMin，那么x>0=false；x-1发生overflow，值为TMax，那么，(x-1)<0为false。因此，当x=TMin时，该表达式的值为false。

B. 我们只需分析当表达式的第一部分为false的时候，第一部分的值是否为true。如果是，则对所有x的取值，该表达式值都为true。

我们先分析表达式的第一部分：。只有当x末三位bit为[111]时，值为false。在这种情况下，x<<29 = [111000…000]，显然值小于0，因此(x<<29<0)为true。

C. 如果x\*x的值overflow，那么x\*x>=0可能为false。当时，

在这里，发生overflow，值为INT\_MIN。因此，最终x\*x小于0。

D. 注意int类型的上界。

当时，成立。

当时，，由于，所以，-x<=0成立。

综上，该表达式为true。

E. 注意int类型的下界。

当时，成立。

当时，，由于，所以-x发生overflow，值为-1。

综上，当x=INT\_MIN时，表达式值为false。

F. 当x=-1, y=0时，ux=UMAX,uy=0。

G. 从bit level观察，我们发现~y+y=-1。

参考figure2.26(p123).

y=1, ~y=-2

x=-1

当x=-1, y=1时，该表达式值为false。

### 2.45

本题是促进理解fractional value的binary representation。本题用的是positional notations。

|  |  |  |
| --- | --- | --- |
| Fractional value | Binary representation | Decimal representation |
|  | 0.001 | 0.125 |
|  | 0.101 | 0.75 |
|  | 1.1001 | 1.5625 |
|  | 10.1011 | 2.6875 |
|  | 1.001 | 1.125 |
|  | 101.111 | 5.875 |
|  | 11.0011 | 3.1875 |

### 2.46

A.

，，

B.

C.

D.

### 2.47

本题考察对IEEE floating point format的理解。参考Figure 2.34。

Case 1: *E = e – bias, M = 1 + f*

Case 2: *E = 1 – bias, M = f*

Case 3: *Infinity, NaN1*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bits | *e* | *E* |  | *f* | *M* |  | *V* | Decimal |  |
| 0 00 00 | 0 | 0 |  | 0 | 0 | 0 | 0 | 0 | Case 2 |
| 0 00 01 | 0 | 0 |  |  |  |  |  | 0.25 |
| 0 00 10 | 0 | 0 |  |  |  |  |  | 0.5 |
| 0 00 11 | 0 | 0 |  |  |  |  |  | 0.75 |
| 0 01 00 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | Case 1 |
| 0 01 01 | 1 | 0 | 1 |  |  |  |  | 1.25 |
| 0 01 10 | 1 | 0 | 1 |  |  |  |  | 1.5 |
| 0 01 11 | 1 | 0 | 1 |  |  |  |  | 1.75 |
| 0 10 00 | 2 | 1 | 2 | 0 | 1 | 2 | 2 | 2 |
| 0 10 01 | 2 | 1 | 2 |  |  |  |  | 2.5 |
| 0 10 10 | 2 | 1 | 2 |  |  | 3 | 3 | 3 |
| 0 10 11 | 2 | 1 | 2 |  |  |  |  | 3.5 |
| 0 11 00 | - | - | - | - | - | - | Infinity | - | Case 3 |
| 0 11 01 | - | - | - | - | - | - | NaN | - |
| 0 11 10 | - | - | - | - | - | - | NaN | - |
| 0 11 11 | - | - | - | - | - | - | NaN | - |

### 2.48

先根据Hex写出Binary notations，然后从two’s complement和IEEE floating point format出发理解差异。

|  |  |  |  |
| --- | --- | --- | --- |
|  | Value | Hexadecimal | Binary Notation |
| Integer | 3510593 | 0x00359141 | 0000 0000 0011 0101 1001 0001 0100 0001 |
| Floating point | 3510593.0 | 0x4A564504 | 0100 1010 0101 0110 0100 0101 0000 0100 |

|  |
| --- |
| 0000 0000 0011 0101 1001 0001 0100 0001 |
| 0100 1010 0101 0110 0100 0101 0000 0100 |

### 2.49

A. 从1开始，寻找不能完整表示的smallest positive integer。

从bit level思考。n bits，整数范围。

既然是正数，那么，可以从Left shift角度看待floating point format。

|  |  |  |
| --- | --- | --- |
| s | e | f |

f是不为0的binary sequence。

那么，。

从Left shift角度观察，。根据题干，k可为任意数值。

B. 代入上述公式即可。

### 2.50

本题应用Round-to-even规则。

|  |  |  |  |
| --- | --- | --- | --- |
| Origin | Before | Round-to-even | After |
|  |  | （是halfway，舍去） |  |
|  |  | （不是halfway，进位） |  |
|  |  | （是halfway，进位） |  |
|  |  | （不是halfway，舍去） |  |

### 2.51

在2.46基础上.

### 2.52

分两部分。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Format A | |  | Format B | |
| Bits | Value |  | Bits | Value |
| 011 0000 | 1 |  | 0111 000 | 1 |
| 101 1110 |  |  | 1001 111 |  |
| 010 1001 |  |  | 0110 100 |  |
| 110 1111 |  |  | 1011 111 |  |
| 000 0001 |  |  | 0100 000 |  |

参考2.47计算value。

### 2.53

不知道

### 2.54

Wait…