Homework Problems

Chapter 2

## 2.55、2.56、2.57

略

## 2.58

本题要写一个函数判断机器编码是否为big endian。关键是从bit seq角度观察data types。

我可以在函数中实例化一个值为1的int类型变量，然后观察它的byte order。假设机器的word size是32 bit。那么，若是big endian，它的byte表示是：0x00000001；若是little endian，它的byte表示是：0x10000000。我们借鉴show\_bytes，定义byte\_pointer，取出上述变量的第一个byte，通过判断其值是否为0，以此判断byte order。代码见is\_big\_endian.c。

## 2.59

本题用一个C表达式实现从已有元素中组合出一个word size的bit sequence。注意没有说明word size的length，那么，既可能是32-bit，也可能是64-bit。同时没有规定input的size。

第一种方式：本题可以参考2.12，使用masking operations实现。

假设x与y是输入。我们用masking operations将x的最后两个bytes设为0，将y除了最后两个bytes之外的bits设为0。Masking code分别是(~0 – 0xFF)和0xFF。然后，将两者组合成最后的结果。表达式如下，代码见combine\_bytes

(x & (~0 – 0xFF)) | (y & 0xFF)

附注：需要考虑byte order吗？不需要，因为byte order已被抽象隔离。

第二种方式：用shift operations实现。

用shift可以去除x和y的指定bytes，但问题在于，我们不知道word size。因此，此处该方法不可行。

## 2.60

本题是对bytes进行操作。用byte pointer操作指定的byte，将其设为目标值即可。代码见put\_byte.c。

## 2.61

本题根据int类型的bit-pattern和value的关系作答。

A. 条件是x的任一bit为1。!x有两个值：1或0。若!x为0，则x的值不为0（为非零整数），其中至少有一个bit为1。这时，我们在!x基础上再取反，得到正确的结果。相应的表达式为!(!x).

B. 条件是x的任一bit为0。我们可以先用NOT运算，将问题规约到（A）。相应的表达式为!(!(~x).

C. 条件是x的the most significant byte中任一bit为0。利用get\_msb(p.115)，取出the most significant byte，将问题规约到（A），以相同方法给出表达式。

D. 条件是x的the least significant byte中任一bit为0。在get\_msb基础上，我们写出get\_lsb（get least significant byte），将问题规约到（B）。

代码见expressions.c.

## 2.62

本题要判断right shift是否为logical。我们可以利用two’s comp的特性设计程序。具体做法是先定义int类型变量x，值为-1，然后用right shift，x>>1。最后，可以（1）用2.61(b)中的表达式判断是否包含0。如果包含0，则是logical；否则不是。（2）也可以从x与－1的关系，得到结果。代码见int\_shifts\_are\_logical.c。

## 2.63

本题要将bit seq中的某些bit设为相应的值，我们用masking operations实现。

1. 在sra中，xsrl的前k位被设为0，我们的工作是将这些位设为x的第一个bit。假设x的第1为是i(0或1)，我们将Mask设置为[i…i]1..1。为了得到这个mask，我们先创建一个所有bit均为i的int变量。使用left shift，向左位移w-i个bit，得到mask，标记为m。执行m|xsrl运算，我们得到期望结果。下面是步骤：
2. 我们先要取得x的第1个bit信息。通过(-1&INT\_MIN)，我们获得mask [10…0]。然后用masking operation，得到sign=[i0…0]，i是x的第1个bit的值。若从bit level观察，。这部分的表达式为int sign = ((-1&INT\_MIN)&x)。
3. 接着，我们要取得一个所有bit都为i的bit seq。sign是int类型，接着，我们从 logical operations角度观察b： int c = !sign。从bit level观察，。下一步，int d = c \* -1。从int观察，；从bit level观察，。再使用inverse：int e = ~d；然后从bit level观察，。

这个部分的表达为：int sign\_seq = ~(!sign \* -1)。现在，我们得到一个所有bit都为i的bit seq。

1. 然后，我们用left shifts，得到期望的bit patterns，用作mask。这部分的表达式为int f = e << (w-k)。从bit level观察，。
2. 最后，将f做为mask，做masking operation：int g = f | xsrl，得到最终结果g。这部分的计算可以表示为：[ii…00] | [00…yy] => [ii…yy]。

代码见sra\_srl.c/sra。

1. 在srl中，xsrl的前k位被设为i，我们的工作是将这些位设为0。我们先得到mask设置为 [0…0]1..1，然后用mask与xsra做OR运算，得到期望的结果。
2. 创建一个所有bit均为1的int变：int a = -1
3. 使用left shift，向左位移w-i个bit，再取Inverse，得到mask，标记为m。表达式如下：int b = a << w-i; int c = ~b;
4. 执行masking operation，得到期望结果：xsrl = c&xsra

代码见sra\_srl.c/srl。

## 2.64

本题可用与2.65相似的角度理解。

原理：对序列中的所有even bit做“|”运算，得到一个bit，该bit表示该序列的even bit是否包含1。

实现：用|和shift运算不断将不同even bit的信息合成到一起。

## 2.65

对于本题，我的思考角度是信息的流动过程，关注的焦点包括目标信息、信息的来源。我认为，我们的任务是在已有信息与目标信息之间建立联系。或者说，我们要建立一个过程，让已有信息“流动”向目标信息。

原理：对所有bit做XOR运算，得到一个bit，该bit表示该序列是否包含even个1。

实现：用^和shift运算不断将不同bit的信息合成到一起。从寻找byte之间的差异角度思考，我们用^运算将4个bytes合成为一个bytes。

原始：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

用^运算合成

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

再用shift调整位置

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

然后用^比较，合成蓝色与黄色区域，重新着色后得到以下：

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

再用shift调整位置

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

然后用^比较，合成蓝色与黄色区域，重新着色后得到以下：

|  |  |
| --- | --- |
|  |  |

再用shift调整位置

|  |  |
| --- | --- |
|  |  |

重复上步骤后，我们得到一个bit。该bit能够说明bit seq中是否有even个1。具体代码见even\_ones.c。

## 2.66

从信息角度考虑，在leftmost one右边的bit是无效信息。参考2.65和Hint。在本题，我们先将第一个值为1的bit信息，赋值到后续bit上。这样，我们便可以得到一致的形式[0…011…1]。

原理：用right shift生成mask，用bit operation |，将后续bit改为1。

实现：假设原始输入x，如下

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |

用right shift，x>>1，得到mask

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 |  | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

使用|运算，a = x | mask，得到

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

用right shift，x>>2，得到mask

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 |  |  | 1 | 1 | 0 | 1 | 1 | 0 | 0 |

使用|运算，b = a | mask，得到

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |

用right shift，x>>4，得到mask

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 |  |  |  |  | 1 | 1 | 1 | 1 | 0 |

使用|运算，c = b | mask，得到

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

用right shift，x>>8，得到mask

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

使用|运算，d = c | mask，得到

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

这样，我们得到Hint中的形式。最后，我们用logical shift运算和addition，以进位的方式，从该形式中获得目标结果。这里有两种方式：1. (d >> 1) + 1；2. (d+1)>>1。考虑到当第一个bit为1时，所有bit都会是1，那么计算d+1时会overflow。再用logical shift，无法得到目标结果。因此这里我们应用第一种方式。结果如下

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## 2.67

略

A. overflow.

B.

int beyond\_msb = 2 << 30;

return !beyond\_msb;

C.

int beyond\_msb = 2 << 14;

return !beyond\_msb;

## 2.68

原理：用shift运算，将前（w-n）位设为0

实现：先用left shift，再用logical right shift。

x = x << (8\*sizeof(int) – n);

x = (unsigned) x >> (8\*sizeof(int) – n);

## 2.69

本题旨在转移信息，我们用masking operation实现。

**原理** 将后n位信息保存到mask中，对x做logical right shift，移动n位。对mask做left shift，移动(w-n)位。用bit operation|，组合mask和x。

**实现** 假设w=8, n=2, x如下

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |

将后2位保存为mask，m=

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

对mask做left shift，移动6位，m=

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

对x做logical right shift，移动2位

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |

对x做|m运算，x=

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |

## 2.70

本题要分析two’s comp.编码方式，要求用更少的bit表示相同数值(语义不变)。一个int类型的值有两种可能：非负整数、正整数。通过分析，我们将知道哪些bit patterns可以减少bit数量，且保持值不改变。

**非负整数** 在two’s comp. encoding中，如果first bit为0，则数值为非负。我们假设bit seq的前k位是0。那么，根据2’s comp. encoding，。（补充分析）

因此，我们可以将前k－1位bit去掉，且保持该数值不改变。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | …… | 0 | 1 | 0 |

**负整数** 在two’s comp. encoding中，如果first bit为1，则数值为负。我们假设bit seq的前k位是1。那么，根据2’s comp. encoding，。（补充分析）

. .

因此，我们可以将前k－1位bit去掉，且保持该数值不改变。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 0 | …… | 0 | 1 | 0 |

**实现** 在本题中，要判断能否用n位的bit seq表示目标数值。根据以上分析，如果原始bit seq的前(w-n+1)位的bit是一致的，则可行。这么一来，我们将问题规约判断bit pattern的一致性。我们将采用arithmetic right shift，在去掉无关信息的同时维护有效信息：y = (int) x >> (n – 1)。如果y等于0或-1，则y所有bit为0或1，该值就可以用n bit表示。表达式为 y == 0 || y == -1。

## 2.71

本题分析编码问题。

A. 没有保留int的符号。

B. 先left shift，再right shift。

## 2.72

本题涉及implicit type conversion。一个隐藏的知识点是：size\_t是unsigned类型。

A. 通常情况下，size\_t是unsigned类型。maxbytes是int类型。在条件判断语句中涉及类型转换。在执行减法前 ，maxbytes将转换为unsinged。减法运算的值将是unsigned。那么，如果maxbytes小于size\_t，结果会overflow，然后转换成对应的unsigned。或者说，maxbytes-sizeof(val)的值是unsigned类型，总是不小于0。因此，条件总是为true。

B. 能否直接用>=运算比较maxbytes和sizeof(val)？

## 2.73

本题有关two’s complement的addition overflow。

**原理** 将int类型分为两个部分考虑：符号位（第1个bit）、非符号部分（其余bit）。两个int数的加法有三种情况：normal, underflow, overflow。以下是各种情况的产生条件：

1. Normal分两种：(1) 符号不同；(2) 符号相同且非符号部分的加法正常进位；
2. Underflow：符号一致，均为负数，同时非符号部分的加法没有进位；
3. Overflow：符号一致，均为负数，同时非符号部分的加法产生进位；

**实现** ??

## 2.75

本题建立在tow’s complement multiplication的二进制形式上。参考Figure 2.26(p.123)。

**原理** 在、、x和y四者的bit patterns之间建立联系。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | 1 | 1 | 0 | 1 |
|  |  |  | \* | 1 | 0 | 1 | 0 |
|  |  |  |  | 0 | 0 | 0 | 0 |
|  |  |  | 1 | 1 | 0 | 1 |  |
|  |  | 0 | 0 | 0 | 0 |  |  |
|  | 1 | 1 | 0 | 1 |  |  |  |
|  |  | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | 0 | 1 | 0 | 1 |
|  |  |  | \* | 0 | 0 | 1 | 0 |
|  |  |  |  | 0 | 0 | 0 | 0 |
|  |  |  | 0 | 1 | 0 | 1 |  |
|  |  | 0 | 0 | 0 | 0 |  |  |
|  | 0 | 0 | 0 | 0 |  |  |  |
|  |  | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | -1 | 1 | 0 | 1 |
|  |  |  | \* | 0 | 0 | 1 | 0 |
|  |  |  |  | -0 | 0 | 0 | 0 |
|  |  |  | -1 | 1 | 0 | 1 |  |
|  |  | -0 | 0 | 0 | 0 |  |  |
|  | -0 | 0 | 0 | 0 |  |  |  |
|  |  | | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | -1 | 1 | 0 | 1 |
|  |  |  | \* | -1 | 0 | 1 | 0 |
|  |  |  |  | -0 | 0 | 0 | 0 |
|  |  |  | -1 | 1 | 0 | 1 |  |
|  |  | -0 | 0 | 0 | 0 |  |  |
|  | 1 | -1 | -0 | -1 |  |  |  |
|  |  | | | | | | |

若均为正数，SHP同UHP。x与y异号时，和的第一个bit都是0，后bits相同。。x与y均为负数时，和的第一个bit都是1，后bits相同。

由此可见，和的差异在之间的bits上。与分别表示x、y除第一个bit后的剩余bits，我们可知

由公式2.18(p.124)知道，。于是，我们知道

## 2.76

参考2.40。

本题的要点在于，将乘法看成shift运算的组合。

|  |  |  |
| --- | --- | --- |
| K | Bits of K | x\*K |
| 5 | 00000101 | x<<1 + x<<0 |
| 9 | 00001001 | x<<3 + x<<1 |
| 30 | 00011110 | x<<4 + x<<3 + x<<2 + x<<1, x<<5 – x<<1 |
| -56 | 11001000 | -(x<<16) + x<<8 + x<<4 |

## 2.77

参考2.42。

整数除法的注意点在于，当被除数小于0时选择合适的rounding方式。

表达式如下

input: x, k

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

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

x + bias\*is\_negative >> k

## 2.78

参考2.76和2.77。

如何处理Overflow？

如何处理除法rounding？

## 2.79

用除法和加法替代分数乘法，确保不会overflow。

注意当x为负时，要加bias，实现rounding toward zero。

## 2.80

题意是在不使用w，用m和n实现效果相应效果。

由于我们不确定是否为two’s complement，因此无法直接获得w长度的1序列。但能以间接方式获得：~0 -> 11…111。

A. 用~0获得w长度的1序列，再用left shift获得。表达式如下

all\_one\_seq = ~0

all\_one\_seq << n

B. 用~0获得w长度的1序列，再用left shift获得。将其作inverse操作，获得。再做left shift，得到。表达式如下

all\_one\_seq = ~0

ones\_n\_zeros = all\_one\_seq << n

zeros\_n\_ones = ~ones\_n\_zeros

zeros\_n\_ones\_m\_zeros = zeros\_n\_ones << m

## 2.80

参考2.44(p.133)。

本题涉及算术运算、类型转换、overflow、取值范围。

解决方法是发现数学性质，或举反例。

要点是理解表达式的意图，然后判断表示是否恰当。

A. 根据int类型取值范围的非对称性，从边界情况入手。当y-值为-128时，由于overflow，-y也为-128(见p.122) 。由此，我们构造以下反例

x = -127, y = -128, -x = 127, -y = -128

B. ~~该表达式可以理解为能否用 “==” 运算符的左右替代右边。我们分析左边部分~~

~~根据<<运算与multiplication的关系，等价于~~

~~假设two’s complement乘法具有distribute law，我们将该式转换为~~

~~two’s complement加法具有associative law，该式等价于~~

~~虽然，但是，当 overflow时，。比如，~~

~~因此，表达式不是一定为真。~~

做题首先要联系题目与知识点

本小题的核心是constants乘法。

该小题涉及4个知识点：

1. <<运算与乘法的联系
2. 乘法的分配律
3. 加法的结合律
4. Multiplying by constants

表达式左边是Form A和Form B的组合(p.128)。

根据multiplication的distribute law和addition的associative law，展开该式

参考p127。

31\*y = y<<4 + y<<3 + y<<2 + y<<1 + y<<0 = y<<5 – y<<0

33\*x = x<<5 + x<<0

因此，该表达式左右等价，值为永远为1。

C. 参考p122，inverse与negation的关系: 。

我们利用上述关系转换表达式

简化后得

显然无论x与y的值，表达式都不成立。

我们以x=y=0为例，由于~0=-1，那么(~x+~y)=-2; ~(x+y)=-1。可以看出表达式不成立。

D. 该小题涉及类型转换。由于int类型取值范围的非对称性，我们知道

例子：

因此，

同一个例子，我们来看不等式左边。

首先，

综上，当时，以下表达式值为0

E. ，相当于将x最后一个设为0. 无论x大于还是小于0，x的值都减小1.

如何从形式上说明？

在两种情况下

因此，表达式值为1.

## 2.82

A.

B.

假设

根据题意，我们可以知道

因此

由于，可知

(a)

(b)

(c)

## 2.83

根据Floating point 序列的排列规则，直接比较bit patterns

## 2.84

IEEE floating point format

A. (unsinged) 5.0 = 00000101

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| s | e | | | | | | f | | | | |
| 0 | 0 | … | 0 | 0 | 1 | 0 | 0 | 1 | 0 | … | 0 |

B. f部分bit都是1，e的值是(n+bias). n是f的长度。

能够准确表示的最大的奇数是：

C.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| s | e | | | | | | f | | | | |
| 0 | 0 | … | 0 | 0 | 0 | 1 | 0 | 0 | 0 | … | 0 |

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

Bit patterns如下

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| s’ | e’ | | | | | | f’ | | | | |
| 0 | 1 | … | 1 | 1 | 0 | 1 | 0 | 0 | 0 | … | 0 |

## 2.85

参考Figure 2.34.

e=0, f=1都有意义了。

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| s | e | | | | | | i | f | | | | |
| 1 | 0 | … | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | … | 1 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| s | e | | | | | | i | f | | | | |
| 1 | 0 | … | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | … | 0 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| s | e | | | | | | i | f | | | | |
| 1 | 1 | … | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | … | 1 |

## 2.86

本题从IEEE floating format出发，分析几个特征值的bit patterns。参考Figure 2.34。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| s | e | | | | | | | f | | | | | | | |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Case | Description | Hex | M | E | V |
| 0 | -0 | 8000 | 0 | -62 | ---- |
| 1 | Smallest value > 1 | 0100 | 1 | -62 |  |
| 2 | 256 | 7700 | 1 | 8 | ---- |
| 3 | Largest denormalized | 0077 |  | -62 |  |
| 4 |  | 7700 | ---- | ---- | ---- |
| 5 | Number with hex  Representation 3AA0 | ---- |  | -5 |  |

方法：如果值已知，如case 0, 2，我们从V=f(S, M, E)角度分析；其他情况，我们从bit patterns角度分析。

注意：figure 2.34 bit patterns的排列规则。

## 2.87

本题涉及format转换。

|  |  |  |  |
| --- | --- | --- | --- |
| Format A | | Format B | |
| Bits | Value | Bits | Value |
| 1 01110 001 |  | 1 0110 0010 |  |
| 0 10110 101 |  | 0 1110 1010 |  |
| 1 00111 110 |  | 1 0000 0001 |  |
| 0 00000 101 |  | 0 0000 0001 |  |
| 1 11011 000 |  | 1 1110 1111 |  |
| 0 11000 100 |  | 0 1110 1111 |  |

Format A的bias是15；Format B的bias是7.

经验：先近似数量级，再调整frac部分。

## 2.88

参考p147.

本题分析类型之间的值域关系。我们要记住：float point 数值是稀疏序列，不是连续分布的。

A. false。

参考2.84，float能够准确表示的最大的奇数是：。

反例是一个不用float完整表示的int数：.

B. false。假设

会overflow，值为。

不会overflow, 值为。

C. 等价于：

true。

不等式两边的加法运算不会导致overflow。参考2.84，Double能够完整表示的最大的奇数是。

D. 等价于：

false。

本小题要仔细分析rounding和double值的分布规律。

int类型为32位，值域是

参考2.84，Double能够完整表示的最大的奇数是。

X是奇数，y是奇数，z是奇数

~~x\*y => 大于的 => roun~~

如果dx与dy乘积是大于的奇数，那么dx\*dy会被rounding。Rounding之后再乘dz。

如果dz是偶数，且dz\*dx

根据这一点，我们构造反例：

## 2.89

本题要点是single-precision format分界点的特征值。

|  |  |  |
| --- | --- | --- |
| s | e | f |
| 31 | 30-23 | 22-0 |

|  |  |  |
| --- | --- | --- |
| Description | Bit pattern | Value |
| zero | 0 00..00 00..00 | 0 |
| Smallest Denorm. | 0 00..00 00..01 |  |
| Largest Denorm. | 0 00..00 11..11 |  |
| Smallest Norm. | 0 00..01 00..00 |  |
| Largest Norm. | 0 11..10 11..11 |  |
| Positive Infinity | 0 11..11 00..00 |  |

## 2.90

A. 0x40490FDB

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 1 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 |
| F | 1 | 1 | 1 | 1 |
| D | 1 | 1 | 0 | 1 |
| B | 1 | 0 | 1 | 1 |

e=1, 表示M部分乘以2

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 1. | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 |

B. 参考2.82(a)知道，22/7的二进制表示为11.001001…(y=001) 。

C. 略

## 2.91

修改sign bit.

## 2.92

修改sign bit.

## 2.93

要区分值的类型（zero, denorm, norm, infinity, nan）

sign, exp, frac

zero -> zero

denorm -> frac >> 1

norm ->

exp == 1 : exp = 0, frac>>1

exp != 1 : exp – 1

infinity -> infinity

NaN -> NaN

## 2.94

要区分值的类型（zero, denorm, norm, infinity, nan）

我们用不同方式处理不同范围的值。特别注意边界情况。

当f=0时，返回0.

当f是denorm.时, 如果f的首位是0，那么只需f<<1即可。如果f的首位是1，那么2\*f是norm.

Implied bit，根据smooth transition。（2\*f是norm，也可以用f<<1，怎么说明？）

2\*f

sign, exp, frac

zero -> zero

denorm -> frac << 1

norm -> exp = exp + 1

infinity -> infinity

NaN -> NaN

## 2.95

从bit level观察int数，转换成形式，其中涉及rounding。

要点是理解floating-point format各部分的含义。

if i == 0 return 0

else

sign = i >>31

if (sign == 1) i =-i

leftmost\_one => k

exp = k

## 2.96

从int类型值域是。

从float角度观察，

1. 时，V<1, round to zero，int类型值为0
2. 时， ，M非整数部被舍去后，int类型值为
3. 时，
4. ，，
5. 为overflow或NaN