Chapter 2.

Practice 2.1

0011 1001 1010 0111 1111 1000

0xC97B

1101 0101 1110 0100 1100

0x26E7B5

Practice 2.2

524288 0x80000

0x4000

16 65536

131072 0x20000

0x20

7 128

Practice 2.3

1010 0111 0xA7

0011 1110 0x3E

1011 1100 0xBC

0x37

0x88

0xF3

0101 0010

1010 1100

1110 0111

Practice 2.4

0x5044

0x4FFC

0x507C

0xAE

Practice 2.5

21 87

21 43 87 65

21 43 65 87 65 43

Practice 2.6

A.

0000 0000 0011 0101 1001 0001 0100 0001

0100 1010 0101 0110 0100 0101 0000 0100

B.

0000 0000 0011 0101 1001 0001 0100 0001

0100 1010 0101 0110 0100 0101 0000 0100

Practice 2.7

这里跟之前的练习的区别主要在于书写习惯造成理解的问题，对于数值0x1234而言，其最低有效位在在右侧，对于字符串”abcdef”，其最低有效位在最左侧。

61 62 63 64 65 66

Practice 2.8

1. a [01101001] b [01010101]
2. ~a [10010110] ~b [10101010]
3. a & b [01000001]
4. a | b [01111101]
5. a ^ b [00111100]

Practice 2.9

1. 颜色的补

黑 - 白 红 - 蓝绿

蓝 - 黄 红紫 - 绿

绿 - 红紫 黄 - 蓝

蓝绿 - 红 白 - 黑

1. 颜色之间的布尔运算

蓝 | 绿 = 蓝绿

黄 & 蓝绿 = 绿

红 ^ 红紫 = 蓝

Practice 2.10

第一步：a a ^ b

第二步：a ^ a ^ b = b a ^ b

第三步：b b ^ a ^ b = b ^ b ^ a = a （利用了布尔环的交换律）

Practice 2.11

1. 最后一次循环，first = last = k
2. 此时，inplace\_swap交换的是同一个位置的数组元素，而且针对^运算而言，每个布尔环元素的加法逆元是其自身，所以a[first] ^ a[last] = a[first] ^ a[first] = 0。
3. void reverse\_array(int a[], int cnt)

{

int first, last;

for (first = 0, last = cnt-1;

first <= cnt / 2 - 1;

first++, last--)

inplace\_swap(&a[first], &a[last]);

}

Practice 2.12

1. x & 0xFF
2. 要注意的一个地方是题目要求对任意字长w >= 8的情况都能支持，所以给出的mask是不能写死最高有效位为0xFF的，只能通过高位自动补零，然后取补来实现。

所以，除了最低有效位之外，其他位取补，~(x & (~0xFF)) | (x & 0xFF)

1. x | 0xFF

Practice 2.13

1. int result = bis(x, m)
2. 假设x的位模式是abc，m的位模式是xyz，针对bic运算的第一个操作数的位模式，如果是1的位，无论m的位模式如何，其结果是符合异或运算的规则的；另外如果第二个操作数的位模式中为0的位，无论x在该位的位模式如何，其结果也是符合异或运算的规则。所以如果将同一个数寄放在bic的第一个操作数就可以得到运算结果在这个数位模式为0的那部分结果，放在第二个操作数得到位模式为1的那部分结果，然后两者用A的答案，或起来就可以了。所以结果是

int result = bis(bic(x, m), bic(m, x))

Practice 2.14

x = 0x66; y = 0x39

x & y = 0010 0000 = 0x20 x && y = 1

x | y = 0111 1111 = 0x7f x || y = 1

~x | ~y = 1101 1111 = 0xdf !x || !y = 0

x & !y = 0 x && !y = 0

Practice 2.15

!(x ^ y)

Practice 2.16

x

1100 0011

0111 0101

1000 0111

0110 0110

x << 3

0001 1000 0x18

1010 1000 0xa8

0011 1000 0x38

0011 0000 0x30

x >> 2 （逻辑）

0011 0000 0x30

0001 1101 0x1d

0010 0001 0x21

0001 1001 0x19

x >> 2 （算术）

1111 0000 0xf0

0001 1101 0x1d

1110 0001 0xe1

0001 1001 0x19

Practice 2.17

0x0 0000 0 0

0x5 0101 2^2 + 2^0 = 5 2^2 + 2^0 = 5

0x8 1000 2^3 = 8 -2^3 = -8

0xd 1101 2^3 + 2^2 + 2^0 = 13 -2^3 + 2^2 + 2^0 = -3

0xf 1111 2^3 + 2^2 + 2^1 + 2^0 = 15 -2^3 + 2^2 + 2^1 + 2^0 = -1

Practice 2.18

其实就是将指令里面给出的十六进制常数（用32位补码表示）转成十进制数

对于负数来说，从二进制位模式转换为十进制数，绝对值可以从反码入手，取反加1，最后补上负号即可。

要理解这个问题可以直观地从补码形成的代数系统入手，正数x，位模式是p`，其相反数-x，位模式是q`，因为x + (-x)为0，所以位模式p`和q`相加也要为0，针对二进制加法的理解，我们只要将p`取反的到q``，p` + q``即为0xffff ffff，此时只要再加一，就可以在低32位（其他位数同理）为0了。具体证明可以从补码和反码的定义入手，两者的差异在于最高位的位权不一样（具体定义可参考CSAPP 2nd P43）。

1. 0x1b8 440
2. 0x14 20
3. 0xfffffe58 -424
4. 0xfffffe74 -396
5. 0x44 68
6. 0xfffffec8 -312
7. 0x10 16
8. 0xc 12
9. 0xfffffeec -276
10. 0x20 32

Practice 2.19

根据题目给出的提示，要计算T2U的值，可以通过计算T2B，B2U完成（在此题后，CSAPP给出了两者的直接关系）。

根据一个有符号数，得到其补码表示，可以利用上题的思路，先取绝对值，然后取反加一后，即为其补码表示。（其实根据补码的定义，补码表示的最高位其位权是2^(w - 1)，如果对其位权的解析方式从补码的-1 \* 2^(w - 1)变为2^(w - 1)，其差值位2^(w - 1) - ( -1 \* 2^(w - 1)) = 2^w，所以负数的位模式转为无符号的解析方式的话，其值在有符号数的基础上加上2^w的值即可）

x T2B B2U

-8 1000 8

-3 1101 13

-2 1110 14

-1 1111 15

1. 0000 0

5 0101 5

Practice 2.20

将公式2-6应用到Practice 2.19，如果x是正数，则保持不变，否则，加上2^4 = 16，即为相同位模式下，有符号负数对应的无符号整数值。

Practice 2.21

-2147483647-1 == 2147483648U unsigned true

-2147483647-1 < 2147483647 signed true

-2147483647-1U < 2147483647 unsigned false

（MSVC编译器下会触发C4308，左操作数因为1U的存在，需要把-2147483647转为无符号数）

-2147483647-1 < -2147483647 signed true

-2147483647-1U < -2147483647 unsigned true

**Practice 2.22**

等式2-3是B2T的定义。

B2T(1011) = -1\*2^3 + 1\*2^1 + 1\*2^0 = -8 + 2 + 1 = -5

B2T(11011) = -1\*2^4 + 1\*2^3 + 1\*2^1 + 1\*2^0 = -16 + 8 + 2 + 1 = -5

B2T(111011) = -1\*2^5 + 1\*2^4 + 1\*2^3 + 1\*2^1 + 1\*2^0 = -32 + 16 + 8 + 2 + 1 = -5

实际上负权的值扩展之后，旧的负权变成正权后，效果相互抵消

-8 = -16 + 8 = -32 + 16 + 8

**Practice 2.23**

1. Fun1 Fun2

0x0000 0076 0x0000 0076 0x0000 0076

0x8765 4321 0x0000 0021 0x0000 0021

0x0000 00c9 0x0000 00c9 0xffff ffc9

0xedcb a987 0x0000 0087 0xffff ff87

1. Fun1的计算是重置所有非最低有效字节的位为0. Fun2的计算是将最低有效字节的最高位，扩展至高位。

**Practice 2.24**

无符号

1. 0

2 2

9 1

11 3

15 7

补码

1. 0

2 2

-7 1

-5 3

-1 -1

需要注意-1截断后，因为w=3，最高位为1，所以-1截断后的位模式在w=3下仍为负数，其负权是-1\*2^2 + 2^1 + 2^0 = -1。

**Practice 2.25**

当length(unsigned)为0，length – 1 = UINT\_MAX，所以i <= length – 1恒定为true，导致访问到非法的内存地址。

for (i = 0; i < length; i++)

**Practice 2.26**

1. 当strlen(s) < strlen(t)时，结果不正确。
2. Strlen(s) – strlen(t)，unsigned负溢出，成为正数，所以此时strlen(s) – strlen(t) > 0返回true。
3. Strlen(s) > strlen(t)。

**Practice 2.27**

uadd\_ok的编写

x和y作为字长w的无符号整数有 x < 2^w，y < 2^w，假设x + y溢出，根据无符号数加法的定义，sum = x + y - 2^w，因为y < 2^w，所以 y - 2^w < 0，所以此时

sum = x + y - 2^w < x。交换x和y的位置同理，所以只要两个无符号浮点数之和，小于其中任意一个操作数，即出现溢出。

**Practice 2.28**

1. 0 0 0
2. 5 11 B
3. 8 8 8

D 13 3 3

F 15 1 1

**Practice 2.29**

x y x + y x +{t,5} y 情况

-12 -15 -27 -5 1

-8 -8 -16 -16 2

-9 8 -1 -1 2

2 5 7 7 3

12 4 16 -16 4

**Practice 2.30**

bool tadd\_ok(int x, int y)

{

Int sum = x + y;

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

return true;

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

return true;

return false;

}

**Practice 2.31**

固定位数下的模数加法会形成一个阿贝尔群，符合结合律和交换律。所以无论x + y是否溢出，x + y – y 恒等于x。

假设x + y正溢出，x +{t,w} y = x + y – 2^w = sum。Sum – x = x + y – 2^w – x = y – 2^w，在w位补码的条件下，y – 2^w = y。同理可证明 sum – y 恒等于x。

我在验证的时候为了方便，假设了w=8的情况，在实现了以下函数后，传入x = -128，y=-128，期望得到返回值为true，但返回值却是false

Bool tadd\_ok\_int8(int8\_t x, int8\_t y)

{

Int8\_t sum = x + y;

Return (sum – x == y) && (sum – y == x);

}

问题在哪里？

关键在于整型提升

<https://zh.wikipedia.org/wiki/%E6%95%B4%E5%9E%8B%E6%8F%90%E5%8D%87>

所以sum – x == y在实际计算时，左运算符是会被提升至int（如果精度不够会提升至unsigned int），所以sum – x的计算结果是128（int），所以y此时也会因隐式类型转换，转换为int与左操作数进行比较，128 != -128，结果返回false。

避免这个问题，可以将代码修改为

Int8\_t tmp0 = sum – x;

Int8\_t tmp1 = sum – y;

使用tmp0和tmp1进行比较即可。虽然在计算tmp0和tmp1时仍会出现整型提升，但是在赋值时，类型转换会截断高位数据，得到期望的结果。

**Practice 2.32**

关键是补码表示下，正负区间的不对称性。

y = TMin时，tsub\_ok就会给出错误的结果。

当 x >= 0，x – y会正溢出，tadd\_ok(x, -y)中，当y = TMin，-y = TMin，x + TMin没有溢出，tsub\_ok返回值错误。

当x < 0，x – y不会溢出，tadd\_ok(x, -y)中，当y = TMin，-y = TMin，x + TMin负溢出，tsub\_ok返回值错误。

正确的写法应该单独处理y = TMin的情况，**具体参考Homework 2.74。**

**Practice 2.33**

0 0 0 0

5 5 -5 B

8 8 -8 8

D -3 3 3

F -1 1 1

与Practice 2.28对比，补码与无符号数的非产生的位模式是一样的。（无符号数的加法逆元是2^w – x，因为x + 2^w – x = 2^w -> 2^w % 2^w = 0）。所以对待补码也好，无符号数也好，某个位模式下的非，其位模式就是取反加一，再按照补码或者无符号数换算成数值。（或者说是遵循补码位模式的计算方式计算即可）