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