第三章

3.1 无符号的话5ED4和074A表示的都是正数，结果为 5730

3.2 有符号的话由于 5 = 0101， 0 = 0000， 所以5ED4和074A的最高位都是0，表示的都是正数，相减的结果也为正数，用符号-数值的方式存储还是 5730

3.14 硬件：由图可知，1位需要3个步骤(加法一次，两个数移位一次，判断一次)，所以总共有24个步骤,所需时间为 24 ×4 = 96

软件： 1位需要五个步骤(判断要加什么一次，加法一次，两个数移位2次，判断一次)，所以总共40个步骤，所需时间为40 × 4 = 160

3.20 由于最高位为0，所以补码表示和无符号表示的值是相同的：0x0C000000 = (1<<24)*12 = 201326592

3.21 查表得opcode = 000011, 表示jar类型，后面的26位表示地址， 前面4位由PC的前四位决定，若PC前四位为00， 表示的是 jar 0x00000000

3.22 0x0C000000 = 0 00011000 00000000000000000000000

符号位为0，指数位为24， 24 - 127 = -103

表示的数字为 1.0×2^{-103}

3.23 $63 = 111111_2$ $0.25 = 0.01_2$ $63.25 = 111111.01 = 1.1111101 \times 2^5$

符号位为0， 指数位为 127 + 5 = 132， 尾数去除第一位为1111101

用IEEE754单精度表示： 0 10000100 11111010000000000000000

3.27 $0.15625 = 0.00101_2 = 1.01 \times 2^{-3}$

符号位为1，指数位为 -3 + 16 = 13， 尾数位隐含0后为01

表示方式为 1 01101 0100000000

与IEEE754单精度相比，表示数的范围减少，若全是0和全是1保留的话，IEEE75能表示出2^{-126}到2^{137}，这种方式只能表示出2^{-15}到2^{14}， 范围还不足int，且尾数只有10位, 与原来的23位相比，精度大大减小

3.29

$2.6125 \times 10^1 = 11010.001_2 = 1.1010001 \times 2^4$

$0.4150390625 = 0.0110101001 = 1.10101001 \times 2^{-2}$

①将最小的指数的数的有效数右移， $1.10101001 \times 2^{-2} = 0.00000110101001 \times 2^4$

②将有效位相加 $1.10101000101001 \times 2^4$

③将和规格化，并检查上溢和下溢，$1.10101000101001 \times 2^4$已经规格化

④尾数是1010100010， $G = 1$，$R = 0$，$S = 1$，进入一位，不用再规格化

⑤得到$1.1010100011 \times 2^4 = 26.546875$

3.30

$-8.0546875 = -1.0000000111 \times 2^3$

$-1.79931640625 \times 10_{-1} = 1.0111000010 \times 2^{-3}$

①将不带偏阶的指数相加， 3 + (-3) = 0

②显示隐藏位，将有效位通过加法和移位来相乘，得到结果为1.01110011000001001110

③检查得到的积已经规格化，没有上溢或者下溢， 尾数为其中尾数为
011100110，$G = 0$，$R = 0$，$S = 1$

④由$001 < 100$， 不需要舍入 1.011100110×2^0

⑤初始时符号相同，所以积为正， 1.011100110×2^0

用3.27得到的答案为$1.011100110 \times 2^0 = 1.44921875$, 用计算器得到的结果为
1.4492931365966796875

经过对比得到小数点后前四位是正确的，后面出现误差，可能是3.27方法的浮点数尾数位太少，十与二进制数转换的误差导致