天津大学

计算机系统基础上机实验报告

实验题目 2：整数与浮点数 int and floats

学院名称 智能与计算学部

专　　业 计算机科学与技术（拔尖班）

学生姓名 牛天溳

学　　号 3024244288

年　　级 2024 级

班　　级 拔尖 1 班

时　　间 2024 年 4 月 28 日

实验 2：整数与浮点数

int and floats

1. 实验目的

熟悉整型和浮点型数据的编码方式，熟悉 C/C++中的位操作运算。

2. 实验内容

请按照要求补全 bits.c 中的函数，并进行验证。包括以下 7 个函数：理解

No	函数定义	说明

1	`int conditional(int x, int y, int z)`	`/*`
		` * conditional - same as x ? y : z`
		` *   Example: conditional(2,4,5) = 4`
		` *   Legal ops: ! ~ & ^ | + << >>`
		` *   Max ops: 16`
		` *   Rating: 3`
		` */`

2	`int isNonNegative(int x)`	`/* isNonNegative - return 1 if x >= 0,`
		` * return 0 otherwise`
		` * Example: isNonNegative(-1) = 0.`
		` *          isNonNegative(0) = 1.`
		` *   Legal ops: ! ~ & ^ | + << >>`
		` *   Max ops: 6`
		` *   Rating: 3`
		` */`

3	`int isGreater(int x, int y)`	`/* isGreater - if x > y  then return 1,`
		` * else return 0`
		` *   Example: isGreater(4,5) = 0,`
		` *            isGreater(5,4) = 1`
		` *   Legal ops: ! ~ & ^ | + << >>`
		` *   Max ops: 24`
		` *   Rating: 3`
		` */`

4	`int absVal(int x)`	`/*`
		` * absVal - absolute value of x`
		` *   Example: absVal(-1) = 1.`
		` *   You may assume -TMax <= x <= TMax`
		` *   Legal ops: ! ~ & ^ | + << >>`
		` *   Max ops: 10`

No	函数定义	说明
		* Rating: 4 */
5	int isPower2(int x)	/*isPower2 - returns 1 if x is a power of 2, * and 0 otherwise * Examples: isPower2(5) = 0, * isPower2(8) = 1, * isPower2(0) = 0 * Note that no negative number is a power of 2. * Legal ops: ! ~ & ^ | + << >> * Max ops: 20 * Rating: 4 */
6	unsigned float_neg(unsigned uf)	/* * float_neg - Return bit-level equivalent * of expression -f for * floating point argument f. * Both the argument and result are passed * as unsigned int's, but they are to be * interpreted as the bit-level * representations of single-precision * floating point values. * When argument is NaN, return argument. * Legal ops: Any integer/unsigned * operations incl. ||, &&. also if, while * Max ops: 10 * Rating: 2 */
7	unsigned float_i2f(int x)	/* float_i2f - Return bit-level equivalent * of expression (float) x * Result is returned as unsigned int, but * it is to be interpreted as the bit-level * representation of a * single-precision floating point values. * Legal ops: Any integer/unsigned * operations incl. ||, &&. also if, while * Max ops: 30 * Rating: 4 */

3. 实验要求

1）在 Unbuntu18.04LTS 操作系统下，按照实验指导说明书，使用 gcc 工具集编译程序和测试

2）代码符合所给框架代码的规范（详见 bits.c 的开始位置注释内容）

3）需提交：源代码 bits.c、电子版实验报告全文。

4）本实验相关要求：注意：除非函数又特殊说明，违背以下原则均视为程序不正确！！

程序内允许使用：	程序内禁止以下行为：
a. 运算符： ! ~ & ^ | + << >> b. 范围在 0 - 255 之间的常数 c. 局部变量	a. 声明和使用全局变量 b. 声明和使用定义宏 c. 声明和调用其他的函数 d. 类型的强制转换 e. 使用许可范围之外的运算符 f. 使用控制跳转语句：if else switch do while for

4. 实验结果

```
int conditional(int x, int y, int z) {
  int mask;
  mask = ~(!(!x)) + 1; // 若x非0，则mask为 0xFFFFFFFF；若x为0则mask为 0x00000000
  return ((mask & y) | (~mask & z)); // 若mask为全 1，结果为y；若mask为全0，结果为z，实现x ? y : z 的效果
}
```

对于函数 conditional，首先通过 !(!x) 将任意整数 x 转换为布尔值 0 或 1：当 x 为非 0 时，该表达式为 1；当 x 为 0 时，表达式为 0。接着通过 ~num+1 构造一个掩码 mask：当 x ≠ 0 时，mask 为全 1（0xFFFFFFFF）；当 x = 0 时，mask 为全 0（0x00000000）。

然后利用该掩码在 y 和 z 之间进行选择。具体来说，mask & y 在 mask 为全 1 时保留 y，~mask & z 在 mask 为全 0 时保留 z，二者相加即得到最终结果。

最终，函数在不使用条件判断语句的前提下，实现了与 x ? y : z 等价的功能。

```
int isNonNegative(int x) {
  return (x >> 31) + 1;
  // 右移31位,取符号位
  // 如果x为负数,则结果为-1,加1后为0
  // 如果x为非负数,则结果为0,加1后为1
}
```

对于函数 isNonNegative，首先通过 x >> 31 右移 31 位，提取出 x 的符号位。当 x 为负数时，符号位为 1，结果为 -1；当 x 为非负数时，符号位为 0，结果为 0。接着将符号位的结果加 1，完成以下逻辑：当 x 为负数时，加 1 后结果为 0；当 x 为非负数时，加 1 后结果为 1。最终通过返回值判断 x 是否为非负数，x >= 0 时返回 1，x < 0 时返回 0。

```c
int isGreater(int x, int y) {
  int sx, sy, diff, ds, sign_diff;
  sx = (x >> 31) & 1; // 提取x的符号位
  sy = (y >> 31) & 1; // 提取y的符号位
  diff = x + (~y + 1); //计算x - y（用补码形式）
  ds = (diff >> 31) & 1; //提取差值的符号位
  sign_diff = sx ^ sy; //判断x和y是否异号
  return (sign_diff & !sx) | (!sign_diff & !ds & !!diff);
  //情况1：若异号且x为正（sx = 0），则x > y，返回 1
  //情况2：若同号且x - y > 0（即ds = 0且diff ≠ 0），返回 1
  //否则返回 0
}
```

对于函数 isGreater，首先通过 (x >> 31) & 1 和 (y >> 31) & 1 提取 x 和 y 的符号位 sx 和 sy，用以判断两数的符号是否相同。
然后通过 x + (~y + 1) 计算 x - y，并将结果存入 diff。接着提取 diff 的符号位 ds，判断 x - y 是否为负数。

接下来通过 sx ^ sy 计算 x 和 y 是否异号：如果异号（sign_diff = 1），则 x 为正（sx = 0）时，返回 1；如果同号（sign_diff = 0），则进一步判断 x - y > 0（即 ds = 0 且 diff ≠ 0）时，返回 1。最终根据这些条件判断，返回 1 或 0，实现 x > y 的功能。

```c
int absVal(int x) {
  int mask, neg_x, abs_x;
  mask = x >> 31; // 提取符号位，x为正则mask为 0，x为负则mask为-1（0xFFFFFFFF）
  neg_x = x + mask; // 若x为负，相当于x - 1
  abs_x = neg_x ^ mask; // 若x为负，相当于-(x)，否则保持不变
  return abs_x;
}
```

对于函数 absVal，我们先通过算术右移 31 位 x >> 31 提取出 x 的符号位，赋值给 mask：当 x 为非负数时 mask 为 0，若 x 为负数则 mask 为 -1（全 1）。接着将原始值 x 与 mask 相加得到 neg_x：对于负数，这相当于执行 x - 1，而对非负数则保持不变。最后，通过对 neg_x 与 mask 执行异或操作 neg_x ^ mask，负数会被取反加一（即得到 -x），非负数则依旧保持原值。整个过程无需任何条件分支，即可实现 x >= 0 ? x : -x 的绝对值计算。

```c
int isPower2(int x) {
  int sign, is_zero, mask;
  sign = x >> 31;
  is_zero = !x;
  mask = x & (x + ~0);  // x & (x - 1)
  return (!sign & !is_zero & !mask);
}
```

对于函数 isPower2，我们首先通过算术右移 31 位 x >> 31 得到 sign，用以判断 x 是否为负数；然后通过逻辑非运算 !x 判断 x 是否为零，因为零不被视为 2 的幂。接下来，利用位运算表达式 x & (x + ~0)（即 x & (x - 1)）计算出 mask，该操作在 x 为 2 的幂时会得到 0。最后，将这三个条件组合起来：当 sign 为 0（表示非负）、!x 为 0（非零）、且 mask 为 0 时，函数返回 1，表明 x 是 2 的幂；否则返回 0，表明 x 不是 2 的幂。通过这一系列纯

粹的位运算，我们高效地实现了对整数是否为 2 的幂的判断。

```
unsigned float_neg(unsigned uf) {
  unsigned sign, exp, frac;
  sign = uf & 0x80000000; // 提取符号位
  exp = uf & 0x7F800000; // 提取指数部分
  frac = uf & 0x007FFFFF; //提取尾数部分
  if (exp == 0x7F800000 && frac != 0) {
    return uf; // NaN
  }
  return sign ^ 0x80000000 | exp | frac; // 翻转符号位返回
}
```

对于函数 float_neg，首先通过 uf & 0x80000000 提取出浮点数的符号位，存入 sign。然后，使用 uf & 0x7F800000 提取浮点数的指数部分，并存入 exp；接着通过 uf & 0x007FFFFF 提取尾数部分并存入 frac。接下来，我们检查 exp 是否为全 1（0x7F800000），并且 frac 不为 0，表示该浮点数为 NaN（不是一个数字），此时直接返回原值。

最后，如果不是 NaN，我们通过 sign ^ 0x80000000 翻转符号位，并保留原始的指数和尾数部分，最终返回修改后的浮点数值，从而实现对浮点数取反的操作。

```
unsigned float_i2f(int x) {
  int sign, highbit, exp, frac, flag;
  unsigned temp, result;
  if(!x) // 特判x=0的情况
    return x;
  sign = (x >> 31) & 1;
  if(sign)
    x =~x + 1; // 获得符号位，并将x变为正值。
   highbit = 0;
  temp = x;
  while(!(temp & 0x80000000))
  {
      temp <<= 1;
      highbit++;
  } //获得x的最高有效位，即确定frac的位数。
  temp = temp << 1;
  exp = 127 + 31 - highbit;  //计算出exp，和frac位数。
  frac = 31 - highbit;
  flag = 0; //进行向偶数舍入
  if((temp & 0x1ff) > 0x100)
  flag = 1; //出现在规定位置后大于0b100的情况就进1.
  if((temp & 0x3ff) == 0x300)
  flag = 1; //出现最后一位为1，且下一位为1的情况也进1(向偶取整)
  result = (sign << 31) + (exp << 23) + (temp >> 9) + flag; //计算最终结果
  return result;
}
```

对于函数 float_i2f，我们首先对输入 x 进行零值特判：若 x 为 0，直接返回 0。然后通过 (x >> 31) & 1 提取出符号位 sign，如果 sign 为 1（表示负数），则将 x 取反加一转换为正值，以便后续处理。接着初始化 highbit 为 0，并将 x 赋值给 temp，通过不断左移 temp 直到最高位变为 1 来确定原始整

数中最高有效位的位置，并统计左移次数 highbit。随后将 temp 再左移一位，为计算尾数做好准备；此时可得尾数应包含的有效位数 frac = 31 - highbit，以及指数值 exp = 127 + 31 - highbit。为了实现 IEEE 754 中的“向偶数舍入”策略，我们首先检查 temp 低 9 位（temp & 0x1ff）：若大于 0x100 或者等于 0x300，则设置进位标志 flag = 1。最后，通过将符号位左移 31、指数左移 23，再加上右移 9 位并加上 flag 后的 temp，组合成最终的 32 位浮点表示 result 并返回。这样，我们在不使用任何浮点运算的前提下，完整地将有符号整数转换为符合 IEEE 754 标准的单精度浮点数。

5. 实验总结及心得体会

通过本次实验，我对整数和浮点数在二进制机器码下的表示方式有了更加清晰和深入的理解，尤其对浮点数的 IEEE 754 标准格式有了直观认识和具体实践。在编写各个函数的过程中，我不仅复习了课本上的理论知识，也通过实际编码加深了对二进制补码、符号扩展、逻辑右移等概念的理解。

我深刻体会到，构造正确且高效的掩码是一种非常关键的技巧，能够帮助我们更精准地定位和处理特定位信息，这在底层开发中非常常见且实用。此外，在实现如 isPower2、bitCount 等函数时，我的位运算能力和模式识别思维得到了明显的提升，对如何在受限操作符下实现功能有了更深刻的体会。

当然，在实验过程中我也遇到了一些困难和挑战，例如如何处理负数除法向零舍入、如何在保证合法操作符的前提下实现复杂功能等。但通过查阅资料、结合已有思路不断调试和验证，我逐步解决了这些问题，提升了独立思考和解决问题的能力。

此次实验不仅加深了我对计算机系统底层机制的理解，也进一步锻炼了我写出结构清晰、逻辑严密代码的能力，为后续学习其他的专业课程，以及参与算法设计、系统开发和学科竞赛等活动打下了扎实的基础。