天津大学

计算机系统基础上机实验报告

实验题目 1：位操作 bit-ops

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 牛天溪

学　　号 3024244288

年　　级 2024 级

班　　级 拔尖 1 班

时　　间 2024 年 4 月 24 日

实验 1：位操作

Bit Operations

1. 实验目的

进一步理解书中第二章《信息的表示和处理》部分的内容，深刻理解整数、浮点数的表示 和运算方法，掌握 GNU GCC 工具集的基本使用方法。

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 6 个函数：理解

No	函数定义	说明
1	int isAsciiDigit(int x)	/* isAsciiDigit - return 1 if * 0x30 <= x <= 0x39 * (ASCII codes for characters '0' to '9') * Example: isAsciiDigit(0x35) = 1. * isAsciiDigit(0x3a) = 0. * isAsciiDigit(0x05) = 0. * Legal ops: ! ~ & ^ | + << >> * Max ops: 15 * Rating: 3 */
2	int anyEvenBit(int x)	/* * anyEvenBit - return 1 if any even-numbered * bit in word set to 1 * Examples: anyEvenBit(0xA) = 0, * anyEvenBit(0xE) = 1 * Legal ops: ! ~ & ^ | + << >> * Max ops: 12 * Rating: 2 */
3	int copyLSB(int x)	/* * copyLSB - set all bits of result to least * significant bit of x * Example: copyLSB(5) = 0xFFFFFFFF, * copyLSB(6) = 0x00000000 * Legal ops: ! ~ & ^ | + << >> * Max ops: 5 * Rating: 2

No	函数定义	说明
		*/
4	int leastBitPos(int x)	/* * leastBitPos - return a mask that marks * the position of the * least significant 1 bit. If x == 0, * return 0 * Example: leastBitPos(96) = 0x20 * Legal ops: ! ~ & ^ | + << >> * Max ops: 6 * Rating: 2 */
5	int divpwr2(int x,int n)	/* * divpwr2 - Compute x/(2^n), for 0 <= n <= 30 * Round toward zero * Examples: divpwr2(15,1) = 7, * divpwr2(-33,4) = -2 * Legal ops: ! ~ & ^ | + << >> * Max ops: 15 * Rating: 2 */
6	int bitCount(int x)	/* * bitCount - returns count of number of * 1's in word * Examples: bitCount(5) = 2, bitCount(7) = 3 * Legal ops: ! ~ & ^ | + << >> * Max ops: 40 * 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：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	a. 声明和使用全局变量 b. 声明和使用定义宏

程序内允许使用：	程序内禁止以下行为：
c. 局部变量	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 isAsciiDigit(int x) {
  return !((x + (~0x30 + 1)) >> 31) & !((0x39 + (~x + 1)) >> 31);
  // 计算 (x + (~0x30 + 1))，即 x - 0x30，检查结果是否为非负数
  // 如果 x >= 0x30 ('0')，则 (x - 0x30) 为非负数，右移后结果为 0，取非得到 1
  // 否则，结果为负数，右移后结果为 -1，取非得到 0
  // 通过右移 31 位来检查是否在 '0' 到 '9' 的范围内
}
```

对于函数 isAsciiDigit，首先我计算 x - 0x30 和 0x39 - x 的值（对于负数，其值等于相反数的补码加 1）。然后，对于 32 位整数，右移 31 位可以提取符号位(当为 -1 时表示负数,0 时表示正数)。取 ！ 的原因是,为了将 0x39 + (~x + 1) 为 -1 时，通过取反变成 1。最后，通过按位与 & 运算得到最终的结果。

```
int anyEvenBit(int x) {
  int num = 0x55555555;// 0x55555555在二进制表示下即为奇数为0，偶数位为1的二进制数
  return !!(x & num);// 将x与num按位与，若x有存在偶数位不为0的情况，x&num的值非0，再通过两次取非将其转换为布尔值为1;反之为0
}
```

对于函数 anyEvenBit，首先我构造了掩码 num = 0x55555555，它在二进制中所有偶数位为 1，奇数位为 0。然后，我 x & num 提取 x 的偶数位——如果 x 在任意偶数位上有 1，则结果非 0；否则结果为 0。取 !! 的原因是为了把非零值转换成 1，把 0 保持为 0，从而得到标准的布尔返回值。最后直接返回这个布尔值，表示 x 是否存在偶数位为 1。

```
int copyLSB(int x) {
  return x << 31 >> 31;
  // x << 31 将 x 的最低有效位移到最高位
  // 然后 >> 31 将移到最高位的值右移到最低位，形成一个全为相同值的 32 位数
  // 如果 x 的 LSB 为 1，结果会是 0xFFFFFFFF；如果 LSB 为 0，结果会是 0x00000000
}
```

对于函数 copyLSB,首先我把 x 的最低有效位移到最高位,也就是执行 x << 31；然后我对这个值做算术右移 31 位，利用符号扩展把最高位的那个比特复制到所有位。这样如果原来的最高有效位是 1，就会得到 0xFFFFFFFF；如果原来的 LSB 是 0，就会得到 0x00000000。

```c
int divpwr2(int x, int n) {
  int sign = x >> 31; // 判断符号，准备处理负数
  int fix = (1 << n) + (~0); // 除法偏移，用于负数情况
  x += (sign & fix); // 如果是负的，加上补偿
  return x >> n;
}
```

对于函数 divpwr2，首先提取 x 的符号位，通过右移 31 位得到 sign，用来判断 x 是正数还是负数：当 x 为正数时，sign 为 0；当 x 为负数时，sign 为 -1。

然后，计算了一个偏置值 fix，它等于 2^n - 1，通过 (1 << n) + (~0) 得到。这个偏置用于处理负数除法的情况，因为负数做右移时结果是向下舍入的，需要加上偏置来实现向零舍入。

接着，通过 x += (sign & fix) 来判断 x 是否为负数：如果 x 为负，sign 为 -1，那么就会加上偏置 fix；如果 x 为正，sign 为 0，结果则不变。

最后，做算术右移 x >> n，完成除法操作，相当于计算 x / 2^n，并且通过偏置保证负数的结果向零舍入。

```c
int bitCount(int x) {
  // 用掩码直接写出常量形式
  int m1 = 0x55555555; // 每两位一个 01，用来统计每两位中的 1 的个数
  int m2 = 0x33333333; // 每四位一个 0011
  int m4 = 0x0F0F0F0F; // 每八位一个 00001111
  int m8 = 0x00FF00FF; // 每十六位一个 0000000011111111
  int m16 = 0x0000FFFF; // 用来统计整个 32 位
  x = (x & m1) + ((x >> 1) & m1);     // 每 2 位统计 1 的个数
  x = (x & m2) + ((x >> 2) & m2);     // 每 4 位统计
  x = (x & m4) + ((x >> 4) & m4);     // 每 8 位统计
  x = (x & m8) + ((x >> 8) & m8);     // 每 16 位统计
  x = (x & m16) + ((x >> 16) & m16);  // 全部统计完
  return x;
}
```

对于函数 bitCount，讲实话这个函数真的好难，我也是上网查阅了资料才了解了这种方法。

首先定义了五个掩码 m1 = 0x55555555、m2 = 0x33333333、m4 = 0x0F0F0F0F、m8 = 0x00FF00FF 和 m16 = 0x0000FFFF，它们分别用于统计每 2 位、每 4 位、每 8 位、每 16 位和所有 32 位中 1 的个数。通过这些掩码，可以分步累加每个部分中的 1 的个数。

然后，首先通过 m1 将 x 中每一对 2 位中的 1 的个数加起来，接着使用 m2 来统计每 4 位中的 1 的个数，再用 m4 处理每 8 位，m8 处理每 16 位，直到用 m16 统计整个 32 位中的 1 的个数。每一步都会通过按位与运算和右移操作，将前一步的结果逐步合并，最终将所有 1 的个数汇总到 x 的低 32 位中。

最后，返回 x，它的低 32 位就存储了原始输入中所有 1 的个数。

5. 实验总结及心得体会

通过本次实验，我进一步强化了对位运算和位操作的理解与掌握。相比于普通的算术运算，位运算在执行速度上具有明显优势，能够有效提高程序的运行效

率。掌握基础的位操作技巧，对于提升代码性能、优化程序设计具有重要意义。

在实验过程中，我也遇到了一些困难。除了个别之前有所了解的方法（如 leastBitPos 函数），其他大部分函数都需要投入较多时间进行思考。有些问题难以直接解决，我还查阅了相关资料，参考了书本外的解释和例题，最终终于顺利完成了所有函数的实现。

通过本次实验，我不仅加深了对理论知识的理解，锻炼了细致严谨的编程习惯，也提升了独立分析和解决问题的能力，对自己思维能力的提升也是很大的。相信这些收获将对我今后进一步学习计算机系统知识以及开发程序打下坚实的基础。