

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_____求是学部
专 业_____计算机科学与技术
学生姓名_____石云天
学 号_____3020205015
年 级_____2020 级
班 级_____2 班
时 间_____2022 年 10 月 5 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	<pre>int isAsciiDigit(int x) { int a = (x + (~48 + 1)); int b = (x + (~58 + 1)); int c = (((a >> 31) + 1) & (b >> 31)); return c; }</pre>	<pre>/* 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 */</pre>
2	<pre>int anyEvenBit(int x) { int a = (x >> 8); int b = (x >> 16); int c = (x >> 24); return !!(x a b c) & 0x55); }</pre>	<pre>/* * 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 */</pre>
3	<pre>int copyLSB(int x) { int a = (x << 31 >> 31); return a; }</pre>	<pre>/* * copyLSB - set all bits of result to least * significant bit of x * Example: copyLSB(5) = 0xFFFFFFFF, * copyLSB(6) = 0x00000000 * Legal ops: ! ~ & ^ + << >> * Max ops: 5 */</pre>

No	函数定义	说明
		<pre> * Rating: 2 */ </pre>
4	<pre> int leastBitPos(int x) { int a = (~x + 1); return (a & x); } </pre>	<pre> /* * 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 */ </pre>
5	<pre> int divpwr2(int x, int n) { int a = x >> 31; // 取出符号位 int b = ((~0) + (1 << n)) & a; return ((x + b) >> n); } </pre>	<pre> /* * 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 */ </pre>
6	<pre> int bitCount(int x) { int k = (0x11) (0x11 << 8) (0x11 << 16) (0x11 << 24); // 取出用于对比的整数 0x11111111 int a = x & k; x = x >> 1; a = a + (x & k); x = x >> 1; a = a + (x & k); x = x >> 1; a = a + (x & k); return ((a & 0xf) + ((a >> 4) & 0xf) + ((a >> 8) & 0xf) + ((a >> 12) & 0xf) + ((a >> 16) & 0xf) + ((a >> 20) & 0xf)); } </pre>	<pre> /* * bitCount - returns count of number of * 1's in word * Examples: bitCount(5) = 2, bitCount(7) = 3 * Legal ops: ! ~ & ^ + << >> * Max ops: 40 * Rating: 4 */ </pre>

No	函数定义	说明
	<pre>+ ((a >> 24) & 0xf) + ((a >> 28) & 0xf)); }</pre>	

3. 实验要求

- 1) 在 Ubuntu18.04LTS 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！！**

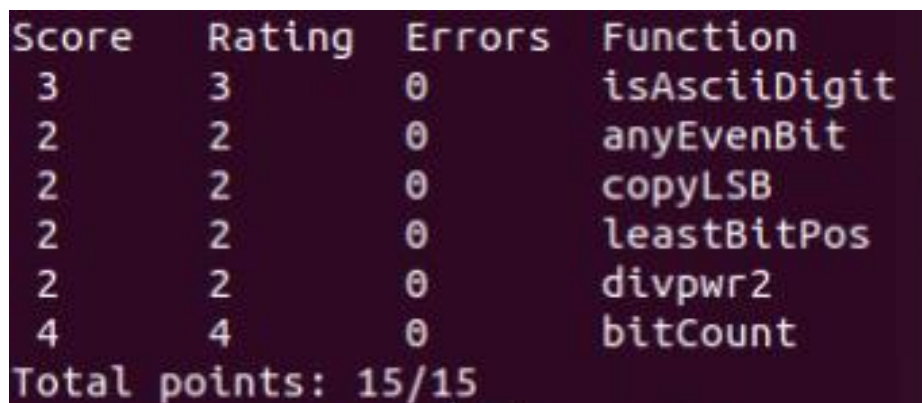
程序内允许使用：	程序内禁止以下行为：
a. 运算符： ! ~ & ^ + << >> b. 范围在 0 - 255 之间的常数 c. 局部变量	a. 声明和使用全局变量 b. 声明和使用定义宏 c. 声明和调用其他的函数 d. 类型的强制转换 e. 使用许可范围之外的运算符 f. 使用控制跳转语句：if else switch do while for

4. 实验结果

（可以包括各函数的代码片段、设计思想、执行各测试工具后的结果截图）

运行结果：

使用 btest 工具测试结果见下图 1：



Score	Rating	Errors	Function
3	3	0	isAsciiDigit
2	2	0	anyEvenBit
2	2	0	copyLSB
2	2	0	leastBitPos
2	2	0	divpwr2
4	4	0	bitCount
Total points: 15/15			

图 1 btest 工具测试结果

使用 driver.pl 工具测试结果见下图 2:

Correctness Results			Perf Results		Puzzle
Points	Rating	Errors	Points	Ops	
3	3	0	2	10	isAsciiDigit
2	2	0	2	9	anyEvenBit
2	2	0	2	2	copyLSB
2	2	0	2	3	leastBitPos
2	2	0	2	7	divpwr2
0	0	0	0	0	from
4	4	0	2	38	bitCount
Score = 27/27 [15/15 Corr + 12/12 Perf] (69 total operators)					

图 2 driver.pl 工具测试结果

(1) int isAsciiDigit(int x);

- 功能：当 $0x30 \leq x \leq 0x39$ 时（即字符 0-9 的 ASCII 码值）返回 1；其他情况下返回 0
- 示例：
isAsciiDigit(0x35) == 1
isAsciiDigit(0x3a) == 0
isAsciiDigit(0x05) == 0
- 难度：3
- 可使用运算符数：15

代码片段：

```
int isAsciiDigit(int x) {  
    int a = (x + (~48 + 1)); //判断ascii码是否大于等于48, 若是, 则符号位为0  
    int b = (x + (~58 + 1)); //判断ascii码是否小于等于57, 若是, 则符号位为1  
    int c = (((a >> 31) + 1) & (b >> 31));  
    return c;  
}
```

设计思想：本题解决思路是让 x 与上下界分别进行比较，若 x 小于等于上界且大于等于下界则其满足条件，其中难点为利用位运算实现比较操作。此处可以利用符号位的特殊性质实现目标， a 为 x 减去下界，若其符号位为 0（可利用右移 31 位取出符号位），则满足条件。 B 位 x 减去上界，若其符号位为 1，则满足条件。最终若两个条件均满足，则返回值为真，反之为假。

(2) int anyEvenBit(int x)

- 功能：当 x 的任意偶数位为 1 时，返回 1；其他情况下返回 0
- 示例：
anyEvenBit(0xA) == 0
anyEvenBit(0xE) == 1
- 难度：2
- 可使用运算符数：12

代码片段:

```
int anyEvenBit(int x) {  
    int a = (x >> 8);  
    int b = (x >> 16);  
    int c = (x >> 24);  
    return !((x | a | b | c) & 0x55); //以8位大小划分为4块, 并将信息全部叠加存放至最后8位  
}
```

设计思想: 本题利用分块的思想, 利用右移运算分别取出 x 的 0-7、8-15、16-23 位, 利用按位或运算将每块的性质叠加到最低 8 位 (若前三块偶数位中存在 1, 则叠加后结果偶数位一定为 1), 将叠加后的结果与 0x55 (最低 8 位为 01010101) 进行按位与运算, 若偶数位存在 1, 则返回值为真, 反之为假。

(3) int copyLSB(int x)

- 功能: 将返回值中的所有位全部设置成 x 中的第 0 位的值
- 示例:
copyLSB(5) == 0xFFFFFFFF
copyLSB(6) == 0x00000000
- 难度: 2
- 可使用运算符数: 5

代码片段:

```
int copyLSB(int x) {  
    int a = (x << 31 >> 31); //先左移31位将第0位值置于最高位, 再利用算数右移性质实现结果  
    return a;  
}
```

设计思想: 本题首先将 x 左移 31 位, 将第 0 位的值移动至最低位, 随后再右移 31 位, 利用算数右移的性质, 根据最低位的值得到结果。

(4) int leastBitPos(int x)

- 功能: 返回一个掩码, 在该掩码中标识了二进制数 x 的最低位编码为 1 的位置
- 示例:
leastBitPos(0x60) == 0x20
- 难度: 2
- 可使用运算符数: 6

代码片段:

```
int leastBitPos(int x) {  
    int a = (~x + 1); //取相反数-x, 随后与x进行按位与便可得到结果  
    return (a & x);  
}
```

设计思想: 本题首先对 x 取相反数, 随后对两者进行按位与运算, 仅有最低位编码为 1 的位置运算后为 1, 其余位均为 0, 直接返回便得到结果 (此处以 $x=0010$

为例进行说明，其相反数为 1110，按位与运算得到结果为 0010，即为所求)。

(5) int divpwr2(int x, int n)

- 功能：计算 $x / 2^n$ ，并将结果取整
- 示例：
divpwr2(15,1) == 7
divpwr2(-33,4) = -2
- 难度：2
- 可使用运算符数：15

代码片段：

```
int divpwr2(int x, int n) {  
    int a = x >> 31; //取出符号位  
    int b = ((~0) + (1 << n)) & a; //设置判据，若x为正数，则直接右移n位  
    //若x为负数，则需加上2的n次方-1后，再右移n位  
    return (x + b) >> n;  
}
```

设计思想：本题首先需要明确 x 除以 2 的 n 次幂，在位操作运算中即为将其对应 2 进制数右移 n 位（正数成立，负数需要先加上 2 的 n 次方减 1 后，再进行右移操作）。在本题中先对 x 右移 31 位取出符号位判断其正负，若 x 为正数，对应判据 b 即为 0；若 x 为负数，对应判据 b 为 2 的 n 次方减 1，随后将 $(x+b)$ 右移 n 位，即可得到结果。

(6) int bitCount(int x)

- 功能：计算二进制数 x 中，对应位值“1”的总位数
- 示例：
bitCount(5) == 2
bitCount(7) == 3
- 难度：4
- 可使用运算符数：40

代码片段：

```
int bitCount(int x) {  
    int k = (0x11 | (0x11 << 8) | (0x11 << 16) | (0x11 << 24)); //取出用于对比的整数0x11111111  
    int a = x & k;  
    x = x >> 1;  
    a = a + (x & k);  
    x = x >> 1;  
    a = a + (x & k);  
    x = x >> 1;  
    a = a + (x & k); //以4位大小划分成8块，对每块中1的位数分别进行计算  
    //（和0x11111111进行按位与并移动1位，重复4次，最后一次不移动），求出8块中的总和，得到结果  
    return ((a & 0xf) + ((a >> 4) & 0xf) + ((a >> 8) & 0xf) + ((a >> 12) & 0xf) + ((a >> 16) & 0xf)  
        + ((a >> 20) & 0xf) + ((a >> 24) & 0xf) + ((a >> 28) & 0xf));  
}
```

设计思想：本题与第(2)题相同，同样采用分块的思想，此处以 4 位为单位，将 x 分为 8 块。随后利用对 0x11 多次左移运算取出用于对比的整数 $k=0x11111111$ ，接下来对每一块分别求出其

中 1 的个数，即用 x 和 k 进行按位与运算，后将 x 右移一位，再进行按位与运算，将所得结果相加，重复 4 次，最后一次按位与运算后不进行右移，将所得结果与 $0xf$ ($0x1111$) 进行按位与运算便可得到最低四位中 1 的个数，接着利用多次右移运算分别求出其他块中 1 的个数，将每块所得结果相加便可得到最终结果。

5. 实验总结及心得体会

(位操作编程总结，实验中遇到的问题及解决方法等)

总结感悟：

通过这次试验，我体会到了位运算的巧妙便捷之处，同时也发现了自己对其的掌握不够不够深入透彻，在做题过程中需要完成某些特定目标时，我不能很快的想出其对应的位运算操作，需要课下不断练习以熟能生巧，还可以多查阅一些资料以开阔自己的思路。在实验过程中，我体会到了分块思想以及位移运算的重要性，对于一些复杂的问题，可以采取分而治之的观点进行看待，利用位移运算的独特优势逐个击破，再统一整合解决问题。同时还可以利用位移运算构造出自己需要的常量，避免无法直接赋值情况的发生。除此之外，在实验过程中，我对程序的机器级表示和处理也有了更深的理解，对数据类型及其转换规则的掌握更加深入，能够不断改进，优化函数性能，实现预期目标与功能所需。