宇佐见函钩 编写

时　　间　　2025 年 4 月 3 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

1~6 的函数代码片段如下：

```
int isAsciiDigit(int x) {
  int a = !(x + (~0x30+1) >> 31);
  int b = !(0x39 + (~x + 1) >> 31);
  return a&b;
}
/*
```

1. isAsciiDigit

功能：判断输入是否是 ASCII 数字（0x30 到 0x39）。

原理：

判断 x >= 0x30：

使用表达式 x + (~0x30 + 1)，其中~0x30 是 0x30 的按位取反，+1 相当于-0x30。

如果 x >= 0x30，则 x - 0x30 的结果是非负数，右移 31 位后结果为 0。

如果 x < 0x30，则 x - 0x30 的结果是负数，右移 31 位后结果为-1。

使用!操作符将结果转换为 0 或 1。

判断 x <= 0x39：

使用表达式 0x39 + (~x + 1)，其中~x + 1 是 x 的负数。

如果 x <= 0x39，则 0x39 - x 的结果是非负数，右移 31 位后结果为 0。

如果 x > 0x39，则 0x39 - x 的结果是负数，右移 31 位后结果为-1。

使用!操作符将结果转换为 0 或 1。

综合结果：

如果两个条件都满足（x >= 0x30 且 x <= 0x39），返回 1，否则返回 0。

```c
int anyEvenBit(int x) {
  int mask = 0x55 | (0x55 << 8);
  mask = mask | (mask << 16);
  return !!(x & mask);
}
/*
```

2. anyEvenBit

功能：检查输入是否有任何偶数位（0、2、4...）被设置为 1。

原理：

构造掩码：

0x55 的二进制是 01010101，表示偶数位为 1。

将 0x55 左移 8 位、16 位，构造出一个 32 位的掩码 0x55555555。

按位与操作：

将输入 x 与掩码 0x55555555 进行按位与操作。

如果结果不为 0，说明有偶数位被设置为 1。

返回结果：

使用!!操作符将结果转换为 0 或 1。

```c
 */
int copyLSB(int x) {

  return (x<<31)>>31;
}
/*
```

3. copyLSB

功能：将输入的最低有效位（LSB）复制到所有位。

原理：

左移 31 位：

将 x 左移 31 位，将最低有效位移到最高位。

右移 31 位：

如果 x 的最低有效位是 1，右移 31 位后结果为-1（全 1）。

如果 x 的最低有效位是 0，右移 31 位后结果为 0（全 0）。

```
 */
int leastBitPos(int x) {
 return (((x+~0)^x)&x);
}
 /*
```

4. leastBitPos
功能：返回输入中最低有效位 1 的位置。
原理：
计算最低有效位 1 的位置：
使用表达式(x + ~0) ^ x，其中~0 是全 1。
x + ~0 相当于 x - 1，将最低有效位 1 变成 0，后面的 0 变成 1。
x ^ (x - 1)会得到一个数，其中只有最低有效位 1 的位置是 1。
按位与操作：
将结果与 x 按位与，得到最低有效位 1 的掩码。

```
 */
int divpwr2(int x, int n) {

  return (x+(x>>31&((1<<n)+~0)))>>n;
}
```

5. divpwr2
功能：计算 x 除以 2^n，结果向零取整。
原理：
处理负数：
如果 x 是负数，右移 n 位会导致结果向下取整。
为了向零取整，先加上偏移量 2^n - 1。
右移操作：
使用右移 n 位来实现除法。

```c
int bitCount(int x) {
  int mask1 = 0x55 | (0x55 << 8); // 01010101 01010101
  int mask1n = mask1 | (mask1 << 16);

  int mask2 = 0x33 | (0x33 << 8); // 00110011 00110011
  int mask2n = mask2 | (mask2 << 16);

  int mask4 = 0x0F | (0x0F << 8); // 00001111 00001111
  int mask4n = mask4 | (mask4 << 16);

  int mask8n = 0xFF | (0xFF << 16); // 00000000 11111111 00000000 11111111

  int mask16n = 0xFF | (0xFF << 8); // 00000000 00000000 11111111 11111111

  x = (x & mask1n) + ((x >> 1) & mask1n);
  x = (x & mask2n) + ((x >> 2) & mask2n);
  x = (x & mask4n) + ((x >> 4) & mask4n);
  x = (x & mask8n) + ((x >> 8) & mask8n);
  x = (x & mask16n) + ((x >> 16) & mask16n);

  return x;
}
```

6. bitCount

功能：计算输入中 1 的位数。

原理：

分治法：

每两位相加：

使用掩码 0x55555555，将每两位的 1 数相加。

每四位相加：

使用掩码 0x33333333，将每四位的 1 数相加。

每八位相加：

使用掩码 0x0F0F0F0F，将每八位的 1 数相加。

每十六位相加：

使用掩码 0x00FF00FF，将每十六位的 1 数相加。

全局相加：

使用掩码 0x0000FFFF，将所有位的 1 数相加，得到总数。

测试工具截图如下所示

```
simpleedu@test-KVM:~/lab1$ ./driver.pl
1. Running './dlc -z' to identify coding rules violations.

2. Running './bddcheck/check.pl -g' to determine correctness score.

3. Running './dlc -Z' to identify operator count violations.

4. Running './bddcheck/check.pl -g -r 2' to determine performance score.

5. Running './dlc -e' to get operator count of each function.

Correctness Results     Perf Results
Points  Rating  Errors  Points  Ops     Puzzle
3       3       0       2       11      isAsciiDigit
2       2       0       2       7       anyEvenBit
0       2       1       0       2       copyLSB
2       2       0       2       4       leastBitPos
0       2       1       0       7       divpwr2
4       4       0       2       36      bitCount

Score = 19/27 [11/15 Corr + 8/12 Perf] (67 total operators)
```

经过多方考究，我目前认为我的 copyLSB 以及 divpw2 的做法是没有问题的，但是测试工具以及测试平台都判错，需要继续深入研究。

5. 实验总结及心得体会

本实验有效帮助我们深刻理解计算机对整数的存储方式，让我们有了熟练操作整数在内存中的形式的能力。

位操作有一定技巧性，需要注意一些技巧，例如正整数的相反数为~x+1，-1=~0，　x*2 的 n 次方==x>>n, x/2 的 n 次方==x<<n。也理解了掩码的含义和作用：通过与掩码进行运算，将一个数据进行形式上的转换。