第二章习题编程

yinxuhao [xuhao_yin@163.com]

December 30, 2022

Contents

位级整数编码规则

在接下来的作业中，我们特意限制了你能使用的编程结构，来帮你更好地理解 C 语言的位级、逻辑和算术运算。在回答这些问题时，你的代码必须遵守以下规则：

- 假设
  - 整数用补码形式表示。
  - 有符号数的右移是算术右移。
- 禁止使用
  - 条件语句 (if 或者?:)、循环、分支语句、函数调用和宏调用。
  - 除法、模运算和乘法。
  - 相对比较运算 (<、>、<= 和 >=)。
- 允许的运算
  - 所有的位级和逻辑运算。
  - 左移和右移，但是位移量只能在 0 和 $w-1$ 之间。
  - 加法和减法。
  - 对 int 和 unsigned 进行强制类型转换。

1 2.61

写一个 C 表达式，在下列描述的条件下产生 1，而在其他情况下得到 0。假设 x 是 int 类型。

限制条件

1. 遵循位级整数编码规则
2. 不能使用==和!=测试。

1.1 x 的任何位都等于 1.

```
unsigned y = -1; int x = (int)y;
```

1.2 x 的任何位都等于 0.

```
int x = 0;
```

1.3 x 的最低有效字节中的位都等于 1.

```
x = x | 17;
```

1.4 x 的最高有效字节中的位都等于 0.

```
x = x & ((unsigned) -1 >> 8);
```

2 2.62

编写一个函数`int_shifts_are_arithmetic()`，在对 int 类型的数使用算术右移的机器上运行时这个函数生成 1，而在其他情况下生成 0. 你的代码应该可以运行在任何字长的机器上。在几种机器上测试你的代码。

```
//
// Created by yinxuhao on 2022/12/29.
//

#include "chapter2.h"

using namespace std;

void get_bit_length(int i, int &a, int &b) {
    auto x = (unsigned long long) i;
    while (x) {
        x >>= 1;
        ++a;
    }
    auto y = (unsigned) i;
    while (y) {
        y >>= 1;
        ++b;
    }
}

bool int_shifts_are_arithmetic() {
    int a = 0;
    int b = 0;
    get_bit_length(-1, a, b);
    unsigned i = -1;
    int x = (int) i;
    int shift = a - b + 4;
//    return x / 2 != x >> 1;
    return x << shift >> shift == i;
}
```

3 2.63

将下面的 C 函数代码补充完整。函数 srl 用算术右移 (由值 xsra 给出) 来完成逻辑右移，后面的其他操作不包括右移或者除法。函数 sra 用逻辑右移 (由值 xsrl 给出) 来完成算术右移，后面的其他操作不包括右移或者除法。可以通过计算`8*sizeof(int)`来确定数据类型 int 中的位数 w。位移量 k 的取值范围为 0∼w-1。

```
//
// Created by yinxuhao on 2022/12/30.
// Exercise 2.63 ***
//
```

```cpp
#include "chapter2.h"

using namespace std;

unsigned srl(unsigned x, int k) {
    /* Perform shift arithmetically */
    unsigned xsra = (int) x >> k;
    /* Begin solve */
    int int_bits = 8 * sizeof(int);
    unsigned xsla = 1 << (int_bits - k);
    unsigned mask = (INT_MAX + xsla) << 1;
    return xsra & mask;
}

unsigned sra(int x, int k) {
    /* Perform shift logically */
    int xsrl = (unsigned) x >> k;
    /* Begin solve */
    int int_bits = 8 * sizeof(int);
    unsigned negative = -1 * ((x | INT_MIN) == x);
    unsigned mask = negative << (int_bits - k);
    return xsrl | mask;
}
```

4 2.64

写出代码实现如下函数：

```cpp
/* Return 1 when any odd bit of x equals 1; 0 otherwise.
   Assume w=32 */
int any_odd_one(unsigned x);
```

```cpp
//
// Created by yinxuhao on 2022/12/30.
// Exercise 2.64 *
//

#include "chapter2.h"

using namespace std;

int any_odd_one(unsigned x) {
    unsigned bit_1 = 1 << 1;
    unsigned bit_3 = 1 << 3;
    unsigned bit_5 = 1 << 5;
    unsigned bit_7 = 1 << 7;
    unsigned bit_9 = 1 << 9;
    unsigned bit_11 = 1 << 11;
    unsigned bit_13 = 1 << 13;
```

```
    unsigned bit_15 = 1 << 15;
    unsigned bit_17 = 1 << 17;
    unsigned bit_19 = 1 << 19;
    unsigned bit_21 = 1 << 21;
    unsigned bit_23 = 1 << 23;
    unsigned bit_25 = 1 << 25;
    unsigned bit_27 = 1 << 27;
    unsigned bit_29 = 1 << 29;
    unsigned bit_31 = 1 << 31;
    unsigned odd = bit_1 | bit_3 | bit_5 | bit_7 | bit_9 |
              bit_11 | bit_13 | bit_15 | bit_17 | bit_19 |
              bit_21 | bit_23 | bit_25 | bit_27 | bit_29 | bit_31;
    return x - (odd & x) == ((odd >> 1) & x);
}
```