Побитовые логические операции

Задание. (Демо-версия теста, ГУ-ВШЭ, 2006)

Десятичное число N = –17 представлено в формате с фиксированной точкой в дополнительном коде. Длина формата – 8 двоичных разрядов. Определите, какому десятичному числу будет соответствовать этот код после замены значения всех бит кода с чётными номерами на инверсное значение. Номера бит отсчитываются справа налево, начиная с нуля. Младший бит – бит с номером 0. Номер 0 – чётный.

Проблема. Представить описание линейного алгоритма, позволяющего выполнить это задание.

Обсуждение хода решения:

1) построить алгоритм “как есть”;
2) построить вспомогательные алгоритмы “атомарных” операций, а затем на их основе построить и сам алгоритм.

Первый способ годится только для тех, кто не ведает о существовании каких-либо “атомарных” операций. Очевидно, что в результате декомпозиции разработанного алгоритма какие-то “атомарные” операции могут появиться. Как увидеть, что они именно те, которые были необходимы?

Второй способ годится, увы, только для тех, кто интуитивно способен осознать, что базовые “атомарные” операции должны быть основаны на применении побитовых логических операторов.

В любом случае необходимы знания булевой алгебры и навыки работы с побитовыми логическими операторами (***not***, ***and***, ***or***, ***xor***). Напомним, что ***not*** – логическое отрицание (НЕ), ***and*** – конъюнкция (И, логическое умножение), ***or*** – дизъюнкция (ИЛИ, логическое сложение), ***xor*** – исключающее ИЛИ ($\oplus$, сложение по модулю 2).

Не исключено, что кому-то может понадобиться и “черновой” вариант решения, который доступен тем, кто когда-то участвовал в каких-либо соревнованиях по программированию или проходил тестирование по информатике.

Известно, что в оперативной памяти ЭВМ неотрицательные целые числа хранятся в прямом коде (ПК), а отрицательные целые числа – в дополнительном коде (ДК). При этом должна быть указана длина двоичного поля, т.е. количество двоичных разрядов.

ДК целого числа => (ОК целого числа) + 1 или ДК целого числа => 0 – ПК целого числа, где ОК – обратный код.

Итак, начнём с “чернового” варианта решения:

1) от десятичной системы счисления поначалу перейдём к восьмеричной:

17_{10} => 21_8 (прямой код неотрицательного целого числа);

-17_{10} => 357_8 (дополнительный код отрицательного целого числа на 8-разрядном двоичном поле);

Примечание. Покажем оба способа получения дополнительного кода. Первый способ опирается на правила целочисленной компьютерной арифметики, где используются операции получения обратного кода и сложения; второй способ – на правила восьмеричной арифметики, где используется операция вычитания.

Первый способ:

ОК 21_8 => ОК 021_8 =>356_8 (обратный код неотрицательного целого числа на 8-разрядном двоичном поле);

$356_8 + 1 = 357_8$ (дополнительный код отрицательного целого числа на 8-разрядном двоичном поле).

Второй способ:

$0 - 21_8 = 357_8$ (дополнительный код отрицательного целого числа на 8-разрядном двоичном поле).

2) от восьмеричной системы счисления теперь перейдём к двоичной:

$17_{10} => 10001_2$ (прямой код неотрицательного целого числа);

$-17_{10} => 11101111_2$ (дополнительный код отрицательного целого числа на 8-разрядном двоичном поле);

Примечание. Покажем оба способа получения дополнительного кода.

Первый способ:

ОК 10001_2 => ОК 00010001_2 => 11101110_2 (обратный код неотрицательного целого числа на 8-разрядном двоичном поле);

$11101110_2 + 1 = 11101111_2$ (дополнительный код отрицательного целого числа на 8-разрядном двоичном поле).

Второй способ:

$0 - 10001_2 = 11101111_2$ (дополнительный код отрицательного целого числа на 8-разрядном двоичном поле).

3) заменим значения всех бит кода с чётными номерами на инверсное значение:

“до” 11101111_2

“после” $10111010_2 => 272_8$

Итак, полученное после преобразования число на 8-разрядном двоичном поле осталось отрицательным, при этом, как видим, по модулю оно стало больше исходного числа (-17_{10}). Чтобы узнать, какому отрицательному десятичному числу соответствует полученное число, от дополнительного кода перейдём к прямому:

ПК целого числа => ОК ДК целого числа + 1

или

ПК целого числа => 0 – ДК целого числа.

Поначалу воспользуемся двоичной арифметикой:

ОК $10111010_2 => 01000101_2$

$01000101_2 + 1 => 01000110_2 => 106_8 => 70_{10}$

или

$0 - 10111010_2 => 01000110_2 => 106_8 => 70_{10}$

Вместо двоичной арифметики можно воспользоваться восьмеричной арифметикой:

ОК $272_8 =>105_8$

$105_8 + 1 => 106_8 => 70_{10}$

или

$0 - 272_8 => 106_8 => 70_{10}$

Итак, полученное число 10111010_2 на 8-разрядном двоичном поле соответствует десятичному числу –70.

Примечание. Обратный код иногда называют поразрядным (побитовым) дополнением (***radix-minus-1 complement***). Поразрядное дополнение в десятичной системе счисления – это дополнение до девяти в каждом разряде числа (***nine’s complement***), а в двоичной системе – дополнение до одного (***one’s complement***). Итак, поразрядное дополнение получается благодаря вычитанию каждой цифры числа из числа, на 1 меньшего основания системы счисления.

Например, выполним поразрядное дополнение для числа 21_8:

ОК 21_8 => 756_8 (обратный код числа на 9-разрядном двоичном поле, который на 8-разрядном поле соответствует коду 356_8).

Например, выполним поразрядное дополнение для числа 10001_2:

ОК 10001_2 => 11101110_2 (обратный код числа на 8-разрядном двоичном поле).

Дополнительный код иногда называют точным дополнением (***true complement*** или ***radix complement***). Точное дополнение в десятичной системе счисления – это дополнение до десяти (***ten's complement***), а в двоичной системе – дополнение до двух (***two's complement***). Точное дополнение – это поразрядное дополнение плюс единица. Точное дополнение получается благодаря вычитанию числа из ближайшей большей степени основания системы счисления.

Например, выполним дополнение до восьми для числа 21_8:

ДК 21_8 => $756_8 + 1$ => 757_8 (дополнительный код числа -21_8 на 9-разрядном двоичном поле, который на 8-разрядном поле соответствует коду 357_8);

или

ДК 21_8 => $400_8 - 21_8$ => 357_8 (дополнительный код числа -21_8 на 9-разрядном двоичном поле, который и на 8-разрядном двоичном поле соответствует этому коду).

Здесь число 8^3 (или 400_8) соответствует ближайшей большей степени основания системы счисления.

Например, выполним дополнение до двух для числа 10001_2:

ДК 10001_2 => $11101110_2 + 1$ => 11101111_2 (дополнительный код числа -10001_2 на 8-разрядном двоичном поле);

или

ДК 10001_2 => $100000000_2 - 10001_2$ => 011101111_2 (дополнительный код числа -10001_2 на 9-разрядном двоичном поле, который на 8-разрядном поле соответствует коду 11101111_2).

Здесь число 2^8 (или 100000000_2) соответствует ближайшей большей степени основания системы счисления.

Описание алгоритма "как есть". Для представления целых чисел будем опираться, например, на 32-разрядную архитектуру. Для операций маскирования будем использовать восьмеричные литералы.

Описание алгоритма представим на естественном языке:

1. ***Начало***
2. ***Положить N = –17***
3. ***Положить M = 0377***
4. ***Положить N = N and M***
5. ***Положить M = 0125***
6. ***Положить N = N xor M***
7. ***Положить M = 037777777400***
8. ***Положить N = N xor M***
9. ***Вывести значение N***
10. ***Конец***

Примечание.

Пункт 3: восьмеричная маска для перехода от 32-разрядного к 8-разрядному двоичному полю.

Пункт 5: восьмеричная маска для замены значения всех бит кода с чётными номерами на инверсное значение.

Пункт 7: восьмеричная маска для перехода от 8-разрядного к 32-разрядному двоичному полю.

Декомпозиция алгоритма приводит к двум “атомарным” операциям:

1) ***clear()*** для побитовой логической операции ***and***;
2) ***set()*** для побитовой логической операции ***xor***.

Очевидно, что в качестве параметра необходимо выбрать маску. Поначалу построим параметризованные вспомогательные алгоритмы “атомарных” операций, а затем на их основе – уже и сам алгоритм. Описание алгоритмов представим на естественном языке.

Описание вспомогательного алгоритма *clear(M)*:

1. ***Начало***
2. ***Положить N = N and M***
3. ***Конец***

Описание вспомогательного алгоритма *set(M)*:

1. ***Начало***
2. ***Положить N = N xor M***
3. ***Конец***

Описание алгоритма:

1. ***Начало***
2. ***Положить N = –17***
3. ***Положить M = 0377***
4. ***Вызвать clear(M)***
5. ***Положить M = 0125***
6. ***Вызвать set(M)***
7. ***Положить M = 037777777400***
8. ***Вызвать set(M)***
9. ***Вывести значение N***
10. ***Конец***

Примечание. Отметим, что переменная ***M***, как и переменная ***N***, обладает глобальным временем жизни, поэтому параметр ***M*** может быть передан вспомогательным алгоритмам как по значению, так и по адресу. Отметим также, что при описании алгоритмов нет средств для явного указания механизма передачи параметров. Однако в некоторых случаях предполагается только неявная передача параметров по значению. Так, литерал в качестве параметра вспомогательному алгоритму может быть передан только по значению.

Представим описание алгоритма, где восьмеричные литералы в качестве параметра ***M*** вспомогательным алгоритмам передаются по значению:

1. ***Начало***
2. ***Положить N = –17***
3. ***Вызвать clear(0377)***
4. ***Вызвать set(0125)***
5. ***Вызвать set(037777777400)***
6. ***Вывести значение N***
7. ***Конец***