Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по лабораторной работе №1

Эмуляция АЛУ. Операции сложения и вычитания целых чисел

Студент: гр. 853504

Вечеринский Максим Сергеевич

Руководитель: старший преподаватель

Шиманский В.В.

Минск 2020

СОДЕРЖАНИЕ

1. Введение
2. Постановка задачи
3. Программная реализация
4. Выводы

Литература

Приложение 1 – Текст программы

1.Введение

1.1. Представление целых чисел.

В двоичной системе счисления числа представляются с помощью комбинации единиц и нулей, знака "минус" и знака разделяющей точки между целой и дробной частью числа. Например, десятичное число -1.312510 в двоичном виде будет выглядеть как -1001.01012. Но в компьютере мы не можем хранить и обрабатывать символы знака и разделяющей точки — для "машинного" представления чисел могут использоваться только двоичные цифры (0 и 1). Если операции выполняются только с неотрицательными числами, то формат представления очевиден. В машинном слове из 8 бит можно представить числа в интервале от 0 до 255. Пример:

|  |  |
| --- | --- |
| *00000000  =* | *0* |
| *00000000  =* | *0* |
| *00000001  =* | *1* |
| *00101001  =* | *41* |
| *10000000  =* | *128* |
| *11111111* | *255* |

В общем случае n-битовая последовательность двоичных цифр an-1an-2…a1a0 интерпретирована как целое число А, значение которого равно

1.2. Прямой код.

Существует несколько соглашений о едином формате представления как положительных, так и отрицательных чисел. Всех их объединяет то, что старший бит слова (с точки зрения европейца — самый левый, или бит, которому при представлении числа без знака должен быть приписан самый большой вес) является битом хранения знака или знаковым разрядом. Все последующие биты слова представляют значащие разряды числа, которые в каждом формате интерпретируются по-своему. Значение 1 в знаковом разряде интерпретируется как представление всем словом отрицательного числа.

00010010 = +18

10010010 = -18

Общее правило математически формулируется следующим образом:

Формат представления чисел в прямом коде неудобен для использования в вычислениях. Во-первых, сложение и вычитание положительных и отрицательных чисел выполняется по-разному, а потому требуется анализировать знаковые разряды операндов. Во-вторых, в прямом коде числу 0 соответствуют две кодовых комбинации:

00000000 = +010

10000000 = -010

Это также неудобно, поскольку усложняется анализ результата на равенство нулю, а такая операция в программах встречается очень часто.

Из-за этих недостатков прямой код практически не применяется при реализации в АЛУ арифметических операций над целыми числами. Вместо этого более широкое применение находит другой формат, получивший наименование дополнительного кода.

1.3. Дополнительный код.

Как и в прямом, в дополнительном коде старший разряд в разрядной сетке отводится для представления знака числа. Остальные разряды интерпретируются не так, как в прямом коде. В табл. 1 перечислены основные свойства дополнительного кода и правила выполнения арифметических операций в дополнительном коде, которые мы рассмотрим в этом и следующем разделах.

Таблица 1. Свойства представления чисел в дополнительном коде

|  |  |
| --- | --- |
| *Диапазон представления на n-разрядной сетке* | от -2n-1 до 2n-1-1 |
| *Количество кодовых комбинаций, соответствующих числу 0* | Одна |
| *Отрицание* | Инвертировать значение в каждом разряде представления исходного числа (положительного или отрицательного), а затем сложить образовавшееся число с числом 0001 по правилам сложения чисел без знака |
| *Расширение разрядности* | Добавить дополнительные разряды слева и заполнить их представления значением, равным значению в знаковом разряде исходного представления |
| *Определение переполнения при сложении* | Если оба слагаемых имеют одинаковые знаки (оба положительны или оба отрицательны), то переполнение возникает в том и только в том случае, когда знак суммы оказывается отличным от знаков слагаемых |
| *Правило вычитания* | Для вычитания числа В из числа А инвертировать знак числа В, как описано выше, и сложить преобразованное число с А по правилам сложения в дополнительном коде |

В большинстве описаний дополнительного кода основное внимание уделяется технике формирования представления отрицательного числа по представлению соответствующего положительного, причем не приводится формальное доказательство работоспособности описанной схемы. Мы решили нарушить эту традицию, и в данном разделе, а также в следующем будем основываться на описании, в котором это представление рассматривается в терминах взвешенной суммы значений разрядов. Такой способ мы уже использовали выше при описании представления чисел без знака и целых чисел со знаком в прямом коде. Преимущество такой методики в том, что она не оставляет ни малейших сомнений в справедливости излагаемых правил выполнения арифметических операций в любых частных случаях.

Рассмотрим n-разрядное двоичное целое число А в дополнительном коде. Если А положительно, то значение его знакового разряда an-1равно 0. В значащих разрядах будет представлена абсолютная величина числа точно так же, как и в прямом коде:

Число 0 считается положительным и, следовательно, в знаковом разряде его представления будет записан код 0, а во всех значащих разрядах также коды 0. Очевидно, что диапазон представления положительных чисел n-разрядным дополнительным кодом простирается от числа 0 до числа 2n-1-1 (для этого числа значения во всех значащих разрядах равны 1). Для представления большего числа потребуется расширение разрядной сетки.

Теперь перейдем к отрицательным числам. Знаковый разряд an-1 дополнительного кода отрицательного числа А (А <0) равен 1. В *n-*1 значащих разрядах может содержаться произвольная комбинация нулей и единиц, а таких комбинаций может быть 2n-1. Следовательно, имеется потенциальная возможность представить отрицательные числа от -1 до -2n-1. Желательно таким образом установить соответствие между двоичными комбинациями и целыми отрицательными числами, чтобы арифметические операции над ним, выполнялись по тем же правилам, что и над числами без знака. В формате целых чисел без знака для вычисления значения числа по его двоичному представлению следует присвоить старшему разряду в разрядной сетке вес +2n-1. При представлении, включающем и знаковый разряд, это приводит к тому, что желаемые арифметические свойства сохраняются, если вес этого разряда будет равен -2n-1. Это соглашение используется при представлении чисел в дополнительном коде. Формально для отрицательного числа в дополнительном коде соблюдается соотношение:

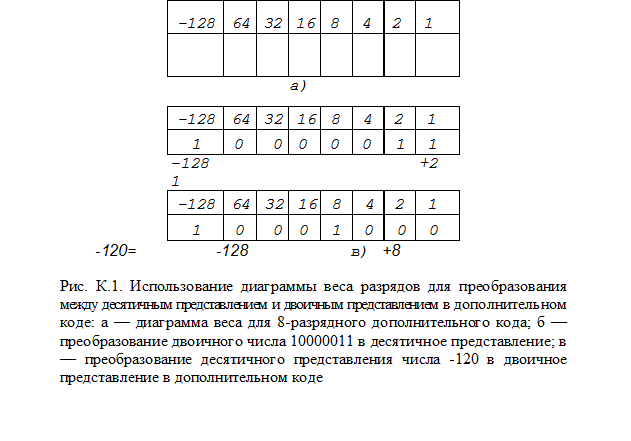
Знаковый разряд an-1, дополнительного кода положительного числа равен 0 и, следовательно, член -2n-1\*an-1=0. Таким образом, верхнее соотношение справедливо для дополнительного кода как положительных, так и отрицательных чисел.

В табл. 2 сравниваются представления 4-разрядных целых чисел в прямом и дополнительном кодах. Хотя с обычной точки зрения представление в дополнительном коде выглядит довольно экзотично, оно значительно упрощает правила выполнения арифметических операций сложения и вычитания. Поэтому такое представление используется для работы с целыми числами в подавляющем большинстве АЛУ современных процессоров.

Таблица 2. Варианты двоичного 4-разрядного представления целых чисел

|  |  |  |  |
| --- | --- | --- | --- |
| Десятичное  представление | Прямой код | Дополнительный код | Смещённое представление |
| +7 | *0111* | *0111* | *1111* |
| +6 | *0110* | *0110* | *1110* |
| +5 | *0101* | *0101* | *1101* |
| +4 | *0100* | *0100* | *1100* |
| +3 | *0011* | *0011* | *1011* |
| +2 | *0010* | *0010* | *1010* |
| +1 | *0001* | *0001* | *1001* |
| +0 | *0000* | *0000* | *1000* |
| -0 | *1000* | *-* | *0111* |
| -1 | *1001* | *1111* | *0110* |
| -2 | *1010* | *1110* | *0101* |
| *-3* | *1011* | *1101* | *0100* |
| *-4* | *1100* | *1100* | *0011* |
| *-5* | *1101* | *1011* | *0010* |
| *-6* | *1110* | *1010* | *0001* |
| *-7* | *1111* | *1001* | *0000* |
| *-8* | *-* | *1000* | *-* |

Хорошей иллюстрацией принципа представления в дополнительном коде является диаграмма веса разрядов (рис. 1), в которой показано, что вес самого младшего разряда (крайней правой позиции на диаграмме) равен 1 (т.е. 20). Вес каждого последующего — возрастает вдвое, и так до крайней левой позиции, знак веса которой инвертируется. Рис. 1-а дает представление о том, почему максимальное по абсолютной величине отрицательное число, которое можно представить в дополнительном коде, равно -2n-1. Код 1 в любом значащем разряде означает добавление во взвешенную сумму положительного числа, равного весу этого разряда. Очевидно также, что положительные числа должны иметь в знаковом разряде код 0, а отрицательные — код 1. Следовательно, самое большое положительное число должно иметь в знаковом разряде код 0, а во всех значащих — код 1 и будет равно 2n-1-1.



На рис. К.1 также показано, как можно использовать диаграмму веса разрядов для преобразования из десятичного представления в двоичное и наоборот.

1.4. Сложение и вычитание в дополнительном коде.

Рассмотрим примеры:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *(а)* | *-7=* | *1001* |  | *(б)* | *-4=* | *1100* |  |
|  | *+5=* | *+0101* |  |  | *+4=* | *+0100* |  |
|  |  | *1110* | *=-2* |  |  | ***1****0000* | *=0* |
| *(в)* | *+3=* | *0011* |  | *(г)* | *-4=* | *1100* |  |
|  | *+4-* | *+0100* |  |  | *-1=* | *+1111* |  |
|  |  | *0111* | *=7* |  |  | *11011* | *=-5* |
| *(д)* | *+5=* | *0101* |  | *(е)* | *-7=* | *1001* |  |
|  | *+4=* | *+0100* |  |  | *-6=* | *+1010* |  |
|  |  | *1001* | *переполнение* |  |  | ***1****0011* | *переполнение* |

Первые четыре примера демонстрируют успешное выполнение операций. Если результат операции должен быть положительным, получается код положительного числа в дополнительном коде, а если отрицательным — код отрицательного числа в дополнительном коде. Обратите внимание на то, что в примере (г) формируется перенос из старшего (знакового) разряда, который игнорируется.

При выполнении сложения чисел с одинаковыми знаками результат может оказаться таким, что не вмещается в используемую разрядную сетку, т.е. получается число, которое выходит за диапазон представления. Появление такого результата расценивается как переполнение (overflow), и на схему АЛУ возлагается функция выявить переполнение и выработать сигнал, который должен воспрепятствовать использованию в дальнейшем полученного ошибочного результата. Существует следующее правило обнаружения переполнения:

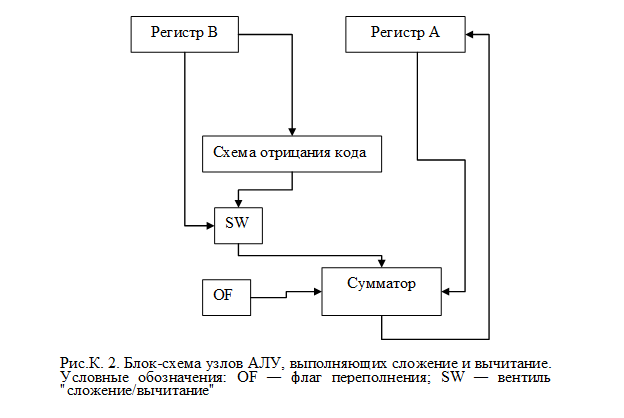
Если знаки слагаемых совпадают, то переполнение возникает в том и только в том случае, когда знак суммы, полученной по правилам сложения в дополнительном коде, отличается от знака слагаемых.

Примеры (д) и (е) иллюстрируют появление переполнения при сложении положительных и отрицательных чисел. Обратите внимание на то, что переполнение может появиться и в том случае, когда возникает перенос из знакового разряда и когда перенос не возникает.

Операция вычитания выполняется по следующему правилу: Для вычитания одного числа (вычитаемого) из другого (уменьшаемого) необходимо предварительно выполнить операцию отрицания над вычитаемым, а затем сложить результат с уменьшаемым по правилам сложен дополнительном коде.

Примеры выполнения с различными знаками.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *(а)   М= 2 =* | *0010* | | | | *(б)* *М= 5 =               0101* | | | |
| *S= 7 =* | *0111* | | | | *S= 2 =               0010* | | | |
| *-S =* | *1001* | | | | *-S =               1110* | | | |
|  | *0010* | | | | *0101* | | | |
|  | *+ 1001* | | | | *+  1110* | | | |
|  | *1011* | | | *=-5* | *3=* | | ***1****0011* | |
| *(в)   М=-5 =* | *1011* | | | | *(г)* *М= 5 =               0101* | | | |
| *S=2 =* | *0010* | | | | *S=-2 =               1110* | | | |
| -S = | *1110* | | | | *-S =                  0010* | | | |
|  | *1011* | | | | *0101* | | | |
|  | *+ 1110* | | | | *+  0010* | | | |
|  | ***1****1001* | | *= -7* | | *7=* | | | *0111* |
| *(д)   М= 7 =* | *0111* | | | | *(е)*  *М=-6 =              1010* | | | |
| *S=-7 =* | *1001* | | | | *S= 4 =              0100* | | | |
| *-S =* | *0111* | | | | *-S=                 1100* | | | |
|  | *0111* | | | | *1010* | | | |
|  | *+ 0111* | | | | *+  1100* | | | |
|  | *1110* | *переполнение* | | | *переполнение* | ***1****0110* | | |
|  |  |  |  |  |  |  |  |  |



На рис. К.2 представлена блок-схема узлов АЛУ, принимавших участие в выполнении операций сложения и вычитания целых чисел. Центральным узлом является двоичный сумматор, на входы которого подают коды слагаемых, а на выходах формируется двоичный код суммы, причём реализация выполняется по правилам сложения чисел без знака. При выполнении сложения оба слагаемых направляются на входы сумматора непосредственно из регистров слагаемых А и В.

Результат передается либо в один из регистров слагаемых (этот вариант показан на схеме), либо в третий регистр результата. Кроме кода результата сумматор формирует сигнал переполнения, который фиксируется в битовом флаге переполнения. Значение флага интерпретируется следующим образом: 0 — переполнение отсутствует, 1 – присутствует. При выполнении операции вычитания код вычитаемого, хранящийся перед началом операции в регистре В, передается на схему, выполняющую операцию отрицания, а уже с выхода этой схемы код поступает на вход сумматора.

2. Постановка задачи

2.1. Текст задания

Эмуляция АЛУ. Реализовать операции сложения и вычитания целых чисел. Использовать дополнительный код.

2.2. Примечание к заданию

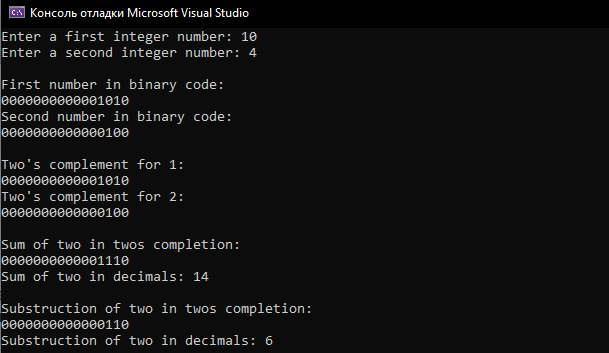
Реализовать ввод двух чисел. Вычислить сумму и разность в двоичной системе счисления в дополнительном коде и вывести результат на экран.

3. Программная реализация

3.1. С консоли вводятся два числа в десятичной системе счисления. Затем реализуется перевод их в двоичную систему счисления. После двоичный код преобразуется в дополнительный. В дополнительном коде реализуется сложение и вычитание. Затем на экран выводится результаты сложения и вычитания в двоичной и десятичной системах счисления.

3.2. Примеры

3.2.1. Тест для «a = 10, b = 4»

Рисунок 1. Скриншот для «a = 10, b = 4»

Ответ: Сумма = 14, Разность = 6.

3.2.2. Тест для «a = 9, b = -5»

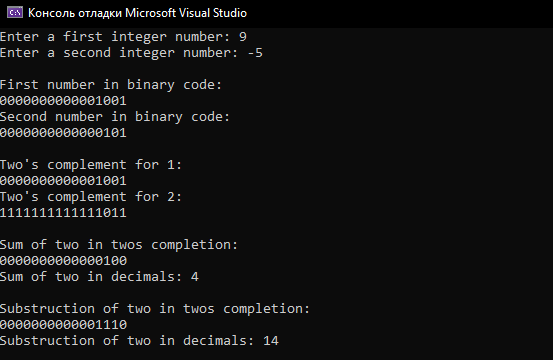


Рисунок 2. Скриншот для «a = 9, b = -5»

Ответ: Сумма = 4, Разность = 14.

3.2.3. Тест для «a = -50, b = 10»

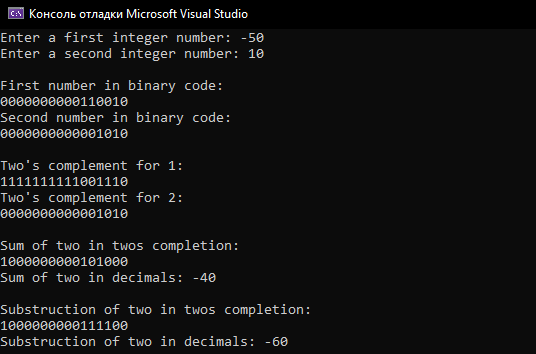


Рисунок 3. Скриншот для «a = -134, b = 100»

Ответ: Сумма = -34, Разность = -234.

3.2.4. Тест для «a = 200, b = 100»

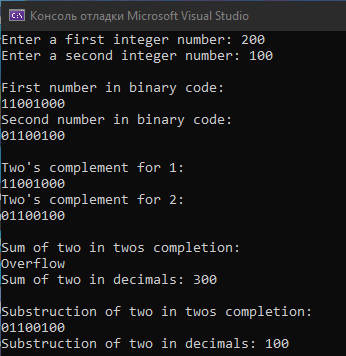


Рисунок 5. Скриншот для «a = 200, b = 100»

Ответ: Сумма = overflow, Разность = 100.

3.2.5. Тест для «a = -200, b = 100»

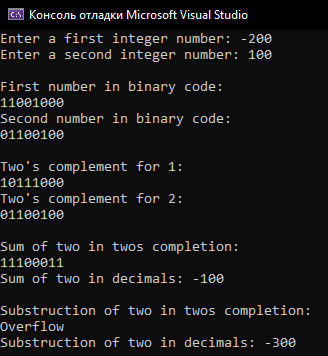


Рисунок 5. Скриншот для «a = -200, b = 100»

Ответ: Сумма = -100, Разность = overflow.

3.2.6. Резюме.

В своих примерах я старался рассмотреть максимально разнообразный набор случаев. Были рассмотрены сложение и разность двух положительных чисел, двух отрицательных чисел, отрицательного и положительного чисел.

Также были рассмотрены экстремальные случаи поведения программы: длинные числа, неправильный ввод.

4. Выводы

В ходе выполнения лабораторной работы ознакомился с понятиями прямого кода, дополнительного кода, сложения и вычитания в прямом и дополнительном коде, изучил различные алгоритмы сложения и вычитания бинарных чисел в дополнительном коде.

Так же рассмотрел работу арифметико-логического устройства (АЛУ), рассмотрел различные нюансы использования АЛУ. Для реализации использовал язык С++ и интегрированную среду разработки Visual Studio.

Литература

1. Волорова Н. А. Лабораторный практикум по курсу «Архитектура вычислительных систем» для студентов специальности «Информатика» /985-444-487-2 – Мн.: БГУИР, 2003. — 32 с.: ил.

2. Tanenbaum A. S. «Structured computer organization» /Vrije Universiteit Amsterdam, The Netherlands, 2013. — 810 c.:ил.

ПРИЛОЖЕНИЕ 1

Текст программы

#include <iostream>

#include <string>

#include <conio.h>

#include <stdio.h>

#include <cstdlib>

#include <math.h>

#include <limits>

using namespace std;

int bits = 16;

long long a, b;

string pryamoiCode(long long x)

{

string value = "";

for (int i = 0; i < bits; i++)value += '0';

//if(x<0)value[0]='1';

x = abs(x);

for (int i = 0; x > 0; i++) {

int j = bits - i - 1;

value[j] = (x % 2) ? '1' : '0';

x /= 2;

}

return value;

}

string obratnyiCode(string value)

{

value[0] = '1';

for (int i = 1; i < value.length(); i++) {

value[i] = (value[i] == '1') ? '0' : '1';

}

return value;

}

string sum(string e, string q)

{

string val = "";

for (int i = 0; i < bits; i++)val += "0";

bool pre = false, last = false;

for (int i = bits - 1; i >= 0; i--) {

pre = last;

last = false;

if (e[i] == q[i]) {

last = e[i] == '1';

val[i] = '0';

}

else val[i] = '1';

if (pre) {

val[i] = (val[i] == '0') ? '1' : '0';

last = (last || val[i] == '0');

}

pre = false;

}

if (last)return sum(val, pryamoiCode(1));

return val;

}

string dopCode(string input)

{

string sec\_str = "";

for (int i = 0; i < bits; i++) sec\_str += '0';

sec\_str[bits - 1] = '1';

string out = sum(input, sec\_str);

return out;

}

int getValue()

{

int f;

cin >> f;

if (cin.fail())

{

cin.clear();

cin.ignore(32676,’\n’);

cout << "You have entered wrong input" << endl;

}

else

{

cin.ignore(32676,’\n’);

return f;

}

}

int main()

{

cout << "Enter a first integer number: ";

int a = getValue();

cout << "Enter a second integer number: ";

int b = getValue();

if (bits <= 1 || powl(2, bits - 1) < abs(a) || powl(2, bits - 1) < abs(b)) {

cout << "Error" << endl;

system("pause");

return 0;

}

string sa, sb, smb; // binary code

string a\_twos, b\_twos; // two's complement

//binary

if (a < 0) sa = obratnyiCode(pryamoiCode(a));

else sa = pryamoiCode(a);

if (b < 0) sb = obratnyiCode(pryamoiCode(b));

else sb = pryamoiCode(b);

if (-b < 0) smb = obratnyiCode(pryamoiCode(-b));

else smb = pryamoiCode(-b);

//two's complement

if (a >= 0) a\_twos = sa; else a\_twos = dopCode(sa);

if (b >= 0) b\_twos = sb; else b\_twos = dopCode(sb);

//output

cout << "\nFirst number in binary code: \n";

cout << pryamoiCode(a) << endl;

cout << "Second number in binary code: \n";

cout << pryamoiCode(b) << endl;

cout << "\nTwo's complement for 1: \n";

cout << a\_twos << endl;

cout << "Two's complement for 2: \n";

cout << b\_twos << endl;

cout << "\nSum of two in twos completion: \n";

cout << ((a + b < 0) ? obratnyiCode(sum(sa, sb)) : (sum(sa, sb))) << endl;

cout << "Sum of two in decimals: ";

cout << a + b << endl << endl;

cout << "\Substruction of two in twos completion: \n";

cout << ((a - b < 0) ? obratnyiCode(sum(sa, smb)) : (sum(sa, smb))) << endl;

cout << "Substruction of two in decimals: ";

cout << a - b << endl;

return 0;

}