Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Архитектура вычислительных систем

ОТЧЁТ

к лабораторной работе

на тему

Арифметические операции с целыми числами

Выполнил: студент группы 153501
Тимофеев Кирилл Андреевич

Проверила: Калиновская Анастасия
Александровна

Минск 2023

Цель

Изучить формат хранения чисел в памяти компьютера. Алгоритмы перевода числа в прямой код, дополнительный код, отрицания, сложения, вычитания, умножения и деления чисел с фиксированной точкой. Составить программу, реализующую все вышеперечисленные алгоритмы. Программный продукт протестировать.

Теоретические сведения

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 -1.3125_{10} в двоичном виде будет выглядеть как -1001.0101_2. 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.

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

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 = $+0_{10}$
10000000 = -0_{10}

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.

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. При отрицании 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. При расширении разрядности добавить дополнительные разряды слева и заполнить их представления значением, равным значению в знаковом разряде исходного представления. Переполнение при сложении происходит, 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. Для вычитания числа B из числа A инвертировать знак числа B, как описано выше, и сложить преобразованное число с A по правилам сложения в дополнительном коде.

Если число положительное, то его дополнительный код аналогичен его прямому коду. А если число отрицательное, то инвертируем значение отрицательного числа(знаковый бит не трогаем), записанного в прямом коде, и к полученной инверсии прибавляем 1.

2. Преобразование при изменении длины разрядной сетки

Иногда возникает необходимость записать n-разрядное целое двоичное число в слово длиной m бит, причем m>n. Если исходное число представлено в прямом коде, такое преобразование выполняется довольно просто — нужно перенести знаковый разряд в крайний левый бит нового слова, а остальные дополнительные биты заполнить нулями.

Преобразование дополнительного кода при расширении разрядной сетки выполняется следующим образом: нужно скопировать значение знакового бита во все дополнительные биты. Если исходное число было положительным, то все дополнительные биты заполнятся нулями, а если отрицательным – единицами. Эта операция называется расширением знака.

3. Представление с фиксированной точкой

И наконец, следует остановиться еще на одном нюансе. Описанные выше форматы объединяются часто одним термином — формат с фиксированной точкой. Суть его в том, что положение разделительной точки между целой и дробной частями числа неявно фиксируется на разрядной сетке. В настоящее время принято фиксировать точку справа от самого младшего значащего разряда. Программист может использовать аналогичное представление для работы с двоичными дробными числами, мысленно фиксируя точку перед старшим значащим разрядом и соответственно масштабируя результаты преобразований, выполняемых стандартными программными или аппаратными средствами.

4. Арифметические операции над целыми числами

1) Отрицание

Операция отрицания числа, представленного в прямом коде, выполняется очень просто - нужно инвертировать значение знакового разряда. Если же число представлено в дополнительном коде, отрицание выполняется несколько сложнее. Правило выполнения этой операции формулируется следующим образом.

1 . Следует инвертировать значение в каждом разряде представления исходного числа (положительного или отрицательного), включая и знаковый, т.е. установить значение 1 в тех разрядах, где ранее было значение 0, и значение 0 — в тех разрядах, где ранее было значение 1 (эту операцию иногда называют поразрядным дополнением — bitwise complement, а ее результат — инверсным кодом).

2.Нужно сложить образовавшееся число с числом 0. . .001 по правилам сложения чисел без знака.

Иногда эту операцию называют вычислением дополнения числа в дополнительном коде.

2) Сложение и вычитание в дополнительном коде

Если оба числа имеют n–разрядное представление, то алгебраическая сумма будет получена по правилам двоичного сложения (включая знаковый разряд), если отбросить возможный перенос из старшего разряда. Если числа принадлежат диапазону представимых данных и имеют разные знаки, то сумма всегда будет лежать в этом диапазоне.

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.

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.

3) Умножение (алгоритм Бута)

Алгоритм Бута включает в себя циклическое сложение одного из двух заранее установленных значений A и S с произведением P, а затем выполнение арифметического сдвига вправо над P. Пусть **m** и **r** — множимое и множитель соответственно, а x и y представляют собой количество битов в **m** и **r**.

1. Установить значения A и S, а также начальное значение P. Каждое из этих чисел должно иметь длину, равную $(x + y + 2)$.

1) A: Заполнить наиболее значимые (левые) биты значением **m**. Заполнить оставшиеся $(y + 2)$ бит нулями.

2) S: Заполнить наиболее значимые биты значением $(-\mathbf{m})$ в дополнительном коде. Заполнить оставшиеся $(y + 2)$ бит нулями.

3) P: Заполнить наиболее значимые x бит нулями. Справа от них заполнить биты значением **r**. Записать 0 в крайний наименее значимый (правый) бит

2. Определить значение двух наименее значимых (правых) битов P и вычислить по ним значение для следующего шага:

1) Если их значение равно 01, прибавить A к P. Переполнение игнорировать.

2) Если их значение равно 10, прибавить S к P. Переполнение игнорировать.

3) Если их значение равно 00, действие не требуется. P используется без изменений на следующем шаге.

4) Если их значение равно 11, действие не требуется. P используется без изменений на следующем шаге.

3. Выполнить операцию арифметического сдвига над значением, полученным на втором шаге, на один бит вправо. Присвоить *P* это новое значение.

4. Повторить шаги 2 и 3 *y* раз.

5. Отбросить старший и младший биты *P*. Это и есть произведение **m** и **r**.

4) Алгоритм деления:

Изначально храним делитель размерности n(M), делимое размерности n(Q), которое в результате будет хранить целое от деления, и массив(A) размерности n для остатка, заполненный значениями, равными знаковому биту делимого.

Следующие операции производит в цикле n раз:

1) делаем логический сдвиг A:Q(A и Q – одно целое)

2) если коды в регистрах M и A имеют одинаковые знаки, вычесть из содержимого A содержимое M и оставить результат в A. В противном случае добавить к содержимому A код из M.

3) предыдущая операция считается успешной, если знак кода в регистре A не изменился в результате ее выполнения.

4) если операция была успешной или содержимое регистров A и Q равно нулю, то установить в младшем разряде частного (разряде Q_0) код 1. В противном случае устанавливаем в младшем разряде код 0.

5) заканчиваем итерацию

После завершения деления в A будет находится остаток, в Q целая часть. Если изначальные делитель и делимое разных знаков, то целое от деления нужно взять с обратным знаком по правилам представления чисел в дополнительном коде.

5) Дополнительный алгоритм деления:

1) Получаем двоичное представление делителя в минус первой степени.

2) Умножаем на 2^{32}.

3) Умножаем полученное число на делимое.

4) Результат делим на 2^{32}.

Код программы

```
#include <bitset>
#include <iostream>
using namespace std;
typedef int8_t integer;
typedef int16_t d_integer;
const size_t bitsize = 8 * sizeof(integer);

void showstep(bitset<bitsize> n1, bitset<bitsize> n2){
                        cout << "A: " << n1.to_string() << " B: " <<
n2.to_string() << "\n";
}

void showstep(bitset<2*bitsize> n1, bitset<bitsize> n2){
                        cout << "A: " << n1.to_string() << " B: " <<
n2.to_string() << "\n";
}

integer toRes(bitset<bitsize> bits){
                        integer res = 0;
                        for (int i = 0; i < bitsize; ++i){
                                                integer shift = 1 & bits[i];
                                                shift <<= i;
                                                res |= shift;
                        }
                        return res;
}
d_integer toRes(bitset <2 * bitsize> bits){
                        d_integer res = 0;
                        for (int i = 0; i < 2 * bitsize; ++i)
                        {
                                                d_integer shift = 1 & bits[i];
                                                shift <<= i;
                                                res |= shift;
                        }
                        return res;
}

integer add(integer _n1, integer _n2, bool mode = false)
{
                        bitset<bitsize> n1 = _n1, n2 = _n2;

                        bool c = 0;

                        for (int i = 0; i < bitsize; ++i)
                        {
                                                bool andop = n1[i] && n2[i];
                                                bool orop = n1[i] || n2[i];

                                                if (andop || !orop) n1[i] = 0;
                                                else n1[i] = 1;

                                                bool ctemp = andop;

                                                andop = n1[i] && c;
                                                orop = n1[i] || c;

                                                if (andop || !orop) n1[i] = 0;
                                                else n1[i] = 1;

                                                c = ctemp || andop;

                                                if (mode)
                                                {
```

```
        showstep(n1, n2);

        cin.get();
                                                    }
                            }

                            return toRes(n1);
}

integer neg(integer _n){
        bitset<bitsize> n = _n;
                            n.flip();
                            return add(toRes(n), 1);
}
integer sub(integer _n1, integer _n2, bool mode = false){
                            return add(_n1, neg(_n2), mode);
}

template <size_t s> bitset<s> math_shift(bitset<s> bits){
                            bool tmp = bits[s - 1];
                            bits >>= 1;
                            bits[s - 1] = tmp;
                            return bits;
}

d_integer mult(integer _n1, integer _n2, bool mode = false){
                            bitset<2 * bitsize> res = _n1;
                            for (int i = 0; i < bitsize; ++i) res[i + bitsize] = 0;
                            bool temp = 0;
                            for (int i = 0; i < bitsize; ++i){
                                                    if (res[0] != temp){

        bitset<bitsize> a = 0;
                                                                                for
(int i = 0; i < bitsize; ++i)

                            a[i] = res[bitsize + i];

                                                                                if
(res[0] == 0)

                            a = add(toRes(a), _n2);
                                                                                else

                            a = sub(toRes(a), _n2);


                                                                                for
(int i = 0; i < bitsize; ++i)

                            res[bitsize + i] = a[i];
                                                    }

                                                    temp = res[0];
                                                    res = math_shift(res);
                                                    if (mode)
                                                    {

        showstep(res, _n2);

        cin.get();
                                                    }
                            }
```

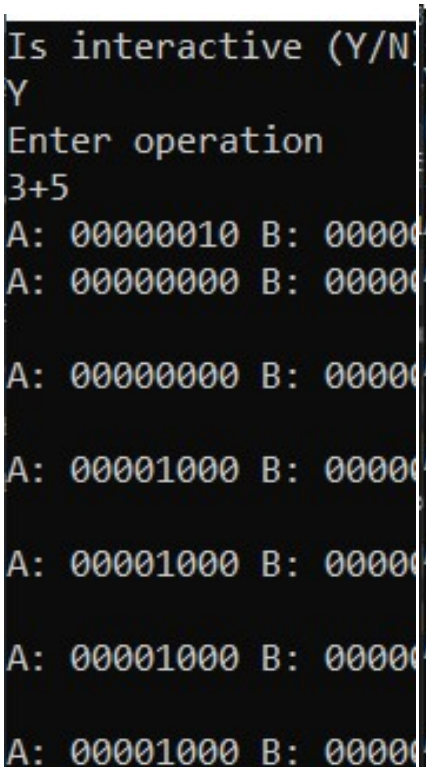
```
                        return toRes(res);
}

integer myDiv(integer _n1, integer _n2, bool mode = false)
{
                        bitset<bitsize> magic = 0;

                        integer count = 1;
                        bool sign1 = _n1 < 0;
                        bool sign2 = _n2 < 0;
                        if (sign1) _n1 = neg(_n1);
                        if (sign2) _n2 = neg(_n2);

                        for (int i = 0; i < bitsize; ++i)
                        {
                                                magic <<= 1;

                                                count = count << 1;
                                                if (count > _n2) //sub(count,
_n2) && (1 << (bitsize - 1)
                                                {
                                                                        magic
= add(toRes(magic), 1);
                                                                        count
= sub(count, _n2);
                                                }

                        }

                        magic = add(toRes(magic), 1);

                        return sign1==sign2 ? mult(_n1, toRes(magic)) >> bitsize:
neg(mult(_n1, toRes(magic)) >> bitsize);
}

integer _div(integer _n1, integer _n2, bool mode = false)
{
                        bitset<2*bitsize> QA = _n1;
                        bool sign = _n2 < 0;
                        for (int i = 0; i < bitsize; ++i)
                        {
                                                QA <<= 1;
                                                bool QA_sign = QA[bitsize - 1];

                                                bitset<bitsize> A = 0;
                                                for (int i = 0; i < bitsize;
++i)
                                                                        A[i] =
QA[bitsize + i];

                                                bitset<bitsize> Q = 0;
                                                for (int i = 0; i < bitsize;
++i) Q[i] = QA[i];

                                                bitset<bitsize> A_temp;
                                                if ( QA_sign == sign)
                                                                        A_temp
= sub(toRes(A), _n2);
                                                else
                                                                        A_temp
= add(toRes(A), _n2);

                                                if (QA_sign == A_temp[bitsize -
1] || A_temp.none() || Q.none())
```

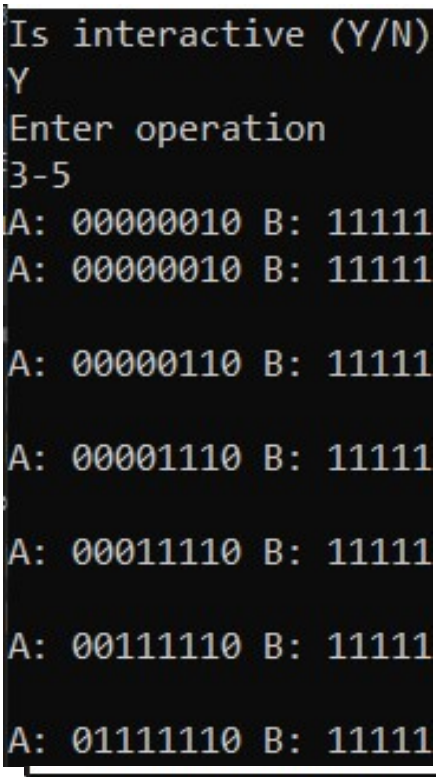
```
                                                {
                                                                        QA[0]
= 1;
                                                                        for
(int i = 0; i < bitsize; ++i) QA[i + bitsize] = A_temp[i];

                                                }

                                                else
                                                {
                                                                        QA[0]
= 0;
                                                                        for
(int i = 0; i < bitsize; ++i) QA[i + bitsize] = A[i];
                                                }

                                                if (mode)
                                                {

        showstep(QA, _n2);

        cin.get();
                                                }
                        }

                        bitset<bitsize> answer = 0;
                        for (int i = 0; i < bitsize; ++i)
                                                answer[i] = QA[i];


                        if (sign != _n1 < 0)
                                                return neg(toRes(answer));
                        else
                                                return toRes(answer);
}

integer _mod(integer _n1, integer _n2, bool mode = false)
{
                        bitset<2 * bitsize> QA = _n1;
                        bool sign = _n2 < 0;
                        for (int i = 0; i < bitsize; ++i)
                        {
                                                QA <<= 1;
                                                bool QA_sign = QA[bitsize - 1];

                                                bitset<bitsize> A = 0;
                                                for (int i = 0; i < bitsize;
++i)
                                                                        A[i] =
QA[bitsize + i];

                                                bitset<bitsize> Q = 0;
                                                for (int i = 0; i < bitsize;
++i) Q[i] = QA[i];

                                                bitset<bitsize> A_temp;
                                                if (QA_sign == sign)
                                                                        A_temp
= sub(toRes(A), _n2);
                                                else
                                                                        A_temp
= add(toRes(A), _n2);
```

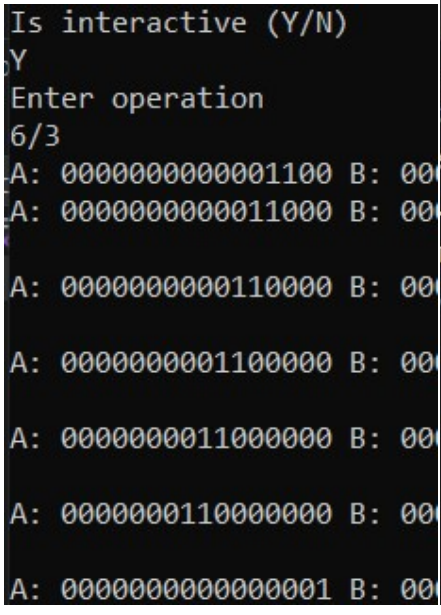
```
                                                    if (QA_sign == A_temp[bitsize -
1] || A_temp.none() || Q.none())
                                                    {
                                                                                    QA[0]
= 1;
                                                                                    for
(int i = 0; i < bitsize; ++i) QA[i + bitsize] = A_temp[i];
                                                    }

                                                    else
                                                    {
                                                                                    QA[0]
= 0;
                                                                                    for
(int i = 0; i < bitsize; ++i) QA[i + bitsize] = A[i];
                                                    }

                                                    if (mode)
                                                    {

        showstep(QA, _n2);

        cin.get();
                                                    }
                            }

                            bitset<bitsize> answer = 0;
                            for (int i = 0; i < bitsize; ++i)
                                                    answer[i] = QA[bitsize + i];

                            return toRes(answer);
}
int main()
{
                            int a, b;
                            char ch;
                            bool mode;
                            cout << "Is interactive (Y/N)\n";
                            cin >> ch;
                            mode = ch == 'Y';
                            cout << "Enter operation\n";
                            cin >> a >> ch >> b;

                            if (ch == '+') cout << (int)add(a, b, mode);
                            if (ch == '/') cout << (int)_div(a, b, mode);
                            if (ch == '-') cout << (int)sub(a, b, mode);
                            if (ch == '*') cout << (int)mult(a, b, mode);
                            if (ch == '\\') cout << (int)myDiv(a, b, mode);
                            if (ch == '%') cout << (int)_mod(a, b, mode);
}
```

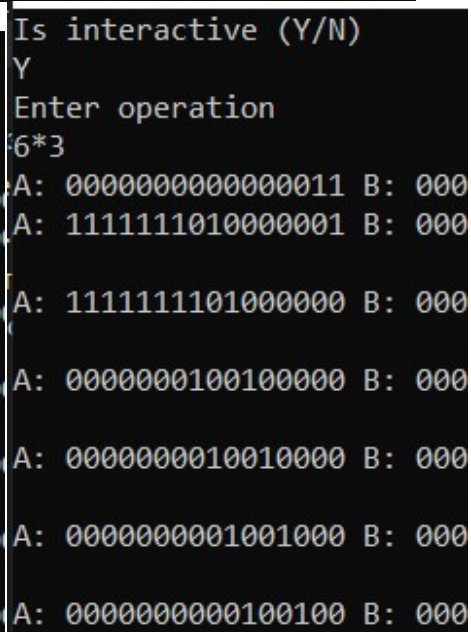
Результат программы

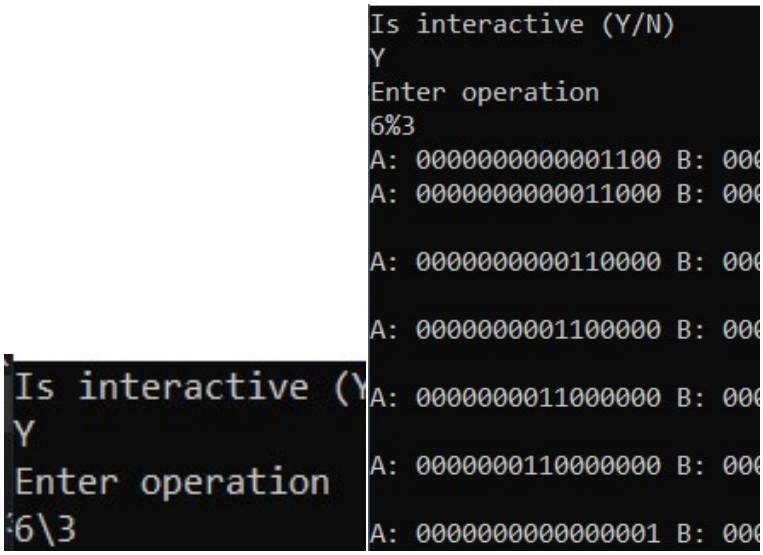
Для проверки были введены числа -15.5 и 6.125











Вывод

В ходе лабораторной работы были изучены формат хранения чисел в памяти компьютера. Алгоритмы перевода числа в прямой код, дополнительный код, отрицания, сложения, вычитания, умножения и деления чисел с фиксированной точкой. Также была составлена программа, реализующая все вышеперечисленные алгоритмы. Программный продукт протестирован, все поставленные задачи выполнены.