Примечание.

Надпись «CORRECT» обозначает что число полностью уместилось в 14-ти разрядную сетку, в случае если число оказалось больше система запишет только ту его часть, которая поместилась в предел, и выведет напротив записи ошибку ввода – «ERROR»

Надпись условие обозначает соответствие записи условию моего варианта «вывод чисел, у которых биты 10, 12, 2 обнулены.»

Также я разбил задание на два класса – основной, в котором происходят сдвиги и печать и вспомогательный, отвечающий за операции над двумя числами.

Консоль отладки Microsoft Visual Studio

```
ДЕМОНСТРАЦИЯ РАБОТЫ ОСНОВНОГО КЛАССА
 Биты  13 12 11 10  9  8  7  6  5  4  3  2  1  0   10 - код     16 - код     Ввод     Услови
        0  1  0  1  1  0  1  1  1  0  0  0  0  1   22241        0x56e1       ERROR      NO
        1  0  0  1  0  1  1  0  1  1  1  1  0  0   9660         0x25bc       CORRECT    NO
        0  1  0  1  1  0  1  0  1  1  0  1  1  1   5815         0x16b7       CORRECT    NO
        1  1  0  1  0  1  0  1  0  1  0  0  0  1   30033        0x7551       ERROR      NO
        1  0  0  1  1  0  0  1  0  1  0  1  0  0   9812         0x2654       CORRECT    NO
        0  0  1  0  0  0  1  0  0  0  0  0  1  1   18563        0x4883       ERROR      YES
        1  0  1  1  1  1  0  1  0  0  0  0  1  1   12099        0x2f43       CORRECT    NO
        0  0  1  1  0  0  0  1  0  0  1  1  1  0   3150         0xc4e        CORRECT    NO
        0  1  0  1  0  1  1  0  1  1  1  1  0  1   5565         0x15bd       CORRECT    NO
        0  1  1  0  1  0  0  1  1  0  1  1  0  0   23148        0x5a6c       ERROR      NO
        1  0  0  0  0  1  0  1  0  1  0  1  1  1   24919        0x6157       ERROR      NO
        1  0  0  1  0  1  1  1  0  1  1  1  1  1   9695         0x25df       CORRECT    NO
        0  0  0  0  1  1  1  1  0  1  1  1  1  1   991          0x3df        CORRECT    NO
        0  0  0  0  0  0  1  1  0  0  0  0  0  0   192          0xc0         CORRECT    YES
        1  1  1  0  0  0  0  1  1  0  1  1  1  1   14447        0x386f       CORRECT    NO
        1  0  1  1  0  1  1  0  1  0  0  0  1  0   28066        0x6da2       ERROR      NO
        0  0  0  0  1  1  1  0  1  1  1  0  0  1   17337        0x43b9       ERROR      YES
        0  0  0  0  0  1  0  0  0  1  0  0  0  1   273          0x111        CORRECT    YES
        1  0  1  0  0  0  0  0  0  1  1  1  0  1   26653        0x681d       ERROR      NO
        1  1  1  0  1  1  1  1  1  1  0  0  0  0   31728        0x7bf0       ERROR      NO
Зададим бит как string (воспользуемся перегрузкой конструктора)
        0  0  0  0  0  1  0  0  1  0  0  1  1  1   295          0x127        CORRECT    NO
SetBit(1234)
        0  0  0  1  0  0  1  1  0  1  0  0  1  0   1234         0x4d2        CORRECT    NO
Invert()
        1  1  1  0  1  1  0  0  1  0  1  1  0  1   1234         0x4d2        CORRECT    NO
MoveLeft(3)
        0  1  1  0  0  1  0  1  1  0  1  0  0  0   1234         0x4d2        CORRECT    NO
MoveRight(3)
        0  0  0  0  1  1  0  0  1  0  1  1  0  1   1234         0x4d2        CORRECT    NO
LoopMoveLeft(5)
        1  0  0  1  0  1  1  0  1  0  0  0  0  1   1234         0x4d2        CORRECT    NO
LoopMoveRight(4)
        0  0  0  1  1  0  0  1  0  1  1  0  1  0   1234         0x4d2        CORRECT    NO


ДЕМОНСТРАЦИЯ РАБОТЫ КЛАССА БИТОВЫХ ОПЕРАЦИЙ
Покажем <<И>> <<ИЛИ>> <<Исключающие ИЛИ>> на примере чисел 1232 и 1833
Conjunction   0  0  0  1  0  0  0  0  0  0  0  0  0  0
Disjunction   0  0  0  1  1  1  1  1  1  1  1  0  0  1
ExclusiveOr   0  0  0  0  1  1  1  1  1  1  1  0  0  1
```

КОД ПРОГРАММЫ.

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <string>
#include <bitset>  //  2 представление
#include <sstream> // 16 представление

using namespace std;

class BitMove
{
public:
    bool Error = false;
    bool Requirement; // вывод чисел, у которых биты 10, 12, 2 обнулены.
    int Number;
    string NumberBit;
    string NumberOx;

    int toInt(string bitString)
    {
        int BitLength = bitString.length(), tempInt;
        int IntResult = 0;

        for (int i = 0; i < BitLength; i++)
        {
            if (bitString[i] != '0' and bitString[i] != '1')
            {
                cout << "Бит введен не корректно!\n";
                break;
            }
        }

        for (int i = 0; i < BitLength; i++)
        {
            tempInt = bitString[i] - '0';
            IntResult |= (1 << (BitLength - 1 - i)) * tempInt;
        }

        return IntResult;
    }

    BitMove(int NewNumber)
    {
        if (NewNumber > 16383)
        {
            Number = NewNumber;
            Error = true;
        }
        else
        {
            Number = NewNumber;
        }

        NumberBit = bitset<14>(Number).to_string();
        ostringstream ss; ss << hex << Number;
        NumberOx = ss.str();
        RequirementFulfilled();
    }

    BitMove(string bitString)
    {
        int BitLength = bitString.length();
        if (BitLength > 14)
```

```
    {
        Number = toInt(bitString);
        Error = true;
    }
    else
    {
        Number = toInt(bitString);
    }

    NumberBit = bitset<14>(Number).to_string();
    ostringstream ss; ss << hex << Number;
    NumberOx = ss.str();
    RequirementFulfilled();
}

/* Функции операций над классом */

void SetBit(int NewNumber) // Установка бита, введенного с клавиатуры
{
    if (NewNumber > 16383)
    {
        Number = NewNumber;
        Error = true;
    }
    else
    {
        Number = NewNumber;
    }

    NumberBit = bitset<14>(Number).to_string();
    ostringstream ss; ss << hex << Number;
    NumberOx = ss.str();
    RequirementFulfilled();
}

void ClearBit() // Сброс бита, введенного с клавиатуры
{
    Number = NULL;
    NumberBit = "";
    NumberOx = "";
}

string Invert() // Инферитирование бита
{
    string InvertBit = "";
    for (int i = 0; i < 14; i++)
    {
        if (NumberBit[i] == '0')
        {
            InvertBit += '1';
        }
        else
        {
            InvertBit += '0';
        }
    }
    return InvertBit;
}

/* Функции сдвигов */
void MoveLeft(int step) // Сдвиг влево на L разрядов
{
    for (int i = 0; i < step; i++)
    {
        for (int ii = 0; ii < 13; ii++)
```

```
            {
                NumberBit[ii] = NumberBit[ii + 1];
            }
            NumberBit[13] = '0';
        }
    }

    void MoveRight(int step) // Сдвиг вправо на P разрядов
    {
        for (int i = 0; i < step; i++)
        {
            for (int ii = 14; ii > 0; ii--)
            {
                NumberBit[ii] = NumberBit[ii - 1];
            }
            NumberBit[0] = '0';
        }
    }

    void LoopMoveLeft(int step) // Циклический сдвиг влево на L разрядов
    {
        for (int i = 0; i < step; i++)
        {
            char save = NumberBit[0];
            for (int ii = 0; ii < 13; ii++)
            {
                NumberBit[ii] = NumberBit[ii + 1];
            }
            NumberBit[13] = save;
        }
    }

    void LoopMoveRight(int step) // Циклический сдвиг вправо на P разрядов
    {
        for (int i = 0; i < step; i++)
        {
            char save = NumberBit[13];
            for (int ii = 14; ii > 0; ii--)
            {
                NumberBit[ii] = NumberBit[ii - 1];
            }
            NumberBit[0] = save;
        }
    }

    void RequirementFulfilled() // вывод чисел, у которых биты 10, 12, 2 обнулены.
    {
        if (NumberBit[1] == '0' and NumberBit[3] == '0' and NumberBit[11] == '0')
        {
            Requirement = true;
        }
        else
        {
            Requirement = false;
        }
    }
};

class BitOperation
{
    int Number1;
    int Number2;
    string BitResult;

    void PrintResult()
```

```
    {
        for (int i = 0; i < 14; i++) { cout << BitResult[i] << "  "; }
        cout << "\n";
    }

public:
    BitOperation(int NewNumber1, int NewNumber2)
    {
        Number1 = NewNumber1;
        Number2 = NewNumber2;
    }

    void Conjunction() // Поразрядная операция «И»
    {
        int Result = Number1 & Number2;
        BitResult = bitset<14>(Result).to_string();
        PrintResult();
    }

    void Disjunction() // Поразрядная операция «ИЛИ»
    {
        int Result = Number1 | Number2;
        BitResult = bitset<14>(Result).to_string();
        PrintResult();
    }

    void ExclusiveOr() // Поразрядная операция «исключающие ИЛИ»
    {
        int Result = Number1 ^ Number2;
        BitResult = bitset<14>(Result).to_string();
        PrintResult();
    }
};

ostream& operator << (ostream& stream, const BitMove& counter)
{
    stream << "\t";
    for (int i = 0; i < 14; i++) { stream << counter.NumberBit[i] << "  "; }
    printf(" %-8d", counter.Number);
    printf("\tOx%-11s", counter.NumberOx.c_str());
    if (counter.Error ? printf("%-11s", "ERROR") : printf("%-11s", "CORRECT"));
    if (counter.Requirement ? printf("%3s", "YES\n") : printf("%3s", "NO\n"));
    return stream;
}

void main()
{
    setlocale(LC_ALL, "Rus");
    srand(time(NULL));
    system("color A");

    cout << "ДЕМОНСТРАЦИЯ РАБОТЫ ОСНОВНОГО КЛАССА\n";
    cout << " Биты  13 12 11 10  9  8  7  6  5  4  3  2  1  0   10 - код     16 -
код     Ввод     Условие\n";
    for (int i = 0; i < 20; i++)
    {
        BitMove NewBit(rand() % 32000); cout << NewBit;
    }
    cout << "Зададим бит как string (воспользуемся перегрузкой конструктора)\n";
    BitMove NewBit("0100100111"); cout << NewBit;

    cout << "SetBit(1234)\n";
    NewBit.SetBit(1234); cout << NewBit;

    cout << "Invert()\n";
```

```
    NewBit.Invert(); cout << NewBit;

    cout << "MoveLeft(3)\n";
    NewBit.MoveLeft(3); cout << NewBit;

    cout << "MoveRight(3)\n";
    NewBit.MoveRight(3); cout << NewBit;

    cout << "LoopMoveLeft(5)\n";
    NewBit.LoopMoveLeft(5); cout << NewBit;

    cout << "LoopMoveRight(4)\n";
    NewBit.LoopMoveRight(4); cout << NewBit;

    cout << "\n\nДЕМОНСТРАЦИЯ РАБОТЫ КЛАССА БИТОВЫХ ОПЕРАЦИЙ\n";
    cout << "Покажем <<И>> <<ИЛИ>> <<Исключающие ИЛИ>> на примере чисел 1232 и
1833\n";
    BitOperation Bit(1232, 1833);
    printf("%-14s", "Conjunction"); Bit.Conjunction();
    printf("%-14s", "Disjunction"); Bit.Disjunction();
    printf("%-14s", "ExclusiveOr"); Bit.ExclusiveOr();
}
```