МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №7
по дисциплине: Теория цифровых автоматов
тема: «Диагностика неисправностей многовыходных комбинационных схем»

Выполнил: ст. группы ВТ-32
Воскобойников И. С.
Проверил: Рязанов Ю. Д,

Белгород 2020 г.

Цель работы: научиться строить диагностические тесты и алгоритмы распознавания неисправностей многовыходных комбинационных схем.

При выполнении лабораторной работы нужно решить следующую задачу.

Дано:

3) многовыходная комбинационная схема, построенная при выполнении лабораторной работы № 6;

4) множество одиночных неисправностей, состоящее из неисправностей «константа 0» и «константа 1» на каждом входе схемы.

Найти: диагностический тест для заданного множества неисправностей.

Построить: алгоритм распознавания неисправностей.

Для решения задачи нужно выполнить следующие задания.

1. Представить в аналитической форме систему булевых функций, реализуемую исправной комбинационной схемой.

2. Представить в аналитической форме систему булевых функций для каждой неисправности.

3. Представить в аналитической форме разностные функции для каждой неисправности.

4. Написать программу, которая строит диагностическую матрицу на основе аналитического представления разностных функций.

5.По диагностической матрице найти минимальный диагностический тест.

6. Написать программу, которая по аналитическому представлению систем булевых функций неисправностей строит таблицу их значений на тестовых наборах. Для компактного представления таблицы рекомендуется значение системы булевых функций неисправности представлять не двоичным вектором, а десятичным числом.

7. По полученной в п. 6 таблице построить алгоритм распознавания неисправностей в виде диагностического дерева.

8. Определить неисправность, которая распознается дольше других, написать программу, которая моделирует схему с этой неисправностью, провести диагностический эксперимент.

1. Представить в аналитической форме систему булевых функций, реализуемую исправной комбинационной схемой.

$f1 = x_2 * x_4 * x_5 + x_2 * x_3 * x_4 + x_1 * \overline{x_5} + x_2 * x_4$
$f2 = \overline{x_2} * x_3 + x_2 * \overline{x_3}$
$f3 = x_3 * \overline{x_4} + x_2 * x_3 * x_4$

2. Представить в аналитической форме систему булевых функций для каждой неисправности.

1) Неисправность $x_1 = 0$
$f1 = x_2 * x_4 * x_5 + x_2 * x_3 * x_4 + x_2 * x_4$
$f2 = \overline{x_2} * x_3 + x_2 * \overline{x_3}$
$f3 = x_3 * \overline{x_4} + x_2 * x_3 * x_4$

2) Неисправность $x_1 = 1$
$f1 = x_2 * x_4 * x_5 + x_2 * x_3 * x_4 + \overline{x_5} + x_2 * x_4$
$f2 = \overline{x_2} * x_3 + x_2 * \overline{x_3}$
$f3 = x_3 * \overline{x_4} + x_2 * x_3 * x_4$

3) Неисправность $x_2 = 0$
$f1 = x_1 * \overline{x_5}$
$f2 = x_3$
$f3 = x_3 * \overline{x_4}$

4) Неисправность $x_2 = 1$
$f1 = x_4 * x_5 + x_3 * x_4 + x_1 * \overline{x_5} + x_4$
$f2 = \overline{x_3}$
$f3 = x_3 * \overline{x_4} + x_3 * x_4$

5) Неисправность $x_3 = 0$
$f1 = x_2 * x_4 * x_5 + x_1 * \overline{x_5} + x_2 * x_4$
$f2 = x_2$
$f3 = 0$

6) Неисправность $x_3 = 1$
$f1 = x_2 * x_4 * x_5 + x_2 * x_4 + x_1 * \overline{x_5} + x_2 * x_4$
$f2 = \overline{x_2}$
$f3 = \overline{x_4} + x_2 * x_4$

7) Неисправность $x_4 = 0$
$f1 = x_1 * \overline{x_5}$
$f2 = \overline{x_2} * x_3 + x_2 * \overline{x_3}$
$f3 = x_3$

8) Неисправность $x_4 = 1$
$f1 = x_2 * x_5 + x_2 * x_3 + x_1 * \overline{x_5} + x_2$
$f2 = \overline{x_2} * x_3 + x_2 * \overline{x_3}$
$f3 = x_2 * x_3$

9) Неисправность $x_5 = 0$
$f1 = x_2 * x_3 * x_4 + x_1 + x_2 * x_4$

$f2 = \overline{x_2} * x_3 + x_2 * \overline{x_3}$
$f3 = x_3 * \overline{x_4} + x_2 * x_3 * x_4$
10) Неисправность $x_5 = 1$
$f1 = x_2 * x_4 + x_2 * x_3 * x_4 + x_2 * x_4$
$f2 = \overline{x_2} * x_3 + x_2 * \overline{x_3}$
$f3 = x_3 * \overline{x_4} + x_2 * x_3 * x_4$

Задание 3: Представить в аналитической форме разностные функции для каждой неисправности.
F_i - функция, реализуемая исправной схемой (i - номер функции в системе)
f_i - функция, реализуемая схемой при наличии неисправности (i - номер неисправности)
R_j^i - разностная функция, где i - номер неисправности, j - номер функции в системе
Тогда аналитически разностные функции будут выглядеть:
1) Неисправность $x_1 = 0$
$R_1^1 = F_1 \oplus f_1$
$R_2^1 = F_2 \oplus f_1$
$R_3^1 = F_3 \oplus f_1$

2) Неисправность $x_1 = 1$
$R_1^2 = F_1 \oplus f_2$
$R_2^2 = F_2 \oplus f_2$
$R_3^2 = F_3 \oplus f_2$

3) Неисправность $x_2 = 0$
$R_1^3 = F_1 \oplus f_3$
$R_2^3 = F_2 \oplus f_3$
$R_3^3 = F_3 \oplus f_3$

4) Неисправность $x_2 = 1$
$R_1^4 = F_1 \oplus f_4$
$R_2^4 = F_2 \oplus f_4$
$R_3^4 = F_3 \oplus f_4$

5) Неисправность $x_3 = 0$
$R_1^5 = F_1 \oplus f_5$
$R_2^5 = F_2 \oplus f_5$
$R_3^5 = F_3 \oplus f_5$

6) Неисправность $x_3 = 1$
$R_1^6 = F_1 \oplus f_6$
$R_2^6 = F_2 \oplus f_6$
$R_3^6 = F_3 \oplus f_6$

7) Неисправность $x_4 = 0$
$R_1^7 = F_1 \oplus f_7$
$R_2^7 = F_2 \oplus f_7$
$R_3^7 = F_3 \oplus f_7$

8) Неисправность $x_4 = 1$
$R_1^8 = F_1 \oplus f_8$
$R_2^8 = F_2 \oplus f_8$

$R_3^8 = F_3 \oplus f_8$

9) Неисправность $x_5 = 0$
$R_1^9 = F_1 \oplus f_9$
$R_2^9 = F_2 \oplus f_9$
$R_3^9 = F_3 \oplus f_9$

10) Неисправность $x_5 = 1$
$R_1^{10} = F_1 \oplus f_{10}$
$R_2^{10} = F_2 \oplus f_{10}$
$R_3^{10} = F_3 \oplus f_{10}$

4. Написать программу, которая строит диагностическую матрицу на основе аналитического представления разностных функций.

```
#include <iostream>
#include <math.h>

using namespace std;
const int NotUsed = system("color F0");
int F1(int *x)
{
   return (x[1]&&x[3]&&x[4]||x[1]&&x[2]&&x[3]||x[0]&&!x[4]||x[1]&&x[3]);
}

int F2(int *x)
{
   return (!x[1]&&x[2]||x[1]&&!x[2]);
}

int F3(int *x)
{
   return (x[2]&&!x[3]||x[1]&&x[2]&&x[3]);
}

int F(int *x)
{
   int res = 0;
   res |= F1(x) << 2;
   res |= F2(x) << 1;
   res |= F3(x);
   return res;
}

int f10(int x1, int x2, int x3, int x4, int x5)
{
   int res = 0;
   res |= (x2&&x4&&x5||x2&&x3&&x4||x2&&x4) << 2;
   res |= (!x2&&x3||x2&&!x3) << 1;
   res |= (x3&&!x4||x2&&x3&&x4);
   return res;
}

int f11(int x1, int x2, int x3, int x4, int x5)
{
   int res = 0;
   res |= (x2&&x4&&x5||x2&&x3&&x4||!x5||x2&&x4) << 2;
   res |= (!x2&&x3||x2&&!x3) << 1;
```

```
   res |= (x3&&!x4||x2&&x3&&x4);
   return res;
}

int f20(int x1, int x2, int x3, int x4, int x5)
{
   int res = 0;
   res |= (x1&&!x5) << 2;
   res |= (x3) << 1;
   res |= (x3 && !x4);
   return res;
}

int f21(int x1, int x2, int x3, int x4, int x5)
{
   int res = 0;
   res |= (x4&&x5||x3&&x4||x1&&!x5||x4) << 2;
   res |= (!x3) << 1;
   res |= (x3&&!x4||x3&&x4);
   return res;
}

int f30(int x1, int x2, int x3, int x4, int x5)
{
   int res = 0;
   res |= (x2&&x4&&x5||x1&&!x5||x2&&x4) << 2;
   res |= (x2) << 1;
   res |= (0);
   return res;
}

int f31(int x1, int x2, int x3, int x4, int x5)
{
   int res = 0;
   res |= (x2&&x4&&x5||x2&&x4||x1&&!x5||x2&&x4) << 2;
   res |= (!x2) << 1;
   res |= (!x4||x2&&x4);
   return res;
}

int f40(int x1, int x2, int x3, int x4, int x5)
{
   int res = 0;
   res |= (x1&&!x5) << 2;
   res |= (!x2&&x3||x2&&!x3) << 1;
   res |= (x3);
   return res;
}

int f41(int x1, int x2, int x3, int x4, int x5)
{
   int res = 0;
   res |= (x2&&x5||x2&&x3||x1&&!x5||x2) << 2;
   res |= (!x2&&x3||x2&&!x3) << 1;
   res |= (x2&&x3);
   return res;
}

int f50(int x1, int x2, int x3, int x4, int x5)
{
   int res = 0;
   res |= (x2&&x3&&x4||x1||x2&&x4) << 2;
   res |= (!x2&&x3||x2&&!x3) << 1;
   res |= (x3&&!x4||x2&&x3&&x4);
```

```
    return res;
}

int f51(int x1, int x2, int x3, int x4, int x5)
{
    int res = 0;
    res |= (x2&&x4||x2&&x3&&x4||x2&&x4) << 2;
    res |= (!x2&&x3||x2&&!x3) << 1;
    res |= (x3&&!x4||x2&&x3&&x4);
    return res;
}


void truth_table(int m, int n, int **table) //функция для построения таблицы
истинности
{
    //заполнение table двоичными векторами
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            table[i][j] = ((i >> (m - 1 - j)) & 1);
        }
    }
}

int R(int F, int f) //F - мднф, f - функция неисправности
{
    if (F ^ f)
        return 1;
    else
        return 0;

}

void output(int m, int n, int **table)
{
    for (int i = 0; i < m; i++)
    {
        cout << "x" << i + 1 << "   ";
    }
    //cout << "F" <<  "     ";
    for (int i = 0; i < 10; i++)
    {
        cout << "R" << i + 1 << "     ";
    }
    cout << "\n";
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            cout << table[i][j] << "    ";
        }
        //cout << F(table[i]) << "      ";
        cout << R(F(table[i]), f10(table[i][0], table[i][1], table[i][2],
table[i][3], table[i][4])) << "      ";
        cout << R(F(table[i]), f11(table[i][0], table[i][1], table[i][2],
table[i][3], table[i][4])) << "      ";
        cout << R(F(table[i]), f20(table[i][0], table[i][1], table[i][2],
table[i][3], table[i][4])) << "      ";
        cout << R(F(table[i]), f21(table[i][0], table[i][1], table[i][2],
table[i][3], table[i][4])) << "      ";
        cout << R(F(table[i]), f30(table[i][0], table[i][1], table[i][2],
table[i][3], table[i][4])) << "      ";
```

```
      cout << R(F(table[i]), f31(table[i][0], table[i][1], table[i][2],
table[i][3], table[i][4])) << "     ";
      cout << R(F(table[i]), f40(table[i][0], table[i][1], table[i][2],
table[i][3], table[i][4])) << "     ";
      cout << R(F(table[i]), f41(table[i][0], table[i][1], table[i][2],
table[i][3], table[i][4])) << "     ";
      cout << R(F(table[i]), f50(table[i][0], table[i][1], table[i][2],
table[i][3], table[i][4])) << "     ";
      cout << R(F(table[i]), f51(table[i][0], table[i][1], table[i][2],
table[i][3], table[i][4])) << "     ";
      cout << endl;
   }
}

int main()
{
   int m = 5, n = pow(2, m);
   int **table = new int*[n];
   for (int i = 0; i < n; i++)
   {
      table[i] = new int[m];
   }
   truth_table(m, n, table);
   output(m, n, table);
   system("pause");
    return  0;
}
```

x1	x2	x3	x4	x5	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
0	0	0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	0	0	0	1	0	1	0	0	0	0
0	0	0	1	0	0	1	0	1	0	1	0	0	0	0
0	0	0	1	1	0	0	0	1	0	1	0	0	0	0
0	0	1	0	0	0	1	0	1	1	0	0	1	0	0
0	0	1	0	1	0	0	0	1	1	0	0	1	0	0
0	0	1	1	0	0	1	0	1	1	0	1	0	0	0
0	0	1	1	1	0	0	0	1	1	0	1	0	0	0
0	1	0	0	0	0	1	1	0	0	1	0	1	0	0
0	1	0	0	1	0	0	1	0	0	1	0	1	0	0
0	1	0	1	0	0	0	1	0	0	1	1	0	0	0
0	1	0	1	1	0	0	1	0	0	1	1	0	0	0
0	1	1	0	0	0	1	1	0	1	0	0	1	0	1
0	1	1	0	1	0	0	1	0	1	0	0	1	0	0
0	1	1	1	0	0	0	1	0	1	0	1	0	0	0
0	1	1	1	1	0	0	1	0	1	0	1	0	0	0
1	0	0	0	0	1	0	0	1	0	1	0	0	0	1
1	0	0	0	1	0	0	0	1	0	1	0	0	1	0
1	0	0	1	0	1	0	0	1	0	1	0	0	0	1
1	0	0	1	1	0	0	0	1	0	1	0	0	1	0
1	0	1	0	0	1	0	0	1	1	0	0	1	0	1
1	0	1	0	1	0	0	0	1	1	0	0	1	1	0
1	0	1	1	0	1	0	0	1	1	0	1	0	0	1
1	0	1	1	1	0	0	0	1	1	0	1	0	1	0
1	1	0	0	0	1	0	1	0	0	1	0	0	0	1
1	1	0	0	1	0	0	1	0	0	1	0	1	1	0
1	1	0	1	0	0	0	1	0	0	1	0	0	0	0
1	1	0	1	1	0	0	1	0	0	1	1	0	0	0
1	1	1	0	0	1	0	1	0	1	0	0	0	0	1
1	1	1	0	1	0	0	1	0	1	0	0	1	1	0
1	1	1	1	0	0	0	1	0	1	0	0	0	0	0
1	1	1	1	1	0	0	1	0	1	0	1	0	0	0

5.По диагностической матрице найти минимальный диагностический тест.

Номер набора	Разностные функции									
	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
x^0	0	1	0	1	0	1	0	0	0	0
x^1	0	0	0	1	0	1	0	0	0	0
x^2	0	1	0	1	0	1	0	0	0	0
x^3	0	0	0	1	0	1	0	0	0	0
x^4	0	1	0	1	1	0	0	1	0	0
x^5	0	0	0	1	1	0	0	1	0	0
x^6	0	1	0	1	1	0	1	0	0	0
x^7	0	0	1	1	1	0	1	0	0	0
x^8	0	1	1	0	0	1	0	1	0	0
x^9	0	0	1	0	0	1	0	1	0	0
x^{10}	0	0	1	0	0	1	1	0	0	0
x^{11}	0	0	1	0	0	1	1	0	0	0
x^{12}	0	1	1	0	1	0	0	1	0	1
x^{13}	0	0	1	0	1	0	0	1	0	0
x^{14}	0	0	1	0	1	0	1	0	0	0
x^{15}	0	0	1	0	1	0	1	0	0	0
x^{16}	1	0	0	1	0	1	0	0	0	1
x^{17}	0	0	0	1	0	1	0	0	1	0
x^{18}	1	0	0	1	0	1	0	0	0	1
x^{19}	0	0	0	1	0	1	0	0	1	0
x^{20}	1	0	0	1	1	0	0	1	0	1
x^{21}	0	0	0	1	1	0	0	1	1	0
x^{22}	1	0	0	1	1	0	1	0	0	1
x^{23}	0	0	0	1	1	0	1	0	1	0
x^{24}	1	0	1	0	0	1	0	0	0	1
x^{25}	0	0	1	0	0	1	0	1	1	0
x^{26}	0	0	1	0	0	1	0	0	0	0
x^{27}	0	0	1	0	0	1	1	0	0	0
x^{28}	1	0	1	0	1	0	0	0	0	1
x^{29}	0	0	1	0	1	0	0	1	1	0
x^{30}	0	0	1	0	1	0	0	0	0	0
x^{31}	0	0	1	0	1	0	1	0	0	0

Минимальный проверяющий тест: $\{x^4, x^{23}, x^{24}\}$
Минимальный диагностический тест: $\{x^4, x^{12}, x^{23}, x^{24}, x^{25}\}$

6. Написать программу, которая по аналитическому представлению систем булевых функций неисправностей строит таблицу их значений на тестовых наборах. Для компактного представления таблицы рекомендуется значение системы булевых функций неисправности представлять не двоичным вектором, а десятичным числом.

```
#include <iostream>
#include <math.h>

using namespace std;
const int NotUsed = system("color F0");
int F1(int *x)
{
```

```
    return (x[1]&&x[3]&&x[4]||x[1]&&x[2]&&x[3]||x[0]&&!x[4]||x[1]&&x[3]);
}

int F2(int *x)
{
    return (!x[1]&&x[2]||x[1]&&!x[2]);
}

int F3(int *x)
{
    return (x[2]&&!x[3]||x[1]&&x[2]&&x[3]);
}

int F(int *x)
{
    int res = 0;
    res |= F1(x) << 2;
    res |= F2(x) << 1;
    res |= F3(x);
    return res;
}

int f10(int x1, int x2, int x3, int x4, int x5)
{
    int res = 0;
    res |= (x2&&x4&&x5||x2&&x3&&x4||x2&&x4) << 2;
    res |= (!x2&&x3||x2&&!x3) << 1;
    res |= (x3&&!x4||x2&&x3&&x4);
    return res;
}

int f11(int x1, int x2, int x3, int x4, int x5)
{
    int res = 0;
    res |= (x2&&x4&&x5||x2&&x3&&x4||!x5||x2&&x4) << 2;
    res |= (!x2&&x3||x2&&!x3) << 1;
    res |= (x3&&!x4||x2&&x3&&x4);
    return res;
}

int f20(int x1, int x2, int x3, int x4, int x5)
{
    int res = 0;
    res |= (x1&&!x5) << 2;
    res |= (x3) << 1;
    res |= (x3 && !x4);
    return res;
}

int f21(int x1, int x2, int x3, int x4, int x5)
{
    int res = 0;
    res |= (x4&&x5||x3&&x4||x1&&!x5||x4) << 2;
    res |= (!x3) << 1;
    res |= (x3&&!x4||x3&&x4);
    return res;
}

int f30(int x1, int x2, int x3, int x4, int x5)
{
    int res = 0;
    res |= (x2&&x4&&x5||x1&&!x5||x2&&x4) << 2;
    res |= (x2) << 1;
    res |= (0);
```

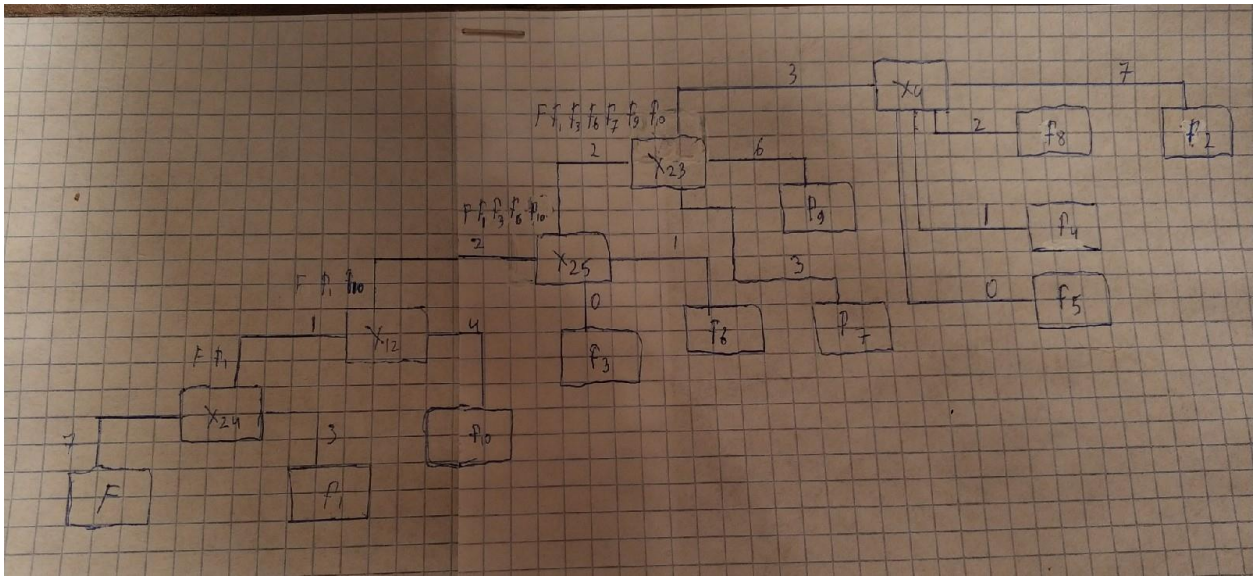
```
    return res;
}

int f31(int x1, int x2, int x3, int x4, int x5)
{
    int res = 0;
    res |= (x2&&x4&&x5||x2&&x4||x1&&!x5||x2&&x4) << 2;
    res |= (!x2) << 1;
    res |= (!x4||x2&&x4);
    return res;
}

int f40(int x1, int x2, int x3, int x4, int x5)
{
    int res = 0;
    res |= (x1&&!x5) << 2;
    res |= (!x2&&x3||x2&&!x3) << 1;
    res |= (x3);
    return res;
}

int f41(int x1, int x2, int x3, int x4, int x5)
{
    int res = 0;
    res |= (x2&&x5||x2&&x3||x1&&!x5||x2) << 2;
    res |= (!x2&&x3||x2&&!x3) << 1;
    res |= (x2&&x3);
    return res;
}

int f50(int x1, int x2, int x3, int x4, int x5)
{
    int res = 0;
    res |= (x2&&x3&&x4||x1||x2&&x4) << 2;
    res |= (!x2&&x3||x2&&!x3) << 1;
    res |= (x3&&!x4||x2&&x3&&x4);
    return res;
}

int f51(int x1, int x2, int x3, int x4, int x5)
{
    int res = 0;
    res |= (x2&&x4||x2&&x3&&x4||x2&&x4) << 2;
    res |= (!x2&&x3||x2&&!x3) << 1;
    res |= (x3&&!x4||x2&&x3&&x4);
    return res;
}


void truth_table(int m, int n, int **table) //функция для построения таблицы
истинности
{
    //заполнение table двоичными векторами
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            table[i][j] = ((i >> (m - 1 - j)) & 1);
        }
    }
}

int R(int F, int f) //F - мднф, f - функция неисправности
{
```

```
    if (F ^ f)
        return 1;
    else
        return 0;

}

void output(int m, int n, int **table)
{
    for (int i = 0; i < m; i++)
    {
        cout << "x" << i + 1 << "  ";
    }
    cout << "F" <<  "    ";
    for (int i = 0; i < 10; i++)
    {
        cout << "f" << i + 1 << "    ";
    }
    cout << "\n";
    int k = 0;
    for (int i = 0; i < n; i++)
    {
        //if (k == 0 || k == 2 || k == 5 || k == 13 || k == 17 || k == 27 ||
k == 28)
        //{
        for (int j = 0; j < m; j++)
        {
            cout << table[i][j] << "   ";
        }
        cout << F(table[i]) << "     ";
        cout << f10(table[i][0], table[i][1], table[i][2], table[i][3],
table[i][4]) << "     ";
        cout << f11(table[i][0], table[i][1], table[i][2], table[i][3],
table[i][4]) << "     ";
        cout << f20(table[i][0], table[i][1], table[i][2], table[i][3],
table[i][4]) << "     ";
        cout << f21(table[i][0], table[i][1], table[i][2], table[i][3],
table[i][4]) << "     ";
        cout << f30(table[i][0], table[i][1], table[i][2], table[i][3],
table[i][4]) << "     ";
        cout << f31(table[i][0], table[i][1], table[i][2], table[i][3],
table[i][4]) << "     ";
        cout << f40(table[i][0], table[i][1], table[i][2], table[i][3],
table[i][4]) << "     ";
        cout << f41(table[i][0], table[i][1], table[i][2], table[i][3],
table[i][4]) << "     ";
        cout << f50(table[i][0], table[i][1], table[i][2], table[i][3],
table[i][4]) << "     ";
        cout << f51(table[i][0], table[i][1], table[i][2], table[i][3],
table[i][4]) << "     ";
        cout << endl;


        //}
        //k++;
    }
}

int main()
{
    int m = 5, n = pow(2, m);
    int **table = new int*[n];
    for (int i = 0; i < n; i++)
    {
```

```
            table[i] = new int[m];
        }
        truth_table(m, n, table);
        output(m, n, table);
        system("pause");
        return 0;
}
```

x1	x2	x3	x4	x5	F	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10
0	0	0	0	0	0	0	4	0	2	0	3	0	0	0	0
0	0	0	0	1	0	0	0	0	2	0	3	0	0	0	0
0	0	0	1	0	0	0	4	0	6	0	2	0	0	0	0
0	0	0	1	1	0	0	0	0	6	0	2	0	0	0	0
0	0	1	0	0	3	3	7	3	1	0	3	3	2	3	3
0	0	1	0	1	3	3	3	3	1	0	3	3	2	3	3
0	0	1	1	0	2	2	6	2	5	0	2	3	2	2	2
0	0	1	1	1	2	2	2	2	5	0	2	3	2	2	2
0	1	0	0	0	2	2	6	0	2	2	1	2	6	2	2
0	1	0	0	1	2	2	2	0	2	2	1	2	6	2	2
0	1	0	1	0	6	6	6	0	6	6	5	2	6	6	6
0	1	0	1	1	6	6	6	0	6	6	5	2	6	6	6
0	1	1	0	0	1	1	5	3	1	2	1	1	5	1	4
0	1	1	0	1	1	1	1	3	1	2	1	1	5	1	1
0	1	1	1	0	5	5	5	2	5	6	5	1	5	5	5
0	1	1	1	1	5	5	5	2	5	6	5	1	5	5	5
1	0	0	0	0	4	0	4	4	6	4	7	4	4	4	0
1	0	0	0	1	0	0	0	0	2	0	3	0	0	4	0
1	0	0	1	0	4	0	4	4	6	4	6	4	4	4	0
1	0	0	1	1	0	0	0	0	6	0	2	0	0	4	0
1	0	1	0	0	7	3	7	7	5	4	7	7	6	7	3
1	0	1	0	1	3	3	3	3	1	0	3	3	2	7	3
1	0	1	1	0	6	2	6	6	5	4	6	7	6	6	2
1	0	1	1	1	2	2	2	2	5	0	2	3	2	6	2
1	1	0	0	0	7	3	6	4	6	1	5	6	6	6	2
1	1	0	0	1	2	2	2	0	2	2	1	2	6	6	2
1	1	0	1	0	6	6	6	4	6	6	5	6	6	6	6
1	1	0	1	1	6	6	6	0	6	6	5	2	6	6	6
1	1	1	0	0	5	1	5	7	5	6	5	5	5	5	1
1	1	1	0	1	1	1	1	3	1	2	1	1	5	5	1
1	1	1	1	0	5	5	5	6	5	6	5	5	5	5	5
1	1	1	1	1	5	5	5	2	5	6	5	1	5	5	5

7. По полученной в п. 6 таблице построить алгоритм распознавания неисправностей в виде диагностического дерева.



8. Определить неисправность, которая распознается дольше других, написать программу, которая моделирует схему с этой неисправностью, провести диагностический эксперимент.

Неисправность f1 (x1 = 0) распознается дольше других.

```
int f1(int *x)
{
    bool Value = (x[1]&&x[3]&&x[4]||x[1]&&x[2]&&x[3]||x[1]&&x[3]);
    return Value;
}
int f2(int *x)
{
    bool Value = (!x[1]&&x[2]||x[1]&&!x[2]);
    return Value;
}
int f3(int *x)
{
    bool Value = (x[2]&&!x[3]||x[1]&&x[2]&&x[3]);
    return Value;
}
```

f1	f2	f3
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	1	1
0	1	1
0	1	1
0	1	1
0	1	0
0	1	0
0	1	0
0	1	0
0	1	0
0	1	0
0	1	0
0	1	0
1	1	0
1	1	0
1	1	0
1	1	0
0	0	1
0	0	1
0	0	1
0	0	1
1	0	1
0	0	1
1	0	1
1	0	1
1	0	1
1	0	1

error f1 (x1 = 0)