МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Вычислительная техника»

Дисциплина «Автоматно-лингвистическое моделирование»

**Пояснительная записка**

**к курсовой работе по теме**

**«Программная реализация микропрограммного автомата с жесткой логикой: объединенная микропрограмма операций умножения и деления»**

Вариант 4

Выполнил:

студент группы ИВТАПбд-21

Вершинин Д. В.

Проверила:

ст. п. Лылова А. В.

Ульяновск, 2018

Оглавление

[Введение 3](#_Toc515488047)

[1. Техническое задание 4](#_Toc515488048)

[2. Анализ предметной области 5](#_Toc515488049)

[3. Анализ исходного описания реализуемого микропрограммного автомата с жесткой логикой 8](#_Toc515488050)

[4. Проектирование микропрограммного автомата с жесткой логикой 10](#_Toc515488051)

[4.1. Составление схемы алгоритма 10](#_Toc515488052)

[4.2. Построение отмеченной ГСА, заданного микропрограммного автомата с жесткой логикой 13](#_Toc515488053)

[4.3 Построение графа на основе полученной ГСА 15](#_Toc515488054)

[4.4. Проведение канонического метода структурного синтеза 15](#_Toc515488055)

[5. Разработка формата протокола входных и выходных данных 19](#_Toc515488056)

[для разрабатываемого интерпретатора 19](#_Toc515488057)

[микропрограммного автомата с жесткой логикой 19](#_Toc515488058)

[6. Разработка интерпретатора микропрограммного автомата с жесткой логикой 21](#_Toc515488059)

[6.1. Разработка для способа задания – логическое выражение 21](#_Toc515488060)

[6.2. Разработка для способа задания – граф-схема 22](#_Toc515488061)

[7. Разработка рабочей нагрузки и тестирование интерпретатора 23](#_Toc515488062)

[8. Сравнительный анализ данных полученных протоколов 25](#_Toc515488063)

[Заключение 26](#_Toc515488064)

[Используемые ресурсы 27](#_Toc515488065)

[Приложения 28](#_Toc515488066)

# Введение

Целью дисциплины «Автоматно-лингвистическое моделирование» является изучение базовых понятий и принципов построения формальных грамматик и различных моделей автоматов. Любое цифровое устройство можно рассматривать как устройство, состоящее из двух частей: операционного и управляющего блоков. Операционный блок, например АЛУ, характеризуется совокупностью определённых в нём микроопераций, каждая из которых представляет собой некоторый выполняемый в данном операционном блоке акт передачи или преобразования информации. Часть цифрового вычислительного устройства, предназначенного для выработки последовательности управляющих функциональных сигналов, называется управляющим блоком или управляющим устройством.

Проектирование и реализация микропрограммых автоматов направлена на модернизацию информационных систем. Для повышения эффективности ЭВМ необходимо использование программных реализаций наиболее производительных и простых, создание которых невозможно без использования методов проектирования.

Целью работы является изучение методов проектирования и программной реализации микропрограммного автомата с жесткой логикой, реализующего объединенную микропропрограмму операций умножения и деления.

# 1. Техническое задание

Требуется проанализировать исходные данные, на основе которых необходимо создать программную реализацию микропрограммного автомата с жесткой логикой на языке программирования высокого уровня С++.

На этапе проектирования необходимо провести анализ схемы алгоритма, на основе которого строится отмеченная ГСА по типу автомата Мура. Далее требуется построить граф переходов на основе полученной отмеченной ГСА. На следующем этапе требуется провести канонический метод структурного синтеза, заключающийся в кодировании входных и выходных сигналов и состояний, построении семиколоночной таблицы и логических функций переходов и выходов на её основе.

* Состояния автомата должны иметь имена в формате Si, где i = 0..n (S0 – начальное и конечное состояния автомата);
* Условия в автомате (условные вершины в графе, входные сигналы автомата) кодируются как Xj, где j = 0..m;
* Действия в автомате (операторные вершины в графе, выходные сигналы автомата) кодируются как Yl, где l = 0..k;
* Функция возбуждения (функции переходов) должны иметь имена формата wz (переход с 0 на 1) и uz (переход с 1 на 0), где z =0..v.

Реализованный интерпретатор микропрограммного автомата с жесткой логикой должен иметь строго заданный формат протокола входных и выходных данных. Программа должна выдавать ответ на любые входные данные: результат, если они корректны, сообщение об ошибке, если нет. Программная реализация должна иметь методы для двух способов задания: «логическое выражение» и «граф-схема». После вывода результата программа должна выводить информацию о времени, затраченном на выполнение в зависимости от выбора способа задания.

# 

# 2. Анализ предметной области

В данной курсовой работе рассматривается построение управляющего устройства с жесткой логикой, его виды, способы задания, разработка функций переходов и выходов, а также работа с ними.

В микропрограммном автомате с жесткой логикой, выходные сигналы управления реализуются за счет однажды соединенных логических схем. В отличие от управляющих устройств с хранимой в памяти логикой у этих автоматов можно изменить логику работы только путем пере­делок схем автомата. [3]

В управляющих автоматах на элементах с жесткой логикой - для хранения информации о состоянии используется набор триггеров, на их тактовый вход подается тактирующий сигнал, входные сигналы X подаются на комбинационные устройства, вырабатывающие сигналы управления для (функции возбуждения) триггеров. Выходные сигналы Y формируются при помощи других комбинационных схем.[4]

Работу операционного блока можно описать микропро­граммой, например, на языке микроопераций или в виде графа. По микропрограмме строится соответствующий управляющий автомат типа Мура или Мили. Причем для выходных данных в автомате Мили (Рис.2.1) необходимы входные данные, в то время как в автомате Мура (Рис.2.2) выходные данные есть всегда.

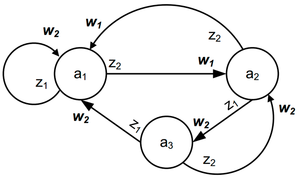


Рис.2.1. Граф автомата Мили

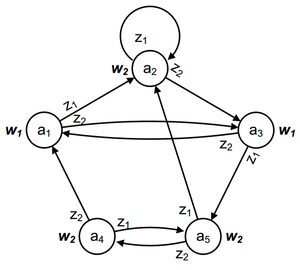


Рис.2.2. Граф автомата Мура

При синтезе управляющего автомата с жесткой логикой выделяются этапы абстрактного и структурного синтеза.

На этапе абстрактного синтеза по алгоритму, заданному на начальном языке строится таблица переходов, записываются системы канонических уравнений и системы выходных функций.

На этапе структурного синтеза строится логическая схема управляющего автомата.

Функция управляющего автомата задаётся кодированной граф-схемой алгоритма микропрограммы. Кодированную ГСА получают путём замены в содержательной ГСА микрооператоров (наборов совместимых микроопераций) на коды микрокоманд, а логических условий на их идентификаторы.

Абстрактный синтез управляющего автомата начинается с отметки внутренних состояний кодированной ГСА. Отметка состояний должна соответствовать закону функционирова­ния автомата Мура или Мили, то есть выполняется для них различным образом. [1]

Для програмной реализации используется два способа задания: логические выражения и граф-схема. Логическое выражение в программировании — конструкция языка программирования, результатом вычисления которой является «истина» или «ложь». Граф-схема алгоритма (ГСА) — конечный связный ориентированный граф, вершины которого соответствуют операторам, а дуги. задают порядок следования вершин (операторов) алгоритма.

# 3. Анализ исходного описания реализуемого микропрограммного автомата с жесткой логикой

Для программной реализации дана уже готовая блок-схема микропрограммного автомата, реализующая объединенную микропрограмму операций умножения и деления. Ниже представлен рисунок данной блок - схемы (см. Рис. 3.1.).

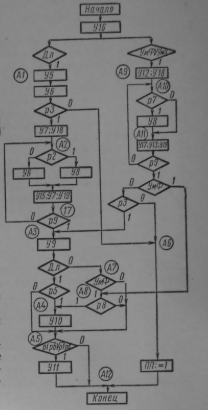


Рис. 3.1. Блок – схема объединенной микропрограммы операций умножения и деления.

Закон функционирования автомата для управления операциями деления и умножения будет определяться на основе объединенной микропрограммы, включающей в себя микропрограмму деления и микропрограммы умножения целых чисел и чисел с фиксированной запятой. Пусть наименование выполняемой операции определяется переменными Дл, Ум и УмФ, которые принимают значения 1 только в том случае, когда выполняется операция деления, умножения целых чисел и умножения целых чисел с фиксированной запятой соответственно. [2]

# 4. Проектирование микропрограммного автомата с жесткой логикой

## 4.1. Составление схемы алгоритма

В исходном описании автомата количество использующихся значений входных (условий) и выходных состояний не соответствует действительному. В схеме пропущены коды некоторых входных и выходных сигналов. Поэтому было принято решение о перекодировании значений X-ов и Y-ов. Новые коды входных и выходных сигналов представлены ниже.

|  |  |
| --- | --- |
| X | Входные сигналы исходной блок-схемы |
| X1 | Дл |
| X2 | P3 |
| X3 | P2 |
| X4 | P9 |
| X5 | P5 |
| X6 | УмФ |
| X7 | P8 |
| X8 | P1!P6V!P1P6 |
| X9 | P7 |

Таблица 4.1. Закодированные входные сигналы

|  |  |
| --- | --- |
| Y | Выходные сигналы исходной блок-схемы |
| Y1 | Y16 |
| Y2 | Y5 |
| Y3 | Y6 |
| Y4 | Y7 |
| Y5 | Y18 |
| Y6 | Y8 |
| Y7 | Y15 |
| Y8 | Y19 |
| Y9 | Y9 |
| Y10 | Y10 |
| Y11 | Y11 |
| Y12 | Y12 |
| Y13 | Y17 |
| Y14 | Y13 |
| Y15 | ПП: = 1 |

Таблица 4.2. «Закодированные выходные сигналы»

Используя новые значения входных и выходных сигналов исходная блок-схема перерисовывается (рис. 4.1).

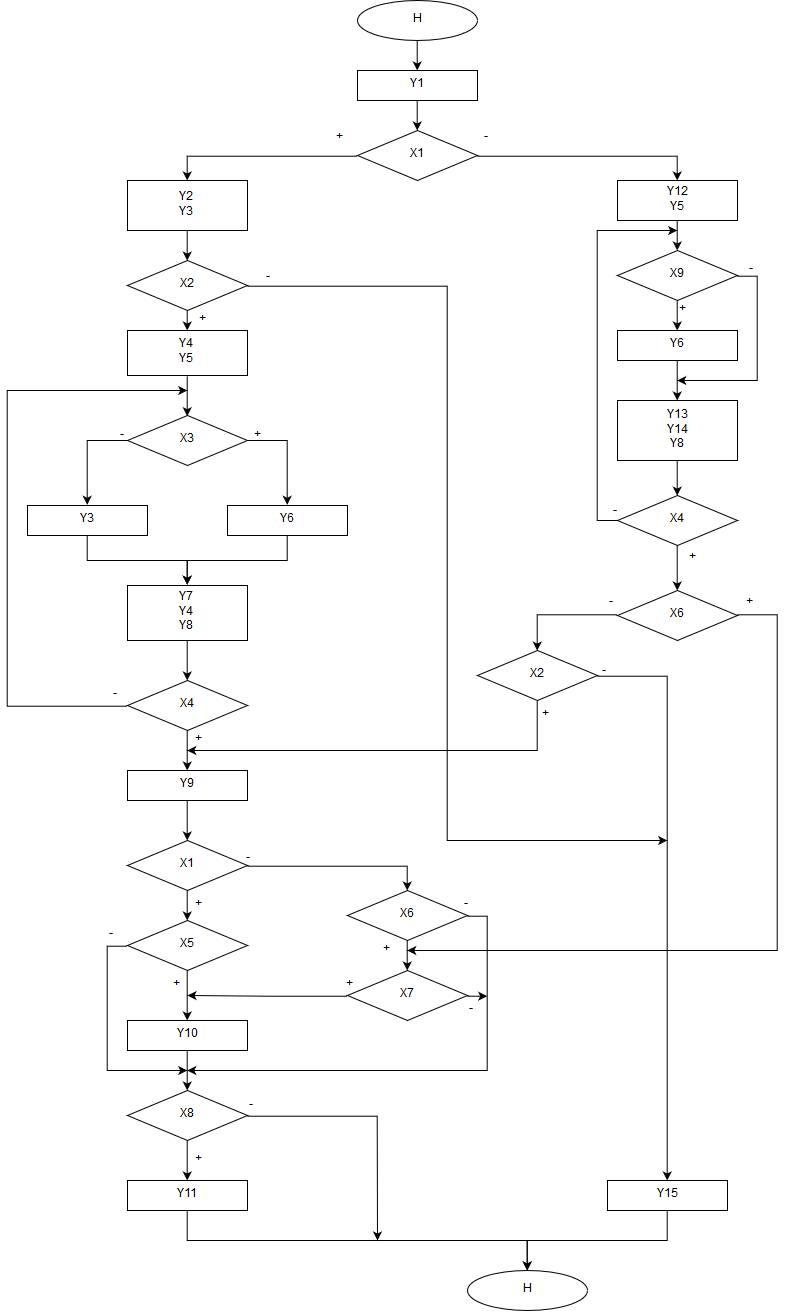


Рис.4.1. Схема алгоритма

## 4.2. Построение отмеченной ГСА, заданного микропрограммного автомата с жесткой логикой

В соответствии с видом автомата, заданным по варианту, расставим состояния S, чтобы получить отмеченную ГСА (см. Рис. 4.2). Для автомата Мура состояния проставляются напротив каждой операторной вершины.

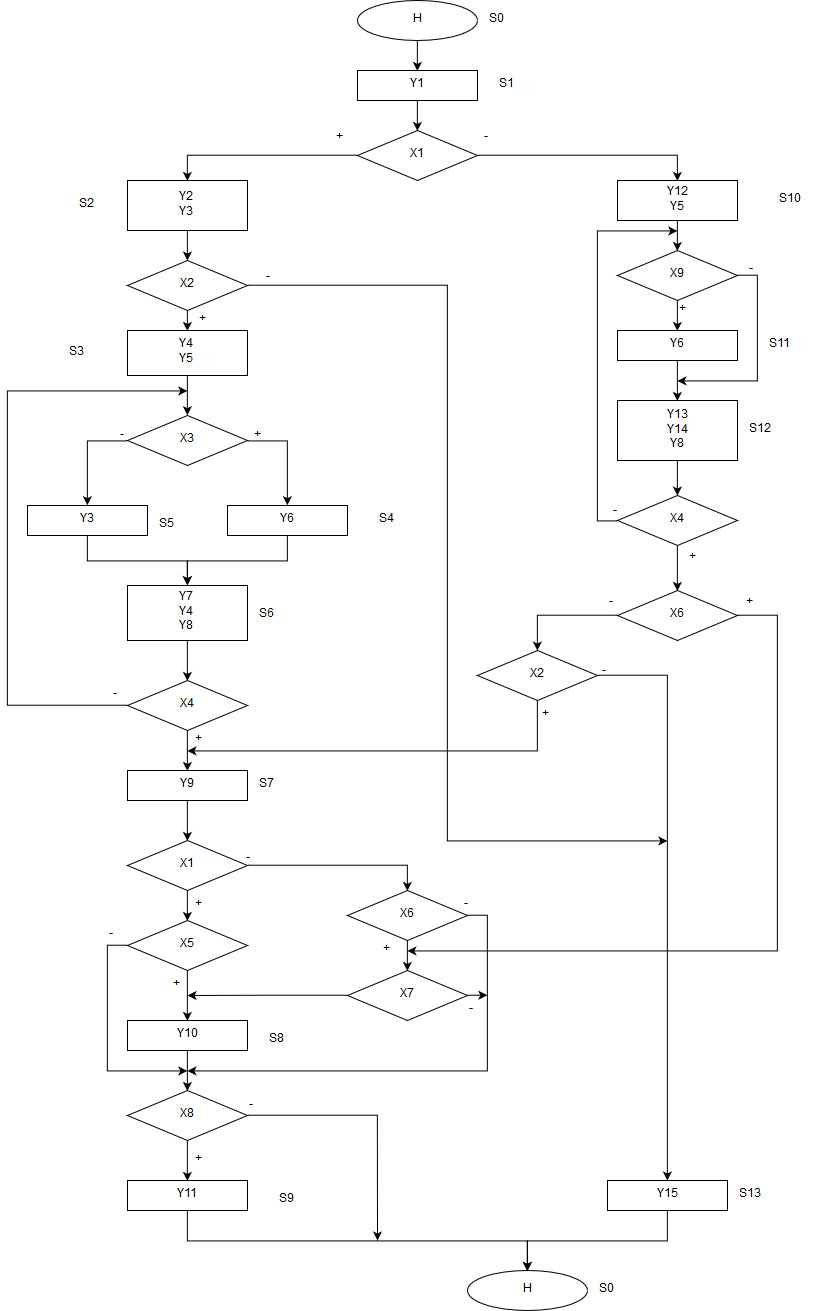


Рис.4.2. Блок-схема объединенной микропрограммы операций умножения и деления.

## 4.3 Построение графа на основе полученной ГСА

Граф переходов (Рис.4.3) строится на основе полученной отмеченной ГСА и имеет столько вершин, сколько различных состояний имеет отмеченная ГСА.

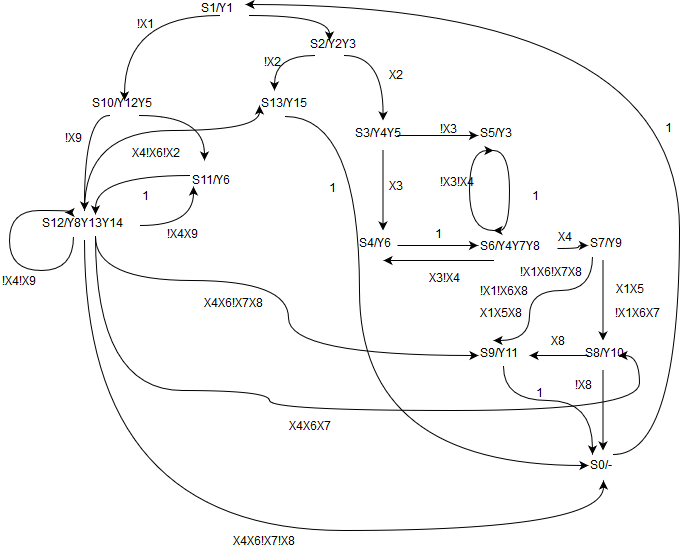


Рис.4.3. Граф переходов на основе полученной отмеченной ГСА

## 4.4. Проведение канонического метода структурного синтеза

На первом этапе канонического метода структурного синтеза необходимо произвести кодирование входных и выходных сигналов, а также состояний (S). Так как после построение схемы получилось 14 состояний (S0…S13), то для кодирования потребуется 4 бита (α1 α2 α3 α4), это следует из формулы:

|  |  |
| --- | --- |
| Si | Код состояния |
| S0 | 0000 |
| S1 | 0001 |
| S2 | 0010 |
| S3 | 0011 |
| S4 | 0100 |
| S5 | 0101 |
| S6 | 0110 |
| S7 | 0111 |
| S8 | 1000 |
| S9 | 1001 |
| S10 | 1010 |
| S11 | 1011 |
| S12 | 1100 |
| S13 | 1101 |

Таблица 4.4.1. Таблица кодов состояний

На втором этапе строится семиколоночная таблица, на основе графа автомата, (Таблица 4.4.2), в которой указываются все возможные переходы между состояниями при различных входных сигналах, а также выходные сигналы при переходе и функции возбуждения wz – переход из 0 в 1, uz – переход из 1 в 0.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Нач. сост. | Код н. с. | Кон. Сост. | Код кон. сост. | Вх. с. | Вых. С. | Ф-ция возб. |
| S0 | 0000 | S1 | 0001 | 1 | - | W4 |
| S1 | 0001 | S2 | 0010 | x1 | y1 | W3 U4 |
| S1 | 0001 | S10 | 1010 | !x1 | y1 | W1 W3 U4 |
| S2 | 0010 | S3 | 0011 | x2 | y2 y3 | W4 |
| S2 | 0010 | S13 | 1101 | !x2 | y2 y3 | W1 W2 W4 U3 |
| S3 | 0011 | S4 | 0100 | x3 | y4 y5 | W2 U3 U4 |
| S3 | 0011 | S5 | 0101 | !x3 | y4 y5 | W2 U3 |
| S4 | 0100 | S6 | 0110 | 1 | y6 | W3 |
| S5 | 0101 | S6 | 0110 | 1 | y3 | W3 U4 |
| S6 | 0110 | S4 | 0100 | !x4 x3 | y7 y4 y8 | U3 |
| S6 | 0110 | S5 | 0101 | !x4 !x3 | y7 y4 y8 | W4 U3 |
| S6 | 0110 | S7 | 0111 | x4 | y7 y4 y8 | W4 |
| S7 | 0111 | S8 | 1000 | x1 x5 | y9 | W1 U2 U3 U4 |
| S7 | 0111 | S8 | 1000 | !x1 x6 x7 | y9 | W1 U2 U3 U4 |
| S7 | 0111 | S9 | 1001 | x1 !x5 x8 | y9 | W1 U2 U3 |
| S7 | 0111 | S9 | 1001 | !x1 !x6 x8 | y9 | W1 U2 U3 |
| S7 | 0111 | S9 | 1001 | !x1 x6 !x7 x8 | y9 | W1 U2 U3 |
| S7 | 0111 | S0 | 0000 | x1 !x5 !x8 | y9 | U2 U3 U4 |
| Нач. сост. | Код н. с. | Кон. Сост. | Код кон. сост. | Вх. с. | Вых. С. | Ф-ция возб. |
| S7 | 0111 | S0 | 0000 | !x1 !x6 !x8 | y9 | U2 U3 U4 |
| S7 | 0111 | S0 | 0000 | !x1 x6 !x7 !x8 | y9 | U2 U3 U4 |
| S8 | 1000 | S9 | 1001 | x8 | y10 | W4 |
| S8 | 1000 | S0 | 0000 | !x8 | y10 | U1 |
| S9 | 1001 | S0 | 0000 | 1 | y11 | U1 U4 |
| S10 | 1010 | S11 | 1011 | x9 | y12 y5 | W4 |
| S10 | 1010 | S12 | 1100 | !x9 | y12 y5 | W2 U3 |
| S11 | 1011 | S12 | 1100 | 1 | y6 | W2 U3 U4 |
| S12 | 1100 | S0 | 0000 | x4 x6 !x7 !x8 | y13 y14 y8 | U1 U2 |
| S12 | 1100 | S7 | 0111 | x4 !x6 x2 | y13 y14 y8 | W3 W4 U1 |
| S12 | 1100 | S8 | 1000 | x4 x6 x7 | y13 y14 y8 | U2 |
| S12 | 1100 | S9 | 1001 | x4 x6 !x7 x8 | y13 y14 y8 | W4 U2 |
| S12 | 1100 | S11 | 1011 | !x4 x9 | y13 y14 y8 | W3 W4 U2 |
| S12 | 1100 | S12 | 1100 | !x4 !x9 | y13 y14 y8 | - |
| S12 | 1100 | S13 | 1101 | x4 !x6 !x2 | y13 y14 y8 | W4 |
| S13 | 1101 | S0 | 0000 | 1 | y15 | U1 U2 U4 |

Таблица 4.4.2. Семиколоночная таблица

На третьем этапе строятся логические функции (выражения) переходов выходов. Для построения функций выходов необходимо учитывать тип автомата.

Для автомата Мура при построении функции выхода учитывается только код начального состояния (в отличии от автомата Мили).

Y1 =

Y2 =

Y3 = V

Y4 = V

Y5 = V

Y6 = V

Y7 =

Y8 = V

Y9 =

Y10 =

Y11 =

Y12 =

Y13 =

Y14 =

Y15 =

Функции возбуждения зависят от начального состояния и входного сигнала всегда, независимо от типа автомата.

*W1 = a̅1̅a̅2̅a̅3̅a4x̅1̅ V a̅1̅a̅2̅a3a̅4̅x̅2̅ V a̅1̅a2a3a4x1x5 V a̅1̅a2a3a4x̅1̅x6x7 V a̅1̅a2a3a4x1x5x8 V a̅1̅a2a3a4x̅1̅x̅6̅x8 V a̅1̅a2a3a4x̅1̅x6x̅7̅x8*

*W2 = a̅1̅a̅2̅a3a̅4̅x̅2̅ V a̅1̅a̅2̅a3a4x3 V a̅1̅a̅2̅a3a4x̅3̅ V a1a̅2̅a3a̅4̅x̅9̅ V a1a̅2̅a3a4*

*W3 = a̅1̅a̅2̅a̅3̅a4x1 V a̅1̅a̅2̅a̅3̅a4x̅1̅ V a̅1̅a2a̅3̅a̅4̅ V a̅1̅a2a̅3̅a4 V a1a2a̅3̅a̅4̅x4x̅6̅x2 V a1a2a̅3̅a̅4̅x̅4̅x9*

*W4 = a̅1̅a̅2̅a̅3̅a̅4̅ V a̅1̅a̅2̅a3a̅4̅x2 V a̅1̅a̅2̅a3a̅4̅x̅2̅ V a̅1̅a2a3a̅4̅x̅4̅x̅3̅ V a̅1̅a2a3a̅4̅x4 V a1a̅2̅a̅3̅a̅4̅x8 V a1a̅2̅a3a̅4̅x9 V a1a2a̅3̅a̅4̅x4x̅6̅x2 V a1a2a̅3̅a̅4̅x4x6x̅7̅x8 V a1a2a̅3̅a̅4̅x̅4̅x9 V a1a2a̅3̅a̅4̅x4x̅6̅x̅2̅*

*U1= a1a̅2a̅3a̅4x̅8 V a1a̅2a̅3a4 V a1a2a̅3a̅4x4x6x̅7x̅8 V a1a2a̅3a̅4x4x̅6x2 V a1a2a̅3a4*

*U2= a̅1a2a3a4x1x5 V a̅1a2a3a4x̅1x6x7 V a̅1a2a3a4x1x5x8 V a̅1a2a3a4x̅1x̅6x8 V a̅1a2a3a4x̅1x6x̅7x8 V a̅1a2a3a4x1x̅5x̅8 V a̅1a2a3a4x̅1x̅6x̅8 V a̅1a2a3a4x̅1x6x̅7x̅8 V a1a2a̅3a̅4x4x6x̅7x̅8 V a1a2a̅3a̅4x4x6x7 V a1a2a̅3a̅4x4x6x̅7x8 V a1a2a̅3a̅4x̅4x9 V a1a2a̅3a4*

*U3= a̅1a̅2a3a̅4x̅2 V a̅1a̅2a3a4x3 V a̅1a̅2a3a4x̅3 V a̅1a2a3a̅4x̅4x3 V a̅1a2a3a̅4x̅4x̅3 V a̅1a2a3a4x1x5 V a̅1a2a3a4x̅1x6x7 V a̅1a2a3a4x1x5x8 V a̅1a2a3a4x̅1x̅6x8 V a̅1a2a3a4x̅1x6x̅7x8 V a̅1a2a3a4x1x̅5x̅8 V a̅1a2a3a4x̅1x̅6x̅8 V a̅1a2a3a4x̅1x6x̅7x̅8 V a1a̅2a3a̅4x̅9 V a1a̅2a3a4*

*U4= a̅1a̅2a̅3a4x1 V a̅1a̅2a̅3a4x̅1 V a̅1a̅2a3a4x3 V a̅1a2a̅3a4 V a̅1a2a3a4x1x5 V a̅1a2a3a4x̅1x6x7 V a̅1a2a3a4x1x̅5x̅8 V a̅1a2a3a4x̅1x̅6x̅8 V a̅1a2a3a4x̅1x6x̅7x̅8 V a1a̅2a̅3a4 V a1a̅2a3a4 V a1a2a̅3a4*

# 5. Разработка формата протокола входных и выходных данных

# для разрабатываемого интерпретатора

# микропрограммного автомата с жесткой логикой

Протокол входных и выходных данных необходим для корректной работы программы и оценки результатов её работы. Формат входных данных четко регламентируется, что позволяет избежать проблем с обработкой входных данных, а также структурировать их проверку.

На вход интерпретатора микропрограммного автомата с жесткой логикой подается строка, представляющая собой последовательность из единиц и нулей. Данная строка считывается из файла input.txt, который находится в папке с проектом. Эта строка является набором условий переходов Xi.

После ввода корректной строки с набором условий переходов необходимо выбрать один из способов задания: логическое выражение или граф-схема. Выбор происходит посредством ввода в консоль кода способа задания 1 или 2, если код введен некорректно, выводится сообщение об ошибке и происходит прерывание работы программы.

Выходные данные представляют собой пошаговое отображение работы программы. На каждой итерации отображается текущее состояние, состояние, в которое осуществляется переход и выходные данные, если таковые имеются.

После считывания последовательности входных данных в программе выполняются следующие проверки:

1. Проверка входных данных на соответствие шаблону – 9-разрядная строка. В случае ввода строки меньшего или большего размера выводится сообщение об ошибке, работа программы завершается.

2. Проверка набора кодов условий переходов на наличие символов отличных от 0 и 1. В случае ввода другого символа, выводится сообщение о некорректном вводе. Работа программы завершается.

Далее выводится сообщение с кодами способов задания, после ввода которого идет проверка на соответствие. В случае ввода некорректного кода, выводится сообщение об ошибке, работа программы завершается.

4. После выбора способа задания, выводятся коды состояния перехода и выходные данные при каждом переходе.

5. По завершению работы программы, выводится время, которое было затрачено программой на выполнение алгоритма в зависимости от способа задания.

Входные данные задаются внутри текстового файла, выходные данные отображаются в консоли.

# 6. Разработка интерпретатора микропрограммного автомата с жесткой логикой

Работа программы начинается до ввода входных данных – объявляются переменные, задается кодировка для вывода сообщений на русском языке. Далее происходит считывание строки из файла и проверка её на корректность. Если входная строка соответствует всем требованиям, то в консоль выводится сообщение с кодами возможных способов задания на выбор. В противном случае выводится сообщение о вводе некорректных данных, работа программы прерывается. Введенный код способа задания также проходит проверку на соответствие, и, в случае ввода несуществующего кода, выводится сообщение об ошибке ввода. Все значения необходимые для расчетов хранятся в виде массивов, как глобальные переменные. Заданы массивы с входными и выходными сигналами (x[9] и y[15] соответственно), массив для хранения кода состояния (a[4]) и массивы, для хранения функций возбуждения (w[4] и u[4]).

## 6.1. Разработка для способа задания – логическое выражение

При выборе способа задания «логическое выражение» происходит вызов функции logic(). Сначала в функции в бесконечном цикле вычисляются выходные значения, а также функции возбуждения. Результатом вычисления логических выражений является значение 0 или 1, которое записывается в глобальный массив. Пересчет состояния функций выходов и возбуждения происходит на каждой итерации цикла.

После пересчета логических выражений выводится код текущего состояния, записанный в массиве a[]. Далее в зависимости от значений, содержащихся в массивах w[] и u[] в массиве a[] происходит замена в соответствующей позиции с 0 на 1 или с 1 на 0. После обновления состояния, оно также выводится в консоль. Далее следует вывод всех ненулевых значений массива y[], хранящего информацию о функциях вывода, где 1 в позиции y[i] указывает на вывод Yi при совершении перехода между состояниями.

В случае попадания из состояния Si в состояние S0, программа завершает работу. Также происходит отслеживание циклов. Для этого создается массив cnt[], содержащий информацию о количестве проходов через каждую вершину. Если хоть одно значение в массиве будет больше 1, происходит прерывание работы программы и выводится сообщение о зацикливании. По завершении выводится время работы программы.

Реализация данного способа задания составила около 100 строк кода. Основная сложность реализации заключается в том, что необходимо прописать все логические выражения для функций выходов и возбуждения, которые без минимизации могут быть довольно громоздкими.

## 6.2. Разработка для способа задания – граф-схема

При выборе способа задания «граф-схема» происходит вызов функции graph (). Сначала в функции в массиве s[] задается код начального состояния.

Далее в бесконечном цикле происходит передвижение по каждой строке из семиколоночной таблицы. С помощью условного оператора проверяется код текущего состояния и входные данные, необходимые для перехода в следующее состояние, если условие удовлетворяет заданным в семиколоночной таблице, переход состоится. После этого на экран выводится код состояния, далее код состояния изменяется и также выводится на экран. Если при переходе существуют выходные данные Y, они также выводятся в консоль.

В случае попадания из состояния Si в состояние S0, программа завершает работу. Также происходит отслеживание циклов. Для этого создана переменная cycle, хранящий информацию о количестве проходов через вершину в соответствующей её номеру ячейке. Если вершина уже была посещена, происходит прерывание работы программы и выводится сообщение о зацикливании. По завершении выводится время работы программы.

Реализация данного способа задания составила около 200 строк кода. Основная сложность реализации также как и в предыдущем методе заключается в прописывании довольно громоздких формул.

# 7. Разработка рабочей нагрузки и тестирование интерпретатора

Разработка рабочей нагрузки (Таблица 7.1) необходима для проверки работы программы при всех возможных вариантах входных данных. Рабочая нагрузка представляет собой строку, подающуюся на вход программы. Рабочая нагрузка для способа задания «логическое выражение» идентична рабочей нагрузке для метода «Проход по графу». Примеры рабочей нагрузки представлены ниже

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| 111111111 | 0000 --> 0001 : -  0001 --> 0010 : Y1  0010 --> 0011 : Y2 Y3  0011 --> 0100 : Y4 Y5  0100 --> 0110 : Y6  0110 --> 0111 : Y4 Y7 Y8  0111 --> 1000 : Y9  1000 --> 1001 : Y10  1001 --> 0000 : Y11  Обход Завершен!  Время работы алгоритма 12 мс |
| 000000000 | 0000 --> 0001 : -  0001 --> 1010 : Y1  1010 --> 1100 : Y5 Y12  1100 --> 1100 : Y8 Y13 Y14  Произошло зацикливание  Время работы алгоритма 5 мс |
| 111001110 | 0000 --> 0001 : -  0001 --> 0010 : Y1  0010 --> 0011 : Y2 Y3  0011 --> 0100 : Y4 Y5  0100 --> 0110 : Y6  0110 --> 0100 : Y4 Y7 Y8  Произошло зацикливание  Время работы алгоритма 25 мс |
| 111001110 | Неверное кол-во входных сигналов |
| 51100111 | Значение во входных данных != 0 или 1 |

Таблица 7.1. Рабочая нагрузка

Так как количество различных вариантов входных данных довольно велико (полное число переборов будет составлять вариантов) перебрать их все вручную не представляется возможным. Поэтому для проверки полной корректности работы програмы, был разработан метод тестирования testing(). В данном методе с помощью конструкции bitset<9> осуществляется полный перебор входных данных, который затем подается на вход тестируемому алгоритму.

# 8. Сравнительный анализ данных полученных протоколов

Сравнение результатов тестирования реализации двух способов задания: «логическое выражение» и «граф-схема» - осуществляется на основе ранее полученных результатов тестирования рабочей нагрузки.

Для получения результатов тестирования (Таблица 8.1) двух протоколов выходных данных использовались одинаковые входные данные.

|  |  |
| --- | --- |
| Логическое выражение | граф-схема |
| 0000-->0001 :  0001-->1010 : Y1  1010-->1100 : Y5 Y12  1100-->1100 : Y8 Y13 Y14  1100-->1100 : Y8 Y13 Y14  Зацикливание…  Время работы алгоритма: 9 мс | 0000 --> 0001 : -  0001 --> 1010 : Y1  1010 --> 1100 : Y5 Y12  1100 --> 1100 : Y8 Y13 Y14  Произошло зацикливание  Время работы алгоритма 6 мс |
| 0000-->0001 :  0001-->1010 : Y1  1010-->1011 : Y5 Y12  1011-->1100 : Y6  1100-->0000 : Y8 Y13 Y14  Время работы алгоритма 6 мс | 0000 --> 0001 : -  0001 --> 1010 : Y1  1010 --> 1011 : Y5 Y12  1011 -->1100 : Y6  1100 --> 0000 : Y8 Y13 Y14  Обход Завершен!  Время работы алгоритма 4 мс |
| 0000-->0001 :  0001-->0010 : Y1  0010-->0011 : Y2 Y3  0011-->0100 : Y4 Y5  0100-->0110 : Y6  0110-->0111 : Y4 Y7 Y8  0111-->0000 : Y9  Время работы алгоритма 9 мс | 0000 --> 0001 : -  0001 --> 0010 : Y1  0010 --> 0011 : Y2 Y3  0011 --> 0100 : Y4 Y5  0100 --> 0110 : Y6  0110 --> 0111 : Y4 Y7 Y8  0111 --> 0000 : Y9  Обход Завершен!  Время работы алгоритма 7 мс |

Таблица 8.1. Сравнительный анализ протоколов выходных данных

Из данной таблицы видно, что время работы двух интерпретатора при различных способах задания отличаются друг от друга. Способ задания «Граф-схема» в общем случае оказывается более эффективным по времени. Это происходит из-за того, что в нем задействованы лишь проверки текущего состояния и входных сигналов, тогда как в методе «Логические выражения» коды состояний постоянно пересчитываются.

# Заключение

В ходе выполнения курсовой работы были изучены основные аспекты проектирования и создания программной реализации микропрограммного автомата.

На основе исходных данных был проведен анализ и проектирование алгоритма, построена отмеченная ГСА и граф автомата Мура. Также был проведен канонический метод структурного синтеза, состоящий из следующих этапов:

* Кодирование входных и выходных сигналов, а также состояний;
* Построение семиколоночной таблицы;
* Построение логических функций переходов и выходов;

Была создана программная реализация микропрограммного автомата для двух способов задания: «логическое выражение» и «граф-схема» - на языке программирования С++.

Для тестирования программы были созданы протоколы входных и выходных данных, а также рабочая нагрузка, позволяющая проверить работу программу при всех возможных вариантах переходов. Также был написан метод тестирования для проверки корректности программы.

Был проведен сравнительный анализ программных реализаций двух способов задания: «логическое выражение» и «граф-схема» – на основе результатов тестирования, с учетом времени выполнения программы, и сложности реализации методов, с учетом количества написанных строк кода.

# Используемые ресурсы

1. С.А. Майоров, Г.И. Новиков, О.Ф. Немолочнов, С.И. Баранов, П.А. Шипилов, В.И. Скорубский, Г.А. Петухов, Б.Д. Тимченок Проектирование цифровых вычислительных машин. Под. ред. С.А. Майорова. Учебное пособие для студентов вузов. М., «Высш. школа», 1972. - 344 с.
2. И.В. Жукалина Теория автоматов: методические указания к курсовому проекту для специальности 230101 / И.В. Жукалина. - Оренбург: ГОУ ОГУ, 2008. – 29 с.
3. https://studfiles.net/preview/1444567/ - Сравнение по быстродействию автоматов с жесткой и программируемой логикой (Дата обращения: 30.05.2018)
4. https://de.ifmo.ru/bk\_netra/page.php?tutindex=25&index=86 – Управляющие автоматы на элементах с жесткой логикой (Дата обращения: 30.05.2018)

# Приложения

Исходный код:

#include <iostream>

#include <string>

#include <fstream>

#include <stack>

#include <vector>

#include <bitset>

#include <ctime>

using namespace std;

string input;

int x[9] = {};

int y[15] = {};

int a[4] = {};

int w[4] = {};

int u[4] = {};

string cond;

void logic() {

int cnt[14] = {};

while (1) {

cond = "";

y[0] = !a[0] && !a[1] && !a[2] && a[3];

y[1] = !a[0] && !a[1] && a[2] && !a[3];

y[2] = (!a[0] && !a[1] && a[2] && !a[3]) ||

(!a[0] && a[1] && !a[2] && a[3]);

y[3] = (!a[0] && !a[1] && a[2] && a[3]) ||

(!a[0] && a[1] && a[2] && !a[3]);

y[4] = (!a[0] && !a[1] && a[2] && a[3]) ||

(a[0] && !a[1] && a[2] && !a[3]);

y[5] = (!a[0] && a[1] && !a[2] && !a[3]) ||

(a[0] && !a[1] && a[2] && a[3]);

y[6] = !a[0] && a[1] && a[2] && !a[3];

y[7] = (!a[0] && a[1] && a[2] && !a[3]) ||

(a[0] && a[1] && !a[2] && !a[3]);

y[8] = !a[0] && a[1] && a[2] && a[3];

y[9] = a[0] && !a[1] && !a[2] && !a[3];

y[10] = a[0] && !a[1] && !a[2] && a[3];

y[11] = a[0] && !a[1] && a[2] && !a[3];

y[12] = a[0] && a[1] && !a[2] && !a[3];

y[13] = a[0] && a[1] && !a[2] && !a[3];

y[14] = a[0] && a[1] && !a[2] && a[3];

w[0] = (!a[0] && !a[1] && !a[2] && a[3] && !x[0]) || (!a[0] && !a[1] && a[2] && !a[3] && !x[1]) ||

(!a[0] && a[1] && a[2] && a[3] && x[0] && x[4]) || (!a[0] && a[1] && a[2] && a[3] && !x[0] && x[5] && x[6]) ||

(!a[0] && a[1] && a[2] && a[3] && x[0] && x[4] && x[7]) || (!a[0] && a[1] && a[2] && a[3] && !x[0] && !x[5] && x[7]) ||

(!a[0] && a[1] && a[2] && a[3] && !x[0] && x[5] && !x[6] && x[7]);

w[1] = (!a[0] && !a[1] && a[2] && !a[3] && !x[1]) || (!a[0] && !a[1] && a[2] && a[3] && x[2]) || (!a[0] && !a[1] && a[2] && a[3] && !x[2]) ||

(a[0] && !a[1] && a[2] && !a[3] && !x[8]) || (a[0] && !a[1] && a[2] && a[3]);

w[2] = (!a[0] && !a[1] && !a[2] && a[3] && x[0]) || (!a[0] && !a[1] && !a[2] && a[3] && !x[0]) ||

(!a[0] && a[1] && !a[2] && !a[3]) || (!a[0] && a[1] && !a[2] && a[3]) ||

(a[0] && a[1] && !a[2] && !a[3] && x[3] && !x[5] && x[1]) ||

(a[0] && a[1] && !a[2] && !a[3] && !x[3] && x[8]);

w[3] = (!a[0] && !a[1] && !a[2] && !a[3]) || (!a[0] && !a[1] && a[2] && !a[3] && x[1]) ||

(!a[0] && !a[1] && a[2] && !a[3] && !x[1]) || (!a[0] && a[1] && a[2] && !a[3] && !x[3] && !x[2]) ||

(!a[0] && a[1] && a[2] && !a[3] && x[3]) || (a[0] && !a[1] && !a[2] && !a[3] && x[7]) ||

(a[0] && !a[1] && a[2] && !a[3] && x[8]) || (a[0] && a[1] && !a[2] && !a[3] && x[3] && !x[5] && x[1]) ||

(a[0] && a[1] && !a[2] && !a[3] && x[3] && x[5] && !x[6] && x[7]) || (a[0] && a[1] && !a[2] && !a[3] && !x[3] && x[8]) ||

(a[0] && a[1] && !a[2] && !a[3] && x[3] && !x[5] && !x[1]);

u[0] = (a[0] && !a[1] && !a[2] && !a[3] && !x[7]) || (a[0] && !a[1] && !a[2] && a[3]) ||

(a[0] && a[1] && !a[2] && !a[3] && x[3] && x[5] && !x[6] && !x[7]) ||

(a[0] && a[1] && !a[2] && !a[3] && x[3] && !x[5] && x[1]) || (a[0] && a[1] && !a[2] && a[3]);

u[1] = (!a[0] && a[1] && a[2] && a[3] && x[0] && x[4]) || (!a[0] && a[1] && a[2] && a[3] && !x[0] && x[5] && x[6]) ||

(!a[0] && a[1] && a[2] && a[3] && x[0] && x[4] && x[7]) || (!a[0] && a[1] && a[2] && a[3] && !x[0] && !x[5] && x[7]) ||

(!a[0] && a[1] && a[2] && a[3] && !x[0] && x[5] && !x[6] && x[7]) || (!a[0] && a[1] && a[2] && a[3] && x[0] && !x[4] && !x[7]) ||

(!a[0] && a[1] && a[2] && a[3] && !x[0] && !x[5] && !x[7]) || (!a[0] && a[1] && a[2] && a[3] && !x[0] && x[5] && !x[6] && !x[7]) ||

(a[0] && a[1] && !a[2] && !a[3] && x[3] && x[5] && !x[6] && !x[7]) || (a[0] && a[1] && !a[2] && !a[3] && x[3] && x[5] && x[6]) ||

(a[0] && a[1] && !a[2] && !a[3] && x[3] && x[5] && !x[6] && x[7]) || (a[0] && a[1] && !a[2] && !a[3] && !x[3] && x[8]) || (a[0] && a[1] && !a[2] && a[3]);

u[2] = (!a[0] && !a[1] && a[2] && !a[3] && !x[1]) || (!a[0] && !a[1] && a[2] && a[3] && x[2]) ||

(!a[0] && !a[1] && a[2] && a[3] && !x[2]) || (!a[0] && a[1] && a[2] && !a[3] && !x[3] && x[2]) ||

(!a[0] && a[1] && a[2] && !a[3] && !x[3] && !x[2]) || (!a[0] && a[1] && a[2] && a[3] && x[0] && x[4]) ||

(!a[0] && a[1] && a[2] && a[3] && !x[0] && x[5] && x[6]) || (!a[0] && a[1] && a[2] && a[3] && x[0] && x[4] && x[7]) ||

(!a[0] && a[1] && a[2] && a[3] && !x[0] && !x[5] && x[7]) || (!a[0] && a[1] && a[2] && a[3] && !x[0] && x[5] && !x[6] && x[7]) ||

(!a[0] && a[1] && a[2] && a[3] && x[0] && !x[4] && !x[7]) || (!a[0] && a[1] && a[2] && a[3] && !x[0] && !x[5] && !x[7]) ||

(!a[0] && a[1] && a[2] && a[3] && !x[0] && x[5] && !x[6] && !x[7]) || (a[0] && !a[1] && a[2] && !a[3] && !x[8]) ||

(a[0] && !a[1] && a[2] && a[3]);

u[3] = (!a[0] && !a[1] && !a[2] && a[3] && x[0]) || (!a[0] && !a[1] && !a[2] && a[3] && !x[0]) ||

(!a[0] && !a[1] && a[2] && a[3] && x[2]) || (!a[0] && a[1] && !a[2] && a[3]) ||

(!a[0] && a[1] && a[2] && a[3] && x[0] && x[4]) || (!a[0] && a[1] && a[2] && a[3] && !x[0] && x[5] && x[6]) ||

(!a[0] && a[1] && a[2] && a[3] && x[0] && !x[4] && !x[7]) || (!a[0] && a[1] && a[2] && a[3] && !x[0] && !x[5] && !x[7]) ||

(!a[0] && a[1] && a[2] && a[3] && !x[0] && x[5] && !x[6] && !x[7]) || (a[0] && !a[1] && !a[2] && a[3]) ||

(a[0] && !a[1] && a[2] && a[3]) || (a[0] && a[1] && !a[2] && a[3]);

for (int i = 0; i < 4; i++) {

cout << a[i];

cond += a[i] + '0';

}

for (int i = 0; i < 4; i++) {

if (w[i])

a[i] = 1;

if (u[i])

a[i] = 0;

}

cout << "-->";

for (int i = 0; i < 4; i++) {

cout << a[i];

}

cout << " : ";

for (int i = 0; i < 15; i++) {

if (y[i])

cout << "Y" << i + 1 << " ";

}

cout << endl;

if (cnt[bitset<4>(cond).to\_ulong()] > 0) {

cout << "\nЗацикливание...";

break;

}

else

cnt[bitset<4>(cond).to\_ulong()]++;

if (a[0] == 0 && a[1] == 0 && a[2] == 0 && a[3] == 0)

break;

}

}

void graph() {

int s[4] = {};

bool cycle = false;

while (1) {

if (cycle) {

cout << "Произошло зацикливание" << endl;

break;

}

if (s[0] == 0 && s[1] == 0 && s[2] == 0 && s[3] == 0) {//2

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 0; s[2] = 0; s[3] = 1;

cout << s[0] << s[1] << s[2] << s[3] << " : -" << endl;

}

if (s[0] == 0 && s[1] == 0 && s[2] == 0 && s[3] == 1 && x[0]) {//3

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 0; s[2] = 1; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y1" << endl;

}

if (s[0] == 0 && s[1] == 0 && s[2] == 0 && s[3] == 1 && !x[0]) {//4

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 1; s[1] = 0; s[2] = 1; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y1" << endl;

}

if (s[0] == 0 && s[1] == 0 && s[2] == 1 && s[3] == 0 && x[1]) {//5

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 0; s[2] = 1; s[3] = 1;

cout << s[0] << s[1] << s[2] << s[3] << " : Y2 Y3" << endl;

}

if (s[0] == 0 && s[1] == 0 && s[2] == 1 && s[3] == 0 && !x[1]) {//6

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 1; s[1] = 1; s[2] = 0; s[3] = 1;

cout << s[0] << s[1] << s[2] << s[3] << " : Y2 Y3" << endl;

}

if (s[0] == 0 && s[1] == 0 && s[2] == 1 && s[3] == 1 && x[2]) {//7

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 1; s[2] = 0; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y4 Y5" << endl;

}

if (s[0] == 0 && s[1] == 0 && s[2] == 1 && s[3] == 1 && !x[2]) {//8

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 1; s[2] = 0; s[3] = 1;

cout << s[0] << s[1] << s[2] << s[3] << " : Y4 Y5" << endl;

}

if (s[0] == 0 && s[1] == 1 && s[2] == 0 && s[3] == 0) {//9

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 1; s[2] = 1; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y6" << endl;

}

if (s[0] == 0 && s[1] == 1 && s[2] == 0 && s[3] == 1) {//10

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 1; s[2] = 1; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y3" << endl;

}

if (s[0] == 0 && s[1] == 1 && s[2] == 1 && s[3] == 0 && !x[3] && x[2]) {//11

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 1; s[2] = 0; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y4 Y7 Y8" << endl;

cycle = true;

}

if (s[0] == 0 && s[1] == 1 && s[2] == 1 && s[3] == 0 && !x[3] && !x[2]) {//12

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 1; s[2] = 0; s[3] = 1;

cout << s[0] << s[1] << s[2] << s[3] << " : Y4 Y7 Y8" << endl;

cycle = true;

}

if (s[0] == 0 && s[1] == 1 && s[2] == 1 && s[3] == 0 && x[3]) {//13

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 1; s[2] = 1; s[3] = 1;

cout << s[0] << s[1] << s[2] << s[3] << " : Y4 Y7 Y8" << endl;

}

if (s[0] == 0 && s[1] == 1 && s[2] == 1 && s[3] == 1 && x[0] && x[4]) {//14

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 1; s[1] = 0; s[2] = 0; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y9" << endl;

}

if (s[0] == 0 && s[1] == 1 && s[2] == 1 && s[3] == 1 && !x[0] && x[5] && x[6]) {//15

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 1; s[1] = 0; s[2] = 0; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y9" << endl;

}

if (s[0] == 0 && s[1] == 1 && s[2] == 1 && s[3] == 1 && x[0] && !x[4] && x[7]) {//16

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 1; s[1] = 0; s[2] = 0; s[3] = 1;

cout << s[0] << s[1] << s[2] << s[3] << " : Y9" << endl;

}

if (s[0] == 0 && s[1] == 1 && s[2] == 1 && s[3] == 1 && !x[0] && !x[5] && x[7]) {//17

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 1; s[1] = 0; s[2] = 0; s[3] = 1;

cout << s[0] << s[1] << s[2] << s[3] << " : Y9" << endl;

}

if (s[0] == 0 && s[1] == 1 && s[2] == 1 && s[3] == 1 && !x[0] && x[5] && !x[6] && x[7]) {//18

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 1; s[1] = 0; s[2] = 0; s[3] = 1;

cout << s[0] << s[1] << s[2] << s[3] << " : Y9" << endl;

}

if (s[0] == 0 && s[1] == 1 && s[2] == 1 && s[3] == 1 && x[0] && !x[4] && !x[7]) {//19

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 0; s[2] = 0; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y9" << endl;

cout << "Обход Завершен!" << endl;

break;

}

if (s[0] == 0 && s[1] == 1 && s[2] == 1 && s[3] == 1 && !x[0] && !x[5] && !x[7]) {//20

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 0; s[2] = 0; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y9" << endl;

cout << "Обход Завершен!" << endl;

break;

}

if (s[0] == 0 && s[1] == 1 && s[2] == 1 && s[3] == 1 && !x[0] && x[5] && !x[6] && x[7]) {//21

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 0; s[2] = 0; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y9" << endl;

cout << "Обход Завершен!" << endl;

break;

}

if (s[0] == 1 && s[1] == 0 && s[2] == 0 && s[3] == 0 && x[7]) {//22

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 1; s[1] = 0; s[2] = 0; s[3] = 1;

cout << s[0] << s[1] << s[2] << s[3] << " : Y10" << endl;

}

if (s[0] == 1 && s[1] == 0 && s[2] == 0 && s[3] == 0 && !x[7]) {//23

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 0; s[2] = 0; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y10" << endl;

cout << "Обход Завершен!" << endl;

break;

}

if (s[0] == 1 && s[1] == 0 && s[2] == 0 && s[3] == 1) {//24

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 0; s[2] = 0; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y11" << endl;

cout << "Обход Завершен!" << endl;

break;

}

if (s[0] == 1 && s[1] == 0 && s[2] == 1 && s[3] == 0 && x[8]) {//25

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 1; s[1] = 0; s[2] = 1; s[3] = 1;

cout << s[0] << s[1] << s[2] << s[3] << " : Y5 Y12" << endl;

}

if (s[0] == 1 && s[1] == 0 && s[2] == 1 && s[3] == 0 && !x[8]) {//26

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 1; s[1] = 1; s[2] = 0; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y5 Y12" << endl;

}

if (s[0] == 1 && s[1] == 0 && s[2] == 1 && s[3] == 1) {//27

cout << s[0] << s[1] << s[2] << s[3] << " -->";

s[0] = 1; s[1] = 1; s[2] = 0; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y6" << endl;

}

if (s[0] == 1 && s[1] == 1 && s[2] == 0 && s[3] == 0 && x[3] && x[5] && !x[6] && !x[7]) {//28

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 0; s[2] = 0; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y8 Y13 Y14" << endl;

cout << "Обход Завершен!" << endl;

break;

}

if (s[0] == 1 && s[1] == 1 && s[2] == 0 && s[3] == 0 && x[3] && !x[5] && x[1]) {//29

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 1; s[2] = 1; s[3] = 1;

cout << s[0] << s[1] << s[2] << s[3] << " : Y8 Y13 Y14" << endl;

}

if (s[0] == 1 && s[1] == 1 && s[2] == 0 && s[3] == 0 && x[3] && x[5] && x[6]) {//30

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 1; s[1] = 0; s[2] = 0; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y8 Y13 Y14" << endl;

}

if (s[0] == 1 && s[1] == 1 && s[2] == 0 && s[3] == 0 && x[3] && x[5] && !x[6] && x[7]) {//31

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 1; s[1] = 0; s[2] = 0; s[3] = 1;

cout << s[0] << s[1] << s[2] << s[3] << " : Y8 Y13 Y14" << endl;

}

if (s[0] == 1 && s[1] == 1 && s[2] == 0 && s[3] == 0 && !x[3] && x[8]) {//32

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 1; s[1] = 0; s[2] = 1; s[3] = 1;

cout << s[0] << s[1] << s[2] << s[3] << " : Y8 Y13 Y14" << endl;

cycle = true;

}

if (s[0] == 1 && s[1] == 1 && s[2] == 0 && s[3] == 0 && !x[3] && !x[8]) {//33

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 1; s[1] = 1; s[2] = 0; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y8 Y13 Y14" << endl;

cycle = true;

}

if (s[0] == 1 && s[1] == 1 && s[2] == 0 && s[3] == 0 && x[3] && !x[5] && !x[1]) {//34

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 1; s[1] = 1; s[2] = 0; s[3] = 1;

cout << s[0] << s[1] << s[2] << s[3] << " : Y8 Y13 Y14" << endl;

}

if (s[0] == 1 && s[1] == 1 && s[2] == 0 && s[3] == 1) {//35

cout << s[0] << s[1] << s[2] << s[3] << " --> ";

s[0] = 0; s[1] = 0; s[2] = 0; s[3] = 0;

cout << s[0] << s[1] << s[2] << s[3] << " : Y15" << endl;

cout << "Обход Завершен!" << endl;

break;

}

}

}

void testing() {

int i = 0;

string buf;

while (i < 512) {

for (int j = 0; j < 4; j++) {

a[j] = 0;

}

buf = bitset<9>(i).to\_string();

cout << "\n\n|" << buf << "|\n\n";

input = bitset<9>(i).to\_string();

for (int j = 0; j < input.length(); j++) {

x[j] = input[j] - '0';

}

graph();

i++;

}

}

int main() {

setlocale(LC\_ALL, "Russian");

//testing();

ifstream f("input.txt");

getline(f, input);

if (input.length() != 9) {

cout << "Неверное кол-во входных сигналов";

exit(0);

}

for (int i = 0; i < input.length(); i++) {

x[i] = input[i] - '0';

if (x[i] != 1 && x[i] != 0) {

cout << "Значение во входных данных != 0 или 1";

exit(0);

}

}

int c;

cout << "Выберите способ задания функций :" << endl << "1 - Логические выражения" << endl << "2 - Обход графа" << endl;

while (1) {

cin >> c;

if (c == 1) {

double start = clock();

logic();

double end = clock();

cout << "Время работы алгоритма " << end - start << " мс" << endl;

}

else if (c == 2) {

double start = clock();

graph();

double end = clock();

cout << "Время работы алгоритма " << end - start << " мс" << endl;

}

else

cout << "Error";

}

}