Министерство образования и науки РФ
Федеральное государственное образовательное учреждение
высшего образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ФГБОУ ВО «АмГУ»)

ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ, СЕТИ И МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ УПРАВЛЕНИЯ

сборник учебно-методических материалов
для направления подготовки 15.03.04
«Автоматизация технологических процессов и производств»

Благовещенск 2017

Печатается по решению
Редакционно-издательского совета
энергетического факультета
Амурского государственного университета

Составитель: Теличенко Д.А.

Вычислительные машины, сети и микропроцессорные системы управления: сборник учебно-методических материалов для направления подготовки 15.03.04 – Автоматизация технологических процессов и производств. – Благовещенск: Амурский гос. ун-т, 2017.

Рассмотрен на заседании кафедры автоматизации производственных процессов и электротехники 24.05.2017, протокол № 9.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Дисциплина Б1.В.06 «Вычислительные машины, сети и микропроцессорные системы управления» относится к базовой вариативной части и базируется на курсах: «Математика», «Физика», «Электротехника и электроника», «Информатика и информационные технологии», «Программирование и алгоритмизация», «Пакеты прикладных программ для ПВЭМ»,

Знания и умения, приобретенные студентами при изучении дисциплины, используется в специальных курсах, в частности «Программирование ПЛК» или «Программное обеспечение систем управления», «Проектирование автоматизированных систем», «Интегрированные системы проектирования и управления», при выполнении курсовых проектов и работ, а также ВКР и в практической деятельности выпускника.

Цель дисциплины: сформировать у студентов знания о методах и способах использования вычислительных машин, компьютерных систем, сетей и микропроцессорных систем управления для решения различных задач в области автоматизации производственных процессов.

Задачи дисциплины:

- ознакомиться с архитектурой вычислительной и управляющей техники;
- привить навыки по оценке, выбору и использованию современной вычислительной и микропроцессорной техники для решения задач в области автоматизации;
- развить умение применять приемы и технологии использования современных информационных и управляющих сетей;
- привить умение проектирования систем управления различной сложности на основе современных микропроцессорных систем.

В результате освоения дисциплины обучающийся должен демонстрировать следующие результаты образования:

1) Знать:

- основные принципы организации и архитектуру вычислительных машин, систем, сетей и микропроцессорных систем управления;
- принципы организации функциональных и интерфейсных связей вычислительных и микропроцессорных систем с объектами автоматизации;
- основные современные информационные технологии передачи и обработки данных, основы построения управляющих локальных и глобальных сетей;
- принципы построения и функционирования микропроцессоров и микроконтроллеров;
- основные тенденции развития микропроцессорных систем управления;

2) Уметь:

- анализировать работу и проектировать отдельные цифровые узлы современных вычислительных машин и их комплексов;
- правильно выбирать средства для анализа и проектирования систем автоматизации и управления;
- анализировать работу вычислительных машин и сетей;
- использовать основные технологии передачи информации в среде локальных сетей, сети Интернет;
- проектировать микроконтроллерные системы управления;
- программировать и отлаживать системы с микроконтроллерами;

3) Владеть:

- навыками работы с вычислительной техникой, передачей информации в среде локальных сетей, Интернет;
- способами сопряжения микропроцессорных систем с элементами систем автоматики;
- способами работы с программными средствами проектирования и отладки микропроцессорных систем.

1. СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

Дисциплина «Вычислительные машины, сети и микропроцессорные системы управления» является комплексной и проводится в семестрах 5, 6 и 7.

Общая трудоемкость дисциплины составляет 10 зачетных единиц, 360 часов.

Структура и содержание дисциплины отражено в следующих видах учебной работы: лекционные занятия; практические занятия; лабораторные работы; самостоятельная работа; текущий и промежуточный контроли.

Формы текущего контроля успеваемости: контрольная работа 1 или 2; допуск и защита лабораторной работы 1,…,5; подготовка к практикам и выполнение домашних заданий.

Формы аттестации: защита и выполнение индивидуальной расчетно-графической работы; выполнение курсового проекта; дифференцированный зачет (5 семестр), зачет (6 семестр) и экзамен (7 семестр).

1.1. Лекционные занятия

5 семестр (36 часов)

1.1. Введение. Основные понятия о процессе автоматизированной обработки данных – 2 ч.

Информация. Меры информации (определение, формулы, примеры): объем данных, количество переданной информации, априорная и апостериорная информация, энтропия, степень информативности, тезаурус, полезность информации, экономический эффект. Показатели качества информации (определение, формулы, примеры): репрезентативность, содержательность, достаточность, доступность, актуальность, своевременность, точность, достоверность, ценность. Выводы и замечания.

1.2. Архитектура ВМ. Классификация ВМ и история развития вычислительной техники – 2 ч.

Определение ВМ, ее структуры и архитектуры. Быстродействие и производительность. Характеристики ВМ. Классификация средств электронно-вычислительной техники. Замечания по способам классификации ВМ. Принцип академика В.М. Глушкова. ВМ с точки зрения использования человеком.

1.3. Архитектура ВМ. Функциональная и структурная организация ВМ – 2 ч.

Общие понятия о функциональной и структурной организации ВМ. Различные точки зрения на функционирование вычислительного процесса. Точка зрения пользователя и программиста. Определение функциональной и структурной организации ВМ. степени детальности структурных схем. Обобщенная структура ВМ. Схема обобщенной структуры. Характеристики каждой из подсистем обобщенной структуры и ее особенности. Структура и состав ВМ. Концепция Дж. Фон Неймана. Устройство управления, арифметическо-логическое устройство, память и устройство ввода-вывода. Потоки данных, команд и управляющих сигналов. Принципы построения ВМ. Основные принципы взаимодействия между элементами вычислительной системы и основы их функционирования.

1.4. Аппаратные особенности ВМ различных поколений. Принцип построения и функционирования ВМ пятого поколения – 2 ч.

Вычислительные машины первого и второго поколений: структура, основные элементы, принципы взаимодействия, особенности построения. Структура простейшего АЛУ. Вычислительные машины третьего поколения: структура, основные элементы, принципы взаимодействия, особенности построения. Вычислительные машины четвертого и пятого поколений: структура, основные элементы, принципы взаимодействия, особенности построения. Основные принципы построения ВМ пятого поколения. Общие принципы функционирования ВМ пятого поколения. Кризис структуры Дж. Фон Неймана. Вычислительные машины шестого поколения.

1.5. Организация микропроцессоров. Функциональная структура МП – 2 ч.

Процессор: предназначение, понятие микропроцессора, основные достоинства, направление применения. Структурно-функциональная схема микропроцессора: Операционный блок – состав, предназначение основных элементов, основные операции, специализированные блоки аппаратного умножения и деления, разрядность микропроцессора, понятие специализации регистров, примеры, особенности изменения содержимого регистра словосостояния. Блок управления – предназначение, выполняемые действия, состав, особенности функционирования, фазы выполнения команд, структура команд, микрокоманды, микрооперации, понятие микропрограммного автомата с мягкой и жесткой логикой, особенности современной реализации элементов блока управления. Интерфейсный блок – предназначение, понятие интерфейса ввода-вывода, функции интерфейсного блока, понятие системной шины и электрической спецификации сигналов на шине, цикл шины, принцип квитирования.

1.6. Особенности организации процессоров при использовании внутренних регистров – 2 ч.

Аккумуляторная архитектура – основы построения, принцип работы, характеристики, примеры. Много аккумуляторная архитектура – основы построения, принцип работы, характеристики, примеры. Стековые процессоры. Понятие стека, дисциплина FIFO, LIFO. Принцип работы стека и его предназначение. Адресация в стеке. Примеры работы со стеком. Характеристики стека. Особенности стековых процессоров, их структура. Пример стекового процессора на базе специализированного арифметического сопроцессора для вычислений с плавающей точкой.

1.7. Система команд – 2 ч.

Система команд как одна из важнейших характеристик микропроцессора. Понятие системы команд (форматы команд и обрабатываемых данных, список команд и их функциональное назначение, способы адресации данных). Группы команд по функциональным признакам (предназначение, принцип работы, примеры, особенности использования): команды пересылок данных и ввода-вывода; арифметические и поразрядные двоичные команды; команды передачи управления. Замечания по системе команд современных микропроцессоров. Структура команд: операционная и адресная часть, их предназначение, характеристика. Особенности адресной части команд. Естественный и принудительный способ адресации команд. Примеры и пояснения.

1.8. Способы адресации – 2 ч.

Общие сведения по способам адресации. Адресация данных. Прямая адресация, принцип работы и особенности. Прямая регистровая адресация. Примеры. Непосредственная адресация – принцип, особенности, примеры. Неявная адресация – принцип, особенности, примеры. Косвенная адресация – принцип, особенности, примеры. Особенности и преимущества косвенной адресации на примере организации цикла. Относительная адресация или базирование – принцип, особенности, примеры. Формирование исполнительного адреса. Страничная организация и сегментированная память. Базовая и индексная адресация (особенности, принцип, примеры). Адресация команд.

1.9. Особенности организация памяти ВМ – 4 ч.

Понятие памяти ВМ, характеристики отдельных устройств памяти. Быстродействие памяти, время доступа к памяти, длительность цикла памяти. Противоречивость требований к увеличению емкости и быстродействию памяти. Уровни памяти. Сверхоперативный уровень, оперативный уровень, внешний уровень. Замечания по производительности ВМ и особенностям организации памяти. Организация внутренней памяти процессора (сверхоперативный уровень). Организация оперативной памяти (оперативный уровень). Базовые типы оперативной памяти принцип работы, особенности, сравнительная характеристика, методы управления оперативной памятью: методы управления без использования виртуальной памяти (распределение памяти фиксированными разделами, распределение памяти динамическими разделами, распределение памяти с перемещаемы-

ми разделами); методы управления с использованием виртуальной памяти (понятие виртуальной памяти, задачи виртуальной памяти, страничное распределение, сегментное распределение, странично-сегментное распределение, свопинг). Системы внешней памяти: жесткие диски, гибкие магнитные диски, CD, DVD, новые форматы записи, flash. Методы организации кэш памяти, ее структура и принцип работы. Способы размещения данных в кэш памяти. Методы обновления строк в основной памяти, связь с кэш-памятью. Методы повышения пропускной способности оперативной памяти.

1.10. Организация обмена данными в ВМ – 2 ч.

Общие сведения. Принципы организации обмена. Обмен данными между периферийными устройствами и вычислительным ядром системы. Особенности организации. Программно-управляемая передача. Передача информации с прерыванием программы. Понятие прерывания аппаратные и программные прерывания. Сигнал запроса прерывания. Работа системы при реакции на прерывания. Сравнения и выводы по программно-управляемой передаче и передаче с прерыванием. Передача информации в режиме прямого доступа к памяти (ПДП). Определение режима ПДП. Предназначение режима ПДП. Достоинства и недостатки. Способы организации, примеры.

1.11. Централизованные и распределенные системы обработки данных – 2 ч.

Понятие о централизованных и распределенных системах обработки данных и системах реального времени. Обобщенная структура типовой системы управления (микроконтроллера), ее состав. Объект управления исполнительные устройства, система датчиков, устройства сопряжения с объектом, пульт управления, микропроцессорный (цифровой регулятор). Реальное время протекания процесса, шаг квантования. Иерархическая организация системы управления сложными, распределенными в пространстве объектами – двух и трех уровневая модель. Организация микроконтроллерных систем. Встраиваемые системы управления. Централизованная и распределенная система. Отличительные черты микропроцессорной организации цифровых регуляторов. Встраиваемые средства на базе микроконтроллеров – функции, способы организации. Типы микроконтроллерных систем: автономная, локальная, сетевая конфигурация. Типовая структура микроконтроллера – общие сведения. Основные типы и семейства микроконтроллеров. Базовые принципы организации, состав, основные модули.

1.12. Особенности организации современных однопроцессорных ВМ – 2 ч.

Понятие однопроцессорных и многопроцессорных систем. Таксономия М. Флина. SISD (ОКОД) - компьютеры: определение, характеристика, основные элементы, структура, принципы функционирования. CISC архитектура, RISC архитектура. Суперскалярная обработка: аппаратная реализация, VLIW архитектура. SIMD (ОКМД) - компьютеры: определение, характеристика, основные элементы, структура, принципы функционирования. Матричная архитектура, векторно-конвейерная архитектура, MMX технология.

1.13. Вычислительные системы параллельной обработки данных – 2 ч.

Параллельная обработка данных как архитектурный способ повышения производительности. Методы увеличения производительности вычислительных систем. Основы параллельной обработки. Мультипроцессорные архитектуры, ее преимущества. Трудности реализации мультипроцессорных архитектур (новые типы ошибок, сложности понимания и анализа параллельных процессов, недостаточная разработанность теоретических моделей и методов параллельного программирования). Классификация систем параллельной обработки. Многопроцессорные вычислительные системы (особенности организации и функционирования, примеры, преимущества и недостатки). MISD компьютеры (МКОД), MIMD компьютеры (МКМД), многопроцессорные вычислительные системы (сильно связанные): с общей шиной, с использованием многовходовой памяти, многомашинные вычислительные системы (слабосвязанные): многомашинные комплексы, системы массового параллелизма.

1.14. Вычислительные системы – состояние, производительность, направления развития – 2 ч.

Состояние производства и использования. Направления развития архитектуры. Направления развития высокопроизводительных вычислительных систем. Тенденции развития архитектур с общей памятью. Тенденции развития архитектур систем с разделяемой памятью. Развитие архитектур микропроцессоров высокопроизводительных вычислительных систем (организация внутрикристальной памяти, увеличение состава и числа функциональных устройств, интеграция функций). Направления развития мультипроцессорных систем с распределенной памятью. Производительность мультипроцессорных систем при увеличении числа процессоров. Вычислительные системы на кристалле. Переход к нанотехнологии производства интегральных схем.

1.15. Телекоммуникационные вычислительные сети – 6 ч.

Принципы построения вычислительных сетей. Основные понятия: телекоммуникационная сеть, абонентская станция, телекоммуникационная система. Обобщенная функциональная схема. Организация и работа простейшей сети. Формат сообщений при обмене. Каналы передачи сообщений. Помехоустойчивое кодирование. Последовательность действий при передаче/приеме сообщений. Реакция на подтверждение приема. Параметры производительности сети: задержка времени в передающем узле, время передачи данных, время продвижения сигналов, задержка в приемном узле, время транспортировки, время обмена. Классификация вычислительных сетей: глобальные, региональные, локальные, системные. Архитектурные принципы построения сетей. Физические блоки, логическая организация, топология сети, основы обработки сообщений. Протокол сети. Семиуровневая эталонная модель взаимодействия открытых систем OSI/ISO. Уровни иерархии (прикладной, представительский, сеансовый, транспортный, сетевой, канальный, физический) – характеристика, предназначение, организация. Коммутация и маршрутизация при передаче данных в сети. Коммутация сообщений и пакетов. Дейтограмма. Виртуальный канал. Основы маршрутизации. Основные типы сетевого оборудования: коммутаторы, концентраторы, повторители, мосты, шлюзы, маршрутизаторы, мультиплексоры. Локальные вычислительные сети (ЛВС). Характеристики ЛВС. Типы каналов, способы организации. Асинхронный и синхронный формат сообщений. Цифровые коды. Топологии ЛВС. Одноранговые и многоранговые сети. Файл-сервер, клиент-сервер. Локальная вычислительная сеть Ethernet. Трех уровневая организация. Средства подключения ВМ и ЛВС Ethernet. Способы доступа к среде. Формат кадра. Основные скорости передачи. Сеть Интернет. Стек протоколов TCP/IP. Уровни протоколов сети Интернет. Понятия FTP, SMTP, HTTP, TELNET, WWW. Способы подключения абонента к сети Интернет. Корпоративные сети.

<u>6 семестр (18 часов)</u>

2.1. Микропроцессорные системы: определение, структура, типы – 2ч.

Основные определения. Системы с жесткой логикой и гибкой логикой. Понятие о системе команд. Состав простейшего микропроцессора. Организация связей в микропроцессорных системах. Организация выходных каскадов в цифровых схемах. Структура микропроцессорной системы с шинной организацией. Общий принцип работы микропроцессорной системы и информационные потоки, их предназначения. Режимы работы микропроцессорной системы. Понятие архитектуры. Архитектура современных микропроцессорных систем. Системы с общей памятью. Архитектура систем с разделяемой памятью. Сравнительные характеристики обеих архитектур.

2.2. Организация обмена информацией в МПС – 2ч.

Понятие и элементарные циклы обмена. Двухнаправленность и разрядность шин, мультиплексированные шины, особенности передачи информации, понятие асинхронного и синхронного обмена. Циклы обмена информацией: цикл программного обмена (чтение, запись, мультиплексированные асинхронные шины, временные диаграммы, фаза адреса, фаза данных, основные сигналы, модифицированные циклы, немультеплексированные магистрали и их особенности, особенности асинхронного и синхронного обмена). Циклы

обмена информацией: цикл обмена по прерываниям (прерывания в системе; организация шин при векторных прерываниях: временная диаграмма, принцип работы, основные сигналы; организация шин при радиальных прерываниях: схема, временная диаграмма, принцип работы, основные сигналы; особенности векторных и радиальных прерываний). Циклы обмена информацией: цикл обмена в режиме прямого доступа к памяти (особенности организации режима прямого доступа к памяти, основные сигналы, принцип работы, структура связей, временные диаграммы). Особенности организации обмена по шинам в микропроцессорной системе: прохождение сигналов по шинам, улучшение организации обмена по шинам.

2.3. Шины: арбитраж и повышение эффективности работы – 2 ч.

Арбитраж шин. Определение и понятия арбитраж шин, его предназначение. Распределение приоритетов, применение схем. Распределение по схеме с динамическим приоритетом, схемы арбитража: централизованный (различные схемы) и децентрализованный опрос (различные схемы) – предназначение и особенности организации. Комбинированные схемы арбитража. Ограничение времени контроля над шиной. Опросные схемы арбитража, централизованный и децентрализованный опрос. Схемы основных опросных методов арбитража, принцип организации и работы.

Методы повышения эффективности шин. Пакетный режим пересылки информации, его особенности и временная диаграмма работы, преимущества и недостатки, примеры использования. Конвейеризация транзакций, ее особенности, временная диаграмма. Протокол с расщеплением транзакций, особенности, принцип работы, временная диаграмма. Увеличение полосы пропускания шин. Ускорение транзакций. Повышение эффективности шин с множеством ведущих. Надежность и отказоустойчивость, стандартизация шин.

2.4. Основные элементы МПС: микропроцессор, память и устройства ввода-вывода – 2ч.

Микропроцессор – основной принцип работы: предназначение, основные сигналы, шины, структура и принцип работы. Характеристики микропроцессора, особенности работы, кварцевый резонатор и тактовая частота и ее влияние на производительность, понятие перегрева процессора и особенности обмена информацией. Организация начального пуска и сброса микропроцессора. Организация питания микропроцессора. Использование буферных регистров в микропроцессоре. Функции микропроцессора. Функциональная структура микропроцессора. Аккумуляторная структура микропроцессора, структура с равноправными регистрами. Служебные функции микропроцессора. Особенности выполнения команд и предназначение счетчика команд. Особенности использования и предназначения регистра признаков. Схемы управления прерыванием и прямым доступом к памяти – предназначение принцип работы. Логика работы.

Память в микропроцессорной системе: предназначение, виды, разрядность, особенности организации, пространство памяти, схема подключения, особенности организации оперативной и постоянной памяти, области памяти, стек, таблица векторов прерываний, память устройств подключенных к системной шине, подключение внешних устройств, разделение адресного пространства. Устройства ввода/вывода: особенности, обмен информацией, дополнительные устройства для организации обмена, функциональная схема подключения, предназначение основных блоков. Порты ввода/ввода, последовательная и параллельная организация, принцип работы, устройства интерфейса пользователя, устройства для длительного хранения информации, таймерные устройства.

2.5. Микроконтроллеры. Основы организации – 2 ч.

Понятие микроконтроллеров, основные элементы. Структура микроконтроллеров: классы микроконтроллеров (8, 16, 32 –х разрядные, сигнальные процессоры DSP), производители современных микроконтроллеров. Особенности микроконтроллеров (модульная организация, закрытая архитектура, типовые и расширенные функциональные периферийные модули). Типовая структура микроконтроллера. Процессорное ядро и изменяемый

функциональный блок. Предназначение основных элементов. Процессорное ядро, его характеристики. Процессоры с CISC-архитектурой, RISC-архитектурой – особенности, отличия, сравнение. Особенности организации памяти в микроконтроллерах: структуры с фон-неймановская (принстонская) и гарвардской архитектурой – особенности, отличия, сравнение. Система команд микроконтроллеров, ее особенности. Схема синхронизации и организации памяти микроконтроллеров, их особенности. Память программ, типы модулей памяти и их особенности. Память данных. Особенности хранения данных и программ. Регистровая и стековая память – предназначение и особенности. Внешняя память.

2.6. Внутренние и внешние связи в микроконтроллерах – 4 ч.

Порты ввода/вывода: параллельные и последовательные порты, типы портов, их предназначение, алгоритмы работы. Типичная схема двунаправленного порта ввода/вывода микроконтроллера. Таймеры и процессоры событий: предназначение, структура типичного 16-разрядного таймера/счетчика, основные недостатки данной схемы, пути усовершенствования данной схемы и современные направления. принцип действия канала входного захвата таймера/счетчика, его схема, типы сигналов, функциональная схема. Структура аппаратных средств канала выходного сравнения – основные сигналы, схема, принцип работы, аппаратные и программные усовершенствования. Модули процессоров событий – предназначение, принцип работы, основные сигналы, реализация режима широтно-импульсной модуляции. Модуль прерываний: принцип работы, источники прерываний, схема приоритетов.

2.7. Аппаратные средства микроконтроллеров – 4 ч.

Особенности режимов энергопотребления, минимизация данного режима: активный режим, режим ожидания, режим останова – особенности, предназначение. Тактовые генераторы микроконтроллеров: определения тактовой частоты генератора с помощью кварцевого резонатора, керамического резонатора и внешней RC-цепи, схемы подключения тактовых генераторов, используемые входы, сравнительная характеристика каждого способа подключения. Аппаратные средства обеспечения надежной работы микроконтроллера: схема формирования сигнала сброса, ее предназначение, основные сигналы, принцип работы, типовые схемы формирования сигнала внешнего сброса; блок детектирования пониженного напряжения питания: предназначение, особенности применения, принцип работы; сторожевой таймер: принцип действия, основные используемые сигналы, предназначение, особенности работы. Дополнительные модули, используемые в микроконтроллерах: модули последовательного и параллельного вода/вывода, задачи решаемые данными модулями, их типы, основы функционирования, протоколы интерфейса, современное состояние проблемы передачи информации через порты ввода/вывода; модули аналогового ввода/вывода, основные схемы, принцип работы, схема типового модуля АЦП, основы работы ЦАП и средства реализации данной функции в современных микроконтроллерах.

7 семестр (14 часов)

3.1. Введение: микроконтроллеры серии PIC и AVR – 2 ч.

Общие сведенья о микроконтроллерах серии PIC и AVR: состав и назначения семейств, их особенности; история появления и развития; особенности центрального процессора; отличительные особенности семейств микроконтроллеров; общие сведенья о сопутствующих программных продуктах. Технические характеристики отдельных подгрупп семейства. Представление микроконтроллера с программной точки зрения. Архитектура и характеристики базовых моделей отдельных подгрупп. Структурная схема базовых моделей отдельных подгрупп. Назначение выводов корпусов.

3.2. Принципы работы, организация памяти и особенности выполнения команд для микроконтроллеров PIC и AVR – 2 ч.

Принцип работы, временные диаграммы, схема тактирования и выполнения команды, принцип выборки команды. Организация памяти: программ и стека, памяти данных.

Особенности выполнения команд: выборка команд, выполнение команд условного и безусловного переходов, аппаратный стек.
Особенности методов адресации в микроконтроллерах PIC и AVR.

3.3. Организация обмена с внешними устройствами, память, прерывания для микроконтроллеров PIC и AVR – 2 ч.

Порты ввода-вывода, назначение, специфика, принцип организации, схемы портов и отдельных линий, особенности практического использования (программирование и электрическое соединение). Таймеры, структурная схема таймера(счетчика) для микроконтроллеров PIC и AVR, состав, предназначение, принцип работы, реакция на прерывания, особенности программного и аппаратного использования, структурные схемы возможных вариантов использования. Память данных, запись и чтение из памяти данных, используемые регистры и назначение отдельных битов в них. Примеры программного кода, поясняющие принцип работы с памятью данных. Организация прерываний, возможные виды прерываний, особенности, схема логики прерываний микроконтроллера, прерывания отдельных устройств.

3.4. Специальные функции и система команд микроконтроллеров PIC и AVR – 2 ч.

Набор специальных функций расширяющих возможности системы: сброс, сторожевой таймер, режим пониженного энергопотребления, выбор типа генератора, защита от кода считываний, биты идентификации, последовательное программирование. Система команд: перечень и формат команд отдельных моделей микроконтроллеров PIC и AVR, описание полей команд, основные форматы команд, количество циклов, изменяемые биты состояния. Специфичные команды: работы с байтами и битами, управления и работы с константами.

3.5. Особенности программирования и отладки, разработка программного кода для микроконтроллеров PIC и AVR – 2 ч.

Особенности программирования и отладки: особенности загрузки констант, арифметико-логических операций, конвейер команд, инструкции для организации ветвлений, стек, память программ и данных, ограниченность ресурсов. Разработка программного кода: различные ассемблеры, цели использования, принципы применения; компоновщики объектного кода; основной текст программы на ассемблере, метки, мнемоники, операнды, формат представления чисел, основные арифметические операторы, комментарии. Расширения файлов используемых при создании программного кода. Листинг. Директивы языка ассемблер, синтаксис при написании программного кода, поясняющие примеры.

3.6. Разработка программного обеспечения для микроконтроллеров PIC и AVR – 2 ч.

Особенности компоновщиков. Особенности менеджеров библиотек. Особенности симуляторов, примеры использования. Завершенные программные пакеты для создания, и исследования программного кода: примеры, принцип работы, использование, преимущества и недостатки, заключительные замечания.

3.7. Макет микропроцессорной системы и программирование простейших задач для микроконтроллеров PIC и AVR – 2 ч.

Описание макета, электрическая схема соединения, параметры основных элементов, предназначение и выполняемые функции. Особенности инициализации и запуска в работы. Примеры завершенных программ (текст программы, комментарии к ней, описание принципа работы): программа считывания состояния кнопки и вывода на светодиодный индикатор; программа считывания состояния кнопки и вывода на светодиодный индикатор при определенных условиях; программа для работы с семисегментным индикатором и кнопками; программа для работы вывода на семисегментный индикатор числа; подпрограммы формирования задержки; программа для работы со звуковым динамиком; программа для работы с мигающим светодиодом; программа для борьбы с дребезгом контактов.

1.2. Практические занятия

5 семестр (18 часов)

1.1. Представление информации в вычислительных машинах – 2 часа.

Системы исчисления. Способы перевода целых чисел. Способы перевода дробных чисел. Представление положительных и отрицательных чисел, чисел с плавающей точкой. Правила сложения двоичных чисел. Перевод в машинные коды. Арифметические операции над числами с фиксированной и плавающей точкой. Арифметические операции над двоично-десятичными кодами чисел.

1.2. Основы алгебры логики – 4 часа.

Упрощение логических функций. Составление таблиц истинности по логическим функциям. Получение логических функций по таблицам истинности (СДНФ, СКНФ, карты Карно). Построение схемы по таблице истинности. Составление логической функции по схеме. Реализация в различных базисах.

1.3. Основы представления информации и алгебры логики – 2 часа.

Контрольная работа №1.

1.4. Построение и применение простейших комбинационных устройств – 2 часа.

Синтез пороговой ячейки. Построение различных схем шифраторов и дешифраторов на базе простейших логических элементов. Реализация логических функций на мультиплексоре.

1.5. Комбинационные устройства средней степени интеграции – 2 часа.

Построение схемы дешифратора 3*8 из двух дешифраторов 2*4. Реализация логических схем на основе дешифратора и логических элементов указанного типа. Построение цифрового устройства формирующего заданные комбинации двоичных чисел на своих выходах.

1.6. Комбинационные устройства средней степени интеграции – 2 часа.

Контрольная работа №2.

1.7. Применение комбинационных устройств – 2 часа.

Синтез сегментных дисплеев различной конфигурации. Построение схемы выборки микросхем памяти (ОЗУ и ПЗУ) на основе дешифраторов различной структуры для микроконтроллера КР-580.

1.8. Последовательностные схемы – 2 часа.

Синтез цифровых устройств для обнаружения спада или установки сигнала на входе микросхемы. Разработка схем счетчиков с измененным коэффициентом пересчета. Техническое устранение иголок в выходном сигнале счетчиков с измененным коэффициентом пересчета.

6 семестр (18 часов)

2.1. Программные основы работы МП: система команд; команды пересылки и ввода вывода – 2 часа.

Обеспечить пересылку данных из памяти в порт. Обеспечить пересылку данных из порта в порт 02. Обеспечить пересылку данных из порта в ячейки памяти. Два варианта, один с привлечением LXI, второй с привлечением STA.

2.2. Функционирование МПС: адресация, особенности, регистры; размещение команд в памяти – 2 часа.

Расписать по байтам команды пересылки данных. Обеспечить декодирование команд. Разместить команды в памяти. Осуществить принудительный переход к следующему участку программы за счет команды пересылки в PC.

2.3. Программирование МП: арифметические и логические команды – 2 часа.

Обеспечить выполнение операции сложения, вычитания. В том числе с привлечением команд для выполнения операций с переносом. Выполнить операцию «И». Обсудить аспекты применения и получения результата с ней. Маскирование. Выполнить операцию «ИЛИ». Обсудить аспекты применения и получения результата с ней. Маскирование. Вы-

полнить операцию исключающее ИЛИ. Обсудить аспекты применения и получения результата с ней. Маскирование.

2.4. Программирование МП: контрольная работа – 2 часа.

Выполнить простейшую формулу. Исходные значения берутся из разных мест (по вариантам), так же различно места сохранения. Выполнить несколько задач на логические операции. Обеспечить по написанным программам кодирование в машинный код и размещение данных в памяти.

2.5. Программирование МП: команды переходов и вызовов подпрограмм – 2 часа.

Использовать команду сравнения для установки флагов. Повторить предназначение флагов. Обеспечить переход по условию сравнения двух чисел. Изучить команды вызова подпрограмм. Составить программу по реализации логических функций.

2.6. Программирование МП: обработка массивов значений – 2 часа.

Сложить элементы массива. Вычесть элементы массива. Найти произведение чисел. Найти количество положительных чисел в массиве.

2.7. Программирование МП: реализация управляющих воздействий – 2 часа.

Обсудить особенности подключения датчиков и исполнительных элементов к порту. Составить программу по обнаружению срабатывания одного/нескольких датчиков. Выработать управляющее воздействие в порт по факту срабатывания одного из датчиков. Составить программу по обнаружению срабатывания любых 3 из 8 датчиков подключенных к порту.

2.8. Программирование МП: реализация вычислительных процедур – 2 часа.

Составить программу по организации временной задержки. Составить программу по умножению чисел. Составить сложную программу по сортировке чисел.

2.9. Проектирование устройств на микроконтроллерах, КР – 2 часа.

Составление и решение задач связанных с проектированием МПСУ. Создание завершенных программ на основе ТЗ.

7 семестр (28 часов)

3.1. Создание макета МК системы в САПР – 4 часа.

Проектирование и реализация макета по исследованию работы простейшей МПСУ.

3.2. Программирование МК: опрос портов и вывод данных – 8 часов.

Считывание состояния кнопок. Зажигание и тушение светодиода

3.3. Программирование МК: использование дисплея отображения данных – 8 часов.

Работа с семисегментными индикаторами. Работа с LCD-дисплеями.

3.4. Программирование МК: формирование временных задержек и сигналов определенной длительности и частоты – 4 часа.

Аппаратная и программная задержка. Генерация звука. Динамический вывод.

3.5. Программирование МК: реализация специальных задач и функций – 4 часа.

Подавление дребезга. Работа с АЦП и ЦАП. Работа с различными интерфейсами

1.3. Лабораторные работы

5 семестр (18 часов)

1.1. Изучение базовых возможностей среды Electronics Workbench – 2 часа.

Работа в среде моделирования и методика проведения экспериментов.

1.2. Изучение логических схем и функций – 4 часа.

Исследование базовых логических элементов. Реализация логических функций при помощи логических элементов. Синтез логических схем.

1.3. Изучение работы шифраторов, дешифраторов и мультиплексоров – 4 часа.

Изучение принципов работы шифраторов, дешифраторов и мультиплексоров. Реализация логических функций с помощью мультиплексоров. Изучение способов применения дешифраторов.

1.4.Изучение работы триггеров – 4 часа.

Изучение структуры и исследование работы асинхронных и синхронных триггеров. Исследование функций переходов и возбуждения основных типов триггеров. Изучение взаимозаменяемости триггеров.

1.5. Изучение сумматоров, полусумматоров, регистров и счетчиков – 4 часа.

Исследование сумматоров и полусумматоров. Изучение структуры и исследование работы суммирующих и вычитающих счетчиков, счетчиков с измененным коэффициентом пересчета. Изучение регистров.

6 семестр (18 часов)

2.1. Изучение средств программирования и эмуляции микропроцессоров – 4 часа.

Изучение способов запуска, инициализации и настройки средств программирования и эмуляции микропроцессора. Рассмотрение возможностей и средств для отображения работы микропроцессора. Исследование способов программирования и отладки простейшего микропроцессора.

2.2. Запись и выполнение простых программ – 4 часа.

Знакомство с форматом команд и этапами их выполнения. Изучение команд пересылки данных и арифметических команд. Исследование простейших программ.

2.3. Организация циклов, обработка массивов и реализация логических функций – 4 часа.

Изучение логических команд, команд переходов и вызовов подпрограмм, команд ввода-вывода и работы со стеком. Рассмотрение алгоритмов решения сложных задач и правил их составления. Исследование программ для обработки массивов чисел, решения алгебраических и логических задач.

2.4. Реализация управляющих воздействий и вычислительных процедур – 4 часа.

Изучение принципов опроса внешних устройств и способов организации работы микропроцессора в режиме ожидания события. Рассмотрение принципов формирования управляющих сигналов и организации временной задержки. Изучение программной реализации типовых вычислительных процедур.

2.5. Проектирование устройств на микроконтроллерах – 2 часа.Разработка завершенных макетов и печатных плат изделий.

7 семестр (14 часов)

3.1. Изучение основ работы с контроллером Mega-128 – 2 часа.

Изучение команд ввода/вывода и логических операций. Изучение основ написания программ на ассемблере, реализующих логические функции. Моделирование работы программы в отладчике PLC. Компилирование программы в машинный код. Пересылка программ в контроллер при помощи программатора.

3.2. Изучение работы контроллера Mega-128 при реализации основных алгоритмов – 6 часов.

Изучить основные команды сравнения. Изучить работу таймера. Создать завершенные команды работы контроллера по: поддержанию уровня, по формированию задержек и заданной установке выхода в требуемое значение, формирования аварийной сигнализации по заданным условиям.

3.3. Изучение работы контроллера Mega-128 при реализации завершенных алгоритмов управления – 6 часов.

Создание программ поддержания уровня в баке в двух заданных диапазонах с учетом задания работы контроллера в автоматическом и в ручном режиме. Создание программ поддержания уровня в резервуаре в двух заданных диапазонах, с учетом задержки на включение и выключение в автоматическом и ручном режиме.

2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

2.1. Методические указания по освоению дисциплины

Изучение дисциплины студентами должно начинаться со знакомства с рабочей программой и выдаваемыми материалами: методическими пособиями и литературой в электронном формате, а также учебно-методическим комплексом для студентов (далее УМКД, формируемого на основе сборника учебно-методических материалов по дисциплине), в случае необходимости. УМКД это отдельно сформированный документ в электронном варианте содержащий абсолютно все необходимое, включая рабочую программу, методическое рекомендации, пояснения по работе с программным обеспечением и специальными средствами и стендами, задания для самостоятельного выполнения и пр.

Весь материал предварительно размещается на сайте (см. рабочую программу) и постоянной доступен, в том числе и в твердой копии на кафедре.

На первом занятии студенты обзорно знакомятся с планом проведения и методикой занятий, узнают конкретные требования к изучению дисциплины, им даются рекомендации, представленные в настоящей программе.

Студентам необходимо помнить, что качественна текущая подготовка и проработка материала является залогом успешного освоения предмета.

Студентам рекомендуется за один день до проведения соответствующих занятий познакомиться с планом работ, изучить рассматриваемые вопросы по рекомендуемой литературе и выполнить пункты самостоятельной работы.

После проведения занятий, в этот же день, повторить изученные теоретические положения, выполнить необходимые расчеты и примеры домашних заданий (РГР или лабораторных работ). При повторении материала желательно охватывать ранее рассмотренные вопросы; сначала более детально, затем ближе к концу семестра – обзорно.

Такая методика позволяет глубоко проработать все вопросы и не оставляет пробелы в знаниях. В итоге, к окончанию семестра, имеющиеся комплексные знания потребуется лишь освежить в памяти за 2-3 дня до итогового контроля (зачета).

Для подготовки к занятиям следует пользоваться литературой, указанной в разделе 10 рабочей программы, в том числе и электронным комплектном материалов. Для общей теоретической подготовки рекомендуется использовать источники п.10.1; при этом вспомогательными источниками п.10.2 необходимо пользоваться по мере необходимости. Для подготовки к практическим, самостоятельным и лабораторным работам рекомендуется использовать пособия, указанные ниже.

Привила проведения аттестации по результатам освоения дисциплины представлены в п.9 рабочей программы. Необходимо помнить при этом, что основой аттестационной оценки является результаты выполнения индивидуальных работ: лабораторных, домашних заданий, РГР и двух контрольных – самостоятельно и с полным осознанием выполненных процедур и их результатов.

2.2. Методические указания к лабораторным работам

5 семестр

Лабораторный практикум представляет собой четыре лабораторные работы, выполняемые по методическому пособию (доступному в электронном варианте, а также воспроизведенному в отдельном издании, см. п.5 раздела 7 рабочей программы; представленного так же в приложении А): Теличенко, Д. А. Цифровые узлы и элементы организации вычислительных систем. Лабораторный практикум / Д. А. Теличенко, М.В. Романова. – Благовещенск: Амурский гос. ун-т, 2004. - 104 с.

Каждая работа содержит необходимые теоретические сведения по исследуемой теме, задания для выполнения и контрольные вопросы. Помимо этого, в ряде работ приведены

упражнения для более глубокого освоения студентами изучаемого материала (данный комплект упражнения является расчетно-графической работой каждого студента).

Для выполнения лабораторных работ студентам предварительно предлагается самостоятельно ознакомиться с краткой теорией к каждой выполняемой работе и выполнять задания самостоятельно. Это даст необходимую теоретическую основу и облегчит выполнение работ, позволив на занятии уделить большее внимание вопросам, обычно вызывающим наибольшее затруднение. Сами задания в лабораторном практикуме расположены в возрастающем порядке сложности. По мере того как вырастает объем работ, которые студенты выполняют в аудитории, уменьшается количество заданий для самостоятельного решения (упражнений).

Все занятия делятся на два цикла: выполнение работы, защита работы. Циклы повторяются для каждой работы, в порядке следования, без нарушения очередности. Для улучшения качества усваиваемого материала не рекомендуется: совмещать в рамках проведения одного цикла разные темы исследования; проводить одновременное снятие и защиту работы. Допускается после выполнения очередного цикла всеми студентами группы в случае оставшегося времени уделить время на ликвидацию образовавшихся задолжностей, если таковые имеются.

На вводном занятии: студенты группы делятся на бригады по два – три человека, им присваиваются варианты, номера которых сохраняются за ними на протяжении всего курса. Каждый из студентов имеет два варианта: первый – личный (для заданий, требующих самостоятельного решения), второй – вариант на бригаду (для заданий, допускающих групповое выполнение).

На первом этапе цикла (снятия работы):

- преподавателем осуществляется допуск к работе, на котором проверяется: знание студентов краткой теории по выполняемой работе; наличие заготовки отчета;
- выясняются вопросы, вызвавшие у студентов затруднения, даются необходимые пояснения по ним;
- даются комментарии по методике проведения экспериментов;
- контролируется выполнение работы каждой бригады и всеми студентами в целом.

Работа считается снятой, если: студенты одной бригады, и каждый в отдельности, выполнили все задания работы, согласно вариантам; зафиксировали снятые данные в заготовку отчета.

На втором этапе цикла (защита работы):

- преподавателем, каждому из студентов, выдается произвольный вариант необходимый для выполнения упражнений (для лабораторных работ 1, 2, 3);
- каждый из студентов лично выполняет упражнения, согласно выданному на данной работе варианту (в случае если работа не защищается на одном занятии, варианты на упражнения изменяются);
- преподавателем проверяется личный отчет каждого из студентов, задаются вопросы по ходу выполнения работы; задаются контрольные вопросы (список вопросов приведен в лабораторном практикуме к каждой работе).

Работа считается защищенной, если: правильно выполнен отчет по работе; даны корректные ответы на вопросы преподавателя; правильно выполнены упражнения.

Представляемый отчет (после успешной защиты работы отчет сдается преподавателю и сохраняется до успешной сдачи студентом экзамена) должен удовлетворять следующим требованиям:

- отчет выполняется на одной стороне белого листа формата А4 в рукописной или печатной форме, в варианте возможном для прочтения (почерк, шрифт, размер, интервал);
- титульный лист должен содержать следующие сведения: название предмета; тему работы, с ее порядковым номером; фамилию студента, выполнившего работу с указанием номера группы и вариантов (личного и на бригаду); фамилию преподавателя, осуществляю-

щего прием работы; дату снятия и защиты (дата защиты заполняется преподавателем лично).

- основная часть работы должна содержать следующие сведения: краткую теорию; цель работы; элементы, приборы и инструменты, используемые в работе; ход работы с необходимыми рисунками, схемами, таблицами и формулами (необходимый перечень приведен в лабораторном практикуме по каждой работе).

В случае если студент не снял или не защитил работу, он может приступать к следующей работе. Ликвидировать возникшую задолженность можно на оставшемся времени после проведения очередной лабораторной работы или на дополнительных занятиях. Если ликвидировать задолженность по лабораторным работам в течение семестра не удается, студент является на экзамен с отчетами по несданным работам, где ему до ответа на экзаменационные вопросы дается возможность защитить каждую работу.

План проведения занятий с указанием последовательности изучаемых тем, объема часов, а также часов для самостоятельной работы представлен выше.

Помимо использования указанных в приложении А средств исследования, можно использовать аналогичные, представленные в виде самостоятельных лабораторных стендов, имеющихся на базе кафедры.

6 семестр

Лабораторный практикум представляет собой пять лабораторных работ, выполняемых по методическому пособию (доступному в электронном варианте, а также воспроизведенному в отдельном издании, см. п.1 раздела 7 учебной программы; представленного так же в приложении Б): Теличенко, Д.А. Микропроцессорные системы управления [Текст] : пособие к выполнению практ. и лаб. работ / Д. А. Теличенко ; АмГУ, Эн.ф. - Благовещенск : Изд-во Амур. гос. ун-та, 2013. - Ч. 1: Программирование простейших микропроцессоров. - 2013. - 100 с.

Каждая работа содержит необходимые теоретические сведения по исследуемой теме, задания для выполнения и контрольные вопросы.

Для выполнения лабораторных работ студентам предварительно предлагается самостоятельно ознакомиться с краткой теорией к каждой выполняемой работе и выполнять задания самостоятельно. Это даст необходимую теоретическую основу и облегчит выполнение работ, позволив на занятии уделить большее внимание вопросам, обычно вызывающим наибольшее затруднение. Сами задания в лабораторном практикуме расположены в возрастающем порядке сложности. По мере того как вырастает объем работ, которые студенты выполняют в аудитории, уменьшается количество заданий для самостоятельного решения (упражнений).

Примечание. Деление занятий на два цикла и правила приема работ аналогичны требованиям 5 семестра. Исключением является то, что здесь при защите работы не предусматривается решение задач, однако требования к знанию теории, в частности системы команд (изучаемых как в данной работе, так и вообще относящихся к рассматриваемой группе) более существенны и строги.

7 семестр

Особенностью лабораторных работ данного семестра является то, что они выполняются по заводскому руководству пользователя. Преподавателем в начале работы даются указания по выполнению и формулируются разные варианты. Это позволяет на финише обучения отказаться от типового алгоритма выполнения работ, дает студентам больше творчества и приучает выполнять задачи пользуясь минимум литературы с короткой постановкой задачи – что и имеется на практике.

При этом первые 2 темы выполняются на программе-эмуляторе. Последняя тема предполагает работу с реально действующим стендом. Здесь студенты самостоятельно без схем изучают его структуру и дают пояснения (после мозгового штурма) преподавателю как собрать схему по изучению установки. Ими самостоятельно разрабатываются схем сопряжения (в том числе и в натуре, с помощью пайки и реализации клеммных соедине-

ний). При этом изучаются и осваиваются не только основы создания завершенных программ для встраиваемых систем управления, но и основы реализации разных систем измерения, контроля и управления. Поэтому последняя тема, на которую выделяется 12 часов - одна из самых важных; без ее выполнения студентами в аудитории и под надзором преподавателя невозможна сдача итогового экзамена по дисциплине.

Помимо использования указанных здесь средств исследования, можно использовать аналогичные, представленные в виде самостоятельных лабораторных стендов, имеющихся на базе кафедры с соответствующим методическим обеспечением.

2.3. Методические указания к практическим работам

5 семестр

Предварительно студенты знакомятся со списком всех изучаемых тем, рассматриваемых на практических занятиях. Характер вопросов, прорабатываемых здесь, связан с лекционным курсом и графиком самостоятельной работы студентов. Большая часть времени уделяется решению конкретных задач (аналогичные по тематике задачи использованы в качестве третьего экзаменационного вопроса). Эти же задачи представлены в качестве РГР и задач выполняемых при защите лабораторных работ.

Задачи решаются студентами самостоятельно. При этом один из студентов вызывается к доске, решает поставленную задачу. Преподавателем контролируется не только правильность решения, но и даются: практические рекомендации по решению подобных заданий, применимость рассматриваемых тем к практике, а также предлагается другим студентам предложить аналогичные способы решения. Каждому вышедшему к доске, а так же студентам принявшим участие в обсуждении выставляется оценка.

Помимо этого, персонально каждый студент на практических занятиях выполняет две контрольные работы.

План проведения практических занятий, включая темы, объем часов представлен выше.

Краткие теоретические и учебно-методические материалы по каждой теме раскрывающие сущность задач представлены в методическом пособии. Здесь же представлены вопросы для обсуждения (ответы на них так же обсуждаются и на лабораторных работах).

6 и 7 семестр

Предварительно студенты знакомятся со списком всех изучаемых тем, рассматриваемых на практических занятиях. Характер вопросов, прорабатываемых здесь, является дополнением к материалу, изучаемому на лекциях и на лабораторных работах. Целью занятий является получение практических навыков по работе с микропроцессорными системами управления на базе современных микропроцессоров и микроконтроллеров. Основные задачи здесь следующие: в 6 семестре – получение навыков программирования простейших микропроцессоров; а в 7 семестре – обучение программированию RISC-микроконтроллеров.

План проведения практических занятий, включая темы, объем часов представлен выше. Краткие теоретические и учебно-методические материалы, по каждой теме раскрывающие сущность задач представлены в методических пособиях выдаваемых студентам в электронном варианте и представленных в перечне литературы, а так же в формируемом УМКД и выдаваемом в электронном варианте.

2.4. Методические указания по самостоятельной работе

5 семестр

Самостоятельная работа состоит в проработке ряда вопросов лекционного и практического курса в соответствии с п.5 и п.7 рабочей программы. При этом студентам предлагается законспектировать рассматриваемый вопрос, в случае необходимости задать возникшие вопросы на практическом занятии. Необходимый материал для этих целей содержится:

раздел 2 Теличенко Д. А., Романова М. В. Цифровые узлы и элементы организации вычислительных систем. Лабораторный практикум. Благовещенск: Амурский гос. ун-т, 2004.

*раздел 12 и 1*3 Бройдо, В. Л. Вычислительные системы, сети и телекоммуникации. Учеб. пособие: рек. Мин. обр. РФ / В. Л. Бройдо, О. П. Ильина. 3-е изд. – Спб.: Питер, 2008. – 766 с.

Укрупненный план вопросов, подлежащих проработке в рамках разделов 2, 12 и 13, представлен ниже.

Раздел 2 «Принцип действия ВМ. Логические основы, построение и работа простейших цифровых устройств».

Системы исчисления (позиционные и непозиционные). Перевод чисел из одной формы записи в другую.

Представление информации в ВМ (числа со знаком, с точкой, целые). Представление других видов информации.

Арифметические основы двоичной системы исчисления: правила сложения и вычитания двоичных чисел. Прямой и обратный коды двоичного числа. Умножение и деление.

Логические основы построения ВМ. Принципы и формы описания.

Алгебра логики – основные понятия и базовые операции.

Формы представления цифровых устройств (логическая функция, таблица истинности, схемное представление). Обозначения, принятые в нашей стране и за рубежом.

Основные законы алгебры логики. Минимизация логических функций и выражений. Понятие базиса, переход от одной формы представления к другой.

Простейшие цифровые узлы. Определение комбинационных схем. Комбинационные схемы средней степени интеграции (шифраторы, дешифраторы, мультиплексоры, сумматоры и вычитатели) – принцип работы, применение, способы представления.

Последовательностные схемы (схемы с памятью). Определение, предназначение. Триггеры и счетчики (виды, характеристики, основные достоинства и недостатки, различные схемы, временные диаграммы).

Основные принципы проектирования цифровых устройств. Этапы проектирования. Примеры применения цифровых устройств средней и малой степени интеграции. Современные тенденции развития.

Раздел 12 «Персональные компьютеры (ПК), особенности архитектуры и применения».

1. Функциональная и структурная организация класса ПК:

Микропроцессор, особенности, современные варианты выполнения. Разрядность, адресное пространство. Рабочая тактовая частота. Состав инструкций, конструктив, рабочее напряжение. Конвейерность, многозадачность работы, защищенный режим. Система виртуальных машин. Динамическое исполнение команд.

Особенности системной шины ПК.

Особенности основной памяти.

Особенности внешней памяти.

Источник питания и таймер.

Внешние устройства (их особенности): диалоговые средства пользователя, устройства ввода-вывода, устройства связи и телекоммуникации.

Дополнительные интегральные схемы: математический сопроцессор, контроллер DMA, сопроцессор ввода-вывода, контроллер прерываний.

Функциональные характеристики ПК.

2. Системная плата:

Состав и предназначение основных модулей, основные микросхемы.

Различные типы системных плат, тактовая частота системной шины, процессорные разъемы, понятие чипсета.

Северный и южный мост, разъемы для подключения памяти и их особенности.
Понятие BIOS, CMOS памяти.

3. Системный и периферийный интерфейс:

Определение интерфейса, его состав.
Шины расширений (ISA, PC/XT, PC/AT, EISA, MCA) – предназначение, характеристики, примеры использования;
Локальные шины (VLB, PCI, AGP) – предназначение, характеристики, примеры использования;
Периферийные шины (IDE, ATA, ATA-2, SATA, ATAPI, SCSI) – предназначение, характеристики, примеры использования, протоколы взаимодействия. Интерфейс RS-232 и стандарт IEEE 1284, последовательные и параллельные порты;
Универсальные последовательные периферийные шины. Универсальная шина USB. Технология Bluetooth. Стандарт IEEE 1394, цифровой последовательный интерфейс Fire Wire. Шина PCMCIA. Расширенный интерфейс ACPI.

4. Микропроцессорная память и кэш память в ПК – особенности (в сравнении с ранее рассмотренными вопросами).

5. Основная память ПК:

5.1 Общие замечания по организации: RAM и ROM, сравнение и особенности SRAM и DRAM. Сигналы управления RAS и CAS.
5.2 Физическая структура основной памяти. Матричная организация памяти в ПК. Кодовые шины адреса, дешифраторы полуадресов, сигналы записи/считывания, регистр данных. Куб памяти.
5.3 Типы модулей оперативной памяти: DIP, SIP, SIPP, SIMM, DIMM, RIM – особенности, основные области применения, сравнительные характеристики, основные частоты и пропускная способность.
5.4 Типы оперативной памяти: FPM DRAM, RAM EDO, BEDO DRAM, SDRAM, DDR SDRAM, DRDRAM – особенности, виды, характеристики, одноканальные и многоканальные модули, разрядность, тактовая частота, пиковая пропускная способность.

6. Постоянные запоминающие устройства ПК: постоянные запоминающие устройства (ПЗУ), программируемые запоминающие устройства (ППЗУ); перезаписываемые запоминающие устройства (EEPROM) – принципы работы, предназначение, особенности использования и реализации.

7. Внешние запоминающие устройства ПК – особенности. Понятие файловой системы. Режимы обмена между внешней памятью и оперативной памятью. Время доступа к информации в ПК. Технология SMART.

8. Особенности Флэш-памяти ПК: принцип работы, характеристики, надежность, конструктивные варианты исполнения: ATA Flash, (PC Card ATA), Compact Flash (CF), Smart Media (SM), xD-Picture, MultiMedia Card (MMC), Secure Digital Card (SD), Miniature Card (ViniCard), Memory Stick.

9. Дисковые массивы RAID. Особенности использования в ПК. Уровни конфигурации RAID. Дисковые массивы различных поколений.

Раздел 13 «Системное и прикладное программное обеспечение современных ВМ и МПС. Интерфейс пользователя».

1. Структура программного обеспечения: общее, или системное (general Software), и специальное, или прикладное (application or special Software) – предназначение и основные характеристики.

2. Общее (системное) программное обеспечение: операционные системы, система автоматизации программирования, комплекс технического обслуживания, пакеты программ дополняющих возможности операционной системы, системы документации.
2.1 Операционные системы – цели применения, набор программных модулей. Основные примеры операционных систем и их особенности (DOS, OS/2, UNIX, Windows). Структура

DOS: программа начальной загрузки, базовая система ввода-вывода (постоянный модуль, модуль расширения), базовый модуль DOS, командный процессор, утилиты.
2.2. Системы автоматизации программирования (инструментальные программные средства): языки программирования, языковые трансляторы, редакторы, средства отладки и другие вспомогательные программы – обзор основные особенности.
2.3. Комплекс технического обслуживания: проверочные тесты, наладочные тесты, диагностические тесты.
2.4. Пакеты программ дополняющих возможности и системы документации.

3. Специальное или прикладное обеспечение: пакеты прикладных программ: предназначение, и основные характеристики. Текстовые процессоры, редакторы широкого назначения, издательские системы, системы обработки электронных таблиц и табличные процессоры, графические редакторы, системы управления базами данных, графические редакторы, интегрированные системы.

Основной формой контроля проработки материала является опрос, проводимый при допуске к лабораторной работе, а так же результаты двух контрольных работ. Вопросы, прорабатываемые студентами самостоятельно, включены в зачетные билеты.

В целом выполнение самостоятельной работы базируется на использовании следующих учебно-методических пособий, доступных в необходимом количестве в библиотеке:

1. Теличенко Д. А., Романова М. В. Цифровые узлы и элементы организации вычислительных систем. Лабораторный практикум. Благовещенск: Амурский гос. ун-т, 2004, 104 с.

2. Теличенко Д. А., Бушманов А. В. Схемотехника. Лабораторный практикум. Благовещенск: Амурский гос. ун-т, 2006, 93 с. (рек. ДВ РУМНЦ).

Вспомогательной литературой могут служить источники указанные п.10.

Помимо самостоятельной работы связанной с проработкой теоретического материала и подготовкой к лабораторным работам, в рамках учебной программы, предусмотрено индивидуальное выполнение домашних заданий (РГР). Задания РГР аналогичны упражнениям, представленным в методическом пособии. Каждый студент самостоятельно согласно варианту (номер варианта выдается преподавателем) выполняет соответствующие упражнения. Расчетно-графическая работа оформляется в виде завершенного документа (согласно требованиям стандарта университета), снабженного необходимыми решениями, построениями и пояснениями. Защита РГР персонально каждым студентом происходит по завершению курса, перед экзаменом на назначенной консультации. Темы соответствующих заданий представлены ниже:

Задание 1. Получение логических функций. По заданной таблице истинности (см. табл. 6.1.а):
1.1. получить логическую функцию в виде СДНФ;
1.2. получить логическую функцию в виде СКНФ;
1.3. упростить логическую функцию, полученную в виде СДНФ;
1.4. упростить логическую функцию, полученную в виде СКНФ.

Задание 2. Построение структурных схем. По заданной логической функции (ЛФ) необходимо:
2.1. построить структурную схему в произвольном базисе (ЛФ из 1.1 и 1.2);
2.2. упростить заданную логическую функцию и простроить схему в произвольном базисе (ЛФ из 1.3 или 1.4);
2.3. простроить схему упрощенной логической функции в базисе «ИЛИ-НЕ» (ЛФ из 1.3 или 1.4),
2.4. простроить схему упрощенной логической функции в базисе «И-НЕ» (ЛФ из 1.3 или 1.4).

Задание 3. Составление логических функций. По заданной схеме (см. рисунки после таблицы 6.1.а):
3.1. составить логическую функцию;

3.2. упростить логическую функцию.

Задание 4. Применение дешифраторов. Сформируйте сигнал выбора определенной микросхемы памяти в микроконтроллере с использованием стандартных дешифраторов (см. табл. 6.1.б).

Задание 5. Применение мультиплексора. По заданной логической функции (ЛФ):

5.1. постройте схему с использованием мультиплексора (ЛФ из 1.1 и 1.2);

5.2. упростите логическую функцию и постройте схему ее реализации на мультиплексоре 3х8 (ЛФ из 1.3 или 1.4).

Задание 6. Построение временных диаграмм. По заданной таблице истинности некоторой логической функции (см. табл. 6.1.а) постройте временную диаграмму. Считайте, что переключение в новое состояние схемы должно происходить по:

- отрицательному фронту импульса *T*;
- положительному фронту импульса *T*.

Задание 7. Применение комбинационных устройств для реализации цифровых автоматов. По заданной логической функции (аналогичной заданию 5):

- с использованием дешифратора (с активным уровнем сигнала на выходе «0») и произвольных логических элементов;
- с использованием дешифратора (с активным уровнем сигнала на выходе «0») и логических элементов «И-НЕ»;
- с использованием дешифратора (с активным уровнем сигнала на выходе «0») и логических элементов «ИЛИ-НЕ»;
- с использованием дешифратора (с активным уровнем сигнала на выходе «1») и произвольных логических элементов;
- с использованием дешифратора (с активным уровнем сигнала на выходе «1») и логических элементов «И-НЕ»;
- с использованием дешифратора (с активным уровнем сигнала на выходе «1») и логических элементов «ИЛИ-НЕ».

6 и 7 семестр

Самостоятельная работа состоит в проработке ряда вопросов лекционного и практического курса в соответствии с п.5 и п.7 настоящей программы.

В рамках подготовки к лабораторным работам, зачету или экзамену студентам предлагается законспектировать рассматриваемый вопрос, в случае необходимости задать возникшие вопросы на ближайшем занятии или консультации. Основной формой контроля проработки данного материала является опрос, проводимый при допуске к лабораторной работе. Косвенной оценкой служат результаты контрольных работ (тестов). Темы, прорабатываемые студентами самостоятельно, включены в экзаменационные (зачетные) билеты. Выполнение данного вида самостоятельной работы базируется на использовании следующих учебно-методических пособий (наглядного материала или соответствующих руководств), доступных в необходимом количестве:

1. Теличенко, Д.А. Микропроцессорные системы управления [Текст] : пособие к выполнению практ. и лаб. работ / Д. А. Теличенко ; АмГУ, Эн.ф. - Благовещенск : Изд-во Амур. гос. ун-та, 2013. - Ч. 1: Программирование простейших микропроцессоров. - 2013. - 100 с.

2. Микроконтроллер Mega-128. Системное руководство. Электронный вариант.

В рамках выполнения курсового проекта (7 семестр) студентами необходимо пользуясь методическим пособием и другой доступной литературой (см. ниже) осуществить разработку следующей общей для всех темы: «Разработка микроконтроллерной системы управления». Каждый студент самостоятельно согласно варианту (номер варианта выдается преподавателем) выполняет соответствующие задания, строго в соответствии с имеющимся графиком. Задания индивидуальны и модифицируются каждый учебный год. Предусматривается поэтапная предварительная сдача каждого из разделов (контроль преподавателем). Проект оформляется в виде завершенного документа (согласно требовани-

ям стандарта университета), снабженного необходимыми решениями, построениями и пояснениями. Оформление предполагает создание чертежей, являющихся как приложением к тексту записки, так и самостоятельными документами, которые оформляются согласно требованиям единой системы конструкторской документации. Защита проекта персонально каждым студентом происходит перед комиссией, по индивидуальному графику. Темы соответствующих разделов и короткие комментарии к ним (курсивом) представлены ниже; весь остальной материал представлен в пособии «Проектирование микропроцессорных систем».

Краткое содержание разделов курсового проекта представлено ниже.

Содержание курсового проекта

Реферат

Выполняется согласно требованиям к курсовому проектированию Амурского государственного университета и нормами русского языка.

Введение

Дается краткая характеристика микроконтроллеров с рассматриваемой архитектурой, преимущества данной архитектуры, область применения микроконтроллеров, и т.п.

1. Концептуальное проектирование устройства

1.1. Постановка и описание задачи разработки

Приводится полное словесное описание разрабатываемой проблемы. Описание не должно повторять «задание», а должно представлять собственное видение поставленной задачи. Данный раздел должен показать насколько глубоко понимается сама проблема разработки. Здесь возможно так же изложение некоторых особенностей и способов решения задач, характеристика представленных этапов выполнения работы и их предназначение (что планируется выполнить и каким образом изложить).

1.2. Разработка исходной структурной схемы устройства

Данная структурная схема используется в дальнейшем (детализируется на каждом уровне разработки). Схема должна содержать полноценные пояснения по имеющимся на ней элементам: для чего нужны, какими обязательными характеристиками должны обладать и т.п. Эта структура принципиальная, и на ней нет типов элементов и точного подключения. Сама схема разрабатывается на основе анализа пункта 1.1.

1.3. Технические требования к элементам системы

Приводятся необходимые или обязательные параметры, которым должно удовлетворять устройство (если не представлены в п.1.2); анализируется, каким образом это может повлиять на выбор решений и на каком этапе разработки должно учитываться. Базой для этого пункта служит схема п.1.2. Окончательный вывод оформляется в виде таблицы для входов-выходов микроконтроллера и их использовании для подключения периферии. Здесь необходимо так же представить конкретизированную схему устройства. Основная задача этого пункта представить и сформулировать характеристики, необходимые для выбора конкретных типов и марок устройств.

1.4. Направление решения поставленной задачи и выбор основных элементов

Описывается принцип предлагаемого вами подхода к решению задачи, альтернативы (как по аппаратной, так и по программной части). Здесь обосновывается выбор конкретных марок основных элементов проектируемой системы, в том числе и микроконтроллера. Каждый элемент дожжен быть выбран обоснованно и предложены (рассмотрены) альтернативы.

1.5. Техническое задание на разработку

Техническое задание разрабатывается согласно требованиям ГОСТ (см. «Источники информации и их характеристика»). Сам текст Технического задания приводится в приложении. В данном пункте только описываются и характеризуются разделы технического задания, а именно почему именно так они были составлены Вами. Возможно написание технического задания с разной степенью детализации (можно сразу уточнить

контроллер, можно оставить этот вопрос на проработку). Формулировка пунктов должна быть именно как у Задания: «проработать», «рассмотреть» и т.п. Желательно составление задания как на «разработку устройства, предназначенного для решения конкретной технической задачи» (выбирается самостоятельно). В целом возможно составления задания, как на программный продукт, так и на автоматизированную систему (второе предпочтительнее).

2. Разработка аппаратной части проектируемой системы

2.1. Принципиальный алгоритм решения задачи

В данном пункте необходимо проанализировать полученную структуру в 1.2 и 1.3, с учетом 1.4, и предложить алгоритм работы программы. Разрабатываемый алгоритм дожжен быть «принципиальным» т.е. без деталей. Основная задача этого раздела принципиально продумать будущую аппаратную реализацию устройства с точки зрения функционирования программы. Алгоритм в обязательном порядке, должен конкретизировать: какие порты опрашиваются (вплоть до наименования линий) и что к ним предлагается подключить; на какие порты подается информация и для каких целей; какие предварительные действия (настройки, задание конфигурации и т.д.) надо осуществить.

2.2. Разработка полной электрической схемы устройства

Здесь на уровне каждого элемента (кнопки, дисплея, контроллера, индикатора, резистора и т.п.) представляется общая схема устройства с точки зрения наличия основных компонентов и их взаимосвязи. Схема вычерчивается строго в соответствии с ГОСТ (см. «Источники информации и их характеристика»), с учетом п.2.1 и приводится в приложении. По тексту самого пункта описываются принципы подключения каждого элемента, пояснения снабжаются небольшими рисунками. Возможно, в данном пункте представить необходимый расчет: частоты, потребляемой мощности и других параметров. Подключение всех элементов должно быть обосновано либо расчетами, либо ссылками на источники, где представлена такая информация. Разрабатываемая схема так же выносится на лист. Так же представляется корректная спецификация ко всем элементам системы.

2.3 Детальная информация по аппаратной реализации системы

Здесь приводятся основные технические характеристики выбранных элементов, которые не были представлены ранее, в том числе и та информация, которая может быть использована позднее – например, корпуса элементов и их «распиновка». Объем пункта не больше 2-3 стр.

3. Принципиальная схема модели устройства

Здесь разрабатывается и приводится схема полученного устройства, которая может быть использована для реализации в программе-эмуляторе (например «Proteus»). Схема выносится на лист. Здесь используются обозначения элементов принятые в программе-эмуляторе. В большинстве случаев такая схема проще, чем разработанная в п.2.2.

В самом разделе внимание уделяется особенностям реализации модели, по сравнению со схемой п.2.2, чем можно пренебречь, что сделать проще, а что необходимо уточнить и по-другому реализовать. Все особенности описываются и излагаются.

Возможно изложение этапов создания модели в программе-симуляторе.

4. Разработка программной части проектируемой системы

4.1. Разработка полного алгоритма работы программы

Здесь разрабатывается полный алгоритм программы, в общем случае отличный от п.2.1. Сам алгоритм составляется согласно требованиям ГОСТ (см. «Источники информации и их характеристика»). Полностью алгоритм приводится только в приложении (и на листе). По тексту записки приводится его описание, снабженное рисунками (отдельные части большого алгоритма) и пояснениями. Рекомендуется начать с относительно небольшого укрупненного алгоритма (для всей задачи), состоящего из нескольких частей, каждая из которых выполняет функционально завершенное действие (подзадачу). Каж-

дую из этих частей необходимо раскрыть и описать по тексту, снабдив рисунками и схемами. Описываемые части алгоритма выбираются таким образом, что бы они были независимыми и неповторяющимися (по ним в дальнейшем разрабатываются фрагменты программного кода).

4.2. Разработка исходного текста программы

В рамках данного пункта разрабатывается программа на ассемблере. Полный текст программы (вернее полученный в результате компилирования листинг – см. п.4.3) приводится только в приложении. Каждая строка программы должна иметь комментарий. По тексту записки приводятся фрагменты кода, и даются пояснения по принципу их реализации. Фрагменты выбираются в порядке и строго в соответствии с частями алгоритма, описанными в разделе 4.1.

4.3. Компиляция и отладка программы

Описывается принцип компиляции и создание завершенного программного продукта с помощью выбранных Вами средств. Приводится так же вариант создания перемещаемого объектного кода (при котором создается отдельный модуль, для дальнейшего использования в других программах). Приводятся полученные в ходе компиляции листинги (в приложении), и тому подобные файлы.

В обязательном порядке описывается принцип отладки и имитационного исследования программы, осуществляемый с помощью специальных программных средств, например MPSIM. Исследование проводится без привлечения аппаратной части раздела 2. Исследуются два варианта «без» и «с» привлечением средств раздела 3. Отладка программы должна быть проведена в любом случае и снабжена поясняющими рисунками и описанием имеющихся здесь возможностей.

5. Тестирование работы проектируемой системы

Приводятся результаты моделирования схемы устройства с записанной в микроконтроллер программой. Исследования проводятся в программе-эмуляторе (например «Proteus»). Поясняющие рисунки и схемы, временные диаграммы представляются на листах, по тексту пункта или по желанию - в приложении. В самом пункте излагается словесное описание проведенных исследований, и формулируются рекомендации к пользователям разработки. В данных рекомендациях необходимо грамотно описать имеющиеся особенности работы полученного устройства, и какие действия надо совершить для выполнения поставленных задач.

6. Расчет и создание монтажной схемы

Описываются принципы трассировки, требования к размещению элементов и компоновке печатной платы. Отдельно рассматриваются особенности автоматической трассировки с помощью специальных программных средств. Окончательный вариант печатной платы (выполненной строго в соответствии с регламентирующими документами), скорректированный вручную, в виде двух независимых проекций, снабженных подписями к каждой монтажной единице, приводится в приложении (и на листе), там же приводится спецификация к изделию. Спецификация может быть отличной от разработанной в рамках пункта 2 и учитывать особенности монтажа элементов на плате.

Предполагается что вся разработка, включая (в случае наличия) дисплеи, кнопки и индикаторы реализуется на одной плате. В исключительных (и обоснованных в данном разделе) случаях допускается изготовление нескольких плат (схемы всех монтажных плат приводятся в приложении и на листах) с проработкой вопросов сопряжения схем друг с другом. Печатная плата в обязательном порядке должна быть снабжена размерами элементов и выполнена в масштабе.

Заключение

Делаются заключительные выводы о работе устройства, приводятся возможные рекомендации по дальнейшему использованию разработки. В случае необходимости возможны рекомендации по тематике проекта (работы) или варианты работ, которые можно проделать в будущем.

Список используемой литературы

Приводится полный перечень используемой литературы. Возможно указание собственноручно обнаруженных источников, не выданных ранее. При этом текст записки должен обязательно содержать корректные ссылки на указанные источники. Количество источников должно удовлетворять не только требованиям стандарта университета, но и в полной мере характеризовать глубину проработки работы.

Приложения:

А. Техническое задание на разработку

Б. Детальная электрическая схема устройства со спецификацией

В. Полный алгоритм программы

Г. Листинг кода программы

Д. Монтажная схема со спецификацией к используемым элементам

Е. Материалы по желанию

Содержание графической части курсового проекта

Представленные рекомендации касаются разработки графической части при выполнении курсового проекта. Общее количество листов – 3, формата А1. Рекомендации по размещению материала ориентировочные, указанный перечень выносимого материала может быть дополнен, либо изменено размещение материала по листам.

Однако указанные ниже материалы должны в обязательном порядке присутствовать на листах.

Лист 1: Детальная электрическая схема устройства (полученная по факту выполнения раздела 2, идентичная приложению Б). Алгоритмическая схема разработанной программы (полученная по факту выполнения раздела 4, идентичная приложению В).

Лист 2: Схема модели устройства и полученные результаты моделирования (полученные по факту выполнения разделов 3, 5 и возможно 6 – в части разработки других моделей для дальнейшей трассировки).

Лист 3: Монтажная схема и спецификации к устройству (полученные по факту выполнения раздела 6, идентично приложению Д).

ПРИЛОЖЕНИЕ А

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ 5 СЕМЕСТРА

ВВЕДЕНИЕ

Лабораторный практикум представляет собой пять лабораторных работ, по курсу «Вычислительные машины, сети и системы». Каждая лабораторная работа содержит необходимые теоретические сведения по исследуемой теме, задания для выполнения и контрольные вопросы. Помимо этого в ряде работ приведены упражнения для более глубокого освоения студентами изучаемого материала.

Пособие охватывает следующие темы: основы алгебры логики; базовые логические элементы; комбинационные логические схемы; шифраторы, дешифраторы, мультиплексоры, и их применение в ЭВМ; триггеры; сумматоры, полусумматоры, регистры и счетчики; основы компьютерных сетей и настройка сетевой операционной системы.

Основные исследования в лабораторных работах предлагается выполнять в программе «Electronics Workbench» – удобной и гибкой среде компьютерного моделирования. Однако это не накладывает ограничение на то, что бы данные работы были проделаны и в других системах моделирования. Для этого в пособии помимо обозначения электронных элементов принятого в нашей стране, описаны обозначения принятые на западе.

Для выполнения лабораторных работ по данному пособию студентам предлагается предварительно и самостоятельно ознакомится с краткой теорией к каждой выполняемой работе. Это даст необходимую теоретическую основу и облегчит выполнение лабораторных работ, позволив на занятии уделить большее внимание вопросам, обычно вызывающим наибольшее затруднение.

Сложность заданий в лабораторном практикуме расположена в возрастающем порядке. По мере того, как вырастает объем работ, которые студенты выполняют в аудитории, уменьшается количество заданий для самостоятельного решения (упражнений).

Теоретической и практической базой данного пособия служит книга Электротехника и электроника в экспериментах и упражнениях, практикум на Electronics Workbench, под общей ред. Д. И. Панфилова. Суть постановки экспериментов и теоретический материал некоторых лабораторных работ были взяты именно оттуда.

ЛАБОРАТОРНАЯ РАБОТА № 1

Изучение логических схем и функций

Цель работы:

1. Исследование базовых логических элементов.
2. Реализация логических функций при помощи логических элементов.
3. Синтез логических схем.

Приборы и элементы:

Логический преобразователь (панель «Instruments/Logic Converter»)
Генератор слов (панель «Instruments/Word Generator»)
Вольтметр (панель «Indicators/Voltmeter»)
Логические пробники (панель «Indicators/Red probe»)
Источник напряжения + 5 В (панель «Basic/Pull-Up Resistor»)
Земля (панель «Sources/Ground»)
Двухпозиционные переключатели (панель «Basic/Switch»)
Двухвходовые элементы И, И-НЕ, ИЛИ, ИЛИ-НЕ (панель «Logic Gates/2-Input AND, NAND, OR, NOR Gates»)
Микросхемы различных серий (панель «Logic Gates/»).

Краткие теоретические сведения

Любые цифровые микросхемы современных вычислительных машин строятся на основе простейших логических элементов «И», «ИЛИ», «НЕ» и их комбинаций. В настоящее время используется несколько технологий построения логических элементов:

- транзисторно-транзисторная логика (ТТЛ, TTL)
- логика на основе комплементарных МОП транзисторов (КМОП, CMOS)
- логика на основе сочетания комплементарных МОП и биполярных транзисторов (БиКМОП)

Прежде чем приступить к изучению базовых логических элементов введем основные понятия булевой алгебры или алгебры логики, на которой базируется все теоретическое обоснование работы цифровых устройств.

1. Основные определения и аксиомы алгебры логики

Переменные, рассматриваемые в алгебре логики, принимают только два значения – «0» или «1». Чаще всего сами переменные обозначаются латинскими буквами: либо малыми (x, y, z, …) или большими (A, B, C,…).

Примечание. Физически самым простым устройством, моделирующим поведение любой булевой переменной, является двух позиционный переключатель. У него за логический сигнал «1» принимается положение «включено», а за сигнал «0» положение «выключено». Поэтому самой простой схемой для изучения логических функций и выражений является электрическая схема, состоящая из исследуемого элемента или элементов, источников питания, заземления и устройств, моделирующих задание уровней логических сигналов (в простейшем случае ими являются переключатели). Причем в данном случае важно подчеркнуть неразрывную связь между физическим сигналом (ток или

напряжение) и логическим сигналом: за логический сигнал «0» и «1» принимаются определенные уровни физических сигналов (чаще за логический «0» принимается сигнал по напряжению в 0 В, а за уровень логической «1» +5 В).

Ниже кратко перечислены основные операции и аксиомы алгебры логики.

Основные операции алгебры логики:

отношение эквивалентности, обозначается знаком «=»;

операция логического сложения (дизъюнкция), обозначаемая знаком «$\vee$» или «+»;

операция логического умножения (конъюнкция), обозначаемая знаком «$\wedge$» или «$\bullet$»;

операция отрицания (инверсии), обозначаемая надчеркиванием или апострофом «'».

Основные аксиомы булевой алгебры:

$$\begin{cases} \overline{0} = 1, \\ \overline{1} = 0. \end{cases} \qquad \begin{cases} 0 \vee 0 = 0 + 0 = 0, \\ 1 \vee 0 = 1 + 0 = 1, \\ 1 \vee 1 = 1 + 1 = 1. \end{cases} \qquad \begin{cases} 0 \wedge 0 = 0 \cdot 0 = 0, \\ 0 \wedge 1 = 0 \cdot 1 = 0, \\ 1 \wedge 1 = 1 \cdot 1 = 1. \end{cases} \qquad (1)$$

2. Логические выражения

Из логических переменных с помощью базовых логических операций можно составить логическое выражение. Логические выражения связывают значение логической функции со значениями логических переменных.

Логическое выражение является одним из способов описания цифрового устройства.

Запись логических выражений обычно осуществляют в конъюнктивной или дизъюнктивной нормальных формах. В дизъюнктивной форме логические выражения записываются как логическая сумма логических произведений, в конъюнктивной форме – как логическое произведение логических сумм.

3. Логические тождества

При преобразованиях логических выражений используются следующие логические тождества:

$$\begin{aligned} &\overline{\overline{x}} = x, \quad x \vee 1 = 1, \quad x \vee 0 = x, \quad x \wedge 1 = x, \quad x \wedge 0 = 0, \quad x \vee x = x, \quad x \vee x \wedge y = x, \\ &(x \wedge y) \vee (x \wedge \overline{y}) = x, \quad (x \vee y) \wedge (x \vee \overline{y}) = x, \quad x \vee (\overline{x} \wedge y) = x \vee y, \\ &\overline{x \wedge y} = \overline{x} \vee \overline{y}, \quad \overline{x \vee y} = \overline{x} \wedge \overline{y}. \end{aligned} \qquad (2)$$

4. Логические функции

Любое логическое выражение, составленное из n переменных x_n, x_{n-1}, …x_1, с помощью конечного числа операций алгебры логики, можно рассматривать как некоторую функцию n переменных. Такую функцию называют логической. В соответствии с аксиомами алгебры логики такая функция может принимать (в зависимости от значения логических переменных входящих в

нее) значение «0» или «1». Функция n логических переменных может быть определена для 2^n значений переменных, соответствующих всем возможным значениям n-разрядных двоичных чисел. Основной интерес представляют следующие базовые логические функции двух переменных x и y:

$f1(x, y) = x \cdot y$ – логическое умножение (конъюнкция), (3)

$f2(x, y) = x \vee y$ – логическое сложение (дизъюнкция), (4)

$f3(x, y) = \overline{x \cdot y}$ – логическое умножение с инверсией, (5)

$f4(x, y) = \overline{x \vee y}$ – логическое сложение с инверсией, (6)

$f5(x, y) = x \oplus y = x\bar{y} \vee \bar{x}y$ – суммирование по модулю 2, (7)

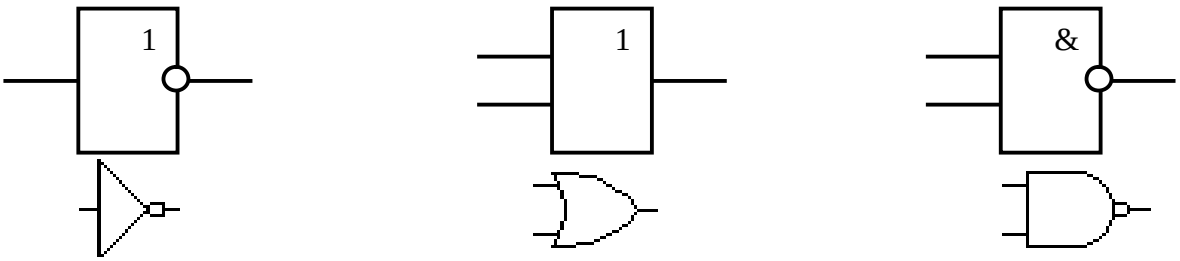
$f6(x, y) = \overline{x \oplus y} = xy \vee \overline{xy}$ – равнозначность. (8)

5. Логические схемы

Физическое устройство, реализующее одну из операций алгебры логики или простейшую логическую функцию, называется *логическим элементом*. Схема, составленная из конечного числа логических элементов по определенным правилам, называется логической схемой. Логическая схема наряду с логическим выражением является одним из способов задания цифровых элементов.

Условное обозначение основных логических элементов принятых в нашей стране и за рубежом приведено на рис. 1.

Условное обозначение основных логических элементов



Инвертор (НЕ) ИЛИ И-НЕ

Рис. 1 – Обозначение логических элементов

Примечание: количество входов в приведенных элементах может быть произвольным; на операцию инвертирования указывает кружок на выходном сигнале.

Здесь на рис. 1 в первом ряду приведено обозначение, принятое в нашей стране, во втором ряду, обозначение, принятое за рубежом.

6. Таблица истинности

Так как область определения любой логической функции n переменных конечна (2^n значений), такая функция может быть задана таблицей значений $f(v_i)$ которые она принимает в точках v_i, где $i = 0,1...2^n-1$. Такие таблицы назы-

вают таблицами истинности. В таблице 1 представлены таблицы истинности, задающие указанные выше функции (3) – (8).

Таблица 1 – Произвольная таблица истинности

№	Значения переменных		Функции					
	x	y	f1	f2	f3	f4	f5	f6
1	0	0	0	0	1	1	0	1
2	0	1	0	1	1	0	1	0
3	1	0	0	1	1	0	1	0
4	1	1	1	1	0	0	0	1

В общем случае таблица истинности должна содержать все возможные комбинации логических переменных, входящих в логическое выражение (в случае таблицы 1 это комиции логических переменных *x* и *y*), и значения логического выражения, соответствующие каждой комбинации логических переменных

Таблица истинности является третьим способом задания цифровых элементов. Необходимо отметить, что все три способа задания (с помощью логических выражений, логических схем и таблиц истинности) являются однозначными и в равной мере взаимно заменяемыми. Так по логическому выражению можно составить схему и записать таблицу истинности и наоборот.

7. Получение логических выражений

Целю преобразования сложных логических выражений (а так же получения по экспериментальной таблице истинности логического выражения, описывающего данную логическую зависимость) является получение компактного логического выражения, которое в полной мере описывает данную логическую зависимость любого числа переменных. Операции с логическими выражениями намного удобней, чем с таблицами истинности и со схемами.

Самый простой способ получения логических выражений это анализ таблицы истинности. Для наглядности рассмотрим пример: пусть требуется найти логическое выражение для функции *fm* трех переменных *X, Y, Z,* описываемой таблицей истинности 2.

Таблица 2 – Таблица истинности

N	X	Y	Z	fm
1	0	0	0	0
2	0	0	1	0
3	0	1	0	0
4	0	1	1	1
5	1	0	0	0
6	1	0	1	1
7	1	1	0	1
8	1	1	1	1

Из правил записи логических выражений следует, что искомую зависимость можно получить в двух видах (конъюнктивной или дизъюнктивной нормальных формах – пункт 2). Чаще всего на практике применяется дизъюнктивно нормальная форма записи логической функции.

Дизъюнктивно нормальная форма записи представляет собой дизъюнкцию (логическое сложение) элементарных конъюнкций (произведение всех логических переменных, в нашем случае – это три переменные *X, Y, Z*).

Для записи логического выражения в дизъюнктивно нормальной форме выберем из таблицы истинности 2 строки, в которых функция *fm* принимает значение «1». Это строки 4, 6, 7, 8. Таким образом элементов нашей дизъюнкции будет 4 (всего слагаемых). Каждое слагаемое это элементарная конъюнкция. В каждую же элементарную конъюнкцию (произведение), как было сказано выше, должны войти все логические переменные данной строки (в нашем случае *X, Y, Z*). При этом если соответствующий элемент (X, Y или Z) имеет для данной строки значение «1», то он входит в элементарную конъюнкцию (произведение) в обычном виде, если же он равен «0», то необходимо взять его отрицание. Так для каждой строки имеем элементарную конъюнкцию (произведение) в следующем виде:

$\overline{X} \cdot Y \cdot Z$ для строки 4;
$X \cdot \overline{Y} \cdot Z$ для строки 6;
$X \cdot Y \cdot \overline{Z}$ для строки 7;
$X \cdot Y \cdot Z$ для строки 8;

Искомой формой записи данного логического выражения в дизъюнктивно нормальной форме записи является дизъюнкция (сложение) полученных выше 4 конъюнкций:

$$fm = \overline{X}YZ \vee X\overline{Y}Z \vee XY\overline{Z} \vee XYZ \qquad (9)$$

По данной логической функции можно записать таблицу истинности, идентичную таблице 2. Для этого необходимо в логическую функцию подставлять все возможные комбинации логических переменных *X, Y, Z* (всего таких значений 2^3) и на каждой подстановке, пользуясь аксиомами алгебры логики вычислять функцию *fm*.

Необходимо отметить, что данный способ получения логических выражений не является самым рациональным. Полученная логическая функция, хоть и содержит исчерпывающую информацию о выбранной комбинации логических переменных, тем не менее, поддается еще упрощению с помощью выражений (2).

Другим способом составления упрощенного выражения по таблице истинности является составление карт Карно. Данный способ в данной работе не рассматривается по двум причинам:

во-первых, наглядность данного метода минимальна при достаточно большой сложности построения самих карт Карно, что вызывает большие трудности у студентов, особенно в начале изучения цифровой техники;

во-вторых, методика построения карт Карно хорошо описана в литературе по цифровой технике и электронике.

Порядок работы

Задание 1. Исследование логической функции «И»

а) Задание уровней логических сигналов

Создайте схему, изображенную на рис. 1.а

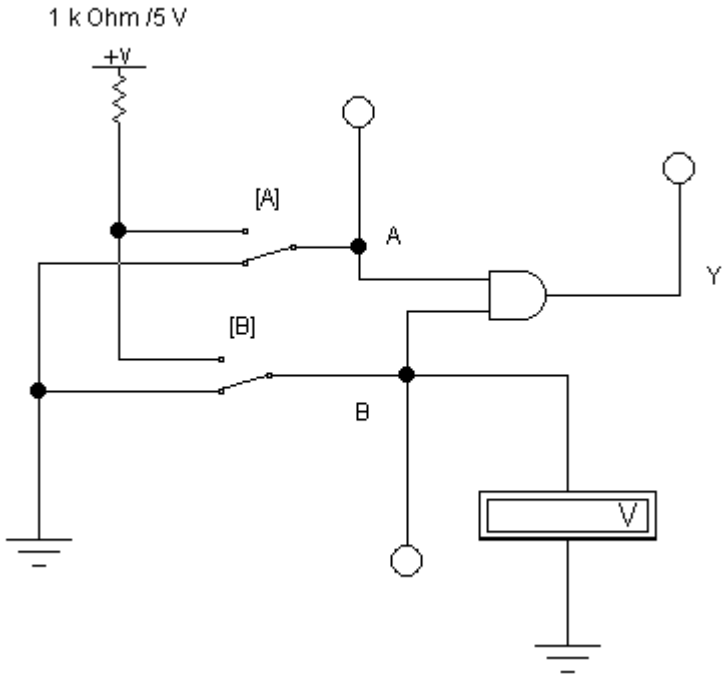


Рис. 1.а – Логическая функция «И»

В этой схеме два двухпозиционных переключателя «А» и «В» подают на входы логической схемы «И» уровни логических сигналов: «0» (контакт переключателя в нижнем положении) или «1» (контакт переключателя в верхнем положении). В физическом плане это соответствует подаче напряжения от источника (+ 5 В) на логический элемент. Уровень физического сигнала на входе или выходе логического элемента можно измерить с помощью вольтметра (в данном случае «0» В или «+5» В), а уровень логического сигнала с помощью логических пробников, которые информируют о наличии на измеряемой линии сигнала вообще (если сигнал есть, то пробник загорается красным цветом).

Включите схему (тумблером, находящимся в правом верхнем углу окна программы). Установите положения ключей в соответствии с табл. 1.а. Результаты замеров (логических и физических сигналов) занесите в табл. 1.а.

Таблица 1.а – Задание уровней логических сигналов

Положение переключателей		**Сигналы на входах и выходе**					
«А»	**«В»**	**Логические (0 или 1)**			**Физические, В**		
		A	**B**	**Y**	**A**	**B**	**Y**
Нижнее	Нижнее						
Нижнее	Верхнее						
Верхнее	Нижнее						
Верхнее	Верхнее						

Примечание: прежде чем начать работу с переключателями удобно каждому переключателю присвоить букву, при нажатии которой он включается/выключается. Это можно сделать до начала работы схемы, дважды щелкнув на переключатель и на закладке «Value» присвоив ему уникальную кнопку.

б) Экспериментальное получение таблицы истинности элемента «И»

Подайте на входы схемы (рис. 1.а) все возможные комбинации уровней сигналов «А» и «В» и для каждой комбинации зафиксируйте уровень выходного сигнала «Y». Заполните таблицу истинности исследуемой логической схемы «И» (таблица 1.б)

Таблица 1.б – Таблица истинности логического элемента «И»

Входы		**Выход**
A	**B**	**Y**

в) Получение аналитического выражения для функции

По таблице 1.б составьте аналитическое выражение функции элемента «И» и занесите его себе в отчет. Для этого можно воспользоваться одним из двух способов получения логических выражений по таблице истинности изложенных в краткой теории к данной лабораторной работе.

Задание 2. Исследование логической функции «И-НЕ»

а) Экспериментальное получение таблицы истинности логического элемента «И-НЕ», составленного из элементов «И» и «НЕ»

Соберите схему, изображенную на рис. 2.а.

Включите схему. Подайте на входы схемы все возможные комбинации уровней входных сигналов и, наблюдая уровни сигналов на входах и выходе с помощью логических пробников, заполните таблицу истинности логической схемы «И-НЕ» (табл. 2.а).

Таблица 2.а – Таблица истинности элемента «И-НЕ» (составного)

Входы		**Выход**
A	**B**	**Y**
0	0	
0	1	
1	0	
1	1	

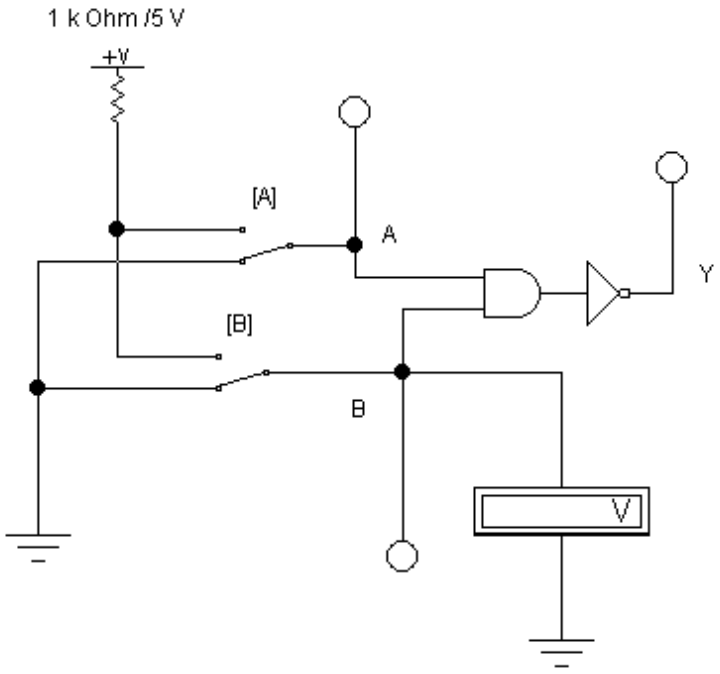


Рис. 2.а – Логическая функция «И-НЕ»

б) Экспериментальное получение таблицы истинности логического элемента «И-НЕ»

Соберите схему, изображенную на рис. 2.б.

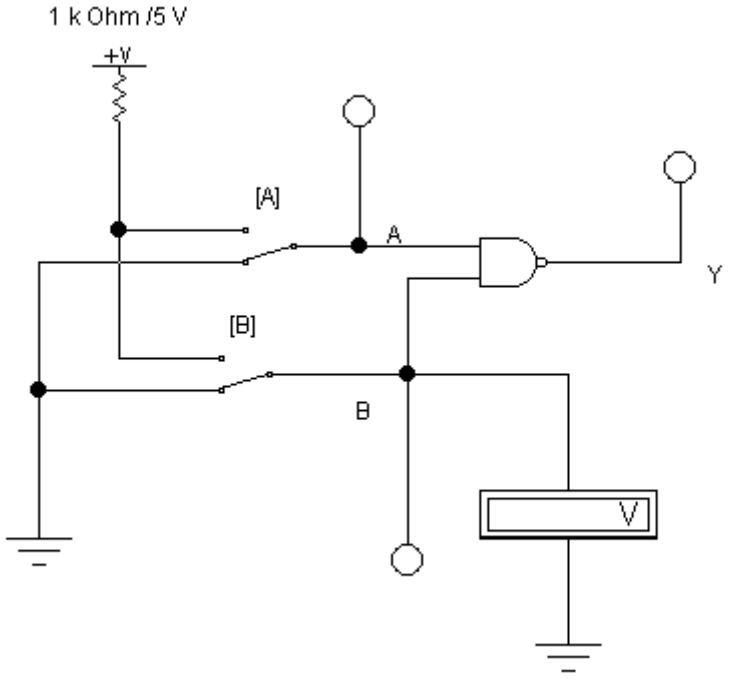


Рис. 2.б – Логическая функция «И-НЕ»

Включите схему. Подайте на входы схемы все возможные комбинации уровней входных сигналов и, наблюдая уровни сигналов на входах и выходе с помощью логических пробников, заполните таблицу истинности логической схемы «И-НЕ» – табл. 2.б.

Таблица 2.б – Таблица истинности элемента «И-НЕ»

Входы		Выход
A	**B**	**Y**
0	0	
0	1	
1	0	
1	1	

Сравните таблицы 2.а и 2.б между собой и сделайте соответствующие выводы.

Задание 3. Исследование логической функции «ИЛИ»

а) Экспериментальное получение таблицы истинности логического элемента «ИЛИ»

Соберите схему, изображенную на рис. 3.а.

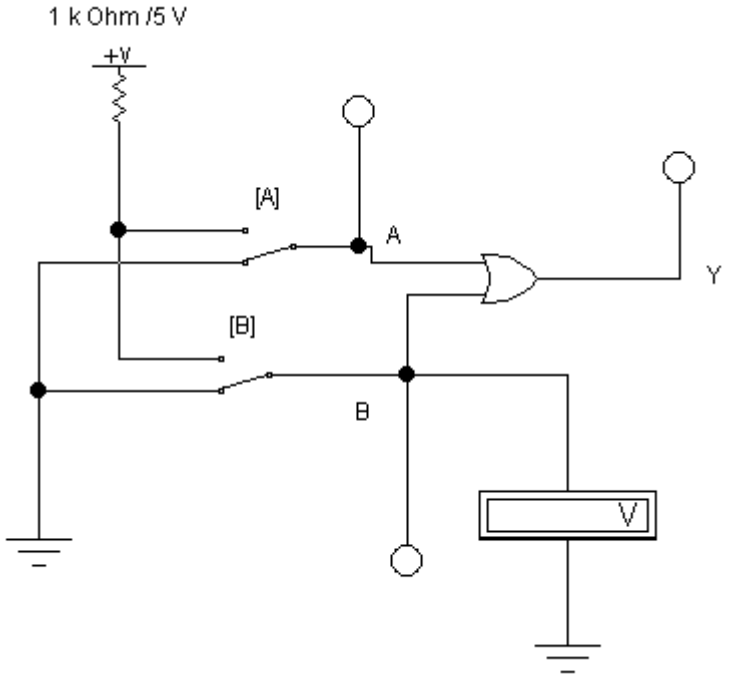


Рис. 3.а – Логическая функция «ИЛИ»

Включите схему. Подайте на входы схемы все возможные комбинации уровней входных сигналов и, наблюдая уровни сигналов на входах и выходе с помощью логических пробников, заполните таблицу истинности логической схемы ИЛИ (табл. 3.а).

Таблица 3.а – Таблица истинности элемента «ИЛИ»

Входы		Выход
A	**B**	**Y**
0	0	
0	1	
1	0	
1	1	

б) Получение аналитического выражения для функции

По таблице 3.а составьте аналитическое выражение функции «ИЛИ» и занесите его в отчет. Для этого можно воспользоваться одним из двух способов получения логических выражений по таблице истинности изложенных в краткой теории к данной лабораторной работе.

Задание 4. Исследование логической функции «ИЛИ-НЕ»

а) Экспериментальное получение таблицы истинности логического элемента «ИЛИ-НЕ», составленного из элементов «ИЛИ» и «НЕ»

Соберите схему, изображенную на рис. 4.а.

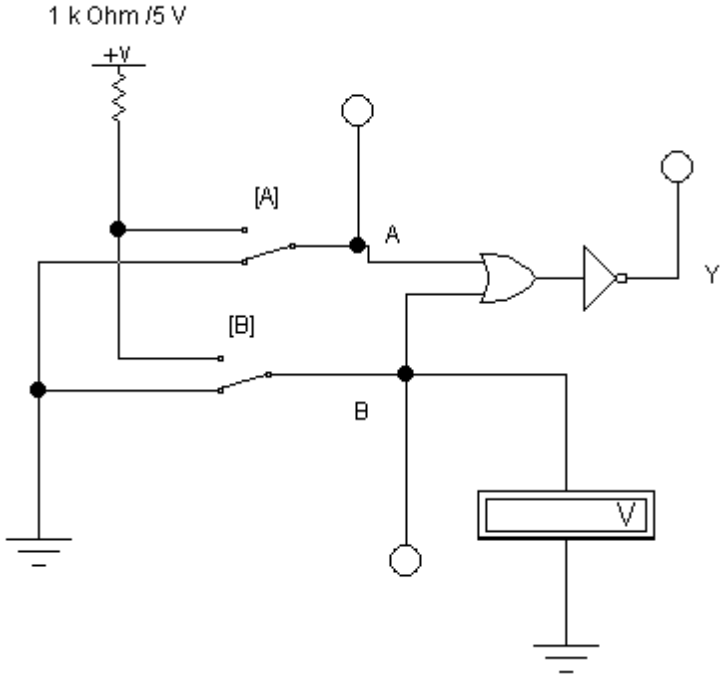


Рис. 4.а – Логическая функция «ИЛИ-НЕ»

Включите схему. Подайте на входы схемы все возможные комбинации уровней входных сигналов и, наблюдая уровни сигналов на входах и выходе с помощью логических пробников, заполните таблицу истинности логической схемы «ИЛИ-НЕ» (табл. 4.а).

Таблица 4.а – Таблица истинности элемента «ИЛИ-НЕ» (составного)

Входы		**Выход**
A	**B**	**Y**
0	0	
0	1	
1	0	
1	1	

б) Экспериментальное получение таблицы истинности логического элемента «ИЛИ-НЕ»

Соберите схему, изображенную на рис. 4.б.

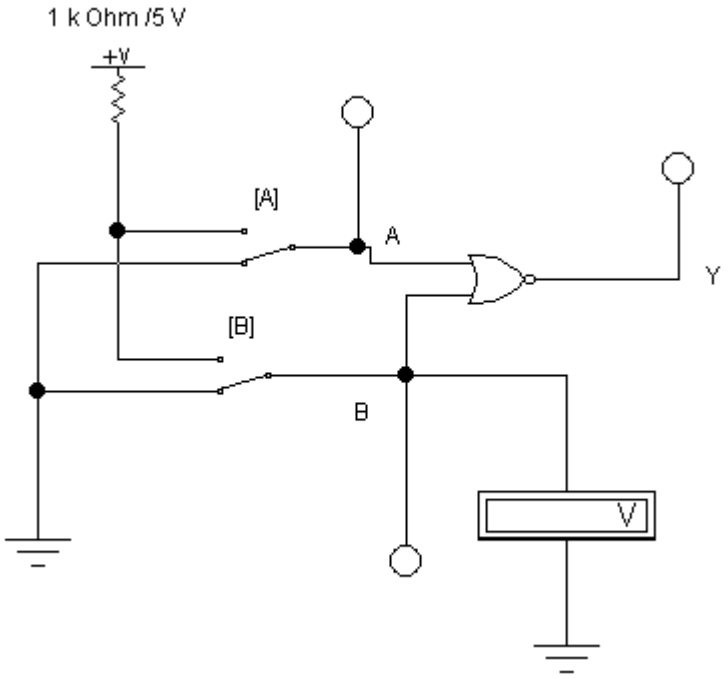


Рис. 4.б – Логическая функция «ИЛИ-НЕ»

Включите схему. Подайте на входы схемы все возможные комбинации уровней входных сигналов и, наблюдая уровни сигналов на входах и выходе с помощью логических пробников, заполните таблицу истинности логической схемы «ИЛИ-НЕ» (табл. 4.б).

Таблица 4.б – Таблица истинности элемента «ИЛИ-НЕ»

Входы		**Выход**
A	**B**	**Y**
0	0	
0	1	
1	0	
1	1	

Сравните таблицы 4.а и 4.б между собой и сделайте соответствующие выводы.

Задание 5. Исследование логических схем с помощью генератора слов

Анализ различных логических функций иногда удобно проводить с помощью микросхем реализующих различные логические элементы. Кроме того, использование микросхем (в отличие от расширенного представления логических функций, рассматриваемого в предыдущих заданиях) бывает необходимо на стадии проектирования сложных систем, когда в руках современного инженера зачастую имеется набор из различных, законченных модулей, реализующих вполне определенные функции.

Отдавая дань доступности и широкому распространению англоязычных микросхем, а так же их аналогов, изготовленных по западным стандартам (в том числе и современные российские разработки) в данной работе рассматриваются именно микросхемы западного стандарта.

Помимо этого для дальнейшего рассмотрения цифровой техники удобно вместо коммутирующих выключателей (задающих уровни логических сигналов) в некоторых случаях использовать «генератор слов» (прибор, предназначенный для выработки последовательности логических сигналов на своих выходах, причем данную последовательность пользователь имеет возможность задавать самостоятельно). Именно использованию законченных микросхем и «генератора слов» посвящено данное задание.

а) Исследование микросхем различных типов

Выбор исследуемой микросхемы производится по номеру варианта, присвоенному студенту преподавателем. Варианты микросхем в соответствии с вариантом задания приведены в таблице 5.а.

Примечание: Данные микросхемы в общем случае могут содержать несколько независимых друг от друга логических элементов (но одного типа, на пример 3 логических элемента «И» – «AND»), причем у каждого логического элемента может быть и несколько входов (1, 2, 3 или 4).

Таблица 5.а – Варианты задания исследуемой микросхемы

Вариант задания	**Тип микросхемы**
1	AND/7408;
2	AND/7421
3	AND/7411
4	NOR/7402
5	NOR/7427
6	NOR/7428
7	NOR/7433
8	OR/7432
9	XOR/7486;
10	XNOR/74266

После выбора соответствующей микросхемы необходимо собрать схему для ее изучения. Данная схема в общем случае должна содержать: источник напряжения, заземление (земля), необходимое количество логических пробников и генератор слов. Пример законченной схемы для исследования микросхемы NOR/7400 приведен на рис. 5.а.

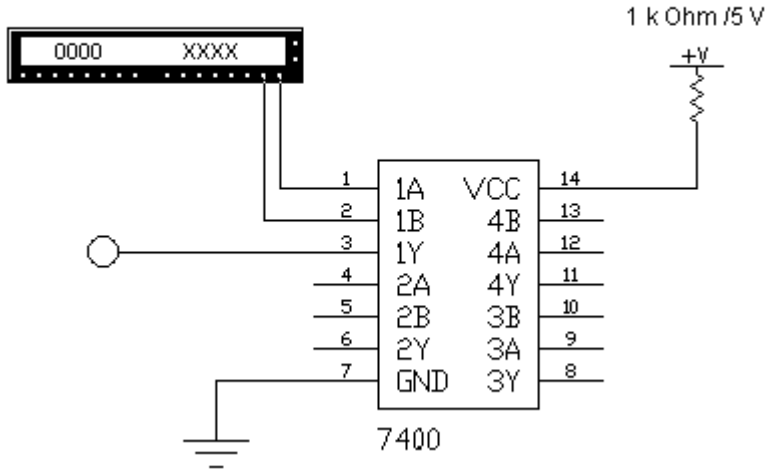


Рис. 5.а – Исследование микросхемы

Примечание: для дальнейшего исследования необходимо принимать во внимание только входы/выходы одного логического элемента на данной микросхеме (на рисунке 5.а используется первый логический элемент).

В отчете по данному пункту необходимо заполнить таблицу 5.а, дающую исчерпывающую информацию об используемой микросхеме.

Таблица 5.а – Информация о микросхеме

Тип микросхемы (полное обозначение по варианту задания)	
Тип базисных элементов (логических функций)	
Число базисных элементов в микросхеме (всего)	
Число исследуемых базисных элементов в микросхеме	
Обозначение выводов микросхемы используемых для подключения источника питания (номера и название)	
Обозначения выводов микросхемы используемых для подключения заземления (номера и название)	
Обозначение используемых входов (номера и название)	
Обозначение используемых выводов (номера и название)	

Примечание: исчерпывающую информацию об используемой микросхеме можно получить в справке (выделите интересующий вас элемент и щелкните на знак вопроса).

б) Экспериментальное получение таблицы истинности микросхемы

Для дальнейшего изучения микросхемы необходимо использование «генератора» слов (предполагается, что в пункте 5.а была собрана схема для изучения). Общий вид открытого окна свойств «генератора» показано на рис. 5.б.

Для открытия окна свойств необходимо дважды щелкнуть мышью на изображении генератора слов. Окно свойств может быть условно разбито на ряд частей.

В левой части, в прокручивающимся окне, отображаются слова (набор логических последовательностей) формируемые «генератором слов» в 16-

ричной системе исчисления (данные последовательности могут быть изменены пользователем).

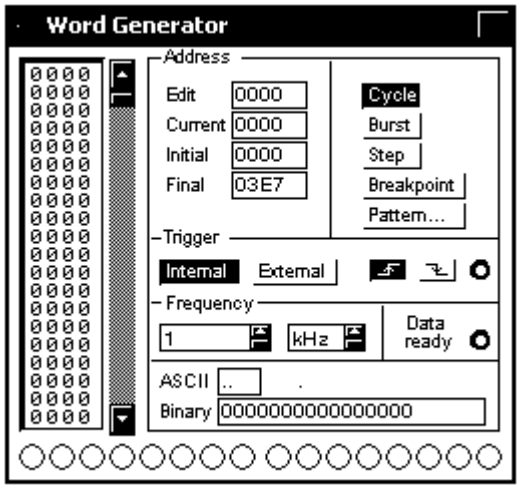


Рис. 5.б – Генератор слов

В правой части окна свойств находится ряд областей («Address», «Trigger», «Data ready»,), несущих техническую информацию. Из этих областей мы будем пользоваться только кнопками «Cycle» и «Step», переводящий генератор слов в цикловой (заданные пользователем слова прокручиваются автоматически) и пошаговый режим работы (каждая следующая последовательность, вырабатываемого «генератором слов», вызывается нажатием кнопки «Step»).

В правой нижней части окна свойств находятся поля «ASCII», «Binary и ряд, состоящий из логических пробников. Поле «Binary» (изменяемое) и ряд пробников (функционирующих только в режиме «Работа») несут одинаковую информационную сущность. Они отображают генерируемые последовательности слов в двоичном коде.

Отметим так же, что часто бывает удобнее задавать последовательность слов именно в поле «Binary», которое связанно (изменение его вызывает изменение другого поля) и с прокручивающимся списком генерируемых слов, находящимся в левой части и рассмотренным ранее.

Для выполнения работы запрограммируйте «генератор слов» так, чтобы на выходе генератора получать последовательно следующие комбинаций*: 00, 01, 10, 11* (необходимо помнить что, несмотря на то, что в анализируемых микросхемах количество базисных логических элементов и входов на каждом элементе множество, мы исследуем только два входа одного базисного логического элемента). Затем переведите генератор в режим пошаговой работы нажатием кнопки "Step" в окне свойств генератора. Каждое нажатие кнопки "Step" вызывает переход к очередному слову заданной последовательности, которое подается на выход генератора. Последовательно подавая на микросхему слова из заданной последовательности, заполните таблицу истинности элемента вашего базисного элемента (табл. 5.б).

Указание: После программирования генератора слов для функционирования схемы необходимо перевести ее в режим работы (тумблером, находящимся в

левой части главного окна программы). Для повторного прогона можно выключить и включить схему заново, тогда последовательности, заданные в «генераторе слов» будут изменяться, начиная с первой.

Таблица 5.6 – Таблица истинности исследуемой микросхемы

Входы		**Выход**
1	**2**	**Y**
0	0	
0	1	
1	0	
1	1	

Задание 6. Реализация логической функции 3-х и большего числа переменных

В заключение изучения базовых логических схем и функций приведем задание, иллюстрирующее возможности «Electronic Workbench» в такой непростой и трудно формализуемой задаче как синтез логических схем и функций. Все предыдущие задания опирались в основном на моделирование уже существующих логических функций, изучение их свойств. Выполнив же это задание, студенты сами смогут автоматически реализовывать любую логическую функцию, заданную булевым выражением в различных базисах, получать таблицы истинности логических выражений и т. п. Все выше перечисленные процедуры можно реализовать с помощью инструмента «Logic Converter». Общий вид окна свойств «Logic Converter» показан на рис. 6. Напомним, что окно свойств вызывается двойным нажатием на изображении инструмента. Рассмотрим окно свойств подробнее.

Здесь в верхней части под буквами A, B,…,H находятся индикаторы, активизируемые путем нажатия на них.

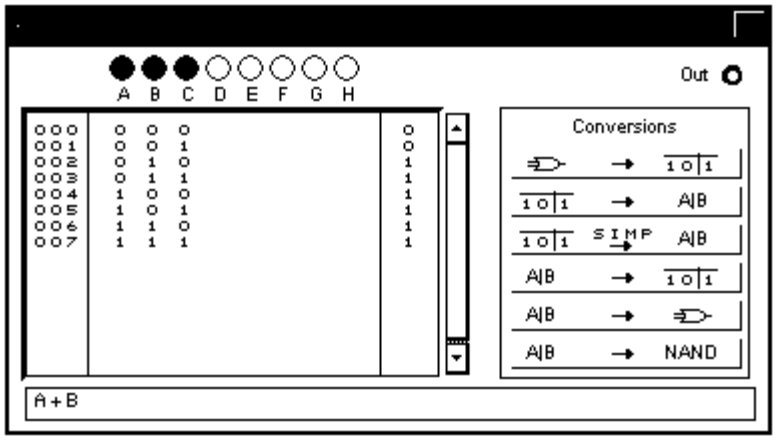


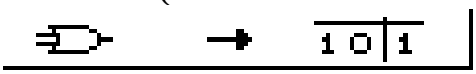
Рис. 6 – «Logic Converter»

На рис. 6 активизированы три индикатора A, B, C. Это подразумевает то, что вы в дальнейшем будите оперировать логической функцией, состоящей из трех переменных: *A*, *B*, *C*. Данный выбор вызывает появление всех возможных комбинаций этих переменных (строки таблицы, изображенной на рис. 6, начиная с нулевой и заканчивая 7). В правой части полученной таблицы истинно-

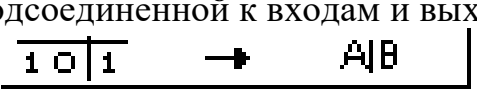
сти, в отдельном столбике по умолчанию присвоены «0» выходной переменной. Пользователь сам, изменяя значения выхода анализируемой логической функции, тем самым полностью задает таблицу истинности.

В самом нижнем окне находится поле ввода/вывода логической функции (поле является выводящим, если пользователь сам задал таблицу истинности, а затем путем нажатия на соответствующую функциональную кнопку потребовал формирование логической функции; и поле является вводящим, если пользователь предварительно сам задал логическую функцию, а затем путем нажатия на соответствующую функциональную кнопку потребовал формирование таблицы истинности).

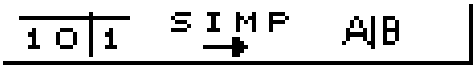
В правой части окна свойств находятся соответствующие функциональные кнопки (панель «Conversion»). Их назначение рассмотрено ниже.

1) 

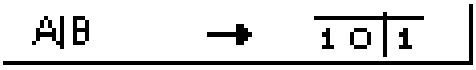
- Получение таблицы истинности для произвольной логической схемы, подсоединенной к входам и выходам «Logic Converter»;

2) 

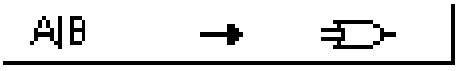
- Получение по таблице истинности логической функции (не упрощенной);

3) 

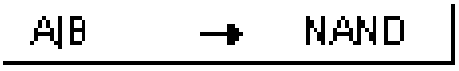
- Получение по таблице истинности упрощенной логической функции (аналогичная функция получается путем использования для анализа таблицы истинности карт Карно);

4) 

- Получение по логической функции таблицы истинности;

5) 

- Получение по логической функции схемы

6) 

- Получение по таблице истинности логической функции реализованной в базисе «И-НЕ».

а) Синтез схемы, реализующей заданную функцию при помощи логического преобразователя

Для получения схемы, реализующей функцию, описываемую логическим выражением *f* (задается преподавателем), можно воспользоваться логическим преобразователем. Варианты заданий логических выражений приведены в таблице 6.а (здесь апостроф указывает на операцию инвертирования).

Таблица 6.а – Варианты задания логического выражения *f*

Вариант задания	**Логическое выражение *f***
1	A+B*C
2	C+A*B
3	A*B*C
4	A’+B+C
5	A+B’C
6	C*B’+A
7	A’+B’+C’
8	C’+A’*B
9	A’+B’+C
10	B+C*A’

Проделайте с анализируемым логическим выражением следующие действия:

• вызовите логический преобразователь;

• введите в нижнее окно панели преобразователя логическое выражение с клавиатуры (операции ИЛИ соответствует знак «+», инверсия обозначается апострофом, логической операции умножения не вводится, например AB=A*B);

• для реализации схемы на элементах «И-НЕ» нажмите соответствующую функциональную кнопку на панели логического преобразователя.

Занесите полученную схему в отчет.

Удалите полученную схему.

• для простой реализации схемы на произвольных элементах нажмите соответствующую функциональную кнопку на панели логического преобразователя.

Занесите полученную схему в отчет.

Удалите полученную схему.

• для получения таблицы истинности заданной логической функции нажмите соответствующую функциональную кнопку на панели логического преобразователя.

Занесите полученную таблицу истинности.

б) Синтез логической функции, реализующей заданную таблицу истинности при помощи логического преобразователя

Для получения функции, реализующей таблицу истинности, необходимо воспользоваться таблицей истинности полученной, в предыдущем пункте. Для этого проделайте следующее:

• удалите с нижней строки логического преобразователя логическую функцию;

• очистите таблицу истинности преобразователя, нажав на его верхнюю функциональную кнопку;

• выберите в поле таблицы истинности количество используемых логических переменных (в нашем случае их 3), путем нажатия на соответствующие индикаторы;

• в правой колонке полученной таблицы истинности задайте значение искомой логической функции в соответствии с каждой строкой;
• для получения не упрошенной логической функции нажмите соответствующую функциональную кнопку на панели логического преобразователя, занесите в отчет полученную функцию;
• для получения упрошенной логической функции нажмите клавишу соответствующую функциональную кнопку на панели логического преобразователя, запишите упрощенную логическую функцию.

Удостоверьтесь, что упрощенные модели совпадают с заданными в начале моделями;

Удостоверьтесь, что упрощенная функция может быть получена из не упрощенной (проделайте операции упрощения).

Контрольные вопросы

1. Дайте определение логическому сигналу.
2. Дайте определение логической переменной.
3. Дайте определение логической функции.
4. Какие значения могут принимать булевы переменные?
5. Приведите основные логические тождества.
6. Что может быть принято за уровни логических сигналов?
7. Подумайте, почему часто в технике за уровень логического нуля принимают физический сигнал (например, по току) отличный от нуля.
8. Как может быть получена логическая функция?
9. Чем в физическом смысле отличается работа схемы составленной по упрощенной логической функции от неупрощенной?
10. Сколько различных комбинаций сигналов надо подать на схему, имеющую 4 входа, для составления таблицы истинности?
11. Какой сигнал должен быть подан на неиспользуемые входы элемента «И-НЕ» имеющего 5 входов, если требуется реализовать ту же логическую функцию, но на 3 входа?
12. Какой сигнал должен быть подан на неиспользуемые входы элемента «ИЛИ» имеющего 5 входов, если требуется реализовать ту же логическую функцию, но на 4 входа?
13. Какой сигнал нужно подать на неиспользуемые входы элемента «И» имеющего 2 входа для реализации на его базе инвертора на один входной сигнал.
14. Какой логической функцией можно описать систему пуска трехфазного двигателя, если двигатель может быть запущен, когда три датчика подтверждают наличие фазных напряжений?

Упражнения

Упражнение 1. Получение логических функций

По заданной таблице истинности (таблица 1.у) получите логическую функцию.

Таблица 1.у – Варианты таблицы истинности

Значения логических переменных (для всех вариантов)				**Варианты задания**									
				1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
A	**B**	**C**	**D**	**Значения логической функции (для каждого варианта)**									
0	0	0	0	1	0	1	1	0	1	0	0	1	0
0	0	1	0	0	0	1	1	1	1	0	0	1	1
0	1	0	0	0	0	1	1	0	1	0	0	1	0
0	1	1	0	0	1	0	0	0	0	1	0	1	0
1	0	0	0	1	1	1	1	1	0	0	0	1	0
1	0	1	0	0	1	1	0	1	1	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	1	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0	1	0	1	0	1	1
0	0	1	1	1	0	1	1	1	0	0	0	1	1
0	1	0	1	1	1	0	1	0	1	0	0	0	1
0	1	1	1	0	0	1	1	1	1	1	1	1	1
1	0	0	1	1	0	1	1	0	0	0	1	0	1
1	0	1	1	0	0	1	1	0	0	0	1	0	1
1	1	0	1	1	1	0	1	0	1	1	1	1	1
1	1	1	1	1	1	0	0	1	1	0	1	1	1

Примечание: Значения логических переменных, входящих в вашу логическую функцию заданы столбцами таблицы (*A, B, C, D*), они одинаковы для всех вариантов. Из второй части таблицы необходимо выбрать один столбец (из 10), согласно вашему варианту, который содержит значения вашей логической функции.

Упражнение 2. Построение структурных схем

По заданной логической функции необходимо построить структурную схему.

Варианты логических функций, приведены ниже, в таблице 2.у.

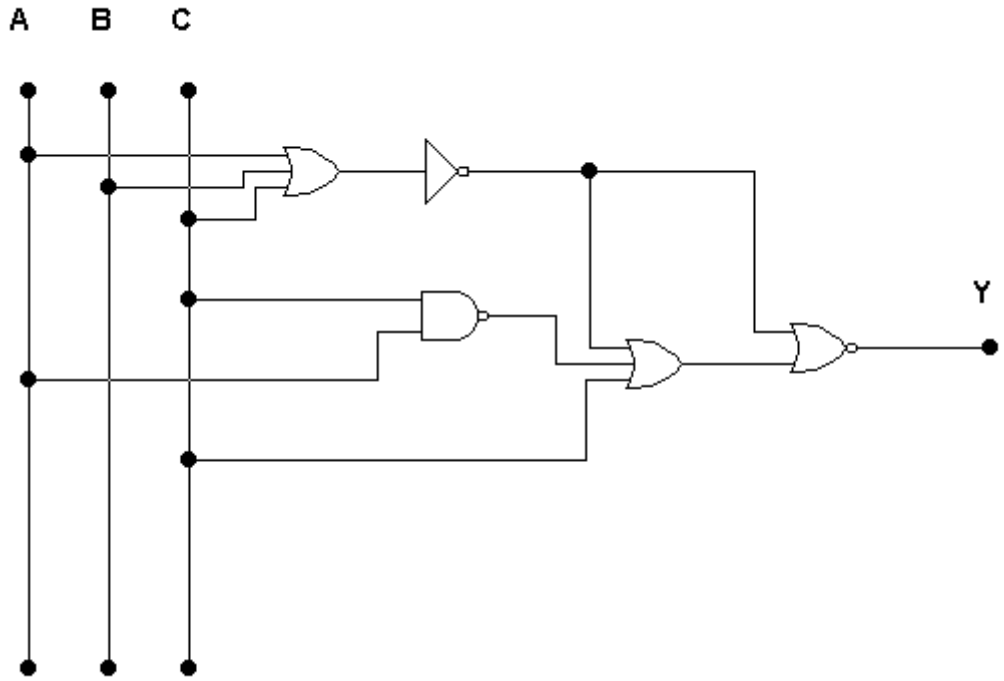
Таблица 2.у – Варианты логических функций

Вариант	**Логическая функция**
1	$f = \overline{a \vee b} \wedge c \wedge d \vee e$
2	$f = \overline{a \wedge c \vee d \vee e \vee b}$
3	$f = \overline{\overline{c} \wedge a} \vee b \wedge \overline{d \vee e}$
4	$f = \overline{\overline{a} \vee \overline{b}} \wedge \overline{\overline{e} \wedge \overline{d}} \vee c$
5	$f = a \vee b \wedge \overline{c \vee d \vee e}$
6	$f = a \wedge \overline{\overline{c} \wedge d} \vee e \vee \overline{b}$
7	$f = \overline{a \vee b} \vee \overline{\overline{c} \vee \overline{d} \wedge \overline{e}}$
8	$f = a \wedge b \wedge \overline{d \vee a} \wedge \overline{e} \vee c$
9	$f = \overline{a \wedge \overline{a}} \wedge c \wedge \overline{d} \vee e$
10	$f = \overline{e \vee \overline{a}} \wedge \overline{\overline{a \vee d} \vee e} \wedge b$

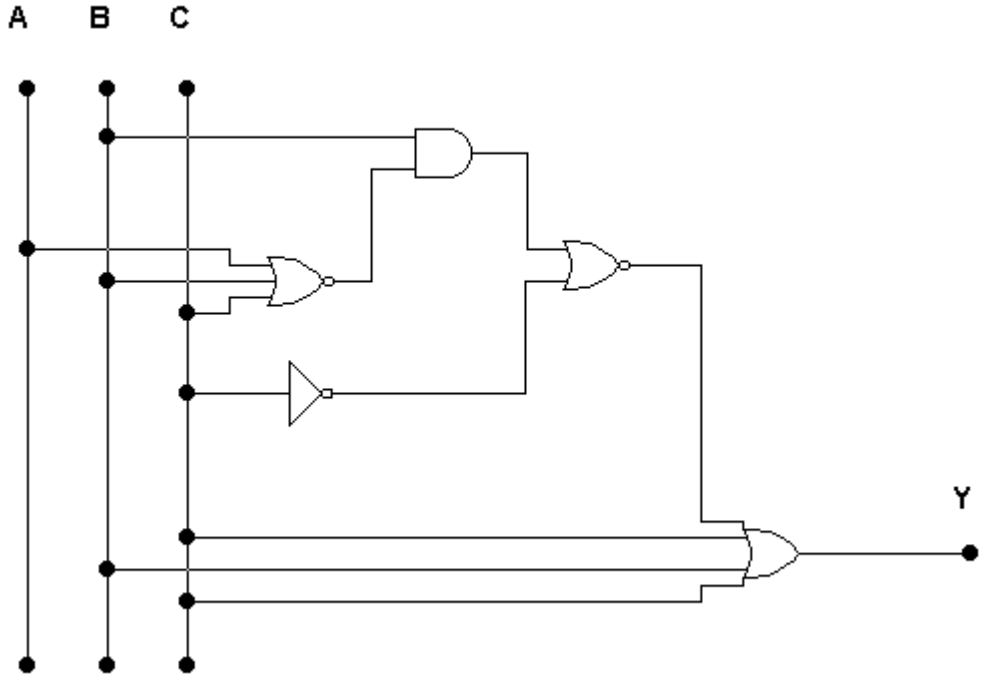
Упражнение 3. Составление логических функций

По заданной схеме напишите логическую функцию.

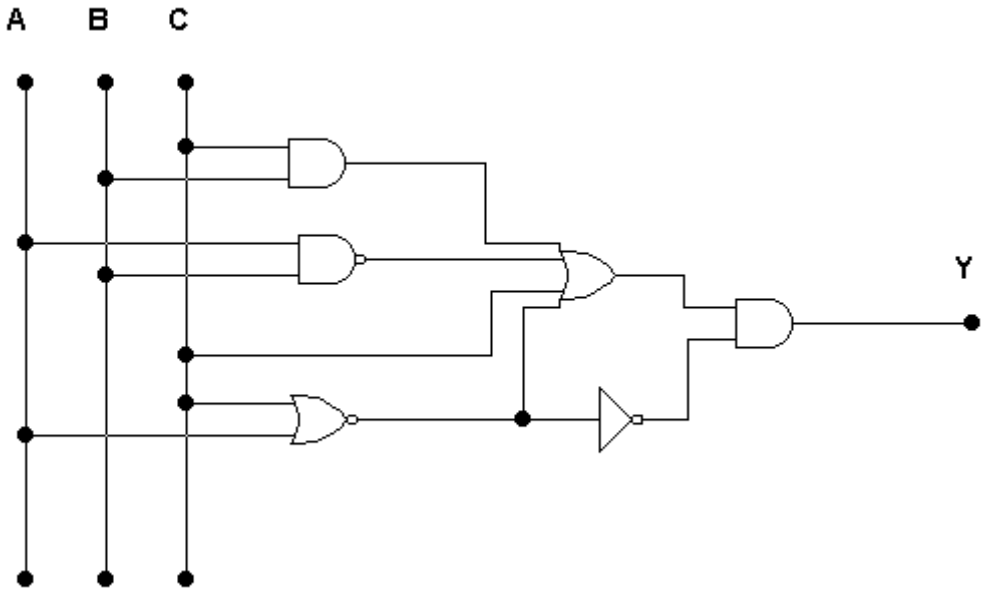
Вариант 1



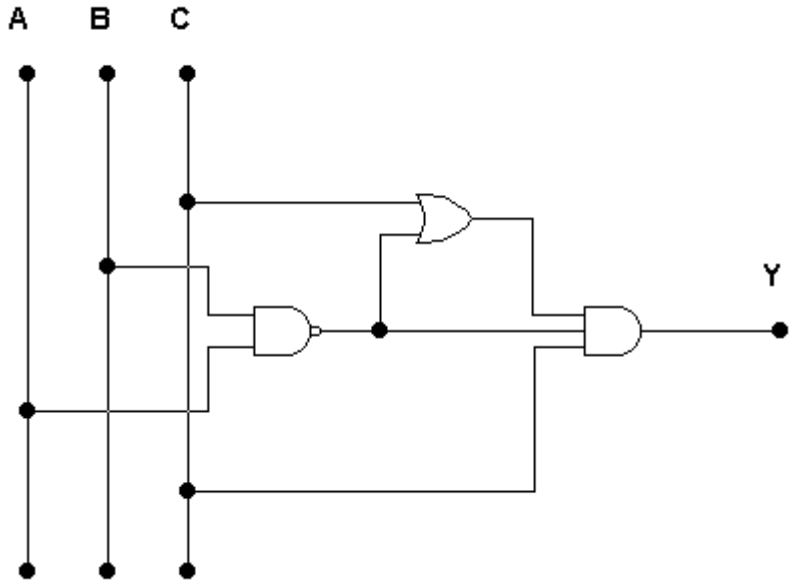
Вариант 2



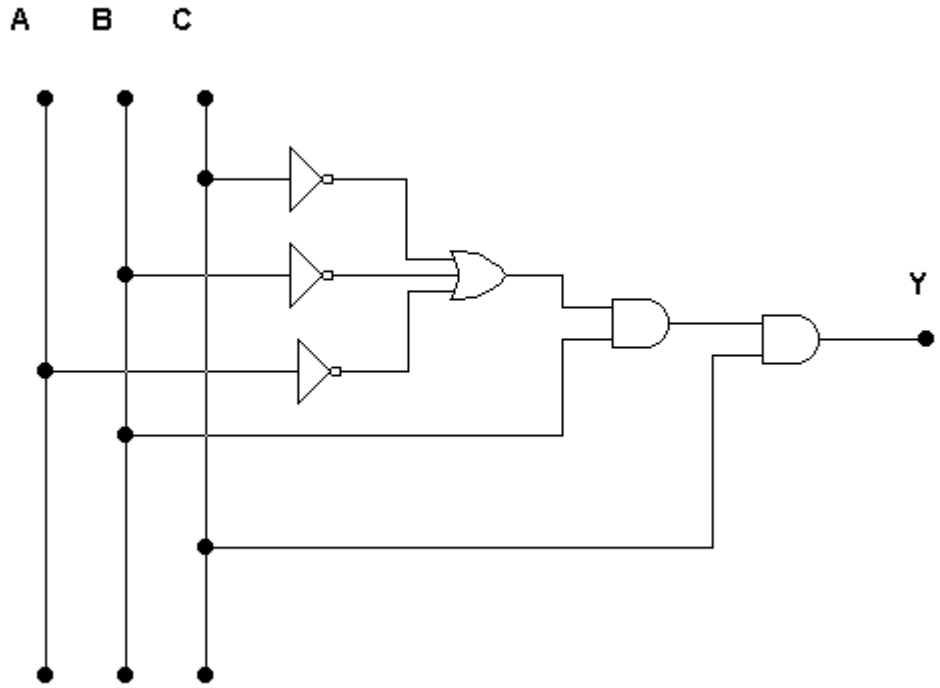
Вариант 3



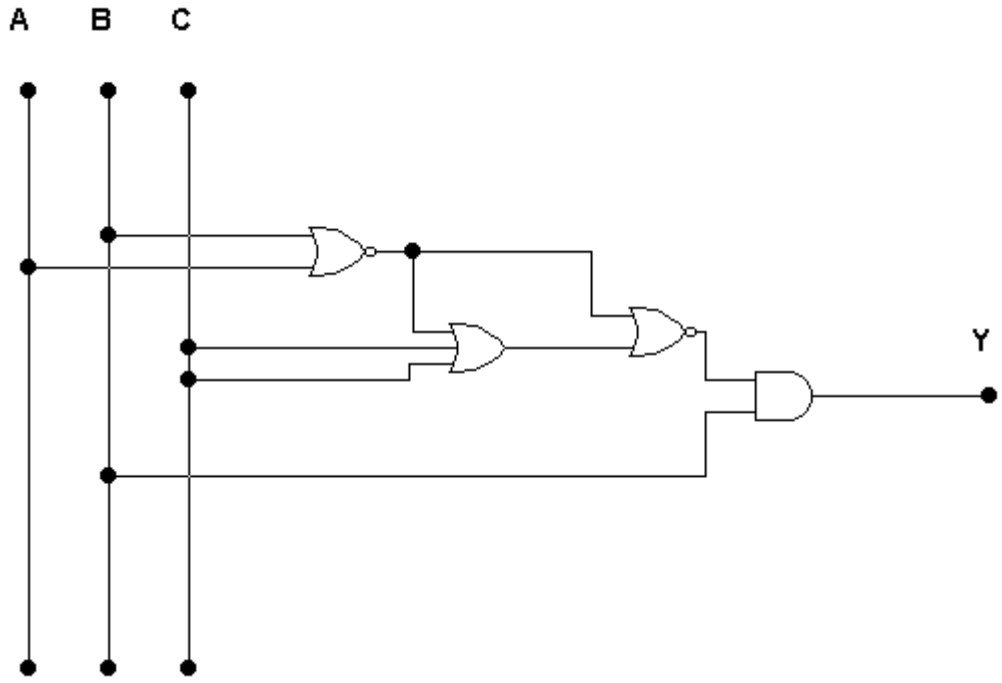
Вариант 4



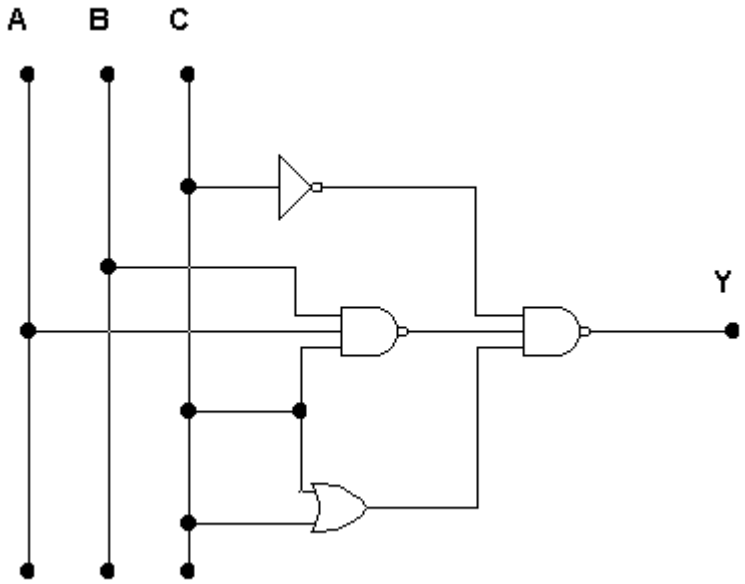
Вариант 5



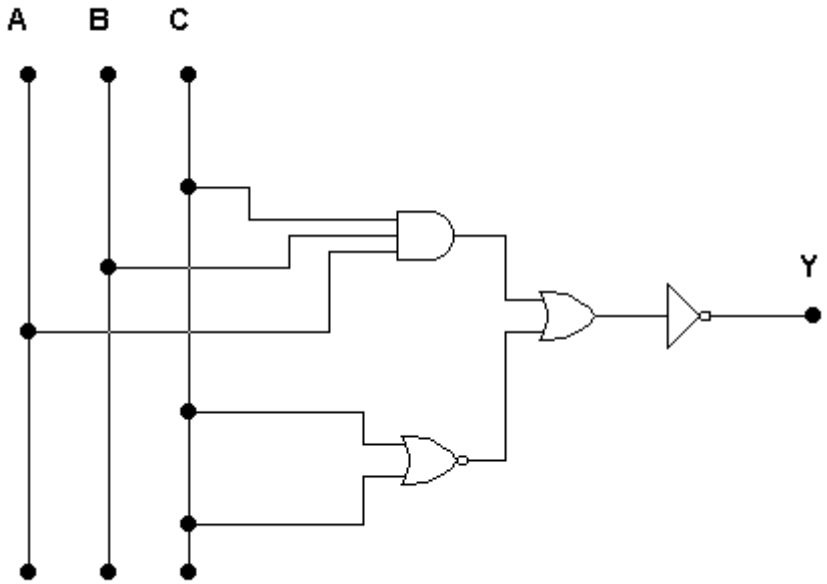
Вариант 6



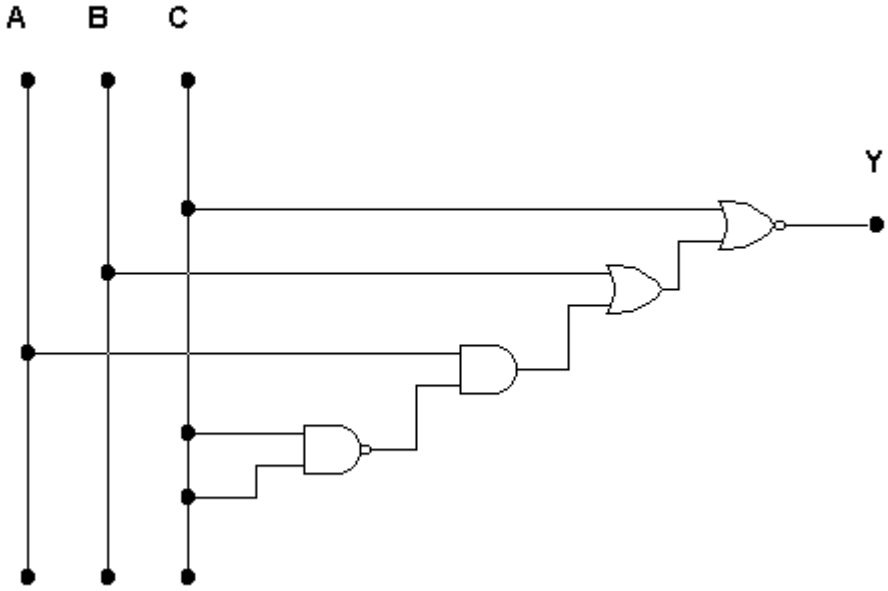
Вариант 7



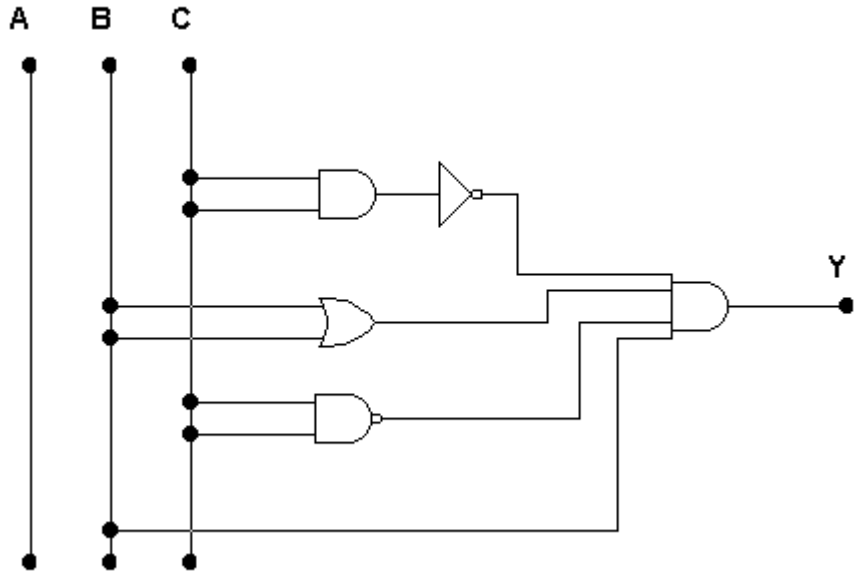
Вариант 8



Вариант 9



Вариант 10



ЛАБОРАТОРНАЯ РАБОТА № 2

Изучение работы шифраторов, дешифраторов и мультиплексоров

Цель работы:

1. Изучение принципов работы шифраторов, дешифраторов и мультиплексоров.
2. Реализация логических функций с помощью мультиплексоров.
3. Изучение способов применения дешифраторов.

Приборы и элементы:

Генератор слов (панель «Instruments/Word Generator»)
Логический анализатор (панель «Instruments/Logic Analyzer»)
Логические пробники (панель «Indicators/Red probe»)
Источник напряжения + 5 В (панель «Basic/Pull-Up Resistor»)
Земля (панель «Sources/Ground»)
Двухвходовые элементы И, И-НЕ, ИЛИ, ИЛИ-НЕ (панель «Logic Gates/2-Input AND, NAND, OR, NOR Gates»)
Двухпозиционные переключатели (панель «Basic/Switch»)
Дешифратор (панель «Digital/DEC/Generic 8-to-1 DEMUX»)
Мультиплексор (панель «Digital/MUX/ Generic 1-of 8 MUX»)

Краткие теоретические сведения

1. Комбинационные схемы

Комбинационной схемой называется логическая схема, реализующая однозначное соответствие между значениями входных и выходных сигналов. Для реализации комбинационных схем используются логические элементы, выпускаемые в виде интегральных схем. В этот класс входят интегральные схемы дешифраторов, шифраторов, мультиплексоров, демультиплексоров, сумматоров.

2. Шифраторы

Шифратор – логическая комбинационная схема, которая имеет 2^n входов (где n – число информационных выходов). Часто $n=3$, тогда $2^n=8$. Подаче на один из входов активного сигнала будет соответствовать двоичное число, которое можно сформировать из его n выходов, эквивалентное номеру входа, на котором появился активный уровень.

Примечание: формирование двоичного числа на выходе шифратора означает следующее: каждый из выходов шифратора считается определенным разрядом искомого двоичного числа. Принцип работы шифратора противоположен принципу работы дешифратора, который подробно рассмотрен ниже.

Простейшим случаем применения шифратора может служить, например, схема отслеживание нажатия одной кнопки. Каждая кнопка представляет собой элементарный переключатель. Пусть таких кнопок всего 8, и они пронумерованы начиная с «0» и заканчивая «7». При нажатии определенной кнопки (например «3») формируется сигнал на входе шифратора (каждый вход – это вполне определенная кнопка), в итоге на выходе шифратора можно получить

двоичный сигнал равный номеру нажатой кнопки. В нашем случае число выходов равно 3 (2^3=8), и на каждом из выходов получится следующая комбинация на 1-м = 1, на 2-м = 1, на 3-м = 0. Полученное двоичное число «011», которое в десятичном коде равно 3.

3. Дешифраторы

Дешифратор – логическая комбинационная схема, которая имеет n информационных входов и 2^n выходов. Каждой комбинации логических уровней на входах будет соответствовать активный уровень на одном из 2^n выходов. Обычно n равно 2, 3 или 4. Работа дешифратора может быть проиллюстрирована в соответствии с таблицей истинности 1. В отечественной литературе входы дешифратора принято обозначать (1, 2, 4, 8, …), в англоязычной (*A*, *B*, *C*, …). На рис. 1 представлен дешифратор, имеющий таблицу истинности 1. Условное обозначения базовых элементов соответствует обозначению, принятому в отечественной литературе.

В дальнейшем для детального рассмотрения дешифратора, мы будем пользоваться обозначением (микросхем и базовых элементов) принятых в EWB.

Таблица 1 – Таблица истинности дешифратора

Входы				**Выходы**									
8	**4**	**2**	**1**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1

На рис. 2 изображен дешифратор (блок микросхемы, использующийся для моделирования дешифратора в EWB) с $n = 3$.

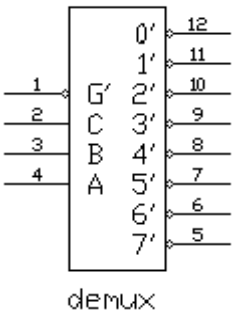


Рис. 2

Активным уровнем сигнала данной микросхемы является уровень логического нуля. То есть в отличие от таблицы истинности 1, у данного дешифратора на выходах, по диагонали расположены нули, а остальные элементы равны единице. На входы *C, B, A* можно подать следующие комбинации логических уровней: «000», «001», «010», …, «111», всего 8 комбинаций. Схема имеет 8 выходов, на одном из которых формируется низкий потенциал, на остальных - высокий. Номер этого единственного выхода, на котором формируется активный (нулевой) уровень,

соответствует числу N, определяемому состоянием входов C, B, A следующим образом:

$$N = C \cdot 2^2 + B \cdot 2^1 + A \cdot 2^0 \qquad (1)$$

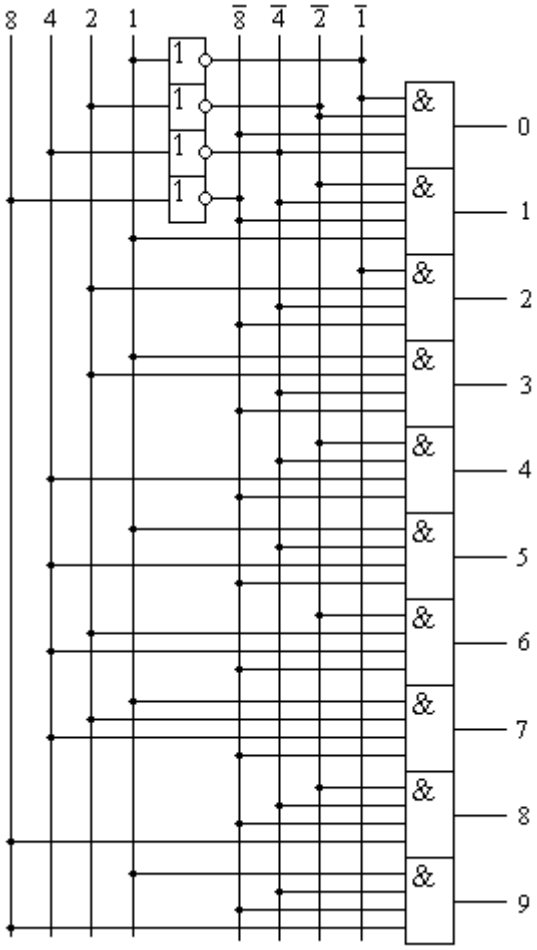


Рис. 1 – Дешифратор

Например, если на входы подана комбинация логических уровней «011», то из восьми выходов микросхемы (рис. 2) на выходе с номером $N=3$ (в двоичном исчислении 3 = 011) установится нулевой уровень сигнала ($Y3=0$), а все остальные выходы будут иметь уровень логической единицы. Этот принцип формирования выходного сигнала можно описать следующим образом:

$$Y_i = \begin{cases} 0, \text{ если } \mathrm{i} = \mathrm{k}, \\ 1, \text{ если } \mathrm{i} \neq \mathrm{k}, \\ \mathrm{k} = 2^2 \cdot C + 2^1 \cdot B + 2^0 \cdot A. \end{cases} \qquad (2)$$

Помимо информационных входов A,B,C дешифраторы обычно имеют дополнительные входы управления G. Сигналы на этих входах, например, разрешают функционирование дешифратора или переводят его в пассивное состояние, при котором, независимо от сигналов на информационных входах, на всех выходах установится уровень логической единицы. Можно сказать, что существует некоторая функция разрешения, значение которой определяется состояниями управляющих входов.

Разрешающий вход дешифратора может быть прямым или инверсным. У дешифраторов с прямым разрешающим входом активным уровнем является уровень логической единицы, у дешифраторов с инверсным входом – уровень логического нуля. На рис. 1 представлен дешифратор с одним инверсным входом управления. Принцип формирования выходного сигнала в этом дешифраторе с учетом сигнала управления описывается следующим образом:

$$Y_i = \begin{cases} 1 \cdot \overline{\overline{G}}, \text{ если } i = k, \\ 1, \text{ если } i \neq k, \\ k = 2^2 \cdot C + 2^1 \cdot B + 2^0 \cdot A. \end{cases} \quad (3)$$

У дешифратора с несколькими входами управления функция разрешения, как правило, представляет собой логическое произведение всех разрешающих сигналов управления. Например, для дешифратора серии 74138 с одним прямым входом управления *G1* и двумя инверсными *G2A* и *G2B* (Рис. 3) функции выхода Y_i и разрешения *G* имеют вид:

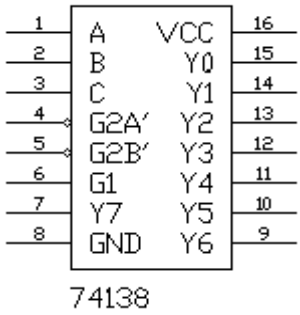


Рис. 3

$$Y_i = \begin{cases} 1 \cdot \overline{\overline{G}}, \text{ если } i = k, \\ 1, \text{ если } i \neq k, \\ k = 2^2 \cdot C + 2^1 \cdot B + 2^0 \cdot A. \end{cases} \quad (4)$$

$$G = G1 \cdot \overline{G2A} \cdot \overline{G2B} \quad (5)$$

Обычно входы управления используются для каскадирования (увеличения разрядности) дешифраторов или при параллельной работе нескольких схем на общие выходные линии.

Дешифратор может быть использован как *демультиплексор* – логический коммутатор, подключающий входной сигнал к одному из выходов. В этом случае функцию информационного входа выполняет один из входов разрешения, а состояние входов *C*, *B* и *A* задает номер выхода, на который передается сигнал с входа разрешения.

4. Мультиплексоры

Мультиплексор – комбинационная логическая схема, представляющая собой управляемый переключатель, который подключает к выходу один из входов данных. Номер подключаемого входа равен числу (адресу), определяемому комбинацией логических уровней на входах управления.

Демультиплексорами – называются устройства, которые позволяют подключать один вход к нескольким выходам.

В простейшем случае переключения (коммутацию) можно осуществить при помощи ключей, как это показано на рис. 4.

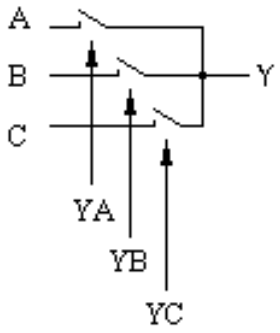


Рис. 4 – Мультиплексор на ключах.

В цифровых схемах управление ключами осуществляется при помощи логических сигналов. Сами ключи при этом, заменяются соответствующими логическим элементами.

Рассмотрим пример простейшей схемы мультиплексора. Для этого воспользуемся базовым логическим элементом «И» с таблицей истинности 2.

Таблица 2 – Таблица истинности элемента «И»

Входы		**Выход**
X	**Y**	**Out**
0	0	0
0	1	0
1	0	0
1	1	1

Теперь один из входов элемента будем рассматривать как информационный вход электронного ключа, а другой вход – как управляющий. По таблице истинности отчетливо видно, что пока на управляющий вход *Y* подан логический уровень «0» сигнал с входа *X* на выход *Out* не проходит. При подаче на управляющий вход *Y* логической «1», сигнал, поступающий на вход *X*, поступает на выход *Out*. То есть логический элемент «И» можно использовать в качестве электронного ключа. При этом не важно, какой из входов элемента будет использоваться в качестве управляющего входа, а какой – в качестве информационного. Остаётся только объединить выходы элементов «И» на один выход. Это делается при помощи элемента «ИЛИ». Условное обозначение такой схемы приведено на рис. 5.

Чаше всего для управления требуется много входов, поэтому в схему мультиплексора включают дешифратор. Это позволяет управлять переключением входов микросхемы на выход при помощи двоичных кодов. Пример такой схемы приведен на рис. 6.

Для дальнейшего рассмотрения мультиплексоров мы будем, как и прежде пользоваться условными обозначениями логических элементов и микросхем, принятых в EWB.

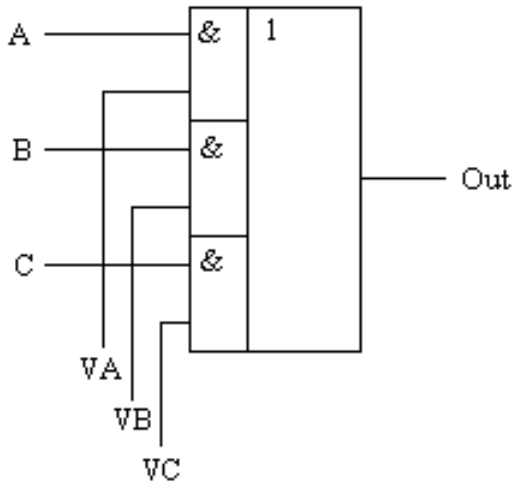


Рис. 5 – Принципиальная схема мультиплексора.

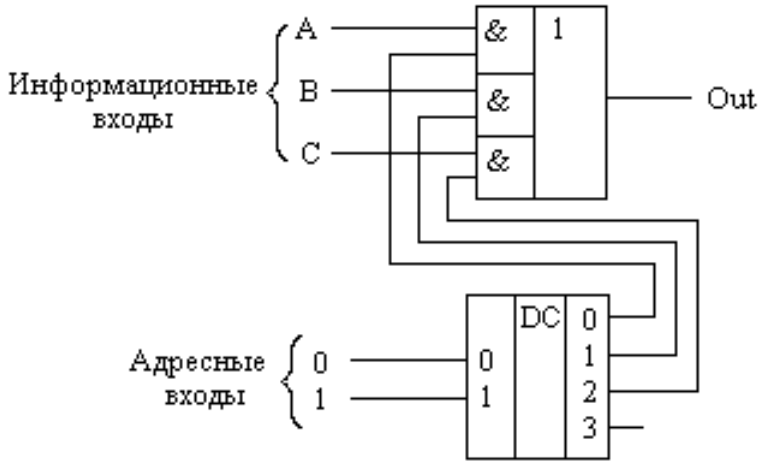


Рис. 6 – Мультиплексор, управляемый двоичным кодом.

Кроме информационных и управляющих входов, схемы мультиплексоров содержат вход разрешения, при подаче на который активного уровня мультиплексор переходит в рабочее состояние. При подаче на вход разрешения пассивного уровня мультиплексор перейдет в нерабочее состояние, в котором сигнал на выходе сохраняет постоянное значение независимо от значений информационных и управляющих сигналов. Число информационных входов у мультиплексоров обычно равно 2, 4, 8 или 16.

На рис. 7 представлен мультиплексор (блок микросхемы, использующийся для моделирования дешифратора в EWB) 8x1 с инверсным входом разрешения *G*, прямым *Y* и инверсным *W*-выходами ($W = \overline{Y}$)

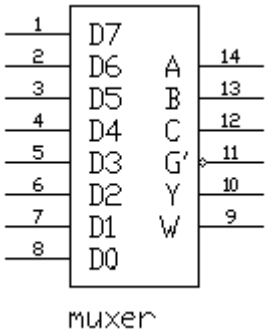


Рис. 7

5. Реализация логических функций

Функционирование мультиплексора, представленного на рис. 7, описывается характеристическим уравнением, связывающим сигнал на выходе (*Y*) с разрешающим (*G*), входными информационными (*D0...D7*) и управляющими (*A*, *B*, *C*) сигналами:

$$Y = \left(\begin{array}{l} \overline{C} \cdot \overline{B} \cdot \overline{A} \cdot D0 + \overline{C} \cdot \overline{B} \cdot A \cdot D1 + \overline{C} \cdot B \cdot \overline{A} \cdot D2 + \overline{C} \cdot B \cdot A \cdot D3 + \\ + C \cdot \overline{B} \cdot \overline{A} \cdot D4 + C \cdot \overline{B} \cdot A \cdot D5 + C \cdot B \cdot \overline{A} \cdot D6 + C \cdot B \cdot A \cdot D7 \end{array} \right) \cdot \overline{G} \qquad (6)$$

Как видно из уравнения, на мультиплексоре можно реализовать логические функции, для чего нужно определить, какие сигналы и логические константы следует подавать на входы мультиплексора.

Логическая функция n переменных определена для 2^n комбинаций значений переменных. Это позволяет реализовать функцию n переменных на мультиплексоре, имеющем n управляющих и 2^n информационных входов. В этом случае каждой комбинации значений аргументов соответствует единственный информационный вход мультиплексора, на который подается значение функции.

Например, требуется реализовать функцию:

$$F_1 = \overline{c} \cdot \overline{b} \cdot \overline{a} \vee c \cdot b \cdot a \vee c \cdot b \cdot \overline{a} \vee \overline{c} \cdot b \cdot a \qquad (7)$$

Эта функция определена только для 8 комбинаций значений переменных, поэтому для её реализации можно использовать мультиплексор 8x1 с тремя управляющими входами. Составим таблицу истинности функции:

Таблица 3 – Таблица истинности логической функции F_1

Вход мультиплексора	**Логические переменные**			**Выход функции**
N	**c**	**b**	**a**	**F_1**
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

Из таблицы видно, что для реализации функции на мультиплексоре необходимо подать на информационный вход мультиплексора с номером *N* сигнал, значение которого равно соответствующему значению функции F_1, т. е. на входы с номерами 1, 2, 4, 5 следует подать уровень логического нуля, а на остальные - уровень логической единицы. Таким образом, при подаче комбинации логических уровней (*a, b, c*)на управляющие входы мультиплексора,

к его выходу подключится вход, значение сигнала на котором равно соответствующему значению функции. Схемная реализация приведена на рис. 8.

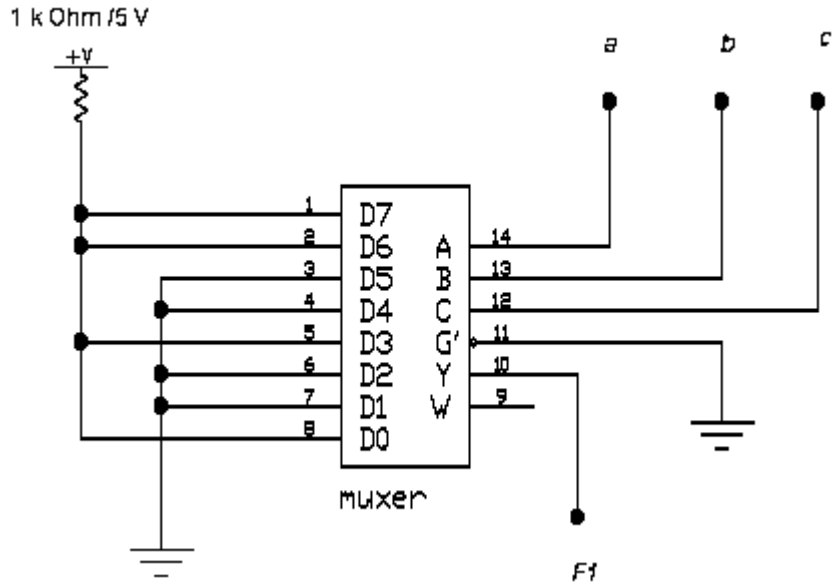


Рис. 8 – Реализация логической функции

При реализации логических функций на информационные входы можно подавать не только константы, но и изменяющиеся входные сигналы. Так, например, рассмотрим другой способ реализации функции F_1, определенной выражением (7). Для этого минимизируем выражение функции с помощью известных логических тождеств (см. лабораторную работу № 1) до вида:

$$F_1 = \overline{c} \cdot \overline{b} \cdot \overline{a} \vee b \cdot c \vee b \cdot a \qquad (8)$$

Составим таблицу истинности функции (8) в зависимости от значений переменных a и b (см. таблицу 4).

Для составления таблицы в выражение (8) подставлялись комбинации a и b и, пользуясь логическим тождествами (см. лабораторную работу № 1) получалось значение функции F_1. Заданную такой таблицей функцию реализуют, как и в предыдущем случае, подав на вход с номером N сигнал, значение которого соответствует значению функции F_1.

Таблица 4 – Таблица истинности упрощенной логической функции F_1

Вход мультиплексора	**Логические переменные**		**Выход функции**
N	**a**	**b**	**F_1**
0	0	0	$\overline{c}$
1	0	1	c
2	1	0	0
3	1	1	1

В данном случае сигналы c и $\overline{c}$ соответствующие переменной c, подаются на информационные входы, как указано в таблице истинности. При этом сокращается число управляющих входов.

Схемная реализация такого способа задания функции приведена на рис. 9.

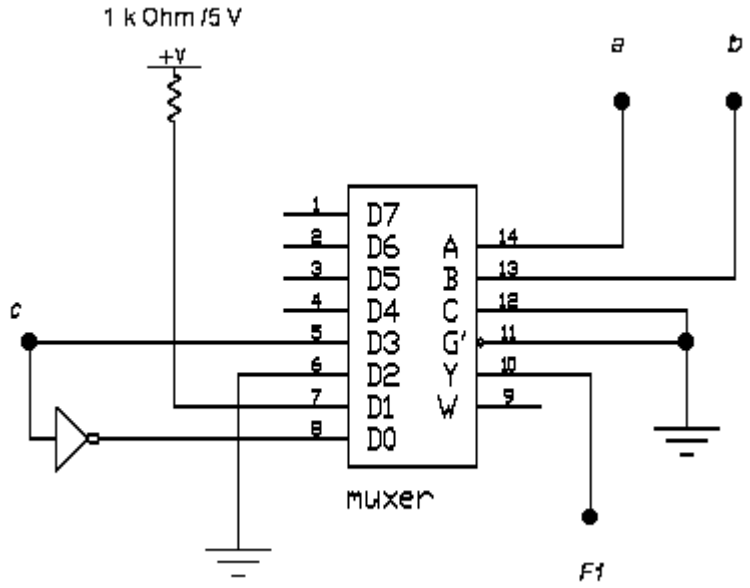


Рис. 9 – Реализация упрощенной логической функции

Так как используются только два адресных входа, управляющий вход C можно заземлить. При этом состояние информационных входов *D4...D7* безразлично. Схема рис. 9 по существу представляет собой мультиплексор 4x1 с двумя управляющими и четырьмя информационными входами.

Если функцию можно представить в виде произведения одночлена на многочлен, то её также можно реализовать при помощи мультиплексора. Как следует из уравнения мультиплексора, сигнал, соответствующий одночлену, нужно подать на вход разрешения. Например, требуется реализовать функцию F_2, описываемую следующим выражением:

$$F_2 = \overline{x} \cdot (d \cdot c \cdot \overline{b} \vee d \cdot \overline{b} \cdot a \vee e \cdot \overline{c} \cdot b \cdot a \vee c \cdot b \cdot a) \qquad (9)$$

При реализации данной функции на мультиплексоре сигнал, соответствующий переменной x, следует подать на его разрешающий вход. Рассмотрим, какие сигналы необходимо подать на управляющие входы мультиплексора. Выражение в скобках можно рассматривать как некоторую функцию f пяти переменных: a, b, c, d, e, из которых наиболее часто используются переменные a, b и c. Поэтому сигналы, соответствующие этим переменным, нужно подать на управляющие входы мультиплексора.

Определим, какие сигналы следует подать на информационные входы, чтобы реализовать функцию f. Для этого составим таблицу истинности функции в зависимости от значений переменных a, b и c (таблица 5).

Таблица 5 – Таблица истинности логической функции F_2

Вход мультиплексора	**Логические переменные**			**Выход функции**
N	**c**	**b**	**a**	f
0	0	0	0	0
1	0	0	1	d
2	0	1	0	0
3	0	1	1	e
4	1	0	0	0
5	1	0	1	d
6	1	1	0	0
7	1	1	1	1

Из таблицы видно, что на информационные входы с номерами N = 0, 2, 4, 6 нужно подать уровень логического нуля. Сигнал, соответствующий переменной d, нужно подать на входы с номерами N = 1, 5, сигнал, соответствующий переменной e, – на вход с номером 3. Схемная реализация такого способа задания функции приведена на рис. 10.

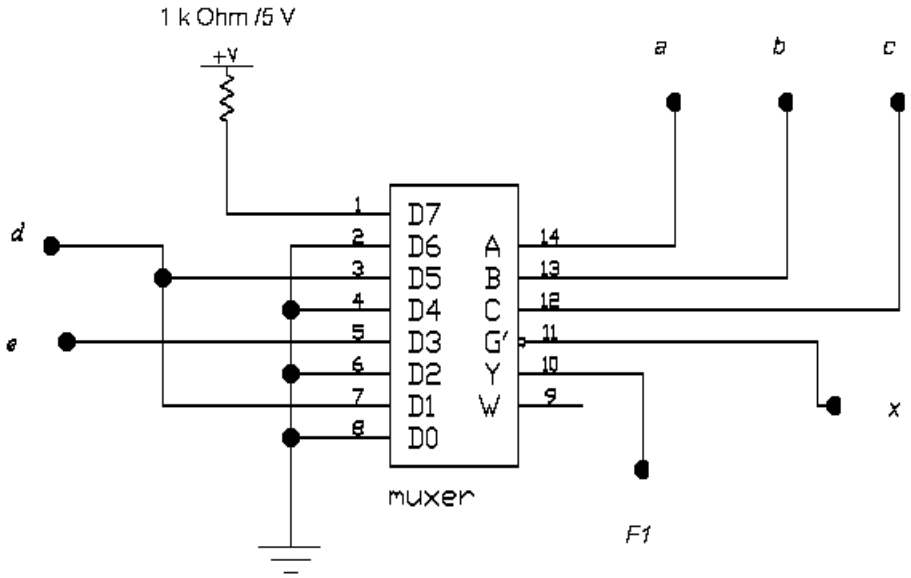


Рис. 10 – Реализация логической функции F_2

Порядок работы

Задание 1. Исследование работы шифратора
Создайте схему изображенную на рис. 1. а.

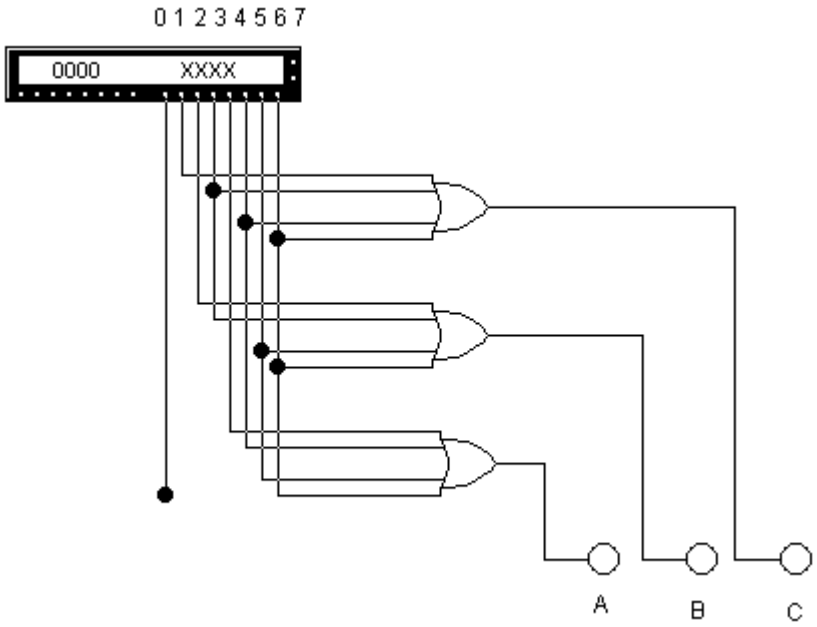


Рис. 1.а – Схема шифратора

Над генератором слов написаны цифры от 0 до 7 – они обозначают номера входов шифратора, на которые соответственно подаются сигналы управления. Сам шифратор составлен из трех элементов «ИЛИ». Выходы шифратора обозначаются здесь *A, B, C*. Где *A* – старший бит, *B* – средний бит, а *C* – младший бит двоичного числа, получаемого на выходе. (Это число показывает, на какой из входов подан логический сигнал).

Запрограммируйте генератор слов, так что бы на его выходах сформировалась двоичная последовательность, эмулирующая поочередную подачу на вход шифратора сигнала логической единицы. Пошагово изменяя значения входов дешифратора (кнопкой «Step» в генераторе слов), заполните таблицу истинности шифратора (таблица 1.а).

Таблица 1.а – Таблица истинности шифратора

Входы шифратора								**Выходы шифратора**			
								Двоичное число			**Десятичное число**
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**A**	**B**	**C**	
0	0	0	0	0	0	0	0				
0	1	0	0	0	0	0	0				
0	0	1	0	0	0	0	0				
0	0	0	1	0	0	0	0				
0	0	0	0	1	0	0	0				
0	0	0	0	0	1	0	0				
0	0	0	0	0	0	1	0				
0	0	0	0	0	0	0	1				

Переведите полученное двоичное число, составленное из разрядов *A B C* в десятичное. Сделайте вывод о работе шифратора.

Задание 2. Исследование работы дешифраторов

а) Исследование развернутой схемы дешифратора

Создайте схему изображенную на рис. 2. а.

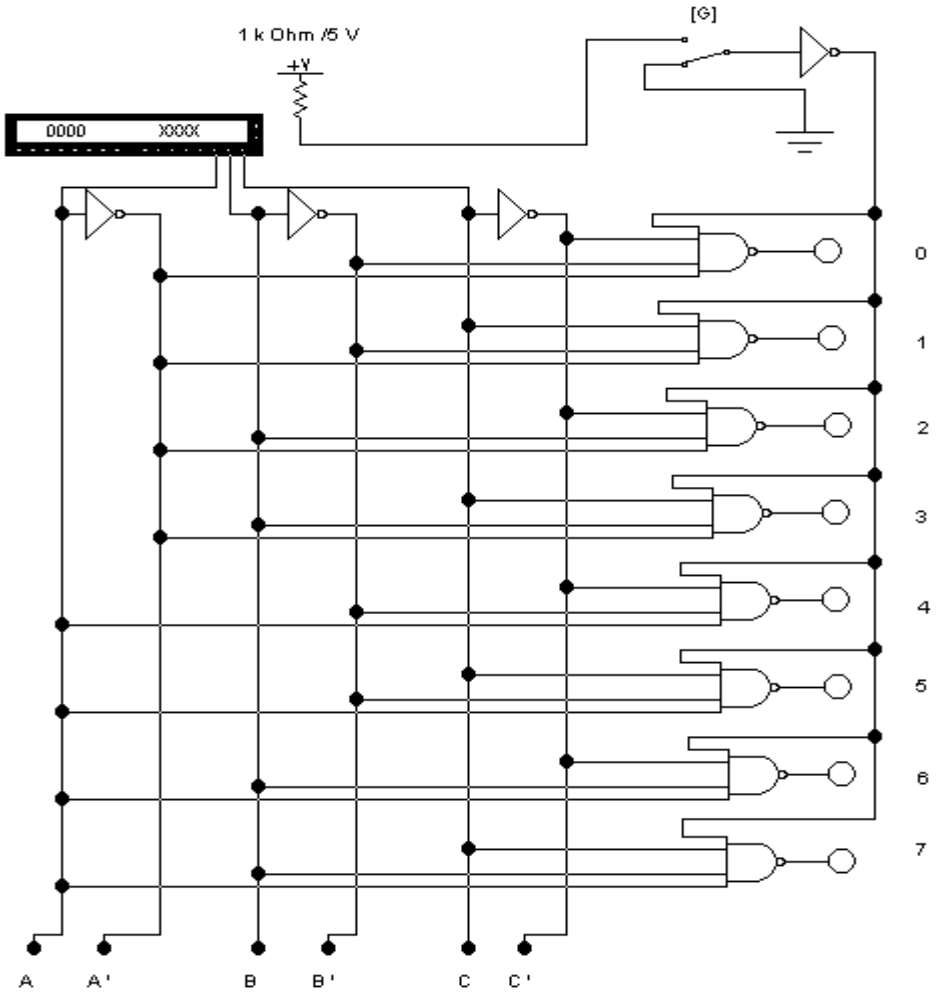


Рис. 2.а – Развернутая схема дешифратора

Здесь представлен дешифратор 3*8 (3 входа, 8 выходов). Дешифратор составлен из элементов «И-НЕ». *A, B, C* – входы дешифратора, 0, 1, …, 7 – выходы дешифратора, *G* – вход разрешения.

Запрограммируйте генератор слов так, что бы на его выходе сформировать все возможнее комбинации трехразрядного двоичного числа. Подавая на вход дешифратора различные комбинации двоичного числа *A, B, C* (кнопкой «Step» в генераторе слов) и разрешения *G* (ключом G), заполните таблицу истинности дешифратора (таблица 2.а).

Сделайте вывод о работе дешифратора.

Таблица 2.а – Таблица истинности развернутой схемы дешифратора

Входы дешифратора					Выходы дешифратора							
Число	**A**	**B**	**C**	**G**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	0	0	0								
1	0	0	1	0								
2	0	1	0	0								
3	0	1	1	0								
4	1	0	0	0								
5	1	0	1	0								
6	1	1	0	0								
7	1	1	1	0								
0	0	0	0	1								
1	0	0	1	1								
2	0	1	0	1								
3	0	1	1	1								
4	1	0	0	1								
5	1	0	1	1								
6	1	1	0	1								
7	1	1	1	1								

*б) Исследование работы схемы дешифратора 3*8 в основном режиме*
Создайте схему изображенную на рис. 2.б

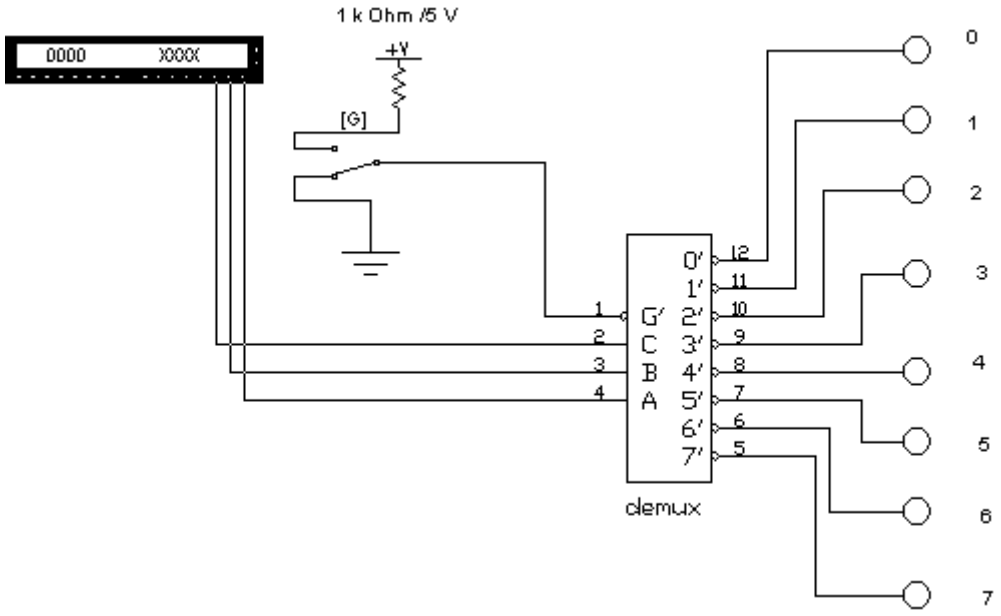


Рис. 2.б – Схема дешифратора

Запрограммируйте генератор слов аналогично пункту *а*. Подавая на вход дешифратора различные комбинации двоичного числа *A, B, C* (кнопкой «Step» в генераторе слов) и разрешения *G* (ключом *G*), заполните таблицу истинности дешифратора (таблица 2.б, аналогично таблице 2.а).

Сделайте вывод о работе дешифратора.

Сравните таблицы 2.а и 2.б.

*в) Исследование работы схемы дешифратора 3*8 в режиме2*4*
Создайте схему изображенную на рис. 2.в.

В схеме рис. 2.в подключите вход *C* к общему проводу (земле), задав *C=0*. Изменяя сигналы на входах *B* и *A*, пронаблюдайте уровни сигналов на выходах схемы с помощью пробников. Укажите выходы, на которых уровень сигнала не меняется.

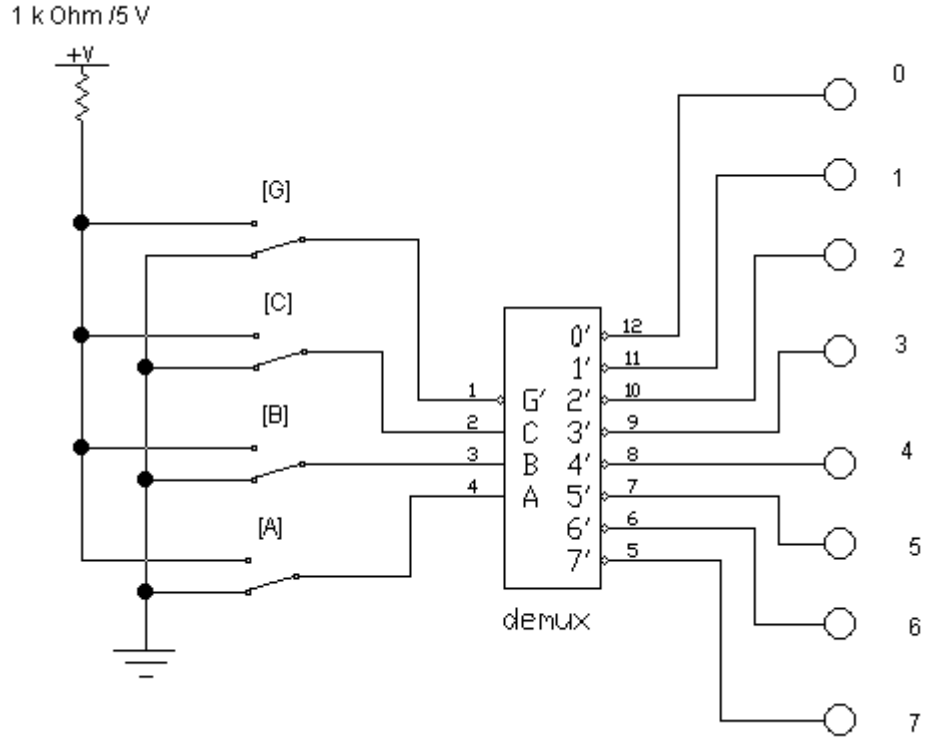


Рис. 2.в – Дешифратор в режиме 2*4

Подключите вход *C* к источнику питания (логической единицы) задав *C=1*. Аналогично изменяя сигналы на входах *B* и *A*, пронаблюдайте уровни сигналов на выходах схемы, с помощью пробников. Укажите выходы, на которых уровень сигнала не меняется.

Заземлите вход *B* (*B=0*), подавая на входы *A* и *C* все возможные комбинации логических уровней, сделайте вывод о работе схемы в этом случае.

Сформулируйте принцип, по которому можно использовать дешифраторы на меньшую разрешающую способность, чем на ту которую он рассчитан.

Задание 3. Применение дешифраторов

Применение дешифраторов в цифровой технике весьма различно. Наиболее часто они используются как формирующие элементы, например, в схемах различных микроконтроллеров для формирования сигнала выбора определенной микросхемы. В этом случае на входы дешифратора подаются сигналы с шины адреса микропроцессора. При этом каждому участку адресного пространства ставится в соответствие определенное назначение (например, для ОЗУ выделяется первых 1024 байт, для ПЗУ - следующие 2048 байт и так далее). Дешиф-

ратор, в таком случае, помогает сформировать сигнал управления (выбора микросхемы), так как согласно его таблице истинности на каждом выходе активный уровень формируется лишь однажды, при вполне определенной комбинации входных сигналов. Таким образом, не составляет особого труда сначала составить карту памяти (это подробная запись всего содержимого адресного пространства, с записью начального и конечного адреса каждого блока), а затем по ней выделить те разряды адреса, которые однозначно определяют обращение к тому или иному блоку адресного пространства. Именно эти разряды и будут являться входами для дешифраторов. Выход каждого дешифратора будет соединен с входом разрешения работы той микросхемы, которая в данном случае необходима.

В данном задании внимание в основном будет акцентировано на другом способе применения дешифраторов, который, в общем, является базовым и для всех других. Это использование дешифратора в совокупности с логическими элементами.

Помимо этого в основном данном задании будет введено понятие временной диаграммы, которая также очень часто используется в технике для иллюстрации работы цифровых устройств.

Временная диаграмма в общем случае представляет собой график, по оси абсцисс которой откладывается время в тактах, а по оси ординат необходимая цифровая величина (вход или выход цифровой схемы), принимающее значения «0» или «1». Необходимо отметить, что чаще всего осей ординат несколько, и они располагаются друг под другом. Собственно говоря, именно в этом и состоит основное преимущество применения временных диаграмм, работа цифрового устройства наглядно представляется во времени. Простейший пример временной диаграммы приведен на рис. 3.а.

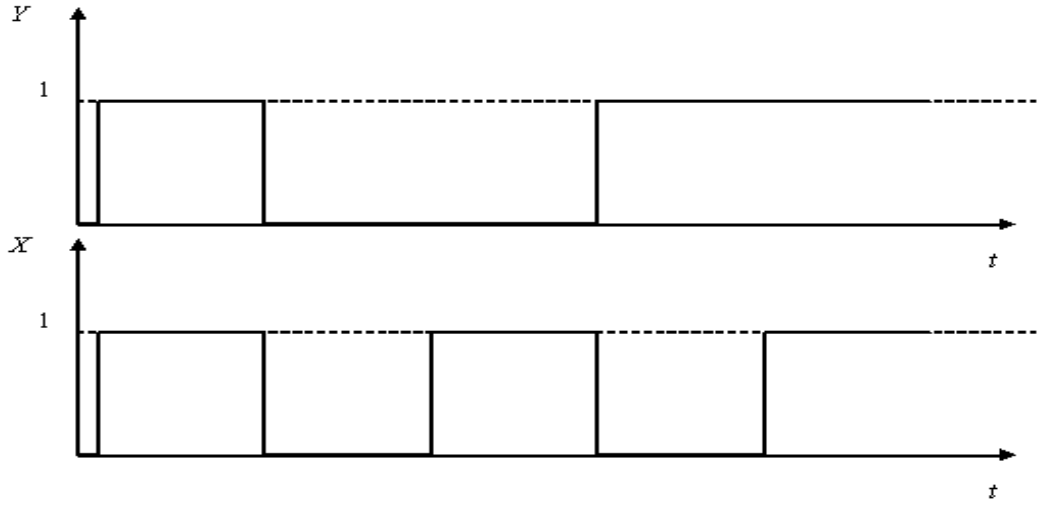


Рис. 3.а – Временная диаграмма

По таблице истинности логической функции можно без труда построить временную диаграмму. В *EWB* кроме инструментов автоматического построения таблицы истинности присутствуют инструменты и для построения временных диаграмм. Это построение можно осуществить с помощью «Logic An-

alyzer». Общий вид окна свойств данного инструмента приведено на рис. 3.б (окно свойств вызывается двойным щелчком по изображению данного инструмента).

Построение временных диаграмм начинается автоматически при включении всей схемы. В меню «Clock per division» можно задавать масштаб просмотра по оси абсцисс (в данном случае времени). В области «Clock» кнопкой «Set» можно изменить установки внутреннего генератора времени (ось абсцисс), в частности задать частоту анализа («Internal clock rate»). В области «Trigger» можно изменить установки триггера (ось ординат).

Соберите схему изображенную на рис. 3.в.

Запрограммируйте генератор слов так, что бы на его выходе сформировать все возможные комбинации трехразрядного двоичного числа.

Установите ключ *G* в разрешающее положение. Подавая на вход дешифратора различные комбинации двоичного числа *A, B, C* (кнопкой «Step» в генераторе слов), постройте временную диаграмму работы дешифратора с логическими элементами на выходе. Оси диаграммы выберите согласно рис. 3.г.

Получите логическую функцию данного цифрового устройства. Для этого удобнее предварительно построить таблицу истинности, а по ней аналогично пунктам, изложенным выше построить саму логическую функцию.

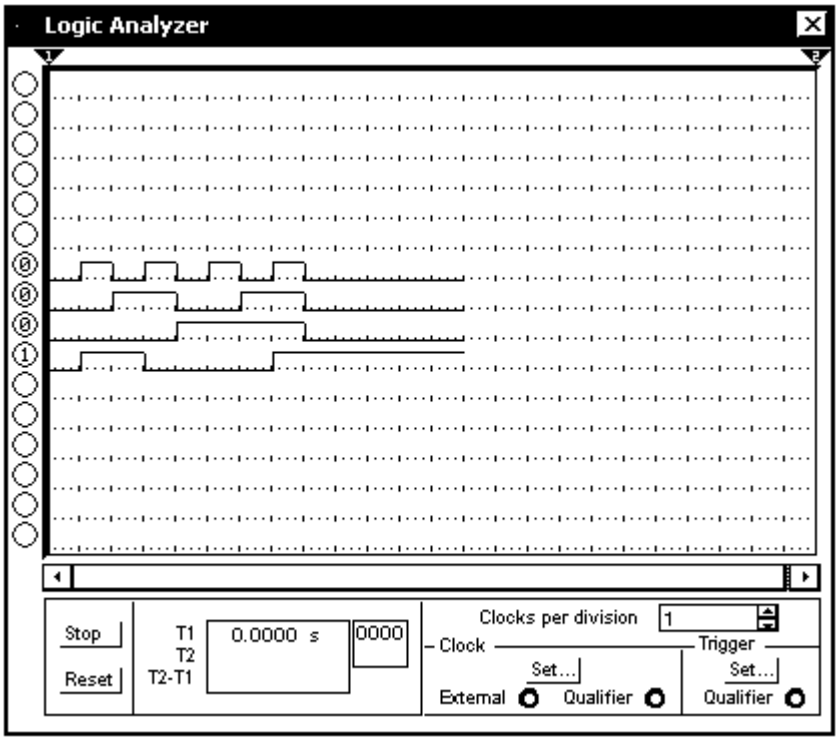


Рис. 3.б – Окно свойств Логического Анализатора

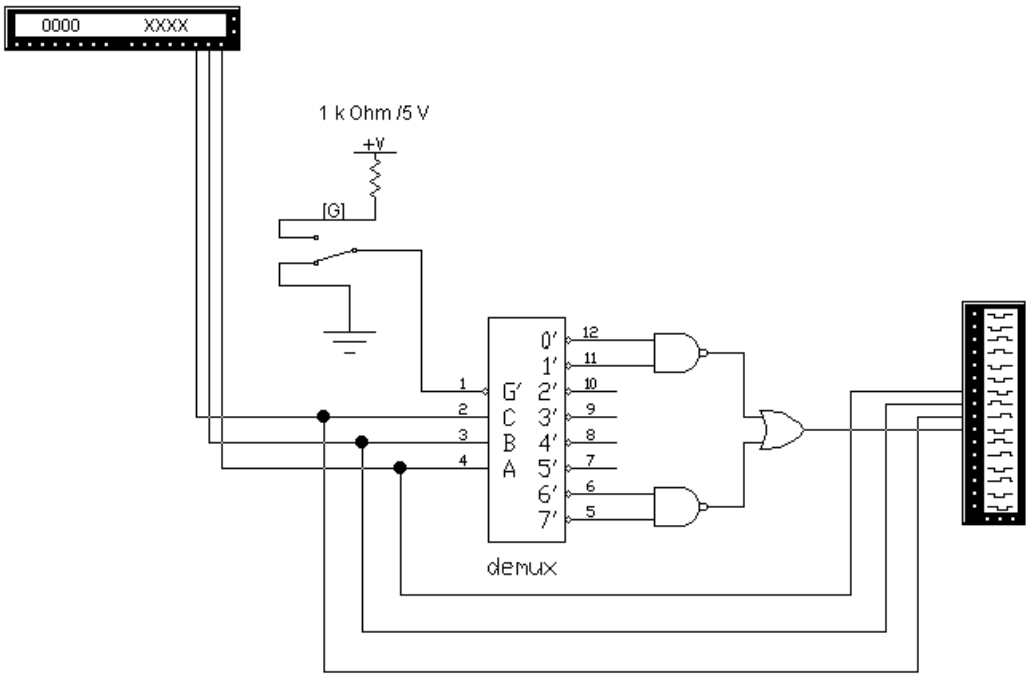


Рис. 3.в – Дешифратор с логическим элементами

Рис. 3.г– Временная диаграмма

Задание 4. Исследование работы мультиплексора

а) Исследование развернутой схемы мультиплексора

Создайте схему изображенную на рис. 4.а.

Здесь, на рис. 4.а приняты следующие условные обозначения: *G* – сигнал разрешения, *d0, d1* – информационные входы, *A* – адресный вход.

Исследуйте поведение схемы мультиплексора, задавая различные сочетания логических уровней на входе схемы, заполнив при этом таблицу истинности, приведенную ниже (таблица 4.а).

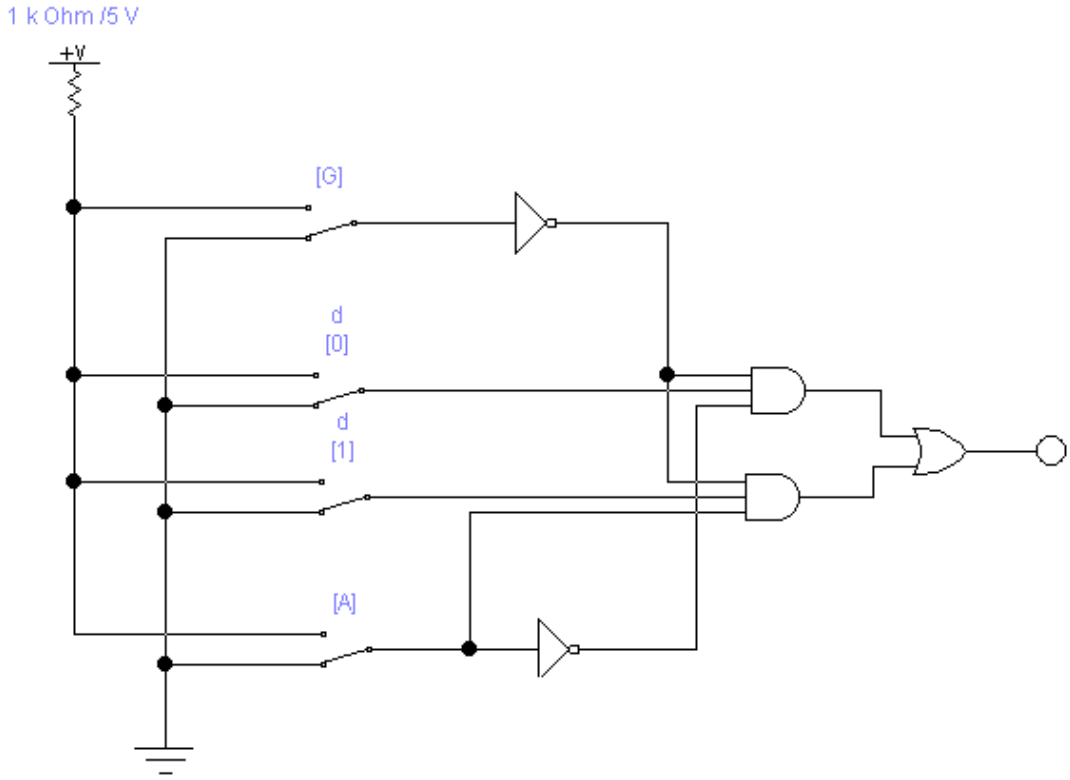


Рис. 4.а – Развернутая схема мультиплексора

Таблица 4.а – Развернутая схема мультиплексора

Входы		**Адрес**	**Выход**
d0	**d1**	**A**	**Q**
0	0	0	
0	1		
1	0		
1	1		
0	0	1	
0	1		
1	0		
1	1		

*б) Исследование работы схемы мультиплексора 3*8 в основном режиме*
Соберите схему представленную на рис. 4.б.

Запрограммируйте генератор слов так, что бы на адресные входы мультиплексора (*A, B, C*) подавались все возможные комбинации логических уровней.

Подавая на адресный вход мультиплексора различные комбинации адреса *A, B, C* (кнопкой «Step» в генераторе слов), заполните таблицу истинности мультиплексора (таблица 4.б).

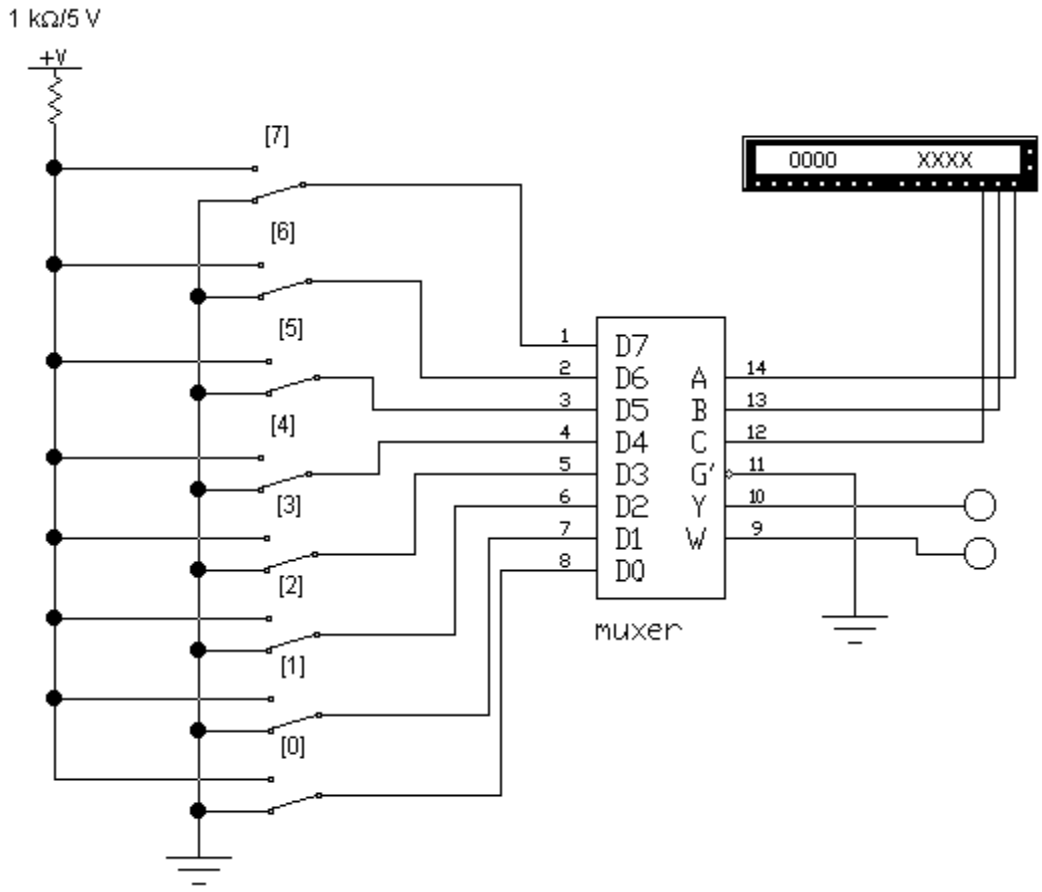


Рис. 4.б – Схема мультиплексора *3*8*

Таблица 4.б – Таблица истинности мультиплексора

Входы											**Выход**	
d0	**d1**	**d2**	**d3**	**d4**	**d5**	**d6**	**d7**	**C**	**B**	**A**	**Y**	**W**
								0	0	0		
								0	0	1		
								0	1	0		
								0	1	1		
								1	0	0		
								1	0	1		
								1	1	0		
								1	1	1		

При этом комбинации соответствующих сигналов на входе *D0, D1, …,D7* выбираются в соответствии с вариантом заданным таблицей 4.в.

По таблице 4.б проследите правильность работы мультиплексора.

Задание 5. Реализация логической функции с помощью мультиплексора

По заданной таблице истинности (таблица 5.а) реализуйте логическую функцию с помощью мультиплексора. Выбор задания для выхода логической функции производится согласно вашему варианту. Реализация логической функции может быть осуществлена с помощью любого стандартного мультиплексора.

Таблица 4.в – Варианты входов данных таб. 4.б

Входы								Вариант
d0	**d1**	**d2**	**d3**	**d4**	**d5**	**d6**	**d7**	
0	1	0	1	0	1	0	1	1
0	0	1	1	0	0	1	1	2
1	1	1	0	0	0	1	1	3
1	1	1	1	0	0	0	0	4
1	0	0	0	1	0	0	1	5
0	0	0	0	1	1	1	1	6
1	1	0	0	0	0	1	1	7
0	0	0	0	0	1	1	1	8
1	0	1	1	0	1	0	1	9
1	0	0	0	1	0	1	0	10

Создайте схему с вашим вариантом реализации логической функции в EWB, занесите полученную схему в отчет.

С помощью Логического Анализатора постройте временные диаграммы работы вашей логической функции. По ним проверьте правильность функционирования схемы.

Таблица 5.а – Варианты таблицы истинности

Значения логических переменных (для всех вариантов)			**Варианты задания**									
			1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
A	**B**	**D**	**Значения логической функции (для каждого варианта)**									
0	0	0	0	0	0	0	1	0	1	0	1	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	1	1	0	0	1	1	0	0	0	1	0	1
1	0	0	1	1	0	1	0	1	1	1	1	1
1	0	1	1	1	0	0	1	1	0	1	1	1
1	1	0	1	0	1	1	0	1	0	0	1	0
1	1	1	0	0	1	1	1	1	0	0	1	1

Контрольные вопросы

1. Дайте определение шифратору, дешифратору.
2. Чем отличается схема шифратора от схемы дешифратора?
3. Как в простейшем случае реализовать на дешифраторе демультиплексор? Постройте полученную схему в случае использования дешифратора 3*8.
4. Как влияет сигнал управления на работу логической схемы?
5. Как из двух дешифраторов 2*4 сделать один 3*8?
6. Как изменить расширенную схему дешифратора (рис. 2.а) так, что бы активным уровнем выходного сигнала данной схемы, был «0»?
7. Дайте определение мультиплексору.
8. Приведите примеры применения мультиплексоров.
9. Как на мультиплексоре можно реализовать логическую функцию?
10. Любую ли логическую функцию можно реализовать на мультиплексоре?
11. Дайте определение временной диаграммы.
12. Можно ли по произвольной временной диаграмме составить таблицу истинности?
13. Можно ли по таблице истинности составить временную диаграмму?
14. В каком виде записывается логическая функция мультиплексора?

Упражнения

Упражнение 1. Применение дешифраторов

Сформируйте сигнал выбора определенной микросхемы памяти в микроконтроллере с использованием стандартных дешифраторов.

Микропроцессор, входящий в состав микроконтроллера, имеет 8 разрядную шину адреса, то есть может работать с объемом памяти 2^8=256 байт (разряды нумеруются *A0 – A7*, начиная с младшего). По каждому адресу хранится информация, объем которой определяется разрядностью шины данный микропроцессора и в данном упражнении не важен. В общем случае вся память делится на определенные блоки, по характеру их использования (например, для ОЗУ, ПЗУ, портов ввода/вывода). Каждый блок памяти имеет начальный и конечный адрес.

Необходимо внутри объема всей доступной памяти сформировать блок, начальный и конечный адреса которых приведены в таблице 1.у. Данный блок будет использоваться для адресации ОЗУ.

Таблица 1.у – Варианты карты памяти для блока ОЗУ

Вариант	**Адреса**															
	Начальный адрес								**Конечный адрес**							
	A7	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**	**A7**	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
2	0	0	0	1	0	0	0	0	0	0	0	1	1	1	1	1
3	0	0	1	0	0	0	0	0	0	0	1	0	1	1	1	1
4	0	0	1	1	0	0	0	0	0	0	1	1	1	1	1	1
5	0	1	0	0	0	0	0	0	0	1	0	0	1	1	1	1
6	0	1	0	1	0	0	0	0	0	1	0	1	1	1	1	1
7	0	1	1	0	0	0	0	0	0	1	1	0	1	1	1	1
8	0	1	1	1	0	0	0	0	0	1	1	1	1	1	1	1
9	1	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1
10	1	0	0	1	0	0	0	0	1	0	0	1	1	1	1	1

Для выполнения задания необходимо проделать следующее:

проанализировать все адреса, входящие в ваш блок, выделить те разряды шины адреса, значения которых одинаковы для всех адресов вашего блока;

из всех полученных комбинаций выбрать достаточное количество разрядов шины адреса, которые однозначно определяют обращение именно к вашему блоку памяти и не повторяются во всем другом объеме доступной памяти;

полученные разряды шины адреса и будут входами дешифратора, а его выход и есть искомый сигнал выбора определенной микросхемы.

Выберите подходящий дешифратор. Занесите полученную схему в отчет, отметьте номера используемых разрядов шины адреса.

Упражнение 2. Применение мультиплексора

По заданной логической функции постройте схему с использованием мультиплексора.

Таблица 2.у – Варианты логических функций

Вариант	Логическая функция
1	$\overline{C}\cdot B\cdot\overline{A}+\overline{C}\cdot B\cdot A+C\cdot\overline{B}\cdot\overline{A}+C\cdot\overline{B}\cdot A+C\cdot B\cdot\overline{A}+C\cdot B\cdot A$
2	$\overline{C}\cdot\overline{B}\cdot A+\overline{C}\cdot B\cdot\overline{A}+\overline{C}\cdot B\cdot A+C\cdot B\cdot\overline{A}+C\cdot B\cdot A$
3	$\overline{C}\cdot\overline{B}\cdot\overline{A}+\overline{C}\cdot\overline{B}\cdot A+C\cdot\overline{B}\cdot\overline{A}+C\cdot\overline{B}\cdot A+C\cdot B\cdot\overline{A}+C\cdot B\cdot A$
4	$\overline{C}\cdot\overline{B}\cdot\overline{A}+\overline{C}\cdot\overline{B}\cdot A+C\cdot\overline{B}\cdot A+C\cdot B\cdot\overline{A}+C\cdot B\cdot A$
5	$\overline{C}\cdot\overline{B}\cdot\overline{A}+\overline{C}\cdot\overline{B}\cdot A+\overline{C}\cdot B\cdot\overline{A}+\overline{C}\cdot B\cdot A+C\cdot B\cdot\overline{A}+C\cdot B\cdot A$
6	$\overline{C}\cdot\overline{B}\cdot\overline{A}+\overline{C}\cdot\overline{B}\cdot A+\overline{C}\cdot B\cdot A+C\cdot\overline{B}\cdot\overline{A}+C\cdot B\cdot\overline{A}$
7	$\overline{C}\cdot\overline{B}\cdot\overline{A}+\overline{C}\cdot\overline{B}\cdot A+\overline{C}\cdot B\cdot\overline{A}+\overline{C}\cdot B\cdot A+C\cdot\overline{B}\cdot\overline{A}+C\cdot\overline{B}\cdot A$
8	$\overline{C}\cdot\overline{B}\cdot\overline{A}+\overline{C}\cdot B\cdot A+C\cdot\overline{B}\cdot\overline{A}+C\cdot B\cdot\overline{A}+C\cdot B\cdot A$
9	$\overline{C}\cdot\overline{B}\cdot\overline{A}+\overline{C}\cdot B\cdot\overline{A}+C\cdot\overline{B}\cdot\overline{A}+C\cdot\overline{B}\cdot A+C\cdot B\cdot\overline{A}+C\cdot B\cdot A$
10	$\overline{C}\cdot\overline{B}\cdot A+\overline{C}\cdot B\cdot\overline{A}+C\cdot\overline{B}\cdot\overline{A}+C\cdot\overline{B}\cdot A+C\cdot B\cdot\overline{A}$

ЛАБОРАТОРНАЯ РАБОТА № 3

Изучение работы триггеров

Цель работы:

1. Изучение структуры и исследование работы асинхронных и синхронных триггеров.
2. Исследование функций переходов и возбуждения основных типов триггеров.
3. Изучение взаимозаменяемости триггеров различных типов.

Приборы и элементы:

Логические пробники (панель «Indicators/Red probe»)
Источник напряжения + 5 В (панель «Basic/Pull-Up Resistor»)
Земля (панель «Sources/Ground»)
Двухвходовые элементы И, И-НЕ, ИЛИ, ИЛИ-НЕ (панель «Logic Gates/2-Input AND, NAND, OR, NOR Gates»)
Двухпозиционные переключатели (панель «Basic/Switch»)
Базовые триггеры RS, JK, D (панель «Digital/»)
Схемы различных серий (панель «Digital/MUX,DEC»)

Краткие теоретические сведения

Последовательные цифровые устройства часто называют последовательными схемами, последовательными автоматами, дискретными автоматами с памятью, многотактными автоматами. Простейшим примером устройств данного типа являются триггеры.

Триггеры

Триггер – простейшая цифровая схема последовательного типа. У рассмотренных в предыдущих работах комбинационных схем состояние выхода *Y* в любой момент времени определяется только текущим состоянием входа *X*. В отличие от них, состояние выхода последовательной схемы (цифрового автомата) зависит еще и от внутреннего состояния схемы *Q*.

Таким образом, цифровой автомат является не только преобразователем, но и хранителем предшествующей и источником текущей информации (состояния). Это свойство обеспечивается наличием в схемах обратных связей.

Триггер имеет два устойчивых состояния: *Q=1* и *Q=0*, поэтому его иногда называют бистабильной схемой. В каком из этих состояний окажется триггер, зависит от сигналов на информационном входе триггера и от его предыдущего состояния, то есть он имеет память. Можно сказать, что триггер является элементарной ячейкой памяти.

Тип триггера определяется алгоритмом его работы. В зависимости от алгоритма триггер может иметь установочные, информационные и управляющие входы. Установочные входы служат для перевода триггера в одно из определенных состояний, независимо от состояния других входов. Входы управления разрешают запись данных, подающихся на информационные входы. Наиболее распространенными являются триггеры *RS*, *JK*, *D* и *T* типов.

1. Триггер типа RS

RS-триггер – простейший автомат с памятью, который может находиться в двух состояниях. Триггер имеет два установочных входа: установки *S* (*set* – установка) и сброса *R* (*reset* – сброс), на которые подаются входные сигналы от внешних источников. При подаче на вход установки активного логического уровня триггер устанавливается в «1» ($Q=1, \bar{Q}=0$), при подаче активного уровня на вход сброса триггер устанавливается в «0» ($\bar{Q}=1, Q=0$). Если подать на оба входа установки (возбуждения) пассивный уровень, то триггер будет сохранять предыдущее состояние выходов: $Q=0$ ($\bar{Q}=1$) либо $Q=1$ ($\bar{Q}=0$). Каждое состояние устойчиво и поддерживается за счет действия обратных связей.

Для триггеров этого типа является недопустимой одновременная подача активного уровня на оба входа установки, т. к. триггер по определению не может одновременно быть установлен в ноль и единицу. На практике, подача активного уровня на установочные входы приводит к тому, что это состояние не может быть сохранено и невозможно определить, в каком состоянии будет находиться триггер при последующей подаче на установочные входы сигналов пассивного уровня.

На рис. 1 и 2 показаны два вида *RS-триггера*, выполненных на элементах «ИЛИ-НЕ» и «И-НЕ».

Рис. 1 – *RS-триггер* на элементах «ИЛИ-НЕ»

Рис. 2 – *RS-триггер* на элементах «И-НЕ»

Для схемы на рис. 1 активным уровнем является уровень логической единицы, для схемы на рис. 2 – уровень логического нуля. Схема на рис. 2 получила название *RS-триггера* с инверсными входами или просто – $\overline{RS}$*-триггер*.

RS-триггер является основным узлом для построения последовательных схем. Напомним, что название схем такого типа «последовательные» означает, то, что состояние выхода зависит от того, в какой последовательности на входы, подаются сигналы, и каково было предшествующее внутреннее состояние. Так, если в *RS-триггере* (рис. 1) вначале установить комбинацию *R=0, S=1*, а потом перейти к *R=0, S=0*, то состояние выхода будет *Q=1*. Если же вначале установить комбинацию *R=1, S=0*, а потом перейти к *R=0, S=0*, то состояние выхода будет другим – *Q=0*, несмотря на одинаковые комбинации сигналов на входах. Таким образом, при одном и том же входном наборе *R=0, S=0* выход триггера может находиться в разных состояниях.

Условия переходов триггеров из одного состояния в другое (алгоритм работы) можно описать табличным, аналитическим или графическим способами. Табличное описание работы RS-триггера (рис. 1) представлено в таблице 1 (таблица переходов и таблица функций возбуждения).

Таблица 1 – Таблица переходов и функций возбуждения RS-триггера

Таблица переходов			**Таблица функций возбуждений**			
R	**S**	**Q_{t+1}**	**Q_t**	**Q_{t+1}**	**R**	**S**
0	0	Q_t	0	0	x	0
0	1	1	0	1	0	1
1	0	0	1	0	1	0
1	1	-	1	1	0	x

В таблицах использованы следующие обозначения: «Q_t» – предшествующее состояние выхода; «Q_{t+1}» – новое состояние, устанавливающееся после перехода (возможно $Q_{t+1} = Q_t$); «*x*» – безразличное значение сигнала: «0» или «1»; «-» – запрещенное состояние.

Аналитическое описание (характеристическое уравнение) можно получить из таблицы 1 по правилам алгебры логики.

$$Q_{t+1} = \overline{R}S + \overline{R}Q_t = \overline{R}(S + Q_t) \qquad (1)$$

Зависимость Q_{t+1} от Q_t характеризует свойство запоминания предшествующего состояния.

Таблица 1 показывает, что схема, которая находилась в состоянии *Q=0*, сохраняет это состояние как при воздействии входного набора *R=0, S=0*, так и при воздействии *R=1, S=0*. Если же на вход схемы, находящейся в состоянии *Q=0*, подействовать набором *R=0, S=1*, то она переходит в состояние *Q=1* и сохраняет его при входных наборах *R=0, S=1*, либо *R=0, S=0*.

Таблицы переходов и функций возбуждения легче запомнить, держа в уме следующее: вход *S* – вход установки, то есть при подаче на него логической единицы триггер должен перейти в активное состояние; установить триггер в ноль можно только подав, сигнал сброса *R*; неопределенным является сигнал одновременного сброса и установки *R=1, S=1*; сигнал *R=0, S=0* – сохраняет предыдущее состояние.

Схема триггера позволяет запоминать состояние логической схемы, но так как в начальный момент времени может возникать переходный процесс (в цифровых схемах этот процесс называется «опасные гонки»), то запоминать

состояния логической схемы нужно только в определённые моменты времени, когда все переходные процессы закончены. Таким образом цифровые схемы требуют синхросигнала. Схемы с сигналом управления (синхронизации) представлены ниже.

2. JK-триггер

Триггер JK-типа имеет более сложную по сравнению с *RS-триггером* структуру и более широкие функциональные возможности. Помимо информационных входов *J* и *K* и прямого и инверсного выходов *Q* и $\overline{Q}$, *JK-триггер* имеет вход управления *C* (этот вход также называют тактирующим или счетным), а также асинхронные установочные *R* и *S* входы. Обычно активными уровнями установочных сигналов являются нули, как в схеме *RS-триггера* изображенной на рис. 2. Установочные входы имеют приоритет над остальными. Активный уровень сигнала на входе *S* устанавливает триггер в состояние *Q=1*, а активный уровень сигнала на входе *R* – в состояние *Q=0*, независимо от сигналов на остальных входах.

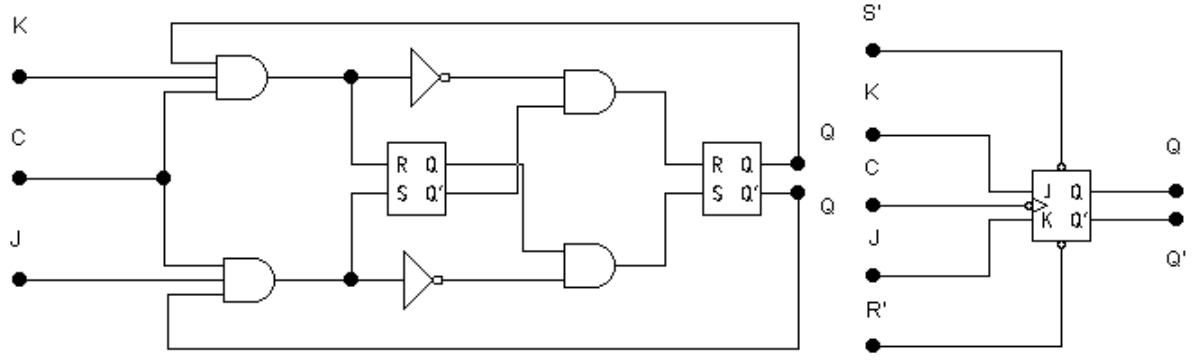
Если на входы установки одновременно подать пассивный уровень сигнала, то состояние триггера будет изменяться по фронту импульса на счетном входе в зависимости от состояния входов *J* и *K*, как показано в таблице 2.

Таблица 2 – Таблица переходов и возбуждения *JK-триггера*

Таблица переходов			**Таблица функций возбуждений**			
J	**K**	$\mathbf{Q_{t+1}}$	$\mathbf{Q_t}$	$\mathbf{Q_{t+1}}$	**K**	**J**
0	0	Q_t	0	0	x	0
0	1	0	0	1	0	1
1	0	1	1	0	1	0
1	1	Q'_t	1	1	0	x

Один из вариантов функциональной схемы *JK-триггера* с входами установки логическим нулем и его условное графическое обозначение приведены на рис. 3. Временные диаграммы его работы при *R=S=1* приведены на рис. 4.

В данном случае все изменения выхода происходят только в момент отрицательного перепада тактового сигнала.



1) Расширенная схема 2) Условное обозначение

Рис. 3 – Схема *JK-триггера*

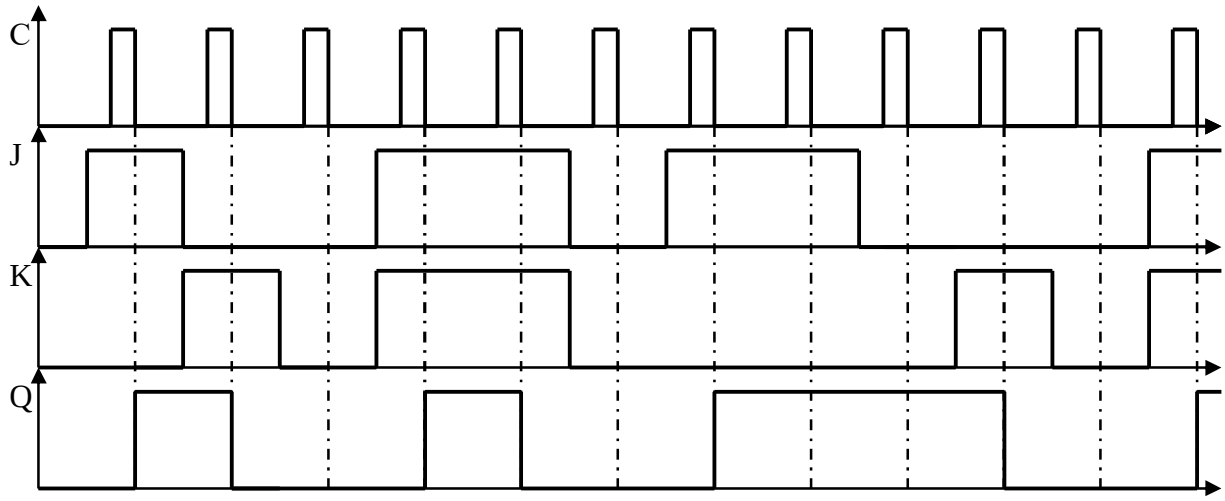


Рис. 4 – Временная диаграмма *JK-триггера*

3. D-триггер

D-триггер имеет один информационный вход *D* (*data* – данные). Информация с входа *D* заносится в триггер по положительному перепаду импульса на счетном входе *C* и сохраняется до следующего положительного перепада на счетном входе триггера. Помимо счетного *C* и информационного *D* входов триггер снабжен асинхронными установочными *R* и *S* входами. Установочные входы имеют наивысший приоритет. Они устанавливают триггер независимо от сигналов на входах *C* и *D*. Функционирование *D-триггера* описывается таблицей 3 переходов и возбуждения и диаграммами рис. 5.

Таблица 3 – Таблица переходов и возбуждения *D-триггера*

Таблица переходов		**Таблица функций возбуждений**		
D	$\mathbf{Q_{t+1}}$	$\mathbf{Q_t}$	$\mathbf{Q_{t+1}}$	**D**
0	0	0	0	0
		0	1	1
1	1	1	0	0
		1	1	1

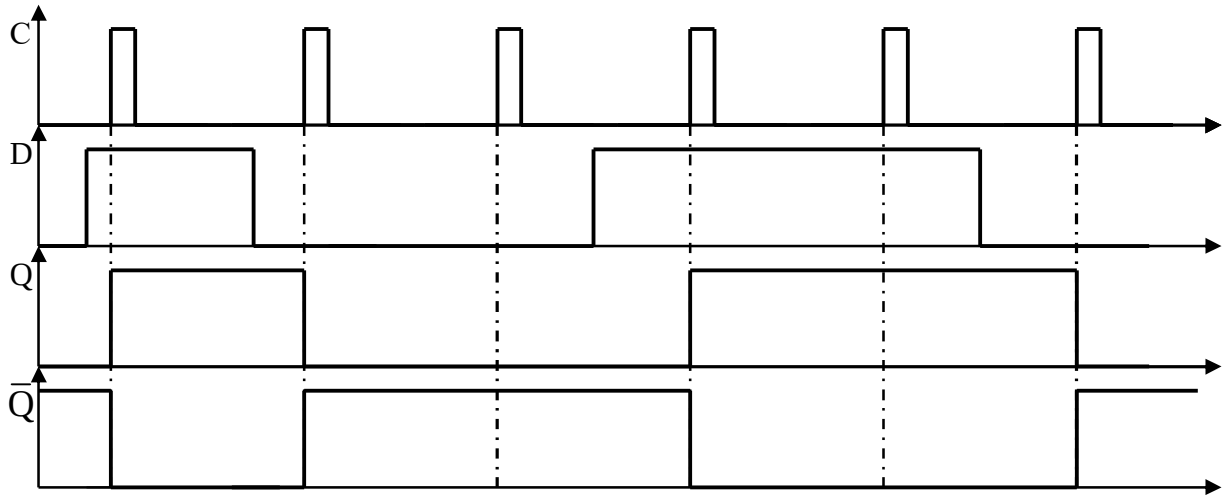
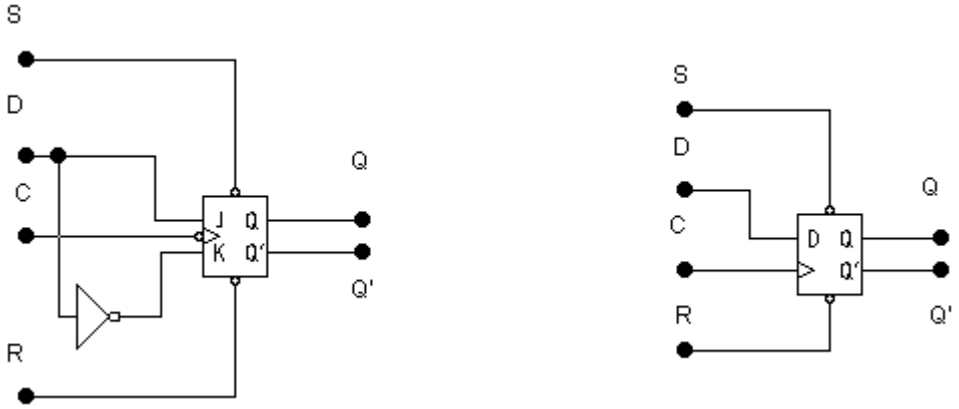


Рис. 5 – Временная диаграмма *D-триггера*

Характеристическое уравнение D-триггера:

$$Q_{t+1} = D_t \qquad (2)$$

Уравнение триггера показывает, что состояние триггера на *t+1* такте равно входному сигналу в момент, предшествующий тактовому перепаду сигнала *C*. Условное обозначение *D-триггера* представлено на рис 6.1. Функциональная схема *D-триггера* может быть получена из схемы *JK-триггера* путем подключения входа *D* к входу *J* через инвертор, рис. 6.2.

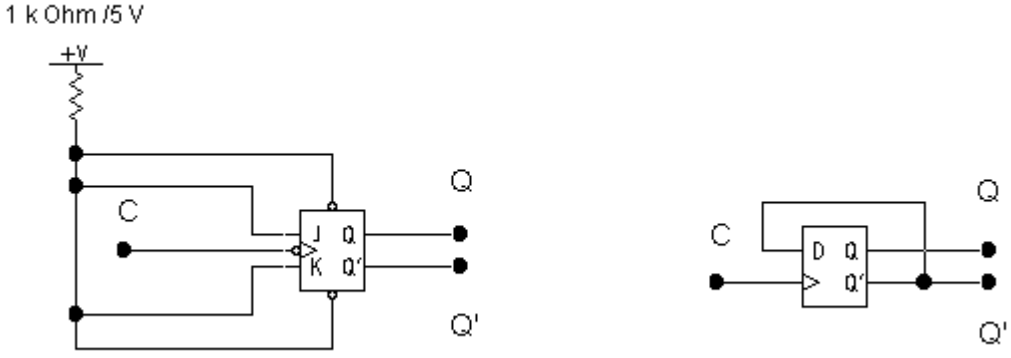


1) Расширенная схема 2) Условное обозначение

Рис. 6 – Схема *D-триггера*

4. Т-триггер (счетный триггер)

На основе *JK-триггеров* и *D-триггеров* можно построить схемы, осуществляющие так называемый счетный режим. Такие схемы называют *Т-триггерами* или счетными триггерами, что связано со способом их функционирования. На рис. 7 представлены схемы организации *Т-триггера* на основе *JK-триггера* (рис. 7.1) и *D-триггера* (рис. 7.2).



1) На базе *JK-триггера* 2) На базе *D-триггера*

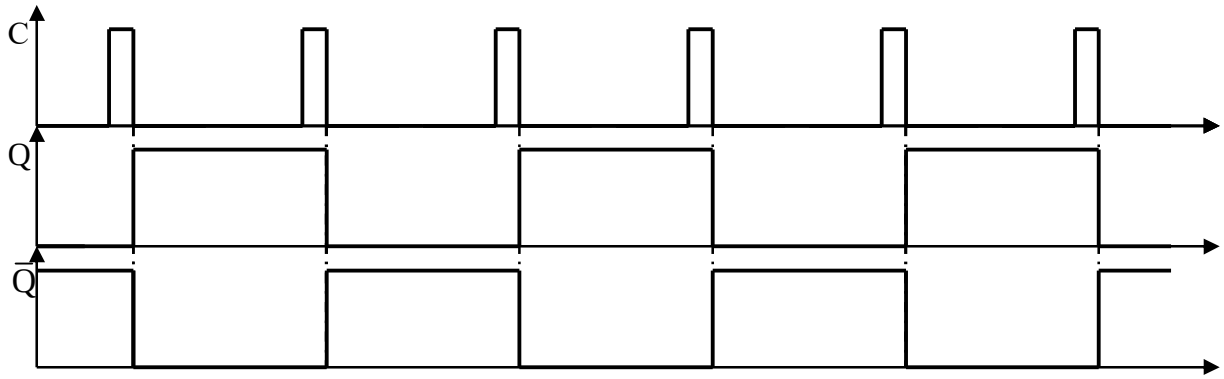
Рис. 7 – Схема *Т-триггера*

Счетный режим иллюстрируется временными диаграммами рис. 8.

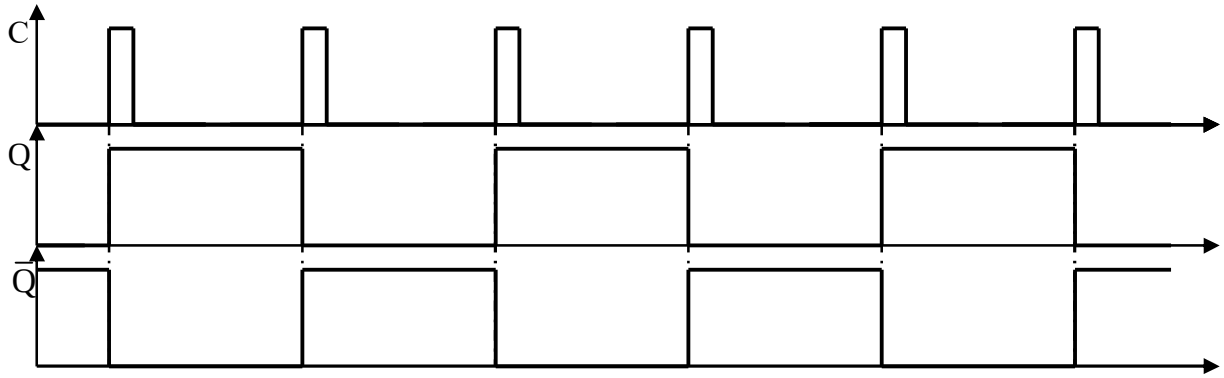
В *JK-триггере* с входами установки логическим нулем счетный режим реализуется путем подачи констант *J=K=1* и *R=S=1* и входного сигнала *T* на вход *C*.

В соответствии с таблицей функционирования при каждом отрицательном перепаде входного сигнала *T* состояние триггера изменяет свое значение на противоположное.

В *D-триггере* счетный режим реализуется при помощи обратной связи (на вход *D* подается сигнал с инверсного выхода). Таким образом, всегда существует неравенство сигнала на входе *D* и сигнала на выходе *Q*: если *Q=1*, *D=0*.



1) на базе *JK-триггера*



2) на базе *D-триггера*

Рис. 8 – Временная диаграмма *T-триггера*

Следовательно, при каждом положительном перепаде сигнала на счетном входе *C* в соответствии с принципом действия *D-триггера* состояние выхода будет изменяться на противоположное.

Таким образом, на каждые два входных тактовых импульса *T-триггер* формирует один период выходного сигнала *Q*. Следовательно, триггер осуществляет деление частоты f_T на его входе на 2:

$$f_Q = \frac{f_T}{2} \qquad (3)$$

Порядок работы

Задание 1. Исследование работы схемы *RS триггера*

а) Исследование RS триггера, составленного из двух элементов «ИЛИ-НЕ»

Создайте схему изображенную на рис. 1.а.

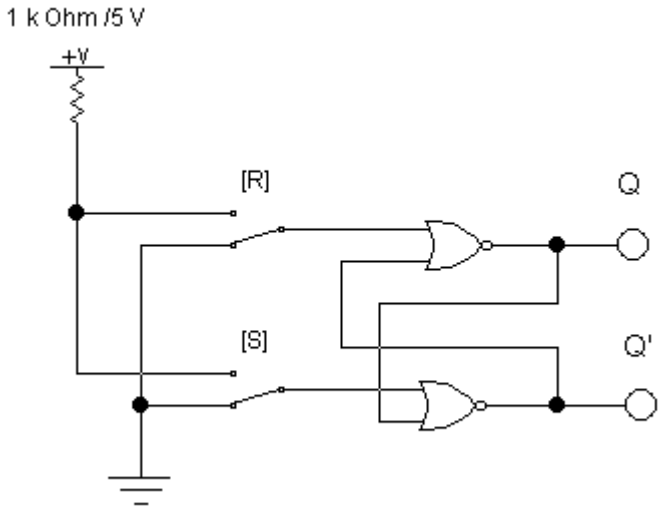


Рис. 1.а – Схема *RS триггера* на базе элементов «ИЛИ-НЕ»

Включите схему. Убедитесь в правильности работы триггера (перехода из одного состояния в другое), проверив следующие утверждения:

при *S=0, R=1* триггер устанавливается в состояние *Q=0*;

при *S=0, R=0* триггер сохраняет свое прежнее состояние *Q=0*;

при *S=1, R=0* триггер устанавливается в состояние *Q=1*;

при *S=0, R=0* триггер сохраняет прежнее состояние *Q=1;*

при *S=1, R=1* на триггер подана запрещенная комбинация.

Заполните таблицу возбуждения данного типа триггера (таблица 1.а).

Таблица 1.а – *RS триггер* на базе элементов «ИЛИ-НЕ»

Q_t	Q_{t+1}	R	S
0		0	0
0		1	0
0		0	1
1		0	0
1		1	0
1		0	1

Примечание: Таблица возбуждения триггера составляется для различных комбинаций *R* и *S* входов. При этом предварительно необходимо добиться начального состояния триггера Q_t помня, что установить триггер в состояние $Q_t=0$ можно подав сигнал *R=1 (S=0),* а установить триггер в состояние $Q_t=1$ можно подав сигнал *S=1 (R=0).*

б) Исследование $\overline{RS}$-триггера, составленного из двух элементов «И-НЕ»
Создайте схему изображенной на рис. 1.б.

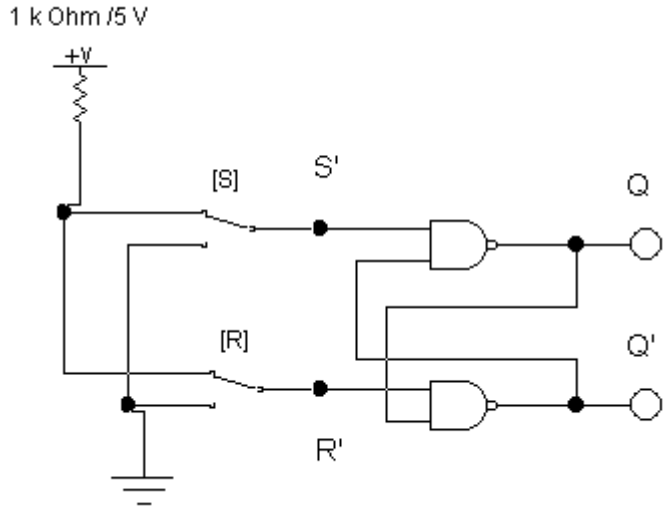


Рис. 1.б – Схема $\overline{RS}$-*триггера* на базе элементов «И-НЕ»

Примечание: на схеме инверсные входы $\overline{RS}$-*триггера* $\overline{R}$ и $\overline{S}$ обозначаются как *R'* и *S'* соответственно. Управляющие клавиши ключей (напомним, что управляющая клавиша задается в закладке «Key» окна свойств ключа, и предназначена для изменения положения ключа) заданны соответственно «R» для входа $\overline{R}$ и «S» для входа $\overline{S}$.
Включите схему. Убедитесь в правильности работы триггера (перехода из одного состояния в другое), проверив следующие утверждения:

при $\overline{S}=1,\ \overline{R}=0$ триггер устанавливается в состояние *Q=0*;
при $\overline{S}=1,\ \overline{R}=1$ триггер сохраняет свое прежнее состояние *Q=0*;
при $\overline{S}=0,\ \overline{R}=1$ триггер устанавливается в состояние *Q=1*;
при $\overline{S}=1,\ \overline{R}=1$ триггер сохраняет свое прежнее состояние *Q=1;*
при $\overline{S}=0,\ \overline{R}=0$ на триггер подана запрещенная комбинация.

Заполните таблицу возбуждения данного типа триггера (таблица 1.а).

Таблица 1.б – $\overline{RS}$-*триггер* на базе элементов «И-НЕ»

Q_t	Q_{t+1}	$\overline{R}=R'$	$\overline{S}=S'$
0		1	1
0		1	0
0		0	1
1		1	0
1		1	0
1		0	1

Примечание: Таблица возбуждения триггера составляется для различных комбинаций $\overline{R}$ и $\overline{S}$ входов. При этом предварительно необходимо добиться начального состояния триггера Q_t помня, что установить триггер в состояние

$Q_t=1$ можно подав сигнал $\overline{S}=0$ $(\overline{R}=1)$, а установить триггер в состояние $Q_t=0$ можно подав сигнал $\overline{R}=0$ $(\overline{S}=1)$.

Задание 2. Исследование работы *JK-триггера*

а) Составление функции возбуждения JK-триггера

Соберите схему, изображенную на рис. 2.а.

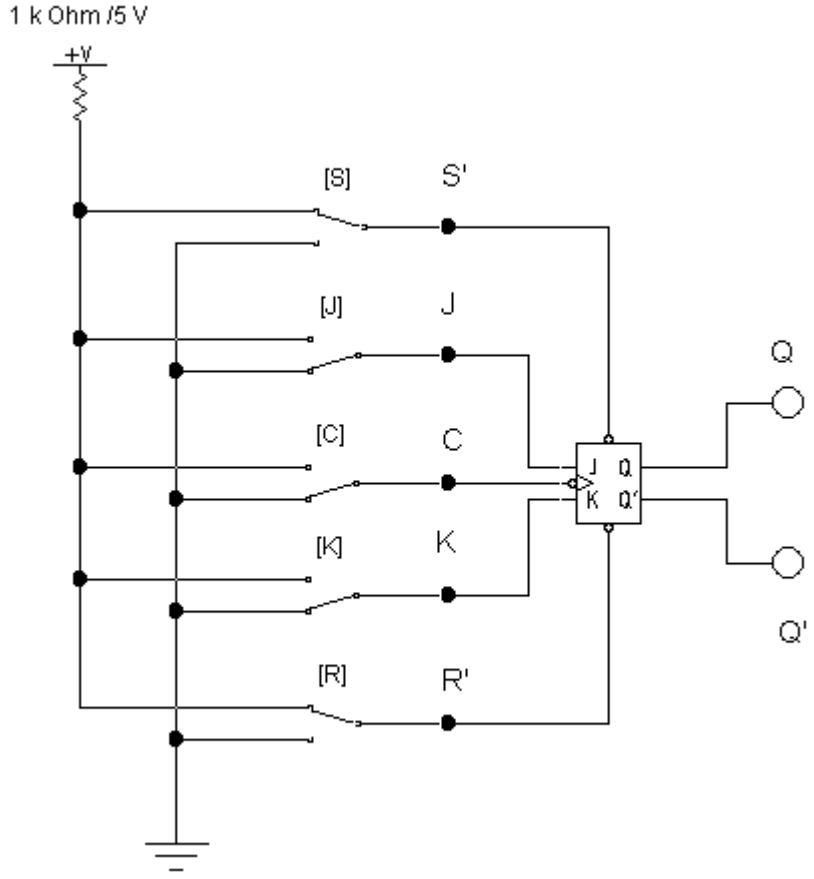


Рис. 2.а – Схема *JK-триггера*

Отметим, что на схеме используется упрощенная микросхема *JK-триггера*, в которой установочные входы $\overline{R}$ и $\overline{S}$ инверсные (на микросхеме такие входы имеют кружок на линии входящей в корпус).

Включите схему. Убедитесь в правильности работы установочных входов триггера, проверив следующие утверждения:

при $\overline{S}=1$, $\overline{R}=0$ триггер устанавливается в состояние $Q=0$, независимо от входов *J, K, C*;

при $\overline{S}=0$, $\overline{R}=1$ триггер устанавливается в состояние $Q=1$, независимо от входов *J, K, C*.

Таким образом, для установки триггера в требуемое исходное состояние необходимо подать на его вход одну из перечисленных комбинаций, затем установить $\overline{S}=1, \overline{R}=1$.

Установив триггер в необходимое исходное состояние, заполните таблицу возбуждения *JK-триггера* (таблица 2.а).

Таблица 2.а – *JK-триггер*

Q_t	Q_{t+1}	J	K
0		0	0
0		0	1
0		1	0
0		1	1
1		0	0
1		0	1
1		1	0
1		1	1

Примечание: переход триггера в новое состояние происходит по заднему фронту импульса *C*. Таким образом, для получения нового состояния Q_{t+1} необходимо установить начальное состояние триггера, затем необходимую комбинацию входов *J* и *K*, и произвести переключение ключа *C*.

б) Составление временных диаграмм JK-триггера

По таблице возбуждения *JK-триггера* (таблица 2.а) составьте временную диаграмму работы триггера. Оси графика необходимо выбрать аналогично рис. 4 (краткой теории). При этом на оси «С» также откладываются последовательно импульсы разрешения. Значения, откладываемые на других осях временной диаграммы необходимо выбирать построчно из таблицы 2.а, по следующим правилам:

до появления импульса *C* значения *J* и *K* должны устанавливаться согласно каждой выбранной строке таблице возбуждения;

значение по оси Q выбирается равным Q_t (согласно выбранной строке) до каждого момента исчезновения импульса *C* (до появления так же называемого – заднего фронта);

после исчезновения импульса *C* значение оси Q выбирается равным Q_{t+1} (согласно выбранной строке) до момента установки новых значений *J* и *K* (перехода к анализу новой строки таблицы возбуждения);

переход к новым значениям *J* и *K* производится после исчезновения исследуемого импульса *C*, но до появления нового (и далее по каждой строке аналогично).

При этом полученные временные диаграммы будут отличными от приведенных на рис. 4, так как в соответствии с таблицей 2.а каждый раз триггер необходимо сбрасывать в состояние Q_t, до анализа исследуемой строки.

Задание 3. Исследование работы *D-триггера*

а) Исследование D триггера составленного на базе JK-триггера

Создайте схему, изображенную на рис. 3.а.

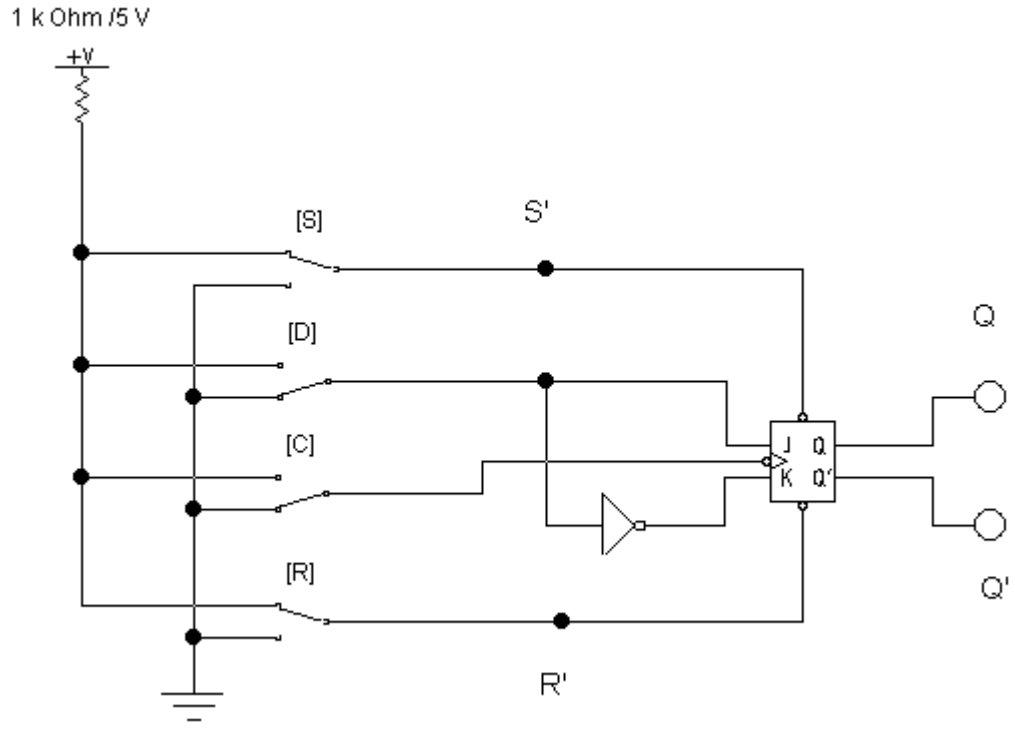


Рис. 3.а – Схема *D-триггера* на базе *JK-триггера*

Включите схему. Убедитесь, в правильности функционирования установочных входов:

при $\overline{S}=1$, $\overline{R}=0$ триггер устанавливается в состояние $Q=0$, независимо от входов *D, C*;

при $\overline{S}=0$, $\overline{R}=1$ триггер устанавливается в состояние $Q=1$, независимо от входов *D, C*.

Заполните таблицу переходов (таблица 3.а), установив предварительно установочные входы в разрешающее состояние $\overline{S}=1$ и $\overline{R}=1$.

Таблица 3.а – Переходы *D-триггера*

D	**Q_{t+1}**
0	
1	

б) Получение функций возбуждения D триггера

Создайте схему изображенную на рис. 3.б.

Проверите правильность работы установочных входов $\overline{R}$ и $\overline{S}$. Убедитесь в аналогичности схем 3.а и 3.б с помощью полученной в предыдущем пункте таблицы переходов.

Заполните таблицу возбуждения *D-триггера* (таблица 3.б), предварительно устанавливая необходимое состояние Q_t сигналами $\overline{R}$ и $\overline{S}$ (после установки необходимо перевести данные входы в разрешающие положение $\overline{R}=1$, $\overline{S}=1$).

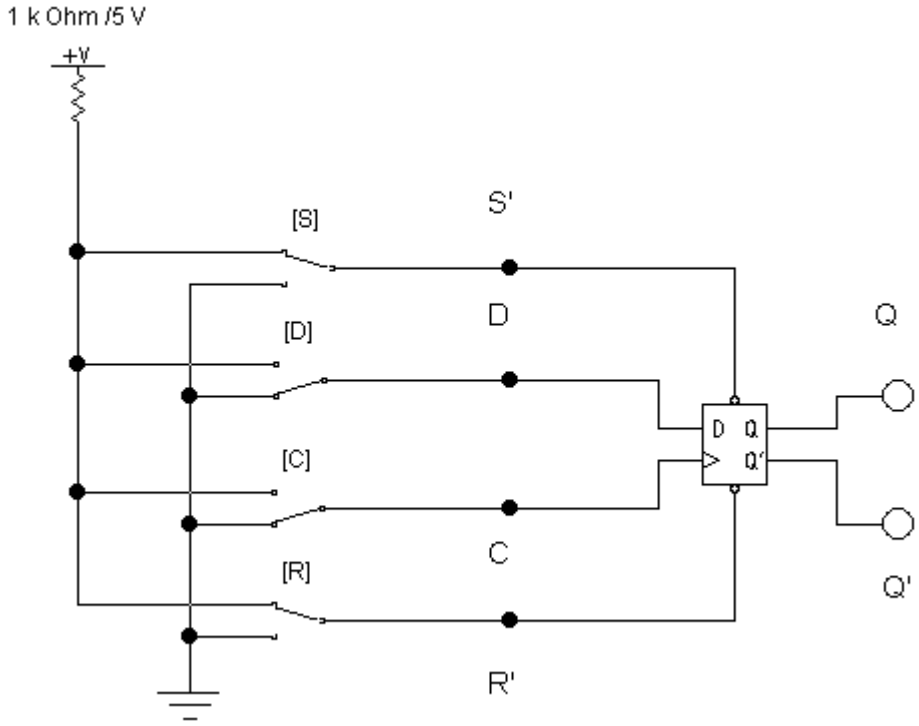


Рис. 3.б – Схема *D-триггера*

Таблица 3.б – Функции возбуждения *D-триггера*

Q_t	D	Q_{t+1}
0	0	
0	1	
1	0	
1	1	

Примечание: необходимо помнить, что переключение триггера в новое состояние происходит по переднему фронту импульса *C*.

в) Составление временной диаграммы D триггера

По таблице возбуждения *D-триггера* (таблица 3.б) составьте временную диаграмму работы триггера. Оси графика необходимо выбрать аналогично рис. 5 (краткой теории). При этом на оси «С» так же откладываются последовательно импульсы разрешения. Значения, откладываемые на других осях временной диаграммы необходимо выбирать из таблицы 3.б, аналогично пункту 2.б.

Полученные временные диаграммы будут отличны от диаграмм рис. 5, так как в соответствии с таблицей 3.б каждый раз триггер необходимо сбрасывать в значение Q_t.

Задание 4. Исследование работы *T триггера*

а) Исследование T-триггера на основе JK-триггера

Соберите *T-триггер* на основе *JK-триггера* согласно схеме, изображенной на рис. 4.а.

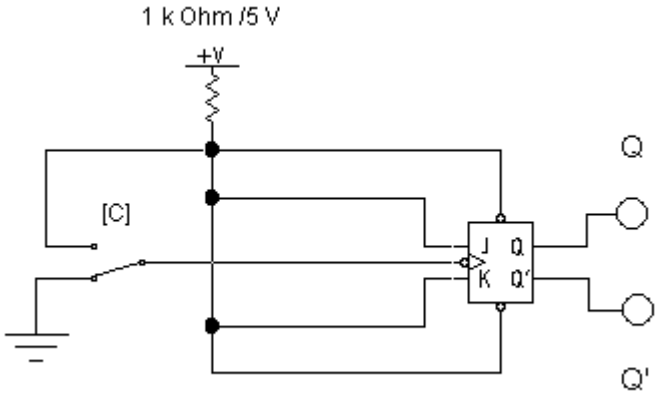


Рис. 4.а – Схема *Т-триггера* на основе *JK-триггера*

Включите схему. Изменяя состояние входа *С* соответствующим ключом, постройте временную диаграмму *Т-триггера*. Сравните полученный результат с рис. 7.а.

б) Исследование Т-триггера на основе D-триггера

Соберите *Т-триггер* на основе *D-триггера* согласно схеме, изображенной на рис. 4.б.

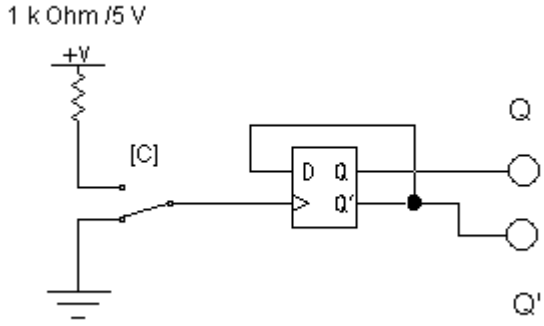


Рис. 4.б – Схема *Т-триггера* на основе *D-триггера*

Включите схему. Изменяя состояние входа *С* соответствующим ключом, постройте временную диаграмму *Т-триггера*.

Сравните полученный результат с рис. 7.2.

Контрольные вопросы

1. Дайте определение триггера.
2. Чем отличается *RS-триггер* с обычными входами от *RS-триггера* с инверсными входами?
3. Что называется неопределенным состоянием триггера?
4. Приведите примеры неопределенного состояния различных триггерных схем.
5. Чем отличается таблица переходов от таблицы возбуждения?
6. Приведите характеристическое уравнение *RS-триггера*.
7. Чем отличается *JK-триггер* от *RS-триггера*?
8. Приведите характеристическое уравнение *JK-триггера*.
9. Зачем нужны установочные входы в триггерах?
10. Каков приоритет установочных входов в триггерах по сравнению с информационными входами?
11. Охарактеризуйте основное применение *D-триггера*?
12. Приведите характеристическое уравнение *D-триггера*.
13. Чем отличается *D-триггер* от *T-триггера*?
14. Чему равна частота следования импульсов в *T-триггере*?

Упражнения

Упражнение 1. Построение временных диаграмм

По заданной таблице истинности некоторой логической функции постройте временную диаграмму.

Считайте, что переключение в новое состояние схемы должно происходить по отрицательному фронту импульса *C*.

Варианты таблицы истинности приведены в таблице 1.у.

Таблица 1.у – Варианты задания логической функции

Значения логических переменных (для всех вариантов)			**Варианты задания**									
			1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
X	**Y**	**Z**	**Значения логической функции (для каждого варианта)**									
0	0	0	0	0	1	1	0	1	0	0	1	0
0	0	1	0	1	0	0	0	0	1	0	1	0
0	1	0	0	1	1	0	1	1	0	0	0	0
0	1	1	1	0	1	1	0	0	0	1	0	1
1	0	0	0	1	0	1	0	1	1	1	1	1
1	0	1	1	1	0	1	0	1	1	1	1	1
1	1	0	0	0	1	1	0	0	0	1	0	1
1	1	1	0	0	1	0	0	1	0	0	0	0

ЛАБОРАТОРНАЯ РАБОТА № 4

Изучение сумматоров, полусумматоров, регистров и счетчиков

Цель работы:

1. Исследование сумматоров и полусумматоров.
2. Изучение структуры и исследование работы суммирующих и вычитающих счетчиков, счетчиков с измененным коэффициентом пересчета.
3. Изучение регистров.

Приборы и элементы:

Генератор слов (панель «Instruments/Word Generator»)
Логические пробники (панель «Indicators/Red probe»)
Источник напряжения + 5 В (панель «Basic/Pull-Up Resistor»)
Земля (панель «Sources/Ground»)
Двухпозиционные переключатели (панель «Basic/Switch»)
Двухвходовые элементы И, И-НЕ, ИЛИ, ИЛИ-НЕ (панель «Logic Gates/2-Input AND, NAND, OR, NOR Gates»)
Сумматор (панель «Digital/Half-Adder»)
D-триггер (панель «Digital/»)
Декодер (панель «Indicators/Decoded 7 segment display)

Краткие теоретические сведения

Сумматоры и полусумматоры

Широкое применение в цифровой технике находят элементы, выполняющие различные арифметические действия. Операция суммирования – базовая арифметическая операция в двоичной алгебре. Поэтому для дальнейшего изучения цифровой техники необходимо исследовать способы получения сумматоров.

1. Сумматоры по модулю два

Построение двоичных сумматоров обычно начинается с сумматора по модулю 2. Ниже представлена таблица истинности этого сумматора.

Таблица 1 – Таблица истинности сумматора по модулю два

Входы		**Выход**
X	**Y**	**Out**
0	0	0
0	1	1
1	0	1
1	1	0

Из таблицы 1 видно, что логическая функция, выражающая принцип работы сумматора по модулю два, имеет вид:

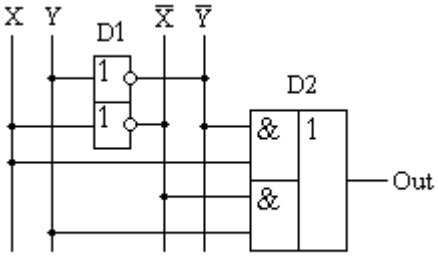
$$F = X \oplus Y = X\overline{Y} \vee \overline{X}Y \qquad (1)$$

и представляет собой описанную ранее (формула 7, Лабораторная работа №1) функцию «исключающего ИЛИ».

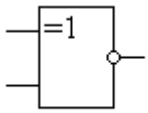
На рис. 1.1 представлена схемная реализация сумматора по модулю два, составленная по таблице истинности 1. На рис. 1.2 приведено условное обозначение этой же схемы в виде одного элемента – «Исключающего ИЛИ».

Однако представленная схема имеет недостаток, который можно увидеть в таблице 1 – не учет переноса при сложении $X = 1$ и $Y = 1$.

Сумматор по модулю 2 выполняет суммирование без учёта переноса. В обычном двоичном сумматоре требуется учитывать перенос, поэтому требуются схемы, позволяющие формировать перенос в следующий двоичный разряд.



1) Расширенная схема



2) «Исключающее ИЛИ»

Рис. 1 – Сумматор по модулю два

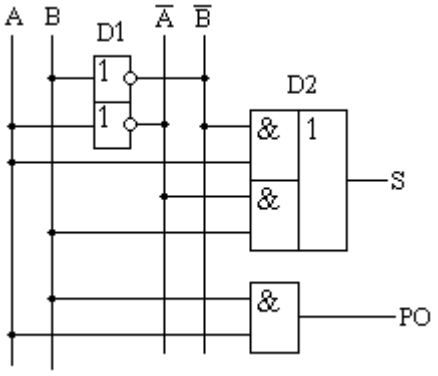
2. Полусумматоры

Таблица истинности полусумматора приведена ниже.

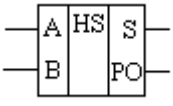
Таблица 2 – Таблица истинности полусумматора

Входы		**Выходы**	
A	**B**	**S**	**PO**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

На рис. 2 представлена соответствующая схемная реализация полусумматора на базе логических элементов (рис. 2.1) и в виде одного устройства (рис. 2.2).



1) Расширенная схема



2) Условное обозначение

Рис. 2 – Схема полусумматора

Схема полусумматора формирует перенос в следующий разряд (*PO*), но не может учитывать перенос из предыдущего разряда, поэтому она и называется полусумматором. Для реализации же полного суммирования (пусть пока и одноразрядного) необходимо помимо формирования переноса в следующий разряд учитывать еще и перенос из предыдущего разряда (это нужно для формирования многоразрядных сумматоров).

3. Одноразрядные сумматоры

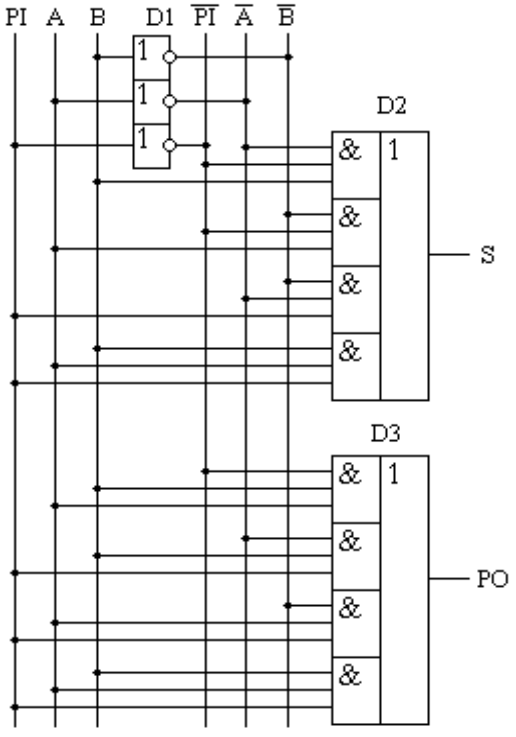
Таблица истинности полного двоичного одноразрядного сумматора приведена в таблице 3.

Таблица 3 – Таблица истинности одноразрядного сумматора

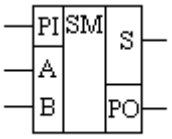
Входы			**Выходы**	
PI	**A**	**B**	**S**	**PO**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Здесь, помимо формирования переноса в следующий разряд (*PO*) учитывается еще и перенос из предыдущего разряда (*PI*).

На рис. 3 представлена соответствующая схемная реализация сумматора на базе логических элементов (рис. 3.1) и в виде одного устройства (рис. 3.2).



1) Расширенная схема

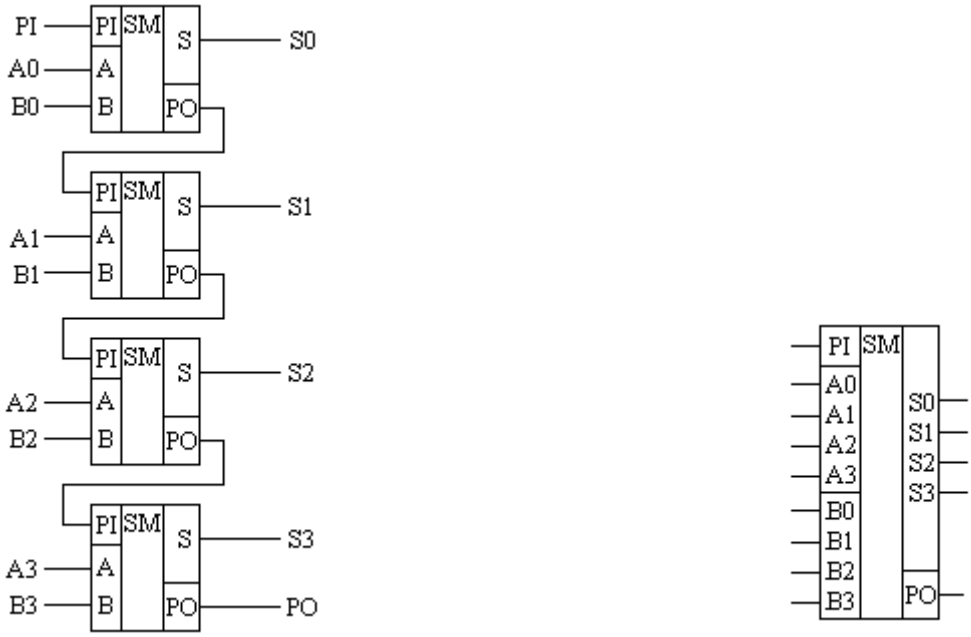


2) Условное обозначение

Рис. 3 – Схема двоичного одноразрядного сумматора

4. Многоразрядные сумматоры

Для того чтобы получить многоразрядный сумматор, необходимо соединить входы и выходы переносов соответствующих двоичных разрядов. Схема соединения для трехразрядного сумматора приведена на рисунке 4.1. Здесь же приведено условное обозначение данного сумматора, применяемого на схемах.



1) Расширенная схема 2) Условное обозначение

Рис. 4 – Схема полного двоичного трехразрядного сумматора

В схеме на рис. 4 рассматриваются только принципы работы двоичных сумматоров. В реальных схемах для увеличения скорости работы применяется отдельная схема формирования переносов для каждого двоичного разряда.

Таблицу истинности для такой схемы легко получить из принципов суммирования двоичных чисел, а затем применить хорошо известные нам принципы построения схемы по произвольной таблице истинности.

Счетчики

Счетчик – устройство для подсчета числа входных импульсов. Число, представляющее состояние его выходов с приходом нового импульса изменяется на единицу. Счетчик можно реализовать на нескольких триггерах. В суммирующих счетчиках каждый входной импульс увеличивает число на его выходе на единицу, в вычитающих счетчиках каждый входной импульс уменьшает это число на единицу. Наиболее простые счетчики – двоичные. На рис. 5 представлен суммирующий двоичный счетчик. Диаграммы работы двоичного суммирующего счетчика представлена на рис. 6.

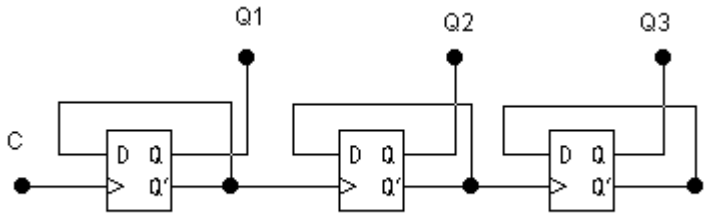


Рис. 5 – Суммирующий двоичный счетчик

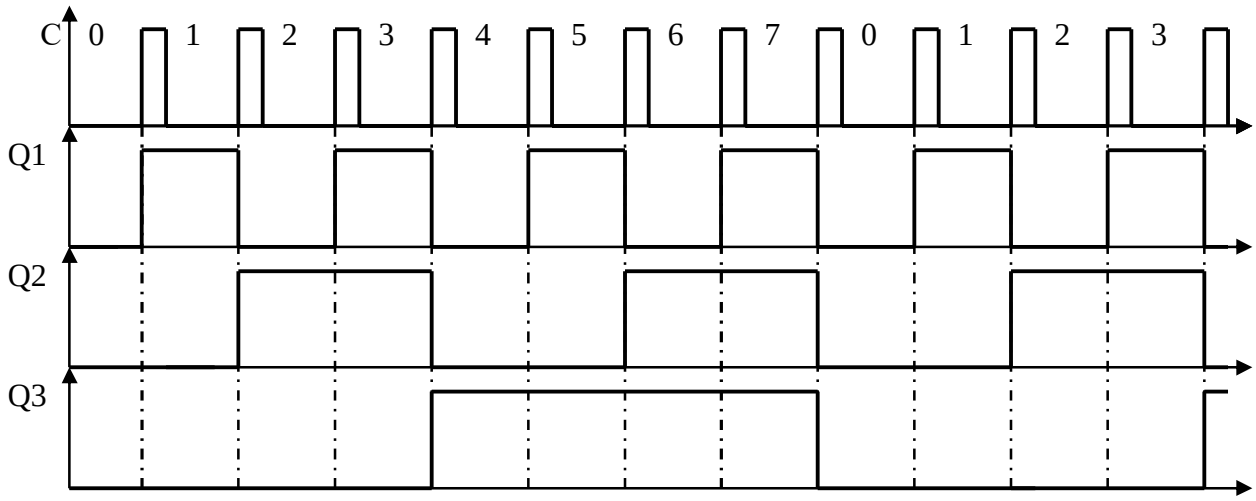


Рис. 6 – Диаграмма работы двоичного суммирующего счетчика

1. Изменение направления счета

Как уже говорилось ранее, счетчики можно реализовать на триггерах. При этом триггеры соединяют последовательно. Выход каждого триггера непосредственно действует на тактовый вход следующего. Для того, чтобы реализовать суммирующий счетчик, необходимо счетный вход очередного триггера подключать к инверсному выходу предыдущего. Для того чтобы изменить направление счета (реализовать вычитающий счетчик), можно предложить следующие способы:

а) Считывать выходные сигналы счетчика не с прямых, а с инверсных выходов триггеров. Число, образуемое состоянием инверсных выходов триггеров счетчика, связано с числом, образованным состоянием прямых выходов триггеров следующим соотношением:

$$N_{пр} = 2^n - N_{инв} - 1 \qquad (2)$$

где n – разрядность выхода$_1$ счетчика. В таблице 4 приведен пример связи числа на прямых выходах с числом на инверсных выходах триггеров счетчика.

Таблица 4 – Связь между прямыми и инверсными выходами счетчика

Состояние прямых выходов			**Число**	**Состояние инверсных выходов**			**Число**
Q3	Q2	Q1	N	$\overline{Q3}$	$\overline{Q2}$	$\overline{Q1}$	N
0	0	0	0	1	1	1	7
0	0	1	1	1	1	0	6
0	1	0	2	1	0	1	5

б) Изменить структуру связей в счетчике: подавать на счетный вход следующего триггера сигнал не с инверсного, а с прямого выхода предыдущего, как показано на рис. 7. Временная диаграмма для такого способа реализации счетчика приведена на рис. 8. В этом случае изменяется последовательность переключения триггеров.

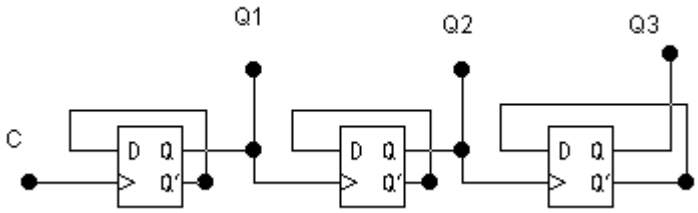


Рис. 7 – Вычитающий счетчик

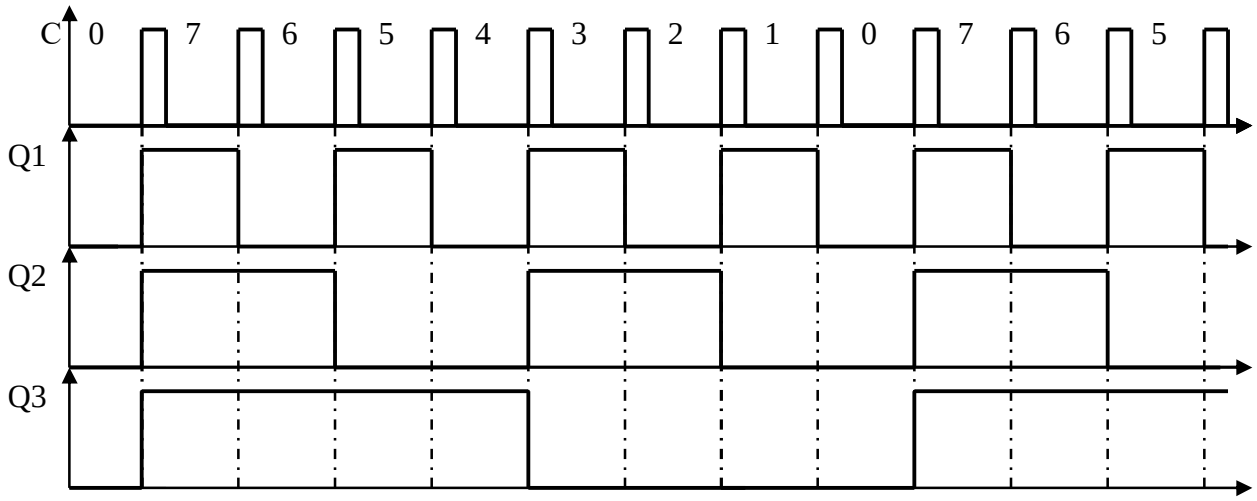


Рис. 8 – Диаграмма работы вычитающего счетчика

2. Изменение коэффициента пересчета

Счетчики характеризуются числом состояний в течение одного периода (цикла). Для схем на рис. 5 и рис. 7 цикл содержит $N=2^3=8$ состояний (от «000» до «111»). Часто число состояний называют *коэффициентом пересчета* $K_{сч}$, который равен отношению числа импульсов N_c на входе к числу импульсов N_{Qcm} на выходе старшего разряда за период:

$$K_{сч} = \frac{N_c}{N_{Qcm}} \qquad (3)$$

Если на вход счетчика подавать периодическую последовательность импульсов с частотой f_c, то частота f_Q на выходе старшего, разряда счетчика будет меньше в $K_{сч}$ раз:

$$K_{сч} = \frac{F_c}{F_Q} \qquad (4)$$

Поэтому счетчики также называют делителями частоты, а величину $K_{сч}$ – коэффициентом деления. Для увеличения величины $K_{сч}$ приходится увеличивать число триггеров в цепочке. Каждый дополнительный триггер удваивает число состояний счетчика и число $K_{сч}$. Для уменьшения коэффициента $K_{сч}$ можно в

качестве выхода счетчика рассматривать выходы триггеров промежуточных каскадов.

Например, для счетчика на трех триггерах $K_{сч}=8$, если взять выход 2-го триггера, то $K_{сч}=4$. При этом $K_{сч}$ является целой степенью числа 2: *2, 4, 8, 16* и т. д.

Можно реализовать счетчик, для которого $K_{сч}$ – любое целое число. Например, для счетчика на трех триггерах можно сделать $K_{сч}$ от «2» до «7», но при этом один или два триггера могут оказаться лишними. При использовании всех трех триггеров можно получить $K_{сч}=5...7$:

$$2^2 < K_{сч} < 2^3 \qquad (5)$$

Счетчик с $K_{сч}=5$ должен иметь 5 состояний, которые в простейшем случае образуют последовательность: {О, 1, 2, 3, 4}. Циклическое повторение этой последовательности означает, что коэффициент деления счетчика равен «5».

Для построения суммирующего счетчика с $K_{сч}=5$ надо, чтобы после формирования последнего числа из последовательности *{О, 1, 2, 3, 4}* счетчик переходил не к числу «5», а к числу «0». В двоичном коде это означает, что от числа «100» нужно перейти к числу «000», а не «101». Изменение естественного порядка счета возможно при введении дополнительных связей между триггерами счетчика. Можно воспользоваться следующим способом: как только счетчик попадает в нерабочее состояние (в данном случае «101»), этот факт должен быть опознан и повлечь последующую выработку сигнала, который перевел бы счетчик в состояние «000». Рассмотрим этот способ более детально.

Факт попадания счетчика в нерабочее состояние описывается логическим уравнением:

$$F = (101) \vee (110) \vee (111) = Q_3 \cdot \overline{Q}_2 \cdot Q_1 \vee Q_3 \cdot Q_2 \cdot \overline{Q}_1 \vee Q_3 \cdot Q_2 \cdot Q_1 = \\ = Q_3 \cdot Q_1 \vee Q_3 \cdot Q_2 \qquad (6)$$

Состояния «110» и «111» также являются нерабочими и поэтому учтены при составлении уравнения. Если на выходе эквивалентной логической схемы *F=0*, значит, счетчик находится в одном из рабочих состояний: $0 \vee 1 \vee 2 \vee 3 \vee 4$. Как только он попадает в одно из нерабочих состояний $5 \vee 6 \vee 7$, формируется сигнал *F=1*. Появление сигнала *F=1* должно переводить счетчик в начальное состояние «000», следовательно, этот сигнал нужно использовать для воздействия на установочные входы триггеров счетчика, которые осуществляли бы сброс счетчика в состояние $Q_1 = Q_2 = Q_3$. При реализации счетчика на триггерах с входами установки логическим нулем для сброса триггеров требуется подать на входы сброса сигнал *R=0*. Для обнаружения факта попадания в нерабочее состояние используем схему, реализующую функцию *F* и выполненную на элементах И-НЕ. Для этого преобразуем выражение для функции:

$$F = Q_3 \cdot Q_1 \vee Q_3 \cdot Q_2 = Q_3 \cdot \overline{(\overline{Q}_1 \cdot \overline{Q}_2)} \qquad (7)$$

Соответствующая схемная реализация приведена на рис. 9.

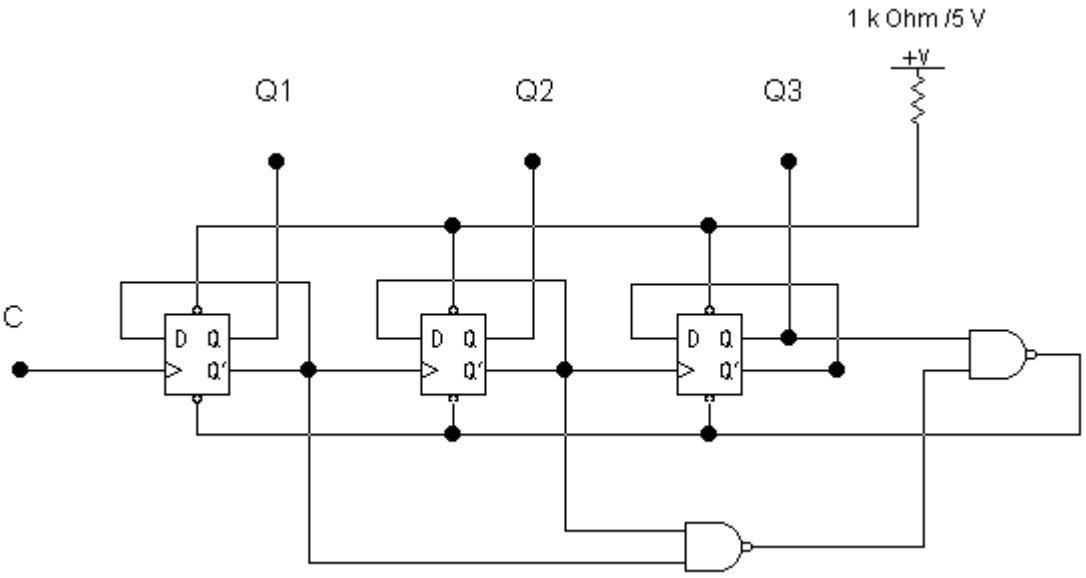


Рис. 9 – Счетчик с измененным коэффициентом пересчета

Счетчик будет работать следующим образом: при счете от «0» до «4» все происходит как в обычном суммирующем счетчике с $K_{сч}=8$. Установочные сигналы равны «1» и естественному порядку счета не препятствуют. Счет происходит по положительному фронту импульса на счетном входе *C*. В тот момент, когда счетчик находится в состоянии «4» («100»), следующий тактовый импульс сначала переводит счетчик в состояние «5» («101»), что немедленно (задолго до прихода следующего тактового импульса) приводит к формированию сигнала сброса, который поступает на установочные входы сброса триггеров. В результате счетчик сбрасывается в «0» и ждет прихода следующего тактового импульса на счетный вход. Один цикл счета закончился, счетчик готов к началу следующего цикла.

Применяя такие схемы с обратной связью для сброса счетчика, нужно иметь в виду, что операция сброса занимает конечное время, поэтому непосредственно перед сбросом счетчика в «0» на выходе первого триггера появляются кратковременные импульсы, или «иголки». Это не имеет значения при подключении счетчика напрямую к индикатору, но при использовании этого выхода счетчика в качестве источника тактовых импульсов могут возникнуть определенные проблемы. Схема, в которой это явление устранено, приведена на рис. 10.

Важным отличием является то, что схема обнаруживает не факт попадания в нерабочее состояние «101», а факт попадания в состояние «100» и в следующем такте вырабатывает сигнал сброса.

Регистры

Регистром называется последовательное или параллельное соединение триггеров. Регистры обычно строятся на основе *D-триггеров*. При этом для построения регистров могут использоваться как универсальные *D-триггеров*, так и триггеры-защелки.

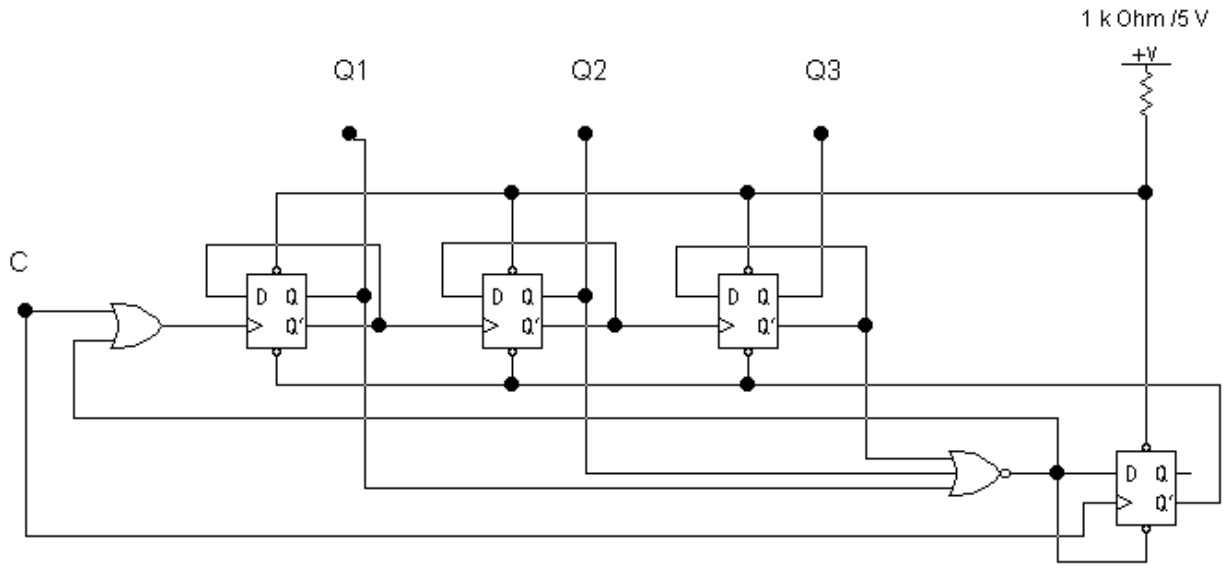
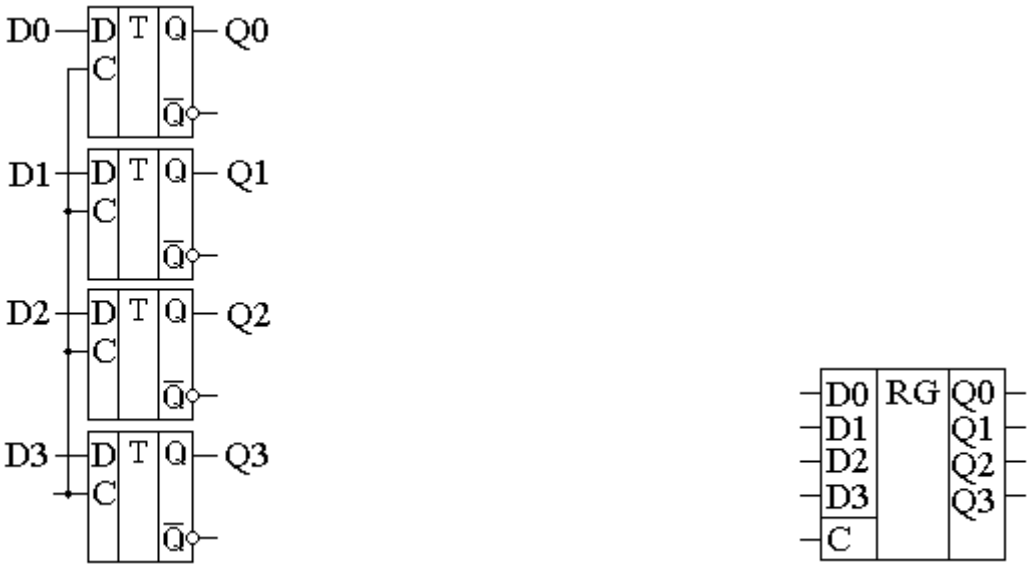


Рис. 10 – Счетчик с измененным коэффициентом пересчета

1. Параллельный регистр

Параллельный регистр служит для запоминания многоразрядного двоичного слова. При использовании для построения параллельного регистра *триггеров-защелок* регистр называется *регистр-защелка*. Параллельный регистр служит для запоминания многоразрядного двоичного слова. Количество триггеров, входящее в состав параллельного регистра определяет его разрядность. При записи информации в параллельный регистр все биты (двоичные разряды) записываются одновременно.

Схема четырёхразрядного параллельного регистра приведена на рисунке 11.1, а его обозначение на принципиальных схемах – на рисунке 11.2.

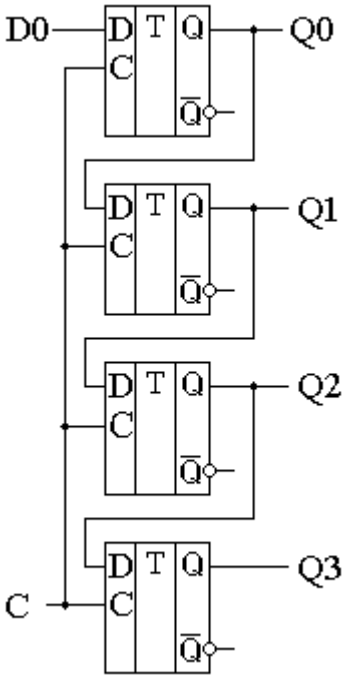


1) Расширенная схема 2) Условное обозначение

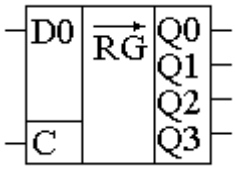
Рис. 11 – Схема параллельного регистра

2. Последовательный регистр

Последовательный регистр (регистр сдвига) обычно служит для преобразования последовательного кода в параллельный и наоборот. Схема регистра, осуществляющего преобразование последовательного кода в параллельный, приведена на рисунке 12.1, а его изображение на принципиальных схемах – на рисунке 12.2.



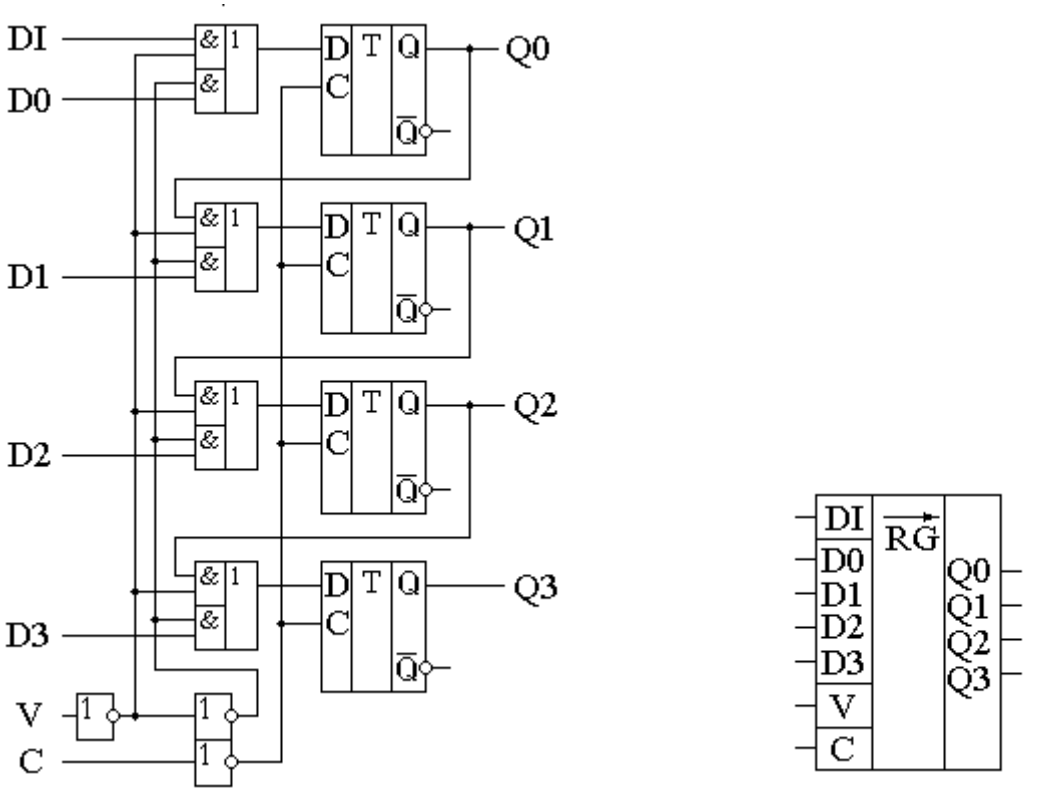
1) Расширенная схема



2) Условное обозначение

Рис. 12 – Схема последовательного регистра

Регистры сдвига выполняют обычно как универсальные *последовательно-параллельные микросхемы*. Переключение регистра из параллельного режима в последовательный и наоборот осуществляется при помощи мультиплексора. Схема такого регистра приведена на рисунке 13.1, а его изображение на принципиальных схемах – на рисунке 13.2.



1) Расширенная схема 2) Условное обозначение

Рис. 13 – Схема универсального регистра

Порядок работы

Задание 1. Изучение полусумматоров и сумматоров

а) Изучение работы полусумматора

Соберите схему, приведенную на рис. 1.а.

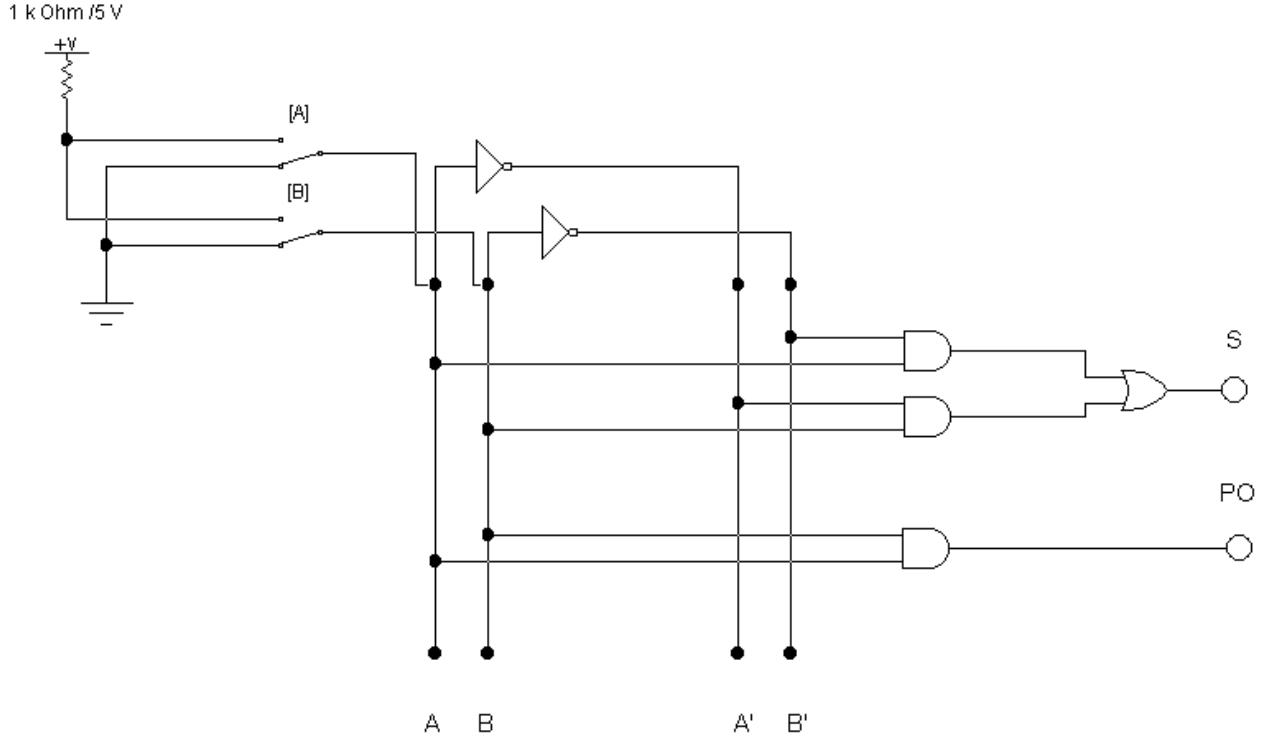


Рис. 1.а – Схема двоичного полусумматора

Задавая различные комбинации логических уровней, на входах полусумматора *A* и *B*, заполните таблицу истинности 1.а.

Таблица 1.а. – Таблица истинности полусумматора

Входы		**Выходы**	
A	**B**	**S**	**PO**
0	0		
0	1		
1	0		
1	1		

Сравните полученную таблицу с таблицей 1 краткой теории, сделайте выводы.

б) Изучение работы одноразрядного сумматора

Соберите схему, приведенную на рис. 1.б.

Задавая различные комбинации логических уровней на входах сумматора *A* и *B*, а так же уровень сигнала переноса из другого разряда *PI* заполните таблицу истинности 1.б.

Сравните полученную таблицу с таблицей 2 краткой теории, сделайте выводы.

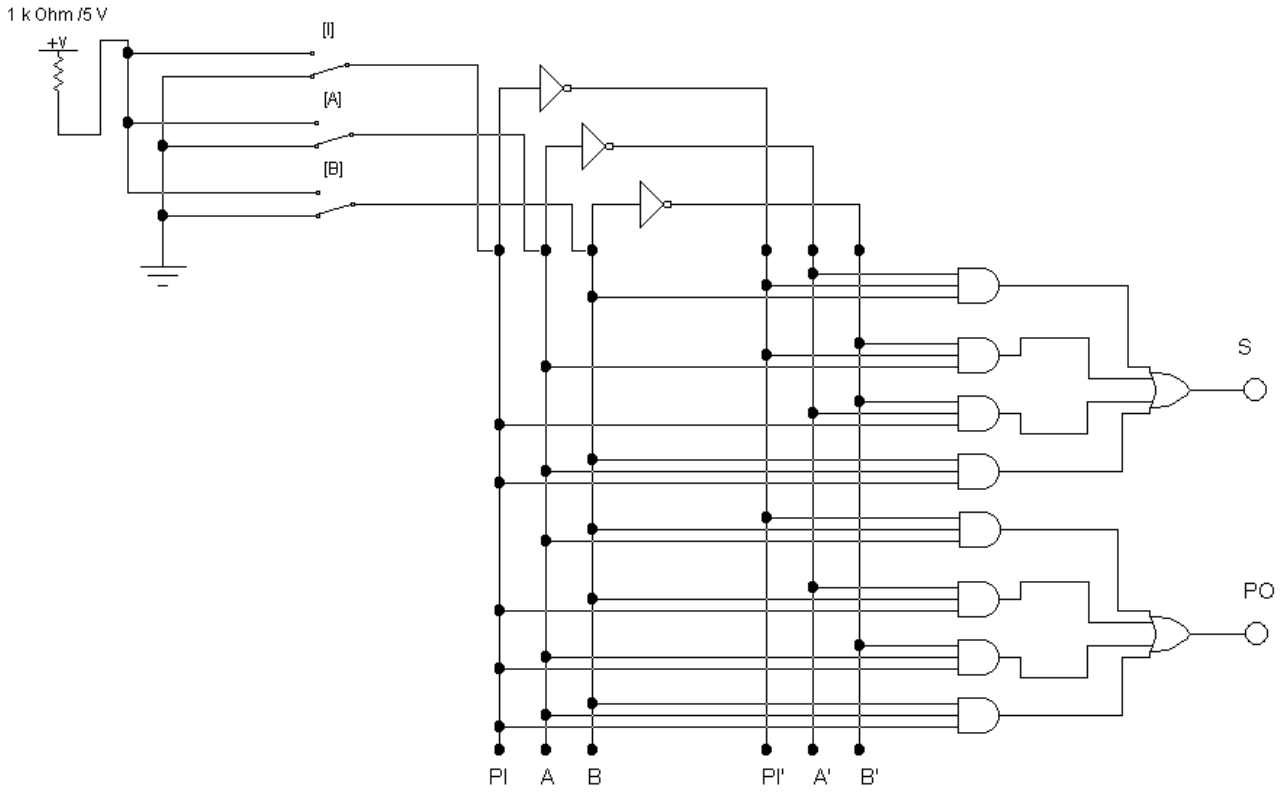


Рис. 1.б – Схема одноразрядного двоичного сумматора

Таблица 1.б. – Таблица истинности одноразрядного сумматора

Входы			**Выходы**	
PI	**A**	**B**	**S**	**PO**
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

в) Изучение работы двоичного трехразрядного сумматора

Соберите схему, аналогичную рис. 1.в.

Здесь в качестве сумматоров используются специальные блоки EWB «Half-Adder» имеющие аналогично условному обозначению рис. 3.2 два входа *A* и *B*, выход сума «S» (в нашем случае на рис. 1.в – «Σ») и сигнал переноса в следующий разряд «PO» (в нашем случае на рис. 1.в – «C_0»). В отличие от сумматора, приведенного на рис. 3.2 у данного сумматора нет учета переноса, как из предыдущего разряда, так и в следующий. Для учета переноса из предыдущих разрядов в схеме рис. 1.в используются дополнительные сумматоры и логические элементы.

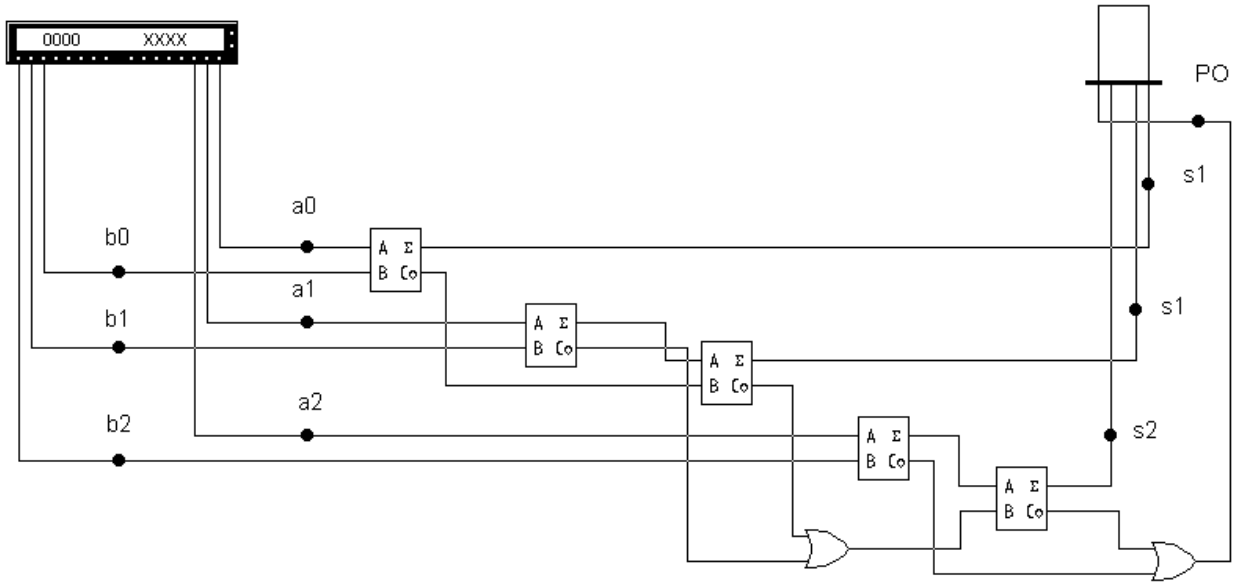


Рис. 1.в – Схема трехразрядного двоичного сумматора

Для проверки правильности функционирования данной схемы необходимо предварительно запрограммировать генератор слов так, что бы на его используемых выходах формировались все возможные комбинации суммируемых двоичных числе «a2 a1 a0» и «b2 b1 b0». Ниже приведена таблица, поясняющая принцип программирования генератора слов (таблица 1.в).

Таблица 1.в – Программирование генератора слов

Шестнадцатеричный код	**Двоичный код**
0000	0000 0000
0001	0000 0001
0002	0000 0010
0003	0000 0011
0004	0000 0100
0005	0000 0101
0006	0000 0110
0007	0000 0111
2000	0010 0000
2001	0010 0001
2002	0010 0010
2003	0010 0011
2004	0010 0100
2005	0010 0101
2006	0010 0110
2007	0010 0111
4000	0100 0000
…	…
4007	0100 0111
6000	0110 0000
…	…
6007	0110 0111

Продолжение таблицы 1.в

Шестнадцатеричный код	**Двоичный код**
8000	1000 0000
…	…
8007	1000 0111
A000	1010 0000
…	…
A007	1010 0111
C000	1100 0000
…	…
C007	1100 0111
E000	1110 0000
…	…
E007	1110 0111

Таким образом, из таблицы 1.в видно, что при подключении к трем младшим разрядам генератора слов одного двоичного числа «a2 a1 a0», а к трем старшим разрядам другого двоичного числа «b2 b1 b0» можно просуммировать все возможные комбинации двух трех разрядных чисел.

Примечание: генератор слов удобнее программировать в шестнадцатеричном коде (задание 5.б лабораторной работы №1); семисегментный дисплей («Decoded Seven-Segment Display») сразу декодирует (переводит двоичное число в шестнадцатеричное) «s2 s1 s0» и отображает его в удобном для анализа виде.

Запрограммировав генератор слов, необходимо изучить работу трехразрядного двоичного сумматора, заполнив таблицу 1.г.

Таблица 1.г – Суммирование в трехразрядном двоичном сумматоре

Входы, в различных кодах							**Выходы, в различных кодах**					
$2^{ый}$						**$10^{ый}$**	**$16^{ый}$**	**$2^{ый}$**				**$10^{ый}$**
a2	**a1**	**a0**	**b2**	**b1**	**b0**			**PO**	**s2**	**s1**	**s0**	
0	0	0	0	0	0							
0	0	1	0	0	0							
0	1	0	0	0	0							
0	1	1	0	0	0							
1	0	0	0	0	0							
1	0	1	0	0	0							
1	1	0	0	0	0							
1	1	1	0	0	0							
0	0	0	0	0	1							
0	0	1	0	0	1							
0	1	0	0	0	1							
0	1	1	0	0	1							
1	0	0	0	0	1							
1	0	1	0	0	1							
1	1	0	0	0	1							
1	1	1	0	0	1							

Продолжение таблицы 1.г

Входы, в различных кодах							Выходы, в различных кодах					
$2^{ый}$						$10^{ый}$	$16^{ый}$	$2^{ый}$				$10^{ый}$
a2	**a1**	**a0**	**b2**	**b1**	**b0**			**PO**	**s2**	**s1**	**s0**	
0	0	0	0	1	0							
…	…	…	…	…	…	…	…	…	…	…	…	…
1	1	1	0	1	0							
0	0	0	0	1	1							
…	…	…	…	…	…	…	…	…	…	…	…	…
1	1	1	0	1	1							
0	0	0	1	0	0							
…	…	…	…	…	…	…	…	…	…	…	…	…
1	1	1	1	0	0							
0	0	0	1	0	1							
…	…	…	…	…	…	…	…	…	…	…	…	…
1	1	1	1	0	1							
0	0	0	1	1	0							
…	…	…	…	…	…	…	…	…	…	…	…	…
1	1	1	1	1	0							
0	0	0	1	1	1							
…	…	…	…	…	…	…	…	…	…	…	…	…
1	1	1	1	1	1							

Задание 2. Исследование работы суммирующего и вычитающего счетчиков

а) Исследование первой реализации счетчика

Соберите схему изображенную на рис. 2.а.

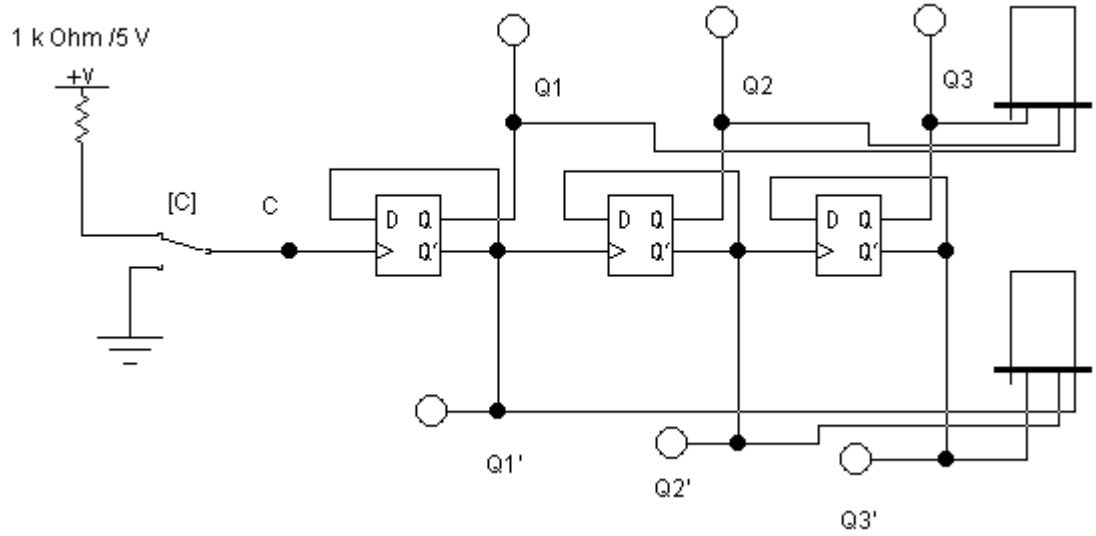


Рис. 2.а – Первая реализация счетчика

Включите схему. Подавая на вход схемы тактовые импульсы при помощи ключа *C* и наблюдая состояние выходов счетчика при помощи семисегментного дисплея декодера и логических пробников, составьте временные

диаграммы работы суммирующего (вычитающего) счетчика. Сделайте выводы о работе схемы.

Примечание: в данном случае семисегментный дисплей предназначен для автоматического перевода поступающего на его вход двоичного числа в десятичное и отображение последнего.

б) Исследование второй реализации счетчика

Создайте вторую реализацию суммирующего и вычитающего счетчика (см. теорию).

Занесите полученную схему.

Составьте временные диаграммы данной реализации.

Задание 3. Исследование счетчика с измененным коэффициентом пересчета

а) Первый вариант

Соберите схему, изображенную на рис. 3.а.

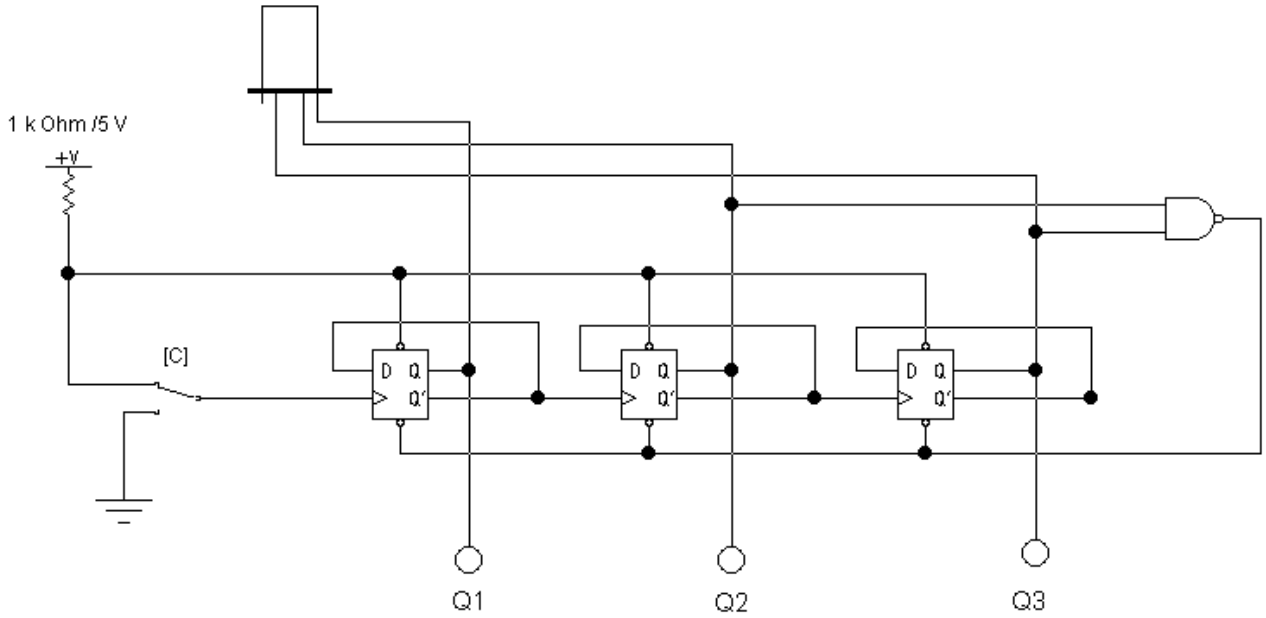


Рис. 3.а – Счетчик с измененным коэффициентом пересчета

Включите схему. Подавая на вход схемы тактовые импульсы при помощи ключа *C* и наблюдая состояние выходов счетчика при помощи логических пробников, составьте временные диаграммы работы счетчика и определите коэффициент пересчета.

б) Второй вариант

Соберите схему изображенную на рис. 3.б.

Включите схему. Подавая на вход схемы тактовые импульсы, при помощи ключа *C* и наблюдая состояние выходов счетчика при помощи логических пробников, составьте временные диаграммы работы счетчика и определите коэффициент пересчета.

Задание 4. Исследование регистров

а) Исследование параллельного регистра

Создайте схему, изображенную на рис. 4.а.

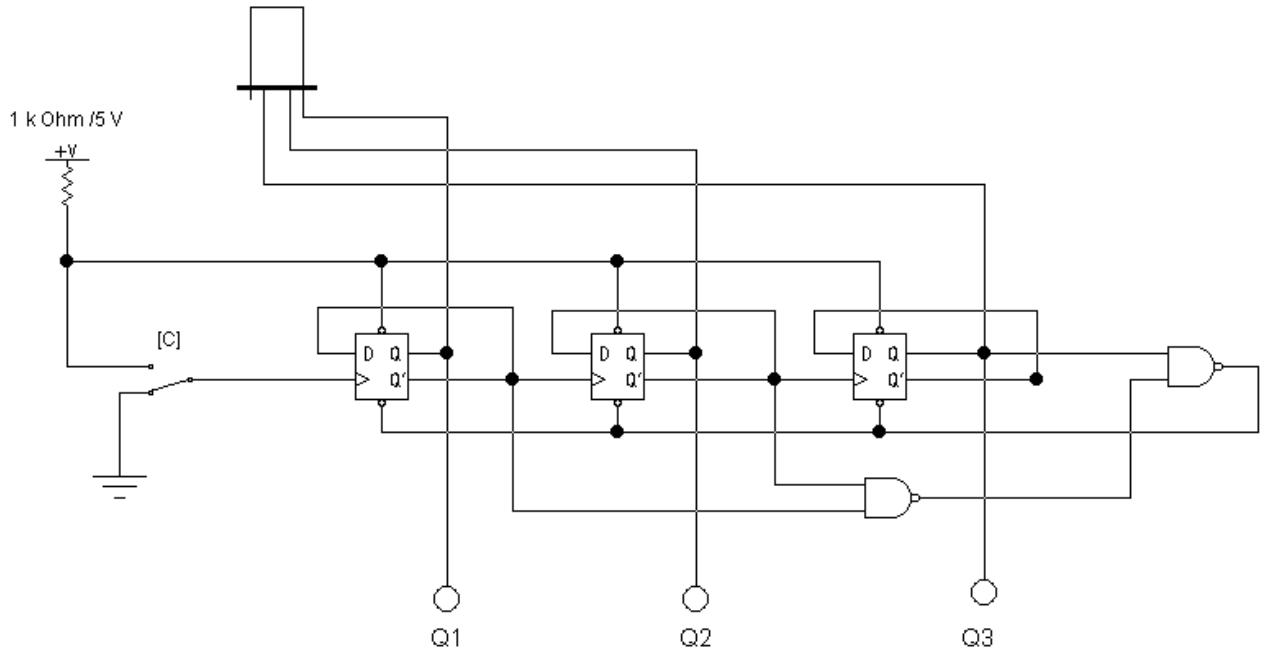


Рис. 3.б – Счетчик с измененным коэффициентом пересчета

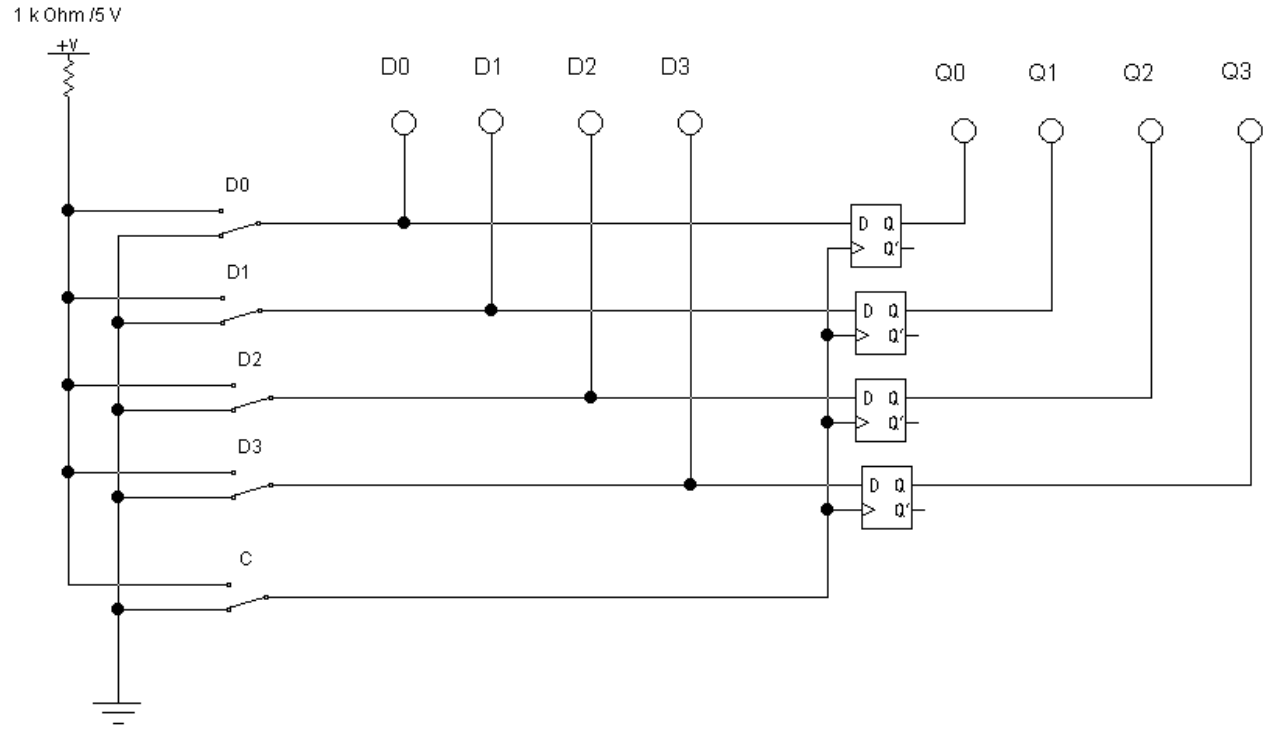


Рис. 4.а – Схема четырехразрядного параллельного регистра

Включите схему. Исследуйте работу полученного двоичного четырехразрядного регистра, заполнив таблицу истинности, приведенную ниже (таблица 4.а).

Таблица 4.а – Параллельный регистр

Входы				**Выходы**			
D0	**D1**	**D2**	**D3**	**Q0**	**Q1**	**Q2**	**Q3**
0	0	0	0				
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	0	0	0				
1	0	0	1				
1	0	1	0				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				

Примечание: запись числа в регистр происходит по переднему фронту разрезающего импульса и сохраняется до появления следующего импульса.

б) Исследование параллельного регистра
Создайте схему, изображенную на рис. 4.б.

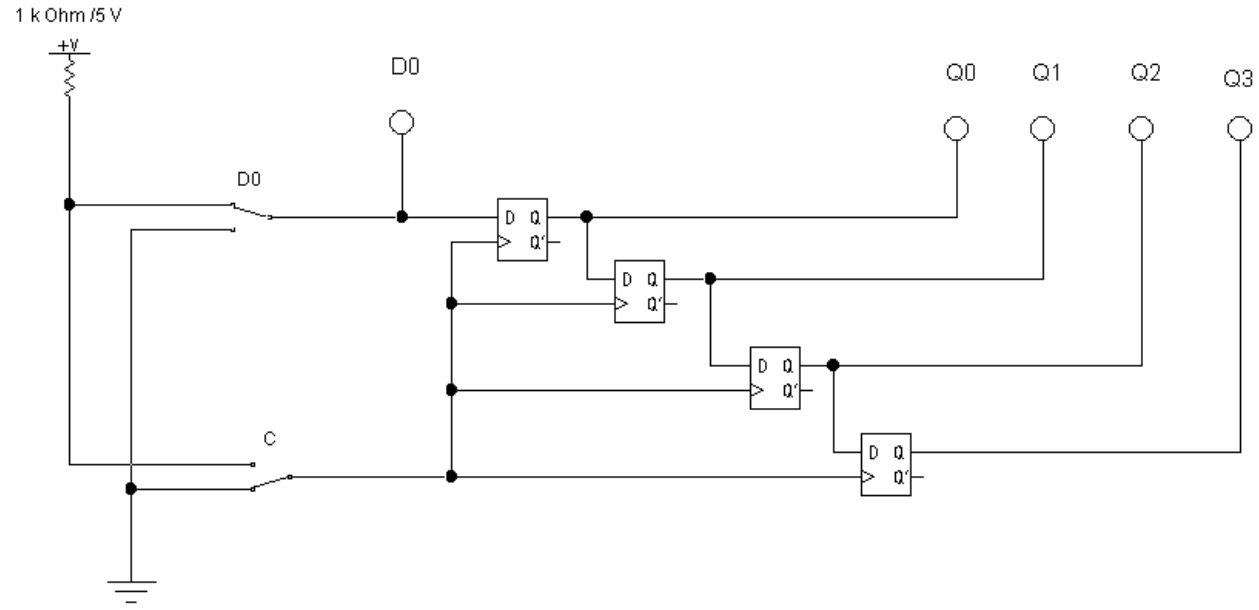


Рис. 4.б – Схема четырехразрядного последовательного регистра

Включите схему. Согласно вашему варианту из таблицы 4.б необходимо выбрать двоичное число, которое требуется получить на выходе регистра сдвига (последовательного регистра).

Задавая необходимые комбинации на входе *D0* и занося информацию в регистр с помощью сигнала разрешения *C*, добейтесь получения на выходе *Q0 Q1 Q2 Q3* требуемого числа.

Постройте временную диаграмму получения в регистре сдвига требуемого числа.

Таблица 4.б – Варианты задания числа для последовательного регистра

Вариант	**Число**			
	D0	**D1**	**D2**	**D3**
1	0	0	1	1
2	0	1	0	0
3	0	1	0	1
4	0	1	1	0
5	0	1	1	1
6	1	0	0	0
7	1	0	0	1
8	1	0	1	0
9	1	0	1	1
10	1	1	0	0

Контрольные вопросы

1. Дайте определение сумматора.
2. Опишите принцип работы сумматоров.
3. Чем отличается полусумматор от сумматора?
4. Как можно составить схему четырех разрядного сумматора в EWB?
5. Приведите примеры применения сумматоров.
6. На основе, каких видов триггеров можно получить схему счетчика?
7. Как из суммирующего счетчика получить вычитающий?
8. Что такое коэффициент пересчета счетчика?
9. Как можно вычислить коэффициент пересчета счетчика?
10. Как можно изменить коэффициент пересчета счетчика?
11. Дайте определение регистра.
12. Чем отличается последовательный регистр от параллельного?
13. Приведите примеры использования параллельных регистров.
14. Приведите примеры использования последовательных регистров.

ЛАБОРАТОРНАЯ РАБОТА № 5

Изучение сетевых возможностей Windows NT

Цель работы:

1. Ознакомиться с администрированием Windows NT.
2. Изучить свойства встроенных учетных записей и групп.
3. Ознакомиться с применением средств системной политики и аудита.
4. Рассмотреть сетевые свойства файловой системы NTFS.
5. Научиться управлять доступом к сетевым и локальным ресурсам.

Краткие теоретические сведения

Концепции Windows NT

Операционная система Windows NT реализована в двух вариантах: Windows NT Server и Windows NT Workstation. Windows NT Server 4.0 – сетевая операционная система с приложениями для Internet, сервисами файлов и печати, службой удаленного доступа, встроенным маршрутизатором, индексированием файлов и управлением сетью. Второй вариант Windows NT – Windows NT Workstation 4.0 во многом напоминает NT Server, но она оптимизирована в качестве операционной системы для рабочей станции. С точки зрения архитектуры и возможностей Windows NT Server является надмножеством Windows NT Workstation и включает в себя все возможности последней.

Средства администрирования Windows NT

Администрирование Windows NT включает как задачи настройки системы непосредственно после инсталляции, так и задачи ежедневной поддержки системы.

Задачи администрирования Windows NT Server и Windows NT Workstation во многом совпадают, однако средства для выполнения этих задач, включенные в каждый из этих продуктов, различаются.

Задачи администрирования могут быть разделены на пять групп:

- Администрирование пользователей и групп пользователей. Включает планирование, создание и поддержку учетной информации пользователей и групп.
- Администрирование средств обеспечения безопасности системы. Включает планирование и реализацию политики безопасности, гарантирующей защиту данных и разделяемых сетевых ресурсов, таких как каталоги, файлы и принтеры.
- Администрирование принтеров. Включает инсталляцию локальных и сетевых принтеров, конфигурирование их для более удобного использования, поиск неисправностей, устранение проблем, возникающих при печати.
- Мониторинг событий и ресурсов. Включает планирование и реализацию политики аудита сетевых событий в целях нахождения брешей в системе защиты, а также мониторинг процессов использования сетевых ресурсов.

- Архивирование и восстановление данных. Включает планирование и выполнение регулярного резервного копирования критических данных.

Средства администрирования Windows NT Workstation используются только для администрирования локального компьютера, а средства администрирования Windows NT Server используются для администрирования любого компьютера сети.

Важнейшими инструментами администрирования пользователей для Windows NT Workstation является *User Manager*, а для Windows NT Server – *User Manager for Domains*.

Другим важным средством администрирования является *Server Manager*. Это средство, включенное в состав Windows NT Server, дает возможность просматривать и управлять компьютерами домена. С помощью Server Manager можно:

- присоединить компьютер к домену;
- создать на удаленном компьютере разделяемый каталог (share) и определить к нему права доступа для пользователей домена;
- просмотреть список работающих на удаленном компьютере сервисов, запускать или останавливать эти сервисы;
- инициировать синхронизацию контроллеров домена, повысит роль BDC (вторичный контроллер домена) до PDC (первичный контроллер домена).

Event Viewer – еще одно средство администрирования, которое оповещает администратора сети обо всех событиях в сети, которые имеют отношение к работоспособности и безопасности системы. Например, такими событиями являются успешный или неуспешный запуск сервиса или приложения, успешная или неуспешная попытка логического входа и т.д.

Backup – средство, которое используется для резервного копирования информации на локальный стример.

Средства администрирования Windows NT Server построены в архитектуре клиент – сервер. После инсталляции Windows NT Server клиентские и серверные части средств администрирования находятся на одном компьютере, но клиентские части могут работать на любых компьютерах сети, работающих под управлением Windows NT Workstation или Windows 9x. Эти клиентские части собраны в одном каталоге Clients\Srvtools дистрибутивного диска Windows NT Server.

Профили пользователей

Профиль пользователя – это набор параметров, определяющих рабочую среду пользователя на том компьютере, на котором он в данный момент работает.

В профиль пользователя входит большое количество переменных:

- состояние рабочего стола (Desktop) – цветовые схемы, обои;
- состояние оболочки (Shell) – элементы графического интерфейса, содержимое папок;
- постоянные сетевые подключения;
- подключаемые сетевые принтеры.

Обычно пользователь сам создает настройки своей рабочей среды, а после выхода его из системы эти настройки сохраняются в файле профиля дан-

ного пользователя, а также в Registry. При очередном логическом входе пользователя параметры его рабочей среды берутся из Registry и восстанавливают для него требуемые установки.

Профиль может также содержать ограничения для пользователя на выполнение некоторых действий. К таким действия относится возможность использования команды Run, доступ к функциям Control Panel, Settings в меню Start, команды Shut Down и некоторые другие действия, которые могут изменить облик системы или позволят получить пользователю доступ к таким функциям или программам, которые недопустимы с точки зрения администратора.

Для ограничения возможностей пользователя с помощью профиля существует специальная утилита *System Policy Editor* – Редактор Системной Политики, входящая в комплект поставки Windows NT Server 4.0. Эта утилита заменила собой User Profile Editor, использовавшейся для этих же целей в Windows NT 3.51. Профили, создаваемые System Policy Editor, теперь воздействуют не только на пользователей, работающих с Windows NT, но и пользователей Windows 9x.

Администрирование пользователей

1. Пользователи, ресурсы и операции доступа

Администрирование пользователей состоит в создании учетной информации пользователей (определяющей имя пользователя, принадлежность пользователя к различным группам пользователей, пароль пользователя), а также в определении прав доступа пользователя к ресурсам сети – компьютерам, каталогам, файлам, принтерам и т.п.

Создание учетной информации пользователей, называемой так же учетной записью, осуществляется в сети Windows NT утилитой User Manager для локального компьютера и User Manager for Domains для всех компьютеров домена. Права доступа к ресурсам задаются в сети Windows NT различными средствами, в зависимости от типа ресурса. Возможность использования компьютеров Windows NT Workstation в качестве рабочих станций – с помощью User Manager for Domains, доступ к локальным каталогам и файлам (только для файловой системы NTFS, поддерживающей права доступа) – с помощью средств Windows NT Explorer, к удаленным разделяемым каталогам – с помощью Server Manager, доступ к принтерам – из панели Printers.

1.1. Типы пользователей и групп пользователей

В сети Windows NT могут быть определены следующие типы пользователей и групп пользователей:

- *локальный интерактивный пользователь компьютера* (пользователь, который заведен в локальной учетной базе данных компьютера, и который работает с ресурсами компьютера интерактивно);
- *локальный сетевой пользователь компьютера* (пользователь, который заведен в локальной учетной базе данных компьютера, и который работает с ресурсами компьютера через сеть);
- *пользователь домена* (пользователь, который заведен в глобальной учетной базе данных домена на PDC);

- *локальная группа компьютера* (может создаваться на всех компьютерах домена, кроме PDC и BDC, в которых она вырождается в локальную группу домена);
- *локальная группа домена* - состоит из пользователей домена (заводится только на PDC);
- *глобальная группа домена* - состоит из пользователей домена (может входить в локальную группу домена).

Для каждого типа групп имеется некоторый набор встроенных групп: Administrators, Server Operators, Users, Everyone, DomainUsers и др.

Для однозначной идентификации глобальной группы в многодоменной сети, используется составное имя, например Marketing\Managers, где Marketing – имя домена, Managers – имя глобальной группы.

1.2. Типы объектов

- *Каталоги и файлы*. Процедуры задания правил доступа различаются для локальных и разделяемых (share) каталогов и файлов. Операции: read, full control, change, add, ...;
- *Принтеры;*
- *Операционная система*. По отношению к этому типу объектов определяются права по выполнению различных сервисов и утилит: вход, архивирование файлов, изменение конфигурации панелей Program Manager, ...

1.3. Типы операций доступа

Операции доступа – это действия объектов над субъектами. Операции могут быть либо разрешены, либо запрещены, либо вообще не иметь смысла для данной пары объекта и субъекта.

Все множество операций разделяется на подмножества, имеющие особые названия:

- разрешения (permissions) – это множество операций, которые могут быть определены для субъектов всех типов по отношению к объектам типа файл, каталог или принтер;
- права (user rights) – определяются для объектов типа группа на выполнение некоторых системных операций: создание резервных копий, выключение компьютера (shutdown) и т.п. Права назначаются с помощью User Manager for Domains;
- возможности пользователей (user abilities) – определяются для отдельных пользователей на выполнение действий, связанных с формированием их операционной среды, например, изменение состава программных групп, показываемых на экране дисплея, включение новых иконок в Desktop, возможность использования команды Run и т.п.

Права и разрешения, данные группе, автоматически предоставляются ее членам, позволяя администратору рассматривать большое количество пользователей как единицу учетной информации.

Возможности пользователей определяются профилем пользователя.

2. Встроенные группы пользователей и их права

Права определяются для объектов типа группа на выполнение некоторых системных операций: создание резервных копий, выключение компьютера (shutdown) и т.п. Права назначаются с помощью User Manager for Domains.

Таблица 1 – Права для встроенных локальных групп

Вид прав	Права							
	Administrators	**Server Operators**	**Account Operators**	**Print Operators**	**Backup Operators**	**Everyone**	**Users**	**Guests**
Log on locally (локальный логический вход)	+	+	+	+	+	-	-	-
Access this computer from network (доступ к данному компьютеру через сеть)	+	-	-	-	-	+	-	-
Take ownership of files (установление прав собственности на файлы)	+	-	-	-	-	-	-	-
Manage auditing and security log (управление аудитингом и учетом событий, связанных с безопасностью)	+	-	-	-	-	-	-	-
Change the system time (Изменение системного времени)	+	+	-	-	-	-	-	-
Shutdown the system (Останов системы)	+	+	-	-	+	-	-	-
Force shutdown from remote system (Инициация останова с удаленной системы)	+	+	-	-	-	-	-	-

Продолжение таблицы 1

Вид прав	Права							
	Administrators	**Server Operators**	**Account Operators**	**Print Operators**	**Backup Operators**	**Everyone**	**Users**	**Guests**
Backup files and directories (Резервное копирование файлов и каталогов)	+	+	+	+	+	-	-	-
Restore files and directories (Восстановление со стримера)	+	+	-	-	+	-	-	-
Load and unload device drivers (Загрузка и выгрузка драйверов устройств)	+	-	-	-	-	-	-	-
Add workstation to domain (Добавление рабочих станций к домену)	+	-	-	-	-	-	-	-
Create and manage user accounts (Создание и управление пользовательской учетной информацией)	+	-	+[1]	-	-	-	-	-
Create and manage global groups (Создание и управление глобальными группами)	+	-	+[1]	-	-	-	-	-
Create and manage local groups (Создание и управление локальными группами)	+	-	+[1]	-	-	-	+[2]	-

Продолжение таблицы 1

Вид прав	Права							
	Administrators	**Server Operators**	**Account Operators**	**Print Operators**	**Backup Operators**	**Everyone**	**Users**	**Guests**
Assign user rights (Назначение прав для пользователей)	+	-	-	-	-	-	-	-
Manage auditing of system events (Управление аудитингом системных событий)	+	-	-	-	-	-	-	-
Lock the server (Блокирование сервера)	+	+	-	-	-	+[3]	-	-
Override the lock of the server (Преодоление блокировки сервера)	+	+	-	-	-	-	-	-
Format server's hard disk (Форматирование жесткого диска сервера)	+	+	-	-	-	-	-	-
Create common groups (Создание общих групп)	+	+	-	-	-	-	-	-
Keep local profile (Хранение локального профиля)	+	+	+	+	+	-	-	-
Share and stop sharing directories (Разделение и прекращение разделения каталогов)	+	+	-	-	-	-	-	-

[1]Операторы учетной информации (Accounts operators) не могут изменять учетную информацию администраторов, или же изменять глобальную группу Domain Admins или локальные группы Administrators, Server Operators, Account Operators, Print Operators или Backup Operators.

[2]Хотя члены группы Users имеют право создавать локальные группы домена, они не смогут им воспользоваться, если им не разрешено входить локально в сервер или не разрешено пользоваться утилитой User Manager for Domains.

[3]Хотя Everyone имеет право блокировать сервер, только пользователи, которые могут также входить локально в этот сервер могут в действительности его заблокировать.

Похожие права можно задать и по отношению к Windows NT Server, не выполняющему роль PDC или BDC – с помощью утилиты User Manager for Domains, а также к Windows NT Workstation с помощью утилиты User Manager.

3. Возможности пользователей

Возможности пользователей – определяются для отдельных пользователей на выполнение немногочисленных действий, касающихся реорганизации их операционной среды:

1. Включение новых программных единиц (иконок) в группу программ панели Program Manager;
2. Создание программных групп Program Manager;
3. Изменение состава программных групп;
4. Изменение свойств программных единиц (например, включение в стартовую группу);
5. Запуск программ из меню FILE в Program Manager;
6. Установление соединений с сетевым принтером, кроме тех (которые уже предусмотрены в профиле пользователя).

Возможности пользователя являются частью так называемого профиля пользователя (User Profile), который можно изменять с помощью утилиты User Profile Editor. Профиль наряду с описанными возможностями включает и установки среды пользователя на его рабочем компьютере, такие как цвета, шрифты, набор программных групп и их состав.

4. Разрешения на доступ к каталогам и файлам

Администратор может управлять доступом пользователей к каталогам и файлам в разделах диска, отформатированных под файловую систему NTFS. Разделы, отформатированные под FAT и HPFS, не поддерживаются средствами защиты Windows NT. Однако можно защитить разделяемые по сети каталоги независимо от того, какая используется файловая система.

Для защиты файла или каталога необходимо установить для него разрешения (permissions). Каждое установленное разрешение определяет вид доступа, который пользователь или группа пользователей имеют по отношению к данному каталогу или файлу. Например, когда вы устанавливаете разрешение Read к файлу MY IDEAS.DOC для группы COWORKERS, пользователи из

этой группы могут просматривать данные этого файла и его атрибуты, но не могут изменять файл или удалять его.

Windows NT позволяет использовать набор стандартных разрешений, которые можно устанавливать для каталогов и файлов. Стандартными разрешениями для каталогов являются: No Access, Read, Add, Add&Read, Change и Full Control.

Стандартными разрешениями для файлов являются:
No Access, Read, Change и Full Control.

Стандартные разрешения представляют собой группы индивидуальных разрешений. Каждому стандартному разрешению соответствует определенная установка фиксированного набора индивидуальных разрешений. Индивидуальные разрешения могут быть:

Read (R), Write (W), Execute (X), Delete (D),
Change Permission (P), Take Ownership (O).

При установке стандартного разрешения рядом с ним в скобках отображаются заглавные буквы установленных индивидуальных разрешений. Например, при установке для файла стандартного разрешения Read рядом со словом Read появляется аббревиатура RX, которая означает, что стандартному разрешению Read соответствует установка двух индивидуальных разрешений – Read и Execute.

Администратор может с помощью утилиты File Manager устанавливать как стандартные, так и индивидуальные разрешения.

Для того чтобы эффективно пользоваться возможностями механизмов безопасности NTFS, нужно помнить следующее:

- Пользователи не могут пользоваться каталогом или файлом, если они не имеют разрешения на это, или же они не относятся к группе, которая имеет соответствующее разрешение.
- Разрешения имеют накопительный эффект за исключением разрешения No Access, которое отменяет все остальные имеющиеся разрешения. Например, если группа CO-WORKERS имеет разрешение Change для какого-то файла, а группа Finance имеет для этого файла только разрешение Read, и Петров является членом обеих групп, то у Петрова будет разрешение Change. Однако если разрешение для группы Finance изменится на No Access, то Петров не сможет использовать этот файл, несмотря на то, что он член группы, которая имеет доступ к файлу.
- Когда вы создаете в каталоге файлы и подкаталоги, то они наследуют разрешения, которые имеет каталог.
- Пользователь, который создает файл или каталог, является владельцем (owner) этого файла или каталога. Владелец всегда имеет полный доступ к файлу или каталогу, так как может изменять разрешения для него. Пользователи - члены группы Administrators - могут всегда стать владельцами любого файла или каталога.
- Самым удобным путем управления защитой файлов и каталогов является установка разрешений для групп пользователей, а не для отдельных пользователей. Обычно пользователю требуется доступ ко многим файлам. Если

пользователь является членом какой-либо группы, которая имеет доступ к этим файлам, то администратору проще лишить пользователя этих прав, удалив его из состава группы, а не изменять разрешения для каждого файла. Заметим, что установка разрешения для индивидуального пользователя не отменяет разрешений, данных пользователю как члену некоторой группы.

Таблица 2 – Индивидуальные разрешения для каталога

Вид работы	**Разрешения**						
	R	**W**	**X**	**D**	**P**	**O**	**FC**
Просматривать имена файлов в каталоге	*	O	*	O	*	O	*
Просматривать атрибуты каталога	*	O	*	O	*	O	*
Добавлять файлы и подкаталоги	O	*	O	*	O	*	*
Изменять атрибуты каталога	O	*	O	*	O	*	*
Переходить в подкаталоги каталога	O	*	*	O	*	O	*
Просматривать владельца каталога и разрешения	*	*	*	O	*	O	*
Удалять каталог	O	*	O	*	O	*	*
Изменять разрешения каталога	O	*	O	*	*	O	*
Становиться владельцем каталога	O	*	O	*	O	*	O

Таблица 3 – Индивидуальные разрешения для файла

Вид работы	**Разрешения**						
	R	**W**	**X**	**D**	**P**	**O**	**FC**
Просматривать данные файла	*	O	O	O	O	O	*
Просматривать атрибуты файла	*	O	*	O	O	O	*
Изменять атрибуты файла	O	*	O	O	O	O	*
Изменять и добавлять данные в файл	O	*	O	O	O	O	*
Выполнять файл, если это программа	O	O	*	O	O	O	*
Просматривать владельца файла и разрешения	*	*	*	O	O	O	*
Удалять файл	O	O	O	*	O	O	*
Изменять разрешения файла	O	O	O	O	*	O	*
Становиться владельцем файла	O	O	O	O	O	*	*

Для файлов имеется следующее соответствие индивидуальных и стандартных разрешений файла:

No Access – Ни одного
Read – RX
Change – RWXD
Full Control – Все разрешения

Стандартные разрешения для каталога представляют собой объединения индивидуальных разрешений для каталога и для файлов, входящих в этот каталог:

No Access	(Ни одного) (Ни одного)
List	(RX) (Не определены)
Read	(RX) (RX)
Add	(WX) (Не определены)

Add&Read	(RWX) (RX)
Change	(RWXD) (RWXD)
Full Control	(Все разрешения) (Все разрешения)

5. Управление профилями пользователей

Когда пользователь локально входит первый раз в какой-либо компьютер, то для него по умолчанию создается профиль. Все настройки среды (цвет фона, обои, шрифты и т.п.) автоматически сохраняются в подкаталоге Profiles системного каталога данного компьютера, например, C:\NT40w\Profiles*username*, где *username* – имя пользователя. Профиль хранится в файле с именем ntuser.dat

Администратор также может настраивать профиль пользователя, входя в какой-либо компьютер под именем этого пользователя.

В отличие от профиля пользователя, который устанавливается по умолчанию, существует также Roaming – перемещаемый профиль пользователя, который формирует одну и ту же среду для данного пользователя, независимо от того, с какого компьютера он вошел в сеть.

Перемещаемые пользовательские профили хранятся, централизовано на сервере, а не на локальных компьютерах пользователей.

Администратор может определить для пользователя один из двух типов перемещаемых профилей.

- Индивидуальный перемещаемый профиль, который пользователь может изменять. Любые изменения, которые пользователь внес в свою среду, вносятся в индивидуальный перемещаемый профиль тогда, когда пользователь логически выходит из сети. Когда тот же пользователь входит снова, с сервера загружается последний вариант профиля. Таким образом, если используются перемещаемые индивидуальные профили, то у каждого пользователя имеется свой собственный перемещаемый профиль. Этот профиль хранится в файле ntuser.dat в одном из разделяемых каталогов сервера.
- Обязательный (mandatory) перемещаемый профиль – это заранее сконфигурированный администратором профиль, который пользователь не может изменить. Один обязательный профиль может быть назначен нескольким пользователям. Этот вид профиля целесообразно назначать тем пользователям, которым требуется одинаковая среда, например, операционистам банка. Обязательный профиль должен иметь расширение .man. Индивидуальный профиль можно сделать обязательным, переименовав его из Ntuser.dat в Ntuser.man.

Начиная с версии 4.0, администратору предлагается более мощное средство управления профилями пользователей – System Policy Editor. С его помощью администратор может изменять профиль пользователя, не входя под его именем. При этом он может устанавливать ограничения, которые невозможно было бы установить, входя под именем пользователя, например, запрет на использование команды Run. System Policy Editor может использоваться для формирования как локальных, так и перемещаемых профилей. Перемещаемый профиль хранится в файле Ntconfig.pol в разделяемом каталоге Netlogon на PDC.

Примечание: Программы-мастера Administrative Wizards позволяют без труда, шаг за шагом, выполнять такие действия, как создание учетных записей пользователей, управление их группами, контроль доступа к файлам и каталогам, установка нового принтера, инсталляция и деинсталляция программ, подключение модема, подготовка пакетов инсталляционных дискет для новых клиентов и контроль за соблюдением лицензионных соглашений для установленных программ. Однако в данной работе использование мастера не рассматривается.

Порядок работы

Задание 1. Изучение встроенных учетных записей и групп

а) Изучение встроенных групп при установке NT Server как контроллер домена

С помощью User Manager for Domains ознакомиться с составом встроенных групп, определив принадлежность (локальная или глобальная). Результаты занести в таблицу 1.а.

Таблица 1.а – Встроенные группы

Имя	Тип	Члены группы
Account Operators		
Administrators		
Backup Operators		
Guests		
Print Operators		
Replicator		
Server Operators		
Users		
Domain Admins		
Domain Guests		
Domain Users		

б) Изучение встроенных групп при установке NT как обычного сервера

С помощью User Manager for Domains ознакомиться с составом встроенных групп. Результаты занести в таблицу 1.б. Отметить какие учетные записи при этом были созданы.

Таблица 1.б – Встроенные группы

Имя	Члены группы
Administrators	
Power Users	
Guests	
Replicator	
Users	

Задание 2. Исследование учетных записей Administrator и Guest

Изучить учетные записи Administrator и Guest, составить подробное описание принадлежности каждой учетной записи к определенным группам и описать права каждой учетной записи.

Задание 3. Изучение системных привилегий

Исследовать список привилегий, заполнить таблицу привилегий встроенных учетных записей и групп (таблица 3).

Таблица 3 – Системные привилегии

Право	Группы и учетные записи, имеющие данное право
Доступ к компьютеру из сети	
Архивирование файлов и каталогов	
Изменение системного времени	
Завершение работы системы	
Принудительное удаленное завершение	
Загрузка и выгрузка драйверов устройств	
Локальный вход в систему	
Овладение файлами или иными объектами	
Управление аудитом и журналом безопасности	
Добавление станций в домен	
Восстановление файлов и каталогов	

Задание 4. Изучение способов внесения новых пользователей в систему
Задайте ряд пользователей с определенными свойствами и правами:

Обычный пользователь – время от времени играет в игры;

Студент – пользователь, использующий некоторые программы, не имеющий право играть в игры и устанавливать программы;

Преподаватель – имеет право запускать все виды программ, может пользоваться всеми сетевыми устройствами, не имеет прав на установку оборудования и программ;

Сформируйте глобальные группы. Определить привилегии новых пользователей путем включения глобальных групп в локальные или прямым назначением привилегий.

Кратко запишите последовательность действий выполняемых в данном задании.

Задание 5. Изучение способов установки прав доступа к файлам и каталогам

В каталоге C:\TEMP создайте произвольные подкаталоги и файлы и назначьте различные права доступа для созданных ранее пользователей. Войдите в систему от имени пользователей и удостоверьтесь в правильном разграничении привилегий и прав.

Задание 6. Изучение разделяемых файловых ресурсов
Создайте разделяемые файловые ресурсы. Проверьте сетевой доступ к разделяемым ресурсам. Выполните передачу права владения файлом. Заполните таблицу прав для разделяемых файловых ресурсов (таблица 6).

Таблица 6 – Права для разделяемых файловых ресурсов

Право	Описание
No Access	
Read	
Change	
Full Control	

Задание 7. Изучение способов создания персонального профиля

С помощью User Profile Editor создать персональный профиль пользователя, назначить домашний каталог. Опишите различные способы и преимущества каждого способа по созданию и расположению домашнего каталога.

Задание 8. Изучение способов редактирования и создания политик

С помощью System Policy Editor изучите способы создания и редактирования политик. Опишите способ, которым можно сделать политику доступной на любой машине в домене.

Контрольные вопросы

1. Перечислите основные средства администрирования Windows NT.
2. Дайте определение учетной записи.
3. Чем локальная учетная запись отличается от глобальной?
4. Какие учетные записи называются встроенными?
5. Перечислите свойства встроенных учетных записей.
6. Дайте определение группе.
7. Что такое привилегии?
8. Как распределяются привилегии во встроенных локальных группах?
9. Перечислите преимущества NTFS перед FAT.
10. Перечислите способы создания общих сетевых ресурсов.
11. Что такое разделяемые файловые ресурсы?
12. Как можно создавать разделяемые файловые ресурсы?
13. Что такое персональный профиль?
14. Что такое системная политика?

ЛИТЕРАТУРА

1. Янсен Й. Курс цифровой электроники: В 4-х т. Т. 2. Проектирование устройств на цифровых ИС. /Пер. с голланд. - М.: Мир, 1987.
2. Блейксли Т.Р. Проектирование цифровых устройств с малыми и большими интегральными схемами. /Пер. с англ. - К.: Вища школа, 1981.
3. Трачик В. Дискретные устройства автоматики. Пер. с польск. Под ред. Д.А. Поспелова. - М.: Энергия, 1978.
4. Аппаратные средства макетирования узлов и устройств ЭВМ / Ковригин Б.Н., Сидуков В.М., Мифтахов Р.К., Тышкевич В.Г., Иванов М.А.; под ред. Б.Н. Ковригина: учебное пособие. М.: МИФИ, 1991.
5. Ковригин Б.Н. Триггерные схемы: Ч. 1. Описание и классификация. - М.: МИФИ, 1976.
6. Ковригин Б.Н. Триггерные схемы: Ч. 2. Синтез и анализ. - М.: МИФИ, 1977.
7. Миллер Р. Теория переключательных схем. В 2-х т. Т. 1. Комбинационные схемы. / Пер. с англ. Под ред. П.П. Пархоменко. - М.: Наука, 1970.
8. Голдсуорт Б. Проектирование цифровых логических устройств. /Пер. с англ.; Под ред. Ю.И.Топчеева. - М.: Машиностроение, 1985.
9. Самофалов К.Г., Корнейчук В.И., Тарасенко В.П. Цифровые ЭВМ: Теория и проектирование / Под общ. Ред. К.Г. Самофалова - 3-е изд., перераб. и доп. - К.: Выща школа, 1989.
10. Антонью А. Цифровые фильтры: анализ и проектирование: Пер. с англ. – М.: Радио и связь, 1983.
10. Электротехника и электроника в экспериментах и упражнениях. Практикум на Electronics Workbench / Под общ. Ред. Д. И. Панфилова. – Т. 2 – Электроника. М.: Додека, 2000.

СОДЕРЖАНИЕ:

ПРИЛОЖЕНИЕ Б

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ 6 СЕМЕСТРА

ПРЕДИСЛОВИЕ

В настоящее время сложно найти область человеческой деятельности, в которой не используются микропроцессоры и цифровые системы на их основе. На протяжении многих лет происходит бурное развитие, как аппаратной, так и программной составляющей в производстве и эксплуатации вычислительных средств; не проходит и месяца без презентаций новых моделей микропроцессоров, программного обеспечения, различных вариантов встраиваемых систем и мощных вычислительных машин. Между тем, несмотря на наличие такого рода динамики, задача создания универсального методического пособия по программированию, отладке и использованию микропроцессоров и их систем (пригодного как для начала изучения, так и для дальнейшего углубления знаний в этой области), пособия актуального и сейчас – является вполне выполнимой. Изучить что-то новое и современное невозможно без освоения азов, глубокого и детального понимания связи аппаратуры и программирования, четкого представления процессов проходящих в микропроцессорных системах при их работе. К сожалению, необходимо отметить, что основа всей микропроцессорной техники, заложенная еще в прошлом веке, в настоящее время существенно не изменились (что является одной из актуальных проблем современного развития всей электронной аппаратуры). В этой связи идеальным началом для изучения являются восьмиразрядные разработки первых микропроцессоров, как отечественных производителей, так и их зарубежные аналоги. Впоследствии это позволит без труда рассмотреть более сложные и современные структуры основанные, например, на «гарвардской архитектуре».

В целом ориентация всей серии методических пособий «Микропроцессорные системы» направлена, прежде всего, на изучение и освоения микропроцессорных систем пригодных для встраиваемых приложений (на основе микроконтроллеров), которые могут быть используемы в бытовой и оргтехнике, в связи, в различных технологических процессах и производствах. Первая часть (данное пособие) посвящено изучению простейших восьмиразрядных микро-

процессоров CISC-архитектуры и их программированию. Вторая часть касается создания микропроцессорных систем как на базе CISC так и RISC архитектуры.

Настоящее пособие представляет собой четыре работы по курсам «Микропроцессорные системы», «Микропроцессорные системы управления» и схожих с ними. Каждая работа содержит необходимые теоретические сведения по исследуемой теме, задания для выполнения и контрольные вопросы.

Пособие охватывает следующие темы: знакомство с системой команд микропроцессора; запись и исследование простых программ; знакомство с принципами формирования микропроцессором управляющих воздействий на различные технологические устройства, их программирование и реализация вычислительных процедур; организация циклов с помощью команд условных переходов; обработка массивов и маскирование данных.

Все исследования по рассматриваемым темам предполагают как минимум наличия специализированных программных средств эмулирования и отладки микропроцессоров, например распространяемых бесплатно и являющихся общедоступными (см. приложение А). При этом наилучшим вариантом является наличие физических моделей микропроцессоров, различных стендов по программированию и отладке, например типа «УМК КР580», «AVR». В любом случае материал пособия подобран и изложен так, что бы была возможность работы с любым из вышеперечисленных способов, и в виде, необходимом для каждой конкретной дисциплины: при проведении практических и лабораторных занятий, для самостоятельной подготовки (в том числе и к экзамену), для выполнения РГР и курсовых проектов.

Для выполнения работ студентам предлагается предварительно самостоятельно ознакомиться с краткой теорией к каждой изучаемой теме. Это даст необходимую теоретическую основу и облегчит выполнение работ, позволив на занятии уделить большее внимание вопросам, обычно вызывающим наибольшее затруднение.

Изучению всего материала учебного пособия может способствовать базовая литература [1 - 6] или вспомогательная [7 - 11].

ВВЕДЕНИЕ

Микропроцессор (МП) серии КР580 (К580) представляет собой однокристальный 8-разрядный процессор, выполненный по n-МОП технологии, который размещен в 40-контактном корпусе, имеет классическую трехшинную структуру с раздельными шинами адреса, данных, управления и выполняет фиксированный набор команд [1, 4]. Существует несколько модификаций этого МП: К580ИК80, К580ИК80А, КР580ИК80, КР580ИК80А (КР580ВМ80А), различающихся исполнением корпуса, быстродействием и некоторыми другими техническими характеристиками. Однако эти различия, за исключением особо отмеченных случаев, несущественны, и поэтому под МП КР580 в дальнейшем будем подразумевать любой из вышеперечисленных типов, а также их зарубежные прототипы Intel 8080 и 8080А. Система команд, изучаемая здесь, так же будет полезна при работе с микроконтроллерами семейства AVR.

Для разработки программ необходимо знание архитектуры микропроцессора и МП системы. Рассмотрим один из вариантов представления архитектуры (структуры) с точки зрения прикладного использования микропроцессора (без учета технической реализации) – через программную модель МП системы и набора команд, с помощью которых обеспечивается программный доступ к элементам этой модели. Под *программной моделью* МП системы понимается совокупность программно-доступных элементов (регистров), объединенных в систему посредством укрупненных направленных связей и дополнительных элементов, обеспечивающих достижение функциональной законченности и целостного представления модели. Программная модель МП системы на базе МП КР580 содержит непосредственно модели: самого микропроцессора МП, памяти МЕМ (MEMORY) и портов ввода-вывода I/O (INPUT/OUTPUT), объединенных тремя шинами: данных – ШД, адреса – ША и управления – ШУ. Порты ввода-вывода используются для сопряжения с различными системными (внутренними и внешними, или периферийными) устройствами.

Модель МП содержит следующие элементы.

1. Шесть 8-разрядных регистров общего назначения (РОН) с однобуквенными именами *B, C, D, E, H, L*. Эти регистры программно доступны как автономно, так и попарно: как три 16-разрядные регистровые пары с однобуквенными именами по первому, старшему регистру пары «*B*» – (*B, C*), «*D*» – (*D, E*) и «*H*» – (*H, L*). Регистры и регистровые пары используются для временного хранения промежуточных данных, адресов (в качестве сверхоперативного запоминающего устройства – СОЗУ) и косвенной адресации основной памяти МЕМ (в качестве указателей памяти). Регистровая пара «*H*» преимущественно используется как указатель памяти в однобайтных командах МП, причем ячейка памяти, адрес которой указан в «*H*»-паре, называется «*M*»-регистром. Этот регистр в функциональном отношении эквивалентен регистрам МП, но имеет большее время доступа.

2. Шестнадцатиразрядный регистр указателя стека *SP* (STACK POINTER) – для хранения адреса вершины программного стека, размещаемого в оперативной памяти и обеспечивающего необходимую глубину вложения подпрограмм при обработке многоуровневых прерываний и модульном построении программ.

3. Шестнадцатиразрядный регистр счетчика команд, или программный счетчик *PC* (PROGRAM COUNTER) – для хранения адреса текущей команды выполняемой программы. При естественном ходе выборки команд, последовательно размещенных в памяти, содержимое счетчика увеличивается от команды к команде на число, равное количеству ячеек, занимаемых в памяти выполненной командой.

4. Восьмиразрядный *A*-регистр, или аккумулятор (накопитель) – основной, узловой рабочий регистр, используемый во всех арифметико-логических командах, командах ввода-вывода данных и др. Совместно с *F*-регистром образует регистровую пару (*A, F*) слова состояния процессора *PSW* (PROCESSOR STATUS WORD). Это слово отражает результаты текущих преобразований данных в 8-разрядном арифметико-логическом устройстве (АЛУ).

5. Восьмиразрядный *F*-регистр, или регистр признаков (флагов) – предназначен для хранения двоичных признаков – бит, отражающих некоторые особенности результата выполнения операции в АЛУ. Признаки могут использоваться последующими командами как для изменения естественной последовательности выборки команд из памяти (передачи управления, или перехода по программе), так и для модификации обрабатываемых данных. *F*-регистр фиксирует 5 различных признаков (3 разряда регистра не используются и содержат биты-константы):

S (SIGN) – бит знака, равен единице, если седьмой бит байта результата равен единице, т. е. результат – отрицательное число;

Z (ZERO) – бит нуля, равен единице, если результат операции равен нулю;

AC (AUXILIARY CARRY) – бит вспомогательного переноса, равен единице, если при выполнении операции был перенос из третьего разряда АЛУ в четвертый, используется в команде десятичной коррекции содержимого *A*-регистра;

P (PARITY) – быт паритета, равен единице, если число единиц результата операции четное;

C,CY (CARRY) – бит переноса, равен единице, если при выполнении операции был перенос из седьмого разряда АЛУ или заем в этот разряд (разряды нумеруются, начиная с младшего, нулевого разряда, размещаемого в регистре в крайней правой позиции); признак очень важен при обработке данных с увеличенной разрядностью.

6. Одноразрядный регистр (триггер) разрешения прерывания *INTE* (INTERRUPT ENABLE). Если триггер установлен в ноль, МП не реагирует на запросы прерывания; если в единицу – прерывание разрешено. После приема запроса прерывания или сброса системы триггер *INTE* автоматически сбрасывается в ноль и для разрешения обработки последующих запросов прерывания его необходимо программно вновь установить в единицу.

7. Связи между элементами МП и другими элементами МП системы: внутренняя двунаправленная 8-разрядная шина данных ВШД(8) – для передачи данных между элементами МП; внешняя двунаправленная 8-разрядная шина

данных ШД (8) – для обмена данными между МП, памятью и портами ввода-вывода; однонаправленная 16-разрядная шина адреса ША(16) – для адресации памяти, портов; 10-разрядная шина управления ШУ(10), связанная с внутренним устройством управления (УУ) микропроцессора и предназначенная для временной коммутации элементов МП системы.

8. Вспомогательные регистры (буферные) для данных и адресов, регистры временного хранения, мультиплексор, дешифратор команд, схемы инкремента и декремента, десятичной коррекции.

Модель памяти представляет собой упорядоченную и пронумерованную последовательность 8-разрядных структурных элементов – ячеек памяти, или внешних регистров. Восьмиразрядное двоичное слово, хранимое в ячейке памяти (регистре), называется байтом, а отдельный двоичный разряд слова – битом. Номер ячейки памяти является ее адресом. Шестнадцатиразрядная шина адреса МП позволяет обращаться к адресному пространству (максимальной совокупности адресуемых ячеек памяти) размером 2^{16} = 65536 = 64 Кбайт (1Кбайт = 1024 байт – общепринятая константа). Для нумерации адресов памяти используется шестнадцатеричная система счисления с цифрами 0, 1, …, 9, A, B, C, D, E, F. Каждый байт может быть представлен двумя полубайтами (старшей и младшей тетрадами бит), значения которых однозначно кодируются указанными шестнадцатеричными цифрами. При этом адрес любой ячейки памяти представляется четырехразрядным («10BCh», «1010h»), а ее содержимое – двухразрядными шестнадцатеричными числами («01h», «BCh).

Примечание: В дальнейшем для указания системы исчисления будем использовать специальное окончание «h» – для шестнадцатеричной, «b» – для двоичной, «d» – для десятичной.

Примем, что в модели памяти время доступа к содержимому любой ячейки памяти не зависит от значения ее адреса и над каждой ячейкой памяти может быть выполнена пара операций: запись байта в ячейку и чтение ее содержимого. Память такого типа называют запоминающим устройством с произвольной выборкой или оперативным запоминающим устройством (ОЗУ) (оперативной

памятью). Эта память используется для хранения программ, исходных, промежуточных и результирующих данных.

В реальных МП системах фактическое количество ячеек памяти – емкость (или рабочее пространство) памяти может быть меньше адресного пространства и, кроме того, разделено на части по каким-либо конструктивным или функциональным признакам. Например, часть памяти может быть предназначена только для хранения информации (память можно только считывать). Такая память называется ПЗУ (постоянное запоминающее устройство) или ППЗУ (перепрограммируемое постоянное запоминающее устройство) и используется для хранения программ и констант. Для учета при программировании подобных реальных ограничений на использование адресного пространства применяют карту памяти – графическое распределение рабочего пространства памяти между отдельными блоками последовательных ячеек, сгруппированных по выделенным признакам.

Модель портов ввода-вывода представляет собой (как и модель памяти) упорядоченную и пронумерованную последовательность 8-разрядных регистров. В системе посредством однобайтных шестнадцатеричных адресов («00h», «01h», …, «FFh») адресуются до 256 портов ввода и вывода информации. Каждое системное устройство обменивается информацией (данными, адресами, управляющими сигналами) с МП путем посылки (приема) байта информации через соответствующий порт и *A*-регистр. В качестве системных устройств ввода-вывода могут рассматриваться как внешние периферийные устройства (АЦП, ЦАП, дисплей, и т.п.), так и внутренние устройства (программируемого контроллера прерываний КР580ВН59, программируемого параллельного адаптера интерфейса КР580ВВ55, и т.п.).

Взаимодействие между элементами МП, памяти и портов ввода-вывода в программной модели сводится к выполнению четырех операций: записи информации в MEM и I/O из регистра МП и ее чтения из MEM и I/O в регистр МП. Вообще говоря, в системе возможны еще две операции: запись в MEM из порта I/O и чтение из MEM в порт I/O, которые появляются при организации

прямого доступа в память (ПДП) со стороны системного устройства, например программируемого устройства прямого доступа КР580ВТ57. Режим ПДП реализуется исключительно техническими средствами (за исключением программной инициализации устройства ПДП) и поэтому здесь не рассматривается. Последовательность функциональных взаимодействий между элементами программной модели МП системы определяется набором команд МП.

Выполнение команд в МП системе, как и в большинстве вычислительных систем, состоит из двух крупных фаз: *выборки* адресованной команды из памяти и ее *выполнения*. Причем вторая фаза в ряде случаев состоит из двух полуфаз: выборки операнда из памяти и выполнения операции над операндом. При естественной последовательности выборки команд в заключение первой фазы или первой полуфазы второй фазы (если она имеет место) происходит автоматическое увеличение на единицу содержимого счетчика команд (адресация следующей команды), и после завершения выполнения текущей команды указанный двухфазный цикл повторяется уже для очередной команды. В реальных системах этот командный цикл выполняется с гораздо большей детализацией фаз (см. ниже).

Длительность командного цикла индивидуальна для каждой команды и может быть выражена количеством тактов, или периодов, генератора синхронизации МП. Зная частоту генератора, можно определить реальное время выполнения команды и фрагментов программ. Длительность команд, оперирующих в МП системе с содержимым внутренних регистров, существенно короче, чем команд, использующих память и порты ввода-вывода. Поэтому при необходимости минимизировать время выполнения программы желательно основную обработку информации вести с использованием внутренних регистров.

Функциональные возможности набора команд определяются в первую очередь форматом данных, команд и способами адресации операндов в командах. Для МП характерна малая разрядность основных структурных элементов (регистров), что приводит к использованию в МП преимущественно одноадресных, реже двухадресных (при межрегистровых пересылках) команд и раз-

личных способов непрямой адресации памяти, сокращающих длину командных слов и размеры программ.

Формат данных (базового информационного слова), принятый в МП КР580 это восьмиразрядное двоичное слово. Хранение и пересылка всех данных в МП системе реализуются в виде последовательности байтов (побайтно), а для представления данных повышенной разрядности используются программно-организуемые многобайтные слова. Байт может использоваться для представления как целых двоичных чисел без знака в диапазоне от «0b» до «255b», так и чисел со знаком: целых положительных в диапазоне от «0b» до «127b» и отрицательных в диапазоне от «-1b» до «-128b» (при представлении чисел в дополнительном коде). Интерпретация числового значения байта (число со знаком или без него) осуществляется программным путем и определяется только программистом (микропроцессорная обработка идентична в обоих случаях). Для чисел со знаком старший седьмой разряд байта интерпретируется как знаковый бит: если он равен «0» то число положительное; если «1» – отрицательное. Кроме рассмотренных представлений байта в виде двоичных или шестнадцатеричных чисел, возможна их интерпретация в качестве двоично-десятичных чисел (байт содержит две десятичные двоично-кодированные цифры) или алфавитно-цифровых символов.

Формат команды (командного слова) зависит от типа операции и может быть одно-, двух- или трехбайтным. Байты многобайтной команды обязательно размещаются в соседних ячейках памяти, и адрес первого байта является адресом команды в целом. *Первый байт* команды представляет код операции, *второй* и *третий* байты – данные или адрес. Заметим, что в трехбайтных командах старший и младший байты адреса или 16-разрядных данных меняются местами, т. е. вначале следует младший, а затем старший байт. Это является особенностью МП КР580, которую необходимо учитывать при программировании. К однобайтным относятся все команды межрегистровых пересылок (или команды типа регистр – регистр), ряд команд обращения к памяти, арифметических и логических операций, команды сдвига, возврата из подпрограммы, обращения к

стеку, разрешения и запрещения прерываний. К двухбайтным относят команды работы с непосредственными данными и команды ввода-вывода. К трехбайтным командам принадлежат команды переходов, вызова подпрограмм, загрузки регистровых пар и прямой записи в память.

В МП КР580 используются пять различных способов адресации:

прямая адресация – второй и третий байты команды содержат прямой адрес операнда в памяти;

регистровая (неявная) адресация – адрес регистра источника и (или) приемника операнда определяется кодом команды;

регистровая косвенная адресация – адрес операнда находится в регистровой паре, адресуемой кодом команды;

непосредственная адресация – операнд размещается во втором байте (для двухбайтной команды) или во втором и третьем байтах (для трехбайтной команды);

стековая адресация – адрес определяется указателем стека; она отличается от регистровой косвенной адресации тем, что при обращении к памяти происходит запись или чтение двух байтов, а содержимое указателя стека автоматически соответственно уменьшается или увеличивается на 2; информация в стеке хранится в том порядке, в котором туда поступает, а извлекается в обратном порядке, по принципу: «последним пришел – первым вышел» (дисциплина LIFO).

Набор команд МП КР580 содержит 78 базовых команд, различающихся мнемоническим обозначением кода операции, а в целом включает 244 различные их модификации [5]. Условные обозначения, используемые для описания каждой команды, их описание, принцип действия, изменяемые флаги, время выполнения – представлено в приложении Б. Шестнадцатеричные значения кода операций команд может быть получены с помощью приложения В.

В целом набор команд МП КР580 ориентирован на решение задач управления и обработки данных, создания систем работающих в реальном масштабе времени [2, 3]. Эффективному решению этих задач способствует широкий спектр команд передачи данных и управления, вызова и рестарта подпрограмм, управления прерыванием и обращения к стеку программируемой глубины.

РАБОТА № 1

ИЗУЧЕНИЕ СРЕДСТВ ПРОГРАММИРОВАНИЯ И ЭМУЛЯЦИИ МИКРОПРОЦЕССОРОВ

Цель работы.

1. Изучение способов запуска, инициализации и настройки средств программирования и эмуляции микропроцессора.
2. Рассмотрение возможностей и средств для отображения работы микропроцессора.
3. Исследование способов программирования и отладки простейшего микропроцессора.

Краткие теоретические сведения

Упрощенная структурная схема микропроцессора КР580 показана на рис. 1.1, см. [6]. Эта схема представляет собой обобщенную программную модель и не содержит: некоторые регистры временного хранения, буферные регистры, мультиплексор и вспомогательные блоки и элементы.

Шина адреса предназначена для передачи адресов в системе, например для адресации ячеек памяти, адресации устройство ввода/вывода.

Шина данных предназначена для передачи данных, например от микропроцессора к памяти, от устройств ввода/вывода к микропроцессору.

Шина управления (набор одинарных линий идущих от устройства управления и синхронизации) предназначена для передачи различных управляющих сигналов, например сигналов разрешения, сигналов синхронизации.

Устройство управления и синхронизации формирует управляющие сигналы, под действием которых выполняются микрооперации в отдельных узлах.

Арифметико-логическое устройство (АЛУ) предназначено для непосредственного выполнения простейших арифметических операций типа сложение, вычитание и логических операция типа конъюнкция, дизъюнкция.

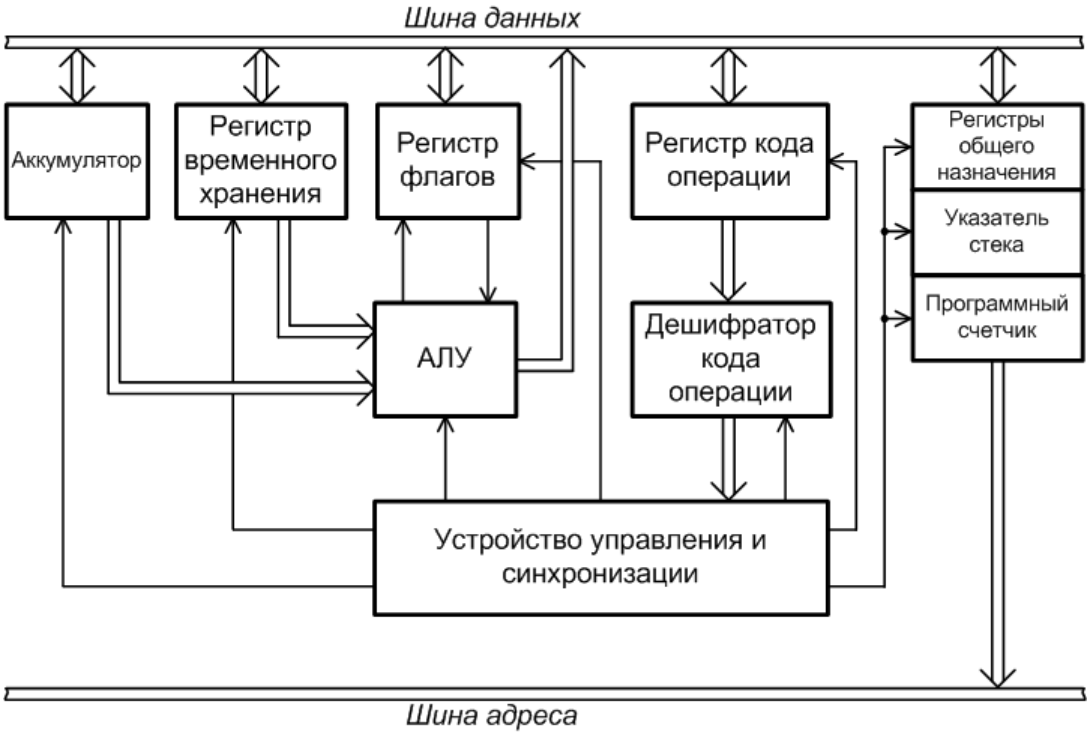


Рис. 1.1. Структурная схема процессора.

Аккумулятор (*A*) это основной регистр системы, используется при выполнении арифметических, логических операций и операций сдвига. Служит источником операнда (числа, участвующего в операции); в него помещается результат выполненной операции.

Регистры (*B, C, D, E, H, L*) образуют так называемый блок регистров общего назначения (РОН) – то есть, эти регистры могут использоваться для хранения, как данных, так и адресов; они могут применяться как одиночные 8-разрядные регистры. В случаях, когда возникает необходимость хранить 16-разрядные двойные числа, они объединяются в пары *BC*, *DE*, *HL*.

Регистр флагов (*F*) предназначен для хранения признаков, выявляемых в числе, которое представляет собой результат выполнения операций в АЛУ. Результат может быть положительным, отрицательным, равным нулю и другим. Эти признаки могут использоваться для изменения последовательности выборки команд из памяти и для модификации обрабатываемых данных.

Регистр кода операции содержит код выбираемой из памяти команды, выполнение которой происходит.

Дешифратор кода операций в зависимости от содержимого регистра кода операций, совместно с устройством управления и синхронизации вырабатывает последовательность сигналов, необходимых для работы системы.

Указатель СТЕКа (*SP*) это 16-разрядный регистр предназначены для хранения текущего адреса вершины стека; при обращении к СТЕКу модифицируется автоматически.

СТЕК это специально организованная область памяти, используемая микропроцессором для временного хранения данных или адресов. Число, записанное в стек последним, извлекается из него первым (дисциплина LIFO).

Программный счетчик (счетчик команд) это специальный регистр обеспечивающий адресацию команд, их последовательную смену и выполнение. В него загружается адрес исполняемой команды (программа это набор последовательно выполняемых команд). После выбора из памяти текущей команды, содержимое счетчика автоматически увеличивается на единицу и таким образом формируется адрес следующей команды (при отсутствии безусловных и условных переходов).

Так как элементную базу микропроцессора составляют элементы, работающие по принципу включено/выключено, то очевидно, что и все операции в нем выполняются в двоичной системе счисления.

Данные и команды в КР580 представляются в виде 8-разрядных двоичных чисел, называемых байтами и имеющих следующий формат:

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

(байт)

где D может принимать значение либо «0», либо «1».

Например, десятичное число 205 в двоичной системе исчисления запишется как «1100 1101». Заметим, что принято разбивать двоичное число на тетрады – группы по 4 бита.

Для работы с адресами в КР580 используются два байта т.к. адресов в системе 65536. Двоичная система, как и десятичная, не являются компактными формами, поэтому принято вместо них использовать шестнадцатеричную си-

стему исчисления. Взаимосвязь между двоичной (*BIN*), десятичной (*DEC*) и шестнадцатеричной (*HEX*) формой показана в виде табл. 1.1.

Таблица 1.1.
Системы исчисления

DEC	HEX	BIN	DEC	HEX	BIN
0	0	0000 0000 0000 0000	…	…	…
1	1	0000 0000 0000 0001	65519	FFEF	1111 1111 1110 1111
2	2	0000 0000 0000 0010	65520	FFF0	1111 1111 1111 0000
3	3	0000 0000 0000 0011	65521	FFF1	1111 1111 1111 0001
4	4	0000 0000 0000 0100	65522	FFF2	1111 1111 1111 0010
5	5	0000 0000 0000 0101	65523	FFF3	1111 1111 1111 0011
6	6	0000 0000 0000 0110	65524	FFF4	1111 1111 1111 0100
7	7	0000 0000 0000 0111	65525	FFF5	1111 1111 1111 0101
8	8	0000 0000 0000 1000	65526	FFF6	1111 1111 1111 0110
9	9	0000 0000 0000 1001	65527	FFF7	1111 1111 1111 0111
10	A	0000 0000 0000 1010	65528	FFF8	1111 1111 1111 1000
11	B	0000 0000 0000 1011	65529	FFF9	1111 1111 1111 1001
12	C	0000 0000 0000 1100	65530	FFFA	1111 1111 1111 1010
13	D	0000 0000 0000 1101	65531	FFFB	1111 1111 1111 1011
14	E	0000 0000 0000 1110	65532	FFFC	1111 1111 1111 1100
15	F	0000 0000 0000 1111	65533	FFFD	1111 1111 1111 1101
16	10	0000 0000 0001 0000	65534	FFFE	1111 1111 1111 1110
…	…	…	65535	FFFF	1111 1111 1111 1111

1. *Стенд УМК КР580*

Стенд УМК КР580 (далее УМК) представляет собой стационарную лабораторную установку и является завершенной микроЭВМ. УМК предназначен для изучения основ микропроцессорной техники, освоения принципов программирования на языке низкого уровня (ассемблере) и выполнен на базе микропроцессора КР580ИК80А.

Общий вид стенда показан на рис. 1.2, он состоит из клавиатуры, дисплея, элементов световой индикации (позиции 1, 2, 3), разъема для подключения макетного ТЕЗа, кнопок и клавиш. Клавиатура состоит из 8 *директивных клавиш* (слева) и 16 *информационных клавиш*. Отображение вводимой и выводимой информации в шестнадцатеричном коде происходит на шестиразрядном дисплее. Четыре разряда слева используются для отображения информации типа «Адрес», два справа – типа «Данные».

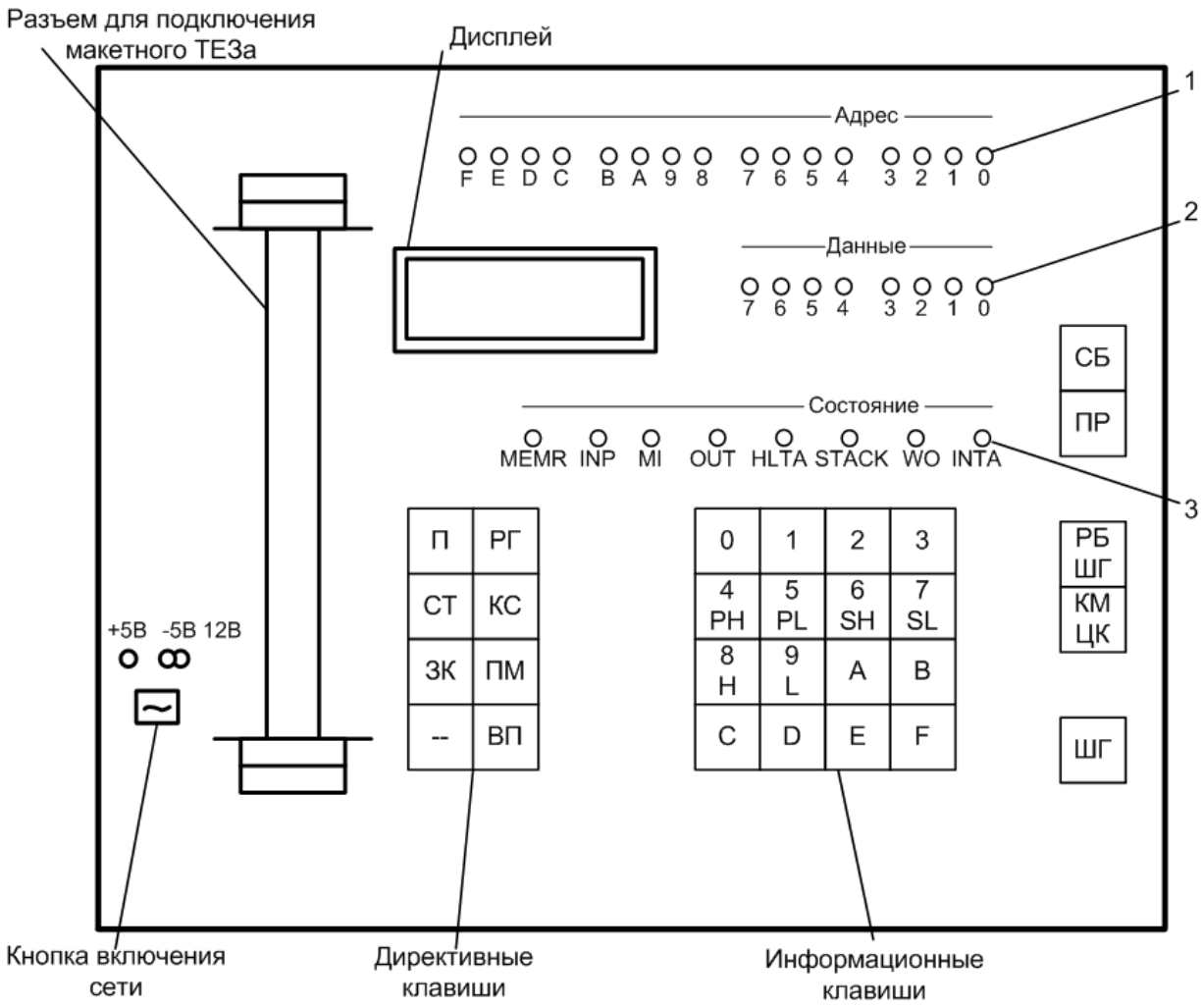


Рис. 1.2. Вид стенда УМК.

Директивные клавиши: «П» – чтение и изменение содержимого памяти; «РГ» – чтение и изменение содержимого регистров; «СТ» – передача управления программе пользователя; «КС» – определение контрольной суммы массива памяти; «ЗК» – заполнение массива памяти константой; «ПМ» – перемещение массива памяти в адресном пространстве; клавиша пробела «__» служит для разделения нескольких переменных при вводе; «ВП» – выполнить, означает конец директивы. *Информационные клавиши* с «0» по «F» служат для ввода чисел в шестнадцатеричном коде и для вызова регистров. *Регистры МП:* «A» (аккумулятор), «B», «C», «D», «E», «H» (старший байт адреса), «L» (младший байт адреса), «F» (регистр флагов), «SL» (младший байт указателя стека), «SH» (старший байт указателя стека), «PL» (младший байт счетчика команд), «PH» (старший байт счетчика команд). *Кнопки*: «СБ» – сброс системы (с нее начина-

ется работа и при нажатии на дисплее в левом углу должен появиться знак «-»); «ПР» – прерывание выполнения программы (состояние всех регистров сохраняется); «ШГ» – выполнение очередного шага при работе в пошаговом режиме. *Переключатели*: «РБ/ШГ» – выбор режима работы (автоматический или пошаговый), при нажатии УМК в пошаговом режиме; «КМ/ЦК» – выбор величины шага, если не нажата то при однократном нажатии кнопки «ШГ» выполняется одна команда, если нажата – один цикл (такт).

В общем случае действия пользователя могут быть представлены в виде:

ДИР [ПАР1, __, ПАР2, __, ПАР3] ВП

где ДИР – соответствует нажатию одной из директивных клавиш (П, РГ, СТ, КС, ЗК, ПМ); ПАР1, ПАР2, ПАР3 – числа, набираемые на информационной клавиатуре и являющиеся параметрами директив (возможно от одного до трех параметров в зависимости от директивы); «__» клавиша пробела служит для разделения нескольких параметров при вводе; ВП – клавиша «выполнить», означает конец ввода. Формат допустимых действий приведен в табл. 1.2.

Таблица 1.2.
Действия в УМК

Формат команд	**Примеры**	**Пояснения**
П, $A_н$, _, $Д_1$, $Д_2$,…, _, $Д_к$, ВП	П, 0803, _, 21, _, 3D, _, 2A, ВП	В ячейку памяти с адресом «0803*h*» записано число «21*h*»; в следующую «0804*h*» – число «3D*h*», в последнюю с адресом «0805*h*» – «2A*h*»
РГ, *и*, *Д*, ВП	РГ, A, 27, ВП	В регистр А заносится «27*h*»
СТ, A_1, _ , $A_к$, ВП	СТ, 0801, _ , 0807, ВП	Выполняется программа, записанная в ячейках памяти с «0801*h*» по «0807*h*».
КС, A_1, _ , $A_к$, ВП	КС, 0927, _ , 0945, ВП	На дисплее высветится сумма содержимого ячеек памяти с «0927*h*» по «0945*h*»
ЗК, A_1, _ , $A_к$, _ , *К*, ВП где *К* – константа (2 значащие цифры)	ЗК, 0800, _ , 0805, _ , 12, ВП	В ячейки памяти с «0800*h*» по «0805*h*» включительно заносится константа «12*h*»
ПМ, A_1, _ , $A_к$, _ , $A_н$, ВП	ПМ, 0927, _ , 093A, _ , 0806, ВП	Массив памяти, ограниченный адресами «0927*h*» и «093A*h*» включительно, переписывается начиная с «0806*h*»
Примечание: В первой колонке *А* – адрес (4 позиции); *Д* – данные (две позиции); нижний индекс «н» – начальный (адрес, элемент и т.п.), «к» – конечный; *и* – имя регистра		

2. *Программа-эмулятор микропроцессорной системы на базе микропроцессора КР580ВМ УМК КР580*

Эмулятор помогает осуществить создание программ на языке ассемблера, используя систему команд МП КР580ВМ80А (см. Приложение А). С помощью него можно проводить отладку и исследование написанных программ в тактовом, командном и сквозном режимах. Программа эмулятор позволяет изучить принципы и порядок выполнения команд, приобрести навыки работы с внешними устройствами, получить представления об организации внешней и внутренней (регистровой) памяти и стековой области. Возможности эмулятора позволяют работать с внешними устройствами: монитором, дисководом гибких дисков (НГМД), жестким диском (НЖМД), сетевым адаптером и принтером. При этом полученные результаты могут быть сохранены в любой момент работы и использованы в дальнейшем. Сама программа-эмулятор имеет интуитивно понятный интерфейс, в состав дистрибутива входит исчерпывающее руководство и возможность контекстной помощи, имеются ранее созданные примеры работы МП.

Работа в программе начинается с главного окна, представленного на рис. 1.3 и разбитого на несколько областей (помечены цифрами).

1. *Главное меню*. Состоит из нескольких подменю, большая часть из них содержит расширенные возможности по работе с программой. Некоторые функции продублированы в других областях, имеются указания на «быстрые клавиши». Содержит подменю: «Файл» (очистка памяти и регистров, открытие сохраненных образов, сохранение образов, экспорт, печать, настройки принтера, выход из программы); «МП-система» (выполнить такт, выполнить команду, выполнить программу, очистить содержимое памяти, очистить регистры); «Вид» (отобразить монитор, буфер дисковод, буфер жесткого диска, сетевого адаптера, принтера, отобразить/скрыть систему команд, показать стековую область памяти); «Настройки» (установка каталогов для внешних ЗУ и настройка сети для принтера); «Помощь» (вызвать справку, о программе).

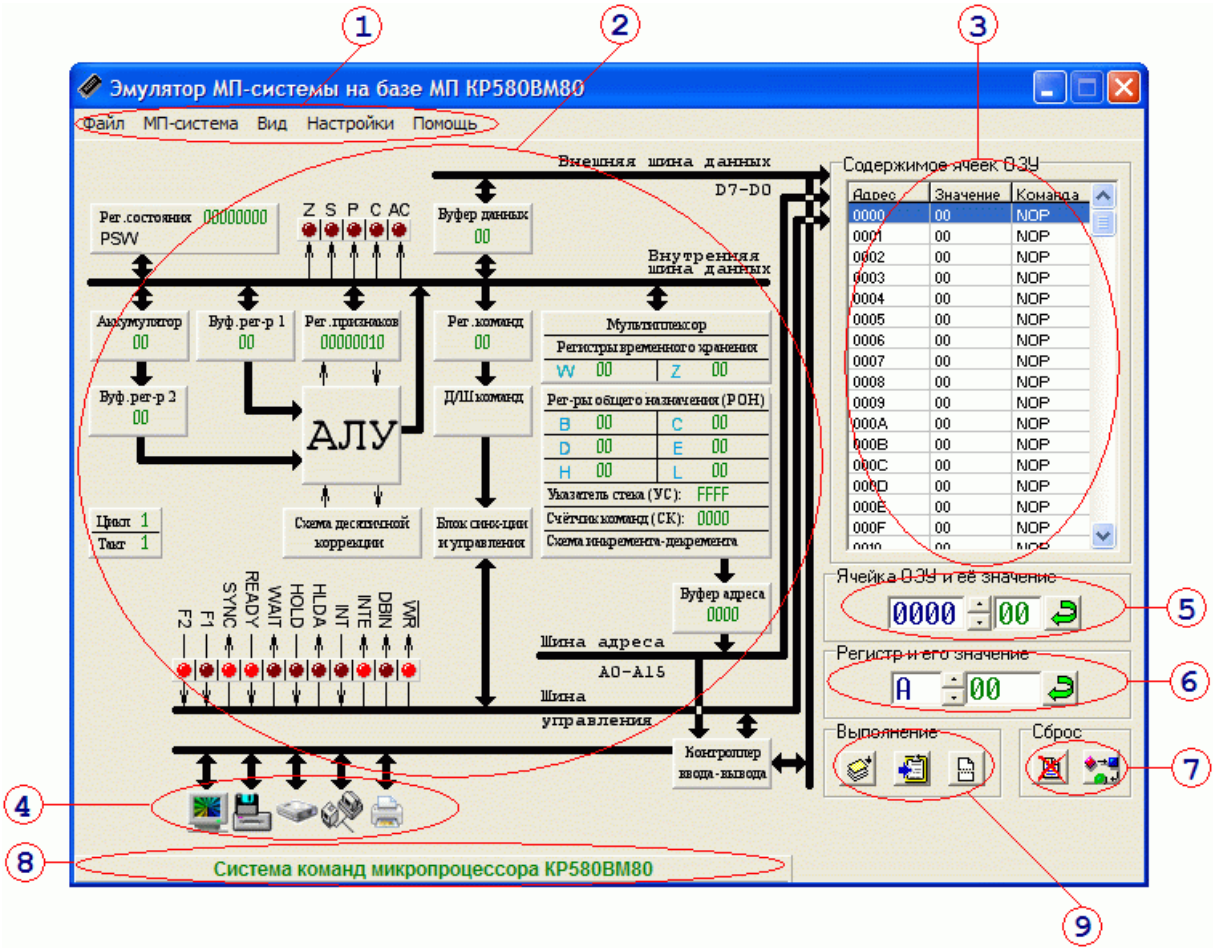


Рис. 1.3. Главное окно программы.

2. *Структурная схема.* Состоит из элементов идентичных рис. 1.1. Регистр слова состояния микропроцессора (PSW) и его значение, представленное в двоичной системе счисления. Буфер данных и его значение, представленное в шестнадцатеричной системе. Регистр-аккумулятор (A) и его значение в шестнадцатеричной системе. Буферные регистры 1 и 2 и их значения в шестнадцатеричной системе счисления. Регистр признаков (флагов) системы и его значение, представленное в двоичной системе счисления, а также индикаторы расшифровки флагов: *Z, S, P, C, AC*. Регистр команд и его значение в шестнадцатеричной системе. Дешифратор команд МП-системы отображающий мнемонику выполняемой команды, закрепленной в регистре команд. Счетчики машинных микроциклов и микротактов иллюстрирующие текущие значения в десятичной системе. Блок АЛУ МП-системы. Схема десятичной коррекции значения регистра-аккумулятора. Блок синхронизации и управления МП-системой. Буфер ад-

реса и его значение, представленное в шестнадцатеричной системе счисления. Блок регистров общего назначения МП-системы и их значения, представленные в шестнадцатеричной системе счисления (*B, C, D, E, H, L*). Блок регистров временного хранения (W, Z) и их значения, представленные в шестнадцатеричной системе счисления. Схема инкремента-декремента, иллюстрирующая свое соответствующее действие условными обозначениями «+1» и «-1» соответственно. Регистр-указатель стека (УС) и его значение, представленное в шестнадцатеричной системе. Регистр-счетчик команд (СК) системы и его значение в шестнадцатеричной системе. Контролер ввода-вывода МП-системы. Индикаторы: F1, F2, …,WR. Порты МП-системы от 00h до 04h для монитора, дисковода, жёсткого диска, сетевого адаптера и принтера соответственно. Все элементы связаны между собой шинами: данных, адреса, управления, внутренней шиной данных и шиной внешних устройств (портов).

3. *Таблица содержимого ячеек ОЗУ:* представлена в виде блока, к которому схематично подведены шины управления, адреса и данных. Таблица условно разделена на 3 столбца: столбец «Адрес» (каждый адрес ячейки ОЗУ представлен в шестнадцатеричном виде и лежит в диапазоне от «0000h» до «FFFFh»); столбец «Значение» (отображает содержимое ячейки памяти по соответствующему адресу, в шестнадцатеричном виде и лежит в диапазоне от «00h» до «FFh»; столбец «Команда» (показывает расшифровку соответствующего значения ячейки в виде мнемокода команды, при этом необходимо помнить, что не всегда мнемокод напрямую связан со значением соответствующей ячейки, ввиду того, что предыдущая команда может быть, к примеру, двухбайтной, а стало быть, в данной ячейке хранятся данные от предыдущей команды, не имеющие никакого отношения к представленному мнемокоду). В нижней области таблицы содержимого ОЗУ установлено выделение коричневого цвета на ту ячейку ОЗУ, на которую указывает указатель стека (SP).

4. *Внешние периферийные устройства:* подключены к общей шине, идущей от контроллера ввода-вывода. Всего подключено 5 устройств к соответствующим портам ввода/вывода (адреса «00h»…«04h»). Порт 00h «Монитор»

представляет собой виртуальный монитор, обеспечивающий вывод графической или текстовой информации; графический режим соответствует разрешению 256x256 пикселей и глубине цвета 128 бит на пиксель, а текстовый 39x20 символов и глубине цвета 128 бит на символ; одновременно монитор поддерживает два этих режима, т.е. может отображать и текст и графику. Порт 01h «Накопитель на гибких магнитных дисках» представляет собой виртуальный буфер дисковода, обеспечивающий вывод данных в файл на гибкий диск в реальном времени при наличии дискеты в дисководе A. Порт 02h «Накопитель на жестких магнитных дисках» представляет собой виртуальный буфер обеспечивающий вывод данных в файл в реальном времени на накопитель на жестких дисках реальной машины. Порт 03h «Сетевой адаптер» представляет собой виртуальный полудуплексный буфер данных, обеспечивающий передачу в реальном времени по сети вычислительных машин по протоколу TCP/IP; адрес и порт указывается в настройках сетевого адаптера. Порт 04h «Принтер» представляет виртуальный буфер данных обеспечивающий вывод на принтер.

5. *Ячейка ОЗУ и ее значение*. Состоит из поля *ввода* (отображения) текущего номера ячейки ОЗУ в виде шестнадцатеричного четырехразрядного числа, может изменяться при ручном вводе, с помощью стрелок «вверх» и «вниз» (на единицу), в режиме работы (автоматически). Поле *значение* (двухразрядное шестнадцатеричное число) может так же изменятся как вручную, так и в режиме «работа»; при вводе информации и нажатии клавиши «Enter» происходит автоматическое увеличение адреса ОЗУ на единицу с переходом к вводу содержимого этой ячейки памяти.

6. *Регистр и его значение*. Состоит из поля отображения имени регистра (выбирается) и его значения. Правила отображения и изменения идентичны п.5.

7. *Сброс*. Состоит из двух кнопок предназначенных для сброса только регистров или всей памяти системы.

8. *Система команд микропроцессора*. Содержит автоматически выдвигающуюся таблицу команд микропроцессора и их коды (идентичную приложению В) с возможностью контекстной помощи по каждой команде (идентично

приложению Б). Команды можно «перетаскивать» мышью в поле значений ячеек ОЗУ, тем самым программируя микропроцессор.

9. *Выполнение*. Представляет собой группу кнопок: выполнить такт позволяет выполнить один такт текущей команды, на которую указывает счетчик команд (если команда состоит из нескольких тактов, становятся недоступными некоторые элементы управления главного окна, а вступившие изменения значений в выполненном такте отмечаются красным цветом); выполнить команду целиком позволяет выполнить все такты текущей команды, на которую указывает счетчик команд; выполнить программу запускает программу на выполнение, начиная с адреса, на который указывает счётчик команд, до адреса в котором содержится код команды HLT ($76h$) либо по принудительному отжатию этой кнопки.

3. *Эмулятор работы микропроцессора КР580ВМ80*

Вид главного окна эмулятора, принцип его работы, назначение основных элементов (см. приложение А) имеет много общего с рассмотренным в п.1 стендом УМК и представлен на рис. 1.4; на рис. 1.5 показана его структурная схема (вызывается из главного окна, в меню «Структурная схема»).

Меню «Файл» главного окна содержит элементы позволяющие как загрузить созданную ранее программу, так и сохранить ее. Меню «Система команд» позволяет перейти в дополнительное окно, идентичное приложению В, в котором можно выбрать интересующие группы команд и получить по ним исчерпывающую информацию, идентичную приложению Б. Меню «Помощь» позволяет осуществить работу со справкой по программе.

Клавиатура доступа к регистрам и ячейкам памяти (см. рис. 1.4): кнопка «ОЗУ» – выбор ячейки памяти; кнопка «Регистр» – выбор регистров МП; кнопки «PcH» и «PcL» – выдают на дисплей содержимое старшего и младшего байта счетчика команд соответственно; кнопки «SpH» и «SpL» – выдают на дисплей содержимое старшего и младшего байта указателя стека соответственно.

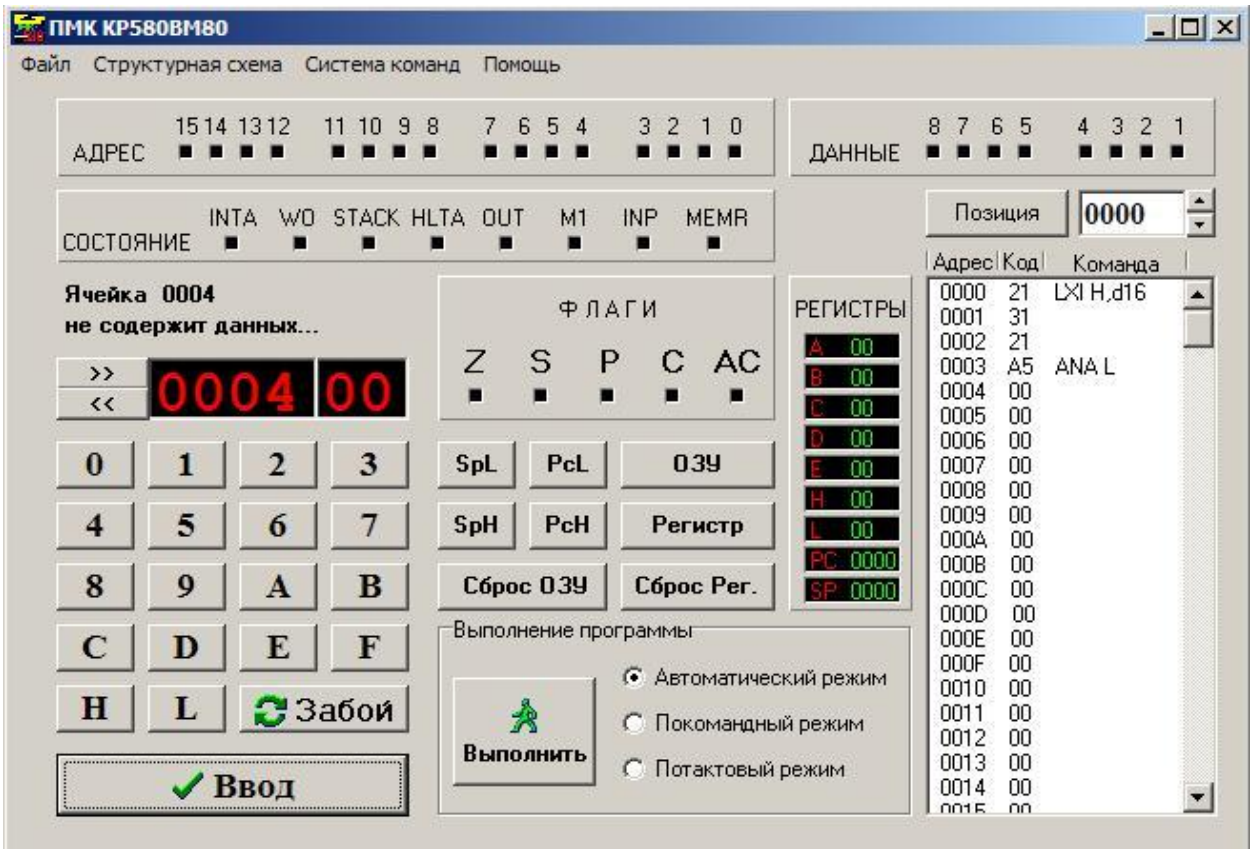


Рис.1.4. Вид главного окна эмулятора.

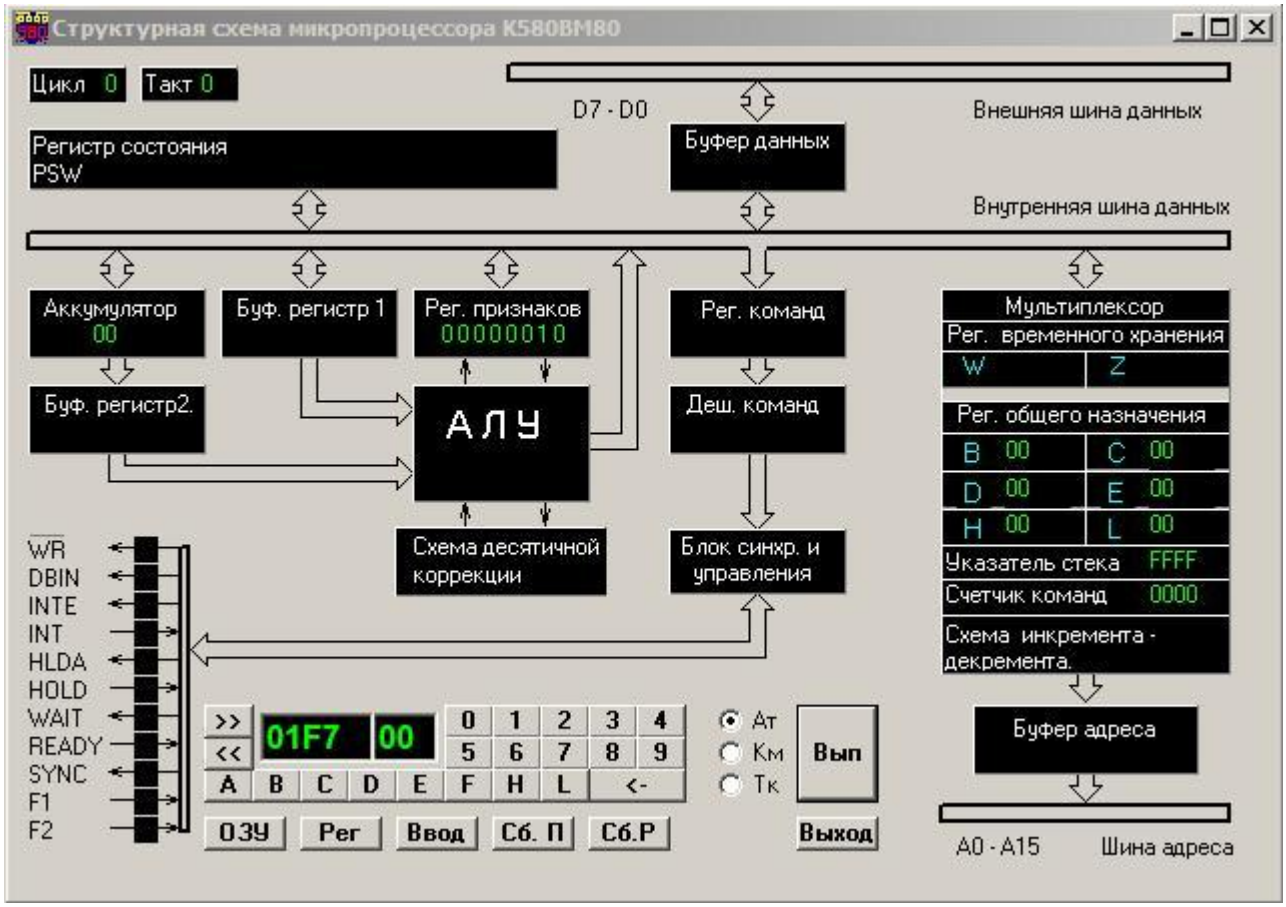


Рис. 1.5. Структурная схема эмулятора.

Для просмотра и изменения содержимого регистров необходимо нажать кнопку «Регистр» и выбрать соответствующую кнопку с обозначением необходимого регистра. Если необходимо просмотреть несколько регистров, то нажатием кнопки «>>» можно вывести на дисплей их содержимое в следующей последовательности: *A, B, C, D, E, F, H, L, Z, W*. Регистры *Z* и *W* не модифицируются, возможен только их просмотр. Кнопка «<<» позволяет просматривать регистры в противоположном направлении. При этом в левой части дисплея отображается имя выбранного регистра, а в правой – его содержимое. Изменение содержимого регистра выполняется набором, с помощью цифровых кнопок требуемой величины, после чего нажимается кнопка «Ввод».

Для просмотра и изменения содержимого ОЗУ необходимо нажать кнопку «ОЗУ» и набирать на клавиатуре необходимый адрес, он отобразится в левой части дисплея, в правой части дисплея отобразится содержимое по выбранному адресу. Работа с ячейками памяти идентична рассмотренной выше работе с регистрами. Все манипуляции с содержимым ОЗУ отражаются в соответствующем поле программы (столбцы адрес, код, команда).

При вызове структурной схемы работа с эмулятором происходит в этом окне (правила работы идентичны рассмотренным выше).

При работе с системой могут использоваться следующие функциональные клавиши: кнопка «Сброс ОЗУ» – предназначена для обнуления содержимого всего пространства памяти МП; кнопка «Сброс Рег» – необходима для обнуления содержимого всего регистров и флагов МП; кнопки «Автоматический режим», «Покомандный режим», «Потактовый режим» – предназначены для включения одного из трех режимов выполнения программ: автоматически, по одной команде, и режим выполнения по тактам команды; кнопка «Выполнить» – предназначена для выполнения программ пользователя в одном из трех (заданных) режимов.

4. *Программная модель УМПК-80*

Главное окно программной модели УМПК-80 (см. приложение А) со всеми активированными элементами меню «Просмотр» представлено на рис. 1.6.

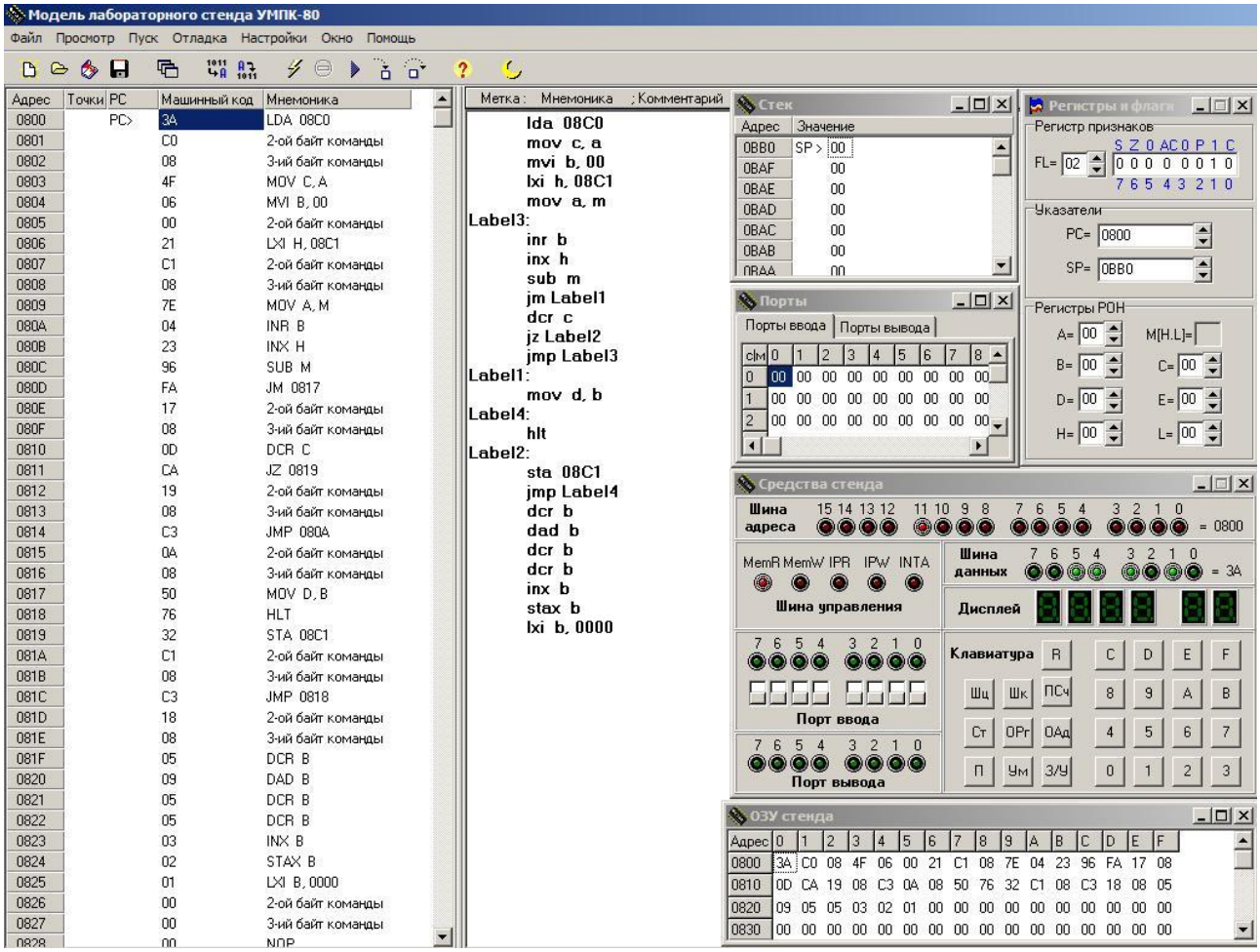


Рис. 1.6. Программная модель УМПК-80.

Программная модель содержит исчерпывающее руководство (см. меню «Помощь») и позволяет аналогично рассмотренным выше средствам работать непосредственно с адресами и их содержимым (левая часть рис. 1.6). При этом так же имеется возможность ассемблирования и дизассемблирования программы пользователя. Вспомогательные элементы «Стек», «Порты», «Средства стенда», «ОЗУ стенда», «Регистры и флаги» (работа которых идентична рассмотренным ранее) позволяют более гибко и наглядно проводить исследования работы микропроцессора.

Порядок работы

Задание 1. Запуск, инициализация и настройка средства программирования и эмуляции

Запустите указанную преподавателем программу (включите в работу стенд) из перечня средств представленного выше.

Изучите общий вид программы (стенда), запустите (в случае наличия) помощь по средству путем нажатия на клавиатуре клавиши «F1» или выбрав соответствующую позицию в меню.

Ознакомьтесь со справкой (при наличии) о работе программного средства и основными правилами работы с ним; изучите информацию о рассматриваемом микропроцессоре.

Изучите методику настройки и выполните ее для выбранного средства (путей и каталогов программы, необходимых адресов, портов, прерываний и прочих параметров).

Занесите в отчет краткие результаты выполнения пунктов данного задания, снабдив их рисунками (скриншотами) и Вашими выводами.

Задание 2. Исследование средств визуального отображения работы микропроцессора

Изучите возможности средства по отображению работы микропроцессора (имеющиеся в наличии индикаторы, средства числового отображения, мониторы, дисплеи, структурные и мнемонические схемы); в случае необходимости активируйте дополнительные окна в программе для отображения этих функций.

Рассмотрите систему команд имеющегося микропроцессора (активируйте дополнительное окно или раздел справки, посмотрите приложения методического пособия), способы записи команд и данных в память МП-системы.

Занесите в отчет информацию и способы ее анализа по следующим ключевым элементам системы: шина адреса, шина данных, шина управления, ячейки ОЗУ, область для стека, флаги, регистры, элементы анализа состояния МП, буферные элементы, порты ввода-вывода.

Задание 3. Изучение основных способов выполнения работ на имеющихся средствах программирования и эмуляции

Запустите (при наличии) имеющиеся примеры программ.

Изучите способы изменения содержимого ОЗУ, регистров, значений сигналов на линиях портов ввода-вывода.

Освойте принципы написания и записи программ для соответствующего средства.

Рассмотрите способы запуска системы в автоматическом режиме, покомандном, по циклам (по тактам).

Рассмотрите возможности и особенности по сбросу (переводу в исходное состояние), сохранению результатов и загрузки ранее созданных программ.

Обеспечьте запись информации в исследуемое программное средство согласно Вашим вариантам и табл. 1.3.

Таблица 1.3.
Информация в микропроцессоре

Вариант	Содержимое элементов, в шестнадцатеричной системе (*h*)																
	Регистры									Адреса ячеек ОЗУ						Порты	
	A	B	C	D	E	H	L	SP(УС)	PC(СК)	0800	0801	0802	0803	080A	080B	01	02
1	01	0A	1A	16	B1	F0	21	E0	AA	30	59	D1	08	2F	8A	00	10
2	02	0B	2A	15	B2	E0	22	D0	BB	31	58	D2	09	3F	9A	20	30
3	03	0C	3A	14	B3	D0	23	C0	CC	32	57	D3	10	4F	1B	40	50
4	04	0D	4A	13	B4	C0	24	B0	DD	33	56	D4	11	5F	1C	60	70
5	05	0E	5A	12	B5	B0	25	A0	EE	34	55	D5	12	6F	1D	80	90
6	06	0F	6A	11	B6	A0	26	99	FF	35	54	D6	13	7F	1E	A0	B0
7	07	1F	7A	10	B7	90	27	98	11	36	53	D7	14	8F	1F	C0	D0
8	08	2F	8A	09	B8	80	28	97	22	37	52	D8	08	2F	8A	E0	F0
9	09	3F	9A	08	B9	70	29	96	33	38	51	D9	09	3F	9A	01	02
10	10	4F	1B	07	C1	60	2A	95	44	39	50	E0	10	4F	1B	03	04
11	11	5F	1C	06	C2	50	2B	94	55	40	49	E1	11	5F	1C	05	06
12	12	6F	1D	05	C3	40	2C	93	66	41	48	E2	12	6F	1D	07	08
13	13	7F	1E	04	C4	30	2D	92	77	42	47	E3	13	7F	1E	09	10
14	14	8F	1F	02	C5	20	2E	91	88	43	46	E4	14	8F	1F	11	12
15	15	9F	2F	01	C6	00	2F	90	99	44	45	E5	15	9F	2F	13	14

Примечание: в таблице для SP и PC (являющихся шестнадцатиразрядными) указаны значения для младшего бита, старший бит необходимо заполнить

так, чтобы не было пересечения областей СТЕКа, области выделенной для данных и области программного кода.

В случае отсутствия возможности напрямую (эмулирующими средствами) обеспечить запись в указанные табл. 1.3 порты можно с адреса, выделенного для программ, создать фрагмент программного кода приведенного ниже – табл. 1.4.

Таблица 1.4.
Программа вывода из порта

Адрес	**Содержимое ячейки ОЗУ, машинный код команды**	**Обозначение команды (мнемокод)**	**Комментарий**
adr 1	3E	*MVI* A,«d8»	Назначение команды *MVI*: в регистр «А» заносится второй байт команды.
adr 2	«d8»		Символическое обозначение 8-и разрядного числа
adr 3	D3	*OUT* «N»	Назначение команды *OUT*: содержимое регистра «А» пересылается в порт, номер которого указан во втором байте команды
adr 4	«N»		Символическое обозначение номера порта

Занесите в отчет краткие результаты выполнения пунктов данного задания, снабдив их рисунками (скриншотами) и Вашими выводами.

Контрольные вопросы

1. Дайте определение основных используемых шин в МП-системе.
2. Из чего состоит обобщенная структура процессора?
3. Что такое аккумулятор?
4. Что такое РОН и для чего они предназначены?
5. Приведите основные флаги, использующиеся в регистре признаков.
6. Что такое байт информации?
7. Для чего нужен указатель стека? Что такое стек?
8. Каким образом используется дешифратор кода операций?
9. Запишите правила и представьте примеры перехода от одной системы исчисления к другой.
10. Для чего нужен программный счетчик? Как он используется?

РАБОТА № 2

ЗАПИСЬ И ВЫПОЛНЕНИЕ ПРОСТЫХ ПРОГРАММ

Цель работы.

1. Знакомство с форматом команд и этапами их выполнения.
2. Изучение команд пересылки данных и арифметических команд.
3. Исследование простейших программ.

Краткие теоретические сведения

1. *Общий принцип работы микропроцессорного устройства*

С помощью программного счетчика микропроцессора на шину адреса системы выдается адрес выполняемой команды. Считанная из памяти по этому адресу команда (ее код) поступает на шину данных, затем считывается микропроцессором, декодируется и выполняется. В программном счетчике (счетчике команд) автоматически формируется адрес следующей команды. После окончания исполнения текущей команды на шину адреса поступает адрес очередной команды, и процедура повторяется вновь.

Наиболее эффективно в микропроцессорах выполняются программы, составленные на языке низкого уровня – Ассемблере. Ассемблер позволяет в удобном для человека виде представлять команды (в виде мнемонической записи) являющиеся для микропроцессора элементарными операциями (пересылки, ввода-вывода, арифметических операций и т.п.). Весь набор команд может быть использован для решения сложных задач обработки информации и управления. Программа, составленная на Ассемблере, в дальнейшем должна быть переведена в машинный код для записи в микропроцессор: либо с помощью специальных программ-компиляторов, либо вручную с помощью знаний правил записи и выполнения операций в МП-системах.

Необходимо отметить, что любая программа, написанная на языке высокого или низкого уровня, может быть декодирована и представлена в виде кода на Ассемблере. Это может быть использовано при отладке, поиске ошибок в

работе системы, настройке взаимодействия между различными аппаратными средствами и т.п. операциях.

Команды в рассматриваемом типе микропроцессора в зависимости от места, занимаемого в памяти, могут быть одно-, двух- и трехбайтовыми (см. приложение Б).

Большинство команд являются однобайтовыми и содержат в себе только код операции. Например (см. приложение В): код команды «78*h*» имеет мнемонику «*MOV* A,B» и означает пересылку содержимого регистра «В» в регистр «А»; код команды «81*h*» имеет мнемонику «*ADD* C» и означает сложение содержимого регистра «С» и регистра «А», результат заносится в регистр «А». Необходимо отметить что аккумулятор (регистр «А») во многих операциях явно не указывается, но подразумевается.

В двухбайтовой команде на первом месте, как и ранее, указывается код выполняемой операции, а во втором байте – приводится число, являющееся операндом (если выполняется какая-либо операция), либо номером устройства ввода-вывода (при выполнении операция обмена данными). Например: (см. приложение В): код команды «06*h* A5*h*» имеет мнемонику «*MVI* B, A5*h*» и означает пересылку константы в шестнадцатеричной системе исчисления заданной как «A5*h*» в регистр «*B*».

Байты трехбайтовой команды имеют следующее назначение: первый байт – код операции; второй байт – младшие разряды двухбайтового операнда; третий байт – старшие разряды двухбайтового операнда. При этом второй и третий байты используются для указания двухбайтового адреса (команды или ячейки памяти), или двухбайтового операнда при работе с парой регистров. Например: (см. приложение В): код команды «3A*h* 2B*h* 09*h*» имеет мнемонику «*LDA* 2B*h*, 09*h*» и означает пересылку в аккумулятор содержимого ячейки памяти (*Я*П), адрес которой указан как «092B*h*». Необходимо еще раз особо подчеркнуть способы задания адресов в КР580 – сначала указывается младшая часть адреса, затем старшая.

Весь набор команд КР580 представлен в приложении Б, ниже рассмотрены некоторые группы команд, необходимые для выполнения заданий данной работы.

2. *Команды пересылки данных*

Команды пересылки данных осуществляют обмен данными между регистрами и памятью. Основные команды этой группы имеют обозначение «*MOV*» или «*MVI*». По команде «*MOV*» происходит пересылка содержимого одного регистра в другой или содержимого регистра в память, или содержимого памяти в регистр. Например, по команде «*MOV* B,H» происходит передача числа из регистра «H» в регистр «B». По команде «*MVI*» осуществляется непосредственная запись числа в аккумулятор, регистр или память (то есть само число непосредственно присутствует в команде, сама команда двухбайтовая). Например, команда «*MVI* B,35*h*» позволяет непосредственно записать число «35» (в шестнадцатеричном коде) в регистр «B».

При записи команд на языке Ассемблера аккумулятор обозначается буквой «A» (в коде некоторых команд явно не присутствует). Константа присутствующая в команде обозначается «d8». Ячейка памяти может обозначаться как «M». При этом используется косвенная адресация, т.е. фактически обращение происходит к содержимому ячейки памяти, адрес которой записан в паре регистров «H» (старшая часть адреса) и «L» (младшая часть адреса). Полное описание команд представлено в приложении Б (раздел команды пересылок), коды команд – см. приложение В.

3. *Арифметические команды*

Команды данной группы предназначены для выполнения операций сложения, вычитания, увеличения или уменьшения содержимого регистров или ячеек памяти на единицу.

Команды сложения и вычитания здесь всегда выполняются между первым операндом (числом), находящимся в аккумуляторе, и вторым операндом (чис-

ло, указанное в команде или находящееся в регистре или в памяти). Результат выполнения операций здесь всегда помещается в аккумулятор.

Характеристики и описания арифметических команд микропроцессора КР580 – см. приложение Б, машинные коды представлены в приложении В.

4. *Пример простейшей программы*

Составим простейшую программу для вычисления формулы:

$$(X - 200b + Y) \rightarrow Z,$$

где «X», «Y» – содержимое ячеек памяти с адресами «0B00*h*» и «0B01*h*» соответственно, где располагаются первый и второй операнд; «200*b*» – заданная константа в десятичной системе исчисления; «Z» – указывает на местоположение результата, а именно на ячейку памяти с адресом «0B02*h*».

В соответствии с формулой требуется загрузить из адреса «0B00*h*» число «X», вычесть из него константу 200 и сложить с числом «Y», загруженным из ячейки памяти «0B01h». Результат необходимо сохранить в памяти по адресу «0B02*h*».

Схему решения данной задачи можно представить в виде нескольких этапов, разбитых в соответствии с используемыми в них типами операций.

Этап 1. Осуществляется загрузка одного из регистров содержимым ячейки памяти с адресом «0B00h». Эта операция может быть выполнена командой прямой загрузки аккумулятора, мнемокод этой команды – *LDA* adr. В нашем случае строка программы на Ассемблере будет иметь вид

LDA 0B00h

Этап 2. Происходит выполнение операции вычитания: из содержимого аккумулятора отнимается числом «200», которое в шестнадцатеричной системе исчисления записывается как «C8*h*». Мнемокод соответствующей команды «*SUI* d8», результат поместится в аккумулятор. Для данных нашей задачи окончательно будем иметь

SUI C8*h*

Этап 3. Полученный промежуточный результат на этапе 2, сохраненный в аккумуляторе, должен быть сложен с содержимым ячейки памяти «Y». Предварительно необходимо загрузить пару регистров «H,L» адресом ячейки Y. Это осуществляется с помощью команды «*LXI* H,d16» – непосредственная загрузка пары регистров «H», «L» указанным в команде двухбайтным операндом «d16». В нашем примере соответствующая строка на Ассемблере запишется следующим образом:

LXI H,0B01*h*

Этап 4. Произведем сложение содержимого «A» с содержимым ячейки памяти, адресуемой парой регистров H,L. Операция сложения выполняется командой

ADD M

Этап 5. Перешлем полученный результат (который пока находится в аккумуляторе) в ячейку памяти «Z» с адресом «0B02*h*». Мнемокод этой команды – «*STA* adr». В нашем случае имеем

STA 0B02h

Для записи всей программы в память микропроцессора, как было отмечено ранее, необходимо перевести полученную программу в машинные коды, где все команды и числа представляются в шестнадцатеричной системе счисления. При этом необходимо помнить о размерах каждой из команд (см. приложение Б), о правилах размещения программ в памяти (изложенных выше), и о машинных кодах каждой из команд (см. приложение В).

Здесь и далее все программы рекомендуется представлять в виде таблицы вида 2.1, при этом условно считается за начало расположения программного кода адрес «0800*h*».

Как видно такая таблица достаточно удобна и информативна – помимо мнемокодов команд на Ассемблере и пояснений она содержит все необходимое для программирования: адрес ячейки памяти и то, что в нее необходимо занести.

Таблица 2.1.
Программа 2.1

№ этапа	Адрес, *h*	Мнемокод	Число байт в команде	Машинный код, *h*	Комментарий
1	0800	*LDA* 0B00*h*	3	3A	Назначение команды *LDA*: содержимое ЯП, адрес которой указан во втором и третьем байтах команды загрузить в регистр «A».
	0801			00	Младший байт адреса.
	0802			0B	Старший байт адреса.
2	0803	*SUI* C8*h*	2	D6	Назначение команды *SUI*: из содержимого «A» отнимается второй байт команды.
	0804			C8	Второй байт команды (операнд)
3	0805	*LXI* H,0B01*h*	3	21	Назначение команды *LXI*: Непосредственная загрузка пары регистров «H»,«L».
	0806			01	Младший байт адреса, который пересылается в регистр «L».
	0807			0B	Старший байт адреса, который пересылается в регистр «H».
4	0808	*ADD* M	1	86	Назначение команды *ADD*: содержимое ЯП, адрес которой записан в регистрах «H»,»L» складывается с содержимым «A»; результат в «A».
5	0809	*STA* 0B02*h*	3	32	Назначение команды *STA*: содержимое «A» пересылается в ЯП, адрес которой указан во втором и третьем байтах команды.
	080A			02	Младший байт адреса.
	080B			0B	Старший байт адреса.
-	080C	*HLT*	1	76	Назначение команды *HLT*: Останов.

5. *Регистр состояний*

Рассмотрим подробнее процесс выполнения команды [7]. Весь процесс разбивается на циклы, обозначаемые M_1, M_2, M_3, M_4, M_5. Выполнение каждой команды занимает от одного до пяти машинных циклов. Машинный цикл требуется всякий раз, когда микропроцессор обращается к памяти или устройствам ввода-вывода (портам).

Если команда занимает несколько байтов, то для выбора каждого байта требуется по одному машинному циклу.

Например, команда «*LDA 0B00h*» (см. выше) занимает три ячейки памяти «0800*h*», «0801*h*», «0802*h*», в которых хранятся коды «3A*h*», «00*h*», «0B*h*». Выполнение команды начинается, когда на программном счетчике имеется число «0800*h*». В первом машинном цикле по адресу «0800*h*» выбирается первый байт команды «3A», передается в микропроцессор, где производится дешифрация этого кода. Во втором машинном цикле из следующей ячейки «0801*h*» выбирается число «00*h*» и записывается в регистр «L». В третьем машинном цикле из ячейки «0802*h*» выбирается число «0B*h*» и записывается в регистр «H». Таким образом, в паре регистров «H», «L» сформировался адрес ЯП, содержимое которой по команде «3A*h*» необходимо передать в аккумулятор. Поэтому в четвертом машинном цикле число из ячейки «0B00*h*» передается в аккумулятор. На этом выполнение команды заканчивается и к содержимому счетчика прибавляется единица, то есть микропроцессор переходит к выполнению следующей команды. Первым машинным циклом при извлечении любой команды является цикл M_1 – выбор первого байта команды. Самые длинные по времени исполнения команды выполняются в пять циклов.

Каждый цикл включает в себя несколько тактов, обозначаемых T_1, T_2, T_3, T_4, T_5. Циклы могут содержать от трех до пяти тактов. Первые три такта во всех циклах используются для организации обмена с памятью и УВВ, такты T_4 и T_5 (если они присутствуют в цикле) – для выполнения внутренних операции и микропроцессоре.

Рассмотрим цикл M_1. В такте T_1 содержимое счетчика команд выдается на шину адреса, адрес принимается памятью, где начинается процесс чтения байта команды из указанной ячейки. В такте T_2 проверяется наличие сигнала (уровня логической единицы) на входе ГОТОВНОСТЬ. Этот сигнал подается на вход микропроцессора через интервал времени, достаточный для завершения процесса чтения из памяти. Если на входе ГОТОВНОСТЬ сигнал отсутствует (действует уровень логического нуля), то микропроцессор устанавливается в режим

ОЖИДАНИЯ до тех пор, пока не появится сигнал на входе ГОТОВНОСТЬ. С приходом этого сигнала микропроцессор выходит из режима ожидания, переходя в такт T_3. в этом такте выданный из памяти байт команды с шины данных принимается в микропроцессор. В такте T_4 анализируется принятый байт команды и выясняется, нужны ли дополнительные обращения в оперативную память. Если такие обращения не требуются (команда однобайтовая и операнды находятся в регистрах), то в этом же такте T_4 либо с использованием дополнительного такта T_5 выполняется предусматриваемая командой операция. Если необходимы дополнительные обращения в оперативную память, то после такта T_4 цикл M_1 завершается и происходит переход к M_2.

Информация о состоянии микропроцессора (о том, какой именно машинный цикл выполняется в данный момент) фиксируется в регистре состояний в начале каждого машинного цикла. В зависимости от состояния данного регистра формируются сигналы, управляющие работой всей системой. В таблице 2.2 приведены возможные состояния микропроцессора.

Таблица 2.2.
Содержание регистра состояний

Состояние МП (машинный цикл)	**Биты регистра состояний**							
	D_7	**D_6**	**D_5**	**D_4**	**D_3**	**D_2**	**D_1**	**D_0**
	MEMR	**INP**	**M1**	**OUT**	**HLTA**	**STACK**	**WO’**	**INTA**
Выбор первого байта команды	1	0	1	0	0	0	1	0
Чтение из памяти	1	0	0	0	0	0	1	0
Запись в память	0	0	0	0	0	0	0	0
Чтение из стека	1	0	0	0	0	1	1	0
Запись в стек	0	0	0	0	0	1	0	0
Ввод в «А» из УВВ	0	1	0	0	0	0	1	0
Вывод из «А» в УВВ	0	0	0	1	0	0	0	0
Прерывание	0	0	1	0	0	0	1	1
Останов	1	0	0	0	1	0	1	0
Прерывание в останове	0	0	1	0	1	0	1	1

Определение битов регистра состояния:

INTA – сигнал подтверждения запроса прерывания;

WO’ – в текущем машинном цикле выполняется запись в память или операция вывода (активный уровень «0»);

STACK – означает наличие на шине адреса содержимого указателя СТЕКа;

HLTA – сигнал подтверждения команды HLT (останов);

OUT – в текущем машинном цикле выполняется операция вывода;

M1 – текущий машинный цикл служит для выборки первого байта команды;

INP – в текущем машинном цикле выполняется операция ввода;

MEMR – в текущем машинном цикле будет производиться чтение памяти.

Порядок работы

Задание 1. Запись в память микропроцессора простейшей программы

Занести программу 2.1 (табл. 2.1) в память микропроцессорной системы.

Записать по адресам «0B00h» и «0B01h» числа по своему варианту, согласно табл. 2.3.

Таблица 2.3.
Варианты чисел для программы 2.1

Адреса ячеек, h	**Значения чисел по вариантам, h**														
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0B00	05	0E	5A	12	B5	B0	25	A0	EE	34	55	D5	12	6F	1D
0B01	06	0F	6A	11	B6	A0	26	99	FF	35	54	D6	13	7F	1E

Задание 2. Исследование результатов выполнения программы в автоматическом режиме

Осуществить запуск программы 2.1 в автоматическом режиме. Считать результат выполнения программы из памяти (ячейка памяти с адресом «0B02h») и занести его в отчет. Сравнить полученный практический результат с расчетами, сделанными вручную (перевести исходные данные в двоичную систему исчисления и выполнить соответствующие преобразования).

Задание 3. Исследование выполнения программы в пошаговом режиме

Очистить содержимое всех регистров системы и содержимое ячейки памяти с адресом «0B02*h*» (остальные ячейки ОЗУ оставить без изменения). Перевести систему в пошаговый режим работы (каждая команда выполняется в несколько этапов). Результат работы эмулятора на каждом шаге выполнения программы использовать для заполнения табл. 2.4. Сверить результаты табл. 2.4 с табл. 2.2. Сделать вывод по полученным результатам.

Примечание: Пошаговый режим может быть или потактовый или поцикловый (выполнение по командам не интересует) в зависимости от возможностей эмулятора.

Таблица 2.4.
Результат исполнения программы 2.1 по тактам

Адрес, *h*	**Мнемокод**	**Число байт в команде**	**Машинный код, *h***	**Число циклов (тактов) в команде**	**Биты регистра состояний**								**Тип машинного цикла (словесное описание)**
					***D*7**	***D*6**	***D*5**	***D*4**	***D*3**	***D*2**	***D*1**	***D*0**	
0800	*LDA* 0B00*h*	3	3A 00 0B	4 цикла									
0803	*SUI* C8*h*	2	D6 C8	2 цикла									
…	…	…	…	…	…								

Примечание: В случае отсутствия возможности анализа битов регистра состояний, соответствующая колонка табл. 2.4 модифицируется, и в ней указываются, например состояния линий шины управления.

Задание 4. Исследование модифицированной программы

Заменить в программе 1 команду «*SUI* d8» на команду по варианту (см. табл. 2.5). Составить для модифицированной программы таблицу, идентичную 2.1, занести ее в отчет. Для модифицированной программы выполнить исследования по заданиям 1 и 2, занести полученные данные в отчет.

Примечание: Следует помнить, что исходная команда в программе 2.1 с мнемокодом «*SUI* C8*h*» занимала в памяти 2 байта. В этой связи если модифицированная команда содержит меньшее количество байт, то исходную программу необходимо модифицировать (либо скорректировать адреса размещения программы, либо воспользовавшись командой «*NOP*» – нет операции).

Таблица 2.5.
Варианты чисел для программы 2.1

Команда, на которую происходит замена «*SUI* d8», по номерам вариантов														
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
ADI 01*h*	*ADI* 1A*h*	*ADI* 03*h*	*ADI* DB*h*	*SBI* DB*h*	*SBI* BB*h*	*SBI* 12*h*	*SBI* 5F*h*	*ACI* 33*h*	*ACI* 81*h*	*ACI* F4*h*	*ACI* 0A*h*	*INR* A	*DCR* A	*DAA*

Контрольные вопросы

1. Поясните принцип работы микропроцессорной системы при выполнении программы пользователя.
2. Для чего предназначен программный счетчик?
3. Какие этапы создания завершенных программ существуют?
4. Что содержится в первом байте любой команды?
5. Сколько всего байт может занимать команда в памяти системы?
6. Что может содержаться во втором и третьем байтах команды?
7. Охарактеризуйте команды пересылки данных (примеры, сколько байтов содержат, специфика выполнения).
8. Охарактеризуйте арифметические команды (примеры, сколько байтов содержат, специфика выполнения).
9. Для чего предназначен регистр состояния системы?
10. Чем отличаются машинные циклы от тактов?

РАБОТА № 3

ОРГАНИЗАЦИЯ ЦИКЛОВ, ОБРАБОТКА МАССИВОВ И РЕАЛИЗАЦИЯ ЛОГИЧЕСКИХ ФУНКЦИЙ

Цель работы.

1. Изучение логических команд, команд переходов и вызовов подпрограмм, команд ввода-вывода и работы со стеком.
2. Рассмотрение алгоритмов решения сложных задач и правил их составления.
3. Исследование программ для обработки массивов чисел, решения алгебраических и логических задач.

Краткие теоретические сведения

1. *Логические команды*

Логические команды выполняют побитные (поразрядные) логические операции над содержимым аккумулятора и операнда, указанного в команде. В качестве операнда в таких командах используется регистр, ячейка памяти или непосредственно заданное в команде число. Отличительной чертой логических команд является отсутствие формирования флагов переносов *AC* и *C* (см. п.2 данной работы). Основными логическими операциями являются: конъюнкция (логическое «И»), дизъюнкция (логическое «ИЛИ»), инверсия (операция «НЕ»), исключающее «ИЛИ». К логическим командам так же иногда относят команды сравнения, сдвигов и работы с битами флагов. Характеристики и описание вышеуказанных команд для микропроцессора КР580 представлено в приложении Б, машинные коды – см. приложение В.

2. *Регистр флагов*

Результат программы можно проанализировать по состоянию регистра флагов «F». Как и все регистры МП КР580, регистр «F» имеет 8 разрядов, однако 3 разряда не используются (в них всегда одно и то же значение, это разряды

D_1, D_3, D_5), а 5 разрядов этого регистра (D_0, D_2, D_4, D_6, D_7) устанавливаются по определенному правилу в соответствии с выполнением последней команды.

Биты признаков результата размещаются в регистре флагов (признаков) следующим образом – см. рис. 3.1.

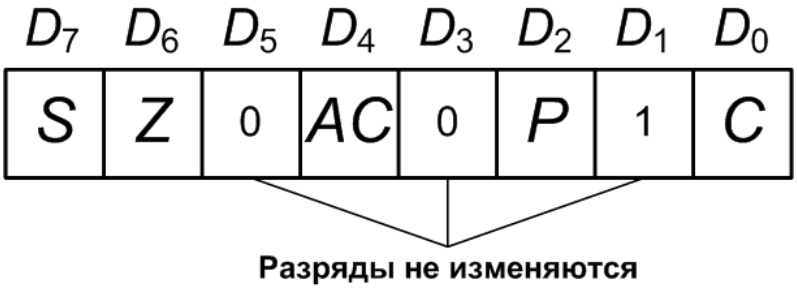


Рис. 3.1. Побитное содержание регистра флагов (признаков).

Флаг знака «S» – SIGN. В него записывается «1», ели при выполнении арифметической или логической команды в старшем, седьмом разряде аккумулятора имеется «1», в противном случае в разряд «S» записывается «0».

Пример: Содержимое регистра «А» после выполнение последней операции равно:

«7F*h*», «0111 1111 *b*» или «45*h*», «0100 0101 *b*» тогда «S» = «0»;

«80*h*», «1000 0000 *b*» или «A5*h*», «1010 0101 *b*» тогда «S» = «1».

Флаг нулевого результата «Z» – ZERO. Если результат равен нулю, в него записывается «1», в противном случае записывается «0».

Пример: Содержимое регистра «А» после выполнение последней операции равно:

«01*h*», «0000 0001 *b*» или «DF*h*», «1101 1111 *b*» тогда «Z» = «0»;

«00*h*», «0000 0000 *b*»» тогда «Z» = «1».

Дополнительный (вспомогательный) флаг переноса AC – AUXILIARY CARRY. В него записывается 1, если при выполнении команд в аккумуляторе возникает единица переноса из третьего разряда числа (перенос между тетрадами байта).

Пример: Содержимое регистра «А» до и после выполнение операции сложения с единицей равно:

до «81*h*», «1000 0001 *b*» после «82*h*», «1000 0010 *b*» тогда «AC» = «0»;

до «AF*h*», «1010 1111 *b*» после «B0*h*», «1011 0000 *b*» тогда «AC» = «1».

Флаг паритета (четности единиц) P – PARITY. В него записывается «1», если при выполнении операции количество единиц в разрядах аккумулятора будет четным, в противном в него записывается «0».

Пример: Содержимое регистра «A» после выполнение последней операции равно:

«1F*h*», «0001 1111 *b*» или «07*h*», «0000 0111 *b*» тогда «P» = «0»;

«81*h*», «1000 0001 *b*» или «C5*h*», «1100 0101 *b*» тогда «P» = «1».

Флаг переполнения C – CARRY. В него записывается 1, если при выполнении последней команды было переполнение аккумулятора (перенос из старшего разряда), в противном случае записывается «0».

Пример: В результате выполнения операции сложения:

«3F*h*» + «0A*h*» = «49*h*» тогда «C» = «0»;

«AA*h*» + «BB*h*» = «65*h*»тогда «C» = «1».

3. *Команды переходов и работы с подпрограммами*

Команды переходов (условных и безусловных), вызова подпрограмм и возвращения из них изменяют нормальную последовательность исполнения команд и предполагают переход по адресу, указанному в них.

Организация условных и безусловных переходов. В системе команд существует группа команд, играющая особую роль в организации выполнения программ. Это команды передачи управления. Пока в программе не встречаются команды этой группы, счетчик команд «PC» постоянно увеличивает свое значение на число равное количеству байт в ткущей команде, и микропроцессор выполняет команду за командой в порядке их расположения в памяти.

Порядок выполнения программы может быть изменен, если занести в регистр счетчика команд «PC» код адреса, отличающийся от адреса очередной команды. Такая передача управления или переход в программе может быть выполнена с помощью трехбайтовой команды безусловного перехода «*JMP* adr».

Безусловную передачу управления можно произвести также по команде «*PCHL*» (которая формально относится к группе команд пересылок), в результате выполнения которой произойдет передача управления по адресу, хранящемуся в регистровой паре «H,L».

Кроме команды безусловного перехода имеется восемь трехбайтовых команд условного перехода. При появлении команды условного перехода, передача управления по адресу, указанному в команде, происходит только в случае выполнения определенного условия. Если условие не удовлетворяется, то выполняется команда, непосредственно следующая за командой условного перехода.

Условия, с которыми оперируют команды условной передачи управления, определяются состоянием регистра флагов «F» (см. п.2).

Связь между состоянием регистра признаков и мнемоникой команды условного перехода:

..*NZ* (NOT ZERO) – ненулевой результат при «Z»=0,

..*Z* (ZERO) – нулевой результат при «Z»=1,

..*NC* (NO CARRI) – отсутствие переноса при «C»=0,

..*C* (CARRY) – перенос при «C»=1,

..*PO* (PARITY ODD) – нечетный паритет при «P=0»,

..*PE* (PARITY EVEN) – четный паритет при «P=1»,

..*P* (PLUS) – число неотрицательное при «S=0»,

..*M* (MINUS) – число отрицательное при «S=1».

Дополнив вышеприведенную мнемонику, частью от команды «*JMP*» – переход, получим соответствующие команды условного перехода: «*JNZ*», «*JZ*», «*JNC*», «*JC*», «*JPO*», «*JPE*», «*JP*», «*JM*».

Команды работы с подпрограммами. Эти команды, так же как и команды переходов изменяют нормальную последовательность выполнения операций в микропроцессоре. Отличие заключается в том, что при вызове подпрограммы (например, с помощью команды «*CALL* adr») используется специальная область памяти – СТЕК (см. ниже, п.4). В СТЕК сохраняется адрес возврата

(адрес участка на котором была вызвана подпрограмма), что позволяет при возвращении из подпрограммы использовать безадресные команды (например, команду «*RET*»). Аналогично, как и команды переходов вызвать подпрограмму и вернуться из нее можно по условию. Мнемонические правила написания соответствующих команд идентичны вышеизложенным правилам.

Характеристики и описание команд переходов, вызова и возврата из подпрограмм для микропроцессора КР580 представлено в приложении Б, машинные коды – см. приложение В.

4. *Команды работы со стеком, ввода-вывода и общие команды*

Команды работы со СТЕКОМ предназначены для работы со специальной областью памяти, сами команды являются однобайтными и имеют мнемонику «*PUSH* …» – загрузить в СТЕК и «*POP* …» – извлечь из стека. Как было отмечено выше (см. лабораторную работу №1) СТЕК представляет собой область памяти со специализированным механизмом обращения (через регистр – указатель «SP»). При каждом обращении к СТЕКУ происходит автоматическое изменение содержимого указателя адреса ячейки в СТЕКЕ «SP»: при загрузке – уменьшается, при считывании – увеличивается.

Команды ввода-вывода предназначены для выполнения операций связанных с портами микропроцессора: команда ввода из порта N с мнемоникой «*IN* N» и команда вывода из порта N с мнемоникой «*OUT* N». Работа с портом происходит через аккумулятор.

Команды управления и общие команды предназначены для организации работы системы. Например, команда рестарта «*RST* n», команды разрешения и запрещения прерываний «*EI*», «*DI*», команды останова «*HLT*».

Более детальную информацию по командам данных групп – см. приложения Б и В.

5. *Алгоритмизация этапов выполнения сложных программ*

Под алгоритмизацией понимается сведение задачи к последовательности этапов, выполняемых друг за другом так, чтобы результаты предыдущих этапов использовались при выполнении следующих. Таким образом, в результате алгоритмизации должен быть получен алгоритм решения задачи.

Алгоритм это точное и понятное описание последовательности действий над заданными объектами (исходными данными), приводящее исполнителя после конечного числа шагов к достижению указанной цели (получение результата) или решению поставленной задачи [8]. Алгоритм обладает следующими основными свойствами: дискретностью, определенность, результативностью и массовостью.

Дискретность: в алгоритме процесс преобразования исходных данных в результат осуществляется дискретно, по шагам. Переход к следующему шагу возможен лишь после завершения предыдущего.

Определенность: алгоритм должен быть четким и однозначным; значение величин получаемых в какой-либо момент времени однозначно определяются величинами, полученными в предыдущие моменты времени; в рамках алгоритма четко определяется какое действие должно быть выполнено следующим.

Массовость: алгоритм решения задачи разрабатывается в общем виде так, что бы его можно было применить для целого класса задач, различающимися исходными данными.

Результативность: алгоритм должен приводить к решению задачи за конечное число шагов; под решением так же понимается наличие сообщения о том, что при заданных данных задача решения не имеет или алгоритм не приемлем.

Основные этапы выполнения алгоритма

Ключевым элементом любого алгоритма является «действие», которое совершается на определенном этапе. Одним из вариантов представления алгоритмов является схема. Условные обозначения и правила выполнения схем алгоритмов, программ данных и систем можно найти в [9], здесь же приведем ос-

новные «действия» и их обозначения имеющие место при решении сложных задач.

К базовым этапам, определяющим элементарные действия, обычно относят: начало, ввода и вывод данных, обработка значений, проверка условия и перехода, окончания или выхода.

1. Этап начала алгоритма, или терминатор отображает вход из внешней среды (начало схемы программы):

НАЧАЛО (ВХОД)

2. Ввод исходных данных, вывод информации (указатель на данные, носитель которых не определен) отображает использование, источник и приемник данных, вспомогательную указательную информацию:

ДАННЫЕ

3. Этап обработки вычисления или процесса, который следует выполнить над данными:

V = *ВЫРАЖЕНИЕ*

где V – переменная; *ВЫРАЖЕНИЕ* – любое вычисление, которое надо осуществить.

4. Этап проверки условия (решение) это функция переключательного типа имеющая один вход и ряд альтернативных выходов, только один из которых может быть активирован после проверки условий:

УСЛОВИЕ?

Если условие выполняется, то осуществляется переход к этапу с номером (меткой) «N»; если условие не выполняется, то переход осуществляется к следующему по порядку этапу.

Переход к этапу с номером N:

ИДТИ К «N»

5. Этап окончания: отображает конец выполнения или выход во внешнюю среду:

ОСТАНОВ (ВЫХОД)

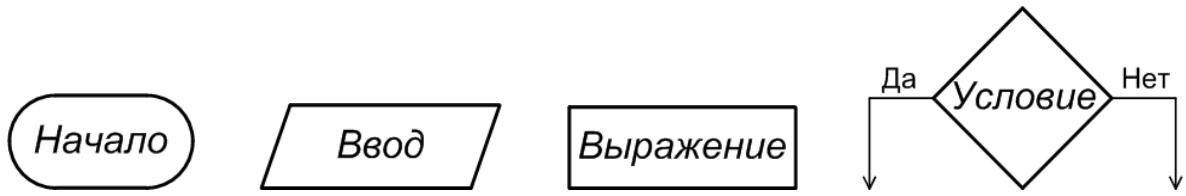
В схеме каждое из «действий» имеет свое условное изображение (см. рис. 3.2), соединяется с другими с помощью линий (показывающих направление движения по алгоритму) и в случае необходимости сопровождается комментариями.

Начало и окончание алгоритма отображается, как показано на рис. 3.2,а, в виде овала внутри которого указывается «начало» (вход) или «останов» (выход).

Ввод или вывод данных, источник или их приемник изображается с помощью параллелограмма (см. рис. 3.2,б) внутри которого указывается «ввод» или «вывод», «печать» и перечисляются переменные.

Этап обработки вычисления обозначаются прямоугольником (см. рис. 3.2,в), внутри которого записывается содержание действия (описывается выражение).

Проверка условия изображается ромбом, внутри которого записывается формулировка. В результате проверки выбирается один из двух возможных путей вычислительного процесса (рис. 3.2,г): если условие выполняется, т.е. имеет значение «ДА», то следующим выполняется переход по соответствующему направлению (аналогично для перехода по ветви «НЕТ»).



а) б) в) г)

Рис. 3.2. Условное обозначение этапов алгоритма.

Составление алгоритмов сложных задач

При разработке алгоритмов решения задач чаще всего придерживаются так называемого структурного подхода [8]: используют метод последовательного уточнения алгоритма, применяют вспомогательные алгоритмы, описывают алгоритмы с помощью трех основных структур: следования, ветвления, повторения.

Схема следования применяется при выполнении следующих друг за другом выражений – см. рис. 3.3,а. Схема ветвления применяется, когда в зависимости от условия нужно выполнять либо одно, либо другое действие (выражение). Простое разветвление показано на рис. 3.3,б; частный случай разветвления, называемый иногда обходом показан на рис. 3.3,в. В обоих случаях направления движения по алгоритму «Да» и «Нет » показаны условно (может быть и наоборот).

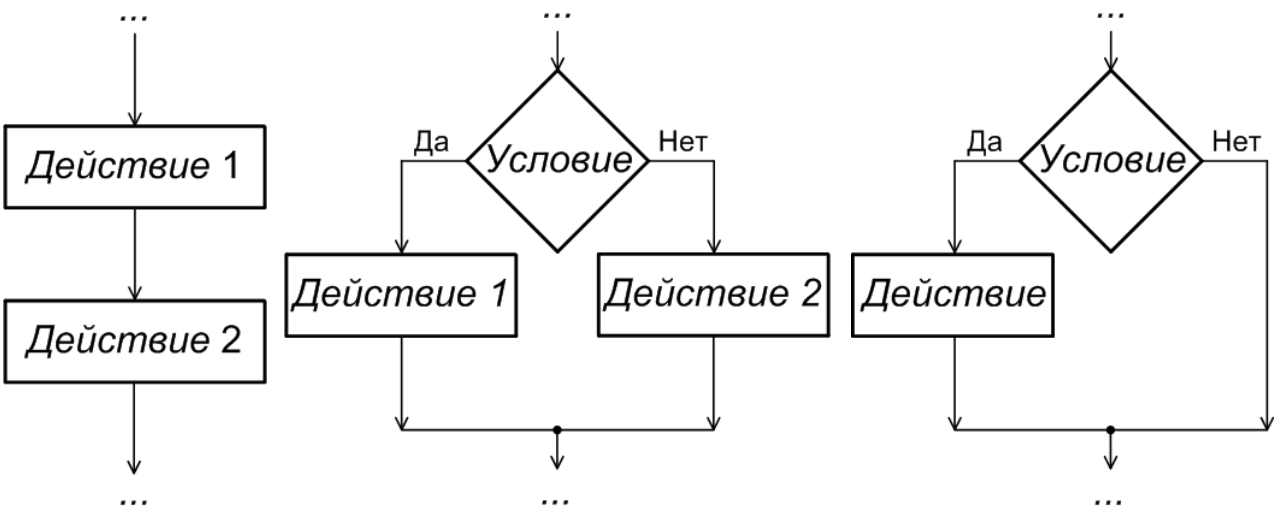
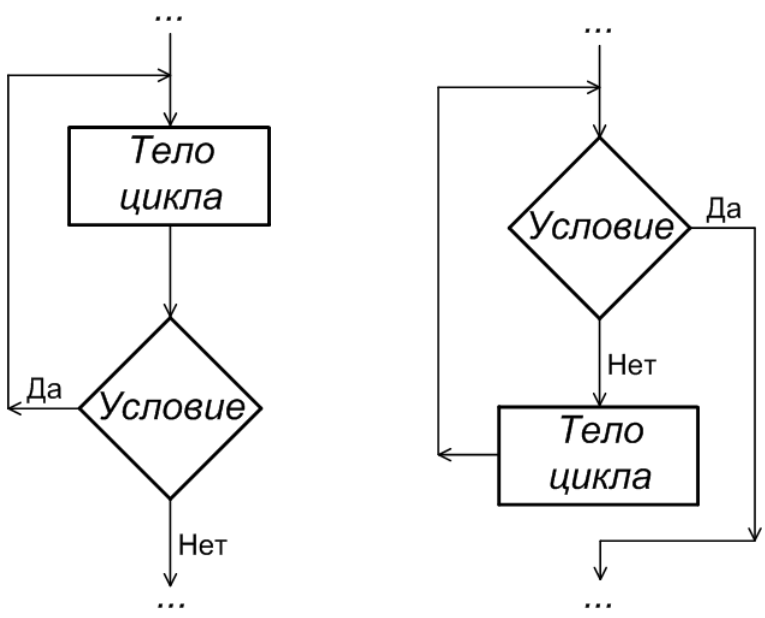


Рис. 3.3. Схемы следования и ветвления.

Примечание: При программировании задач на ассемблере для организации «условий» используют команды условного перехода (см. п.3).

Схема повторения является ключевой при организации циклически повторяющихся участков алгоритма. Такие схемы называют циклами, и выделяют отдельно так называемые циклы «До» и циклы «Пока». Разница между этими двумя схемам заключается в месте расположения условия: в цикле «До» (выполнять до условия) условие располагается в конце цикла (см. рис. 3.4,а), в цикле «Пока» условие располагается в начале цикла (см. рис. 3.4, б).

Объединяя вышеприведенные структуры, следуя принципу уточнения, можно получить решение сколь угодно сложной задачи, через проведение элементарных действий.



а) б)

Рис. 3.4. Схемы повторения.

Порядок работы

Задание 1. Исследование программы обработки массива

Пусть требуется найти сумму элементов массива (числа без знака), расположенного начиная с адреса «0B02h». Количество обрабатываемых элементов массива содержится по адресу «0B01h». Результат обработки необходимо поместить в ячейку памяти с адресом «0B00h».

Общий алгоритм решения такой задачи может быть представлен в виде рис. 3.5. Сначала требуется выполнить «начальные преобразования»: очистить регистры, использование которых предполагается (по необходимости; при этом желательно использовать команды, выполняющиеся за меньшее количество тактов); установить начальные адреса ячеек памяти, считать первую ячейку (количество элементов). В самом теле цикла требуется: увеличить адрес на единицу (адресовать следующую ячейку памяти и перейти к очередному обрабатываемому элементу); сложить содержимое одного из регистров (отвечающего за накопление суммы) с очередным элементом массива; уменьшить количество обрабатываемых элементов на единицу.

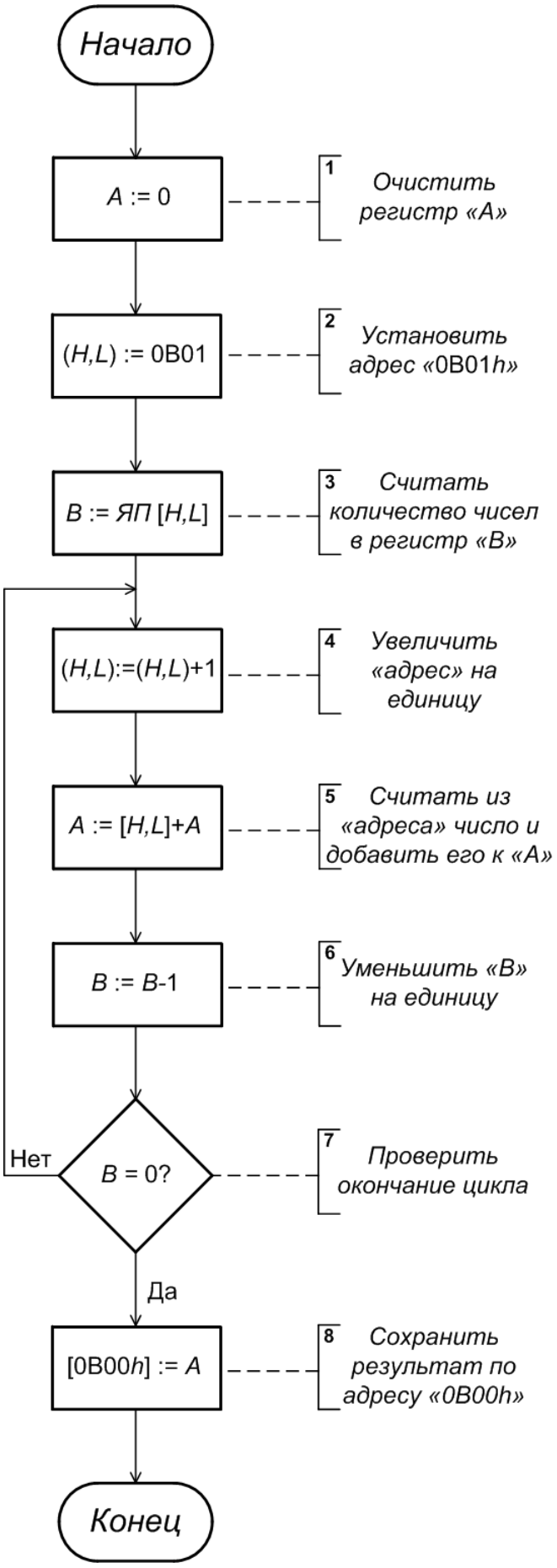


Рис. 3.5. Алгоритм программы 3.1.

В конце цикла необходимо проверить условие окончания: все ли элементы обработаны (равен ли счетчик элементов нулю), если условие не выполнено перейти в начало цикла. Если условие выполняется необходимо осуществить завершающие действия: сохранить общую сумму по требуемому адресу.

Таким образом, в теле цикла повторяются одни и те же действия: переход к очередному элементу (ячейке памяти) добавление очередного элемента к общей сумме, модификация и проверка условия окончания обработки. Заметим что до «первого прохода цикла» содержимое регистра отвечающего за общую сумму должно быть обнулено, т.к. к нему в теле цикла добавляется очередное число. Т.е. при «первом проходе» в регистре содержится только первый элемент (сумма нуля и очередного считанного, который сейчас является первым), затем к этому регистру нему на «втором проходе» добавляется второй элемент (ранее полученная сумма и очередной считанный элемент, который сейчас будет вторым), и т.д.

Программа, составленная по данному алгоритму, представлена в виде таблицы 3.1.

Таблица 3.1.
Программа 3.1

№ этапа	Комментарий по этапу алгоритма	Метка	Мнемокод	Комментарий по команде
1	Очистить регистр «А»		`SUB A`	Назначение команды *SUB*: Содержимое регистра «А» вычитается из «А», результат помещается в «А».
2	Установить адрес «0B01*h*»		`LXI H,0B01h`	Назначение команды *LXI*: Непосредственная загрузка пары регистров «H»,«L».
3	Считать количество чисел в регистр «В»		`MOV B,M`	Назначение команды *MOV*: Содержимое ЯП, адрес которой указан в регистрах «H,L» пересылается в «В».
4	Увеличить «адрес» на единицу	Mk1:	`INX H`	Назначение команды *INX*: Содержимое пары регистров «H,L» увеличивается на единицу.
5	Считать из «адреса» число и добавить его к «А»		`ADD M`	Назначение команды *ADD*: Содержимое ЯП, адрес которой указан в «H,L», складывается с содержимым «А». Результат помещается в «А».
6	Уменьшить «В» на единицу		`DCR B`	Назначение команды *DCR*: Содержимое регистра «В» уменьшается на единицу.
7	Проверить окончание цикла		`JNZ Mk1`	Назначение команды *JNZ*: Если последний результат не равен «0», то переход по метке Mk1, представляющей собой явно заданный адрес участка программы.
8	Сохранить результат по адресу «0B00*h*»		`STA 0B00h`	Назначение команды *STA*: Содержимое «А» пересылается в ЯП по указанному в команде адресу.
-			`HLT`	Назначение команды *HLT*: Останов.

<u>В рамках выполнения данного задания требуется:</u>

а) составить программу по нахождению суммы элементов массива (аналогично табл. 3.1), учитывая изменения для каждого варианта заданные в виде табл. 3.2;

Таблица 3.2.
Варианты заданий для программы 3.1

Назначение адреса памяти	**Значения адресов ячеек памяти по вариантам, *h***														
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Количество обрабатываемых чисел	0B11	0B21	0B31	0B41	0B51	0B61	0B71	0B81	0B91	0BA1	0BB1	0BC1	0BD1	0BE1	0BF1
Начало расположения массива чисел	0B12	0B22	0B32	0B42	0B52	0B62	0B72	0B82	0B92	0BA2	0BB2	0BC2	0BD2	0BE2	0BF2
Место-расположение результата	0B10	0B20	0B30	0B40	0B50	0B60	0B70	0B80	0B90	0BA0	0BB0	0BC0	0BD0	0BE0	0BF0

б) подготовить программу к записи в эмулятор, переведя ее в машинный код (при этом необходимо составить таблицу, идентичную 2.1, которую так же представить в отчете по работе);

в) занести созданную программу в ОЗУ эмулятора, исследовать ее (при этом элементы массива выбрать по своему усмотрению, но их количество должно быть не менее 41), проанализировать результат, сделать вывод о работе, отдельно рассмотреть вопрос функционирования системы и изменение содержимого регистра состояния (флагов) составив и заполнив таблицу 3.3 для командного режима работы эмулятора:

Таблица 3.3.
Результат исполнения программы 3.1 по командам

Адрес, *h*	**Мнемокод**	**Число байт в команде**	**Машинный код, *h***	**Биты регистра флагов**								**Пояснение по факту изменения битов регистра флагов, и использование этого в программе**
				***D*7**	***D*6**	***D*5**	***D*4**	***D*3**	***D*2**	***D*1**	***D*0**	
				S	***Z***	**0**	***AC***	**0**	***P***	**1**	**C**	
0800	*SUB* A	1	97									
…	…	…	…	…								

В таблице 3.3 допустимо пропускать повторяющиеся элементы (при прохождении цикла).

Задание 2. Исследование программы нахождения максимума

Пусть требуется найти максимальный элемент среди массива чисел, расположенных начиная с адреса «0B02h». Количество обрабатываемых элементов массива содержится по адресу «0B01h». Результат обработки необходимо поместить в ячейку памяти с адресом «0B00h». Считается что элементы массива – числа без знака (от «0d» до «127d»), размером каждое восемь бит.

Алгоритм решения задачи схож с представленным на рис. 3.5. Первоначально производится загрузка регистровой пары «H», «L» адресом «0B01h» указывающим на число элементов обрабатываемого массива. Затем производится считывание этого числа и загрузка его в регистр «B», который будет служить счетчиком числа обработанных чисел. Блок обработки начинается с увеличения адреса и чтения первого значащего элемента массива, который помещается в регистр «A». Предполагается, что это число максимальное. Затем это число сравнивается со вторым числом, выбранным из памяти (предварительно происходит увеличение адреса) с помощью команды сравнения «*CMP* M» и устанавливаются флаги в регистре состояний. Отсутствие переноса указывает на то, что первое число больше второго (т.е. максимальное число пока еще в регистре «A»), и далее просто необходимо осуществить переход к следующему обрабатываемому числу. Если же это условие не выполняется, то до перехода к обработке очередного элемента необходимо осуществить замену максимального числа (т.е. теперь это то, которое было считано, а не хранилось в «A»). Цикл, как и ранее, организован по факту проверки счетчика (регистра «B»), который после очередного сравнения уменьшается на единицу, пока не станет нулевым. После обработки всех числе максимальное число, сохраненное в «A», загружается по адресу «0B01h».

Программа, составленная по данному алгоритму – см. рисунок 3.6, представлена в виде таблицы 3.4.

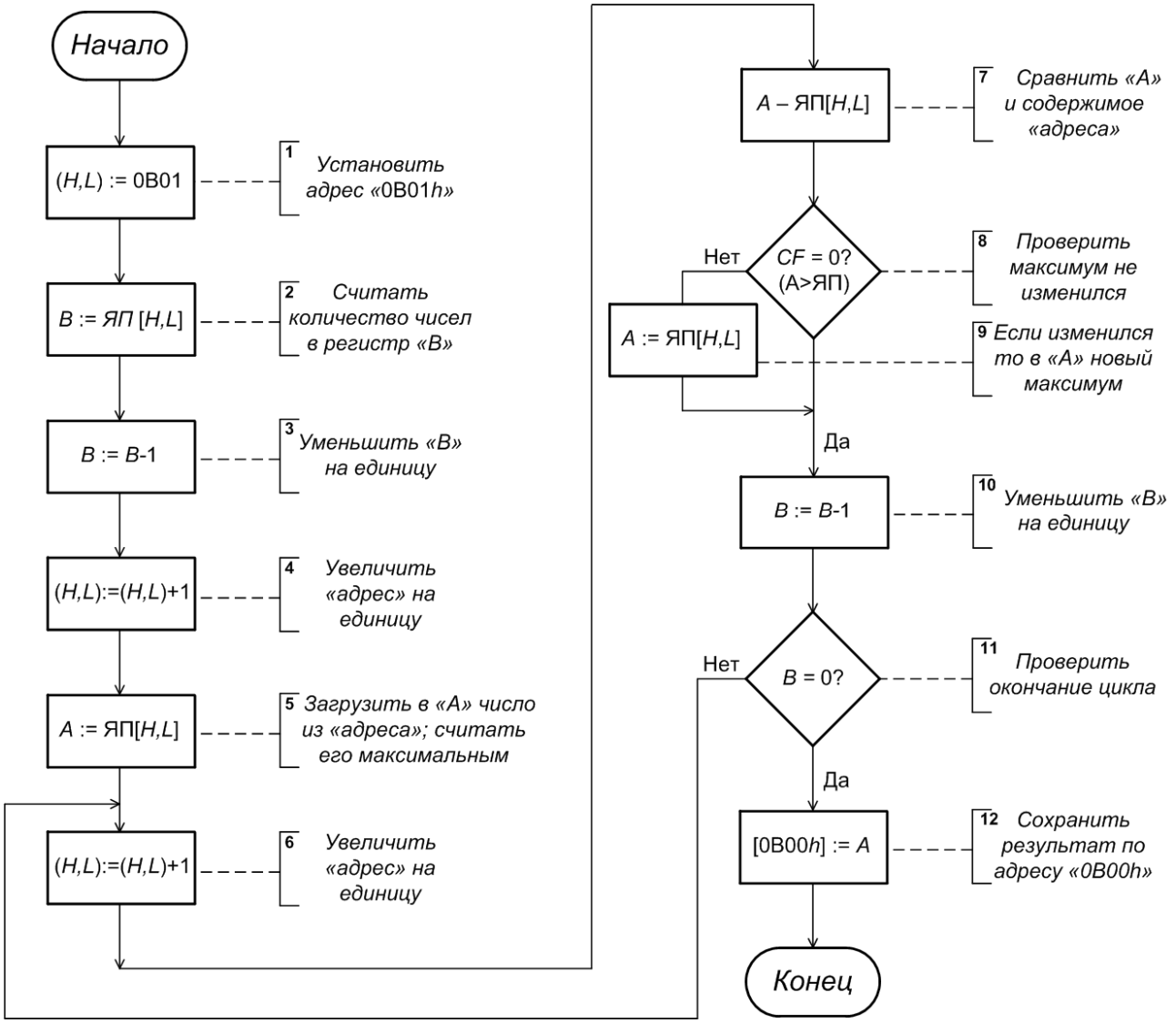


Рис. 3.6. Алгоритм программы 3.2.

В рамках выполнения данного задания требуется: выполнить пункты исследования, идентичные заданию 1.

Задание 3. Создание программы нахождения минимума

Требуется: пользуясь составленной программой в задании 2 составить программу по нахождению минимума. Считается что элементы массива – числа без знака (от «0d»до «127d»), размером каждое восемь бит.

С составленной программой провести исследования, идентичные представленным в задании 1 и 2.

Таблица 3.4.
Программа 3.2

№ этапа	Комментарий по этапу алгоритма	Метка	Мнемокод	Комментарий по команде
1	Установить адрес «0B01*h*»		*LXI* H,0B01*h*	Назначение команды *LXI*: Непосредственная загрузка пары регистров «H»,«L».
2	Считать количество чисел в регистр «B»		*MOV* B,M	Назначение команды *MOV*: Содержимое ЯП, адрес которой указан в регистрах «H,L» пересылается в «B».
3	Уменьшить «B» на единицу		*DCR* B	Назначение команды *DCR*: Содержимое регистра «B» уменьшается на единицу.
4	Увеличить «адрес» на единицу		*INX* H	Назначение команды *INX*: Содержимое пары регистров «H,L» увеличивается на единицу.
5	Загрузить в «A» число из «адреса»; считать его максим-м		*MOV* A,M	Назначение команды *MOV*: Содержимое ЯП, с адресом в «H,L» пересылается в «A».
6	Увеличить «адрес» на единицу	Mk1:	*INX* H	Назначение команды *INX*: Содержимое пары регистров «H,L» увеличивается на единицу.
7	Сравнить «A» и содержимое «адреса»		*CMP* M	Назначение команды *CMP*: Содержимое ЯП, адрес которой указан в «H», «L» вычитается из содержимого «A». При этом содержимое A не изменяется. По результатам сравнения устанавливаются флаги: C=1, если A<ЯП[H,L].
8	Проверить максимум не изменился		*JNC* Mk2	Назначение команды *JNC*: Если флаг C не равен «1», то переход по метке Mk2, представляющей собой явно заданный адрес.
9	Загрузить в «A» число из «адреса»; считать его максим-м		*MOV* A,M	Назначение команды *MOV*: Содержимое ЯП, с адресом в «H,L» пересылается в «A».
10	Уменьшить «B» на единицу	Mk2:	*DCR* B	Назначение команды *DCR*: Содержимое регистра «B» уменьшается на единицу.
11	Проверить окончание цикла		*JNZ* Mk1	Назначение команды *JNZ*: Если последний результат не равен «0», то переход по метке Mk1, представляющей собой явно заданный адрес участка программы.
12	Сохранить результат по адресу «0B00*h*»		*STA* 0B00*h*	Назначение команды *STA*: Содержимое «A» пересылается в ЯП по указанному адресу.
-			*HLT*	Назначение команды *HLT*: останов.

Задание 4. Исследование программы реализации логических функций

Логические команды и команды переходов микропроцессора КР580 позволяют реализовать любую логическую функцию, а вследствие этого, по сути, имеется возможность задать работу произвольной комбинационной схемы или цифрового автомата с памятью.

Рассмотрим задачу реализации логической функции $Y(X)$. При программировании такой процедуры необходимо помнить, что микропроцессор обрабатывает данные последовательно, только для любых двух операндов и лучше всего определяется факт равенства нулю.

Например, требуется реализовать логическую функцию, содержащую восемь аргументов. Хорошо известно, что для любой функции можно записать уравнение в виде СДНФ (совершенно дизъюнктивно нормальная форма). Пусть для примера это уравнение задано следующим:

$$Y(X) = X_1\ X_3\ X_5\ X_7 + X_2'\ X_3\ \ X_4\ \ X_8' = Y_1(X) + Y_2(X),$$

где $Y_1(X) = X_1\ X_3\ X_5\ X_7$; $Y_2(X) = X_2'\ X_3\ \ X_4\ \ X_8'$; верхний апостроф ($'$) указывает на операцию инвертирования.

Будем считать, что аргументы (например, результаты опроса 8-ми двоичных датчиков) предварительно сохранены в виде двоичного числа в ячейке памяти, с символическим наименованием ARGUM. Т.е. самый младший бит этого двоичного числа есть X_1, более старший X_2, и так далее до восьмого, самого старшего бита, который есть X_8.

Представим каждый из членов $Y_1(X)$ и $Y_2(X)$ полной конъюнкцией, при этом, не изменяя их значения, расставив так же аргументы в соответствии с их положением в байте ARGUM:

$$YY_1(X) = Y_1(X) = 1\ X_7\ 1\ X_5\ 1\ X_3\ 1\ X_1,\ \ YY_2(X) = Y_2(X) = X_8'\ 1\ 1\ 1\ X_4\ X_3\ X_2'\ 1.$$

Запишем эквивалентное преобразование

$$Y_1(X) = Y_1(X)'' = Z_1(X)',\ \ Y_2(X) = Y_2(X)'' = Z_2(X)',$$

где $Z_1(X) = Y_1(X)'$, $Z_2(X) = Y_2(X)'$.

Таким образом, сама функция $Y(X) = 1$ тогда, когда

- либо $YY_1(X) = Y_1(X) = 1$, т.е. $Z_1(X) = 0$,

либо $YY_2(X) = Y_2(X) = 1$, т.е. $Z_2(X) = 0$.

Определим когда $Z_1(X) = 0$. Представим его через $YY_1(X)$:

$$Z_1(X) = 0+X_7'+0+X_5'+0+X_3'+0+X_1'.$$

Теперь становится очевидным, что считанное значение ARGUM (которое надо проверить на равенство нулю для каждого случая):

а) необходимо сначала логически умножить (операция «И») на двоичное число, в котором «1» стоят на местах присутствия аргументов, а «0» на всех других (пусть это число *gamma*1 = «0;1;0;1;0;1;0;1»); в итоге получим преобразованное ARGUM_11, для компоненты $YY_1(X)$:

$$\text{ARGUM\_11} = \text{ARGUM} \wedge \text{«0;1;0;1;0;1;0;1»} = \text{«0;}X_7\text{;0;}X_5\text{;0;}X_3\text{;0;}X_1\text{»}.$$

б) затем для проверки всего полученного выражения ARGUM_1 на равенство нулю провести с ним операцию «Исключающее ИЛИ», которая даст результат равный нулю (все биты равны нулю) если число целиком совпадает с маской в которой «1» задается на всех местах, где соответствующая переменная в исходном выражении представлена без инверсии (в выражении для Z она соответственно с инверсией); для всех других случаев ставится «0» (пусть это число *sigma*1, для нашего случая равное «0;1;0;1;0;1;0;1»):

$$\text{ARGUM\_12} = \text{ARGUM\_11} \oplus \text{«0;1;0;1;0;1;0;1»},$$

при этом если получится что ARGUM_12 = «0», то исходное выражение будет равно «1» и можно остановить проверку.

Для второй компоненты $YY_2(X)$ имеем аналогичную последовательность действий:

$$\text{ARGUM\_21} = \text{ARGUM} \wedge \text{«1;0;0;0;1;1;1;0»} = \text{« }X_8\text{;0;0;0;}X_4\text{;}X_3\text{;}X_2\text{;0»},$$

$$\text{ARGUM\_22} = \text{ARGUM\_12} \oplus \text{«0;0;0;0;1;1;0;0»}.$$

Программа, составленная для данного случая, представлена в виде таблицы 3.5, считается что ARGUM = «0B00*h*», а результат необходимо сохранить в ячейке памяти с адресом «0C00*h*».

<u>В рамках выполнения данного задания требуется:</u> составить программу (аналогично решению табл. 3.5) по вычислению логической функции, заданной в таблице 3.6; перевести программу в машинный код (аналогично табл. 2.1), за-

писать ее эмулятор и проверить правильность работы; сделать соответствующие выводы.

Таблица 3.5.
Программа 3.3

№ этапа	Комментарий по этапу алгоритма	Метка	Мнемокод	Комментарий по команде
1	Запись в аккумулятор содержимого ячейки ARGUM		*LDA* 0B01*h*	Назначение команды *LDA*: Содержимое ЯП, адрес которой указан в команде пересылается в А.
2	Дублирование считанного числа в регистре «С»		*MOV* C,A	Назначение команды *MOV*: Содержимое регистра «А» пересылается в регистр «С».
3	Нахождение ARGUM_11, т.е. уничтожение отсутствующих в $Y_1(X)$ аргументов с помощью *gamma*1 = «01010101*b*» = «55*h*»		*ANI* 55*h*	Назначение команды *ANI*: Над содержимым «А» и второго байта выполняется логическая операция «И», результат помещается в регистр «А».
4	Нахождение ARGUM_12, т.е. определение факта равенства 0 при совпадении считанного числа с имеющимися аргументами в $Y_1(X)$ с помощью *sigma*1 = «01010101*b*» = «55*h*»		*XRI* 55*h*	Назначение команды *XRI*: Над содержимым «А» и второго байта выполняется логическая операция «Исключающее ИЛИ», результат помещается в регистр «А».
5	Переход, если «А»=0, т. е. $Y_1(X) = 1$, а значит и всегда будет $Y(X) = 1$.		*JZ* Mk1	Назначение команды *JZ*: Если последний результат равен «0», то переход по метке Mk1, представляющей собой явно заданный адрес участка программы.
6	Восстановление считанного числа в «А»		*MOV* A,C	Назначение команды *MOV*: Содержимое регистра «С» пересылается в регистр «А».
7	Нахождение ARGUM_21, т.е. уничтожение отсутствующих в $Y_2(X)$ аргументов с помощью *gamma*2 = «10001110*b*» = «8E*h*»		*ANI* 8E*h*	Назначение команды *ANI*: Над содержимым «А» и второго байта выполняется логическая операция «И», результат помещается в регистр «А».

Продолжение 3.5.
Программа 3.3

№ этапа	Комментарий по этапу алгоритма	Метка	Мнемокод	Комментарий по команде
8	Нахождение ARGUM_22, т.е. определение факта равенства 0 при совпадении считанного числа с имеющимися аргументами в $Y_2(X)$ с помощью *sigma*2 = «00001100*b*» = «0C*h*»		`XRI 0Ch`	Назначение команды *XRI*: Над содержимым «А» и второго байта выполняется логическая операция «Исключающее ИЛИ», результат помещается в регистр «А».
9	Переход, если «А»=0, т. е. $Y_2(X) = 1$, а значит и всегда будет $Y(X) = 1$.		`JZ Mk1`	Назначение команды *JZ*: Если последний результат равен «0», то переход по метке Mk1, представляющей собой явно заданный адрес участка программы.
10	Если ни один из членов не равен 1, то $Y(X) = 0$.		`MVI A,00h`	Назначение команды *MVI*: Второй байт команды пересылается в регистр «А».
11	Переход к выходу результата.		`JMP Mk2`	Назначение команды *JMP*: Безусловный переход по метке Mk2, представляющей собой явно заданный адрес участка программы.
12	Если хотя бы один из членов равен 1, то $Y(X) = 1$.	Mk1:	`MVI A,01h`	Назначение команды *MVI*: Второй байт команды пересылается в регистр «А».
13	Сохранить результат по адресу «0C00*h*»	Mk2:	`STA 0C00h`	Назначение команды *STA*: Содержимое «А» пересылается в ЯП по указанному в адресу.
14	-		*HLT*	Назначение команды *HLT*: останов.

Таблица 3.5.
Варианты заданий

Номер варианта	Логическая функция по вариантам
1	$Y(X) = X_1' X_3' X_5' X_7' + X_2 X_3 X_4 X_8'$
2	$Y(X) = X_1' X_4 X_7' + X_3' X_4' X_6'$
3	$Y(X) = X_7' X_6' X_5' X_1 + X_2 X_1 X_4' X_8'$
4	$Y(X) = X_2 X_3 ' X_5 X_4 + X_2' X_4 X_8$
5	$Y(X) = X_1 X_2 X_3 X_4 + X_5' X_6 X_7 X_8'$
6	$Y(X) = X_1' X_2' X_3' X_4 ' + X_2 X_3 X_4 X_8$
7	$Y(X) = X_7 X_2 X_4 + X_2 X_6' X_1 X_4'$
8	$Y(X) = X_3 X_4' X_7' X_8 + X_1' X_2' X_3'$
9	$Y(X) = X_7' X_5' X_1 X_2 + X_8' X_1 X_5' X_7'$
10	$Y(X) = X_1 X_7' X_6 X_5' + X_1 X_2 X_4 X_5$
11	$Y(X) = X_4' X_6' X_1 X_5 + X_2 X_3' X_1' X_7'$
12	$Y(X) = X_4' X_3 X_5 X_6' + X_1 X_2' X_3' X_4$
13	$Y(X) = X_7' X_1 X_2 X_6 + X_1 X_2' X_7 X_8$
14	$Y(X) = X_4 X_6' X_3 X_8' + X_2' X_1' X_3 X_5'$
15	$Y(X) = X_1' X_3 ' X_7 X_8 + X_1' X_2' X_3' X_6$

Контрольные вопросы

1. Какие флаги не изменяют логические команды?

2. Какие группы команд условно относят к логическим?

3. Приведите побитное содержание регистра признаков микропроцессора.

4. Каким образом используются флаги (признаки)?

5. Какие команды переходов Вы знаете?

5. Сколько байт содержится в командах переходов и работы с подпрограммами?

6. Каким образом происходит возвращение из подпрограмм?

7. Как может использоваться СТЕК в микропроцессорных системах?

8. Дайте определение алгоритму.

9. Какими свойствами обладает алгоритм?

10. Покажите основные этапы алгоритмов в виде схем и примеров на ассемблере.

РАБОТА № 4

РЕАЛИЗАЦИЯ УПРАВЛЯЮЩИХ ВОЗДЕЙСТВИЙ И ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕДУР

Цель работы.

1. Изучение принципов опроса внешних устройств и способов организации работы микропроцессора в режиме ожидания события.
2. Рассмотрение принципов формирования управляющих сигналов и организации временной задержки.
3. Изучение программной реализации типовых вычислительных процедур.

Краткие теоретические сведения

При разработке программного обеспечения для микропроцессорных систем, использующихся для управления различными механизмами и машинами, а так же предназначенных для реализации на их основе встраиваемых систем (для регулирования параметров технологических процессов и производств) возникает необходимость программирования таких типовых задач как: опрос состояния двоичного датчика, ожидание события, формирование выходных управляющих сигналов, формирование временных задержек и т.п. В данной работе рассмотрим некоторые способы программной реализации типовых задач управления на базе однокристального восьмиразрядного микропроцессора КР580, а так же методику программирования вычислительных процедур, разработка которых проблематична из-за ограничений, связанных с особенностями организации самого микропроцессора.

Микропроцессорные системы управления технологическими объектами должны учитывать события, проходящие в самом объекте, реагировать на них и вырабатывать управляющие воздействия в так называемом реальном масштабе времени. Для начала рассмотрим вопрос организации взаимодействия системы с внешними устройствами.

1. *Опрос двоичного датчика*

Взаимодействие микропроцессора с другими элементами системы, а так же с внешними устройствами, осуществляется с помощью специализированных микросхем, например: КР580ВВ55, КР580ВН59, КР580ВТ57, КР580ВИ53, КР580ВВ51, КР580ВН28 и т.д.

Некоторые схемотехнические аспекты организации такого взаимодействия рассмотрены во второй части пособия «Микропроцессорные системы», здесь же большее внимание уделим программным аспектам. Будем считать, что имеет место простейший случай [3] подключения к микропроцессору внешнего устройства, например с помощью многорежимного буферного регистра (МБР) К589ИР12. Упрощенная схема подключения представлена на рис. 4.1 для случая организации режима ввода.

Здесь на рис. 4.1. имеют место следующие обозначения: «CPU» – микропроцессор КР580; «D1» – многорежимный буферный регистр К589ИР12; «D2.1», «D2.2», …, «D2.8» – микросхемы, предназначенные для усиления сигнала и гальванической развязки; «R1», «R2», …, «R16» – сопротивления; «S1», «S2», …, «S8» – ключи; «VD1», «VD2», …, «VD8» – светодиоды; «ША», «ШД» – шины адреса и данных; «На ШУ» – показывает направление передачи сигнала на шину управления системы; «В Сх.ДщА» – показывает направление передачи сигнала на дешифратор адреса; «CS1», «CS2» – выходы выбора кристалла; «INT» – выход запроса прерывания; «R» – вход для установки нуля; «STB» – вход для стробирующего сигнала; «MD» – вход выбора режима; дополнительно на схеме так же показаны лини питания «+5В» и общая точка.

В схеме ключи имитируют поведение дискретного контактного датчика типа «сухой контакт». Если соответствующий ключ разомкнут, то на соответствующем входе D присутствует высокий уровень напряжения (логическая единица), если соответствующий ключ замкнут – то на одноименном входе логический ноль. Светодиоды в схеме предназначены для индикации имеющегося на порту числа.

Будем считать что регистр D1 подключен к порту «01» процессора.

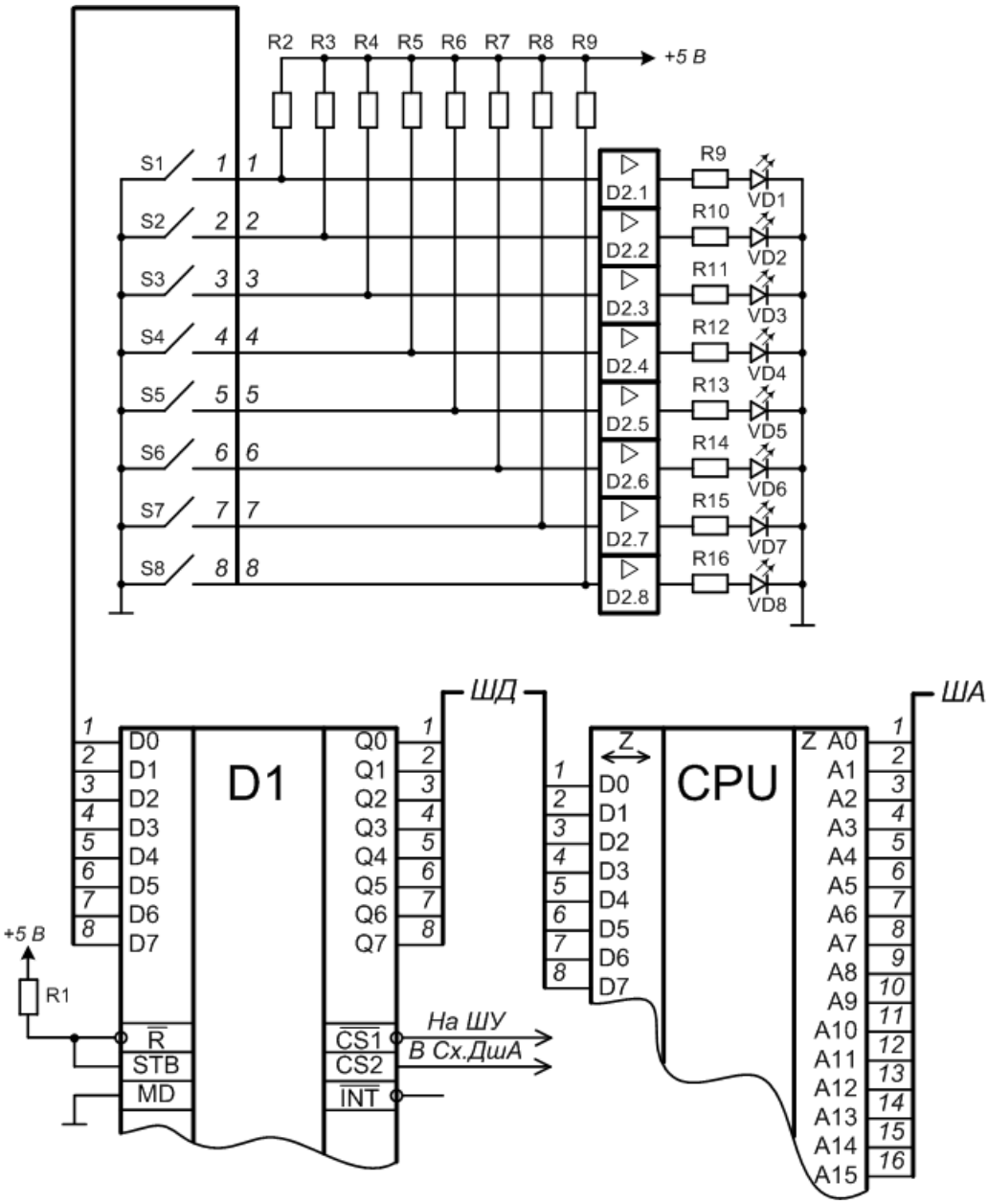


Рис. 4.1. Схема организации ввода с помощью К589ИР12.

Пусть, например, требуется в некоторой части управляющей программы опросить сигнал от четвертого датчика (ключ «S4», линия порта «D3») и в зависимости от результата опроса передать управление фрагменту программы с меткой LABEL_N если датчик не сработал (т.е. D3 = 1) или по адресу отмеченному меткой LABEL_Y если датчик сработал (т.е. D3 = 0).

На рис. 4.2 приводится алгоритм программы 4.1, реализующий описанную выше процедуру опроса. Программа имеет символическое имя INPKEY

(ввод ключа), которое используется в качестве метки первой команды этой программы.

Большая часть этапов алгоритма (рис. 4.2) уже встречалась ранее и их программная реализация не составит труда. Остановимся более подробно на процедуре выделения определенного бита.

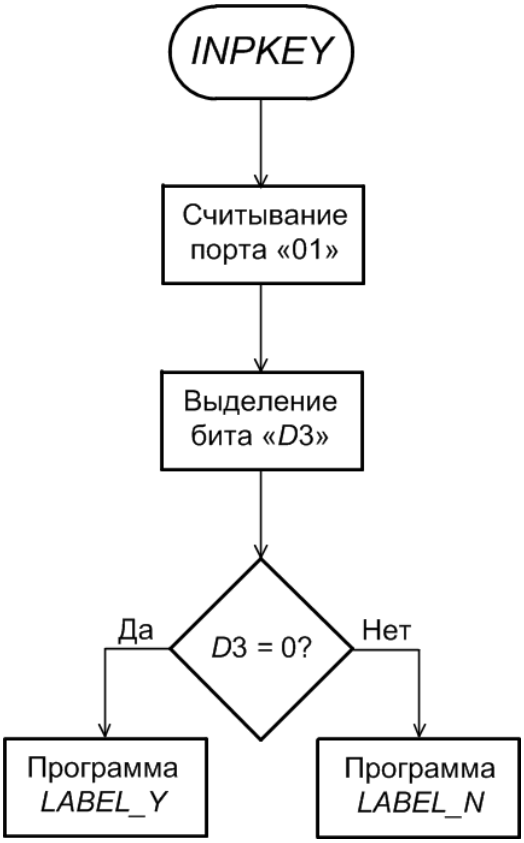


Рис. 4.2. Алгоритм программы 4.1.

Во многих случаях при выполнении программ необходимо проверять или изменять состояние одного или нескольких разрядов числа хранимого в аккумуляторе.

Такая процедура получила наименование «маскирование» т.к. в этой процедуре помимо содержимого аккумулятора используется заранее заданное число или по другому «маска». В маске значения каждого из битов задаются исходя из желаемого результата.

Сама процедура маскирования заключается в применении одной из логических операций над содержимым аккумулятора и маски, что позволяет затем использовать команды условных переходов по флагам «S», «Z», «P».

Если задать маску так, что бы в ней логические единицы «1» были в тех разрядах, которые следует оставить неизменными, а логические нули «0» поставить в разрядах подлежащих очистке, то в результате логического умножения (операция «И») аккумулятора и маски можно осуществить выделение интересующих разрядов. При этом если логическая «1» была только в одном разряде маски, то в результате можно осуществить проверку: если результат равен «0» – то и соответствующий бит (на его позиции в маске была «1») был равен «0», в противном случае этот бит был равен «1».

Если задать маску так, что бы в ней логические единицы «1» были в тех разрядах, которые следует принудительно установить в единицу «1», а логические нули «0» поставить в разрядах, не подлежащих изменению, то в результате логического сложения (операция «ИЛИ») аккумулятора и маски можно осуществить принудительную установку интересующих битов в единицы «1».

Если задать маску так, что бы в ней логические единицы «1» были на тех позициях, которые следует инвертировать и логические нули «0» на тех позициях, которые стоит оставить без изменения, то в результате применения операции «исключающего ИЛИ» над содержимым аккумулятора и маски можно получить инверсию указанных в маске битов (в этих позициях стоят логические единицы «1»).

Дополнительно стоит обратить внимание на еще одно важное свойство операции «исключающего ИЛИ». Если заданная маска и число в аккумуляторе побитно равны, то после выполнения данной операции результат будет равен числу, у которого во всех битах стоят логические нули «0» (т.е. оно арифметически равно «0»). Похожее свойство имеется у операции «исключающее ИЛИ в инверсии» (если такая команда присутствует в системе команд МП): если маска представляет собой побитную инверсию числа в аккумуляторе, то в результате выполнения данной операции получится результат, у которого во всех битах стоят логические нули «0» (т.е. он арифметически равен «0»).

Примеры использования вышеописанных команд приведены в табл. 4.1.

Особо подчеркнем, что процедура маскирования с использованием логических команд не обеспечивает возможность перехода по флагу «С» (он устанавливается в «0») и «АС» – который вообще не изменяется. При этом имеется возможность проведения маскирования в случае наличия маски в одном из регистров.

С учетом всего вышесказанного алгоритм программы 4.1, представленный на рис. 4.2 может быть без труда записан в виде программы на ассемблере вида табл. 4.2.

Таблица 4.1.

Примеры использования логических команд для маскирования

Мнемокод	Машинный код	Число в аккумуляторе	Маска, d8	Комментарий	Результат в аккумуляторе
ANI d8	E6 d8	0011 1010 1111 1111 0000 0000 1010 1010 1111 0000	1010 1100 0010 0010 0010 0010 0010 0010 1111 1111	Логическое умножение содержимого аккумулятора с d8	0010 1000 0010 0010 0000 0000 0010 0010 1111 0000
ORI d8	F6 d8	0011 1010 0000 1111 1111 0000	1010 1100 0000 1111 0000 1111	Логическое сложение содержимого аккумулятора с d8	1011 1110 0000 1111 1111 1111
XRI d8	EE d8	0011 1010 0000 1111 1111 0000	1010 1100 0000 1111 0000 1111	Логическое «исключающее ИЛИ» содержимого аккумулятора с d8	1001 0110 0000 0000 1111 1111

Таблица 4.2.

Программа 4.1

№ этапа	Комментарий по этапу алгоритма	Метка	Мнемокод	Комментарий по команде
1	Считывание порта «01*h*»	INPKEY:	*IN* 01*h*	Назначение команды *IN*: Данные, имеющиеся на 8 битах двунаправленной шины указанного порта, пересылаются в «А».
2	Выделение бита «D3», маска «0000 0100*b*» = «04*h*»		*ANI* 04*h*	Назначение команды *ANI*: Над содержимым «А» и второго байта выполняется логическая операция «И», результат помещается в регистр «А».
3	Проверка «D3»=0(сработал)? если «да» то на «*LABEL_N*»; иначе дальше на LABEL_N		*JZ LABEL_Y*	Назначение команды *JZ*: Если условие «А = 0» истинно, то управление передается по адресу, который указан во втором и третьем байтах команды; если условие ложное, то последовательный ход программы не изменяется
	Подпрограмма LABEL_N:	LABEL_N:	…	…
	…	…	…	…
	Подпрограмма LABEL_Y:	LABEL_Y:	…	…
	…	…	…	…

2. *Работа в режиме ожидания события*

Рассмотрим вопрос программной реализации режима ожидания события. Один из самых простых примеров событий – это срабатывание дискретного датчика, например типа сухой контакт. Такой датчик может быть использован в качестве концевого переключателя ограничивающего движение исполнительного органа, может так же сигнализировать о достижении регулируемого параметра требуемой величины или служит сигналом об аварийной ситуации.

Пусть требуется в управляющей программе реализовать процедуру приостановки ее выполнения до тех пор, пока в результате процессов, происходящих в объекте управления, не замкнется контакт *K* некоторого дискретного датчика. Будем считать, что датчик подключен аналогично рис. 4.1 и его поведение имитируется ключом «S5» (линия порта «D4»). Тогда если «D4 = 1» то датчик не сработал и программу надо приостановить. Очевидно, что алгоритм выполнения такой программы схож с рис. 4.2 (здесь не приводится) и может быть реализован в виде программного кода – см. табл. 4.3.

Таблица 4.3.
Программа 4.2

№ этапа	Комментарий по этапу алгоритма	Метка	Мнемокод	Комментарий по команде
1	Считывание порта «01*h*»	INPKEY:	`IN 01h`	Назначение команды *IN*: Данные, имеющиеся на 8 битах двунаправленной шины указанного порта, пересылаются в «A».
2	Выделение бита «D4», маска «0000 1000*b*» = «08*h*»		`ANI 08h`	Назначение команды *ANI*: Над содержимым «A» и второго байта выполняется логическая операция «И», результат помещается в регистр «A».
3	Проверка «D4≠0»?, если «да» то на «INPKEY»		`JNZ INPKEY`	Назначение команды *JZ*: Пока условие «A ≠ 0» истинно, то управление передается по адресу, который указан во втором и третьем байтах команды; если условие ложное, то последовательный ход программы не изменяется
	Продолжение работы системы	…	…	…

3. *Формирование управляющего сигнала*

Если взаимодействие микропроцессорной системы с внешней средой обеспечено (первый и второй пункт краткой теории), то очевидно, что следующим этапом создания полноценной микроконтроллерной автоматической системы регулирования является проработка вопросов выработки управляющего сигнала. Аналогично пункту 1 будем считать, что имеет место простейший случай [3] подключения к микропроцессору внешнего устройства, например с помощью многорежимного буферного регистра (МБР) К589ИР12. Упрощенная схема подключения представлена на рис. 4.3 для случая организации режима вывода.

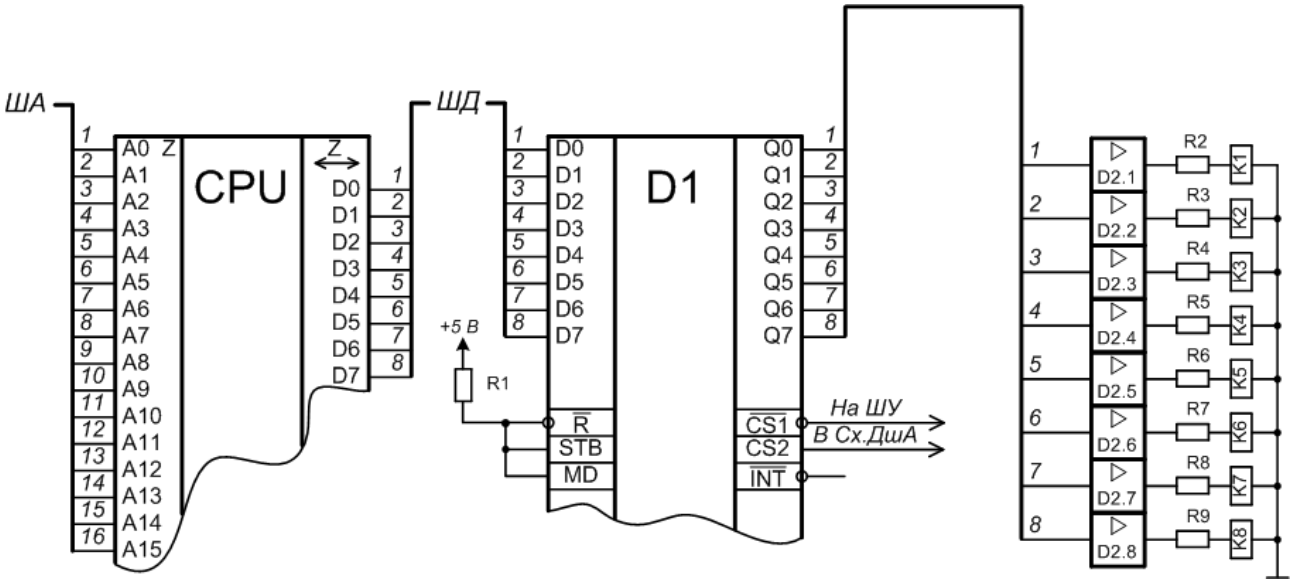


Рис. 4.3. Схема организации вывода с помощью К589ИР12.

В схеме рис. 4.3 «К1», «К2», …, «К8» слаботочные (по входным токам) контакторы, коммутирующие например цепи питания исполнительного механизма работающего по принципу «включено»/«выключено», который открывает или закрывает клапан. При этом «К1» может отвечать за движение исполнительного механизма №1 в одну сторону (условно назовем ее «больше»), а «К2» может отвечать за движение этого же исполнительного механизма №1 в другую сторону (условно назовем ее «меньше»). Возможна и другая логика подключения, возможна организация управления несколькими механизмами при этом очевидно, что на порт необходимо подавать целый байт (8 бит) так называемого

управляющего слова. Подача управляющего воздействия осуществляется с помощью команды «*OUT* N», где «N» – номер порта вывода, а управляющее слово должно храниться в аккумуляторе «А».

4. *Временная задержка*

Если при выполнении рабочей программы микропроцессору требуется функционировать в реальном масштабе времени, например: проводить опрос порта не регулярно, а через заданное время; либо формировать управляющее воздействие определенной длительности; либо ожидать наступления события в течение установленного временного интервала и т.п. то очевидно, что необходим механизм подсчета реально затраченного времени ,в секундах.

Программная реализация временной задержки использует метод организации циклов. При этом в некоторый рабочий регистр блока РОН загружается число, которое затем в каждом проходе цикла уменьшается на «1». Так продолжается до тех пор, пока содержимое рабочего регистра не станет равным «0», что интерпретируется программой как момент выхода из программного цикла. Полученное время задержки (в течение этого времени процессор не выполнял рабочую программу) при этом определяется числом, загруженным в рабочий регистр, и временем выполнения команд, образующих программный цикл.

Схема алгоритма временной задержки (программа 4.3) показана на рис. 4.4. Вход в алгоритм имеет наименование TIME и может использоваться как ссылка при организации сложных программ, если использовать структуру типа «подпрограмма». Фрагмент программного кода, полученного по алгоритму рис 4.4, представлен в табл. 4.4.

Для получения требуемой величины временной задержки необходимо определить значение числа «X», загружаемого в рабочий регистр «B». Число «X» определяется на основе расчета времени выполнения команд, образующих тело цикла.

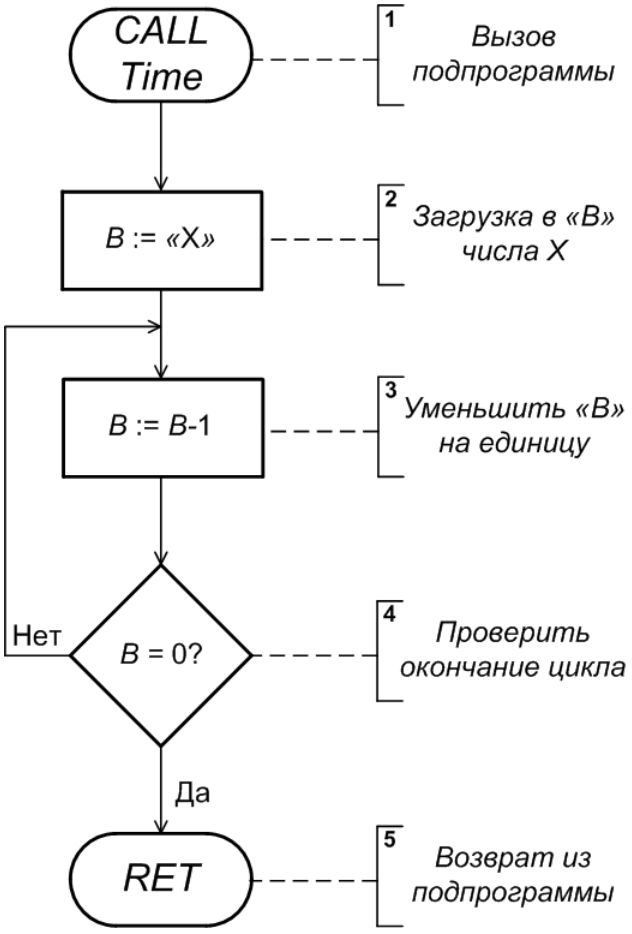


Рис. 4.4. Временная задержка.

Предположим, что в программе, управляющей работой микропроцессорной системы, МП работает с тактовой частотой 2 МГц (период составляет 0,5 мкс). Необходимо реализовать временную задержку длительностью 251 мкс.

В описании системы команд КР580 (см. приложение Б) указывается, за сколько тактов основной частоты синхронизации исполняется каждая команда. На основе этих данных можно записать:

CALL TIME – 17 тактов = 8,5 мкс;

MVI B,«X» – 7 тактов = 3,5 мкс;

DCR B – 5 тактов = 2,5 мкс;

JNZ Mk – 10 тактов = 5 мкс;

RET – 11 тактов – 5,5 мкс.

Таким образом, однократно исполняемые команды «*CALL*», «*MVI*», «*RET*» в этой подпрограмме требуют 17,5 мкс (8,5+3,5+5,5). Следовательно, для получения требуемой задержки в 251 мкс, необходимо выполнить команды «*DCR*» и «*JNZ*» столько раз, чтобы время их исполнения составило 233,5 мкс (251 – 17,5):

$$X = (233,5)/(2,5+5,0) \approx 31,$$

при этом общая задержка составит (31*7,5 + 17,5) = 250 мкс.

Если точность программной реализации временной задержки длительностью 251 мкс с погрешностью 1 мкс удовлетворяет разработчика, то на этом процедура решения останавливается. В противном случае необходимо скорректировать программу: добавив незначащие операции (как в тело цикла, так и вне его) и пересчитав общее время выполнения программы.

Таблица 4.4.
Программа 4.3 «Временная задержка»

№ этапа	Комментарий по этапу алгоритма	Метка	Мнемокод	Комментарий по команде
1	Вызов подпрограммы «TIME»		*CALL* TIME	Назначение команды *CALL*: [SP – 1] ← PCH; [SP – 2] ← PCL; SP ← (SP – 2); PC ← adr; Старше 8 бит адреса следующей команды пересылаются в ячейку памяти, адрес которой на единицу меньше содержимого указателя СТЕКа SP. Младшие 8 бит адреса следующей команды пересылаются в ЯП, адрес которой на 2 меньше содержимого указателя СТЕКа SP. Содержимое указателя СТЕКа уменьшается на 2. Управление передается команде, адрес которой указан во втором и третьем байтах команды вызова – adr.
…	…	…	…	…
2	Загрузка в «B» числа «X»	TIME:	*MVI* B,«X»	Назначение команды *MVI*: Второй байт команды «X» пересылается в регистр «B».
3	Уменьшить «B» на единицу	Mk:	*DCR* B	Назначение команды *DCR*: Содержимое регистра «B» уменьшается на единицу.
4	Проверить окончание цикла		*JNZ* Mk	Назначение команды *JNZ*: Если последний результат не равен «0», то переход по метке Mk, представляющей собой явно заданный адрес участка программы.
5	Возврат из подпрограммы		*RET*	Назначение команды *RET*: PCL ← [SP]; PCH ← [SP + 1]; SP ← (SP + 2); Содержимое ЯП, адрес которой находится в указателе СТЕКа SP, пересылается в младшие 8 бит программного счетчика PC. Содержимое ЯП, адрес которой на 1 больше содержимого указателя СТЕКа пересылается в старшие 8 бит программного счетчика. Содержимое указателя СТЕКа увеличивается на 2.

Реализация временной задержки большей длительности при относительно высокой частоте синхронизации 2 МГц, с использованием описанного выше метода не представляется возможной, т.к. максимальной емкости регистра «FF*h*» не хватит для того, чтобы представить число X, достаточное для формирования задержки, равной, например 1 сек. Сформировать столь большую (для МП) задержку можно с использованием метода вложенных циклов подобно тому, как это показано на рис.4.5 (алгоритм для программы 4.4) и в таблице 4.5. (код программы 4.4).

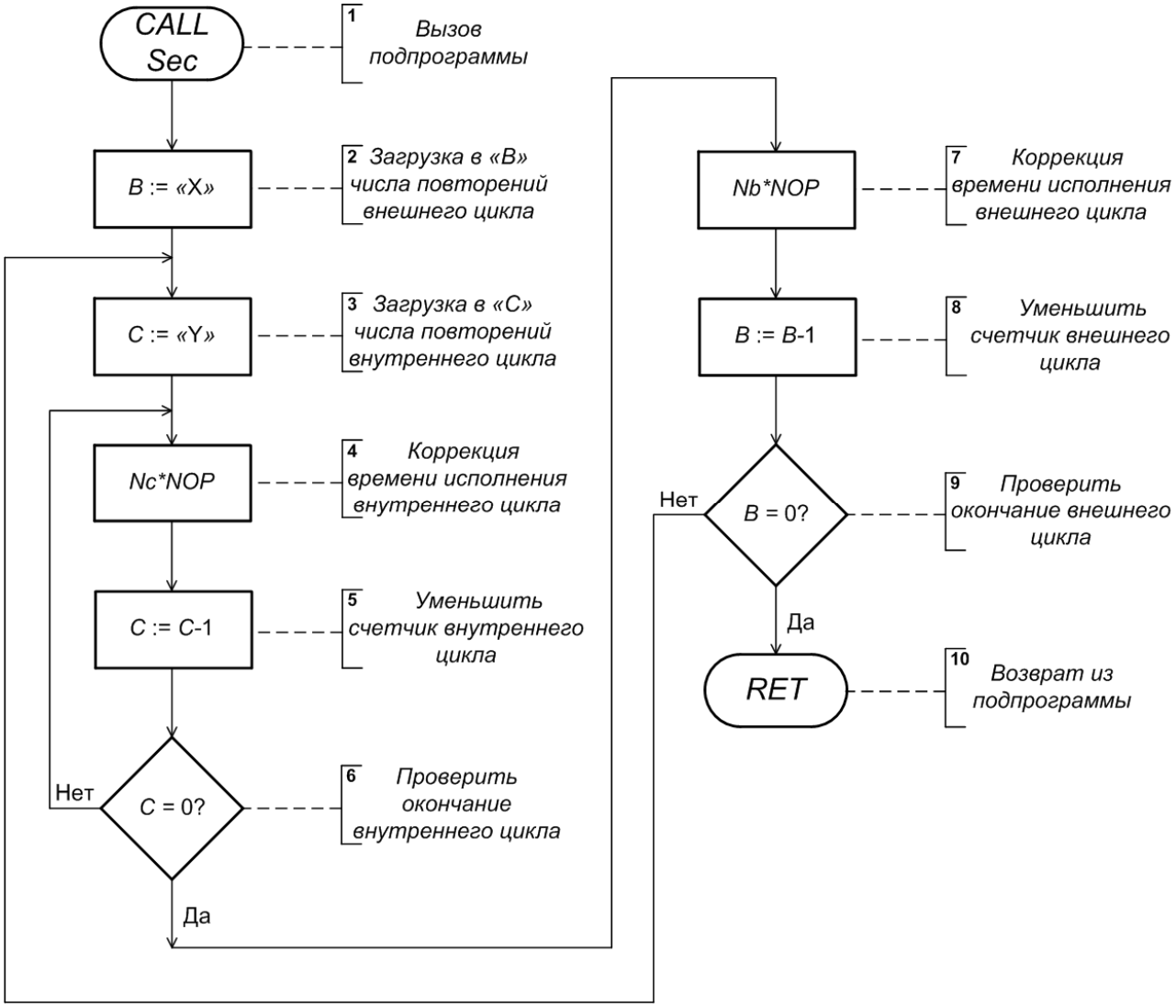


Рис. 4.5. Временная задержка (программа 4.4).

Таблица 4.5.
Программа 4.4 «Временная задержка»

№ этапа	Комментарий по этапу алгоритма	Метка	Мнемокод	Комментарий по команде
1	Вызов подпрограммы «Sec»		`CALL Sec`	Назначение команды *CALL*: [SP – 1] ← PCH; [SP – 2] ← PCL; SP ← (SP – 2); PC ← adr; Управление передается команде, адрес которой указан во втором и третьем байтах команды вызова.
…	…	…	…	…
2	Загрузка в «B» числа повторений внешнего цикла	Sec:	`MVI B,FBh`	Назначение команды *MVI*: Второй байт команды «FB*h*» пересылается в регистр «B».
3	Загрузка в «C» числа повторений внутреннего цикла	Mk1:	`MVI C,E3h`	Назначение команды *MVI*: Второй байт команды «E3*h*» пересылается в регистр «C».
4	Коррекция времени исполнения внутреннего цикла	Mk2:	*NOP* *NOP* *NOP* *NOP* *NOP*	Назначение команды *NOP*: Нет операции.
5	Уменьшить счетчик внутреннего цикла на единицу		`DCR C`	Назначение команды *DCR*: Содержимое регистра «C» уменьшается на единицу.
6	Проверить окончание цикла		`JNZ Mk2`	Назначение команды *JNZ*: Если последний результат не равен «0», то переход по метке Mk2, представляющей собой явно заданный адрес участка программы.
7	Коррекция времени исполнения внешнего цикла		*NOP*	Назначение команды *NOP*: Нет операции.
8	Уменьшить счетчик внешнего цикла на единицу		`DCR B`	Назначение команды *DCR*: Содержимое регистра «B» уменьшается на единицу.
9	Проверить окончание цикла		`JNZ Mk1`	Назначение команды *JNZ*: Если последний результат не равен «0», то переход по метке Mk1, представляющей собой явно заданный адрес участка программы.
10	Возврат из подпрограммы		*RET*	Назначение команды *RET*: PCL ← [SP]; PCH ← [SP + 1]; SP ← (SP + 2); Управление возвращается к прерванной команде, следующей после CALL.

5. *Программная реализация типовых вычислительных процедур*

Остановимся на вопросе программной организации работы системы при необходимости решения сложных вычислительных процедур.

Возникающие в процессе разработки прикладных программ для МП-систем (построенных на основе КР580) сложности определяются: отсутствием команд умножения и деления; малым разрядом используемых операндов и, следовательно, низкой точностью обработки; ограниченным диапазоном представления данных из-за отсутствия команд обработки чисел с плавающей точкой; отсутствием операций десятичной арифметики.

Эти ограничения преодолимы, однако полученные программы являются достаточно сложными. В этой связи использование МП-систем на основе подобных КР580 микропроцессоров наиболее эффективно при решении задач управления, где не требуется вычисление сложных алгоритмов. Если же такие задачи все же ставятся, то существуют типовые решения, которые и представлены в этом разделе.

Задача нахождения суммы элементов массива была решена в работе №3, см. программу 3.1. Эта программа имеет один недостаток: результат не должен превышать числа «FF*h*». Помимо этого предполагалось, что обрабатываемые числа восьмиразрядные. Если требуется обработать числа большей разрядности, то операции с ними необходимо проводить в несколько этапов. Сначала осуществляется операция сложения с восьмью младшими разрядами обоих складываемых чисел, запоминается промежуточный результат и факт переноса в более старшие разряды (анализируется состояние флага «С»). Затем осуществляется сложение старших восьми разрядов чисел с учетом возможного переноса возникшего при сложении младших разрядов. Наличие переноса можно так же учитывать и при сложении восьмиразрядных чисел для учета случая получения шестнадцатиразрядного результата. Это решается путем введения (см. табл. 3.1) в основной цикл дополнительных инструкций: проверяют наличие флага «С» (после выполнения очередной операции сложения), и если

такой факт обнаружен, то увеличивают значение одного из регистров (отвечающего за хранение старших разрядов результата) на единицу.

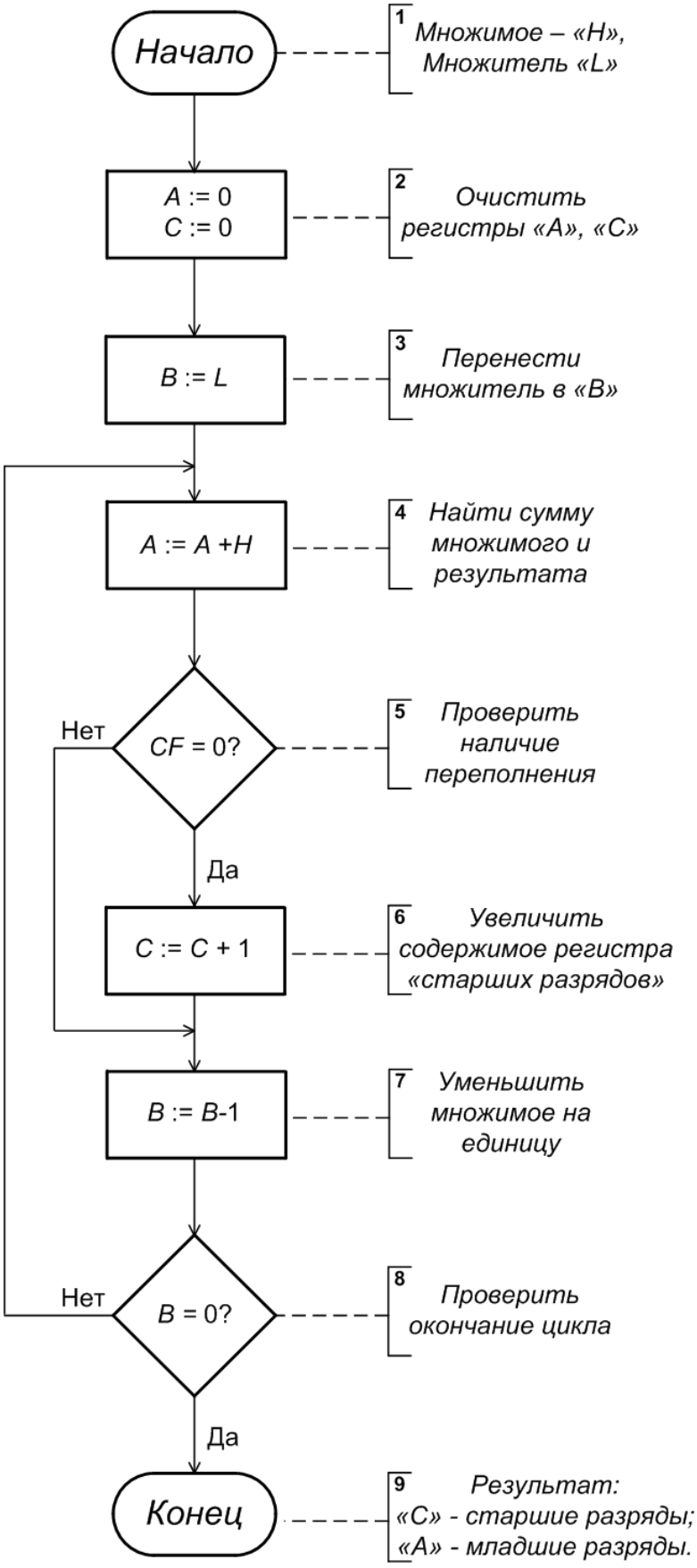


Рис. 4.6. Алгоритм умножения.

Операцию умножения (отсутствующую в системе команд КР580) элементов можно осуществить путем использования циклического сложения. Например, если требуется умножить два однобайтных числа без знака «A» и «B» то можно программно организовать цикл в котором число «A» складывать само с собой «B» раз – см. алгоритм рис. 4.6.

Программа на ассемблере, составленная по этому алгоритму может быть представлена в виде табл. 4.6, где предполагается, что в регистре «H» хранится множимое, а в регистре «L» множитель. Полученный результат (шестнадцатиразрядный) образовывает содержимое пары регистров «C», «A» (где «C» – старшие разряды результата, «A» – младшие разряды результата).

Таблица 4.6.
Программа 4.5 «Умножение чисел»

№ этапа	Комментарий по этапу алгоритма	Метка	Мнемокод	Комментарий по команде
1	Множимое – «H», Множитель – «L»		…	
2	Очистить регистры «A», «C»		`SUB A`	Назначение команды *SUB*: Содержимое регистра «A» вычитается из «A», результат помещается в «A».
			`MVI C,00h`	Назначение команды *MVI*: Второй байт команды «00*h*» пересылается в регистр «C».
3	Перенести множитель в «B»		`MOV B,L`	Назначение команды *MOV*: Содержимое регистра «L» пересылается в регистр «B».
4	Найти сумму множимого и результата	Mk2:	`ADD H`	Назначение команды *ADD*: Сложение содержимого «A» и регистра «H», результат помещается в регистр «A».
5	Проверить наличие переполнения		`JNC Mk1`	Назначение команды *JNC*: Если последний результат не имеет перенос «C», то переход по метке Mk1, представляющей собой явно заданный адрес участка программы.
6	Увеличить содержимое регистра «старших разрядов»		`INR C`	Назначение команды *INR*: Содержимое регистра «C» увеличивается на единицу.
7	Уменьшить множимое на единицу	Mk1:	`DCR B`	Назначение команды *DCR*: Содержимое регистра «B» уменьшается на единицу.
8	Проверить окончание цикла		`JNZ Mk2`	Назначение команды *JNZ*: Если последний результат не равен «0», то переход по метке Mk2, представляющей собой явно заданный адрес участка программы.
10	Конец Результат: «C» - старшие разряды; «A» - младшие разряды.		`HLT`	Назначение команды *HLT*: Останов.

Порядок работы

Задание 1. Анализ работы микропроцессора в режиме ожидания события

Составить программу ожидания события, аналогично п.2 краткой теории, перевести ее в машинный код, занести в ОЗУ эмулятора и проверить правильность работы. В качестве «события» (которое требуется ожидать) принять факт замыкания *любого* из указанных в табл. 4.7 ключей (датчиков, помеченных «х»), подключенных к порту «05h».

Таблица 4.7.

Номера ключей, срабатывание которые необходимо обнаружить

№ ключа	№ бита порта	Варианты														
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
S1	D0	x	x		x	x			x	x				x		
S2	D1			x		x		x		x				x	x	
S3	D2			x				x		x			x	x	x	
S4	D3			x				x		x			x		x	
S5	D4			x			x	x	x	x	x		x	x	x	x
S6	D5			x			x				x		x		x	
S7	D6			x	x		x				x		x		x	
S8	D7	x		x	x		x					x	x		x	

Примечание: в случае отсутствия в используемом эмуляторе возможности изменения состояния на выбранной внешней линии (путем изменения положения ключей или тумблеров) необходимо изменить программу и вместо команды «*IN* N» использовать обращение к группе последовательно расположенных ячейках памяти (ожидать факт обнаружения в разрядах очередной ячейке памяти требуемого условия). Обрабатываемые ячейки разместить, начиная с адреса «0B00h» и заполнить по своему усмотрению. Необходимо при этом предусмотреть возможность определения, на каком адресе произошло событие.

Задание 2. Исследование работы микропроцессора в режиме формирования управляющего сигнала

Используя результаты краткой теории, п.3. создать программу по формированию управляющего воздействия, подаваемого на группу исполнительных

механизмов, которые через систему гальванической развязки подключены к порту «07*h*». Программу перевести в машинный код, записать в память эмулятора и исследовать в пошаговом режиме. Информация о том, какой механизм требуется включить представлена для каждого из вариантов в виде табл. 4.8. Не указанные в таблице механизмы требуется выключить.

Таблица 4.8.
Состояние исполнительных механизмов

№ механизма	№ бита порта	Варианты														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
M1	D0	вкл.	вкл.	вкл.	вкл.	вкл.	вкл.			вкл.		вкл.	вкл.		вкл.	
M2	D1		вкл.					вкл.	вкл.	вкл.		вкл.		вкл.		
M3	D2	вкл.			вкл.	вкл.	вкл.	вкл.	вкл.		вкл.			вкл.	вкл.	вкл.
M4	D3			вкл.	вкл.						вкл.		вкл.	вкл.	вкл.	вкл.
M5	D4			вкл.		вкл.			вкл.	вкл.	вкл.	вкл.	вкл.			вкл.
M6	D5	вкл.	вкл.		вкл.		вкл.	вкл.	вкл.	вкл.		вкл.	вкл.		вкл.	вкл.
M7	D6	вкл.	вкл.		вкл.	вкл.	вкл.		вкл.	вкл.				вкл.		
M8	D7		вкл.	вкл.		вкл.		вкл.				вкл.	вкл.	вкл.		вкл.

Задание 3. Создания программы формирования заданной временной задержки

Используя информацию, представленную в п.4, о принципах и методах формирования программной задержки необходимо:

- рассчитать точно время выполнения программы 4.4;
- модифицировать программу так, что бы время задержки составляло величину, указанную в табл.4.9.;
- перевести полученную программу в машинный код, записать в эмулятор и исследовать ее выполнение.

Таблица 4.9.
Необходимое время программной задержки

Время в сек. для указанного варианта														
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0,3	*1,25*	*1,4*	*0,75*	*0,4*	*0,67*	*1,3*	*2,0*	*0,12*	*0,66*	*0,9*	*1,6*	*2,5*	*1,5*	*0,57*

Примечание: необходимо учесть, что время «эмулирования» при запуске исследуемой программы может не совпасть с расчетным временем работы процессора т.к. эмулятор работает с частотой, большей, чем 2 МГц.

Задание 4. Создание программы реализации процедуры сложения массива чисел

Создать программу по вычислению суммы элементов массива с учетом замечаний п.5 краткой теории: т.е. обеспечить возможность получения шестнадцатиразрядного результата. При составлении программы считать, что массив обрабатываемых чисел формируется с учетом таблицы 4.10.

Таблица 4.10.
Варианты распределения массива чисел

Назначение адреса памяти	**Значения адресов ячеек памяти по вариантам, *h***														
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Количество обрабатываемых чисел	0B11	0B21	0B31	0B41	0B51	0B61	0B71	0B81	0B91	0BA1	0BB1	0BC1	0BD1	0BE1	0BF1
Начало расположения массива чисел	0B12	0B22	0B32	0B42	0B52	0B62	0B72	0B82	0B92	0BA2	0BB2	0BC2	0BD2	0BE2	0BF2
Место-расположение результата	0B10	0B20	0B30	0B40	0B50	0B60	0B70	0B80	0B90	0BA0	0BB0	0BC0	0BD0	0BE0	0BF0

Перевести полученную программу в машинный код, записать в память эмулятора и исследовать на работоспособность.

При исследовании количество обрабатываемых чисел задании в соответствии с номером своего варианта:

Количество чисел = (*№ варианта*) * 10.

Элементы массива обрабатываемых чисел задать произвольно, но так, что бы полученный в итоге результат быт шестнадцатиразрядным.

Задание 5. Изучение программы реализации процедуры умножения массива чисел

Перевести программу 4.5 (см. п. 5 краткой теории) по умножению массива однобайтных двоичных чисел в машинный код, Занести ее в память эмулятора и исследовать. Сверить полученные данные (при выполнении программы) с расчетными. Умножаемые числа выбрать произвольно, но не менее трех наборов. Сделать вывод о работе программы и о способах оптимизации программного кода с точки зрения улучшения производительности.

Контрольные вопросы

1. Каким образом в микропроцессорной системе может происходить опрос дискретного датчика?
2. Как можно выделить определенный разряд в двоичном числе?
3. Каким образом может быть использована команда логического умножения при маскировании данных?
4. Как используется логическое сложение при маскировании?
5. Для чего может быть использована логическая операция исключающее «ИЛИ» (исключающее «ИЛИ» в инверсии)?
6. В чем суть работы системы в режиме ожидания события?
7. Как можно сформировать управляющий сигнал в микропроцессорной системе управления?
8. Приведите программные способы формирования временной задержки.
9. Где может быть использована временная задержка?
10. Опишите способы программной реализации процедуры умножения.

ЗАКЛЮЧЕНИЕ

В настоящем пособии достаточно подробно рассмотрены вопросы создания программ для простейших микропроцессоров – КР580 и Intel 8080. Если представленного материала окажется недостаточным то можно также обратиться к литературе [3, 4, 6, 10].

Сами задания практикума не представляют большой сложности, но как было отмечено ранее, служат «первой попыткой» построения программ для встраиваемых систем управления.

Предполагается что основная задача первой части выполнена – показать, что концепция построения современных одноплатных микроконтроллеров не возникла на пустом месте, а стала ответом на объективные сложности, которые имелись при реализации первых микропроцессорных систем управления технологическими процессами. Это, прежде всего сложности побитного управления линиями портов, необходимость сложной схемотехнической и программной реализации задач управления в реальном времени и сопряжения системы с другими элементами автоматики (схемотехническим проблемам и описание сложности получающейся в итоге микропроцессорной системы управления посвящена вторая часть).

Конечно, нельзя сравнивать микроконтроллер и микропроцессор т.к. каждый из них решает свои задачи, что требует своих архитектурных особенностей. В этой связи еще больший интерес вызывает рассматриваемый КР580(i8080) т.к. он по сути, как в нашей стране, так и зарубежном явился «прародителем» – на его основе строились как локальные системы управления [1 - 7], так и персональные компьютеры [11].

Хотя, конечно же КР580(i8080) в современных условиях представляет лишь академический интерес, коммерчески реализуемых систем с его применением создать невозможно, хотя и изучить высшую математику невозможно, без того что бы научится считать столбиком…

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Нестеров, П. В.* Микропроцессоры. В 3 ч. Ч. 1. Архитектура и проектирование микро-ЭВМ. Организация вычислительных процессов / П. В. Нестеров, В. Ф. Шаньгин, В. Л. Горбунов и др. ; под. ред. Л. Н. Преснухина. – М. : Высшая школа, 1986. – 495 с.

2. *Вернер, В. Д.* Микропроцессоры. В 3 ч. Ч. 2. Средства сопряжения. Контролирующие и информационно-управляющие системы / В. Д. Вернер, Н. В. Воробьев, А. В. Горячев и др. ; под. ред. Л. Н. Преснухина. – М. : Высшая школа, 1986. – 383 с.

3. *Воробьев, Н. В.* Микропроцессоры. В 3 ч. Ч. 3. Средства отладки, лабораторный практикум и задачник / Н. В. Воробьев, В. Л. Горбунов, А. В. Горячев и др. ; под. ред. Л. Н. Преснухина. – М. : Высшая школа, 1986. – 351 с.

4. *Гуртовцев, А. Л.* Программы для микропроцессоров. Справочное пособие. / А. Л. Гуртовцев, С.В. Гудыменко. – Минск: Вышэйша школа, 1989.

5. *Зубчук, В. И.* Справочник по цифровой схемотехнике / В. И. Зубчук, В. П. Сигорский, А. Н. Шкуро. – К. : Тэхника, 1990. – 448 с.

6. Микропроцессорная лаборатория «МИКРОЛАБ КР580ИК80». – М. : Внешторгиздат. – 129 с.

7. *Гришин, Ю. П.* Микропроцессоры в радиотехнических системах / Ю. П. Гришин, Ю. М. Казаринов, В. М. Катиков. Под ред. Ю. М. Казаринова. – М. : Радио и связь, 1982. – 280 с.

8. *Алтухов, Е. В.* Основы информатики и вычислительной техники / Е. В. Алтухов, Л. А. Рыбалко, В. С. Савченко. – М. : Высш. шк., 1992. – 303 с.

9. *ГОСТ 19.701-90.* Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. – Введ. 01.01.92. – М. Государственный стандарт Союза ССР.

10. *Палагута, К. А.* Микропроцессоры INTEL 8080, 8085 (КР580ВМ80А, КР1821ВМ85А) и их программирование / К. А. Палагута. – М. : МГИУ, 2007. – 104 с.

11. *Барышников, В. Н.* Персональный компьютер «Ириша» / В. Н. Барышников, М. А. Воронов, В. Б. Кулаков и др. – М. : Патриот, 1990. – 176 с.

ПРИЛОЖЕНИЕ А

ПРОГРАММНЫЕ СРЕДСТВА

1. Программа-эмулятор микропроцессорной системы на базе микропроцессора КР580ВМ80. Версия 1.0.2.1735.

Производитель: МАИ. Кафедра электронно-вычислительных средств и информатики.

Авторы: Пенкин Ю. И., Улыбашев Д. А.

Сайт: http://www.zic-homepage.narod.ru/Main/Pages/KP580.htm,
http://kaf403.rloc.ru/; http://kaf403.mai.ru/

Обнаруженные неточности:

- не работают команды *MOV* M,R (в память пересылается только содержимое A);
- не правильно работает переход *JM* adr;
- флаг «S» устанавливается некорректно;
- не правильно работает команда сравнения, например *CMP* M выставляет флаг C=0 если A < ЯП[H,L] и C=1 если A > ЯП[H,L].

2. Эмулятор работы микропроцессора КР580ВМ80.

Производитель: МИРЭА. Кафедра вычислительной техники.

Авторы: Иващенко Д. И., Иванов Е. Л.

Сайт: http://www.malshakov.tushino.com/EMULATOR580.rar
http://www.vt.fvms.mirea.ru/

Обнаруженные неточности:

- опция [Файл \ Сохранить] не работает, надо использовать [… Сохранить как];
- не корректно работает команда *SUB* M с кодом 96*h* (ошибка в единицу), вместо нее, если возможно, необходимо использовать команду *SBB* M с кодом 9E*h*;
- команда *CMP* M не работает (эмулятор зависает), можно использовать *CMP* R, предварительно загрузив в R содержимое M;
- в команде *JMP* adr и *CALL* adr в начале необходимо написать старший байт адреса, а потом младший (в отличие от других команд перехода, где сначала указывается младший байт);
- имеется ограничение на количество повторений циклов.

3. Программная модель стенда УМПК-80 на базе микропроцессора К580ВМ80 (v 2.0).

Производитель: Курский государственный технический университет. Кафедра вычислительной техники.

Автор: Финаков С. Ю.

Сайт: http://www.twirpx.com/file/472/
http://www.kurskstu.ru/structura/up/fivt/kvt/

ПРИЛОЖЕНИЕ Б

СИСТЕМА КОМАНД МИКРОПРОЦЕССОРА

Набор команд микропроцессора КР580 может быть представлен в виде таблицы, приведенной ниже. Здесь представлено: обозначение операции на языке ассемблер (колонка «мнемокод»), количество байт занимаемое каждой командой в памяти (колонка «байты»), количество машинных циклов (колонка «циклы») и тактов (колонка «такты») затрачиваемых микропроцессором на выполнение команды, сокращенное обозначение выполняемой команды (колонка «выполняемая операция»), пояснения по выполняемым в данном случае действиям (колонка «комментарии»), состояние регистра флагов после выполнения указанной команды (колонка «признаки»).

Символы и сокращения, используемые в таблице:

«A» – аккумулятор (регистр A);

«adr» – константа, задаваемая в команде, размером 16 бит (адрес);

«d16» – константа, задаваемая в команде, размером 16 бит (данные);

«d8» – константа, задаваемая в команде, размером 8 бит;

«R» – один из регистров A, B, C, D, E, H, L;

«M», «ЯП» – содержимое ячейки памяти, адрес которой хранится в регистрах H, L;

«YZ» – регистровая пара (BC, DE, HL);

«SP» – 16-битовый регистр указателя СТЕКа;

«PC» – 16-битовый регистр программного счетчика;

«PSW» – двухбайтовое содержимое регистров A и F;

«N» – номер порта ввода-вывода;

«$\wedge$», «$\vee$», «$\oplus$» – логические команды И, ИЛИ, исключающее ИЛИ соответственно;

«$\leftarrow$» – обозначение направления присвоения (пересылки);

«+» – в колонке признаки обозначает возможность изменения соответствующего флага;

«-» – в колонке признаки обозначает невозможность изменения соответствующего флага.

Таблица Б. Система команд микропроцессора

Мнемокод	Байты	Циклы	Такты	Выполняемая операция	Комментарий	Признаки				
						S	Z	AC	P	C
Команды пересылок										
MOV R1,R2	1	1	5	Пересылка данных из регистра в регистр	R1 ← R2; Содержимое регистра R2 пересылается в регистр R1	-	-	-	-	-
MOV R,M	1	2	7	Пересылка данных из памяти	R ← ЯП[H,L]; Содержимое ЯП, адрес которой указан в регистрах H,L, пересылается в R	-	-	-	-	-
MOV M, R	1	2	7	Пересылка данных в память	ЯП[H,L] ← R; Содержимое регистра R пересылается в ЯП, адрес которой содержится в регистре H,L.	-	-	-	-	-
MVI R,d8	2	2	7	Непосредственная пересылка	R1 ← d8 Второй байт команды (d8) пересылается в регистр R.	-	-	-	-	-
MVI M,d8	2	3	10	Непосредственная пересылка в память	ЯП[H,L] ← d8; Второй байт команды (d8) пересылается в ЯП[H,L], адрес которой указан в регистрах H,L.	-	-	-	-	-
LXI YZ,d16	3	3	10	Непосредственная загрузка пары регистров	YZ ← d16; Второй и третий байты команды пересылаются в пару регистров YZ.	-	-	-	-	-
LDA adr	3	4	13	Прямая загрузка аккумулятора	A ← ЯП[adr]; Содержимое ЯП, адрес которой указан в команде [adr] пересылается в A.	-	-	-	-	-
STA adr	3	4	13	Прямая запись содержимого A в память	ЯП[adr] ← A; Содержимое A пересылается в ЯП по указанному в команде адресу.	-	-	-	-	-
LHLD adr	3	5	16	Прямая загрузка регистров	L ← [adr]; H ← [adr +1]; Содержимое ячейки памяти по адресу [ADR] пересылается в регистр L; Содержимое ЯП со следующим адресом [ADR+1] пересылается в регистр H.	-	-	-	-	-
LDAX YZ	1	2	7	Косвенная загрузка аккумулятора	A ← ЯП[YZ]; Содержимое ЯП, адрес которой указан в паре регистров YZ помещается в A.	-	-	-	-	-

Продолжение таблицы Б

Мнемокод	Байты	Циклы	Такты	Выполняемая операция	Комментарий	Признаки				
						S	Z	AC	P	C
STAX YZ	1	2	7	Косвенная запись содержимого аккумулятора в память	ЯП[YZ] ← A; Содержимое A пересылается в ЯП, адрес которой указан в паре регистров YZ.	-	-	-	-	-
XCHG	1	1	4	Обмен данными между регистрами H,L и D,E	(H) ↔ (D); (L) ↔ (E); Содержимое регистров H,L обмениваются содержимым регистров D,E.	-	-	-	-	-
XTHL	1	5	18	Обмен содержимого верхушки СТЕКа и регистров H,L	L ↔ [SP]; H ↔ [SP+1]; Содержимое регистра L обменивается на содержимое ЯП, адрес которой содержится в указателе СТЕКа SP. Содержимое регистра H обменивается на содержимое ЯП, адрес которой на 1 больше содержимого указателя СТЕКа.	-	-	-	-	-
SPHL	1	1	5	Пересылка содержимого регистров H,L в указатель СТЕКа	SP ← HL; Содержимое регистров H,L (16 бит) пересылается в указатель СТЕКа.	-	-	-	-	-
PCHL	1	1	5	Загрузка содержимого регистров H и L в программный счетчик	PCH ← H; PCL ← L; Содержимое регистра H пересылается в старшие 8 бит программного счетчика PC. Содержимое регистра L пересылается в младшие 8 бит программного счетчика.	-	-	-	-	-
Арифметические команды										
ADD R	1	1	4	Сложение содержимого A с содержимым R	A ← (A+R); Результат помещается в регистр A.	+	+	+	+	+
ADD M	1	2	7	Сложение содержимого A с содержимым ЯП	A ← (A + ЯП[H,L]); Содержимое ЯП, адрес которой указан в [H,L], складывается с содержимым A. Результат помещается в регистр A.	+	+	+	+	+

Продолжение таблицы Б

Мнемокод	Байты	Циклы	Такты	Выполняемая операция	Комментарий	Признаки				
						S	Z	AC	P	C
ADI d8	2	2	7	Непосредственное сложение	A ← (A+d8); Содержимое второго байта команды складывается с содержимым A. Результат помещается в A.	+	+	+	+	+
SUB M	1	1	7	Вычитание содержимого ЯП	A ← (A-ЯП[H,L]); Содержимое ЯП с адресом [H,L] вычитается из содержимого A.Результат помещается в A.	+	+	+	+	+
SUI d8	2	2	7	Непосредственное вычитание	A ← (A–d8); Содержимое второго байта команды вычитается из A. Результат помещается в регистр A.	+	+	+	+	+
SBB R	1	1	4	Вычитание содержимого регистра и бита флага переноса	A ← (A–R–C); Содержимое регистра R и бит флага переноса C вычитаются из A. Результат в регистр A.	+	+	+	+	+
SBB M	1	2	7	Вычитание содержимого ЯП и бита флага переноса	A ← (A–ЯП[H,L] –C); Содержимое ЯП с адресом [H,L] и бит флага переноса C вычитаются из A. Результат помещается в регистр A.	+	+	+	+	+
SBI d8	2	2	7	Непосредственное вычитание с займом	A ← (A–d8–C); Содержимое второго байта команды и бит флага переноса C вычитаются из аккумулятора. Результат помещается в регистр A.	+	+	+	+	+
ADC R	1	1	4	Сложение содержимого A с содержимым регистра R и битом флага переноса	A ← (A+R+C); Содержимое регистра R и бит флага переноса C складывается с содержимым A. Результат помещается в A.	+	+	+	+	+
ADC M	1	2	7	Сложение содержимого A с содержимым ЯП и битом флага переноса	A ← (A+ЯП[H,L]+C); Содержимое ЯП с адресом [H,L] и содержимое бита флага переноса складывается с содержимым A. Результата помещается в A.	+	+	+	+	+

Продолжение таблицы Б

Мнемокод	Байты	Циклы	Такты	Выполняемая операция	Комментарий	Признаки				
						S	Z	AC	P	C
ACI d8	2	2	7	Непосредственное сложение с битом флага переноса	A ← (A+d8+C); Содержимое второго байта команды и бита флага переноса складываются с содержимым A. Результат помещается в A.	+	+	+	+	+
SUB R	1	1	4	Вычитание содержимого регистра	A ← (A–R); Содержимое регистра R вычитается из A. Результат помещается в A.	+	+	+	+	+
INR R	1	1	5	Увеличение содержимого регистра R	R ← (R+l); Содержимое регистра R увеличивается на единицу.	+	+	+	+	-
INR M	1	3	10	Увеличение содержимого ЯП	ЯП[H,L] ← (ЯП[H,L]+1); Содержимое ЯП с адресом [H,L] увеличивается на единицу.	+	+	+	+	-
DCR R	1	1	5	Уменьшение содержимого регистра R	R ← (R–l); Содержимое регистра R уменьшается на единицу.	+	+	+	+	-
DCR M	1	3	10	Уменьшение содержимого ЯП	ЯП[H,L] ← (ЯП[H,L]–1); Содержимое ЯП с адресом [H,L] уменьшается на единицу.	+	+	+	+	-
INX YZ	1	1	5	Увеличение содержимого пары регистров	YZ ← (YZ+1); Содержимое пары регистров увеличивается на единицу.	-	-	-	-	-
DCX YZ	1	1	5	Уменьшение содержимого пары регистров	YZ ← (YZ+1); Содержимое пары регистров уменьшается на единицу.	-	-	-	-	-
DAD YZ	1	3	10	Сложение содержимого пары регистров с содержимым регистров H,L	(H,L) ← ((H,L)+(YZ)); Содержимое пары регистров складывается с содержимым регистров H,L. Результат помещается в пару регистров H,L.	-	-	-	-	+
DAA	1	1	4	Десятичное дополнение аккумулятора	8-битовое число A дополняется до представления в виде двух 4-битовых чисел в двоично-десятичном коде с помощью специальных операций.	+	+	+	+	+

Продолжение таблицы Б

Мнемокод	Байты	Циклы	Такты	Выполняемая операция	Комментарий	Признаки				
						S	Z	AC	P	C
DCX SP	1	1	5	Уменьшение содержимого указателя СТЕКа на единицу	SP ← (SP–1); Содержимое указателя СТЕКа уменьшается на единицу.	-	-	-	-	-
INX SP	1	1	5	Увеличение содержимого указателя СТЕКа на единицу	SP ← (SP+1); Увеличение указателя СТЕКа уменьшается на единицу.	-	-	-	-	-
Логические команды										
ANA R	1	1	4	Операция «И» над содержимым регистра и аккумулятора	A ← (A∧R); Результат помещается в регистр A.	+	+	-	+	0
ANA M	1	2	7	Операция «И» над содержимым ЯП и A	A ← (A∧ЯП[H,L]); Результат помещается в регистр A.	+	+	-	+	0
ANI d8	2	2	7	Непосредственная операция «И»	A ← (A∧d8); Над содержимым A и второго байта (d8) выполняется логическая операция «И». Результат помещается в регистр A.	+	+	-	+	0
XRA R	1	1	4	Операция «исключающее ИЛИ» над содержимым R и A	A ← (A⊕R); Результат помещается в регистр A.	+	+	-	+	0
XRA M	1	2	7	Операция «исключающее ИЛИ» над содержимым ЯП и A	A ← (A⊕ЯП[H,L]); Результат помещается в регистр A.	+	+	-	+	0
XRI d8	2	2	7	Непосредственная операция исключающее ИЛИ	A ← (A⊕d8); Результат помещается в регистр A.	+	+	-	+	0
ORA R	1	1	4	Логическая операция «сложение (ИЛИ)» содержимого A с содержимым регистра	A ← (A∨R); Результат помещается в регистр A.	+	+	-	+	0
ORA M	1	2	7	Логическая операция «ИЛИ» содержимого ЯП и A	A ← (A∨ЯП[H,L]); C=0; AC=0; Результат помещается в регистр A.	+	+	-	+	0
ORI d8	2	2	7	Логическая операция «ИЛИ»	A ← (A∨d8); C=0; AC=0.	+	+	-	+	0

Продолжение таблицы Б

Мнемокод	Байты	Циклы	Такты	Выполняемая операция	Комментарий	Признаки				
						S	Z	AC	P	C
Команды инвертирования и установки переноса										
CMA	1	1	4	Инвертирование аккумулятора	A ← A’; Содержимое A инвертируется (бит, равный единице, становится равным нулю; бит, равный нулю – единице).	-	-	-	-	-
CMC	1	1	4	Инвертирование флага переноса	C ← C’; Инвертируется бит флага переноса C.	-	-	-	-	+
STC	1	1	4	Установка флага переноса	C ← 1; Бит флага переноса устанавливается = 1.	-	-	-	-	1
Команды сравнения										
CMP R	1	1	4	Сравнение содержимого указанного регистра с содержимым A	A – R; Содержимое R сравнивается с содержимым A. При этом содержимое A не изменяется. По результату сравнения устанавливаются флаги: Z = 1, если A = R; C = 1, если A < R.	+	+	+	+	+
CMP M	1	2	7	Сравнение содержимого ЯП с содержимым A	A – (ЯП[H,L]); Содержимое ЯП, адрес которой указан в регистрах H,L вычитается из содержимого A. При этом содержимое A не изменяется. По результатам сравнения устанавливаются флаги: Z = 1, если A = ЯП[H,L]; C =1, если A < ЯП[H,L].	+	+	+	+	+
CPI d8	2	2	7	Непосредственное сравнение	A ← d8; Содержимое второго байта команды вычитается из содержимого A. По результатам устанавливаются флаги: Z = 1, если A = d8; C = 1, если A < d8; C = 0, если A > d8.	+	+	+	+	+

Продолжение таблицы Б

Мнемокод	Байты	Циклы	Такты	Выполняемая операция	Комментарий	Признаки				
						S	Z	AC	P	C
Команды сдвига										
RAL	1	1	4	Сдвиг влево через перенос	$A_{n+1} \leftarrow A_n$; $C \leftarrow A_7$; $A_0 \leftarrow C$; Содержимое А сдвигается на одну позицию влево через бит флага переноса С. Младший бит устанавливается равным флагу переноса, а бит флага переноса С равным старшему биту А.	-	-	-	-	+
RAR	1	1	4	Сдвиг вправо через перенос	$A_n \leftarrow A_{n+1}$; $C \leftarrow A_0$; $A_7 \leftarrow C$; Содержимое А сдвигается на одну позицию вправо через бит флага переноса С. Старший бит А=С, а С = младшему биту А.	-	-	-	-	+
RLC	1	1	4	Циклический сдвиг влево	$A_{n+1} \leftarrow A_n$,; $A_0 \leftarrow A_7$; $C \leftarrow A_7$; Содержимое А сдвигается влево на одну позицию. Содержимое старшего бита заносится в младший бит и бит флага переноса.	-	-	-	-	+
RRC	1	1	4	Циклический сдвиг вправо	$A_n \leftarrow A_{n+1}$; $A_7 \leftarrow A_0$; $C \leftarrow A_0$; Содержимое А сдвигается на одну позицию вправо. Содержимое младшего бита заносится в самый старший бит и в флаг С.	-	-	-	-	+
Команды переходов, вызова и возврата										
JMP adr	3	3	10	Переход безусловный	PC ← adr Управление передается команде, адрес которой указан во втором и третьем байтах команды.	-	-	-	-	-

Продолжение таблицы Б

Мнемокод	Байты	Циклы	Такты	Выполняемая операция	Комментарий
J(условие) adr *например:* *JNC* adr	3	3	10	Условие перехода	Если указанное условие истинно, то управление передается команде, адрес которой указан во втором и третьем байтах команды перехода. Если условие ложное, то последовательный ход программы не изменяется
CALL adr	3	5	17	Вызов	[SP – 1] ← PCH; [SP – 2] ← PCL; SP ← (SP – 2); PC ← adr; Старше 8 бит адреса следующей команды пересылаются в ячейку памяти, адрес которой на единицу меньше содержимого указателя СТЕКа SP. Младшие 8 бит адреса следующей команды пересылаются в ЯП, адрес которой на 2 меньше содержимого указателя СТЕКа SP. Содержимое указателя СТЕКа уменьшается на 2. Управление передается команде, адрес которой указан во втором и третьем байтах команды вызова – adr.
C(условие) adr, *например*: *CNZ* adr	3	5	17	Условный вызов подпрограммы	Если указанное условие истинно, то выполняются действия, описанные в команде CALL; в противном случае последовательность выполнения команд не меняется
RET	1	3	11	Возврат из подпрограммы	PCL ← [SP]; PCH ← [SP + 1]; SP ← (SP + 2); Содержимое ЯП, адрес которой находится в указателе СТЕКа SP, пересылается в младшие 8 бит программного счетчика PC. Содержимое ЯП, адрес которой на 1 больше содержимого указателя СТЕКа пересылается в старшие 8 бит программного счетчика. Содержимое указателя СТЕКа увеличивается на 2.
R (условие) *например*: *RPO*	1	3	11	Условный возврат из подпрограммы	Если указанное условие истинно, то выполняются действия, описанные в команде RET, а в противном случае последовательность выполнения команд не нарушается.
Примечание: В командах условных переходов параметр (условие) в зависимости от флагов (Z, C, P, S) может иметь следующий вид: *NZ* – не ноль (Z=0); *Z* – ноль (Z=1); *NC* – нет переноса (C=0); *C* – есть перенос (C=1); *PO* – нечетный результат (P=0); *PE* – четный результат (P=1); *P* – плюс (S=0); *M* – минус (S=1).					

Продолжение таблицы Б

Мнемокод	Байты	Циклы	Такты	Выполняемая операция	Комментарий
Команды работы со стеком					
PUSH YZ	1	3	11	Загрузка в СТЕК содержимого пары регистров	[SP–1] ← YZH; [SP–2] ← YZL; SP← (SP–2); Содержимое старшего (H) регистра из пары YZ пересылается в ЯП, адрес которой на 1 меньше содержимого указателя СТЕКа SP. Содержимое младшего (L) регистра пары YZ пересылается в ЯП, адрес которой на 2 меньше содержимого SP. Содержимое SP уменьшается на 2.
PUSH PSW	1	3	11	Загрузка в СТЕК слова состояния программы	[SP-1] ← A; [SP-2] ← (*d*0 *d*1 *d*2 *d*3 *d*4 *d*5 *d*6 *d*7), где *d*0 ← C; *d*1 ← 1; *d*2 ← P; *d*3 ← 0; *d*4 ← AC; *d*5 ← 0; *d*6← Z; *d*7← S; SP ← (SP-2); Содержимое A пересылается в ЯП, адрес которой на 1 меньше содержимого указателя СТЕКа SP. Содержимое флагов объединяется в слово состояния, это слово пересылается в ЯП, адрес которой на 2 меньше содержимого указателя СТЕКа. Содержимое указателя СТЕКа уменьшается на 2.
POP YZ	1	3	10	Считывание из СТЕКа содержимого в пару регистров	YZL ← [SP]; YZH← [SP+1]; SP ← (SP+2); Содержимое ЯП, адрес которой определяется содержимым указателя СТЕКа SP, пересылается в младший регистр (L) пары регистров YZ. Содержимое ЯП, адрес которой на единицу больше содержимого указателя СТЕКа, пересылается в старший (H) регистр пары YZ. Содержимое указателя СТЕКа увеличивается на 2.
POP PSW	1	3	10	Считывание из СТЕКа слова состояния программы	[SP] → (*d*0 *d*1 *d*2 *d*3 *d*4 *d*5 *d*6 *d*7), где C ← *d*0; P ← *d*2; AC ← *d*4; Z ← *d*6; S ← *d*7; A ← [SP+1]; SP ← (SP+2); Содержимое ЯП, адрес которой определяется содержимым SP, используется для восстановления состояния. Содержимое ЯП, адрес которой на единицу больше содержимого SP, пересылается в A. Содержимое SP увеличивается на 2.

Продолжение таблицы Б

Мнемокод	Байты	Циклы	Такты	Выполняемая операция	Комментарий
Команды ввода-вывода и управления					
IN N	2	3	10	Ввод	A ← d8; Данные, имеющиеся на 8 битах двунаправленной шины данных указанного порта, пересылаются в A.
OUT N	2	3	10	Вывод	d8 ← A; Содержимое A помещается на двунаправленную шину данных для передачи в указанный порт.
RST n *здесь* n *подразумевается кодом команды и может быть равно 1,2,…,7*	1	3	11	Рестарт	[SP–1]←PCH; [SP–2]← PCL; SP←(SP–2); PC ← 8*(nnn); Старшие 8 бит адреса следующей команды пересылаются в ячейку памяти, адрес которой на единицу меньше содержимого указателя СТЕКа. Младшие 8 бит адреса следующей команды пересылаются в ЯП, адрес которой на 2 меньше содержимого указателя СТЕКа. Содержимое указателя СТЕКа уменьшается на 2.Управление передается команде, адрес которой соответствует коду NNN, умноженному на 8.
EI	1	1	4	Разрешение прерываний	Прерывания разрешаются при выполнении следующих за EI команд.
DI	1	1	4	Запрещение прерываний	Прерывания запрещаются при выполнении следующих за DI команд.
NOP	1	1	4	Нет операции	В счетчик PC заносится адрес, следующей команды, и микропроцессор переходит к ее обработке. Состояние регистров не изменяется.
HLT	1	1	5	Останов	В счетчик PC заносится адрес следующей команды. Процессор затем бездействует до прихода команды прерывания. Состояние регистров не изменяется.

ПРИЛОЖЕНИЕ В

МАШИННЫЕ КОДЫ КОМАНД МИКРОПРОЦЕССОРА

Машинные коды команд микропроцессора КР580 определяются из таблицы, представленной ниже. Первая часть однобайтового кода команды – это номер строки (от «0» до «F»), вторая часть – номер столбца (от «0» до «F») соответствующей команды. Например, код команды *MOV D,H* равен «54*h*».

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	*NOP*	*LXI* B,d16	*STAX* B	*INX* B	*INR* B	*DCR* B	*MVI* B,d8	*RLC*	-	*DAD* B	*LDAX* B	*DCX* B	*INR* C	*DCR* C	*MVI* C,d8	*RRC*
1	-	*LXI* D,d16	*STAX* D	*INX* D	*INR* D	*DCR* D	*MVI* D,d8	*RAL*	-	*DAD* D	*LDAX* D	*DCX* D	*INR* E	*DCR* E	*MVI* E,d8	*RAR*
2	-	*LXI* H,d16	*SHLD* adr	*INX* H	*INR* H	*DCR* H	*MVI* H,d8	*DAA*	-	*DAD* H	*LHLD* adr	*DCX* H	*INR* L	*DCR* L	*MVI* L,d8	*CMA*
3	-	*LXI* SP,d16	*STA* adr	*INX* SP	*INR* M	*DCR* M	*MVI* M,d8	*STC*	-	*DAD* SP	*LDA* adr	*DCX* SP	*INR* A	*DCR* A	*MVI* A,d8	*CMC*
4	*MOV* B,B	*MOV* B,C	*MOV* B,D	*MOV* B,E	*MOV* B,H	*MOV* B,L	*MOV* B,M	*MOV* B,A	*MOV* C,B	*MOV* C,C	*MOV* C,D	*MOV* C,E	*MOV* C,H	*MOV* C,L	*MOV* C,M	*MOV* C,A
5	*MOV* D,B	*MOV* D,C	*MOV* D,D	*MOV* D,E	*MOV* D,H	*MOV* D,L	*MOV* D,M	*MOV* D,A	*MOV* E,B	*MOV* E,C	*MOV* E,D	*MOV* E,E	*MOV* E,H	*MOV* E,L	*MOV* E,M	*MOV* E,A
6	*MOV* H,B	*MOV* H,C	*MOV* H,D	*MOV* H,E	*MOV* H,H	*MOV* H,L	*MOV* H,M	*MOV* H,A	*MOV* L,B	*MOV* L,C	*MOV* L,D	*MOV* L,E	*MOV* L,H	*MOV* L,L	*MOV* L,M	*MOV* L,A
7	*MOV* M,B	*MOV* M,C	*MOV* M,D	*MOV* M,E	*MOV* M,H	*MOV* M,L	*HLT*	*MOV* M,A	*MOV* A,B	*MOV* A,C	*MOV* A,D	*MOV* A,E	*MOV* A,H	*MOV* A,L	*MOV* A,M	*MOV* A,A
8	*ADD* B	*ADD* C	*ADD* D	*ADD* E	*ADD* H	*ADD* L	*ADD* M	*ADD* A	*ADC* B	*ADC* C	*ADC* D	*ADC* E	*ADC* H	*ADC* L	*ADC* M	*ADC* A
9	*SUB* B	*SUB* C	*SUB* D	*SUB* E	*SUB* H	*SUB* L	*SUB* M	*SUB* A	*SBB* B	*SBB* C	*SBB* D	*SBB* E	*SBB* H	*SBB* L	*SBB* M	*SBB* A
A	*ANA* B	*ANA* C	*ANA* D	*ANA* E	*ANA* H	*ANA* L	*ANA* M	*ANA* A	*XRA* B	*XRA* C	*XRA* D	*XRA* E	*XRA* H	*XRA* L	*XRA* M	*XRA* A
B	*ORA* B	*ORA* C	*ORA* D	*ORA* E	*ORA* H	*ORA* L	*ORA* M	*ORA* A	*CMP* B	*CMP* C	*CMP* D	*CMP* E	*CMP* H	*CMP* L	*CMP* M	*CMP* A
C	*RNZ*	*POP* B	*JNZ* adr	*JMP* adr	*CNZ* adr	*PUSH* B	*ADI* d8	*RST* 0	*RZ*	*RET*	*JZ* adr	-	*CZ* adr	*CALL* adr	*ACI* d8	*RST* 1
D	*RNC*	*POP* D	*JNC* adr	*OUT* N	*CNC* adr	*PUSH* D	*SUI* d8	*RST* 2	*RC*	-	*JC* adr	*IN* N	*CC* adr	-	*SBI* d8	*RST* 3
E	*RPO*	*POP* H	*JPO* adr	*XTHL*	*CPO* adr	*PUSH* H	*ANI* d8	*RST* 4	*RPE*	*PCHL*	*JPE* adr	*XCHG*	*CPE* adr	-	*XRI* d8	*RST* 5
F	*RP*	*POP* PSW	*JP* adr	*DI*	*CP* adr	*PUSH* PSW	*ORI* d8	*RST* 6	*RM*	*SPHL*	*JM* adr	*EI*	*CM* adr	-	*CPI* d8	*RST* 7

Здесь в таблице используются следующие условные обозначения:

«d16» – константа, задаваемая в команде, размером 16 бит;

«adr» – константа, задаваемая в команде, размером 16 бит (адрес);

«d8» – константа, задаваемая в команде, размером 8 бит;

«N» – номер порта ввода-вывода.

СОДЕРЖАНИЕ

ПРИЛОЖЕНИЕ В

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ 7 СЕМЕСТРА

ОГЛАВЛЕНИЕ

ТЕМА № 1. ОБЩЕЕ УСТРОЙСТВО МИКРОКОНТРОЛЛЕРОВ AVR

Вопросы, рассматриваемые в данной теме:

1. Семейства AVR
2. Структурная схема МК
3. Типы памяти, используемые в МК
4. Основные периферийные устройства

Краткие теоретические сведения

1. Краткие сведения о МК AVR

Микроконтроллеры (МК) AVR представляют собой достаточно несложные в изучение 8-разрядные RISK МК, это связано не только с тем, что набор команд RISK МК сам по себе прост, но и с тем, что данные МК уже достаточно давно на рынке. Вследствие чего они достаточно хорошо описаны и документированы, в том числе и на русском языке. Так же для МК данной архитектуры любителями было создано множество устройств, схемы и программный код, которых можно найти в сети Internet, что тоже в свою очередь может помочь в освоении МК.

Кроме простоты в изучении и хорошей документированности данные МК легкодоступны для покупки, легко прошиваются и обладают преемственной архитектурой, то есть при смене МК, ранее написанный код нуждается лишь в небольшой доработке. *Далее в работе, если не указано иное под МК подразумевается МК архитектуры AVR.*

2. Основные семейства МК AVR

Существуют четыре основных семейства МК:

1. МК семейства Classic (AT90S*xxx*), данное семейство МК уже не выпускается, а самая популярная модель AT90S2313, была заменена на ATtiny 2313.

2. МК семейства Tiny (ATtiny*xx*), самые простые с точки зрения внутреннего устройства МК, в основном размещаются в корпусах с 8-20 выводами,

имеют небольшую память программ объемом 1-8 Кб, аппаратный стек и используются в простых и дешевых устройствах, а также как вспомогательные контроллеры.

3. МК семейства Mega (Mega*xx*) обладают большим объемом памяти программ (8-256Кб), и корпусами с 28-100 выводами, по сравнению с МК семейства Tiny имеют более богатую периферию и вследствие этого большую функциональность. Применение МК Mega хватает для решения практически всех не узкоспециализированных задач.

4. МК семейства XMega характеризуются повышенным быстродействием, еще большими объемами памяти программ и другими улучшениями. В связи с появлением более дешевых 32-разрядных МК архитектур ARM-Cortex M3 с теми же функциональными возможностями, семейство XMega практически не используется, поэтому *в данной работе не рассматриваются.*

Кроме вышеперечисленных семейств существуют специализированные МК основанные на ядре AVR, например, для работы с ЖК-дисплеями, USB-интерфейсом, CAN-интерфейсом, аккумуляторами. Это не значит, что данный функционал нельзя реализовать на стандартных МК, но зачастую использование специализированного МК ускоряет и упрощает разработку целевого устройства.

3. Структурная схема МК AVR

Рассмотрим структурную схему МК (рис. 1.1) и кратко разберем основной функционал входящих в ядро блоков:

1. Регистры общего назначения (регистровый файл). Ядро МК содержит 32 8-рязрядных регистра общего назначения, которые связанны непосредственно с арифметико-логическим устройством (ALU), это дает возможность напрямую обращаться к регистрам МК для выполнения над ними различных операций (например, сложения, сравнения и т.п.) напрямую, без переноса значения в аккумулятор. РОН входящие в состав МК можно разделить на три группы:

Младшие R0…R15 – не могут быть использованы в операциях непосредственно с операндом (SBR, CBR, LDI и другие), это особенность архитектуры.

Старшие R16…R31– работают со всеми командами без исключения.

Индексные R26…R31 – помимо использования как обычные регистры, могут объединяться в регистровые пары X(R26:R27), Y(R28:R29), Z(R30:R31) использующиеся в виде указателей для работы с памятью.

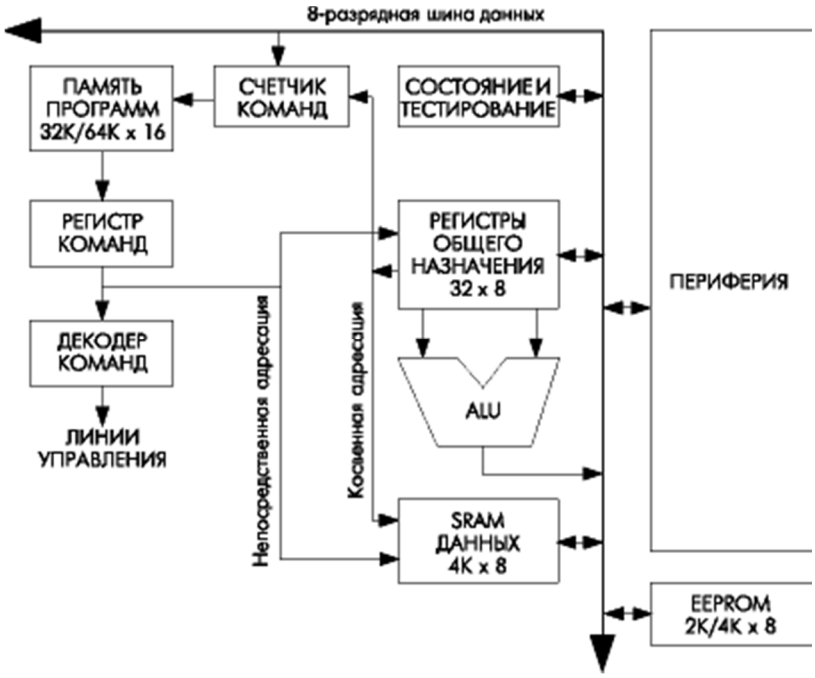


Рис. 1.1. Общая структурная схема МК

2. Арифметико-логическое устройство (ALU). Используется для обработки всей математики в МК, доступны команды сложения, вычитания, сравнения, сдвигов, логических операций, переносов, умножения. Команда деления отсутствует. Так как ALU напрямую подключен к РОН поддерживаются арифметические, логические операции и битовые операции, как между регистрами, так и между регистром и константой.

3. Счетчик команд и декодер команд. По умолчанию при старте МК значение программного счетчика указывает на адрес первой команды в памяти программ ($0000). Микроконтроллер выбирает из адреса два байта (код команды и аргумент) и передает их на выполнение в декодер команд. Счетчик команд инкрементируется, команда выполняется, и происходит выборка очередной команды.

Если же встречается команда перехода, в счетчик команд загружается адрес, указанный в команде (абсолютный переход), либо его значение увеличивается не на единицу, а на необходимое значение (команды проверки-пропуска) и в следующем такте команда будет взята с нового адреса и выполнена.

Далее рассмотрим память, в структуре AVR имеются три разновидности:

1. Память программ (Flash-память). Данный тип памяти, как и любая другая flash-память имеет страничную организацию, причем размер страницы зависит от модели и составляет от 64 до 256 байт. Программирование страницы памяти возможно только целиком.

Память программ можно считать построенной из отдельных ячеек – слов по два байта каждое. Страничная организация памяти программ обусловлена тем, что подавляющее большинство команд в AVR имеют длину ровно 2 байта. Исключение составляют некоторые команды переходов (JMP, CALL) и команды работы с памятью (LDS) оперирующие с 16 и более разрядными адресами, длина таких команд составляет 4 байта. Такие команды используются в моделях МК с памятью программ свыше 8 Кб.

2. Память данных (ОЗУ, SRAM). К данному типу памяти в отличие от вышерассмотренной памяти программ адресация происходит линейно, без какого-либо деления на страницы или сегменты. Многие младшие МК семейства Tiny памяти данных не имеют, ограничиваясь наличием РОН. Многие старшие модели семейства Mega позволяют подключить дополнительную память объемом до 64 Кб с параллельным интерфейсом.

Рис 1.2. Адресное пространство SRAM

Разделение адресного пространства памяти SRAM представлено на рисунке 1.2.

В связи с тем, что операции чтения/записи одинаково работают со всеми адресами доступного пространства памяти SRAM, нужно быть внимательным, так как вместо SRAM можно попасть в один из РОН или регистр ввода/вывода (РВВ), например, команда загрузки значения регистра R0 в регистр R16 (MOV R0, R16) равносильна записи регистра R16 по нулевому адресу (STS $0000, R16). Такой прямой способ запись значения в регистр не очень удобен, так как занимает два такта вместо одного, но иногда это позволяет обойти ограничения записи в некоторые регистры.

3.Энергонезависимая память (EEPROM). Данный тип памяти используется для хранения констант и данных при отключении питания, ее объем обычно варьируется в пределах от 64 байт у младших моделей до 4 Кб у старших. Одно из отличий от Flash-памяти при аналогичной страничной организации (до 4Кб страница) это возможность выборочного программирования побайтно. Чтение из памяти осуществляется в течение одного машинного цикла, а вот процесс записи проходит более медленно (от 2 до ~4 мс), время записи зависит от встроенного в МК тактового RC-генератора, частота которого не стабильна, и зависит от напряжения питания.

4. Основные периферийные устройства

Кратко рассмотрим основные периферийные устройства (более подробно с практическими примерами наиболее используемые устройства будут рассмотрены в следующих темах), входящие в состав МК AVR, наиболее популярные из них, такие как таймеры, порт UART, аналоговый компаратор и сторожевой таймер имеется практически во всех моделях МК. Несмотря на то, что при написании программы не обязательно использовать все устройство, при выборе оптимального МК стоит учитывать, что даже незадействованное устройство потребляет энергию. Адресация всех периферийных устройств происходит через РВВ, так как прямая модификация регистров РВВ ограничена, то перенос значений РВВ в РОН и обратно осуществляется с помощью команд IN и OUT.

1. Порты ввода/вывода. Используются для обмена данными с внешними устройствами (от кнопки и лампочки до ЖКИ), в различных моделях может быть от 1 до 7 портов, номинально каждый порт 8 разрядный, то есть на один порт приходится 8 линий ввода/вывода, но в некоторых МК количество линий на порт может быть и меньше. Порты обозначаются буквами A, B, C и далее. Так же для сокращения контактов корпуса выводы МК, соответствующие портам ввода вывода, несут дополнительную функцию. Например, на рисунке 1.3 представлен МК AtMega8 в корпусе PDIP у которого 3 порта ввода/вывода (A, B, C), причем один порт имеет только 7 линий ввода вывода, и все порты МК несут альтернативный функционал, например, линия 3 порта D является так же линией внешнего прерывания.

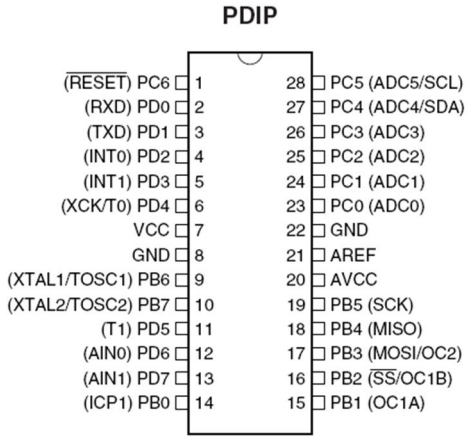


Рис.1.3. МК AtMega8

2. Таймеры-счетчики. Данные периферийные устройства имеют очень широкую область применения от подсчета времени до работы в режимах ШИМ. Базовая частота таймеров равняется частоте МК, но может быть поделена на 8, 64, 256 или 1024, так же возможно тактирование таймеров от внешнего сигнала. Таймеры бывают двух разновидностей 8 и 16 разрядными, причем 8-разрядные таймеры имеют номера 0 и 2, а 16 разрядные 1 и 3. Один из 8-разрядных счетчиков в большинстве МК может работать в асинхронном режиме от отдельного тактового генератора, что позволяет, например, реализовать часы. На рисунке 1.3. таймеры представлены выводами T0, T1, OC1A, OC1B и другими.

Так же во всех без исключения МК существует сторожевой таймер (Watchdog) предназначенный для вывода МК из режимов энергосбережения или перезапуска МК в случае его зависания.

3. Аналогово-цифровой преобразователь (АЦП). Входит в большинство моделей МК, обычно число каналов равно 8, разрядность АЦП 10 бит. Служит

для преобразования аналоговых сигналов в цифровые, может работать как в режиме одиночного, так и непрерывного преобразования. На рисунке 1.3. АЦП представлен выводами ADC0-ADC5.

Кроме вышеперечисленных устройств в МК аппаратно реализованы протоколы последовательной передачи данных, такие как *UART, SPI, TWI (I^2C), USI.* Основное преимущество последовательных портов перед параллельными – снижение числа соединений, а также более стабильная работа на высоких скоростях. Стоит сказать, что практически любой порт как последовательный, так и параллельный можно имитировать программно, например, можно реализовать аппаратно не поддерживаемый МК AVR протокол 1-Wire.

Задания для самостоятельной работы

На основании выданных преподавателем индивидуального задания выполните необходимые исследования.

Составьте отчет по выполненным заданиям

ТЕМА № 2. ТАКТИРОВАНИЕ,СБРОС И ПОДКЛЮЧЕНИЕ МК AVR

Вопросы, рассматриваемые в данной теме:

1. Способы тактирования МК
2. Сброс МК
3. Подключение питания к МК

Краткие теоретические сведения

1. Способы тактирования МК

В большинстве случаев, когда не требуется высокая стабильность частоты работы контроллера (например, в системе не используется таймеры для отсчета точных временных интервалов) используется встроенный RC-генератор, способный работать на частотах приблизительно равных 1, 2, 4 и 8 МГц.

Если же для работы устройства требуется отличная от этих величин частота, но требований к стабильности частоты так же нет, то можно подключить внешнюю RC цепочку (рис.2.1, в). В такой схеме емкость C1 не должна быть менее 22 пФ, а резистор выбирается из диапазона 3.3-100кОм, с учетом того что частота генератора определяется по формуле F=2/3RC. Так же МК позволяет вместо конденсатора C1 подключить встроенный конденсатор емкостью 36 пФ, записав 0 в конфигурационную ячейку CKPOT.

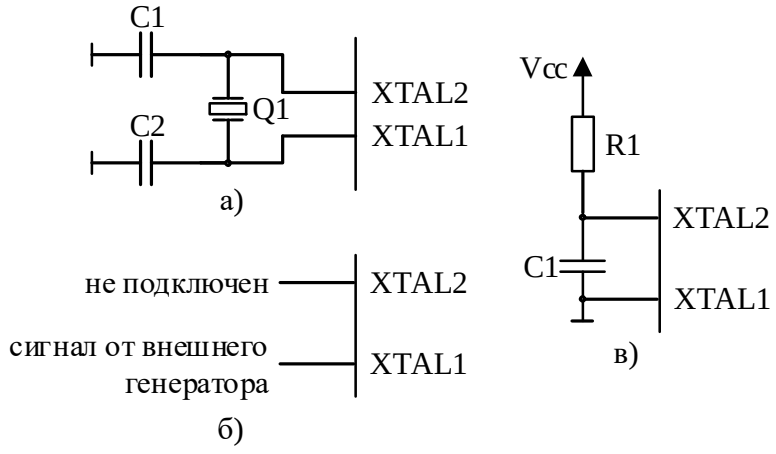


Рис.2.1. Способы тактирования МК

Если же устройство требовательно к стабильности тактовой частоты (например, используется протокол UART, или протокол, основанный на выдержке четких интервалов времени, такой как 1-Wire), то одним из решений будет подключение внешнего кварцевого резонатора к соответствующим выводам (рис.2.1, а). Емкость конденсаторов C1 и C2 должна составлять 15-22пФ.

Вторым решением будет являться тактирование МК от внешнего генератора (рис. 2.1, б), такой режим тактирования удобен в случаях, если необходимо синхронизировать МК с внешними компонентами, или запустить МК на нестандартной очень точной частоте тактирование.

По умолчанию МК семейств Tiny и Mega настроены на частоту работы от встроенного тактового генератора на частоте 1 МГц (CSEL=0001) , следовательно, для других режимов работы необходимо перенастроить конфигурационные биты МК в соответствии с таблицей 2.1.

Таблица 2.1. Установки конфигурационных ячеек CKSEL в зависимости от режимов тактирования

CKSEL3…0	Источник тактирования	Частота
0000	Внешняя частота	0…16 МГц
0001	Встроенный RC-генератор	1 МГц
0010	Встроенный RC-генератор	2 МГц
0011	Встроенный RC-генератор	3 МГц
0100	Встроенный RC-генератор	8 МГц
0101	Внешняя RC-цепочка	<0.9 МГц
0110	Внешняя RC-цепочка	0.9…3.0 МГц
0111	Внешняя RC-цепочка	3.0…8.0 МГц
1000	Внешняя RC-цепочка	8.0…12 МГц
1001	Низкочастотный резонатор	32.768 кГц
101x	Кварцевый резонатор	0.4…0.9 МГц
110x	Кварцевый резонатор	0.9…3.0 МГц
111x	Кварцевый резонатор	3.0…8.0 МГц
1xxx (CKPOT=0)	Кварцевый резонатор	>1.0 МГц

При конфигурировании МК всегда необходимо быть очень внимательным в связи с тем, что, неправильно выставив биты МК его нельзя будет перепрограммировать до установки выбранного источника частоты.

Более подробно о конфигурационных битах так же называемых FUSE битами будет рассказано далее.

2. Сброс МК

Сброс (RESET) это установка МК в начальный режим работы, при этом РВВ устанавливаются в состояние по умолчанию, а РОН могут принимать случайные значения, поэтому используемые в РОН переменные всегда необходимо инициализировать значениями.

Источниками сброса могут быть следующие события:

1. Включение МК

2. Аппаратный сброс, то есть подачи сигнала низкого уровня на вход /RESET.

3. Срабатывание сторожевого таймера.

Во многих МК на выводе /RESET присутствует подтягивающий резистор номиналом (100-500кОм) на котором могут наводиться помехи, приводящие к непредсказуемым сбросам МК, для избегания этого рекомендуется подключение внешнего резистора величиной 2-5кОм от вывода /RESET к напряжению питания. Так же для более надежной работы МК рекомендуется установить конденсатор 0.1-0.5 мкФ от вывода /RESET на «землю» (рис.2.2).

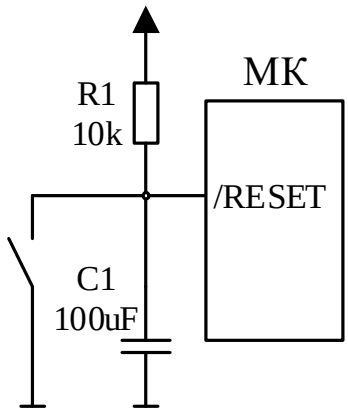


Рис.2.2. Подключение вывода /Reset

3. Питание МК

МК AVR работают от напряжения питания в пределах 1.8-5В в зависимости от серии и модели. Напряжение питания обычно обозначается как Vcc, земля или минус питания GND, перед подключением МК к источнику питания всегда

рекомендуется уточнить допустимый диапазон питания в документации на МК. Так стоит обратить внимание на то, что возможность работы МК на определённых частотах зависит от величины питание, чем она больше, тем на больших частотах сможет работать МК. Если в МК больше чем один вывод Vcc и GND, то необходимо запитать их все. Так же в большинстве МК присутствуют выводы AGND и AVCC, аналоговая земля и аналоговое питание соответственно которые питают АЦП контроллера, таким образом, если использовать АЦП не планируется, то его можно запитать тем же напряжением что и Vcc и GND, в противном случае АЦП желательно запитать через дополнительные фильтры (например, дроссель), чтобы помехи, возникающие в основной цепи не влияли на результаты измерений. Полная схема подключения МК к питанию приведена на рисунке 2.3.

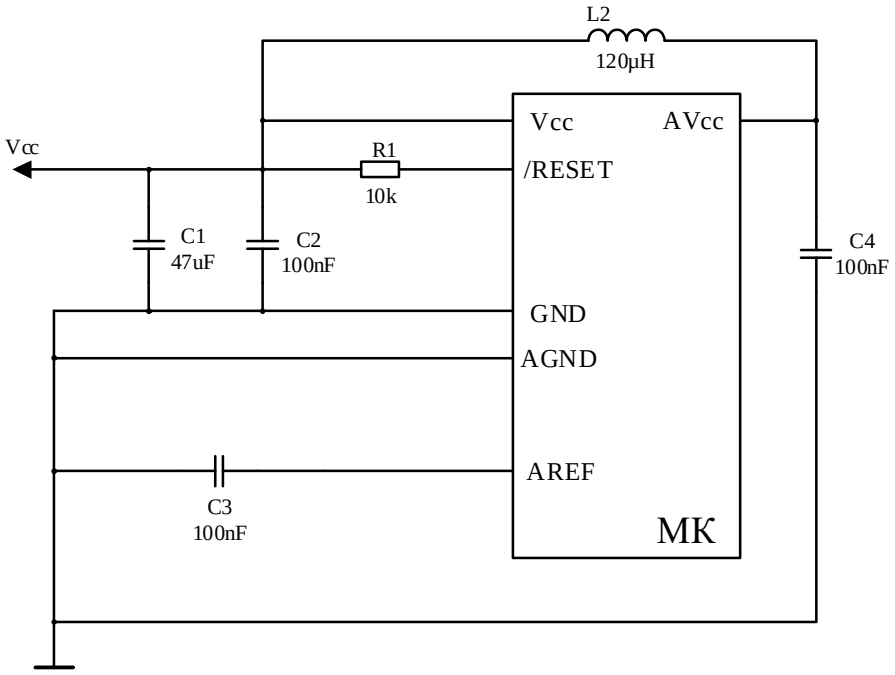


Рис.2.3. Подключение МК к питанию

Как видно из рисунка в дополнение к вышесказанному были добавлены два конденсатора в цепь питания, один из них емкостью 47мкФ сглаживает броски напряжения, конденсатор емкостью 100нФ сглаживает краткие импульсные помехи в шине питания, вызванные работой цифровых схем. Так же через конденсатор емкостью 100нФ подключен вход AREF МК, это вход опорного

напряжения МК относительно которого свои значения будет рассчитывать АЦП, использование конденсатора несколько улучшит качество опорного напряжения.

Задания для самостоятельной работы

На основании выданных преподавателем индивидуального задания выполните необходимые исследования.

Составьте отчет по выполненным заданиям

ТЕМА № 3. ОБЩИЕ ПРИНЦИПЫ ПРОГРАММИРОВАНИЯ МК

Вопросы, рассматриваемые в данной теме:

1. Atmel Studio
2. Команды инструкции и нотации AVR-ассемблера
3. Прерывания
4. Структура AVR программ
5. Регистр SREG, Отладка и компиляция AVR программ
6. Конфигурационные биты (Fuse биты)

Краткие теоретические сведения

1. Atmel Studio

При написании каких-либо программ всегда встает вопрос в выборе языка, на котором программа будет писаться. При изучении микроконтроллеров обычно используется язык ассемблера, как известно ассемблер — это не универсальный язык программирования, это просто несколько правил, по которым последовательность команд процессора, записанных в мнемоническом виде, может объединяться в программу. Эта последовательность команд затем компилируется с помощью собственно ассемблера, который переводит мнемоники команд в форму пригодную для записи в МК. Одной из причин, по которой свои первые программы под МК рекомендуется писать на ассемблере является то, что это позволяет более подробно разобраться в его архитектуре, и в дальнейшем при написании сложных программ на языках высокого уровня будет полное понимание как что работает, что зачастую позволяет найти ошибки в программе и написать архитектурно более «правильный код». Еще одна причина, это простота ассемблера, его фирменное описание занимает всего несколько страниц, а команды понятны и легко запоминаются. Ассемблера обладает одним большим недостатком: программы получаются большими, и не всегда удобочитаемыми, так же многие математические операции, такие как деление, операции с 16 разрядными

числами отдаются на откуп программиста, а не компилятора и встроенных библиотек высокого уровня.

Как было сказано выше, для того чтобы получить последовательность команд и данных в двоичном формате допустимую для загрузки в МК, написанную программу нужно скомпилировать. Написание, отладка и компиляция программ под МК AVR обычно выполняются в среде разработки Atmel Studio (ранее AVR Studio) последнюю версию которого можно скачать на официальном сайте. Данный программный продукт позволяет создавать программы под все МК производства фирмы Atmel как на языке ассемблера, так и на С.

Рассмотрим создание простейшего проекта программы на языке ассемблера. При запуске Atmel Studio появляется окно (рис.3.1):

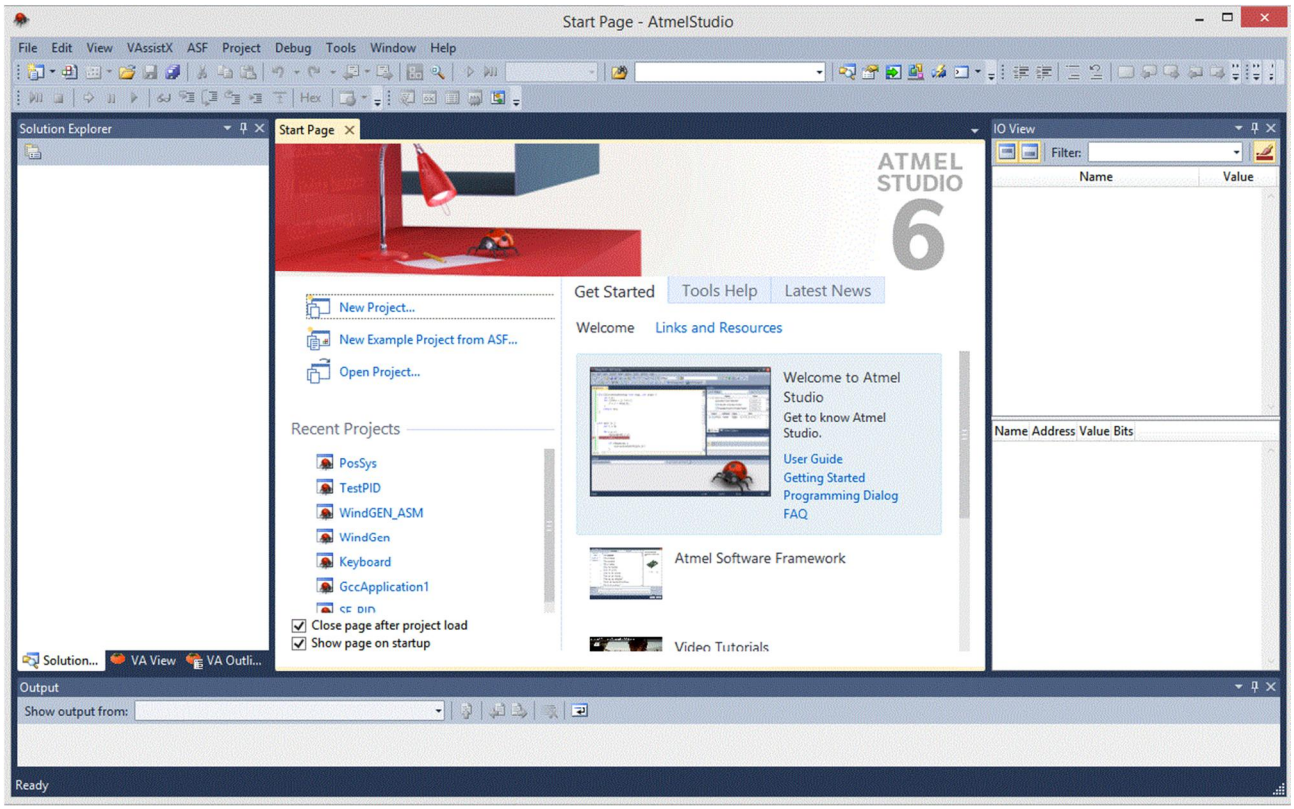


Рис.3.1. Главное окно Atmel Studio

Стартовое окно сразу предлагает создать новый проект, открыть старый, и показывает недавно открывавшиеся проекты. Создадим новый проект, нажав кнопку New Project (Так же новый проект можно создать через меню File-New-Project).

Откроется окно нового проекта (рис.3.2.). В котором предлагается на выбор создание проекта на языках C/C++ с выбором под какую готовую платформу пишется код, или Ассемблер, а также предлагается дать имя проекту, и выбрать его местоположение (не рекомендуется названия и использовать пути к проекту содержащие кириллицу, проект может не компилироваться). После выбора необходимых параметров (Assembler), для перехода к следующему окну создания проекта щелкаем OK.

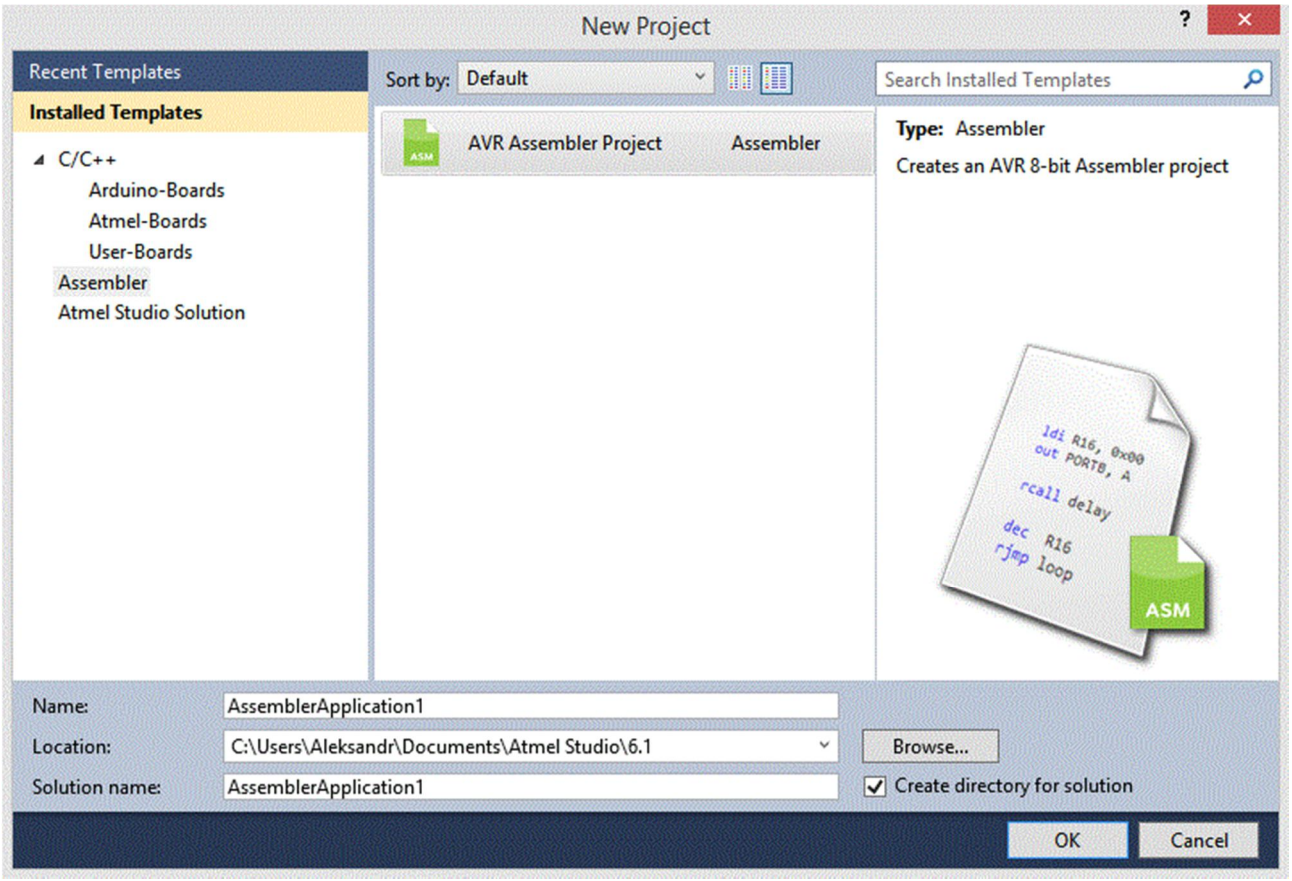


Рис.3.2. Окно нового проекта

Следующее окно (рис.3.3) – окно выбора МК, в этом окне можно отсортировать семейства (выпадающий список Device Family), так и найти нужный МК в строке поиска, для примера выбран МК AtMega8. В разделе со списком МК даны их базовые характеристики: объем памяти программ, данных и EEPROM. В разделе Device Info приведены: имя МК, его напряжение питания, тип семейства, дана ссылка на техническое описание МК (Datasheet), а также представлены

поддерживаемые данным МК отладчики. Программный отладчик, используемый в Atmel Studio, называется Simulator. Выбранный МК является избыточным для решения большинства простых задач, но очень удобен для примеров, так как содержит всю необходимую периферию. После выбора МК для перехода к следующему окну необходимо нажать кнопкуOK.

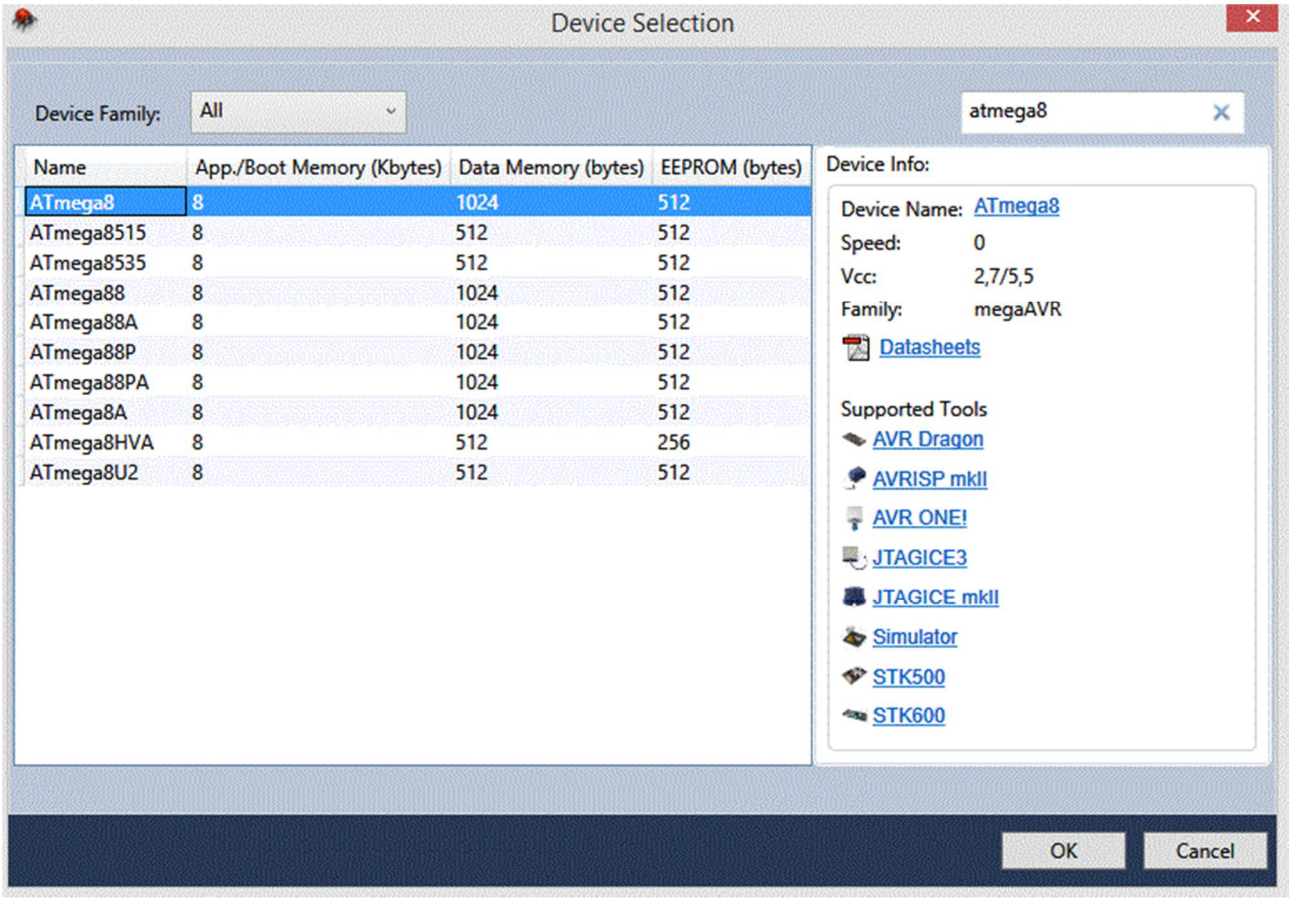


Рис.3.3. Окно выбора МК

Откроется окно среды разработки (рис.3.4), в котором можно приступать к написанию программы. Рассмотрим его основные составляющие, слева располагается Solution Explorer, путеводитель по вашему проекту, в разделе Dependencies можно увидеть все дополнительно подключенные к проекту библиотеки, в разделе Labels - метки переходов, в Output Files – получаемые после компиляции выходные файлы, и последним пунктом идет файл .asm с названием созданного проекта. Слева находится панель I/O View в которой можно увидеть абсолютно все устройства МК, а также их регистры с описанием за что какой отвечает (если

оно отсутствует включить его можно из меню Debug-Windows- I/O View), данную панель удобно использовать не только при отладке, но и при написании программы для того чтобы посмотреть название регистра и его назначения не открывая справочник. Снизу находится панель выходных данных, в которой при компиляции программы буду появляться ошибки, предупреждения, отчеты об успешной компиляции и другие сообщения.

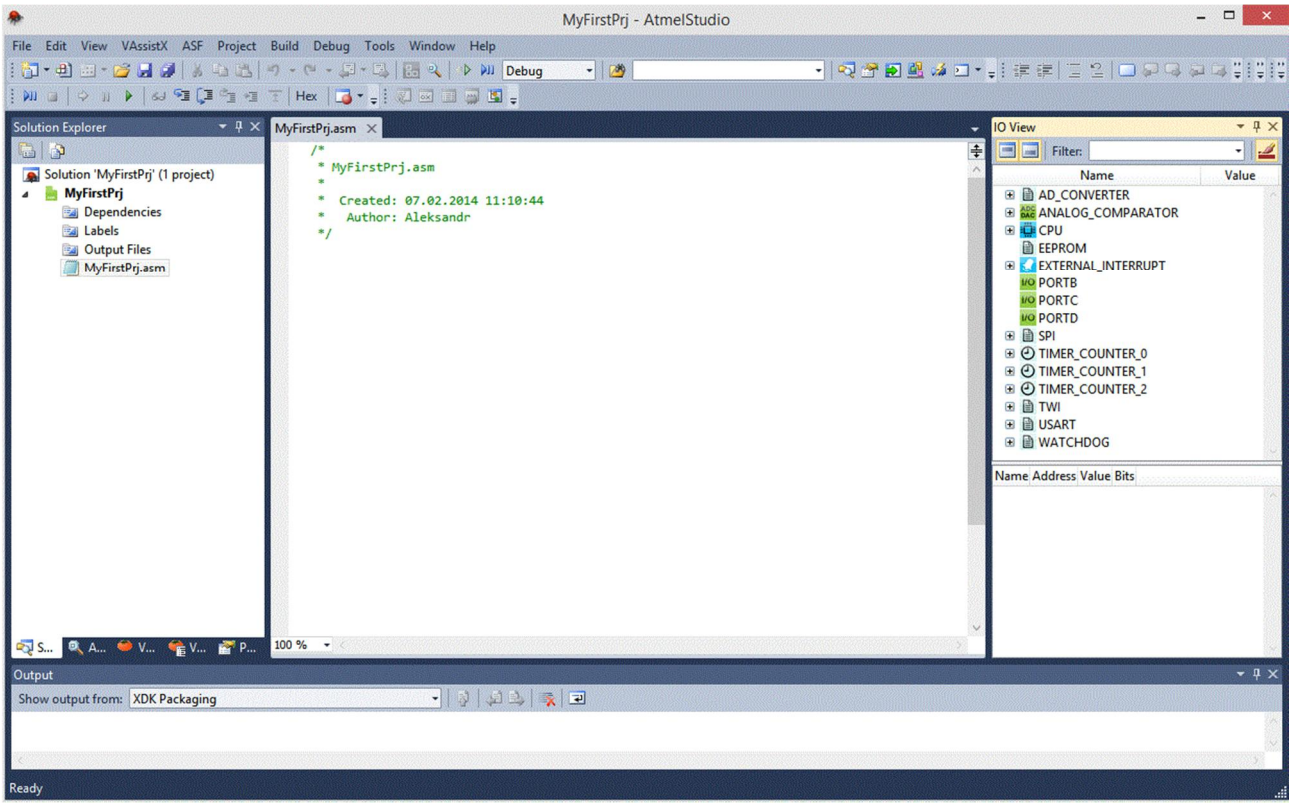


Рис.3.4. Окно среды разработки

Среда Atmel Studio очень гибко и удобно настраивается, и ее первоначальный вид может немного отличаться от представленного на рисунке. Так же внешний вид среды при написании программы и ее отладке тоже несколько отличаются.

2. Команды инструкции и нотации AVR-ассемблера, прерывания

Перед тем как перейти непосредственно к написанию программы рассмотрим некоторые базовые понятия ассемблера, особенности записи команд, а также разберем, что такое прерывания и как проходит их обработка.

Начнем с особенности записи команд в ассемблере, сначала идет команды, которая может быть двух-, трех- или четырехбуквенная, затем через пробел (или через несколько, количество пробелов не влияет на исполнение команды) идут операнды. Есть команды, не имеющие операндов (LPM, RET), другие команды имеют только один операнд (DEC R16). В командах, которым необходимо два операнда используется «прямая польская запись», то есть сначала указывается приемник, затем источник. Операнды разделяются запятой. Например, команда ADD R16, R17 сложить значения указанных регистров и результат операции поместит в регистр R16. Каждая команда должна занимать отдельную строку, кроме команды, строка может содержать метки и комментарии (выделяются символом ; или //). Регистр букв ассемблером не различается и значения не имеет, то есть команда RJMP LABEL и rjmp label идентичны.

Далее о метках, команда из примера выше означает, что будет совершен относительный переход на метку LABEL, в коде программы метка, на которую будет совершаться переход обозначается как произвольно придуманное имя с двоеточием на конце (LABEL:), после метки может быть расположена какая-либо инструкция.

Во многие команды, работающие с константами можно включать числа и выражения, все числа по умолчанию считаются десятичными, за исключением чисел с ведущим нулем (06), которым обозначаются восьмеричные числа. Двоичные числа записываются в виде: 0b00101100, шестнадцатеричные числа могут быть записаны как 0xAA, так и $AA.

Стоит обратить внимание на то, что адреса в программе являются числовыми константами и запись вида RJMP LABEL+1 допустима и будет интерпретирована как перейти на следующую команду после той, на которую ссылается метка.

Кроме операции сложения с константами можно выполнять любые математические и логические операции, одной из часто употребляемых является операция сдвига влево (<<), данная используется для установки определённых именованных битов регистров (для удобства все биты PBB имеют имена), вместе с

операцией побитовое «ИЛИ» (|), в одном регистре можно установить сразу несколько битов, например:

```
LDI R16, (1<<CS00)|(1<<CS01)
OUT TCCR0, R16
```

То есть сначала в один из регистров МК (напрямую записывать константы в РВВ нельзя) записывается необходимые значения регистров в данном примере отвечающие за предделитель таймера 0, то есть биты CS00 и CS01 устанавливаются 1, и копируются в конфигурационный регистр таймера 0.Так происходит настройка всех устройств МК.

Часто в программах кроме команд используются различные директивы компилятора, самые распространённые из них это: .def, .equ, #include. Первая директива используется для именования регистров:

```
.def temp=R16 ;теперь R16 является переменной temp
```

Хорошим тоном считается именовать используемые РОН, так как это упрощает чтение и понимание логики программы.

Директива .equ применяется для именования констант:

```
.equ min_value = 5  ; Минимальное значение = 5
.equ max_value = 10 ; Максимальное значение = 10
```

Именование и описание констант, используемых в программе очень важная возможность, которой не стоит пренебрегать, позволяет избегать «волшебных чисел», констант в командах которые не говорят о том, что это за константа совершенно ничего. Так же с помощью директивы .equ можно определять довольно сложные выражения, так например, чтобы вручную не рассчитывать значения констант UART для скорости обмена можно использовать такую конструкцию:

```
.equ   XTAL = 8000000             ;Частота МК
.equ   baudrate = 9600            ;Необходимая скорость передачи
.equ   bauddivider = XTAL/(16*baudrate)-1       ;Расчет константы
```

Помимо вышеперечисленных способов использования .equ, эта директива часто используется для обозначения переменных, которые располагаются в области SRAM, для этого вместо десятичной константы определяется константа в шестнадцатеричном формате равная адресу переменной в памяти.

Изначально ассемблер не знает об именовании регистров, он знает только их физические адреса в памяти, для того чтобы связать мнемонические обозначения с адресами необходимо подключить специальный .inc файл в котором именуются все адреса регистров, данный файл для каждого МК отличается, для его подключения служит директива #include:

#include "m8def.inc"

Данная директива подключает файл с определениями всех констант и адресов для МК AtMega8. #include позволяет подключить абсолютно любой файл, например, какую-либо библиотеку. Файлы библиотек необходимо подключать очень внимательно, так как компилятор вставляет все содержимое файла в месте использования директивы, и, если вставить библиотеку, которая содержит команды, таблица векторов прерываний будет фактически затерта, что может привести к неработоспособности программы. Во избежание этого такую библиотеку необходимо вставить не вначале программы, а после таблицы векторов прерываний.

Дополнительно рассмотрим директивы сегментов памяти .DSEG, .CSEG, .ESEG, они обозначают начало сегмента памяти SRAM, сегмент кода и сегмент EEPROM. Эти директивы важны при создании массивов в памяти МК с помощью директивы .db, они однозначно указывают в каком типе памяти должны размещаться константы, если не указать тип памяти, константы будут сохранены в память программ. Например:

```
N_mask:
.db 1, 3, 5, 7, 9
```

Здесь в память программ (так как не указан тип памяти), по адресу, обозначенному меткой N_mask, сохранен набор целочисленных констант.

Директива .db работает только с памятью программ и EEPROM, для размещения данных в SRAM используется директива .byte которая указывает число резервируемых в памяти байт .

Еще одна часто используема директива - .org позволяет задать компилятору начальный адрес в пределах сегментов кода, данных и EEPROM-памяти, то есть тот адрес, с которого пойдет запись в выбранный тип памяти.

3. Прерывания

Перед рассмотрением структуры программы на языке ассемблера, разберем вопрос использования прерываний. Работа МК это, прежде всего ожидание и обработка событий. Таким образом, аппаратные прерывания являются важной частью МК, которая позволяет автоматически определять наступление какого-либо события, останавливать выполнения основной программы и переходить непосредственно к коду прерывания.

Это происходит следующим образом: при возникновении условия прерывания (например, таймер переполнен) МК выставляет бит в регистре прерываний таймера, если прерывания разрешены контроллер, автоматически вычисляет адрес соответствующего вектора прерывания и переходит к обработке прерывания. Перед переходом к прерыванию МК запрещает все прерывания во избежание срабатывания прерывания во время исполнения текущего, а также сохраняет содержимое счетчика команд в стеке для того чтобы после завершения прерывания продолжить выполнение программы с места где исполнение программы было прервано. По возвращению из прерывания выполнение других прерываний автоматически разрешается. Так как для работы прерываний необходим стек, в моделях МК в которых отсутствует аппаратный стек его необходимо инициализировать сразу после начала выполнения программы:

```
//Инициализация стека: (обязательно во всех МК с программным стеком AtMega)
	LDI R16, LOW(RAMEND)//Загрузка указателя стека в конец SRAM
	OUT SPL, R16
	LDI R16, HIGH(RAMEND)	//Загрузка указателя стека в конец SRAM
	OUT SPH, R16
```

Рассмотрим другие важные моменты, связанные с прерываниями:

1. По умолчанию прерывания запрещены.

2. При возникновении прерывания содержимое регистра флагов SREG не сохраняется, поэтому если он важен для корректного выполнения основной программы, его необходимо сохранить в стек.

3. Кроме глобального разрешения прерываний, каждое прерывание должно быть разрешено индивидуально в специальном регистре прерываний

имеющегося у каждого периферийного устройства способного вызывать прерывания.

4. Вложенные прерывания допустимы, но необходимо учитывать размер стека, так например аппаратный стек в МК Atiny имеет всего три уровня, и при вызове более трех подпрограмм (в том числе и прерываний), контроллер не сможет «вспомнить» по какому адресу он должен продолжить выполнение программы.

5. Если во время прерывания или сразу по выходу из прерывания сработает другое прерывание, переход к выполнению прерывания произойдет только после выполнения одной команды основной программы.

6. Во время выполнения атомарных процедур, то есть тех процедур, прерывание которых может вызвать некорректность работы программы необходимо запрещать прерывание. Примером такой операции может служить запись в EEPROM.

Каждый МК имеет свою индивидуальную таблицу прерываний, которую можно найти в техническом описании МК, так таблица прерываний для МК ATMega8 выглядит следующим образом:

```
.ORG   INT0addr       ; External Interrupt Request 0
RETI
.ORG   INT1addr       ; External Interrupt Request 1
RETI
.ORG   OC2addr        ; Timer/Counter2 Compare Match
RETI
.ORG   OVF2addr       ; Timer/Counter2 Overflow
RETI
.ORG   ICP1addr       ; Timer/Counter1 Capture Event
RETI
.ORG   OC1Aaddr       ; Timer/Counter1 Compare Match A
RETI
.ORG   OC1Baddr       ; Timer/Counter1 Compare Match B
RETI
.ORG   OVF1addr       ; Timer/Counter1 Overflow
RETI
.ORG   OVF0addr       ; Timer/Counter0 Overflow
RETI
.ORG   SPIaddr        ; Serial Transfer Complete
RETI
.ORG   URXCaddr       ; USART, Rx Complete
RETI
.ORG   UDREaddr       ; USART Data Register Empty
RETI
.ORG   UTXCaddr       ; USART, Tx Complete
RETI
.ORG   ADCCaddr       ; ADC Conversion Complete
RETI
```

```
.ORG   ERDYaddr       ; EEPROM Ready
RETI
.ORG   ACIaddr        ; Analog Comparator
RETI
.ORG   TWIaddr        ; 2-wire Serial Interface
RETI
.ORG   SPMRaddr       ; Store Program Memory Ready
RETI
.ORG   INT_VECTORS_SIZE
```

Листинг 3.1. Таблица векторов прерываний AtMega8

Рассмотрим ее подробней, с помощью директивы .org определяется физический адрес каждого прерывания в МК, команда RETI выступает заглушкой на случай самопроизвольного срабатывания прерывания, в программе для реализации прерывания необходимо заменить RETI на RJMP LABEL, где LABEL это метка обработчика прерывания, выход из которого необходимо совершать по команде RETI. Более подробно реализация прерываний будет рассмотрена далее на примерах.

4. Структура AVR программ

Каждая программа, написанная для МК написанная на языке ассемблера, включает в себя несколько обязательных элементов (указаны в порядке расположения в программе):

1. Директивы препроцессора, в которой подключаются, необходимы библиотеки, определяющие тип МК.

2. Именование переменных и определение констант.

3. Сегмент SRAM, где выделяется память под переменные.

4. Таблица векторов прерываний, за которой обычно располагают обработчики прерываний.

5. Сегмент кода, и собственно программа, исполнение каждой программы начинается с нулевого адреса, где всегда располагается одна и та же команда безусловного перехода RJMP RESET. Естественно данная метка (именование метки RESET общеупотребимо, но называться она может как угодно) должна

быть дальше указана в программе, в любом удобном месте после таблицы векторов прерываний. Обычно по метке RESET располагают инициализацию стека, настройку периферии, портов ввода/вывода.

6. В каждой программе должен присутствовать бесконечный цикл, он может быть основан как на метке RESET, так и на любой другой.

7. В конце программы обычно располагается сегмент данных EEPROM.

С учетом вышеперечисленных требований структура программы (таблица прерываний указана не полностью) будет выглядеть следующим образом:

```
//Директивы препроцессора
 #include "m8def.inc"

//Определение переменных (директива .def)

//Определение констант   (директива .equ)

//Сегмент ОЗУ (RAM)
.DSEG

 //Сегмент кода (Flash)
.CSEG
.ORG 0x0000
       RJMP RESET
//Таблица векторов прерываний:
.ORG   INT0addr       ; External Interrupt Request 0
RETI
…
.ORG   SPMRaddr       ; Store Program Memory Ready
RETI
.org   INT_VECTORS_SIZE
//Конец таблицы векторов прерываний

//Сегмент обработчиков прерываний

//Конец сегмента обработчиков прерываний

RESET:
//Инициализация стека: (обязательно во всех МК с программным стеком AtMega)
       LDI R16, LOW(RAMEND)//Загрузка указателя стека в конец SRAM
       OUT SPL, R16
       LDI R16, HIGH(RAMEND)      //Загрузка указателя стека в конец SRAM
       OUT SPH, R16

MAIN:

RJMP MAIN

//Сегмент энергонезависимой памяти (EEPROM)
.ESEG
```

Листинг 3.2. Структура программы

5. Отладка и компиляция простейшей AVR программ

Может возникнуть вопрос, в случае если же прерывания не используются, можно ли начать выполнение программы с нулевого адреса? Конечно можно, рассмотрим такую простейшую программу, а также процесс ее компиляции и отладки.

```
//Директивы препроцессора
 #include "m8def.inc"

//Сегмент кода (Flash)
RJMP RESET

RESET:
LDI R16, 12   //Загрузим в регистр константу
LDI R17, 18    //Загрузим в регистр константу
ADD R16,R17    //Сложим значения в регистрах, результат в R16

MAIN:

RJMP MAIN
```

Листинг 3.3. Простейшая программа

Как видно из листинга 3.3 в данной программе отсутствует таблица векторов прерывания, и инициализация стека, таким образом использовать прерывания и вызов функций командой CALL в такой программе нельзя. Также не именуются никакие переменные и константы, присутствует только обязательный переход с нулевого адреса и бесконечный цикл. Процесс работы программы прост, в каждый регистр загружается константа, затем значения, находящиеся в регистрах, складываются, и результат помещается в первый из них, рассмотрим процесс отладки программы. Для перехода к отладке необходимо в меню Debug выбрать пункт Start debugging and break, при первом запуске отладчика может появиться окно с выбором типа отладчика, необходимо выбрать Simulator. При запуске отладчика вид рабочей среды видоизменяется, рассмотрим его (рис.3.5). В окне с кодом появился указатель, который указывает текущее место выполнения программы. На панели справа кроме вкладки I/O VIEW появилась вкладка Processor, в которой можно увидеть текущее состояние контроллера, значение счетчика команд, указателя стека, регистровых пар, регистра SREG, количество тактовых циклов, частоту МК, время реальной работы программы и все РОН. Переход к следующей команде осуществляется нажатием кнопки F11. Так же важным

моментом является то, что можно самостоятельно менять значения битов в PBB и PON щелкая по ним мышкой.

Посмотрим, как выполняется программа, изначально указатель стоит по нулевому адресу на команде перехода RJMP, затем происходит переход по метке RESET: и в регистры R16 и R17 записываются значения констант, в окне Processor изменяются соответствующие значения регистров, далее происходит сложение регистров и результирующее значение помещается в регистр R16.

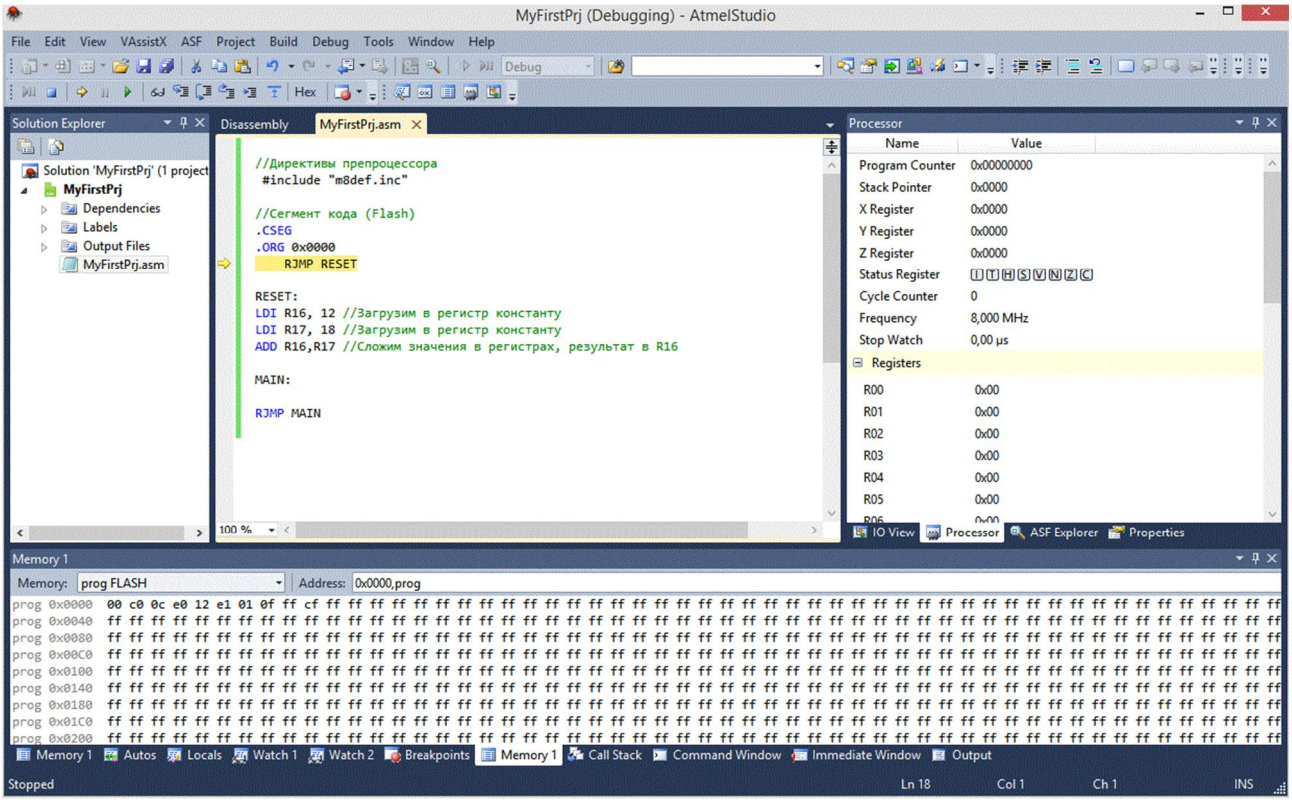


Рис.3.5. Окно отладчика

Так же внизу окна программы можно увидеть карты памяти МК, в выпадающем меню можно выбрать различные сегменты памяти, flash-память, сегмент памяти регистров и другие.

Компиляция программы выполняется из меню Build командой Build solution, результат компиляции в окно Output, и в случае успешной компиляции включает в себя пути компиляции, таблицу с указанием используемой памяти, сведения о количестве ошибок и предупреждений. В случае наличия ошибок и

предупреждений, будет выведен их код и краткое описание с указанием на строку с ошибкой.

6. Конфигурационные биты (Fuse биты)

Fuse биты или конфигурационные биты МК необходимы для «жесткого» конфигурирования МК, данная операция производится при прошивке устройства и требует внимательности, так как неверно выставленные Fuse биты могут привести микроконтроллер в состояние полной неработоспособности. Для разных моделей МК набор конфигурационных битов практически одинаков. Рассмотрим типовое состояние ячеек, заданное производителем. По умолчанию любой МК тактируется от внутренней RC-цепи, таким образом для базовой работы МК наличие внешней цепи тактирования не требуется. Для работы от другого источника тактового сигнала как было сказано выше необходимо изменить соответствующие конфигурационные биты (CKSEL0-3).

Рассмотрим другие достаточно часто используемые ячейки:

BOOT_ - ячейки настройки загрузки МК, позволяют изменить начальный адрес программы, адрес вектора прерывания, выбирать тип запуска МК.

Ячейки BODEN и BODLEVEL, разрешают работу схемы сброса контролера при падении питания ниже допустимого порога (схема BOD) и позволяют установить этот самый порог соответственно. Верный выбор значений предохранит МК от выполнения команд при недостаточном питание, которое может привести к «странностям» в работе устройства.

SPIEN – разрешает или запрещает последовательное программирование по SPI.

EESAVE – разрешает или запрещает обнуление EEPROM при программировании памяти программ.

В случае же если были выставлены неверные значения и МК не реагирует ни на какие внешние воздействия, можно переустановить Fuse-биты с помощью параллельного программатора.

Задания для самостоятельной работы

На основании выданных преподавателем индивидуального задания выполните необходимые исследования.

Составьте отчет по выполненным заданиям

ТЕМА №4. СИСТЕМА КОМАНД AVR

Вопросы, рассматриваемые в данной теме:

1. Команды условных переходов и регистр SREG
2. Команды пересылки данных
3. Выполнение типовых процедур на ассемблере
4. Битовые операции на ассемблере

Краткие теоретические сведения

1. Команды условных переходов и регистр SREG

Практически во всех микроконтроллерах различных архитектур присутствует регистр состояния, в архитектуре AVR он называется SREG, о нем не раз упоминалось в предыдущих разделах, рассмотрим его подробней.

Название расшифровывается как Status Register, этот регистр состоит из 8 независимых битов-флагов, которые могут принимать значения 0 или 1 в зависимости от операций, выполняемых контроллером. По значениям флагов в этом регистре можно судить о том, что произошло с процессором и определить результаты некоторых операций, например, если флаг Z установлен в значение 1, значит, в предыдущей математической операции результатом был 0. Биты регистра состояния приведены в таблице 4.1.

Таблица 4.1. Биты регистра SREG

№ бита	Обозначение	Наименование	Описание
7	I	Общее разрешение прерываний	Для разрешения прерываний данный флаг должен быть установлен в 1 (команда SEI), данный флаг сбрасывается аппаратно после входа в прерывание, и автоматически устанавливается после выхода из прерывания по команде RETI

6	T	Хранение копируемого бита	Используется в качестве источника или приемника бита при применении команд BLD (bit load) и BST (bit store)
5	H	Флаг половинного переноса	Устанавливается в случае если по результатам математической операции был совершен перенос бита из младшей половины байта в старшую или заем бита из старшей половины в младшую
4	S	Флаг знака	Устанавливается если результат математической операции меньше 0
3	V	Флаг переполнения дополнительного кода	Устанавливается в 1 при переполнении разрядной сетки знакового результата (числа представленного в виде с дополнительным кодом)
2	N	Флаг отрицательного значения	Устанавливается в 1 если старший разряд результата равен 1
1	Z	Флаг нуля	Устанавливается в 1 если результат операции равен нулю.
0	C	Флаг переноса	Устанавливается в 1, в если результате выполнения операции произошел выход за границы байта

Кроме автоматической установки каждый флаг можно устанавливать и сбрасывать с помощью команды SE* и CL* соответственно, где на месте * подставляется буквенное обозначение флага.

На базе этого регистра состояния работают такие важные команды, как команды условных переходов. Разберем эти команды на примере: представим, что у нас есть два неких значения, снятые с датчика и в зависимости от их разности МК должен выполнять различные действия, в данном случае устанавливать значение равное значению регистра R17 (Листинг 4.1).

```
//Директивы препроцессора
 #include "m8def.inc"

//Сегмент кода (Flash)
RJMP RESET
```

```
RESET:
LDI R16, 12   //Загрузим в регистр константу
LDI R17, 18   //Загрузим в регистр константу

MAIN:
CP R16,R17   //Сравним значения в регистрах
BRNE NOT_EQUAL
RJMP MAIN

NOT_EQUAL:
MOV R16, R17  // Скопируем в R16 значение R17
RJMP MAIN
```

Листинг 4.1. Пример использования команды условного перехода (BRNE)

Посмотрим, как изменяются значения бит в регистре SREG при работе команды сравнения (CP) в случае если первый операнд команды меньше второго. Запустив отладчик можно увидеть, что биты, устанавливаются во флагах S, N, C, а это значит, что операция сравнения фактически вычитает из первого значения второе и, не меняя значения в регистрах, выставляет флаги в регистре SREG. Далее выполняется команда условного перехода BRNE (brunch not equal – перейти если не равно) которая проверяет наличие вставленного флага Z, и, если он не выставлен, совершается переход на указанную метку (NOT_EQUAL). По данной метке командой MOV копируется значение регистра R17 в регистр R16 и происходит переход в основной бесконечный цикл, где снова происходит операция сравнения, по итогам которой выставляется флаг Z, и команда перехода естественно не выполняется.

Существует множество команд условных переходов, которые можно посмотреть в справочнике команд, самые распространённые из них это переходы по условиям равенства (BREQ), неравенства (BRNE), а также команды перейти, если значение больше или равно (BRSH) и перейти, если значение меньше (BRLO). В дополнение к ним существуют команды типа BRxS и BRxC, которые в зависимости от того установлен (S) или сброшен (C) флаг x регистра SREG совершают переход на метку.

Одной из особенностей команд переходов BR** является то, что эти команды являются командами относительного перехода и их разрядности может не

хватить, чтобы перепрыгнуть на метку, которая находится достаточно далеко в адресном пространстве, если это произойдет, компилятор выдаст ошибку «Relative branch out of reach», что делать в этом случае? Использовать промежуточные пустые переходы.

Кроме команд типа «перейти, если» существует еще один тип команд переходов: проверить и пропустить. Такие команды работаю по принципу, проверяем условие, если оно верно – пропускаем следующую команду. Существует две пары команд данного типа — это команды SBRC/SBRS, данные команды работают с РОН, и команды SBIC/SBIS которые проверяют биты РВВ. Рассмотрим пример команды работающей с РОН:

```
//Директивы препроцессора
 #include "m8def.inc"

//Сегмент кода (Flash)
RJMP RESET

RESET:
LDI R16, 0b00000000 //Загрузим в регистр константу


MAIN:
SBRC R16, 4                 //Если 4 бит в регистре не установлен
RJMP MAIN                   //Пропускаем следующую команду

LDI R16, 0b00010000 //Установим четвертый бит в R16
RJMP MAIN
```

Листинг 4.2. Пример использования команды условного перехода (SBRC)

В данной программе в регистр R16 записано значение равное нулю, затем в основном цикле проверяется, установлен ли бит 4 в регистр, если нет, то следующая команда пропускается, и программа выполняется дальше.

Команды условных переходов очень важны при написании программ на ассемблере, с помощью них могут быть реализованы все типовые конструкции языков высокого уровня, такие как if…then и им подобные.

2. Команды пересылки данных

Теперь рассмотрим другой важный класс команд, команды, которые переносят данные из одной области памяти в другую. В архитектуре AVR для записи и переноса данных между разными типами памяти существуют разные команды.

Команды, работающие с РОН уже знакомы, так как были использованы ранее, это LDI и MOV, их отличие состоит в том, что первая команда в РОН загружает непосредственно константу, а вторая копирует значение одного регистра в другой.

Следующий класс команд служит для работы с РВВ, это OUT и IN, первая команда позволяет записать в РВВ значение, вторая соответственно считать его.

Команды LD и ST, позволяют загружать и читать значения из памяти SRAM. В любом случае при записи и чтении SRAM используются регистровые пары X,Y и Z. Существует несколько режимов обращения к SRAM: чтение одной ячейки, и режимы с преддекрементом и постинкрементом служащие для чтения/записи фрагмента данных. Рассмотри основной порядок действий при чтении из памяти одной ячейки используя в качестве указателя на нее регистровую пару Z. Для чтения из памяти используются регистровые пары по причине того, что если бы использовался только один 8 разрядный регистр, то нельзя было бы адресовать памяти больше чем 2^8, тогда же как в МК памяти SRAM больше. Чтение из ячейки памяти с заданным адресом SAMPLE, изменение и запись нового значения выполняются так:

```
//Директивы препроцессора
 #include "m8def.inc"

//Сегмент данных (SRAM)
.dseg
SAMPLE: .BYTE  1    //Выделим в SRAM 1 байт по адресу SAMPLE

.cseg
//Сегмент кода (Flash)
RJMP RESET

RESET:
 LDI ZH, HIGH (SAMPLE)     //Установим старший разряд регистра-указателя
 LDI ZL, LOW (SAMPLE)      //Установим младший разряд регистра-указателя
 //Таким образом регистровая пара Z указывает на первую ячейку выделенной
 //памяти SAMLE

 LD R16,Z                        //Загрузим значение в регистр
 LDI R16, 0xFF                   //Модифицируем его

 ST Z, R16                       //Сохраним модифицированное значение обратно в память


MAIN:                            //Бесконечный цикл

RJMP MAIN
```

Листинг 4.3. Пример чтения и записи в SRAM

Запустим программу (листинг 4.3) в режиме отладки и исследуем ее работу. В сегменте данных программы директивой .BYTE выделяется один байт памяти, который располагается по адресу SAMPLE (имя естественно может быть любое, фактически это метка), затем происходит установка указателя на данный сегмент памяти в регистровую пару Z, таким образом можно увидеть, что в один из регистров этой пары записывается значение адреса (R30=0x60), из этого можно сделать вывод что чтение и запись будет происходить именно по этому адресу в SRAM, откроем вкладку Memory 1отладчика и выберем из выпадающего списка память – data IRAM, и посмотрим, что находится в необходимой нам ячейке памяти (0x00). Далее происходит загрузка этого значения в регистр (LD), его изменение (LDI) и сохранение по адресу в памяти (ST), теперь посмотрев значение ячейки SRAM можно увидеть в нем новое значение.

При работе с памятью данных нужно внимательно следить за двумя вещами, первое, это не выходить за границы существующей памяти, второе: если в программе используются прерывания и вызовы процедур, нужно помнить, что последние адреса памяти занимаются под стек.

В тех случаях, когда надо считать и записать массив данных используются команды:

```
ST -Z, Rx //запись в ячейку с адресом Z-1 значения из регистра x,
          //после выполнения команды Z-1
ST Z+, Rx //запись в ячейку с адресом Z значения из регистра x,
          //после выполнения команды Z
```

Команды чтения выглядят аналогично:

```
LD Rx, -Z //чтение из ячейки с адресом Z-1 значения из регистра x,
      //после выполнения команды Z-1
LD RX, Z+ //чтение из ячейки с адресом Z значения из регистра x,
      //после выполнения команды Z
```

Используя простейший цикл можно, например, записать в выделенную область памяти массив чисел от 0 до 16:

```
//Директивы препроцессора
 #include "m8def.inc"

 //Определение переменных (директива .def)
 .def TMP = R16 //Переменная для хранения временных значений
```

```
 //Определение констант   (директива .equ)
 .equ MAX_VALUE=16  //Количество чисел которые мы ходим записать

//Сегмент данных (SRAM)
.dseg
SAMPLE: .BYTE  MAX_VALUE   //Выделим в SRAM MAX_VALUE байт по адресу SAMPLE

.cseg
//Сегмент кода (Flash)
RJMP RESET

RESET:
 LDI ZH, HIGH (SAMPLE)     //Установим старший разряд регистра-указателя
 LDI ZL, LOW (SAMPLE)      //Установим младший разряд регистра-указателя
 //Таким образом регистровая пара Z указывает на первую ячейку выделенной
 //памяти SAMLE

 CLR TMP                           //Отчистим регистр TMP (а вдруг не 0)

LOOP:                              //Начало цикла записи
 ST Z+, TMP                        //Запишем значение в память и увеличим указатель на 1
 INC TMP                           //Увеличим число во временном регистре на 1
 CPI TMP,MAX_VALUE                 //Проверим условие, дошли до максимального числа?
 BRNE LOOP                         //Если нет, то повторяем, если да выходим из цикла

MAIN:                              //Бесконечный цикл

RJMP MAIN
```

Листинг 4.4. Запись блока данных в SRAM

Рассмотрим подробнее код программы и листинга 4.4, сразу необходимо обратить внимание в этой программе определяются переменные и константы директивами .DEF и .EQU, это удобно и делает код более наглядным, далее, так же как и в предыдущем примере выделяется определенный объем памяти, и по вектору RESET происходит переход к выполнению непосредственно программы, в которой устанавливается указатель на нужный сегмент памяти, обнуляется регистр (CLR) и начинается цикл записи с постинкрементом. В цикле сначала записывается новое значение в память (для первой итерации 0, для второй 1 и так далее), затем значение регистра, хранящего в себе число, которое копируется в память увеличивается на 1 (INC) и происходит проверка условия достижения границы выделенной области памяти, если достигли, выходим из цикла, если нет, переходим по метке LOOP и продолжаем выполнение цикла. При отладке этой программы можно увидеть, как изменяются значения регистра-указателя и как в цикле память наполняется значениями.

Кроме сохранения каких-либо значений в SRAM архитектура МК AVR позволяет создавать массивы констант в памяти программ, это может быть полезно для вывода каких-либо значений, которые изначально известны и не могут изменяться в процессе выполнения программы, например, масок цифр семисегментных дисплеев. Для чтения данных из памяти программ используется команда LPM, которая считывает указанный байт из памяти программ и помещает результат в регистр R0. Рассмотрим пример использование команды:

```
//Директивы препроцессора
 #include "m8def.inc"

 //Определение переменных (директива .def)
 .def TMP = R16 //Переменная для хранения временных значений

 //Определение констант   (директива .equ)

//Сегмент данных (SRAM)
.dseg

.cseg
//Сегмент кода (Flash)
RJMP RESET

RESET:
 LDI ZH, HIGH (N_MASK*2)   //Установим старший разряд регистра-указателя
 LDI ZL, LOW (N_MASK*2)    //Установим младший разряд регистра-указателя
 //Для адресации к памяти программ может быть использована только регистровая пара Z
 //Так как Flash память имеет двухбайтовую организацию, необходим множитель 2

LDI TMP,3
ADD ZL,TMP    //Установим указатель на 3 элемент массива N_MASK
LPM           //Выгрузим значение в регистр R0

MAIN:               //Бесконечный цикл

RJMP MAIN

N_MASK:             //Массив значений
.db 1, 3, 5, 7, 9, 11
```

Листинг 4.5. Чтение из памяти программ

Основное отличие при чтении из памяти программ заключается в том, что только регистровая пара Z может выступать указателем и то что организация памяти программ страничная (двухбайтовая) вследствие этого при инициализации указателя появляется умножение на 2.

3. Выполнение типовых процедур на ассемблере

Рассмотрим реализацию таких типичных для языков высокого уровня конструкций как условная конструкция if…then…else, оператор выбора case, цикл while. При рассмотрении конструкций, все полезные операции заменены на команды NOP.

Начнем рассмотрение с конструкции if…then…else. Для задания условий в блоке if используются команды сравнения (CP,CPI) и команды условного перехода (BR**). Важно понимать, что команды сравнения фактически вычисляют разность двух значений и выставляют соответствующие результату флаги в регистр SREG.

Для примера рассмотрим реализацию условия в виде строго неравенства A>B:

```
//IF...THEN...ELSE (строгое неравенство)
CP R16, R17 //Сравним два значения A(R16) и B(R17)       if(A>B)
//Значения в регистрах равны, переход на ELSE
BREQ ELSE_ACTION
//Если R16>R17, то флаг C регистра SREG не установится
BRCS ELSE_ACTION
IF_ACTION:          //if                                        {
                    NOP                                        IF_ACTION
                    NOP                                       }
                    RJMP NEXT_ACTION
ELSE_ACTION:        //else                                  else
                    NOP                                       {
                    NOP                                        ELSE_ACTION
                                                              }
NEXT_ACTION:                                              NEXT_ACTION
                    NOP
                    NOP
```

Листинг 4.6. Реализация конструкции if(A>B)…else на языке ассемблера

Рассмотрим задание условий подробнее. Так как у нас задано строгое условие, то одной командой условного перехода обойтись не удастся (в отличии от условия «больше либо равно»). Однозначно сказать что A>B, можно только в случае соблюдения двух условий: их разность не равна 0 (флаг нуля Z регистра SREG не установлен), и в случае если разность двух значений больше 0 (то есть флаг переноса C регистра SREG не установлен). За первое условие отвечает команда BREQ (перейти, если равно), за второе BRCS (перейти, если флаг C реги-

стра SREG установлен), в случае выполнения одного из условий происходит переход по метке ELSE_ACTION, в ином случае выполняется условие по метке IF_ACTION.

Таким образом комбинируя различные условия, можно реализовать проверку на условие практически любой сложности.

Теперь рассмотрим простой оператор выбора switch-case:

```
//Оператор выбора SWITCH-CASE
CPI R16, CONST1     //Выполнено первое условие?
BREQ CASE1          //Переходим по метке
CPI R16, CONST2     //Второе?
BREQ CASE2          //Переходим по метке

DEFAULT_CASE: //Ни одно условие не выполнено
              NOP
              NOP
              RJMP NEXT_ACTION
CASE1:              //Условие 1
              NOP
              NOP
              RJMP NEXT_ACTION
CASE2:              //Условие 2
              NOP
              NOP
              RJMP NEXT_ACTION
NEXT_ACTION:
              NOP
              NOP
```

```
switch (TMP)
       {
        case CONST1:
         action CASE1
         break;

        case CONST2:
         action CASE2
         break;

        default:
         action DEFAULT_CASE
        break;
       }
NEXT_ACTION
```

Листинг 4.7. Реализация конструкции switch-case на языке ассемблера

Рассмотрим листинг 4.7: определение к какому блоку case перейти выполняется с помощью сравнения (CPI) переменной, хранящейся в каком-либо из регистров и константы. В случае равенства этих значений и происходит переход по необходимой метке. Если ни одно из условий не верно, то выполняется условие по умолчанию DEFAULT_CASE, и выполнение конструкции завершается переходом к метке NEXT_ACTION.

Теперь посмотрим, как реализовать цикл с предусловием типа while, использование данного типа цикла пригодится в дальнейшем, например, для реализации программной задержки:

```
//Цикл с предусловием WHILE
CLR CNT //Отчищаем значение счетчика
WHILE:
       INC CNT    //Увеличиваем счетчик на 1
       //Досчитали до необходимого значения?
       CPI CNT, MAX_VAL
```

```
while(CNT<MAX_VAL)
       {
        WHILE_ACTION;
       }
NEXT_ACTION
```

```
       BREQ NEXT_ACTION //Если да, переход по
метке
WHILE_ACTION:      //Нет? Выполняем к/л операции
       NOP
       RJMP WHILE //И возвращаемся в начало цикла
NEXT_ACTION:
       NOP
```

Листинг 4.8. Реализация цикла с предусловием while на языке ассемблера

Сначала для правильной работы цикла обнуляется счетчик CNT, а затем по метке WHILE: начинается выполнение цикла, увеличивается на 1 значение счетчика и проверяется условие выхода из цикла (значение в счетчики CNT равно константе MAX_VAL), если условие выполняется, то происходит переход по метке NEXT_ACTION и выполнение цикла завершается, в ином случае выполняются операции из тела цикла по метке WHILE_ACTION: и происходит возврат в начало цикла (метка WHILE:).

4. Битовые операции на ассемблере

Битовые операции являются одними из самых распространённых операций при программировании МК, и большая часть таких операций идет через битовые маски. Ранее уже была рассмотрена операция сдвига влево (<<), операция побитовое «ИЛИ» (|), а также конструкция:

```
LDI R16, (1<<CS00)|(1<<CS01)
OUT TCCR0, R16
```

позволяющая установить определенные биты в регистре. Основной недостаток такой операции, то что так в регистр установятся только указанные биты, все остальные биты регистра будут обнулены. Разберем пример установки и сброса битов регистра без изменения состояния регистра. Допустим изначально необходимо на базе таймера (TIM1) настроить ШИМ на выход COM1A, а затем последовательно (не одновременно) подключить выход COM1B, и отключить выход COM1A.

С первоначальной установкой битов (листинг 4.9а) в пустой регистр никаких проблем нет, вышеуказанным способом используем операцию сдвига и побитовый «ИЛИ».

```
//ЧАСТЬ А
//Настройка таймера TIM1 для работы в режиме ШИМ
//Настроим ШИМ на сброс при обнулении, выход COM1A
       LDI TMP, (1<<COM1A1)|(1<<COM1A0)|(1<<WGM10)
```

```
OUT TCCR1A, TMP
LDI TMP, (1<<WGM12)
OUT TCCR1B, TMP
```

Листинг 4.9а. Установка и сброс битов в регистре

Чтобы подключить выход COM1B и не обнулить ранее установленные биты регистра (листинг 4.9б) сначала нужно прочитать старое значение из регистра, а затем с помощью команды ORI (логическое «ИЛИ» с константой, наложить на старое значение новую маску.

```
//ЧАСТЬ Б
//Необходимо дополнительно подключить выход COM1B
//При этом в регистре настройки TCCR1A уже установлены
//Другие настроечные биты

	IN TMP, TCCR1A                                  //Читаем значение из регистра
	ORI TMP, (1<<COM1B1)|(1<<COM1B0)//Накладываем на считанное значение маску (лог. ИЛИ)

	OUT TCCR1A, TMP                                 //Выводим новое значение в регистр
```

Листинг 4.9б. Установка и сброс битов в регистре

Сбросить биты и отключить выход COM1A можно так же используя битовую маску, но уже с операцией логического «И». Для удобства чтения кода (листинг 4.9в), в конструкции сброса используется установка битов в 1, а маска инвертируется операцией побитого «НЕ» (~).

```
//ЧАСТЬ В
//Необходимо дополнительно отключить выход COM1A
//При этом в регистре настройки TCCR1A уже установлены
//Другие настроечные биты
	IN TMP, TCCR1A                                  //Читаем значение из регистра
	ANDI TMP, ~(1<<COM1A1|1<<COM1A0)//Накладываем на считанное значение инвертированную
маску (лог. И)
	OUT TCCR1A, TMP                                 //Выводим новое значение в регистр
```

Листинг 4.9в. Установка и сброс битов в регистре

В дополнение к операциям сбросов и установки бит, достаточно часто используются операции сдвигов, такие операции бывают двух типов без переноса (LSR и LSL) и через перенос (ROL и ROR), отличие этих операций в том, что при сдвиге без переноса в зависимости от направления байт будет заполняться нулями, а при сдвиге с переносом бит уходящий за край будет попадать во флаг C регистра SREG и выходить с другого конца байта. Одно использование команды

дает один шаг влево или вправо. Операции сдвига часто используется если необходимо разделить или умножить какое-либо число, на число, являющееся степенью двойки. Разберем на примере: разделим 2496 на 8.

```
//Директивы препроцессора
 #include "m8def.inc"

//Определение переменных (директива .def)
 .def CNT     = R17        //Счетчик
 .def DATA_H  = R19        //Старший разряд числа
 .def DATA_L  = R20        //Младший разряд числа

//Определение констант   (директива .equ)
.equ DIV_NUM  = 3          //Степень числа 2, чтобы получить 8

MAIN:
LDI DATA_H,0x09            //Загрузим значение 2496 (0x09C0)
LDI DATA_L,0xC0

CLR CNT                    //Отчистим регистр счетчик

DIV8:                      //Операция деления на 8
LSR DATA_H                 //Сдвинули старший разряд вправо
ROR DATA_L                 //Сдвинули младший разряд вправо с переносом
INC CNT                    //Увеличили счетчик
CPI CNT,DIV_NUM            //Проверили сместили значение необходимое число раз
BRNE DIV8                  //Нет? Повторяем.

RJMP MAIN
```

Листинг 4.10. Деление чисел с помощью команд сдвига

После определения переменных и констант, загрузим значение, которое необходимо разделить в регистры, так как значение выход за 8-битный предел, для его хранения используется два регистра (DATA_H и DATA_L), в которых соответственно сохраняются старший и младший разряды числа. Затем следует сама операция деления на 8, так как 8 является результатом возведения 2 в 3 степень, то для деления необходимо три итерации сдвига, что и реализовано в функции DIV8, сначала старший байт (DATA_H) сдвигает вправо без переноса, затем младший (DATA_L) так же сдвигается вправо, но уже с переносом, после происходит проверка количества сдвигов, если три сдвига проведено, программа начинает выполняться заново, иначе запускается новая итерация сдвига.

Умножение выполняется аналогичным образом, только разряды смещаются не вправо, а влево.

Задания для самостоятельной работы

На основании выданных преподавателем индивидуального задания выполните необходимые исследования.

Составьте отчет по выполненным заданиям.

ТЕМА №5. ПОРТЫ ВВОДА/ВЫВОДА

Вопросы, рассматриваемые в данной теме:

1. Общие сведения о портах ввода/вывода
2. Режимы работы портов ввода/вывода
3. Подключение к портам ввода/вывода внешних устройств и их программирование.
   3.1. Подключение светодиода.
   3.2 Подключение кнопки.
   3.3 Подключение семисегментного индикатора.
   3.4 Подключение матричной клавиатуры.
4. Внешние прерывания

Краткие теоретические сведения

1. Общие сведения о портах ввода/вывода

Для взаимодействия с «внешним миром» в каждом МК предусмотрен набор портов ввода/вывода, в зависимости от модели МК может различаться как количество портов, так и их разрядность (номинально порты 8-разрядные). Обозначаются порты ввода/вывода латинскими заглавными буквами A, B, C и так далее, причем не обязательно по порядку, то есть в МК может присутствовать только порт B и D.

Так же для сокращения общих физических выводов МК порты ввода/вывода часто сочетают в себе и дополнительные функции. В документации на распиновке МК дополнительная функциональность вывода указывается в скобках. На этот момент стоит особо обратить внимание при проектировании микроэлектронного устройства и при возможности стараться не задействовать одну линию МК для обоих типов функций. Все линии портов автономны друг от друга и могут управляться как группами, так и независимо. При включении МК порты по умолчанию работают на вход в состоянии с высоким импедансом (Hi-Z), дополнительные функции выводов отключены.

Рассмотрим упрощенное устройство линии порта ввода/вывода:

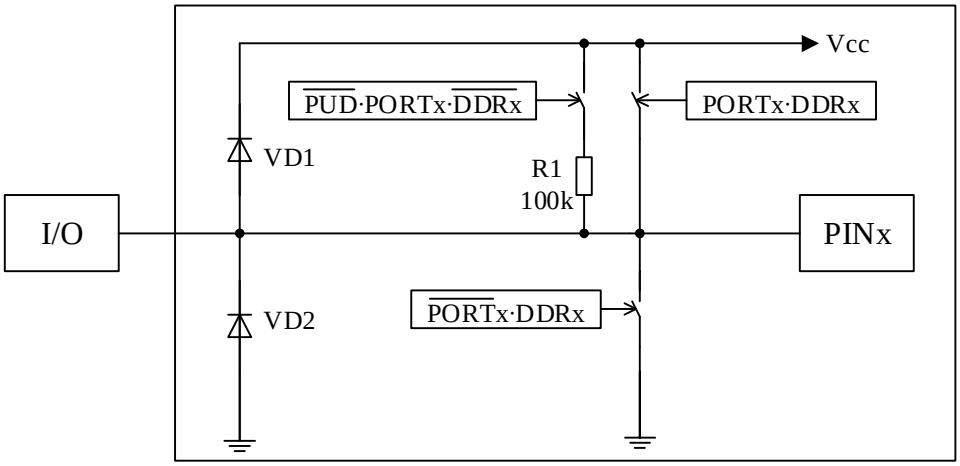


Рис. 5.1. Упрощенное устройство линии порта ввода/вывода

Диоды VD1 и VD2 на входе МК защищают линии МК от перенапряжения, например, если поданное на ножку МК напряжение будет выше напряжения питания, то откроется диод VD1 и напряжение будет подано на шину питания, в случае если на ввод попадет отрицательное напряжение, то оно через диод VD2 будет стравлено на землю. Но как показывает практика диоды стоят маломощные и помогают только от помех.

Далее из рисунка 5.1 видно, что в каждой линии порта есть три ключа управления (полевых транзистора) каждый со своим условием переключения, в случае выполнения условия ключ замыкается. Как можно легко понять из рисунка за состояние порта отвечают три регистра DDRx, PORTx, PINx. Рассмотрим каждый из этих регистров, а так возможные комбинации значений в этих регистрах:

DDRx – регистр направления порта, очевидно что в каждый момент времени, порт может работать либо в режиме входа, либо в режиме выхода. При DDRx=0 вывод работает как вход, при 1 как выход.

PORTx – регистр управляющих состоянием вывода, его функциональность зависит от того в какое значение выставлен регистр DDRx, в случае если DDRx=1 (линия порта настроена на выход), то значение в регистре PORTx определяет состояние вывода, при значении PORTx=1 на выводе будет логическая 1, в случае PORTx=0 на выходи соответственно будет логический 0.

Если DDRx=0 (линия порта настроена на вход), то возможны два режима работы вывода при PORTx=1 это режим с подтяжкой вывода к питанию (PullUp), при PORTx=0 то вывод находится в высокоимпедансном состоянии (Hi-Z), подробнее эти два режима работы будут рассмотрены ниже.

PINx – регистр чтения состояния порта, доступен только для чтения. В данном регистре содержится информация о текущем логическом уровне на выводах порта. То есть для того чтобы узнать какой сигнал в текущий момент на входе линии МК необходимо прочесть соответствующий разряд регистра PINx.

Если внимательно посмотреть на логические условия к ключам выводов МК, то можно заметить наличие еще одного регистра PUD (PullUp Disable), данный регистр запрещает подтяжку для всех портов, по умолчанию установлен в 0.

2. Режимы работы портов ввода/вывода

Режим выхода (DDRx=1) используется для управления внешними устройствами, если необходимо выдать на выход логическую 1, то в соответственный бит регистр PORTx записываем 1, если 0, то соответственно записываем 0.

Вход Hi-Z ((DDRx=0, PORTx=0), режим по умолчанию, все ключи разомкнуты, при этом сопротивления порта очень велико и его принято считать бесконечным, физически линия никуда не подключена и не может ни на что влиять. Кроме того, что это самый безопасный режим для работы МК он так же может использоваться в следующих случаях:

1. *Прослушивание шины данных*. Так как в этом режиме работы МК не оказывает никакого влияние на внешние устройства, но при этом постоянно опрашивает себя и записывает состояние в регистр PINx.
2. *Генератор случайных чисел.* Если вход оставить висеть в воздухе, то напряжение будет изменяться на нем в зависимости от внешних наводок, что позволяет сгенерировать более-менее случайную последовательность чисел. В бытовых условиях основная наводка — это наводка от сети 220В, в следствии этого в регистре PIN будет меняться 0 и 1 с частотой около 50 Гц.

Вход PullUp (DDRx=0, PORTx=1), вход с подтяжкой к шине питания через резистор 100кОм, таким образом линия, висящая в воздухе, переходит в состоянии логической единицы. Данный режим работы позволяет не допустить случайного изменения значений на линиях портов МК при его работе. В случае же появления на входе логического нуля (линия замкнута на землю каким-то внешним воздействием, например, нажатие кнопки) резистор не сможет удержать подтяжку к шине питания и на входе появиться логический 0. Так же с целью снижения энергопотребления в этот режим рекомендуется устанавливать все неиспользуемые порты ввода/вывода, так как в таком режиме работы от каждой помехи МК не придется переключать значение в регистре PINx.

3. Подключение к портам ввода/вывода внешних устройств

3.1 Подключение кнопки и светодиода.

Начнем рассмотрение подключения внешних оборудования к портам ввода/вывода МК с простейших устройств таких как кнопки и светодиоды.

Самый простой способ (рис. 5.2а) подключения кнопки (SB1) — это подключить кнопку напрямую к порту МК и при инициализации настроить линии порта МК на вход с подтяжкой (PullUp). Недостаток такого способа в том, что встроенный подтягивающий резистор достаточно «слаб», а значит помехоустойчивость системы значительно снизится. Второй способ (рис. 5.2б) – подключение параллельно внутреннему внешнего подтягивающего резистора (R1) номиналом 1-10кОм, что значительно улучшит помехоустойчивость системы.

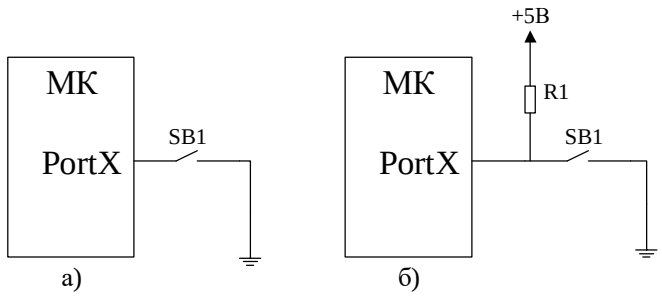


Рис. 5.2. Схемы подключение кнопки к МК

Основной момент, на который надо обратить внимание при подключении светодиодов это необходимость использования ограничительного резистора.

Для расчета номинала ограничительного резистора ($R_{ог}$) необходимо знать две характеристики:

1. Максимальный прямой ток светодиода и максимальный ток которым можно нагрузить линии порта МК.
2. Падение напряжение на светодиоде.

Обе характеристики можно посмотреть в документации на светодиод и МК. Для примера рассчитаем номинал ограничивающего резистора для красного светодиода с падением напряжения 1.8В и током около 20мА (данный ток для линии МК допустим). Очевидно, что при если на выводах МК напряжение будет равняться 5В, то на резисторе должно падать напряжение порядка 3.2В. Таким образом найдем сопротивление ограничительного резистора по закону Ома:

$$R_{\text{ог}} = \frac{U}{I} = \frac{3.2}{0.02} = 160 \text{ Ом},$$

В стандартном значении сопротивлений такое значение имеется, но для большей надежности рекомендуется взять следующее значение из ряда, яркость светодиода практически не уменьшиться. Если же значение в стандартном ряду отсутствует, то необходимо всегда выбирать резистор с сопротивлением больше чем рассчитанное. Схема подключения светодиода к линии порта МК представлена на рисунке:

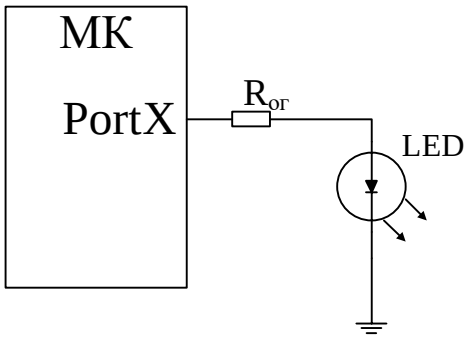


Рис. 5.3. Схема подключения светодиода к МК

Рассмотрим пример программы, управляющей линейкой из 8 светодиодов с помощью двух кнопок UP и DOWN, при первом нажатии кнопки UP появляется бегущая дорожка из зажжённых светодиодов вверх, при повторном нажатии дорожка гаснет, кнопка DOWN работает аналогично, за исключением того

то что линейка светодиодов зажигается в противоположном направлении. При разборе исходного кода программы опустим раннее рассмотренные конструкции, такие как определение переменных, таблицы векторов прерываний, инициализацию стека и тому подобное. Рассмотрим общую логику программы, инициализацию портов ввода/вывода и решение таких проблем как дребезг контактов и необходимость задержки при переключении на новую группу светодиодов.

Для удобства и более легкого восприятия кода (листинг 5.1) сначала определим переменные и константы, которые будут использоваться в дальнейшем.

```
//Определение переменных (директива .def)
 .def TMP     = R16        //Временная переменная
 .def CNT     = R17        //Счетчик
 .def CNT2    = R18        //Дополнительный счетчик
 .def SB_FLAG = R19        //Флаг нажатия кнопки

//Определение констант   (директива .equ)
.equ  LED_MSK  = 0b00010001
.equ  FST_PUSH = 1
```

Листинг 5.1. Инициализация констант и переменных

Константа LED_MSK отвечает за движение горящих светодиодов по линейки, и изменяя ее можно добиться разной конфигурации движения. Определение первый ли раз нажата кнопка происходит благодаря константе FST_PUSH использование которой в программе далее будет рассмотрено подробнее.

После определения переменных, векторов прерываний, и инициализации стека, необходимо настроить порты ввода/вывода на необходимый режим работы (листинг 5.2), а так как напрямую в порт ввода/вывода ничего записать нельзя, то для этого используется дополнительный регистр, в нашем случает это регистр R16 которому присвоено имя TMP. Команда LDI загружает константу во временный регистр TMP, а затем с помощью команды OUT, значение из регистра TMP загружается в один из управляющих регистров порта ввода/вывода, в случае линейки светодиодов, порт B полностью настраивается на выход (в регистр DDRB записывается значение 0xFF), для кнопок, так как порт по умолчанию настроен на вход, к необходимым линиям порта (0,1) просто подключаются встроенные резисторы подтяжки.

```
//Настройка портов ввода/вывода
//DDRx  - направление работы линии порта x (1-выход, 0-вход)
```

```
//PORTx - Значение уровня на линии порта x (1-высокий, 0-низкий)
//        если порт x настроен как вход    (1-PullUp)
//PINx  - Уровень сигнала на линии порта x (Только для чтения)

//Подключение линейки светодиодов (PORTB) - все линии на выход
       LDI TMP, 0xFF
       OUT DDRB, TMP
//Подключение кнопок (PORTD)-линии 0,1 - вход с подтяжкой
       LDI TMP, 0b00000011
       OUT PORTD, TMP
```

Листинг 5.2. Инициализация портов ввода/вывода

После инициализации необходимых устройств начинается выполнение основного бесконечного цикла (MAIN:):

```
MAIN:
//Проверяем нажата ли кнопка "ВВЕРХ"?
SBIS PIND,0
RJMP SB_UP

//Нажата ли кнопка "ВНИЗ"?
SBIS PIND,1
RJMP SB_DOWN

RJMP MAIN
```

Листинг 5.3. Основной цикл программы

В данном цикле с помощью команды проверки-пропуска (SBIS), определяется значение из необходимого регистра (в нашем случает это регистр состояния порта PIND и линии 0,1) и в случае если разряд PBB установлен, то следующая команда пропускается (регистр адресов PC+2), и опрашивается состояние следующей кнопки. Может возникнуть вопрос, почему кнопка считается не нажатой если на линии порта высокий уровень? Ответ прост, ранее, при инициализации эти линии порта во избежание ложных срабатываний были подтянуты через встроенные резисторы к напряжению питания МК.

В случае же если одна из кнопок нажата пропуска команды не происходит. И по команде относительного перехода RJMP происходит переход к обработчику одной из кнопок.

Для примера рассмотрим обработчик кнопки ВВЕРХ (SB_UP):

```
//Нажата кнопка вверх
SB_UP:
SBIS PIND,0          //Кнопка опущена?
RJMP SB_UP           //Нет, ждем отпускания, иначе продолжаем выполнять программу

INC SB_FLAG          //Установим флаг нажатия кнопки
```

```
CPI SB_FLAG,FST_PUSH//Кнопка нажата первый раз?
BRNE CLR_MASK              //Нет? Надо остановиться и перейти к обнулению линейки

LDI TMP, LED_MSK           //Иначе запишем во временный регистр маску движения светоидов

UP_CYCLE:                  //Цикл движения вверх (фактически вправо)
ROR TMP                    //Сдвигаем маску вправо
RCALL DELAYnS              //Для наглядности вводим маленькую задержку (на частоте 1МГц до-
статочно)
OUT PORTB,TMP              //Выводим маску в порт

//Проверяем нажимались ли кнопки, если да обрабатываем нажатие и останавливаем вывод
SBIS PIND,0
RJMP SB_UP

SBIS PIND,1
RJMP SB_DOWN
//Если нет, продолжаем вывод в цикле
RJMP UP_CYCLE
```

Листинг 5.4. Обработка нажатия кнопки вверх

Для борьбы с дребезгом контактов существует несколько приемов, в данной программе используется прием (листинг 5.4), при котором нажатие кнопки начинает обрабатывается не сразу после нажатия на кнопку, а только после ее отпускания, то есть до момента отпускания кнопки происходит процедура аналогичная определению нажата ли кнопка. После отпускания кнопки собственно и начинается обработка события «кнопка нажата». Так как в условии задания указано что при повторном нажатии на кнопку устройство должно останавливать свою работу, введем в программу флаг нажатия кнопки SB_FLAG (смотри инициализацию переменных и констант). С помощью команды инкрементирования (INC) значение флага увеличивается на единицу и далее сравнивается (CPI) с константой FST_PUSH, в случае если значения равны (то есть кнопка нажата первый раз), команда условного перехода (BRNE) не срабатывает и продолжается дальнейшее выполнение программы. Во временный регистр TMP командой LDI записывается маска движения светодиодов. И запускается цикл вывода (UP_CYCLE:) значений маски на порты ввода/вывода МК. Для того чтобы дорожка двигалась вверх воспользуемся командой логического сдвига вправо через перенос (ROR), а для того чтобы человеческий глаз успевал замечать движение дорожки введем подпрограмму небольшой программной задержки (RCALL DELAYnS), после задержки маска выводится в порт и светодиоды зажигаются (OUT

PORTB,TMP). Так же в цикле вывода постоянно опрашиваются кнопки, и в случае нажатия любой из кнопок вывод на линейку светодиодов останавливается.

Теперь рассмотрим подпрограмму задержки (DELAYnS:) и функцию отчистки флага кнопки и отключения вывода значений на линейку:

```
//Отчистка флага кнопки и обнуление показаний на линейке
CLR_MASK:
CLR SB_FLAG         //Устанавливаем флаг в 0
OUT PORTB,SB_FLAG   //Выводим на линейку 0
RJMP MAIN

//Задержка
DELAYnS:
       LDI CNT,255   //Записываем в счетчик 255
       LDI CNT2,255  //Записываем в дополнительный счетчик 255
DELAY_CYCLE:
       DEC CNT          //Уменьшаем
       BRNE DELAY_CYCLE //Дошли до 0?
DELAY2_CYCLE:
       LDI CNT,255         //Загружаем в счетчик новое значение
       DEC CNT2            //Уменьшаем на единицу дополнительный счетчик
       BRNE DELAY_CYCLE    //Дошли до нуля? Если нет возвращаемся к первому циклу
RET                         //Выходим
```

Листинг 5.5. Функция отчистки и подпрограмма задержки

В функции отчистки (CLR_MASK:) командой CLR обнуляется значения флага нажатия кнопки, а затем это же значение выводится в порт гася все светодиоды.

Программная задержка реализована в виде подпрограммы с двумя вложенными циклами внутри (DELAY_CYCLE и DELAY_CYCLE2). Что позволяет на частоте 1МГц реализовать задержку которая позволяет наблюдать движение зажигающихся светодиодов по линейке.

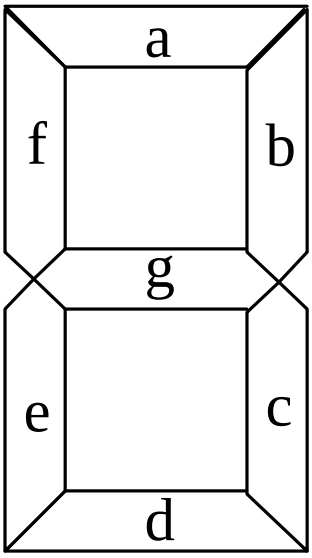
Сначала в подпрограмме задержки (листинг 5.5) оба счетчика CNT и CNT2 выставляются в максимальное значение (255), затем в первом цикле подпрограммы (DELAY_CYCLE) уменьшаем значение счетчика CNT командой декремента (DEC) и проверяем командой условного перехода (BRNE) 0 ли в регистре счетчика, если да, то переходим к выполнению второго цикла (DELAY2_CYCLE), в котором снова восстанавливаем значение в счетчике CNT (LDI CNT,255) и уменьшив значение второго счетчика на 1, и если значение счетчика CNT2 не равно 0 то возвращаемся к выполнению первого цикла. Если же равно, то значит подпрограмма задержки отработала и можно возвращаться

(RET) к дальнейшему выполнению обработки кнопки. Таким образом на 8-разрядном МК был реализован 16-разрядный программный счетчик, который если рассчитать точное значение задержки можно использовать и для отсчета каких-то более точных промежутков времени. Основной недостаток (кроме точности отсчета) такого подхода это то что во время выполнения отсчета МК никакими другими полезными операциями фактически не занимается.

В следующем разделе рассмотрим подключение семисегментного дисплея и работы с константами, записанными в памяти программ (Flash-памяти).

3.2 Подключение семисегментного индикатора.

Рассмотрим светодиодные (LED) семисегментные индикаторы, фактически они являются группой светодиодов, подключенных и расположенных в определенном порядке. Данные индикаторы могут быть как с общим анодом, в этом случае вывод плюса питания общий, а зажигаются сегменты коммутацией их к нулю питания, так и с общим катодом (общий отрицательный вывод). Так же существуют семисегментные индикаторы со встроенным дешифратором что существенно упрощает вывод на них цифр, но ничего кроме цифр вывести на них нельзя. Здесь мы рассмотрим классические семисегментные индикаторы. Изображение на индикаторах строиться с помощью зажигания определенных светодиодов, таким образом вывести на индикатор можно как цифры, так и большую часть алфавита. Обозначение выводов семисегментных индикаторов, и таблица формирования цифр можно посмотреть в документации, но зачастую они соответствуют представленным на рисунке 5.3:



DEC	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

Рис. 5.3. Обозначение выводов семисегментных индикаторов, и таблица формирования цифр

Схема подключения индикаторов аналогична подключению обычных светодиодов. Но при этом при подключении LED-индикаторов, состоящих из большого числа светодиодов стоить учитывать ограничение суммарного тока через порты МК (порядка 140мА в зависимости от модели), то есть сколько угодно индикаторов напрямую к портам ввода/вывода МК не подключишь. В случае же если требуется подключить большее чем способен запитать МК количество устройств, часто используются транзисторные ключи.

Рассмотрим пример управление семисегментным индикатором решив простейшую задачу: при включении устройства и нажатии на кнопку на дисплей выводятся последовательно числа от 0 до 9, при следующем нажатии снова выводится ноль и так далее. Самым очевидном способом решения этой задачи является вывод из регистра в порт маски необходимой в данный момент цифры. Основной недостаток такого способа в том, что такую операцию тяжело обрабатывать в цикле, плюс постоянно будет присутствовать операции чтения-записи. Более оптимальным способом является размещение массива масок чисел в памяти, а так как маски чисел константы лучше всего будет разместить их во flash-памяти МК. Кроме уменьшения количества операций чтения-записи при выполнении

программы размещение массива констант во flash-памяти позволяет обращаться к данным как к обычному массиву.

При рассмотрении кода опустим подробное описание инициализации портов ввода/вывода, обработку нажатия кнопки и другие процедуры, рассмотренные ранее. Начнем с инициализации констант и переменных, которые будут использоваться далее.

```
//Определение переменных (директива .def)
 .def TMP       = R16             //Временная переменная
 .def L_Counter = R17             //Счетчик символов

//Определение констант   (директива .equ)
.equ END_STRING = 10
```

Листинг 5.6. Инициализация констант и переменных

Константа END_STRING необходима для того чтобы определить, что на дисплее выведено максимальное значение и при следующем нажатии необходимо сбросить значение на дисплее в 0.

Так как константы необходимо разместить во flash памяти, то директива их размещения (.db) размещается в сегменте кода (.CSEG) сразу за программным кодом перед сегментом EEPROM (.ESEG). Согласно таблице формирования цифр (рис. 5.3) создадим массив констант в памяти программ МК:

```
//Маска цифр для вывода на семисегментный дисплей (расположен во FLASH памяти)
N_mask:
.db 0b00111111, 0b00000110, 0b01011011, 0b01001111, 0b01100110, 0b01101101, 0b01111101,
0b00000111, 0b01111111, 0b01101111
```

Листинг 5.7. Массив констант в памяти программ МК

Для обращения к конкретным элементам массива данных расположенных в памяти программ необходимо установить указатель на начало массива, а так как память программ имеет страничную организацию, для корректной адресации к данным младший и старший разряд регистра указателя умножаются на 2 (листинг 5.7). Так же стоит обратить внимание на то что данные в памяти размещаются последовательно друг за другом и что для адресации к памяти программ может быть использована только регистровая пара Z.

```
MAIN:
//Установим указатель на начало массива с символами
```

```
LDI ZH, HIGH (N_MASK*2)   //Установим старший разряд регистра-указателя
LDI ZL, LOW (N_MASK*2)    //Установим младший разряд регистра-указателя
//Для адресации к памяти программ может быть использована только регистровая пара Z
//Так как Flash память имеет двухбайтовую организацию, необходим множитель 2
```

Листинг 5.8. Установка указателя на начало массива данных.

Данная конструкция (листинг 5.8) располагается в начале главного бесконечного цикла MAIN: то есть указатель будет переинициализироваться каждый раз при его запуске цикла, зачем это сделано будет понятно далее.

Продолжим рассмотрения исходного кода программы (листинг 5.9), после обработки нажатия кнопки обязательно идет проверка, достигли ли мы максимального значения (константа END_STRING), если достигли, что проверяется командой проверки-пропуска BRNE, то значение переменной L_Counter обнуляется и запускается функция вывода символа на дисплей (STR_OUT:). Рассмотрим эту функцию подробно. Сначала установим указатель на адрес в памяти по которому находится необходимое нам число. По умолчанию указатель стоит в начале массива где находится число 0. Таким образом если необходимо вывести на дисплей число 5, то к значению адреса расположенного в регистровой паре Z необходимо прибавить число 5. В программе же прибавляется (ADD) значение с именем L_Counter. Далее командой обращения к памяти LPM происходит обращения к данным находящимся по адресу Z и данные автоматически помещаются в регистр R0. Далее полученное значение выводится на дисплей (OUT) и значение счетчика L_COUNTER увеличивается на единицу (INC). Далее происходит переход к основному бесконечному циклу MAIN: указатель устанавливается на начало массива и устройство ожидает повторного нажатия кнопки.

```
//Нажата кнопка
SB_PUSH:
	SBIS PIND,0		//Кнопка опущена?
	RJMP SB_PUSH  //Нет, ждем отпускания, иначе продолжаем выполнять программу

	CPI L_COUNTER, END_STRING  //Проверяем, дошли до конца строки?
	BRNE STR_OUT               //Нет, выводи следующий символ

	CLR L_Counter              //Да, обнуляем счетчик

STR_OUT:
	ADD ZL,L_Counter    //Сместим указатель на элемент массива символов на величину равную номеру требуемого к выводу символа
	LPM                 //Достанем символ из адреса, на который указывает указатель
	OUT PORTC,R0        //Выведем его на дисплей
```

```
        INC L_Counter       //Увеличим номер выводимого символа на единицу
RJMP MAIN
```

Листинг 5.9. Функция чтения из памяти программ и вывода на дисплей.

Теперь должно быть понятно почему при переходе к основному бесконечному циклу указатель устанавливается на начало массива, ведь если этого не сделать, то при следующем смещении указателя на необходимое значение мы получим не требуемый адрес, а предыдущий плюс указанное при новой итерации смещение.

В заключении можно сказать что, хотя управление одним дисплеем не представляет особых сложностей, то при подключении достаточного большого количество светодиодных индикаторов схема подключения оказывается достаточно громоздкой, а при нехватке портов ввода/вывода приходится применять такие методы как динамическая индикация (будет рассмотрена далее). В таких случаях более рационально использование ЖК индикаторов.

4. Внешние прерывания

Часто для индикации какого-либо внешнего события могут быть использованы внешние прерывания. В каждом МК за внешние прерывания отвечают определенные выводы портов ввода/вывода называющиеся INTx, где x – номер по порядку. Так, например, у МК AtMega8 две линии внешних прерываний INT0 и INT1 привязаны к линиям PD2 и PD3 соответственно.

Для разрешения внешних прерываний существует специальный регистр настройки GICR, в котором установка бита INTx, разрешает прерывание x, а сброс запрещает.

Внешнее прерывание может происходит по одному из условий задаваемом битами ISC01 и ISC00 конфигурационного регистра MCUCR. Основные условия срабатывания прерывания сведены в таблицу 5.1:

Таблица 5.1. Условия генерации внешнего прерывания (INTx)

ISCx1	ISCx0	Условие генерации прерывания

0	0	По низкому уровню
0	1	По любому изменению логического уровня
1	0	По спадающему фронту
1	1	По нарастающему фронту

А разница между уровнями сигнала представлена на рисунке 5.4.

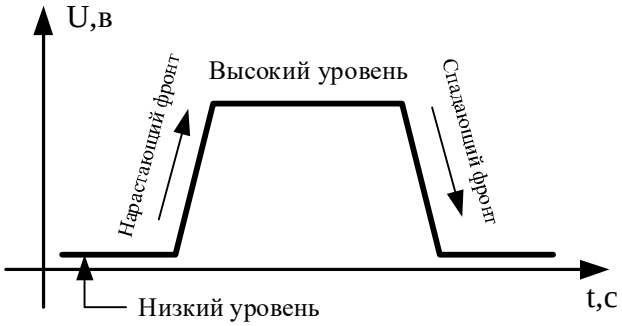


Рис. 5.4. Разница между фронтом и уровнем сигнала

Перепишем программу из раздела 3.2 обрабатывая нажатие кнопки как внешнее прерывание. Опустим ранее рассмотренные фрагменты программы и разберем конфигурацию прерывания (листинг 5.10):

```
//Настройка внешнего прерывания INT0 (для обработки нажатия на кнопку)
LDI TMP, 1<<ISC01          //Прерывание по спадающему фронту
OUT MCUCR, TMP
LDI TMP, 1<<INT0           //Разрешение прерывания
OUT GIMSK,TMP

SEI                                  //Не забываем вообще разрешить прерывания
```

Листинг 5.10. Настройка прерывания

В листинге 5.10 выполняется конфигурация прерывания INT0 по спадающему фронту (1<<ISC01), разрешение этого прерывания (1<<INT0), а также глобальное разрешение прерываний (SEI).

Вывод на дисплей выполняется аналогично примеру, рассмотренному в разделе 3.2, только в обработчике прерывания INT0.

Задания для самостоятельной работы

На основании выданных преподавателем индивидуального задания выполните необходимые исследования.

Составьте отчет по выполненным заданиям.

ТЕМА № 6. ПРОГРАММИРОВАНИЕ ТАЙМЕРОВ

Вопросы, рассматриваемые в данной теме:

1. Общие сведения о таймерах и режимы их работы
2. Особенности программирования таймеров
3. Простейший секундомер
4. Таймеры в режиме ШИМ

Краткие теоретические сведения

1. Общие сведения о таймерах и режимы их работы

Любой таймер в МК является простым двоичным счетчиком, и в зависимости от разрядности может считать 0 до 255 в случае 8 –разрядного таймера и от 0 до 65535 в случае 16-разрядного таймера. Именно такой разрядности таймеры присутствуют в МК AVR. В зависимости от модели может присутствовать разное количество таймеров, но 8-разрядные таймеры всегда имеют четные номера (0,2), а 16-разрядные – нечетные (1,3). В чем же основные отличая от обычного программного таймера счетчика? Самое главное отличие — это возможность аппаратного таймера считать независимо от работы процессора и генерировать прерывания по различным событиям. Так же стоить отметить возможность работы таймеров в асинхронном режиме, в режиме подсчета импульсов и генерации различных ШИМ сигналов. В данной главе будут рассмотрены только основные режимы работы и конфигурационные регистры таймеров, так как полному описание режимов работы таймеров с примерами и особенностями использования можно посвятить отдельную книгу.

Рассмотрение таймеров начнем со способов их тактирования. Тактироваться таймер может как от внутреннего генератора, так и от счетного входа (выводы T0, T1), при тактировании от счетного входа, в зависимости от его настройки, может считаться либо задний, либо задний фронт импульсов посту-

пающего на вход сигнала. Основное условие для работы таймера при тактировании от счетного входа это то что частота тактового сигнала не должна превышать частоту МК, иначе МК не будет успевать обрабатывать импульсы.

Так же один из таймеров может работать в асинхронном режиме, обычно это таймер 2, в этом случае таймер считает импульсы от отдельного кварцевого генератора подключаемого к выводам МК TOSC1 и TOSC2. Благодаря этому режиму можно легко организовать часы использовав для тактирования обычный часовой кварц на 32768Гц. Так же асинхронный таймер может продолжать работать в режимах глубокого энергосбережения МК и выводить из таких режимов МК по прерыванию.

Для удобства пользования таймерами в МК предусмотрен предделитель. Делить частоту можно на 8, 32, 64, 128, 256, 1024. Предделитель удобно использовать для отсчета достаточно больших промежутков времени, ведь если единственный источник тактовых сигналов — это тактовый генератор процессора, работающий на частоте 8МГц, то даже 16-разрядный таймер будет переполняться очень быстро, в случае же использования предделителя на 1024 работа с таймером станет гораздо удобнее. З настройки предделителя обычно отвечают биты CSx2..CSx0 регистра TCCRx(в некоторых таймерах регистр TCCRx фактически 16 разрядный и представляет собой два регистра TCCRxA и TCCRxB и в целом является основным регистром настройки таймера) где x это номер таймера. В зависимости от типа таймера и модели МК настройка предделителя может отличаться и ее необходимо уточнить в документации на МК. Режимы работы предделителя для таймеров МК AtMega8 представлены в таблице:

Таблица 6.1. Режимы работы предделителя таймеров МК

CS02…CS00	Режим работы
000	Таймер остановлен
001	Предделитель 1 (выключен), таймер считает тактовые импульсы
010	Предделитель 8
011	Предделитель 64

100	Предделитель 256
101	Предделитель 1024
110	Импульсы с вывода T0, при переходе с 1 на 0
111	Импульсы с вывода T0, при переходе с 0 на 1

Все результаты отсчетов таймеров помещаются в счетный регистр TCNTx в случае 8-разрядного таймера, или TCNTxH и TCNTxL в случае 16-разрядного таймера где x – номер таймера. Особенности их чтения и записи рассмотрим позже.

Так же каждый таймер может генерировать множество прерываний, за прерывания в таймерах отвечают регистры TIMSK и TIFR, регистр TIMSK, это регистр масок, то есть все его биты локально разрешают какие-либо прерывания от таймера (пока прерывание явно не разрешено, даже при глобальном разрешении прерываний, оно не сработает). Регистр TIFR это непосредственно флаговый регистр, при срабатывании прерывания в него устанавливается флаг о том, что условие прерывания выполнены, и когда программа уходит по вектору прерываний флаг сбрасывается. В случае же если прерывания запрещены глобально (CLI), то флаг так и будет стоять и прерывание сработает при их повторном разрешении (SEI), для того чтобы этого не произошло флаг необходимо сбросить вручную, для этого в нужный бит регистр TIFR необходимо записать 1. Все основные биты отвечающие за прерывания приведены в таблицы:

Таблица 6.2. Биты разрешения прерываний таймеров

Имя бита	Тип прерывания	Таймеры
TOVx	Переполнение таймера	T0,T1,T2
ICFx	Захват входа	T1
OCFxA	По совпадению A	T1

OCFxB	По совпадению B	T1
OCFx	По совпадению данных OCRx	T2

Кроме вышеперечисленных режимов таймеры могут работать в режиме ШИМ. Данный режим реализован на базе регистра сравнения таймера OCRxx, принцип его работы заключается в том, что когда значение в счетном регистре таймера TCNTx достигает значения находящегося в регистре сравнения то, возникает аппаратное событие: прерывание по совпадению или изменение состояния вывода OCxy, где x – номер таймера, y регистр сравнения (A или B).

Допустим, что мы настроили ШИМ так, что когда значение в счетном регистре (TCNTx) больше чем в регистре сравнения (OCRxy), то на выходе OCxy высокий уровень сигнала, иначе низкий. Таким образом таймер будет считать, как ему и положено, а при переполнении сбрасываться и продолжать отсчет заново, при этом на выходе OCxy будут появляться импульсы, скважность которых будет зависеть от значения в регистре сравнения (OCRxy). Временная диаграмма такого режима работы представлена на рисунке 6.1.

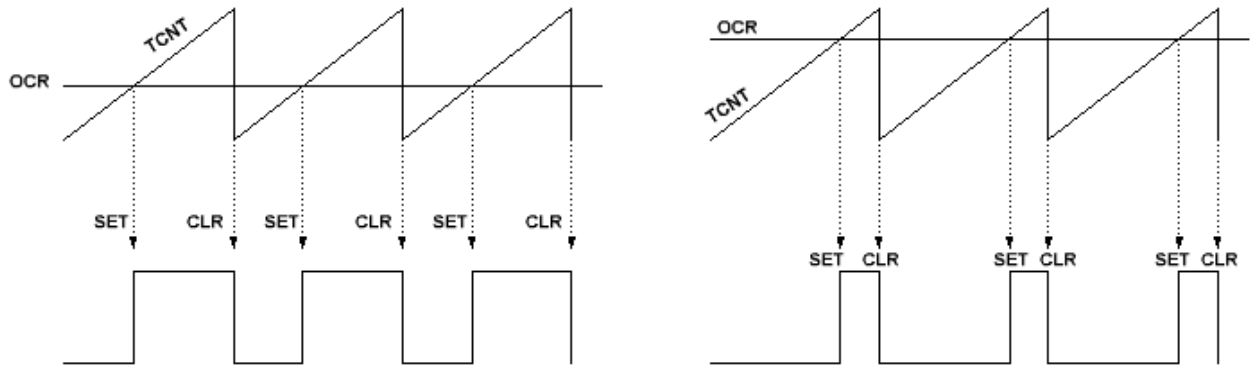


Рис. 6.1. Временная диаграмма работы ШИМ

Рассчитаем частоту ШИМ, для 8-разрядного таймера и МК работающего на максимальной для внутреннего тактового генератора частоте 8МГц:

$$\frac{\text{Частота МК}}{2^{\text{разрядность ШИМ}}} = \frac{8000000}{255} = 31250\text{Гц}$$

Большую частоту ШИМ можно получить, увеличив частоту МК, то есть подключив внешний кварц или другой источник тактового сигнала более высокой частоты. Так же существует возможность повысить разрядность ШИМ (если это позволяет разрядность таймера), но это вместе с повышением дискретности выходного аналогового сигнала, резко снижается частота ШИМ.

Рассмотрим типы ШИМ аппаратно реализованные в МК:

1. Fast PWM (рис.6.2а) – режим в котором счетчик считает от 0 до 255, после переполнения автоматически сбрасывается и продолжает счет. Когда значение в счетчике TCNTx становится равным значению лежащем в регистре сравнения OCRxy тогда соответствующий ему вывод OCxy сбрасывается в 0, при обнулении счетчика на выводе устанавливается 1.

2. Phase correct PWM (рис.6.2б) - ШИМ с корректировкой фазы, отличается от предыдущего режима в способе отсчета. Данный тип ШИМ сначала считает от 0 до 255, а затем в обратном направлении от 255 до 0, выход OCxy при первом совпадении сбрасывается, при втором устанавливается. Но так как период возрастает, то частота ШИМ падает в два раза. При таком режиме ШИМ при изменении скважности не изменяется угол фазового сдвига между двумя ШИМ сигналами, то есть центры импульсов в разных каналах и разной скважности будут совпадать.

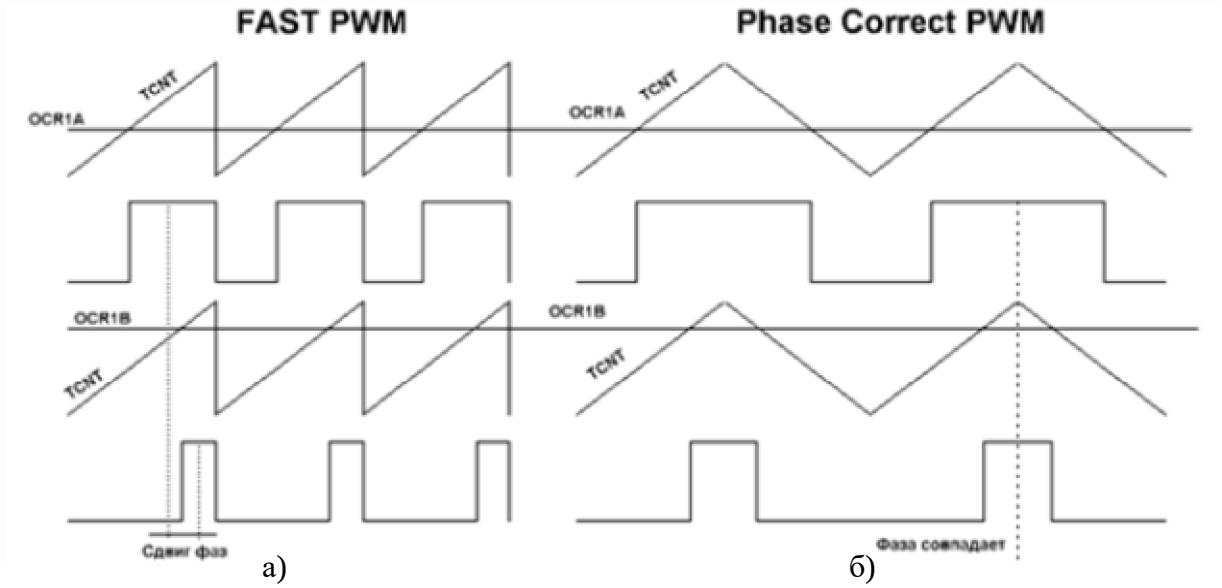


Рис. 6.2. Временные диаграммы режимов работы ШИМ

3. Clear timer on compare - CTC Mode (рис. 6.3) – сброс при сравнении, в этом режиме таймер работает не от 0 до максимального значения, а от 0 до значения записанного в регистре сравнения и после достижения этого значения сразу сбрасывается. Как результат на выходе образуются выходы одинаковой скважности, но разной частоты.

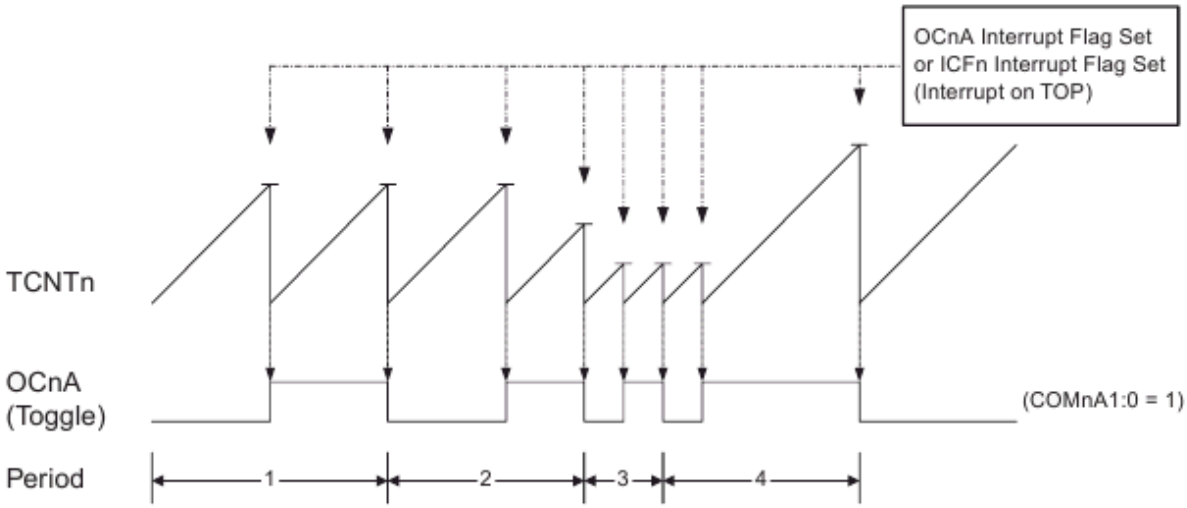


Рис. 6.3. Временная диаграмма работы ШИМ (режим CTC)

Данный режим часто используется, когда необходимо с помощью таймера отсчитывать периоды и генерировать прерывания.

Рассмотрим на примере таймера 1 (TIM1) основные биты TCCR1A и TCCR1B, используемые для настройки ШИМ:

За поведение выводов сравнения OC1A и OC1B отвечают биты COM1A1:COM1A0 и COM1B1:COM1B0 соответственно. Режимы работы выходов приведены в таблице:

Таблица 6.3. Режимы работы выходов OC1A и OC1B

COMxx1	COMxx0	Режим работы выхода
0	0	Вывод не задействован
0	1	Поведение зависит от режима, заданного в битах WGM, и различается для разных МК
1	0	Сброс при совпадении (прямой счет)
1	1	Сброс при обнулении (обратный счет)

Биты WGM11, WGM10, WGM12, WGM13 задают режим работы генератора:

Таблица 6.4. Режим работы ШИМ

WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Режим работы
0	0	0	0	Обычный режим работы
0	0	0	1	PWM, Phase Correct, 8-bit
0	0	1	0	PWM, Phase Correct, 9-bit
0	1	0	0	PWM, Phase Correct, 10-bit
0	1	0	0	CTC (сброс при сравнении)
0	1	0	1	Fast PWM, 8-bit
0	1	1	0	Fast PWM, 9-bit
0	1	1	1	Fast PWM, 10-bit

В таблице 6.4 приведены только основные режимы работы ШИМ, полную таблицу можно найти в документации на конкретный МК

После настройки режима ШИМ, для его запуска необходимо только запустить таймер. Пример программы для работы с ШИМ будет рассмотрен далее.

2. Особенности программирования таймеров

Начнем рассмотрение особенностей программирования таймеров со счетного регистра TCNT. Для 8-разрядного таймера в случае если в этот регистр нужно записать число проблем не возникает, используется обычная команда вывода значения РОН в РВВ (OUT). В случае же использования 16-разрядного таймера счетный регистр состоит из двух регистров TCNTxH и TCNTxL. А так как таймер считает независимо от процессора, то в промежутке между чтением или

записью «половинок» счетного регистра один из них может измениться, например, считаем содержимое младшего байта счетного регистра (TCNTxL) которое в текущий момент времени имеет значение 0xFF, затем считаем содержимое старшего байта (TCNTxH), в это время состояние таймера становится на единицу больше и младший байт обнуляется, в итоге вместо реального значения лежащего в регистрах (0x0100) мы прочтем неверное значение (0x01FF), что совершенно не одно и тоже. При записи числа в счетный регистр может происходить аналогичная ситуация. Поэтому во избежание такой ситуации в16-разрядных таймерах предусмотрен специальный механизм чтения и записи счетного регистра таймера. Рассмотрим его на примере записи в счетный регистр:

```
//Запись в 16-разрядный регистр TCNT
CLI                 //Для атомарности операции запретим прерывания
OUT    TCNTxH, Rx   //Первым пишется старший байт
OUT    TCNTxL, Rx   //Вторым младший
SEI                 //Операция завершена, прерывания можно разрешить
```

Листинг 6.1. Запись в 16-разрядный счетный регистр TCNT

Так как запись в счетный регистр по сути является операцией атомарной то срабатывание прерывание в процессе записи может привести к тому что после записи в регистре окажется фактически «мусор», поэтому прерывания запрещаются на глобальном уровне, далее согласно документации, сначала записывается значение в старший байт регистра TCNT, а затем в младший. Порядок записи менять нельзя. Процедура чтения аналогично, за исключение того что при чтении сначала читается младший байт, а затем старший:

```
//Чтение из 16-разрядного регистра TCNT
CLI                 //Для атомарности операции запретим прерывания
IN     Rx, TCNTxL   //Первым читается младший байт
IN     Rx, TCNTxH   //Вторым старший
SEI                 //Операция завершена, прерывания можно разрешить
```

Листинг 6.2. Чтение из 16-разрядного счетного регистра TCNT

При запуске таймеров несмотря на то что формально все РВВ устанавливаются в 0 при включении питания, фактически для получения точного первого отсчета рекомендуется регистрTCNT обнулить перед первым запуском в работу (не забывая о порядке записи в него).

Зачем писать значения в счетный регистр таймеров? Ответ прост, возможность записи позволяет начинать отсчёт не с нуля и регулировать интервалы счета.

Следующее что стоит учитывать при программировании таймеров это особенности предделителя. Особенность это заключается в том, что при запуске таймера с предделителем установленным, например, на 1024, то первый отсчет не обязательно придется через 1024 импульса. Это зависит от того в каком состоянии предделитель находился в предыдущий момент времени (то есть до какого значения он досчитал), предделитель один и работает независимо от того включен таймер или нет. Таким образом для точности отсчета предделитель нужно сбрасывать, но стоит учитывать, что предделитель один для всех таймеров и при сбросе его значения может сбиться выдержка до следующего отсчета у других работающих в настоящий момент времени таймеров. Например, может возникнуть ситуация, когда один таймер работает на выводе предделителя 1:64, а второй на 1:1024, и для точности работы первого таймера предделитель сбрасывается в цикле чаще чем один раз в 1024 такта, таким образом второй таймер работать не будет, хотя и должен согласно всем настройкам.

Для сброса предделителя необходимо записать бит PSRx регистра SFIOR. Бит PSRx будет сброшен автоматически на следующем такте МК:

```
//Очистка предделителя таймеров
      LDI R16, (1<<PSR2) //Запишем значение в регистр R16
      OUT SFIOR, R16      //Выведем значение в PBB
```

Листинг 6.3. Сброс предделителя таймеров

3. Простейший секундомер

Рассмотри программу простейшего секундомера, который фактически умеет считать секунды от 0 до 9, результат выводится на семисегментный дисплей с дешифратором, а запуск или остановка производится нажатием на соответствующую кнопку. Как обычно в процессе разбора программного кода будет опущено подробное описание всех ранее рассмотренных моментов. Подробно будут рассмотрены собственно сама настройка таймера и использование прерываний.

Кратко ход программы можно описать так, по нажатию на кнопку запускается таймер, при его переполнении срабатывает прерывание, в обработчики прерывания значение секунд увеличивается на 1, и выводится на семисегментный дисплей, МК ожидает следующего срабатывания прерывания или нажатия на кнопку.

Для удобства программирования определим переменные и константы, которые будут использоваться в программе:

```
//Определение переменных (директива .def)
 .def TMP     = R16        //Временная переменная
 .def CNT     = R17        //Счетчик
 .def NUM       = R18              //Значение выводимое на дисплей

//Определение констант   (директива .equ)
.equ MAX_V      = 245       //Значение регистра счетчика для таймера(до 1 секунды)
.equ MAX_NUM  = 10          //Значение для того чтобы вовремя обнулить значение на дисплее
```

Листинг 6.4. Определение констант и переменных

Рассмотрим используемые константы (листинг 6.4), первая MAX_V, указывает сколько раз должен переполнится таймер для того чтобы прошла одна секунда, это значение рассчитано из соображений что при тактовой частоте МК 4МГц и коэффициенте предделителя 1:64, частота работы таймера будет равна 62500Гц, таким образом 8-разрядный таймер будет переполняться порядка 62500/256=244,1 раз в секунду.

Вторая константа MAX_NUM используется для ограничения вывода символов на дисплей, то есть для перехода после 9 к 0.

Так как в программе будет использоваться прерывание, посмотрим, как это указывается в векторе прерываний:

```
.ORG   OVF1addr       ; Timer/Counter1 Overflow
RETI
.ORG   OVF0addr       ; Timer/Counter0 Overflow
RJMP TIM0OVF
.ORG   SPIaddr        ; Serial Transfer Complete
RETI
.ORG   URXCaddr       ; USART, Rx Complete
RETI
```

Листинг 6.5. Фрагмент вектора прерываний.

Как видно из листинга 6.5, для того чтобы задать обработчик прерывания, в векторе прерываний для необходимого прерывания команда RETI заменятся на команду перехода RJMP с указанием метки, имя метки может быть любым, в

нашем случае метка носит говорящее название TIM0OVF. Сам обработчик прерывания рассмотрим позже.

После инициализации стека и настройки портов ввода/вывода, перейдем к настройке таймера. Используем 8-разрядный таймер (TIM0).

```
//Настройка таймера TIM0
	LDI TMP, (1<<TOIE0) //Разрешим прерывания по переполнению таймера
	                    //(по умолчанию запрещены)
	OUT TIMSK, TMP
	LDI CNT,MAX_V       //Запишем в регистр-счетчик значение для задержки в 1 сек

	SEI                 //Разрешим прерывания
```

Листинг 6.6. Настройка таймера

Так как нам необходимо прерывание по переполнению таймера, а каждое прерывание должно быть разрешено явно, то установим бит разрешения прерывания по переполнению TOIE0 регистра TIMSK в 1. Затем загрузим в регистр CNT значение константы, отвечающей за количество переполнений таймера. И так как в программе будут использоваться прерывания глобально разрешим их командой SEI. Особо стоит обратить внимание на то, что таймер еще не запущен, так как настройки предделителя не выставлены (листинг 6.6).

Далее начинается выполнение основного бесконечного цикла программы, в которой ожидается нажатие одной из кнопок. При нажатии одной из кнопок (СТАРТ или СТОП) происходит переход к обработчику кнопки. Рассмотрим их:

```
//Нажата кнопка "Старт"
SB_START:
SBIS PIND,0         //Кнопка опущена?
RJMP SB_START //Нет, ждем отпускания, иначе продолжаем выполнять программу

//Таймеры запускаются выставлением значения в предделителе (биты CSxx регистра TCCR0)
LDI TMP, (1<<CS00)|(1<<CS01)      // Предделитель (/64)
OUT TCCR0,TMP                     // Записываем значение в регистр

RJMP MAIN
```

Листинг 6.7. Обработчик кнопки СТАРТ

При обработке кнопки СТАРТ (листинг 6.7) Запускается таймер TIM0. Запуск производится установкой бит CSxx в регистр настройки таймера TCCRx, в нашем случае необходимо запустить таймер с предделителем 1:64, согласно таблице 6.1 для этого необходимо установить биты CS00 и CS01 в 1, сразу после установки этих бит в регистр TCCR0 таймер запускается и работает независимо

от работы МК, обработка нажатия кнопки завершается, и программа переходит к выполнению бесконечного цикла, ожидая следующего нажатия кнопки или прерывания.

Обработка кнопки СТОП происходит аналогичным образом (листинг 6.8), за исключением того, что таймер выключается, остановка счета таймера производится обнулением битов регистра TCCR0 отвечающих за предделитель, то есть фактически проводится операция обратная запуску таймера.

```
SB_STOP:
SBIS PIND,0          //Кнопка опущена?
RJMP SB_STOP  //Нет, ждем отпускания, иначе продолжаем выполнять программу

//Останавливаем таймер обнуляя биты предделителя
LDI TMP, (0<<CS00)|(0<<CS01)      // Предделитель обнуляем
OUT TCCR0,TMP                     // Записываем значение в регистр

RJMP MAIN
```

Листинг 6.8. Обработчик кнопки СТОП

Каждый раз при переполнении таймера в регистре прерываний таймера (TIFR) выставляется флаг прерывания, и если прерывания разрешены, то обычный ход программы прерывается и начинается обработка прерывания. При срабатывании прерывания автоматически сохраняется в стек адрес возврата, а также глобально запрещаются все прерывания. При выходе из прерывания по команде RETI, глобальный запрет на прерывания автоматически сбрасывается. **Регистр SREG автоматически при возникновении прерывания не сохраняется.**

Рассмотрим обработчик прерывания подробно:

```
//Сегмент обработчиков прерываний
TIM0OVF:
       DEC CNT              //Уменьшаем счетчик
       BREQ DELAY1S         //Счетчик равен 0? (переполнение произошло 256 раз)
       RETI                 //Нет, ждем
DELAY1S:
       LDI CNT,MAX_V        //Запишем в регистр-счетчик новое значение для задержки в 1 сек
       INC NUM              //Увеличим число которое собираемся выводить
       CPI NUM,MAX_NUM      //Дошли до предельного значения?
       BREQ ZERO            //Да, обнуляем
       OUT PORTC,NUM        //Нет, выводим на дисплей
RETI
ZERO:
       CLR NUM              //Обнуляем значение
       OUT PORTC,NUM        //Выводим на дисплей
RETI
```

Листинг 6.9. Обработчик прерывания по переполнению таймера.

Первым делом уменьшаем счетчик количества переполнений таймера, и проверяем равен ли он 0, в случае если нет, то выходим из прерывания по команде RETI и продолжаем выполнение программы до следующего прерывания. В ином случае (CNT=0), таймер переполнился необходимое количество раз (секунда отсчитана) и происходит переход к метке DELAY1S: в которой счетчик CNT устанавливается в значение MAX_V, затем командой инкремента увеличивается на единицу значение числа, которое будет выводится на дисплей, а так же происходит проверка нового числа, на максимальное значение, если максимальное значение не достигнуто, выводит число на дисплей (OUT), если достигнуто, то перейдя по метке (ZERO:) значение числа обнуляется и выводится на дисплей. После вывода числа по команде RETI обработка прерывания из листинга 6.9 заканчивается и продолжается выполнение программы с адреса возврата, взятого из стека.

Как можно было обратить внимание в данной программе в порт напрямую выводится двоичное число, без какой-либо обработки. Это возможно так как предполагается что в устройстве используется семисегментный дисплей с дешифратором. Такой дисплей имеет 4 управляющих вывода, и автоматически преобразуется двоичный код в символы от 1 до F, для того чтобы получить на таком дисплее 0, нужно просто подать на него питание.

4. Таймеры в режиме ШИМ

Для примера настроим таймер (TIM1) для работы в режиме ШИМ, на два выхода OC1A и OC1B с длительностью импульсов 1/2 и 1/3 периода соответственно. Сначала разберем настройку таймера в режиме Fast PWM 8-bit, а затем в режиме Phase correct PWM 8-bit.

Как ранее было сказано главный настроечный регистр таймера это TCCR, установим режим работы ШИМ согласно таблицам 6.3 и 6.4. (листинг 6.10) Установим сброс при обнулении (обратный счет ШИМ) установив биты COM1A0, COM1A1, COM1B0, COM1B1. За выбор режима работы ШИМ отвечают биты WGM, для задания Fast PWM 8-bit установим биты WGM10, WGM12. А затем

запустим таймер включив предделитель (1<<CS10). При записи битов в настроечный регистр стоит обратить внимания на то что в 16-разрядном таймере их два, TCCR1A и TCCR1B, какие биты в них есть можно посмотреть либо в документации на МК, либо в окне отладчика AVR Studio. Далее соблюдая порядок записи в 16-разрядные регистры сравнения (сначала старший, затем младший байт), запишем значения для необходимой скважности:

```
///Настройка таймера TIM1 для работы в режиме ШИМ (FAST PWM 8-bit)
	LDI TMP, 2<<COM1A0|1<<WGM10|0<<WGM11|2<<COM1B0 //Настроим ШИМ на сброс при обнулении
	OUT TCCR1A, TMP

	LDI TMP, 1<<WGM12|1<<CS10|0<<WGM13
	OUT TCCR1B, TMP
	//Запишем необходимые значения в регистры сравнения (не забывая про порядок записи!)
	LDI TMP,0
	OUT OCR1AH,TMP
	LDI TMP,128
	OUT OCR1AL,TMP

	LDI TMP,0
	OUT OCR1BH,TMP
	LDI TMP,85
	OUT OCR1BL,TMP
```

Листинг 6.10. Настройка TIM1 на работу в режиме Fast PWM 8-bit

Прошив МК, осциллографом можно проверить работу ШИМ подключившись к ножкам OC1A и OC1B. (рисунок 67)

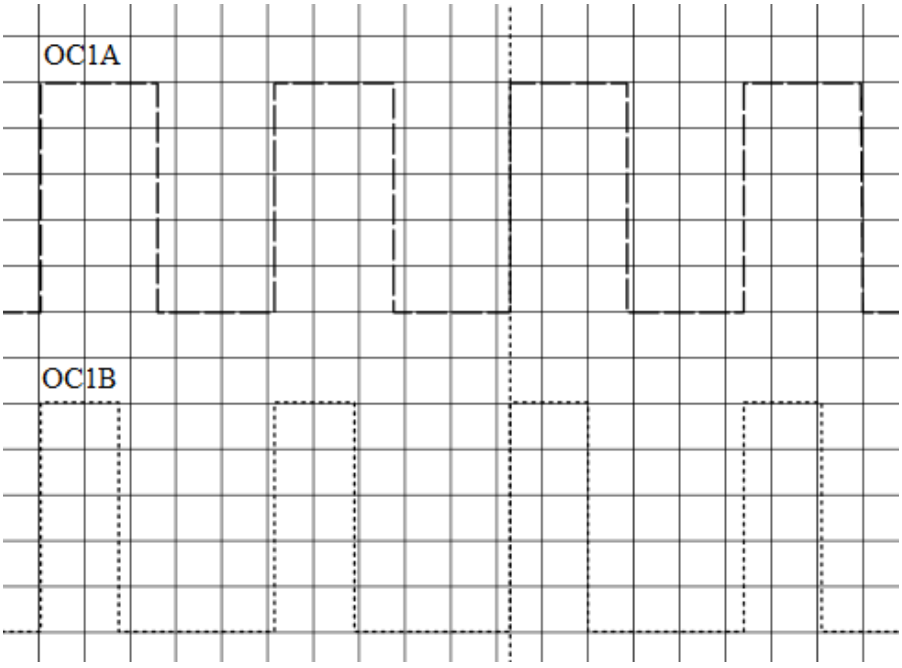


Рис. 6.4. Работа ШИМ в режиме Fast PWM 8-bit

Из рисунка 6.4 видно, что импульсы идут с заданной длительностью, а также происходит сдвиг фаз, который должен отсутствовать в режиме Phase correct PWM 8-bit. Изменим программу для работы в этом режиме сбросив бит WGM12 регистра TCCR1B:

```
LDI TMP, 0<<WGM12|1<<CS10|0<<WGM13
OUT TCCR1B, TMP
```

После перепрошивки МК можно будет увидеть следующую картинку:

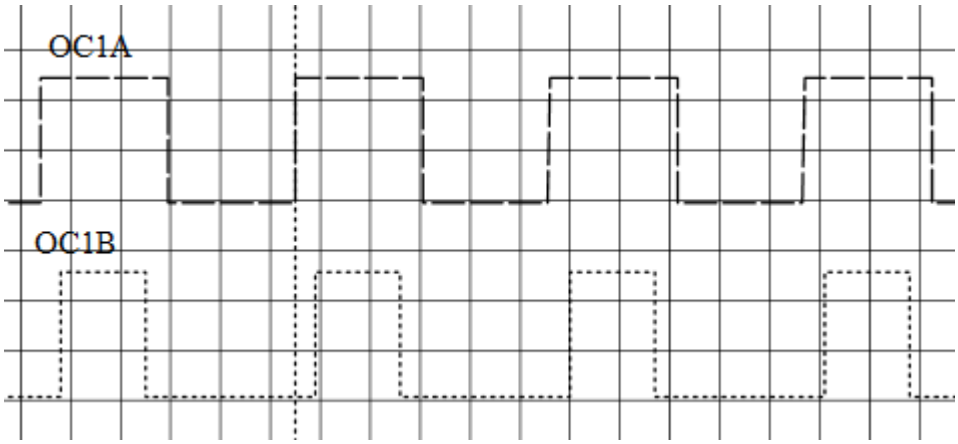


Рис. 6.5. Работа ШИМ в режиме Phase correct PWM 8-bit

Из которой видно, что сдвига фаз не происходит. Такой режим работы часто используется для создания многофазных ШИМ сигналов, ведь так как сигналы с разных каналов и разной скважности будут совпадать по фазе, угол фазового сдвига между двумя сигналами не будет сбиваться.

Практическое применение ШИМ рассмотрим в главе 11 на примере управления сервоприводом.

Задания для самостоятельной работы

На основании выданных преподавателем индивидуального задания выполните необходимые исследования.

Составьте отчет по выполненным заданиям.

ТЕМА № 7. ПЕРЕДАЧА И ПРИЕМ ДАННЫХ UART

Вопросы, рассматриваемые в данной теме:

1. Краткое описание протокола
2. Реализация UART в МК AVR
3. Программа приема-передачи данных через UART
4. Сопряжение МК и ПК

Краткие теоретические сведения

1. Краткое описание протокола

Большинство современных МК содержат в себе модули асинхронного приёмопередатчика(UART) или асинхронно-синхронного (USART). Так как в работе эти два передатчика ничем кроме наименования нескольких регистров не отличаются, в дальнейшем чтобы избежать путаницы будем называть этот модуль UART.

Протокол работы передатчика и формат кадра достаточно прост (рис. 7.1):

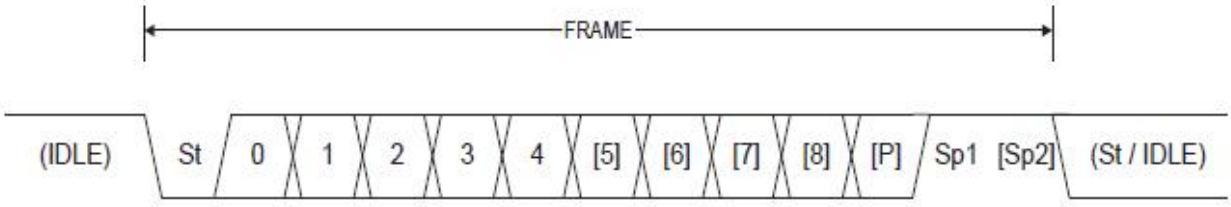


Рис.7.1. Протокол работы и формат кадра UART

St – Старт-бит, всегда низкий уровень

(n) – Биты данных (с 0 по 8)

P – Бит четности

Sp – Стоп-бит, всегда высокий уровень

[] – настраиваемые параметры

IDLE – Ожидание на линии высокий уровень, прием/передача отсутствуют)

Изначально линия находиться в состоянии IDLE, то есть прием/передача отсутствует, и линии подтянута к высокому уровню. Для того чтобы начать

прием/передачу, передатчик просаживает линии к низкому уровню, то есть отправляет в линии старт-бит (St), определив, что линии просела, приемник выдерживает интервал T1, а затем считывает первый и последующие биты с интервалом времени T2. Дальше работа протокола зависит от его настройки, для контроля передачи данных, после всех битов, данных может стоять бит четности (P), за которым обязательно следует один стоп бит(Sp1), либо для большей надежности два стоп-бита (Sp1 и Sp2). После стопового бита (битов), прием передача заканчивается, и начинаете или передача нового кадра со стартового бита (St) и протокол переходит в режим IDLE.

Естественно все настройки протокола должны быть произведены до начала приема/передачи. Самым популярным вариантом настройки является – 8 бит данных, 1 старт-бит, 1 стоп-бит, проверка четности отсутствует.

Подключение двух устройств по протоколу UART между собой (рис. 7.2) выполняется соединением передатчика одного устройство с приемником другого.

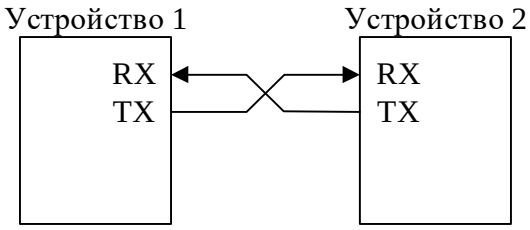


Рис 7.2. Подключение двух устройств по USART между собой

Кстати по такому же протоколу работает COM порт компьютера, вопрос сопряжения МК и ПК будет рассмотрен далее.

2. Реализация UART в МК AVR

UART в МК AVR реализован аппаратно, поэтому вручную следить за состоянием линии, форматом кадра, и управляющими битами не нужно. В целом постоянно следить за состоянием UART не нужно, так как все его обслуживание (прием и отправка данных) можно повесить на прерывания.

Для принятых и передаваемых данных существует регистр UDR (UART Data Register), физически это два разных регистра, (один для приема, а второй

для передачи) имеющие один адрес. То есть для того чтобы передать байт, необходимо записать его в регистр UDR, а для того что бы принять, его необходимо считать из регистра.

О том, что байт поступил в регистр UDR можно определить по срабатыванию прерывания, которое вызывается каждый раз, когда приемник заканчивает получать новый байт. Так как передача идет довольно медленно, и для приема и передачи доступен только один общий регистр UDR, перед записью в него нужно убедиться, что он свободен и готов к приему нового байта, для этого существует бит UDRE, который кроме того, что говорит о готовности к записи в буфер, вызывает аппаратное прерывание по опустошению буфера.

Теперь рассмотрим настройку UART. Все основные параметры хранятся в регистрах конфигурации (UCSRA, UCSRB, UCSRC), а скорость работы приемопередатчика задается записью значений в регистровую пару UBBRH:UBBRL.

Подробнее рассмотрим основные биты регистров конфигурации:

1. *Регистр UCSRA:*

Биты RXC и TXC – флаги завершения приема или передачи соответственно, бит RXC устанавливается, когда принятый байт попадает в UDR, бит TXC устанавливается после отправки байта и опустошения UDR. Биты RCX и TXC.генерируют прерывание по завершению приема (Rx Complete) или передачи (Tx Complete).

Бит UDRE указывает на то что регистр UDR приемника пуст и в него можно записывать новое значение. Сбрасывается аппаратно, генерирует порывание UDR пуст (Data Register Empty).

Бит U2X – бит удвоения скорости при работе в асинхронном режиме.

Биты FE, OR, PE – биты ошибки кадрирования, переполнения буфера и контроля четности соответственно.

2. *Регистр UCSRB:*

Биты разрешения приема (RXEN) и передачи (TXEN).

Биты разрешения прерываний по завершению приема (RXCIE), передачи (TXCIE) и опустошению буфера передачи (UDRIE).

Бит UCSZ2 совместно с битами UCSZ0 и UCSZ1 находящимися в регистре UCSRC, задают формат передаваемых данных.

3. Регистр UCSRC:

Так как регистры UCSRC и UBRRH находятся в одном адресном пространстве, того чтобы записать в регистр UCSRC в старшем бите URSEL должна быть записана единица, в противном случае запись будет идти в регистр UBRRH.

Остальные биты задают различные параметры протокола, такие как число стоп-битов, контроль четности, формат посылки, при необходимости их можно посмотреть в документации на МК. Режим по умолчанию – 1 старт-бит, 1 стоп-бит, проверка четности отсутствует. Формат посылки равный 8 битам можно задать, записав в биты UCSZ0 и UCSZ1 единицу.

Скорость обмена и значение необходимое для записи в UBRR соответствующее режиму работы может быть вычислено по формулам из таблицы 7.1:

Таблица 7.1. Вычисление значений для необходимой скорости обмена

Режим работы	Формула расчета скорости передачи	Расчет значения регистра UBRR (для требуемой скорости)
Асинхронный режим работы (U2X=0)	$BAUD = \frac{f_{osc}}{16(UBRR + 1)}$	$UBRR = \frac{f_{osc}}{16BAUD} - 1$
Асинхронный режим работы с удвоенной скоростью (U2X=1)	$BAUD = \frac{f_{osc}}{8(UBRR + 1)}$	$UBRR = \frac{f_{osc}}{8BAUD} - 1$
Синхронный режим работы	$BAUD = \frac{f_{osc}}{2(UBRR + 1)}$	$UBRR = \frac{f_{osc}}{2BAUD} - 1$

где, BAUD – скорость передачи (бит в секунду), f_{OSC} – частота МК

UBRR – значение в регистрах UBRRH и UBRRL (0-4095)

3. Программа приема-передачи данных через UART

Рассмотрим программу, выводящую по протоколу UART в терминал запрос на ввод значения от 0 до 9, и при вводе значения выводящее его на индикатор. Для вывода строки символов в порт будет использоваться прерывание по наличию в регистре UDR данных для передачи (UD_OK) и прерывание по окончанию приема (RX_OK).

Рассмотрение программы начнем с используемых переменных и констант:

```
//Определение переменных (директива .def)
 .def TMP     = R16        //Временная переменная

//Определение констант   (директива .equ)
 .equ  XTAL = 4000000                          //Частота контроллера
//Расчет константы для требуемой частоты UART
 .equ  baudrate = 9600                         //Необходимая частота UART
 .equ  baud_const = XTAL/(16*baudrate)-1       //Константа для частоты


//Сообщение для вывода на дисплей, хранится во Flash-памяти
ServiceMsg: .db "ENTER NUMBER (0-9):",0x0D, 0,0
```

Листинг 7.1. Инициализация констант и переменных

В листинге 5.5 инициализируется только одна переменная TMP, а из констант автоматически в зависимости от требуемых частот МК (XTAL) и передачи данных (baudrate) рассчитывается значение константы для записи в регистр UBRR. Строковый массив (ServiceMsg) располагается во flash-памяти (0x0D – переход на новую строку, 0 – конец массива).

Инициализируем UART (листинг 7.2), задав частоту приемопередатчика записав ранее рассчитанную константу baud_const в регистр UBRR, разрешив прерывания по приему и передаче (но запретив прерывание по наличию данных в регистре UDR), и настроив параметры кадра (8 бит, без проверки четности):

```
//Инициализация UART:
       LDI    TMP, LOW(baud_const)             //Регистр частоты
       OUT    UBRRL,TMP
       LDI    TMP, HIGH(baud_const)            //Регистр частоты
       OUT    UBRRH,TMP

       LDI    TMP, (1<<RXEN)|(1<<TXEN)|(1<<RXCIE)|(1<<TXCIE)|(0<<UDRIE) //Прерывания разре-
шены, прием-передача разрешен
       OUT    UCSRB,TMP
       LDI    TMP, (1<<URSEL)|(1<<UCSZ0)|(1<<UCSZ1) //Формат кадра – 8 бит, без проверки
четности
       OUT    UCSRC, TMP
```

Листинг 7.2. Инициализация UART

Для вывода строки в терминал (листинг 7.3) установим указатель на массив содержащий требуемую строку, и разрешим прерывание по наличию данных в регистре UDR:

```
MAIN:
	LDI ZL, LOW (ServiceMsg*2) //Установим указатель на область памяти с сообщением
	LDI ZH, HIGH(ServiceMsg*2)
H_MSG:
//Разрешим прерывание при наличии данных для передачи
	LDI		TMP,(1<<RXEN)|(1<<TXEN)|(1<<RXCIE)|(1<<TXCIE)|(1<<UDRIE)
	OUT		UCSRB, TMP
BREQ STOP		//Если данные для передачи закончились, останавливаем работу
RJMP H_MSG		//Иначе выводим следующий символ

STOP:
RJMP STOP
```

Листинг 7.3. Вывод строки в терминал

Вывод будет посимвольно продолжаться в прерывании (листинг 7.4) до окончания строки (символ 0 в массиве), и далее программа, перейдя по метке STOP уйдет в бесконечный цикл ожидая приема данных.

```
UD_OK:		//В регистре UDR есть данные для передачи
	LPM TMP, Z+1
	CPI TMP,0
	BREQ STOP_TX
	OUT UDR, TMP
RETI

STOP_TX:
	//Запретим прерывание при наличии данных для передачи
	LDI		TMP,(1<<RXEN)|(1<<TXEN)|(1<<RXCIE)|(1<<TXCIE)|(0<<UDRIE)
	OUT		UCSRB, TMP
	SEZ		//Установим 1 в бит Z регистра SREG
RETI
```

Листинг 7.4. Прерывание по наличию бита в UDR

Подробнее рассмотрим прерывание UD_OK (листинг 7.4), в нем в регистр TMP считывается значение, лежащее по адресу, на который указывает указатель Z, и значение Z инкрементируется, затем проверяется условие конца массива, и, если оно выполнено, прерывание по наличию данных в регистре UDR запрещается. В противном случае символ из массива выводится в терминал.

Прием значений из терминала организован с помощью прерывания прием завершен (листинг 7.5), в котором полученный байт сохраняется в регистр TMP и выводится на дисплей:

```
RX_OK:              //Прием завершен
IN TMP,UDR          //Забираем число из UDR
OUT PORTC,TMP //Выводим в порт
RETI
```

Листинг 7.5. Прием данных

4. Сопряжение МК и ПК

Один из самых простых способов сопряжения МК и ПК является соединение их через COM-порт согласовав уровни протоколов UART и RS-232.

В UART используется положительная логика, при которой логическая единица соответствует высокому уровню сигнала, и логический ноль – низкому. В RS-232 биты передаются разнополярными уровнями напряжения, при этом инвертированными относительно уровней UART, так логическая единица соответствует отрицательному уровню (-3…-12В), логический ноль положительному уровню (0…12В).

Таким образом сначала нужно согласовать физические параметры протоколов, это можно сделать как использовав готовые решение на базе микросхем типа MAX232, MAX202, ADM202 и других, так и сделав собственный преобразователь-инвертор напряжения. Схемы таких преобразователей нетрудно найти в Интернете. В случае отсутствия COM-порта хорошо зарекомендовали себя преобразователи RS-232/USB на базе микросхем FTDI.

Согласовав физические параметры протокола, нужно также согласовать программную часть, то есть установить одинаковые формат кадра и скорость в каждом приемо-передатчике.

Такой вариант сопряжения с ПК является наиболее простым, так как сопряжение с ПК по другим протоколам зачастую требует более дорогостоящих преобразователей интерфейсов, и зачастую более сложную программную часть.

Задания для самостоятельной работы

На основании выданных преподавателем индивидуального задания выполните необходимые исследования.

Составьте отчет по выполненным заданиям.

ТЕМА №8. АНАЛОГОВЫЙ КОМПАРАТОР И АЦП

Вопросы, рассматриваемые в данной теме:

1. Использование аналогового компаратора
2. Использование АЦП
3. Методы повышения точности АЦП

Краткие теоретические сведения

1. Использование аналогового компаратора

В МК AVR существует два основных способов работы с аналоговыми сигналами, это использование аналогового компаратора и 10-разрядного многоканального АЦП. Данные периферийные устройства входят во все МК AVR. В данной теме опустим подробности аналогово-цифровых преобразований, и разберем практические вопросы использования аналогового компаратора и АЦП.

Начнем с аналогового компаратора. Аналоговый компаратор сравнивает две величины напряжения на своих входах (положительном и отрицательном) и в зависимости от их соотношения устанавливает свой выход в одно из логических состояний. В МК AVR эти входы называются AIN0 (положительный) и AIN1 (отрицательный). Согласно документации, ошибка аналогового компаратора AtMega8 составляет не более 40мВ, а время отклика не более 0.5 мкс. Напряжение питания на его входах не может быть больше чем напряжение питания МК.

Управление аналоговым компаратор осуществляется регистром ACSR, рассмотрим его биты:

Бит ACD управляет включением компаратора, причем его нулевое состояние означает что компаратор включен (режим по умолчанию), поэтому если в схеме компаратор не используется, и вопрос энергопотребления важен, компаратор необходимо отключать.

Бит ACBG отвечает за подключение к положительному входу компаратора (AIN0) источника опорного напряжения величиной 1.22В, данный источник довольно неточен и погрешность его значений может достигать до 0.1В.

Бит ACIE отвечает за разрешение прерывания при смене состояния выхода компаратора.

Биты ASIS1 и ASIS0 отвечают за событие вызывающее прерывание, все события приведены в таблице 8.1:

Таблица 8.1. События вызывающие прерывания

ASIS1	ASIS0	Событие
0	0	Прерывание по любому перепаду уровней (по умолчанию)
1	0	Прерывание по низкому уровню на выходе
1	1	Прерывание по высокому уровню на выходе

Бит ACO указывает на текущее состояние выхода компаратора и в любой момент может быть прочитан.

С учетом вышесказанного рассмотрим простую программу: на положительный вход компаратора (AIN0) подается опорное напряжение равное 3.3В, на отрицательный вход (AIN1) какое-то напряжение в диапазоне от 0 до 5 вольт, необходимо в случае снижения напряжения ниже опорного уровня подавать сигнал зажигая светодиод.

Инициализируем компаратор:

```
//Настроим аналоговый компаратор
	LDI TMP, (1<<ACIE)
	OUT ACSR,TMP
```

Листинг 8.1. Инициализация аналогового компаратора

Так как компаратор по умолчанию запущен, разрешим его прерывания (листинг 8.1) установив бит ACIE регистра ACSR, так как биты ASIS1:ASIS0 не изменялись то прерывание будет срабатывать при любом перепаде уровня на выходе компаратора.

Пока напряжение на входе AIN1, выше чем опорное напряжение на входе AIN0, выход компаратора (бит ACO) не установлен, как только напряжение на

входе AIN1 упадет ниже опорного напряжение, бит ACO установится и сработает прерывание:

```
ACI_R:
SBIS ACSR,ACO //Если бит ACO установлен, зажигаем LED
RJMP RES_LED  //Иначе гасим
SBI PORTB,0
RETI

RES_LED:
CBI PORTB,0
RETI
```

Листинг 8.2. Прерывание аналогового компаратора

В прерывании (листинг 8.2) проверяется установлен ли бит ACO, если да, то напряжение на входе AIN1 ниже опорного, и необходимо зажечь светодиод, в противном случае, бит ACO сброшен и светодиод необходимо потушить.

Другие неочевидные способы использования компаратора такие как реализация интегрирующего АЦП на компараторе или подключение компаратора ко входу захвата таймера TIM1 для формирования входных импульсов и измерения длительности временных интервалов или подсчета событий можно найти в документации и специализированной литературе посвященной аналоговым измерениям.

2. Использование АЦП

АЦП в МК AVR может работать в двух основных режимах, в режиме непрерывного преобразования, когда после окончания преобразования сразу начинается следующее, и в режиме одиночного преобразования, когда для начала преобразования необходимо подавать команду на запуск.

Рассмотрим основные управляющие и конфигурационные биты АЦП разделив их для удобства на группы.

1. *Включение, запуск и выбор режима преобразования АЦП:* включение АЦП производится установкой бита ADEN конфигурационного бита ADCSR(ADCSRA), а выбор режима преобразования с помощью бита ADFR(ADATE), при установке которого АЦП конфигурируется режим непрерывного преобразования. Если выбран режим запуска по умолчанию (смотри

следующий пункт), то преобразование запускается записью бита ADCS. В непрерывном режиме преобразования, бит ADCS запустит первое преобразования, а остальные будут происходить автоматически. В режиме однократного преобразования, установка бита ADCS запускает однократное преобразование.

2. *Выбор режима запуска преобразования*: возможность выбора режима запуска преобразования есть не во всех МК, там, где она есть выбор режима осуществляется установкой битов ADTS2…0 регистра SFIOR согласно таблице:

Таблица 8.2. Режим запуска преобразования

ADTS2	ADTS1	ADTS0	Режим запуска преобразования
0	0	0	Преобразования в непрерывном режиме или по ручному запуску (по умолчанию)
0	0	1	Запуск АЦП от аналогового компаратора
0	1	0	Запуск от внешнего прерывания INT0
0	1	1	По совпадению таймера T0
1	0	0	По переполнению таймера T0
1	0	1	По совпадению таймера T1
1	1	0	По переполнению таймера T1
1	1	1	«Захват» таймера T1

3. *Тактовая частота АЦП:* согласно документации, оптимальная скорость преобразования лежит в диапазоне 50-200кГц, и выбирается установкой значений в биты предделителя ADPS2…0 регистра ADCSR(ADCSRA). Коэффициенты устанавливаются по степеням двойки, то есть все нули в трех битах соответствуют коэффициенту деления 2, все единицы 128.

4. *Выбор каналов и режимов их взаимодействия:* сигнал в АЦП подается через мультиплексор, а количество каналов зависит от типа МК. Выбор входа с которого будет сниматься аналоговый сигнал осуществляется битами MUX3…MUX0 регистра ADMUX. Записанное в эти регистры число определяет выбранный канал. Кроме того, существует несколько служебных наборов битов, некоторые из которых будут рассмотрены далее.

5. *Выбор источника опорного напряжения:* битв REFS1…REFS0 позволяют выбрать источник опорного напряжения согласно таблице:

Таблица 8.3. Выбор источника опорного напряжения

REFS1	REFS0	Источник опорного напряжения
0	0	Внешний источник (AREF) (по умолчанию)
0	1	AV_{CC}
1	1	Встроенный 2.56В

Напряжение внешнего источника питания, подключенного к выводу AREF должно лежать в пределах от 2В до напряжения питания аналоговой части AVcc, при этом напряжение питания аналоговой части не должно отличаться от напряжения питания цифровой части на величину большую чем 0.3В.

Для вариантов использования AVcc и встроенного источника опорного напряжения рекомендуется к ножке AREF подключить фильтрующий конденсатор. Так же стоит отметить что встроенный источник опорного напряжения имеет достаточно неточен и может иметь значение от 2.4 до 2.7В.

Кроме источника опорного напряжения величиной 2.56В можно подключить источник опорного напряжения величиной 1.22В установив биты MUX3…0=1110.

6. *Прерывания и результат преобразования:* бит ADIE регистра ADCSR(ADCSRA) разрешает прерывание по окончанию преобразования, в котором можно сохранить результат преобразования и провести над ним какую-либо математическую обработку.

Результат преобразования автоматически записывается в регистровую пару ADCH:ADCL, из которых фактически используются только 10 битов. При этом результат можно выровнять как по правому краю (режим по умолчанию), тогда два старших бита результата будут в ADCH, остальные в ADCL, так и по левому (установив бит ADLAR регистра ADMUX):

Выравнивание по правому краю		Выравнивание по левому краю	
ADCH	ADCL	ADCH	ADCL
[x][x][x][x][x][x][9][8]:	[7][6][5][4][3][2][1][0]	[9][8][7][6][5][4][3][2]:	[1][0][x][x][x][x][x][x]

Учитывая то что зачастую в младших разрядах только мусор и шумы, имеет смысл делать выравнивание и забирать весь результат из регистра ADCH, то есть АЦП будет фактически работать в 8-битном режиме. Если же используются все разряды, то необходимо соблюдать порядок чтения этих регистров, сначала младший (ADCL) потом старший (ADCH), иначе можно получить значения состоящее в котором старший байт будет из текущего измерения, а младший, например, из следующего.

Кроме вышеописанных режимов АЦП может работать в дифференциальном режиме. В этом режиме АЦП измеряет разность напряжение между двумя каналами с последующим умножением на коэффициент усиления.

Пример использования АЦП рассмотрим на программе, которая будет снимать значение с 0 канала АЦП, преобразовывать полученное значение и выводить его целую часть на семисегментный дисплей.

По традиции начнем рассмотрение программы с определения переменных и констант:

```
//Определение переменных (директива .def)
 .def TMP           =R16          //временная переменная
 .def ADC_Res       =R17          //результат АЦ преобразования

//Определение констант   (директива .equ)
.equ ADC_Const  = 5
//Сегмент ОЗУ (RAM)
```

Листинг 8.3. Определение переменных и констант

Результат преобразования будет хранится в переменной ADC_Res, а для временных нужд воспользуемся TMP. Константу преобразования АЦП рассчитаем из тех условий что при диапазоне измерений от 0 до 5В и разрядностью АЦП шаг измерений будет равен 0.02В. Для того чтобы работать с вещественными числами в МК (например, умножать) необходимо это число «загнать» в диапазон целых чисел, умножив его на число степень двойки, например, на 2^8 и округлив до целого, таким образом умножив 0.02 на 2^8 округлив получим константу (ADC_Const) равную 5.

Чтобы перейти к реальному значению, нужно будет умножить получено значение с АЦП на константу ADC_Const и разделить на степень двойки сдвигом вправо.

Настроим АЦП (листинг 8.4) на работу в режиме непрерывных измерений с предделителем 64, что при частоте работы МК равной 4МГц, даст частоту работы АЦП 62.5КГц, что укладывается в диапазон. Данные будем считывать с 0 канала, за эталонное напряжение примем напряжение питания МК. Результат выровняем по левому краю. Разрешим прерывания, включим АЦП и дадим старт преобразованию:

```
//Настройка АЦП
//ADLAR-Выравнивание по левому краю
//REFS- Выбор источника опорного напряжения
//MUX-  Выбор канала АЦП
       LDI Temp, (1<<ADLAR|1<<REFS0)     //Устанавливаем нужные нам биты в 1
       OUT ADMUX,TMP                     //Записываем значение в управляющий регистр

//Инициализируем АЦП
//ADEN-Включить АЦП
//ADSC-Начать преобразование
//ADFR-Режим непрерывных измерений
//ADIE-Разрешение прерывания
//ADPS-Предделитель тактовой частоты АЦП
       LDI TMP, (1<<ADEN|1<<ADSC|1<<ADFR|1<<ADIE|1<<ADPS0|1<<ADPS2)
       OUT ADCSRA, TMP
```

Листинг 8.4. Инициализация АЦП

Вся дальнейшая работа программы будет заключаться получение результата преобразования и его обработке в прерывании по окончанию преобразования (ADC_END):

```
//Прерывание по окончанию АЦ преобразования
ADC_END:
LDI TMP,ADC_Const          //Запишем константу для получения реального значения с АЦП
IN  ADC_Res,ADCH           //Перенесем полученное значение АЦП в РОН
MUL ADC_Res,TMP            //Умножим на константу (результат в R0-R1)
MOV ADC_Res,R1             //Сдвиг на 8 разрядов вправо это фактически перенос старшего раз-
ряда в младший)
OUT PORTB, ADC_Res         //Выведем полученное значение
RETI
```

Листинг 8.5. Прерывание по окончанию преобразования

Здесь (листинг 8.5) сначала во временную переменную TMP записывается константа ADC_Const, для дальнейшего математического преобразования, затем результат преобразования сохраняется в ADC_Res, и умножается на

ADC_Const, далее, чтобы получить реально значение, нужно полученный результат разделить на 8, что фактически есть перенос старшего разряда в младший, считаем старший разряд результата умножения (R1) и выведем его на дисплей.

Стоит заметить, что ввиду всех сокращений, точность выводимого на экран значения не очень велика.

3. Методы повышения точности АЦП

Рассмотрим основные методы повышения точности аналогово-цифрового преобразования, их можно разделить на программные и аппаратные методы повышения точности. Рассмотрим их

1. Если скорость измерения значительно превышает скорость измерения сигнала, то можно снимать не одно значение, а группу с последующим усреднением результата. Так же для эффективного подавления шумов и помех нужно чтобы частота выборки была ниже частоты различных паразитных колебаний, например, наводок от сети, примерно в 2-3 раза.

2. Использование специального режима работы АЦП – ADC Noise Reduction, который останавливает все схемы МК за исключением асинхронного таймера и «сторожевого пса». Неудобство данного способа заключается в том, что при каждом измерении МК «засыпает» не менее чем на 20 тактов, поэтому при достаточно частых измерениях это может негативно сказаться на других функциях, выполняемых МК.

3. Использование высокоточного источника опорного напряжения, например, ADR420 или REF195 либо завести напряжение на AV_{cc} через дроссель (рисунок 2.3).

4. Фильтрация питания МК, установкой керамический конденсаторов емкостью порядка 0.1uF между Vcc и GND (рисунок 2.3), как можно ближе к выводам МК, это позволит гасить помехи, возникающие при переключении логических уровней внутри МК.

5. Так же можно организовать как можно более сплошную аналоговую землю, окружающую всю аналоговую часть и не содержащую в себе замкнутых

контуров. С цифровой землей ее следует соединить в одной точке как можно дальше от точных цепей.

6. Использование фильтров для сигналов, у которых известна частота, что позволит отфильтровывать все что не входит в требуемы диапазон.

Задания для самостоятельной работы

На основании выданных преподавателем индивидуального задания выполните необходимые исследования.

Составьте отчет по выполненным заданиям.

ТЕМА № 9. ИСПОЛЬЗОВАНИЕ EEPROM

Вопросы, рассматриваемые в данной теме:

1. Энергонезависимая память данных EEPROM
2. Сохранность данных в EEPROM
3. Запись и чтение EEPROM

Краткие теоретические сведения

1. Энергонезависимая память данных EEPROM

Как было отмечено ранее (в главе 3), практически все МК AVR имеют встроенную память EEPROM для хранения констант и данных при отключении питания. Рассмотрим основные моменты важные при программировании данного типа памяти.

Одной из особенностей EEPROM является то что, запись в память протекает значительно медленнее чем чтение из памяти. И так как процесс записи зависит от встроенного RC-генератора то длительность цикла записи одного байта в EEPROM не определена и может занимать в среднем от 2 до 4 мс. С точки зрения МК время порядка 2 мс огромно, и за него можно успеть выполнить несколько тысяч команд, поэтому к процессу записи нужно подходить внимательно и следить за его целостностью.

Так же содержимое памяти EEPROM очень чувствительно к снижению питающего напряжения, то есть при снижении питания ниже порога стабильной работы, но недостаточного для того чтобы МК «отключился» начнется выполнение случайных команд, в том числе может быть выполнена команда записи в EEPROM что с большой вероятностью испортит ее содержимое. Как с этим бороться рассмотрим далее.

Кроме записи в EEPROM во время работы МК, можно осуществить запись в память и при прошивке контроллера. Но такая возможность требуется не

часто, так как в основном в EEPROM хранят какие-либо пользовательские установки, константы вычисляемые при работе устройства и другие данные которые необходимо сохранить в памяти устройства при выключении питания.

2. Сохранность данных в EEPROM

Разберем подробнее какие меры можно принять для сохранности данных в EEPROM, существует два метода, использование встроенных средств МК и подключение дополнительного монитора питания.

Для начала рассмотрим первый способ –использование встроенного супервизора питания - схемы BOD (по умолчанию выключена), которая позволяет при снижении напряжения ниже допустимого порога сбрасывать МК и возвращать его в рабочее состояние после восстановления напряжения. Настройка режима работы BOD выполняется с помощью fuse-битов BODLEVEL2…0 в соответствии с таблицей 9.1, а его включение битом BODEN.

Таблица 9.1. Режимы работы BOD

BOD-LEVEL2	BODLEVEL1	BOD-LEVEL0	Режим работы
1	1	1	Схема BOD выключена (значение по умолчанию)
1	0	1	Включает BOD с порогом срабатывания 2.7В
1	0	0	Включает BOD с порогом срабатывания 4В

Очевидно, что чем меньше разница между напряжением питания и порогом срабатывания, тем надежнее будет работа МК, но также стоит учитывать, что при питании от батарей можно привести схему в неработоспособное состояние в момент, когда батареи еще не исчерпали свой ресурс.

Стандартное время срабатывания встроенной схемы BOD составляет порядка 4мс, а максимальное время включение может быть около 68мс, что не всегда перекрывает дребезг, возникающий при снижении питания.

Даная схема помогает защитить данные EEPROM, но в некоторых случаях ее все же недостаточно. Тогда имеет смысл установить в схему внешний монитор питания, например, MC34064 имеет порог срабатывания 4,6В и обладает малым собственным потреблением, порядка 0,5мА. При снижении напряжения его время срабатывания составляет 200нс, а время обратного включения занимает доли секунд, что предотвращает срабатывания при дребезге и позволяет добиться большей сохранности данных в памяти.

3. Запись и чтение EEPROM

Рассмотрим запись и чтение EEPROM, на примере программы позволяющей кнопкой (SB_UP) выбрать одно значение от 1 до 9, выводимое на один из дисплеев и сохранить кнопкой (SB_SAVE) его, и при нажатии на кнопку (SB_LOAD) вывести на второй дисплей, причем оба дисплея, семисегментные, с дешифратором и подключенные к одному порту МК.

Как обычно рассмотрение программы начнем инициализации констант и переменных:

```
//Определение переменных (директива .def)
 .def TMP     = R16        //Временная переменная
 .def CNT_U   = R17        //Счетчик первого дисплея
 .def CNT_L     = R18      //Счетчик второго дисплея

 .def EE_AddrH = R19//Адрес для записи (старший)
 .def EE_AddrL = R20//Адрес для записи (младший)

//Определение констант   (директива .equ)
.equ CNT_MAX    =10         //Для ограничение вывода на дисплей значений (0-9)
.equ D1_MASK    =0b00001111
.equ D2_MASK    =0b11110000
```

Листинг 9.1. Процедура записи в EEPROM

В листинге 9.1 сначала определяются три переменных: обычная временная TMP, и две переменных счетчика CNT_U и CNT_L, отвечающие за значения, чисел, которые будут выводится на дисплеи. Для задания адреса EEPROM в который будет производится запись определенны две переменные EE_AddrL и EE_AddrH, младший и старший байты адреса соответственно.

Константами определенны: число при достижение которого значение на дисплее сбрасывается в 0 (CNT_MAX) и две маски (D1_MASK, D2_MASK), использующиеся для того чтобы не портить значение в одном из дисплеев при записи в другой, ведь они подключены к одному порту МК.

Далее разберем процедуру записи:

```
//Процедура записи EEPROM
EE_WRITE:
SBIC EECR,EEWE      //Ожидаем готовность памяти к записи
RJMP EE_WRITE       //То есть установку флага EEWE

OUT EEARL, EE_AddrL  //Загружаем адрес необходимой ячейки (0)
OUT EEARH, EE_AddrH  //(старший и младший)
OUT EEDR,  CNT_U     //и сами данные

SBI EECR, EEMWE     //Устанавливаем бит предохранитель
SBI EECR, EEWE      //Записываем байт
RET
```

Листинг 9.2. Процедура записи в EEPROM

Так как запись в EEPROM процесс достаточно медленный то сначала необходимо дождаться готовности памяти к записи, то есть сброса (автоматического) флага записи EEWE регистра EECR. Далее в регистры EEARL и EEARH, записывается адрес ячейки нужной для записи, а в регистр EEDR сами данные которые необходимо сохранить. Затем в регистр EECR устанавливается специальный бит предохранитель EEMWE, после установки которого для записи нужного значения в память необходимо в течении четырех тактов необходимо установить бит EEWE регистра EECR. Если не успеть это сделать, то бит EEWE сбросится и его необходимо будет выставить снова. Это сделано для защиты от случайной записи в EEPROM случаи которой рассматривались выше.

Чтение происходит практически аналогичным образом:

```
//Процедура чтения EEPROM
EE_READ:
SBIC EECR, EEWE     //Ждем окончания предыдущей записи
RJMP EE_READ

OUT EEARL, EE_AddrL //Выставляем адрес
OUT EEARH, EE_AddrH

SBI EECR,EERE       //Выставляем бит чтения
IN CNT_L, EEDR      //Читаем бит из ячейки памяти
RET
```

Листинг 9.2. Процедура чтения EEPROM

Ожидается готовность памяти (листинг 9.2), выбирается адрес, в котором лежат нужные данные, затем выставляется бит чтения EERE регистра EECR, и данные забираются из регистра EEDR в любой из регистров общего назначения.

Кроме вышеперечисленного стоит отметить что из-за наличия бита предохранителя EEMWE процедура записи в EEPROM процесс атомарный, и очевидно не должен ничем прерываться, поэтому если в программе используются прерывания, на время записи их необходимо запретить. Так же если в программе используется сторожевой таймер (по умолчанию выключен), то в цикле ожидания готовности памяти, необходимо сбрасывать его командой WDR.

Кратко рассмотрим другие части программы, в обработчике нажатия кнопки SB_SAVE вызывается процедура записи в память EE_WRITE. Обработчик нажатия кнопки SB_LOAD рассмотрим подробнее, так как в нем во второй дисплей загружаются данные так чтобы не испортить значение в первом:

```
SB_LOAD:
SBIS PINB,1
RJMP SB_LOAD

RCALL EE_READ              //Вызовем функцию чтения
       DEC CNT_L           //Уменьшаем так как в память записывается значение на 1 больше.
       SWAP CNT_L          //Переставляем тетрады (дисплеи на одном порту)
       IN TMP, PORTD       //Сохраняем текущее значение порта (чтобы не испортить)
       ANDI TMP, D1_MASK   //Маскируем
       OR TMP, CNT_L       //Обновляем значение
OUT PORTD,TMP              //И выводим

RJMP MAIN
```

Листинг 9.3. Загрузка и вывод данных на дисплей

В листинге 9.3 за стандартным ожиданием отпускания кнопки, следует вызов процедуры чтения из памяти, в котором значение из EEPROM копируется в CNT_L, и уменьшается на единицу (по причине того, что в память записываться значение на единицу больше реального), затем командой SWAP меняются тетрады байта, так как значение лежит в младшей тетраде, а дисплей подключен к старшей. Далее текущее значение битов PORTB копируется во временный регистр TMP, на который для сохранности значения первого дисплея с помощью

операции логического «И» накладывается маска D1_MASK, и операцией логическое «ИЛИ» устанавливается новое значение для второго дисплея.

Разберем пример (таблица 9.2): допустим, что на первом дисплее (D1) – выведено число 5 (0b00001001), а на втором дисплее (D2) выведено число 4 (0b10000000) которое должно быть обновлено на число 3 (0b00110000):

Таблица 9.2. Логические преобразования при выводе на дисплеи

Операция	Старшая тетрада (D2)	Младшая тетрада (D1)	Регистр/ маска
Загрузка из памяти числа 3	0000	0011	CNT_L
Перестановка тетрад (SWAP)	0011	0000	CNT_L
Сохраним значение PORTB в TMP	1000	1001	TMP
Логическое «И» с маской D1_MASK	0000	1111	D1_MASK
Результат:	0000	1001	TMP
Логическое «ИЛИ» с CNT_L	0011	0000	CNT_L
Результат:	0011	1001	CNT_L

То есть после всех операций в регистре CNT_L оказалось обновленное значение второго дисплея, и неизменное значение первого дисплея.

Обработчик кнопки выбора числа (SB_UP), в целом аналогичен обработчику кнопки SB_LOAD, за исключением того, что в нем нет чтения из памяти, и есть проверка на максимальное значение в регистре.

Так же кроме рассмотренного метода записи EEPROM во всех современных моделях МК существует специальное прерывание, которое генерируется по окончанию очередного цикла записи. Данное прерывание удобно использовать при записи достаточно больших массивов данных, время записи которых длится порядка секунд. Данный способ организации записи заметно сложнее, и даже официальная документация для небольших объемов данных рекомендует использовать вышеприведенные методы.

Задания для самостоятельной работы

На основании выданных преподавателем индивидуального задания выполните необходимые исследования. Составьте отчет.

ТЕМА № 10. ДИНАМИЧЕСКАЯ ИНДИКАЦИЯ И ОПРОС МАТРИЦЫ КНОПОК

Вопросы, рассматриваемые в данной теме:

1. Динамическая индикация
2. Опрос матричной клавиатуры

Краткие теоретические сведения

1. Динамическая индикация

Теперь после изучения принципов работы портов ввода/вывода, а также режимов работы таймеров, можно рассмотреть такие практические задачи как вывод информации на группу семисегментных дисплеев с помощью динамической индикации, и опрос клавиатуры или матрицы кнопок.

Разберем способ подключения группы семисегментных дисплеев (с общим анодом) к МК, и реализуем на базе 4-х семисегментных дисплеев секундомер, считающий от 0 до 1000 и управляющийся кнопки START и STOP.

Смысл динамической индикации заключается в том, что выводы дисплея, отвечающие за включение сегментов, объединяются в общую шину (шину данных) и подключаются к одному порту МК, а общие аноды (катоды), далее будем называть их управляющими выводами, соответственно подключаются к другим линиям портов ввода/вывода МК, таким образом если с достаточно большой частотой переключать дисплеи и выводить на них данные создастся эффект постоянно горящего изображения. И в отличии от раздельного подключения дисплеев будет задействовано всего 12 выводов МК (8 – шина данных, 4 – управляющие выводы).

Рассмотри подключение группы из двух дисплеев (с общим анодом) к МК (рис. 10.1):

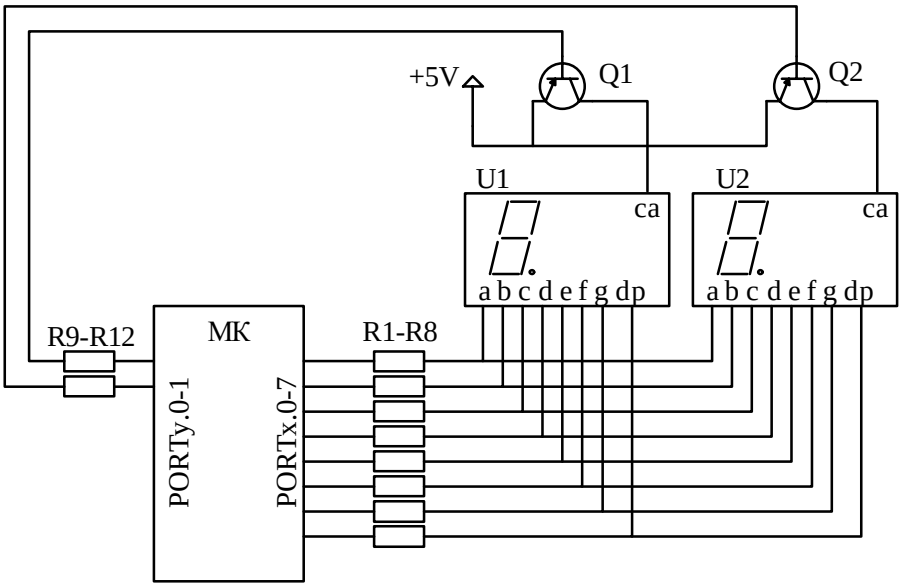


Рис. 10.1. Схема подключения группы семисегментных дисплеев

Шина данных подключается аналогично выводам семисегментного дисплея через ограничительные резисторы (R1-R8) номиналом 470 Ом, а выводы управления в зависимости от выбранных дисплеев подключаются либо напрямую (если хватает мощности МК), либо через транзисторы (Q1, Q2). Выбор типа транзистора зависит от типа дисплея, если используется дисплей с общим анодом (включение сегментов дисплея производится низким уровнем), то используется транзисторы PNP типа, в случае общего катода, включение сегментов дисплея производится высоким уровнем и соответственно используется транзисторы NPN типа. Кроме раздельного подключения нескольких дисплеев существуют сборки из двух, четырех и более дисплеев в которых шина данных объединена внутри корпуса, что упрощает проектирование печатной платы.

Кратко принцип работы программы будет заключатся в следующем: отсчет времени производится благодаря таймеру TIM1, значения для вывода на дисплей будут обрабатываться в его прерывании при совпадении СТС (TIM1M_A), а вывод динамической индикации будет осуществляться в прерывании по переполнению(TIM0OVF) таймера TIM0. Маска для вывода значений на дисплей хранится во flash-памяти и используется аналогично ранее рассмотренному случаю вывода чисел на обычный семисементный дисплей.

Перейдем к рассмотрению вышеуказанной программы. Для удобства зададим константы и переменные (листинг 10.1):

```
//Определение переменных (директива .def)
 .def TMP     = R16 //Временная переменная
 .def CNT_1N  = R17         //Число для загрузки из памяти (1 разряд)
 .def CNT_2N  = R18         //Число для загрузки из памяти (2 разряд)
 .def CNT_3N  = R19         //Число для загрузки из памяти (3 разряд)
 .def CNT_4N  = R20         //Число для загрузки из памяти (4 разряд)
 .def COUNTER = R22         //Выбор текущего выводимого разряда на индикаторе

 .def MASK          = R21         //Маска для вывода на дисплей (из FLASH)

//Определение констант   (директива .equ)
.equ IND_MSK = 0b00000001  //Маска управляющего сигнала
.equ MAX_CNT = 10          //Для определения переполнения разряда
```

Листинг 10.1. Настройка портов ввода/вывода МК

Переменные CNT_xN будут использоваться для хранения, числа которое должно быть выведено на дисплей x, COUNTER используется для определения на какой из дисплеев необходимо вывести значение, MASK для считанной из памяти маски числа (аналогично обычному семисегментному дисплею). Константа IND_MSK, для указания на линию порта управляющего выбором дисплеев, MAX_CNT, для определения переполнения.

Далее настроим порты ввода вывода МК, с учетом того что используются дисплеи с общим анодом (листинг 10.2):

```
//Подключение 7 сегментных дисплеев (общий анод, включение низким уровнем)
       LDI TMP,0xFF
       OUT DDRD,TMP         //8 бит данных
       OUT PORTD,TMP

       LDI TMP,0x0F         //4 управляющих бита (биты выбора активного дисплея)
       OUT DDRC,TMP
```

Листинг 10.2. Настройка портов ввода/вывода МК

Порт D используется для подключения шины данных дисплеев и полностью конфигурируется как выход, на его линиях выставляется высокий уровень (все разряды выключены). 4 бита порта C к которым подключены управляющие линии дисплеев, так же устанавливаются на выход.

Рассмотрим настройку таймеров и сам принцип динамической индикации (листинг 10.3):

```
//Настройка таймеров TIM0, TIM1
LDI TMP, 1<<TOIE0|1<<OCIE1A//Разрешим прерывание TIM0 переполнению
```

```
OUT TIMSK, TMP              //и TIM1 по совпадению с A

LDI TMP, 1<<CS00|1<<CS01    //Запустим таймер динамической индикации
OUT TCCR0,TMP

LDI TMP, 1<<WGM12           //Режим работы таймера TIM1, сброс при достижении значения
OUT TCCR1B, TMP
//Запись в 16-разрядный регистр TCNT
LDI TMP, HIGH(62500)
OUT    OCR1AH, TMP          //Первым пишется старший байт
LDI TMP, LOW(62500)
OUT    OCR1AL, TMP          //Вторым младший
```

Листинг 10.3. Настройка таймеров МК

Существует два очевидных варианта переключения с определенной частотой, это программные счетчики (низкая точность) и аппаратный таймер. Принцип работы переключения в случае аппаратного таймера строится на том, что при каждом его переполнении срабатывает прерывание (TIM0OVF) и в его обработчике выводится следующий сегмент дисплея, использование прерывания гарантирует обновления дисплея через равные промежутки времени. Например, если МК работает на частоте 4 МГц, а 8-разрядный таймер (TIM0) подключен через предделитель 64, то прерывание будет срабатывать с частотой примерно равной 245 Гц, а весь дисплей обновляться с частотой примерно равно 60Гц, что достаточно, чтобы человеческий глаз не замечал переключения.

Принцип отсчета времени для секундомеров был рассмотрен ранее, здесь же для отсчета одной секунды используем 16-разрядный таймер (TIM1), в режиме работы CTC (сброс при совпадении). Рассчитаем значение, при котором таймер должен сбрасываться, при частоте работы МК равной 4 МГц, и предделителе 64, частота переполнения 16-разрядного МК составит 0.9537 Гц, таким образом, чтобы считаться с частотой 1 Гц, необходимо сбрасывать таймер при достижении значения в счетчике равном 62500.

Можно обратить внимание что в листинге 10.3 таймер отсчета времени TIM1 в отличии от таймера динамической индикации TIM0 не запускается, это связано с тем что запускается он по нажатию кнопки START.

Теперь рассмотрим прерывания, начнем с прерывания по переполнению таймера TIM0 отвечающего за динамическую индикацию (TIM0OVF) (листинг 10.4):

```
TIM0OVF:
//Установим указатель на начало массива с символами
       LDI ZH,HIGH (N_mask*2)
       LDI ZL,LOW (N_mask*2)

       CPI COUNTER,0
       BREQ DISP1
...
       CPI COUNTER,3
       BREQ DISP4

DISP1:
       INC COUNTER          //Увеличим счетчик, для того чтобы в следующей итерации
                            //обновить другую часть дисплея
       OUT PORTC,MASK       //Выберем необходимый дисплей
//Вывод на дисплей символа из памяти по адресу(ZL+CNT_xN)
       ADD ZL,CNT_1N        //Прибавим к указателю на Flash нужное значение
       LPM                  //Достанем его из памяти (регистр R0)
       OUT PORTD,R0         //Выведем
RETI

DISP2:                      //Аналогично
       INC COUNTER
       LSL MASK             //Сместим маску влево
       OUT PORTC,MASK

       ADD ZL, CNT_2N
       LPM
       OUT PORTD,R0
RETI
...

DISP4:                      //Аналогично
       LSL MASK
       OUT PORTC,MASK

       ADD ZL, CNT_4N
       LPM
       OUT PORTD,R0

//Отчистим счетчик итераций и вернем маску к начальному значению
       CLR COUNTER
       LDI MASK,IND_MSK
RETI
```

Листинг 10.4. Прерывание TIM0OVF

Листинг приводится в сокращенном виде, так как фактически состоит из повторяющихся операций. Сначала устанавливается указатель на начало массива в котором лежат маски значений, далее определяется на какой дисплей необходимо вывести значение, и происходит переход к необходимой метке DISPx, где если необходимо увеличивается счетчик для вывода при следующем прерывании

на следующий дисплей (либо этот счетчик обнуляется для вывода на первый дисплей) и смещается (или сбрасывается) маска выбора управляющего вывода. Так же по каждой из меток (аналогично ранее рассмотренному выводу на семисегментный дисплей) выводится нужное число.

Прерывание TIM1 для отсчета временных интервалов практически аналогично прерыванию для ранее рассмотренного секундомера, за исключением того, что в нем обрабатываются четыре разряда, а не два разряда, поэтому рассматриваться подробно не будет.

Использовать метод динамической индикации можно как на большем, так и на меньшем числе дисплеев, а также на различных светодиодных матрицах, ограничиваясь только доступным количеством линий ввода/вывода МК. А если использовать 2 сдвиговых регистра HC595, то количество необходимых выводов для рассматриваемой задачи можно сократить до 4.

2. Опрос матричной клавиатуры

Зачастую для опроса большого количества кнопок выводов может не хватить, в таком случае можно использовать матричную клавиатуру (рис. 10.2). Такую клавиатуру можно найти как в готовом виде, так и собрать самостоятельно, подключая кнопки по схеме, представленной на рисунке 10.2б.

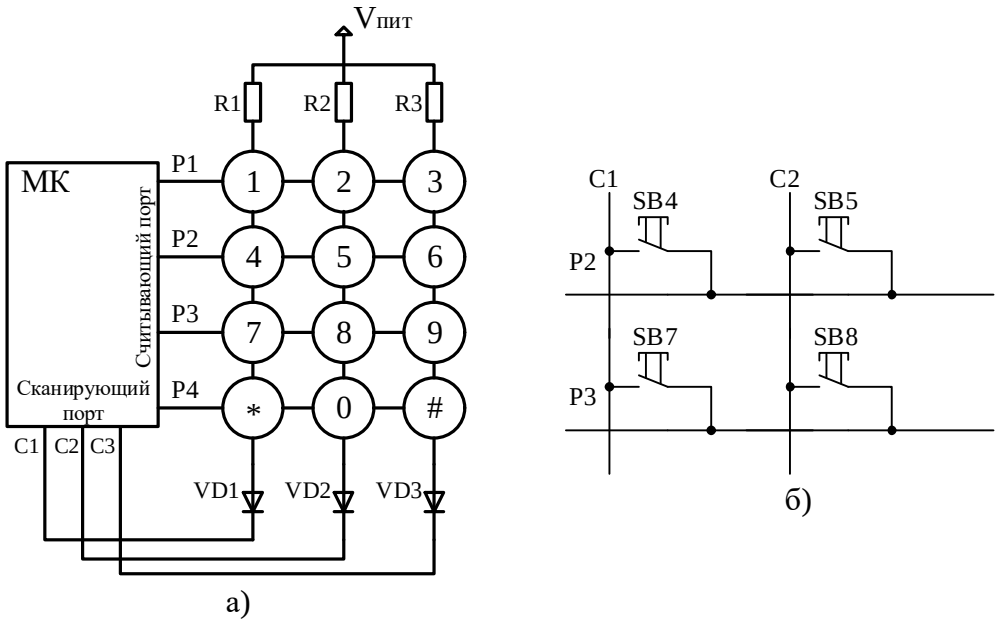


Рис. 10.2. Схема подключения матричной клавиатуры (а) и ее кнопок (б)

Принцип опроса матричной клавиатуры (рис. 10.2а) схож с принципом динамической индикации и заключается в том, что в МК выделяется два порта, сканирующий (C1-C3) и считывающий (P1-P4). Сканирующий порт в режиме выхода (через диоды VD1-VD3 служащие для защиты от короткого замыкания между линиями строк и столбцов) подключается к столбцам и подтягивается к напряжению питания. Хотя логика работы опроса клавиатуры не допускает такой возможности, на период отладки установка диодов рекомендуется. Считывающий порт работает в режиме входа с подтяжкой (Pull-Up), для этого могут использоваться как внешние резисторы (R1-R3), так и встроенные Pull-Up резисторы МК.

Затем в одну из линий сканирующего порта (например, C1) выводится низкий уровень сигнала (0), что придавливает весь столбец к земле. В это время поочередно считываются сразу все значения из считывающего порта (P1-P4), и если ни одна из кнопок не нажата, то на всех линиях считывающего порта будет высокий уровень сигнала (1), в противном случае нажатая кнопка замкнет вывод C1 на соответствующую линию считывающего порта изменив на нем сигнал с высокого уровня на низкий. Таким образом будут получены однозначные координаты нажатой кнопки. Например, если нажата кнопка 3, то на линии P1 будет низкий уровень сигнала (0).

Далее по аналогичному принципу сканируются оставшиеся порты столбцов (C1-C3). Зная принцип сканирования матрицы кнопок можно составить таблицу соответствия считываемых кодов с реальными значениями. Другой способ составление таблицы, использованный в данном случае, вывод поочередно масок в неиспользуемый порт и их последующая запись в таблицу:

Таблица 10.1. Соответствия маски и реальных значений

Значение маски (MSK)		Реальное значение
Двоичное	Шестнадцатеричное	
0b11000001	0xC1	1
0b10100001	0xA1	2

0b01100001	0x61	3
0b11000010	0xC2	4
0b10100010	0xA2	5
0b01100010	0x62	6
0b11000011	0xC3	7
0b10100011	0xA3	8
0b01100011	0x63	9
0b11000100	0xC4	*
0b10100100	0xA4	0
0b01100100	0x64	#

Рассмотрим программную реализацию вышеописанного метода опроса матричной клавиатуры на примере устройства считывающего нажатую кнопку клавиатуры и выводящую ее значение на цифровой индикатор.

Для начала (листинг 10.5) определим переменные и константы, а также во flash-памяти создадим массив соответствия маски и реальных значений для вывода на цифровой индикатор:

```
//Определение переменных (директива .def)
 .def TMP = R16             //Временная переменная
 .def CNT = R17             //Счетчик
 .def MSK = R18             //Маска и результат


 //Определение констант   (директива .equ)
 .equ KEYMSK  = 0b11011111        //Маска для снятия значения со столбцов
 .equ SCANMSK = 0b11100000        //Маска для того чтобы не "портить" оставшиеся линии порта

//Таблица соответствия
Code_table:
.db 0xC1,0x01 //1
.db 0xA1,0x02 //2
.db 0x61,0x03 //3
.db 0xC2,0x04 //4
.db 0xA2,0x05 //5
.db 0x62,0x06 //6
.db 0xC3,0x07 //7
.db 0xA3,0x08 //8
.db 0x63,0x09 //9
.db 0xC4,0x0A //*(a)
.db 0xA4,0x00 //0
.db 0x64,0x0B //#(b)
.db 0xFF,0    //Конец массива
```

Листинг 10.5. Определение констант и переменных

В данной программе используется три переменных: временная (TMP), счетчик столбцов (CNT), и переменная для записи считаной маски (MSK). Для удобства переключений столбцов используется маска KEYMASK, в которой 0 указывает на опрашиваемый столбец, и для перехода к следующему столбцу сдвигается. Маска SCANMSK используется, для того чтобы при опросе клавиатуры не изменять значения других линий ввода/вывода которые могут использоваться для других целей. Массив Code_table содержит таблицу соответствий между считаной маской и реальными значениями.

Основная часть программы состоит из двух частей, вызывающихся последовательно, сначала вызывается функция опроса клавиатуры, а затем функция преобразования и вывода считаного значения на дисплей.

Сначала опустив повторяющиеся операции рассмотрим функцию опроса клавиатуры:

```
//Функция сканирования клавиатуры
KEYB_INIT:
	LDI CNT,3			//Выставляем счетчик проходов (по количеству столбцов)
	LDI MSK, KEYMSK			//Загружаем маску опроса

KEYB_SCAN:
	IN  TMP, PORTB			//Во временный регистр записываем значения опрашиваемого
порта (чтобы не "испортить")
	ORI TMP, SCANMSK	//Выставляем в 1 все линии опроса

	AND TMP, MSK		//Логическим И выделяем опрашиваемую линию
	OUT PORTB, TMP			//Выводим в маску в порт

	NOP				//Задержка для того чтобы сигналы
	NOP				//точно установились

	SBIS PIND,0			//Нажата кнопка в 1 ряду?
	RJMP SB1
...
	SBIS PIND,3			//Нажата кнопка в 4 ряду?
	RJMP SB4

	ROL MSK				//Сдвигаем опрашиваемый бит влево

	DEC CNT				//Уменьшаем счетчик циклов

	BRNE KEYB_SCAN			//Если не равен 0, продолжаем опрос
	CLR MSK				//Иначе "чистим" значение маски
	RET				//Возвращаемся в основной цикл

//Нажата кнопка в 1 ряду
SB1:
	ANDI MSK,SCANMSK	//Формируем сканированное значение, выделяя значащие биты
	ORI MSK, 0x01		//Кнопка нажата в первом ряду
RET
```

```
...
//Нажата кнопка в 4 ряду
SB4:
	ANDI MSK,SCANMSK	//Формируем сканированное значение, выделяя значащие биты
	ORI MSK, 0x04		//Кнопка нажата в первом ряду
RET
```

Листинг 10.6. Функция опроса клавиатуры

В листинге 10.6 по метке KEI_INIT происходит инициализация опроса в которой выставляется по количеству столбцов счетчик проходов (CNT), а так же в регистр MSK загружается маска опроса KEYMASK. После инициализации начинается сам опрос (KEY_SCAN), в котором сначала во временный регистр TMP сохраняется значения из порта B, затем линии опроса выставляется в 1, логическим «И» выделяется опрашиваемая линия, и полученное значение выводится в порт. Потом после некоторой задержки необходимой для того чтобы все линии порта точно установились в нужный уровень, начинается опрос считывающего порта. Если ни одна из кнопок не нажата, маска MSK опроса сдвигается влево, а счетчик CNT уменьшается. Если еще не все столбцы опрошены, то опрос продолжается, если опрос завершен, то маска обнуляется, и функция опроса завершает свое выполнение. В случае если кнопка нажата командой проверки-пропуска (SBIS) происходит переход по метке SBx (x –номер ряда), в которой из координат столбца и строки окончательно формируется маска MSK указывающая на нажатую кнопку.

После формирования значения выполнения функции прекращается. И начинается выполнение функции перевода координат в реальное значение:

```
//Инициализация функции декодирования
FIND_SYMBOL_INIT:
CLR TMP                     //На всякий случай отчищаем временный регистр
LDI ZH,HIGH(Code_table*2)  //Загружаем указатель на массив
LDI ZL,LOW(Code_table*2)

//Непосредственно поиск символа
FIND_SYMBOL:
LPM TMP,Z+    //Считаем значение по адресу Z и инкрементируем адрес(Z+)
//Теперь указатель указывает на реальное значение
CPI TMP,0xFF//Конец массива?
BREQ MAIN           //Да, выходим из функции

CP MSK,TMP          //Значение найдено в массиве?
BREQ DISPLAY_OUT    //Выводим

LPM TMP, Z+   //Если нет, то снова увеличиваем адрес чтобы пропустить
//адрес в котором лежит реальное значение и продолжаем поиск
```

```
RJMP FIND_SYMBOL

DISPLAY_OUT:
LPM TMP,Z           //Загружаем значение из массива
OUT PORTC,TMP //Выводим его
RJMP MAIN
```

Листинг 10.7. Функция декодирования значения

Рассмотрим подробно листинг 10.7 Инициализации функции включает в себя, обнуление временной переменной TMP и установку указателя на массив соответствия значений Code_table. Далее по метке FIND_SYMBOL идет поиск соответствия полученной маски и реального значения, для этого в TMP загружается значение, лежащее по адресу Z, указатель инкрементируется и теперь указывает на реальное значение. Если конец массива не достигнут (TMP≠0xFF), то значение, считанное из массива, сравнивается с маской MSK, и в том случае если они равны, то по метке DISPLAY_OUT реальное значение загружается в TMP из массива и выводится в порт. В ином случае, значение указателя снова инкрементируется и начинается новая итерация поиска.

Данный алгоритм работы с матричной клавиатурой с небольшими изменениями также может быть использован для опроса клавиатур других размерностей.

Задания для самостоятельной работы

На основании выданных преподавателем индивидуального задания выполните необходимые исследования.

Составьте отчет по выполненным заданиям.

ТЕМА № 11. УПРАВЛЕНИЕ ДВИГАТЕЛЯМИ

Вопросы, рассматриваемые в данной теме:

1. Шаговые двигатели

1.1 Разновидности шаговых двигателей и способы управления ими

1.2 Программная реализация полушагового режима работы

2. Сервоприводы

2.1 Способы управления сервоприводами

2.2 Программная реализация управления сервоприводом

Краткие теоретические сведения

1. Шаговые двигатели

1.1 Разновидности шаговых двигателей и способы управления ими

Шаговый двигатель — это электромеханическое устройство способное преобразовывать сигнал управления в угловое перемещение ротора с последующей фиксацией его в этом положении без устройств обратной связи.

Основными плюсами использование шаговых двигателей являются обеспечение полного момента в режиме остановки в случае если обмотки запитаны, точность их позиционирования без обратной связи, высокая надежность, связанная с отсутствием щеток, возможность регулировать скорость вращения изменяя частоту импульсов. К недостаткам можно отнести, достаточно высокое потребление энергии, возможную потерю контроля положения и относительно сложную схему управления.

Кратко рассмотрим основные типы шаговых двигателей:

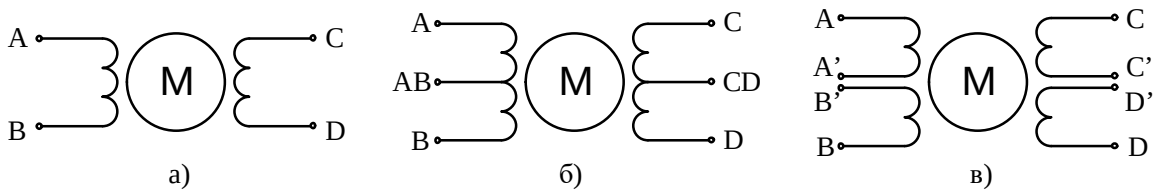


Рис. 11.1. Типы шаговых двигателей:

а) биполярный, б) униполярный в) четырехобмоточный

Биполярный двигатель (рис 11.1а), имеет одну обмотку в каждой фазе, и для изменения направления магнитного поля должен переполюсовываться драйвером. Всего биполярный двигатель имеет две обмотки и соответственно 4 вывода. Для управления таким двигателем используется мостовой драйвер, либо полумостовой с двухполярным питанием.

В отличии от биполярного двигателя, униполярный (рис 11.1б) в середине каждой обмотки имеет отвод, благодаря этому изменять направление магнитного поля можно простым переключением половинок обмоток. Это позволяет использовать для управления двигателем четыре простых ключа. Обычно такие двигатели имеют 5 или 6 выводов, в зависимости от того объединены ли средние выводы обмоток внутри двигателя.

Иногда униполярные двигатели имеют четыре раздельные обмотки (рис 11.1в), и в зависимости от соединения обмоток такой двигатель может использоваться как биполярный или униполярный. Так же стоит заметить, что биполярные двигатели более мощные и при одних и тех же размерах по сравнению с униполярными обеспечивают больший момент.

С точки зрения программирования МК управления разными типами шаговых двигателей идентично, отличие только в используемом драйвере, так для управления биполярными двигателями часто используется драйвер L293D (рис. 11.2а), а для управления униполярными драйвер LM2003 (рис. 11.2б), который фактически является сборкой мощных транзисторных ключей.

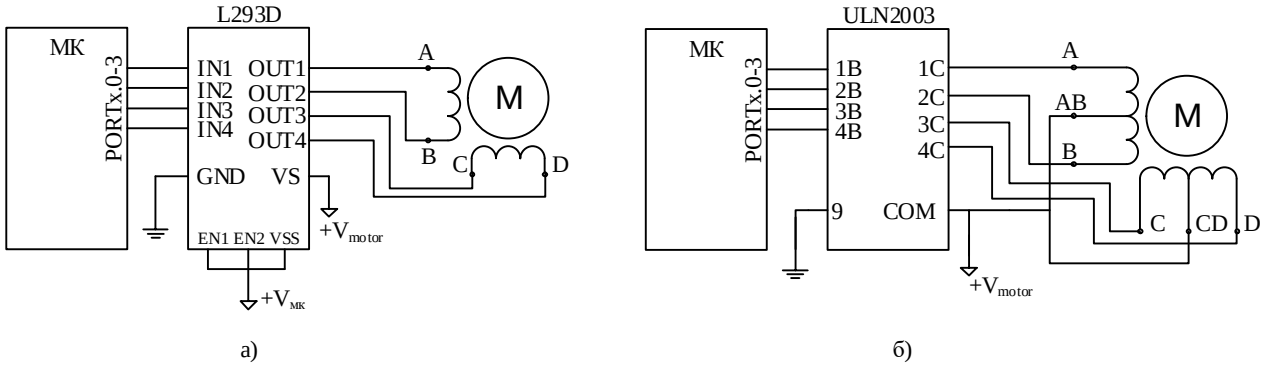


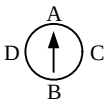
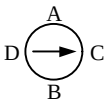
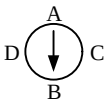
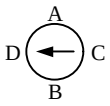
Рис. 11.2. Схема подключения шагового двигателя к МК:

а) биполярный, б) униполярный

Существуют три основных способа управления шаговыми двигателями, полношоговый, полушаговый и микрошаговый, кратко рассмотрим каждый из них:

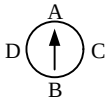
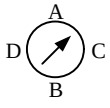
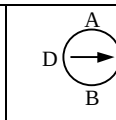
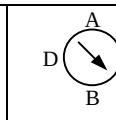
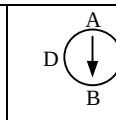
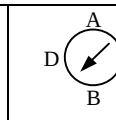
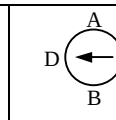
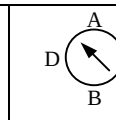
1. Полношаговый режим— это режим в котором в один момент включена только одна фаза, и точки равновесия ротора совпадают с точками равновесия у незапитанного двигателя, основной недостаток этого режима в том, что в один момент времени используется только одна фаза, а вследствие этого уменьшается момент двигателя и количество точек позиционирования. Для реализации такого типа управления на МК, необходимо с линии порта ввода/вывода МК в один момент подавать требуемый уровень сигнала на одну фазу двигателя. Все сигналы управления можно свести в таблицу 11.1, в которой стрелкой указан «север». Так же стоит учитывать, что вид таблицы может отличаться в зависимости от того как подключены обмотки двигателя к МК.

Таблица 11.1. Последовательность коммутации фаз (полношаговый режим)

				
A	1	0	0	0
B	0	0	1	0
C	0	1	0	0
D	0	0	0	1

2. Полушаговый режим – это режим отличный от полношагового тем, что каждый второй шаг запитаны две фазы, в результате угловое перемещение ротора составляет половину угла шага полношагового режима, что улучшает точность позиционирования.

Таблица 11.2. Последовательность коммутации фаз (полушаговый режим)

								
A	1	1	0	0	0	0	0	1
B	0	0	0	1	1	1	0	0
C	0	1	1	1	0	0	0	0
D	0	0	0	0	0	1	1	1

Последовательность коммутации фаз (с теми же допущениями что и в полношаговом режиме) для полушагового режима приведена в таблице 11.2.

3. Микрошаговый режим – режим в котором величина шага уменьшается еще больше, благодаря чему это режим обеспечивает еще более точное позиционирование и значительно меньшие вибрации при работе вплоть до нулевой частоты. Принцип работы данного режима в том, что две обмотки двигателя запитываются разными по величине токами в следствие чего точка равновесия ротора смещается. Таким образом ротор можно зафиксировать в любой произвольной позиции. Программная реализация этого метода в настоящей работе рассматриваться не будет.

1.2 Программная реализация полушагового режима работы

Рассмотрим программу управления шагового двигателя в полушаговом режиме, которая реализует возможность вращения шагового двигателя с определенной частотой, как в прямом, так и в обратном направлении с возможностью переключения направления вращения в ходе работы двигателя, а также позволяет остановить его кнопкой STOP.

В программе будут задействован таймер и его прерывание по переполнению для задания частоты вращения двигателя. Последовательность коммутации фаз записана во flash-память по адресу SM_mask:

```
SM_mask:
.db  0b00001110,0b00001100, 0b00001101, 0b00001001, 0b00001011, 0b00000011, 0b00000111,
0b00000110, 0b00001110,0
```

Рассмотрим переменные и константы введённые для удобства понимания исходного кода и программирования:

```
//Определение переменных (директива .def)
 .def TMP       = R16             //Временная переменная
 .def CNT       = R17             //Счетчик указатель
 .def FR_FLAG = R18               //Флаг направления вращения

//Определение констант   (директива .equ)
.equ CNT_MAX    = 8               //Для определения переполнения разряда
.equ F_FLAG_EN  = 1               //Прямой ход
.equ R_FLAG_EN  = 2               //Реверс
```

Листинг 11.1 Инициализация переменных и констант

Как видно из листинга 11.1, в программе будут использоваться три переменные, временная переменная TMP, переменная CNT в которой будет храниться указатель на конкретный элемент массива последовательности коммутации фаз и переменная FR_FLAG, которая может принимать значения F_FLAG_EN и R_FLAG_EN, если двигатель вращается в прямом и обратном направлении соответственно. Константа CNT_MAX служит для ограничения размера массива SM_mask.

Подробное рассмотрение настройки периферии, портов/ввода вывода и обработчиков кнопок опустим, отметив что в данной программе фазы двигателя коммутируются низким уровнем с линий портов ввода/вывода МК. Для задания частоты вращения используется таймер TIM0 работающий с предделителем 1024, таким образом прерывание по переполнению (TIM0OVF) будет срабатывать с частотой приблизительно равной 4 Гц (частота МК 1МГц), а частота вращения двигателя с полным шагом 90º составит примерно 0.5 об/с. Такие параметры выбраны для более наглядного представления работы двигателя в симуляторе, и для использования в практических задачах могут быть легко пересчитаны и изменены.

Теперь рассмотрим само прерывание по переполнению (TIM0OVF) в котором и происходит управление коммутацией фаз двигателя:

```
TIM0OVF:
MOTOR_CONTROL:                          //Определение режима работы
       CPI FR_FLAG,R_FLAG_EN            //Флаг реверса поднят?
       BREQ SM_REVERSE                  //Да, переходим к SM_REVERSE
//Прямое вращение двигателя
SM_FORWARD:
       ADD ZL, CNT                      //Считываем из памяти значение
       LPM
       OUT PORTD,R0              //Выводим в порт
       INC CNT                   //Увеличиваем
       CPI CNT,CNT_MAX           //Достигли максимума?
       BREQ RES_CNT              //Сбрасываем счетчик
RETI

//Реверс (аналогично прямому ходу)
SM_REVERSE:
       ADD ZL, CNT
       LPM
       OUT PORTD,R0
       DEC CNT
       CPI CNT,0
       BREQ RES_CNTR
       CPI CNT,-1                //На случай если вылезли за границу
       BREQ RES_CNTR
```

```
RETI

//Сброс счетчика прямого хода
RES_CNT:
	LDI CNT,0
RETI

//Сброс счетчика реверса
RES_CNTR:
	LDI CNT,CNT_MAX
RETI
```

Листинг. 11.2 Коммутация фаз двигателя в прерывании

В листинге 11.2 первоначально определяется ход вращения, прямой или обратный, сравнением значения, хранящегося в регистре FR_FLAG и константой-флагом указывающей на реверс (R_FR_FLAG), в зависимости от результата сравнения, происходит переход по одной из меток. В случае вращения в прямом направлении (SM_FORWARD) сначала из массива достается новая маска коммутации, затем она выводится в порт ввода/вывода, увеличивается и проверяется на адекватность, в случае если при увеличении произошел выход за пределы массива, указатель сбрасывается в 0 по метке RES_CNT. Реверсивное вращение двигателя обрабатывается аналогично, за исключением того, что массив считывается не с начала в конец, а наоборот и сброс счетчика CNT выполняется не в 0, а в его максимальное значение CNT_MAX.

2. Управление сервоприводом

2.1 Способы управления сервоприводами

Сервопривод — это в общем случае привод с управлением через отрицательную обратную связь, которая позволяет точно управлять параметрами движения. Если рассматривать термин более узко, то это электропривод с обратной связью по положению, что позволяет использовать его в задачах точного позиционирования.

Основные преимущества сервопривода перед шаговым двигателем это:

1. Из-за конструктивных особенностей сервопривод в отличии от шагового двигателя может быть значительно мощнее и быстроходнее.

2. Наличие обратной связи гарантирует более высокую точность.

3. Меньшие затраты энергии в работе.

Из недостатков основных недостатков можно отметить:

1. Более сложная составная конструкция двигателя, включающая в себя, датчик и блок управления.

2. Более сложная программа управления.

3. Более высокая стоимость.

Рассмотрим управление одним из самых распространённых типов сервопривода, привода, удерживающего заданный угол поворота.

Управляется сервопривод импульсами постоянной частоты (обычно 50Гц) и переменной ширины, то есть положение двигателя зависит от длины подаваемых на него импульсов. Для примера (рисунок 11.3) рассмотрим двигатель, работающий в диапазоне от 0^o (α_{min}) до 180^o(α_{max}), причем минимальному положению 0^o соответствуют импульсы длительностью 1 мс (H_{min}), среднему положению 90^o импульсы длительностью 1.5 мс, а максимальному (180^o) импульсы длительность 2 мс (H_{max}).

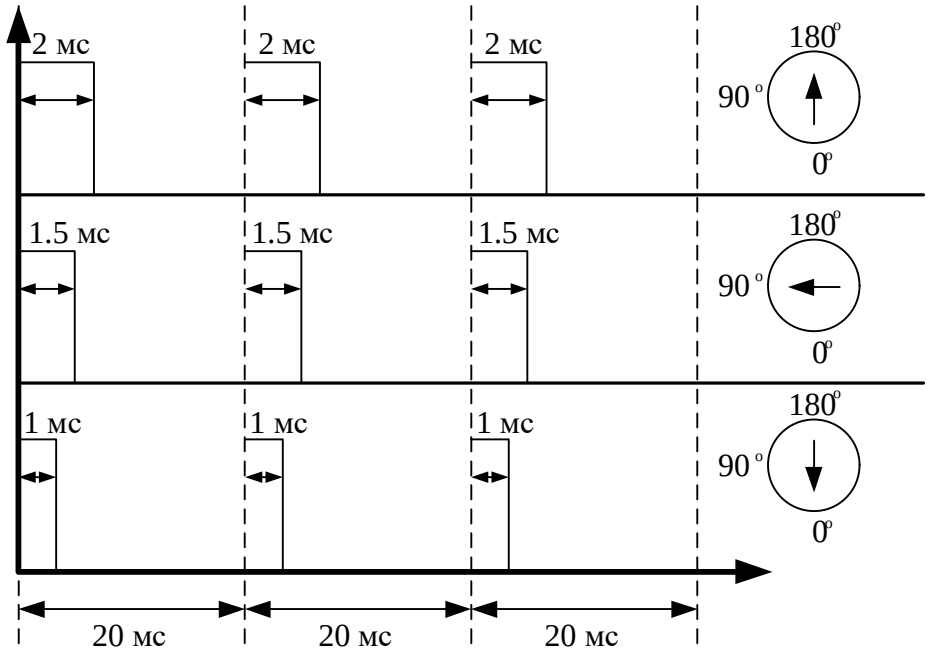


Рис 11.3. Принцип управления сервоприводом

В реальности как диапазон вращения, так и длины управляющих импульсов могут отличаться, и должны быть взяты из документации, так же стоит заметить, что даже в рамках одной модели сервоприводов может существовать некоторая погрешность и если она превышает максимально возможную, то двигатель необходимо откалибровать, например, экспериментально подобрать именно тот диапазон, который характерен для имеющегося привода.

Для управления сервоприводами используется режим PDM (pulse duration modulation) в котором при постоянной частоте импульсов меняется только их скважность.

2.2 Программная реализация управления сервоприводом

Такой режим работы можно легко организовать на базе одного из режимов ШИМ МК. Данный режим в документации имеет номер 14 и в дальнейшем будем называть его именно так. Суть работы данного режима заключается в том, что счетчик ШИМ будет сбрасываться при достижении определенного значения (ICR1) и соответственно работать с необходимой нам частотой. Длина импульсов будет задаваться в регистре OCR1. То есть мы получим требуем нам тип модуляции (рисунок 11.3). Рассчитаем эти значение с учётом того что частота работы МК 2МГц, а предделитель таймера TIM1 равен 8:

$$f_{pwm} = \frac{f_{cpu}}{N(1 + TOP)},$$

Где f_{cpu} – частота МК, f_{pwm} – требуемая частота ШИМ, N – предделитель, TOP – значение необходимое для записи в регистр ICR1.

Тогда:

$$TOP = \frac{f_{cpu} - Nf_{pwm}}{Nf_{pwm}} = \frac{2000000 - 8 \cdot 50}{8 \cdot 50} = 4999$$

Рассчитаем константу, соответствующую углу поворота в 1°:

$$n = \frac{1}{\alpha_{max}} = 0.0055$$

Тогда длина импульса $H_т$, соответствующая требуемому углу поворота $\alpha_{треб}$, будет равняться: $$H_т = n \cdot \alpha_{треб} + H_{min}$$

С учетом того что период работы таймера TIM1 составляет:

$$T_{TIM1} = \frac{N}{f_{cpu}} = \frac{8}{2000000} = 4мкс$$

Значение, которое необходимо записать в регистр OCR1 будет равно:

$$OCR1 = \frac{H_т}{T_{TIM0}}$$

Рассчитаем и внесем в таблицу 11.3 значения длин импульсов от 0° до 180° с шагом 45°:

Таблица 11.3. Рассчитанные значения OCR1A

Требуемый угол	Длина импульса	Значение в OCR1A
0	1	250
45	1.25	312
90	1.5	375
135	1.75	437
180	2	500

Разберем программу, которая при нажатии на кнопку поворачивает двигатель на угол 45°, а после достижении максимального значения угла, возвращает двигатель в начальное положение. Подробно разберем две особенности: реализацию нажатия кнопки с помощью прерывания, и настройка требуемого режима ШИМ.

Зададим в программе (листинг 11.3) две переменные TMP и CNT, первую будем использовать для хранения различных временных значений, а во второй хранить текущее положение двигателя. Значения длин импульсов для углов поворота из таблицы 11.3 запишем в явном виде, а значение константы (PWM_const) для регистра ICR1 рассчитаем в формуле:

```
//Определение переменных (директива .def)
 .def TMP     = R16        //Временная переменная
 .def CNT     = R17        //Счетчик

//Определение констант   (директива .equ)
```

```
 .equ  XTAL   = 2000000                                  //Частота контроллера
 .equ  T1_PSK = 8                                              //Предделитель таймера
//Расчет констант для требуемой частоты ШИМ и необходимой скважности импульсов
 .equ  PWM = 50                                          //Необходимая частота ШИМ
 .equ  PWM_const = (XTAL-T1_PSK*PWM)/(T1_PSK*PWM)              //Константа для частоты
//Расчет констант для определения угла вращения
 .equ   RP0   = 250        //Вычисляется как длина импульса для угла поворота (в мкс)
 .equ   RP45  = 312        //разделить на период работы таймера (в мкс)
 .equ   RP90  = 375        //период работы таймера равен 1/(XTAL/T1_PSK)
 .equ   RP135 = 437
 .equ   RP180 = 500
```

Листинг 11.3. Определение переменных и констант

Настройка (листинг 11.4) портов ввода/вывода и внешнего прерывания для кнопки заключается в настройке линии OC1A на выход (выход ШИМ), и настройки регистров MCUCR и GIMSK, отвечающих за тип внешнего прерывания (1<<ISC01) и разрешения прерывания (1<<INT0) соответственно.

```
//Настроим линию OC1A на выход
       LDI TMP, 0b000000010
       OUT DDRB,TMP
//Настройка внешнего прерывания INT0 (для обработки нажатия на кнопку)
       LDI TMP, 1<<ISC01          //Прерывание по спадающему фронту
       OUT MCUCR, TMP
       LDI TMP, 1<<INT0           //Разрешение прерывания
       OUT GIMSK,TMP
```

Листинг 11.4. Настройка линии OC1A и внешнего прерывания

Теперь рассмотрим настройку таймера TIM1 в режим ШИМ №14:

```
//Настройка таймера TIM1 для работы в режиме ШИМ (режим 14, datasheet p.98)
//Данный режим сбрасывает ШИМ при достижении определенного значения что
//позволяет добиться необходимой частоты его работы 50Гц
       LDI TMP,HIGH(PWM_const)                         //Записываем ранее вычесленное значени
       OUT ICR1H,TMP                                   //Не забывая о порядке записи
       LDI TMP,LOW(PWM_const)
       OUT ICR1L,TMP

       LDI TMP, 1<<COM1A1|1<<WGM11//Настроим ШИМ WGM11,WGM12,WGM13 - режим 14
       OUT TCCR1A, TMP            //COM1A1 - сброс линии при совпадении (non-inverting mode)

       LDI TMP, 1<<WGM12|1<<CS11|1<<WGM13              //CS11 - предделитель 8
       OUT TCCR1B, TMP
```

Листинг 11.5. Настройка TIM1 в режим ШИМ №14

В листинге 11.5 видно, что сначала в регистр ICR1 записывается ранее вычисленное значение PWM_const определяющее частоту ШИМ, а затем согласно документации, устанавливаются нужные биты регистров настройки TCCR1A и TCCR1B:

1. COM1A1 – включает сброс при совпадении, то есть сигнал на линии OC1A изменит уровень с высокого на низкий в момент когда таймер досчитает до значения равного значению в OCR1A, таким образом будет выдержана требуемая длина импульса.

2. WGM11, WGM12, WGM13 – включает режим ШИМ сбрасывающий таймер в том случае, если таймер досчитал до значения равного ICR1, что дает постоянную частоту ШИМ.

3. CS11 – включает таймер с предделителем 8.

Вся работа с двигателем проводится в прерывании, возникающем при нажатии кнопки. Рассмотрим его в сокращённом виде (опустив промежуточные положения двигателя):

```
INT0I:
INC CNT

CPI CNT,2
BREQ ROTATE90
CPI CNT,3
BREQ ROTATE135
CPI CNT,4
BREQ ROTATE180
CPI CNT,5
BREQ ROTATE0

//Поворот в 45
ROTATE45:
//Запишем необходимые значения в регистры сравнения (не забывая про порядок записи!)
	LDI TMP,HIGH(RP45)
	OUT OCR1AH,TMP

	LDI TMP,LOW(RP45)
	OUT OCR1AL,TMP
RETI
...
//Поворот в 0
ROTATE0:
//Запишем необходимые значения в регистры сравнения (не забывая про порядок записи!)
	LDI TMP,HIGH(RP0)
	OUT OCR1AH,TMP

	LDI TMP,LOW(RP0)
	OUT OCR1AL,TMP

	CLR CNT
RETI
```

Листинг 11.5. Управление углом поворота двигателя

Из листинга 11.5 видно, что выбор угла поворота производится на основании значения переменной CNT. После каждого нажатия на кнопку переменная

инкрементируется, а затем с помощью команд сравнения и условного перехода происходит переход к метке по которой устанавливается новое значение длины импульсов (OCR1A) однозначно определяющее положение двигателя.

В бесконечном цикле программы не выполняется никаких действий, просто происходит ожидания прерывания.

Задания для самостоятельной работы

На основании выданных преподавателем индивидуального задания выполните необходимые исследования.

Составьте отчет по выполненным заданиям

РАБОТА № 1

ВЫВОД ЧИСЕЛ НА ДИСПЛЕИ С ИСПОЛЬЗОВАНИЕМ КНОПОК

Цель работы:

1. Знакомство с микроконтроллерами, регистрами, командами.
2. Выполнение простейших программ.
3. Приобретение навыков в программирование и сборе электрических схем.

Краткие теоретические сведения

1. Общий принцип работы микроконтроллера и сведения о нем

В настоящее время микроконтроллер это специальная микросхема, которая рассчитана на управления разными электронными устройствами. Поэтому в его описание входят различные параметры, относящиеся как к электронным приборам, так и к вычислительным средствам. Микроконтроллеры PIC содержат *RISC*-процессор с системой команд, которая будет выполнять операции с любым регистром, используя при этом произвольный метод адресации. Пользователь имеет право сохранять полученный результат операции в самом регистре-аккумуляторе или во втором регистре, используемом для операции.

В настоящее время компания Microchip выпускает 5 основных семейств 8-разрядных RISC-микроконтроллеров: *PIC12CX, PIC16C5X, PIC16CXXX, PIC17CX XX, PIC18CXXX*. Мы рассмотрим поближе подсемейство *PIC16F8X*.

Работа МК заключается в сравнении и изменении чисел в регистрах, в сравнении и изменении отдельных битов в регистрах. Сравнение и изменение происходит в соответствии с программой. Программа – это последовательность строк с командами. Команды обращаются к тем или иным регистрам и модифицируют их или изменяют их содержимое, точнее, изменяют числа; еще точнее – изменяют биты восьмибитных чисел.

2. Память

В каждом микроконтроллере есть области памяти. Память бывает трёх типов: память программы, оперативная память, энергонезависимая память.

В память программы загружаются строчки текста программы. Загруженный текст программы не изменяется и не пропадает после выключения питания МК. Размер памяти программ определяется типом используемого МК.

Оперативная память используется для обращения к ней из текста программы. При включении МК ячейки оперативной памяти пусты (точнее, их содержимое неизвестно). Данные (числа) загружаются в оперативную память в ходе выполнения программы. Оперативная память МК на практике используется для временного хранения данных. Размер оперативной памяти относительно памяти программ гораздо меньше.

Энергонезависимая память содержит данные, которые при выключении питания МК не пропадают. Содержимое энергонезависимой памяти мы можем определить в процессе написания программы, а также содержимое может измениться в ходе выполнения программы в МК.

3. Регистры

Память состоит из так называемых ячеек или иначе регистров, соответственно числа записываются в регистры. В регистрах специального назначения содержится служебная информация, определяющая настройки работы МК. Под настройками понимается запись восьмибитных чисел в регистры специального назначения, где каждый бит в этих числах определяет ту или иную настройку. Регистры специального назначения находятся в области оперативной памяти.

Регистры специального назначения имеют жестко определенные адреса и регламентированные наименования, которые определены в документации для того или иного МК. Как правило, в разных МК адреса регистров специального назначения совпадают, что подтверждает простоту переноса программ из одних МК в другие. Регистры специального назначения в

оперативной памяти МК как правило расположены в самом начале, например, имеется 10 регистров специального назначения, занимающих диапазон адресов от 00 до 09.

Для примера, регистры специально назначения PIC16F886, а именно регистры *STATUS с адресом H'0003', PORTA* с адресом *H'0005', PORTB* с адресом *H'0006', PORTC* с адресом *H'0007*'.

Регистры общего назначения используются для записи и хранения переменных и констант. Регистры общего назначения также находятся в области оперативной памяти, и нумерация их адресов продолжает нумерацию регистров специального назначения, например, регистр *TRISA* с адресом *H'0085'*. В регистр может быть записано одно положительное число от 0 до 255. Десятичное число *255* в двоичной системе выглядит как *"11111111",* т.е. восемь единиц, а двоичное число *"0"* – как восемь нулей *"00000000".* Следует заметить, что обозначение нуля одним символом "0", в любой системе счисления равно нулю. Такие двоичные числа называют восьмибитными числами или байт.

1 БАЙТ = 8 БИТ.

Таким образом, 1 байт представляет собой последовательность нулей и единиц (или набор битов).

Примечание:

- регистры специального назначения имеют определенные имена, которые отражены в документации; не рекомендуется регистрам специального назначения присваивать другие имена (например, регистр ***STATUS*** переименовать в ***STAT***);
- регистрам общего назначения, как правило, не присваивают имена, определенные для регистров специального назначения;
- имена регистров не должны совпадать по написанию с командами;
- в тексте программы необходимо придерживаться одинакового написания имени. Например, имя ***NOM, Nom и nom*** абсолютно разные имена;

– имена набираются на английской раскладке без пробелов "разумной" длины, допускается символ нижнего подчеркивания "_" и цифры.

4. Команды ассемблера

Ассемблер – это язык программирования из 35 команд. Рассмотрим команды в следующем порядке:

- директива сопоставления имен и чисел;
- команды сложения и вычитания регистров;
- команды определения бита;
- команды взаимодействия с аккумулятором W;
- команды сложения и вычитания констант;
- команды очистки регистров F и W (обнуления);
- пустышки и метки; команды переходов;
- команды счётчики;
- переходы по событиям в счётчиках;
- переходы по результатам бит-проверки;
- команды сравнения

Такое разделение команд на группы улучшает их понимание и предназначение. Важен не только смысл команд, но связь между командами, последовательность их выполнения, изменения хода выполнения команд. Перечень команд представлен в Приложении А

5. Составление программы на ассемблере в MPLAB

Текст программы, составленный из команд на языке ассемблер, называется исходным текстом. Исходный текст для МК будет составляться в программе MPLAB. Затем в этой программе исходный текст компилируется, т.е. переводится в машинные коды. Результатом компиляции является файл HEX– простой текстовый файл с расширением *.hex, который можно открыть через стандартный блокнот

Порядок работы

Задание: Написать программу для отображения чисел на дисплее в порядке возрастания и убывания, управление организовать при помощи кнопок SBup и SBdown. Диапазон выводимых чисел ограничен, от 0 до 9. Собрать схему в программной среде «Proteus».

В повседневной жизни люди очень часто сталкиваются с вводом, какой бы то ни было информации с помощью кнопок, для дальнейшего вывода информации на индикационные устройства. В данной лабораторной работе информация будет выводиться на семисегмнентные дисплеи, которые используются во многих бытовых приборах, с которыми людям приходиться работать день ото дня.

Подключение 7сегментного индикатора

Схема подключения семисегментного индикатора приведена на рисунке 1.

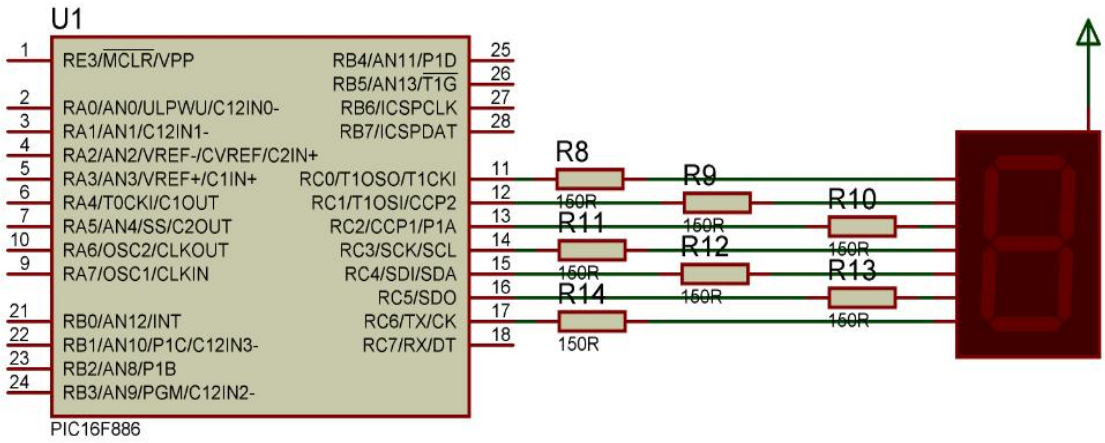


Рисунок 1 – Схема подключения семисегментного дисплея к микроконтроллеру.

Как видно из рисунка, дисплей подключается к микроконтроллеру с использованием резисторов, делается это для того чтобы ограничить ток. Так же следует отметить, что дисплеи бывают с общим анодом и с общим катодом, различие между дисплеями с общим катодом и с общим анодом не существенно и заключается в том, что дисплей с общим анодом подключается к питанию, а дисплей с общим катодом к земле.

Так же необходимо отметить то что сопротивление резисторов рассчитывается по формуле:

R=(Vs-Vd)/I (1)

где Vs - напряжение источника питания, Vd - прямое напряжение светодиода, а I - номинальный ток светодиода.

В данном случае сопротивление рассчитывается следующим образом:

R = (5 В - 2.5 В) / 0.020 А = 125 Ом.

Примем максимально приближенное стандартное значение сопротивления R=150 Ом.

Изменение определенных битов в определенных регистрах приводит к тому, что на определенных ножках (выводах) МК появляются сигналы. главная задача – получить эти сигналы в определенный момент времени и в определенной последовательности для того, чтобы МК выполнял полезные функции. Так же МК может не только выдавать сигналы на ножки, но и "реагировать" на внешние сигналы. Отметим важное, что данные в некоторых регистрах специального назначения могут изменяться не только в результате работы программы, но и в результате внешнего воздействия, например, по нажатию кнопки или в результате определенных событий в МК (например, снижения напряжения).

Таблица 1. Название выводов МК семейства PIC16F8X

Обозначение	**Тип**	**Буфер**	**Описание**
OSC1/CLKIN	**I**	ТШ/КМОП	Вход кристалла генератора, RC-цепочки или вход внешнего тактового сигнала
OSC2/CLKOUT	**O**	**-**	Выход кристалла генератора. В **RC**-режиме – выход **1/4** частоты **OSC1**
/MCLR	**I/P**	ТШ	Сигнал сброса/вход программирующего напряжения. Сброс низким уровнем.

RA0	**I/O**	ТТЛ	Сигнал сброса**/**вход программирующего напряжения**.** Сброс низким уровнем**.**
RA1	**I/O**	ТТЛ	**PORTA –** двунаправленный порт ввода**/**вывода **RA4/T0CKI** может быть выбран как тактовый вход таймера**/**счетчика **TMR0.** Выход с открытым стоком**.**
RA2	**I/O**	ТТЛ	
RA3	**I/O**	ТТЛ	
RA4	**I/O**	ТШ	
/T0CKI			
RB0/INT	**I/O**	ТТЛ**/**ТШ[2)]	**PORTB –** двунаправленный порт ввода**/**вывода**.** Может быть запрограммирован в режиме внутренних активных нагрузок на линию питания по всем выводам**.** Вывод **RB0/INT** может быть выбран как внешний входы прерывания по изменению состояния на любом из входов**.** При программировании МК **RB6** используется как тактовый**,** а**RB7** как вход**/**выход данных**.**
RB1	**I/O**	ТТЛ	
RB2	**I/O**	ТТЛ	
RB3	**I/O**	ТТЛ	
RB4	**I/O**	ТТЛ	
RB5	**I/O**	ТТЛ	
RB6	**I/O**	ТТЛ**/**ТШ[3)]	
RB7	**I/O**	ТТЛ**/**ТШ[4)]	
Vdd	**P**	**-**	Положительное напряжение питания
Vss	**P**	**-**	Общий провод **(**земля**)**

В таблице использованы следующие обозначения: I - вход; O - выход; I/O - вход/выход; P - питание; «–» - не используется; ТТЛ - ТТЛ вход; ТШ - вход триггера Шмитта.

Подключение кнопок

Схема подключения кнопок к микроконтроллеру показана на рис. 2.

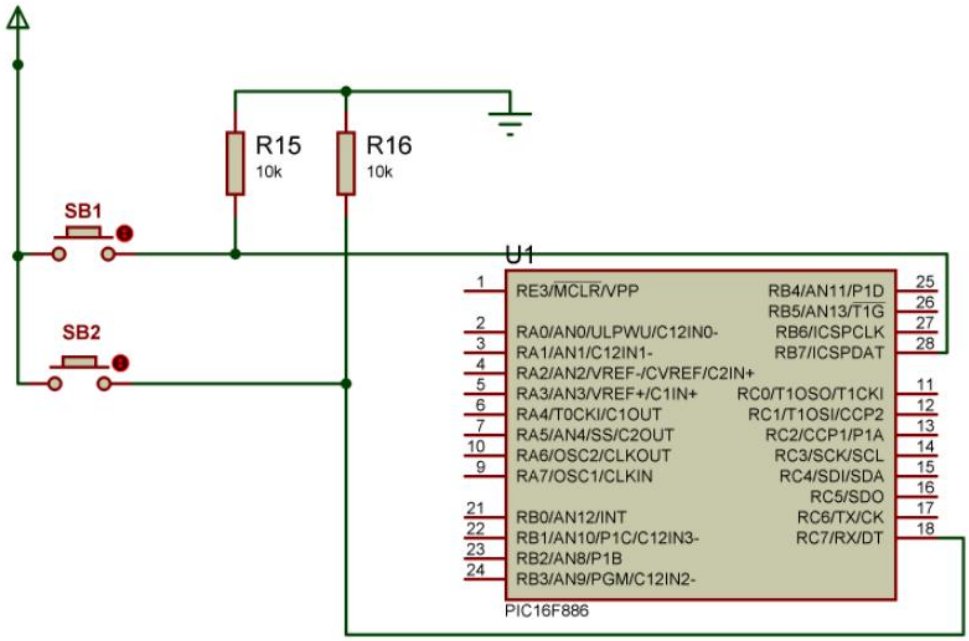


Рисунок 2 - Схема подключения кнопок к микроконтроллеру.

Для правильного считывания сигнала используем резистор, установленный одним концом в промежуток между кнопкой и контактом микроконтроллера, а другим соединенный с землей (GND). В этом случае все возможные шумы, способные дать неверный сигнал на микроконтроллер, будут уходить в землю. Сопротивление не должно быть слишком маленьким, чтобы ток, текущий через него при замкнутом контакте, не был слишком большим. Обычно используют значение порядка 10-100 кОм. Исходя из выше сказанного, сопротивление резистора выбирается 10 кОм. Когда кнопка нажата, цепь замыкается, ток из-за большого сопротивления резистора не уходит весь в землю и «нулевой» сигнал поступает на контакт микроконтроллера.

Итак, подключив к микроконтроллеру кнопки и семисегментные дисплеи, получаем схему, представленную на рисунке 3.

Так же необходимо отметить то, что принципиальная схема разрабатывалась с помощью специальных программных средств «Proteus Professional» в среде Isis.

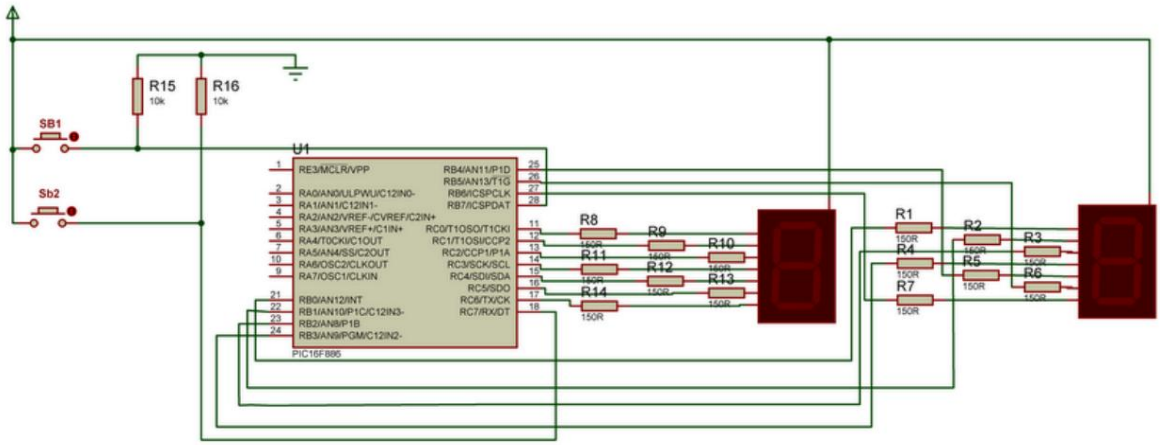


Рисунок 3 – Принципиальная схема.

Разработка полного алгоритма программы

На рисунке 4 представлен алгоритм работы программы.

Представленный алгоритм программы показывает, каким образом работает программа. Составление алгоритма помогает в дальнейшем написании программного кода и в представлении законченного рабочего устройства.

При нажатии кнопки SB1 происходит увеличение числа на DISP1 на 1, и результат выводится на DISP1. При нажатии кнопки SB2 происходит уменьшение, числа на DISP2 на 1, и результат выводится на DISP2.

Исходный текст программы

В данном разделе представлен текст программы на языке ассемблер, соответствующий частям алгоритма. Полученный текст необходимо будет преобразовать в шестнадцатеричный код с помощью программы компилятора MPLAB IDE.

Программа начинается с определения типа процессора и с определения меток замены текста.

Процесс определения типа процессора и определения меток замены текста показан на рисунке 5.

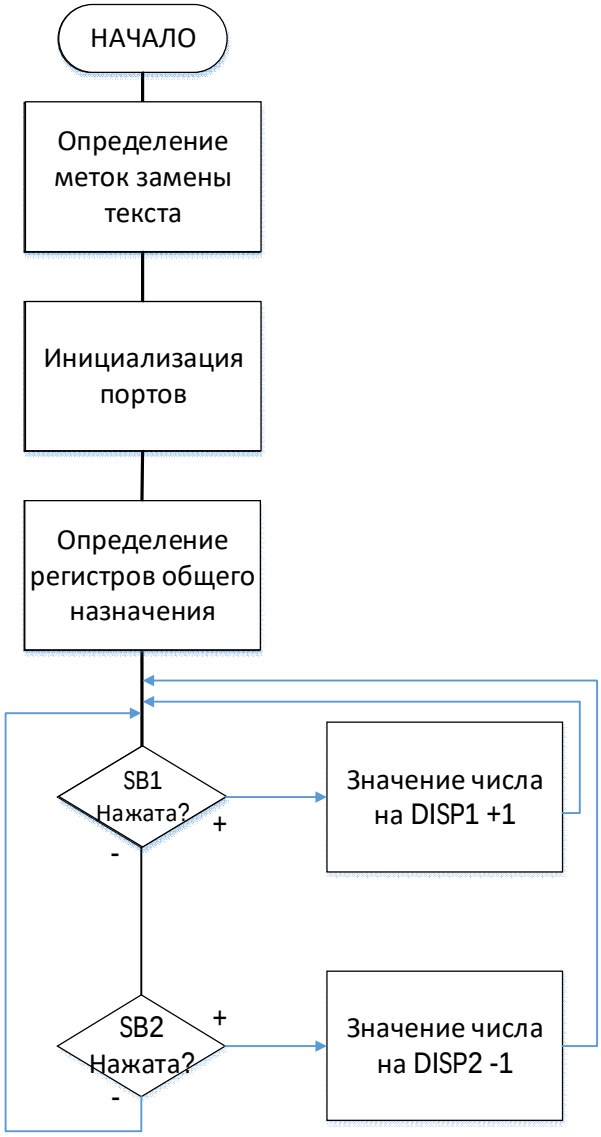


Рисунок 4 – алгоритм работы программы.

```
            LIST        P=PIC16F886 ;директива определяющая тип процессора
;======определение регистров специального назначения=======
W           EQU         0     ;СОПОСТАВЛЕНИЕ ЗНАЧЕНИЙ СЕЛЕКТОРА
F           EQU         1     ;СОПОСТАВЛЕНИЕ ЗНАЧЕНИЙ СЕЛЕКТОРА
C           EQU         0
Z           EQU         2
PC          EQU         0x02 ;СОПОСТАВЛЕНИЕ АДРЕСОВ РЕГИСТРОВ
STATUS      EQU         0x03 ;СОПОСТАВЛЕНИЕ АДРЕСОВ РЕГИСТРОВ
PORTA       EQU         0x05 ;СОПОСТАВЛЕНИЕ АДРЕСОВ РЕГИСТРОВ
PORTB       EQU         0x06 ;СОПОСТАВЛЕНИЕ АДРЕСОВ РЕГИСТРОВ
PORTC       EQU         0x07 ;СОПОСТАВЛЕНИЕ АДРЕСОВ РЕГИСТРОВ
TRISA       EQU         0x05 ;СОПОСТАВЛЕНИЕ АДРЕСОВ РЕГИСТРОВ
TRISB       EQU         0x06 ;СОПОСТАВЛЕНИЕ АДРЕСОВ РЕГИСТРОВ
TRISC       EQU         0x07 ;СОПОСТАВЛЕНИЕ АДРЕСОВ РЕГИСТРОВ
```

Рисунок 5 – Начальный этап

Далее определяются регистры общего назначения. Также назначаются адреса ячеек для хранения переменных.

Фрагмент программного кода, в котором определяются регистры общего назначения, показан на рисунке 6.

```
;=======определение регистров общего назначения=========
Reg_1       EQU         27H  ;СОПОСТАВЛЕНИЕ АДРЕСОВ РЕГИСТРОВ
Reg_2       EQU         28H  ;СОПОСТАВЛЕНИЕ АДРЕСОВ РЕГИСТРОВ
Reg_3       EQU         29H  ;СОПОСТАВЛЕНИЕ АДРЕСОВ РЕГИСТРОВ
Reg_4       EQU         30H  ;СОПОСТАВЛЕНИЕ АДРЕСОВ РЕГИСТРОВ
Reg_5       EQU         31H  ;СОПОСТАВЛЕНИЕ АДРЕСОВ РЕГИСТРОВ

            org         0    ; начало программы
```

Рисунок 6 – Определение регистров

Далее устанавливается начальный адрес программы и выполняется инициализация портов, этот процесс показан на рисунке 7.

```
bsf         STATUS,5    ; переход в Банк 1
movlw       b'10000000' ; число в аккомулятор
movwf       TRISB       ; число из аккомулятора в регистр TRISB
movlw       b'10000000' ; число в аккомулятор
movwf       TRISC       ; число из аккомулятора в регистр TRISC
bcf         STATUS,5    ; переход назад в Банк 0
```

Рисунок 7 – Инициализация портов

На рисунке 8 представлен фрагмент программы, в котором происходит ввод чисел пользователем.

В представленном фрагменте происходит опрос ножек «RC7,RB7». При нажатии кнопки «SB1» с ножки «RB7» поступает сигнал единица, затем происходит переход по метке m2. Далее производится проверка бита C на равенство нулю, делается это для того чтобы учесть был ли перенос, если переноса не было, то происходит переход по метке m3, затем происходит увеличение содержимого регистра на единицу, делается это для того чтобы сложить содержимое аккумулятора с числом 247 ранее записанным в регистр, и увидеть был ли перенос. Если число в регистре больше 255, то флаг C становится равным единице и вывод чисел на дисплей опять начинается с нуля.

```
m1
            btfsc       PORTC,7     ; бит-проверка ножки RC7
            goto        m2
            btfsc       PORTB,7     ; бит-проверка ножки RB7
            goto        m5
            goto        m1

m2
            bcf         STATUS,C    ; опускаем флаг C в ноль
            movlw       0xF7        ; (255-9)+1 = 247 -> W
            addwf       Reg_4,W     ; (Reg_7)+W
            btfss       STATUS,C    ; делаем бит-проверку C-флага
            goto        m3
            clrf        Reg_4
            goto        m4
m3
            incf        Reg_4,F     ; увеличить значение на 1 и сохранить
m4
            movf        Reg_4,W     ; посылаем значение из регистра в аккумулятор
            call        TABLE2      ; переход по метке с возвратом
            movwf       PORTB       ; посылаем значение из аккумулятора на PORTB
            call        Pause       ; переход по метке с возвратом
            goto        m1
m5
            bcf         STATUS,C    ; опускаем флаг C в ноль
            movlw       0xF7        ; (255-9)+1 = 247 -> W
            addwf       Reg_5,W     ; (Reg_8)+W
            btfss       STATUS,C    ; делаем бит-проверку C-флага
            goto        m6
            clrf        Reg_5
            goto        m7
m6
            incf        Reg_5,F     ; увеличить значение на 1 и сохранить
m7
            movf        Reg_5,W     ; посылаем значение из регистра в аккумулятор
            call        TABLE       ; переход по метке с возвратом
            movwf       PORTC       ; посылаем значение из аккумулятора на PORTC
            call        Pause       ; переход на метку с возвратом
            goto        m1
```

Рисунок 8 – Ввод чисел пользователем

Сами числа выбираются таблицы представленной на рисунке 9.

```
TABLE
            addwf       PC,F            ; Содержимое счетчика команд PC = PC + W
            retlw       b'01000000'     ; 0 возврат к команде после call TABLE
            retlw       b'01111001'     ; 1 возврат к команде после call TABLE
            retlw       b'00100100'     ; 2 возврат к команде после call TABLE
            retlw       b'00110000'     ; 3 возврат к команде после call TABLE
            retlw       b'00011001'     ; 4 возврат к команде после call TABLE
            retlw       b'00010010'     ; 5 возврат к команде после call TABLE
            retlw       b'00000010'     ; 6 возврат к команде после call TABLE
            retlw       b'01111000'     ; 7 возврат к команде после call TABLE
            retlw       b'00000000'     ; 8 возврат к команде после call TABLE
            retlw       b'00010000'     ; 9 возврат к команде после call TABLE
```

Рисунок 9 – Фрагмент программного кода

Здесь при вызове подпрограммы происходит сложение программного счетчика с содержимым аккумулятора, делается это для того чтобы осуществился выбор нужного числа.

Аналогичным образом производится ввод числа пользователем на DISP2, за исключением того, что таблица из которой выбираются числа будет иметь другой вид и производится опрос другой кнопки.

На рисунке 10 представлена таблица, из которой выбираются числа для вывода на второй дисплей.

```
TABLE2
        addwf   PC,F            ; Содержимое счетчика команд PC = PC + W
        retlw   b'01000000'     ; 0 возврат к команде после call TABLE
        retlw   b'00010000'     ; 9 возврат к команде после call TABLE
        retlw   b'00000000'     ; 8 возврат к команде после call TABLE
        retlw   b'01111000'     ; 7 возврат к команде после call TABLE
        retlw   b'00000010'     ; 6 возврат к команде после call TABLE
        retlw   b'00011001'     ; 5 возврат к команде после call TABLE
        retlw   b'00110000'     ; 4 возврат к команде после call TABLE
        retlw   b'00100100'     ; 3 возврат к команде после call TABLE
        retlw   b'01111001'     ; 2 возврат к команде после call TABLE
        retlw   b'01000000'     ; 1 возврат к команде после call TABLE
```

Рисунок 10 – Фрагмент программного кода

Задания для самостоятельного выполнения

1. Выполнить модернизацию программы так что числа выводились на одном дисплее.

2. Сделать так что бы на дисплей выводились четные или нечетные числа.

3. Реализация задачи по вариантам: 1 вариант выводить 0,1,3,4 и т.п.

4. Реализовать программу устраняющую дребезг контактов.

5. Реализовывать программу так, чтобы имело место замена дисплея, на систему с дешифратором.

6. Ввести в систему диоды, отображающие факт нажатия кнопки.

РАБОТА № 2

СХЕМЫ ПОДКЛЮЧЕНИЯ СВЕТОДИОДОВ И КНОПОК. МИГАЮЩИЙ СВЕТОДИОД

Цель работы:

1. Изучение способов подключения микроконтроллера со светодиодами и кнопками.
2. Исследование программы мигания светодиода.

Краткие теоретические сведения

1. Подключение кнопки.

Типовая схема, использующаяся для подключения кнопки или клавиши к линии порта, приведена на рис. 1.

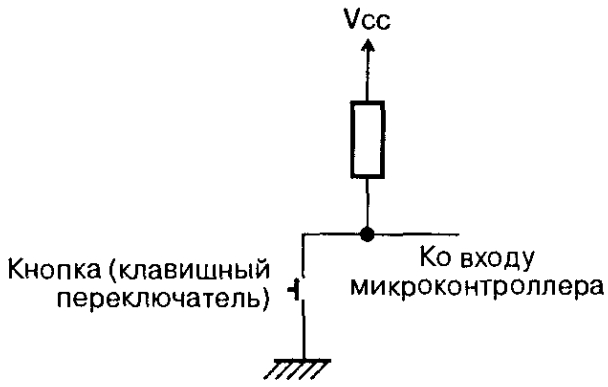


Рисунок 1 – Простая схема подключения кнопки (клавиши)

В том случае, если ваше устройство работает вблизи мощного источника помех (например, двигателя), желательно использовать резистор с небольшим сопротивлением (обычно 4,7 или 10 кОм).

Когда контакты выключателя разомкнуты, на входе будет высокий логический уровень, при замыкании контактов – низкий. [2]

2. Подключение светодиодов.

Для корректной работы светодиоды подключаются к портам микроконтроллера через резисторы (см. рис. 2).

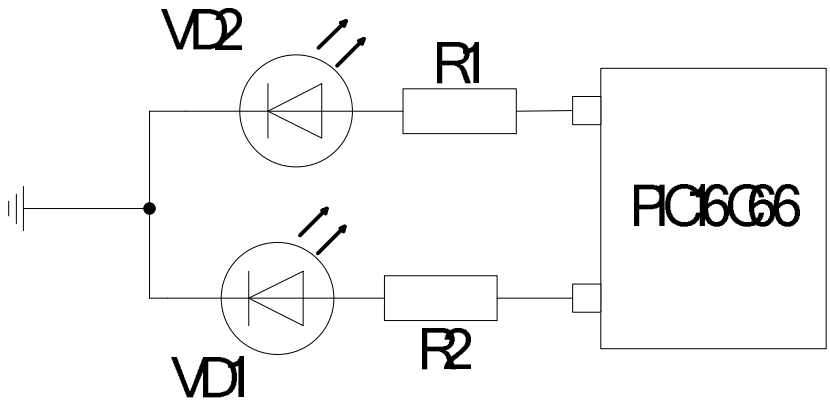


Рисунок 2 – Схема подключения светодиодов

Это необходимо, так как непосредственное подключение светодиода к источнику питание может привести к его выгоранию. Для расчета резистора используется закон Ома:

$$R = \frac{U}{I} \tag{1}$$

при этом

$$U = U_S - U_d \tag{2}$$

U_S - напряжение источника питания;

U_d - прямое напряжение светодиода.

Так же необходимо учесть коэффициент надежности. Для светодиодов он составляет 0,7 – 0,8, соответственно для расчетов обычно берется значение 0,75. Таким образом, формула для расчета сопротивления резистора примет следующий вид:

$$R = \frac{U_S - U_d}{I \cdot 0{,}75} \tag{3}$$

Порядок работы

Задание: На основе микроконтроллера смоделировать схему имитирующую работу дорожного светофора с кнопкой для пешеходов.

1. Для начала необходимо собрать схему с помощью программы-эмулятора Proteus, как продемонстрированно на рис. 3.

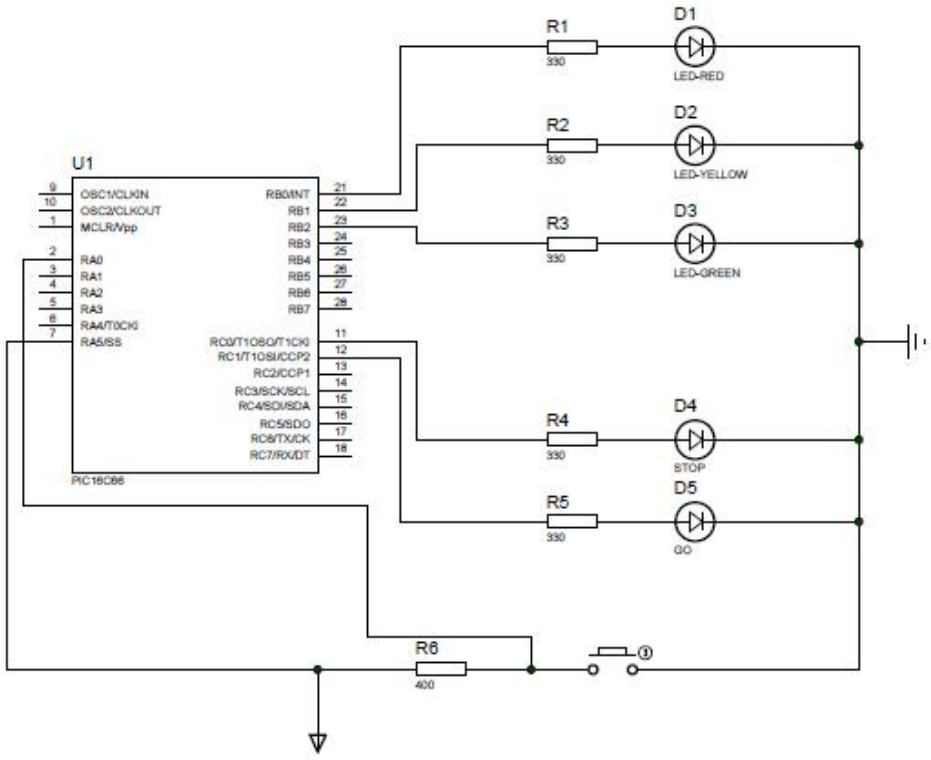


Рисунок 3 – Схема макета светофора

2. Затем необходимо разработать полный алгоритм программы, демонстрирующий ее работу. На рис. 4 и 5 представлен алгоритм работы дорожного светофора с кнопкой вызова для пешеходов.

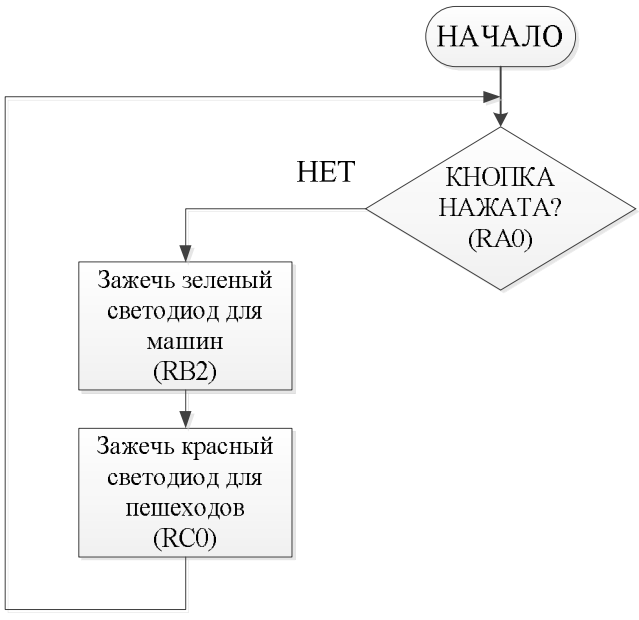


Рисунок 4 – Фрагмент алгоритма работы светофора

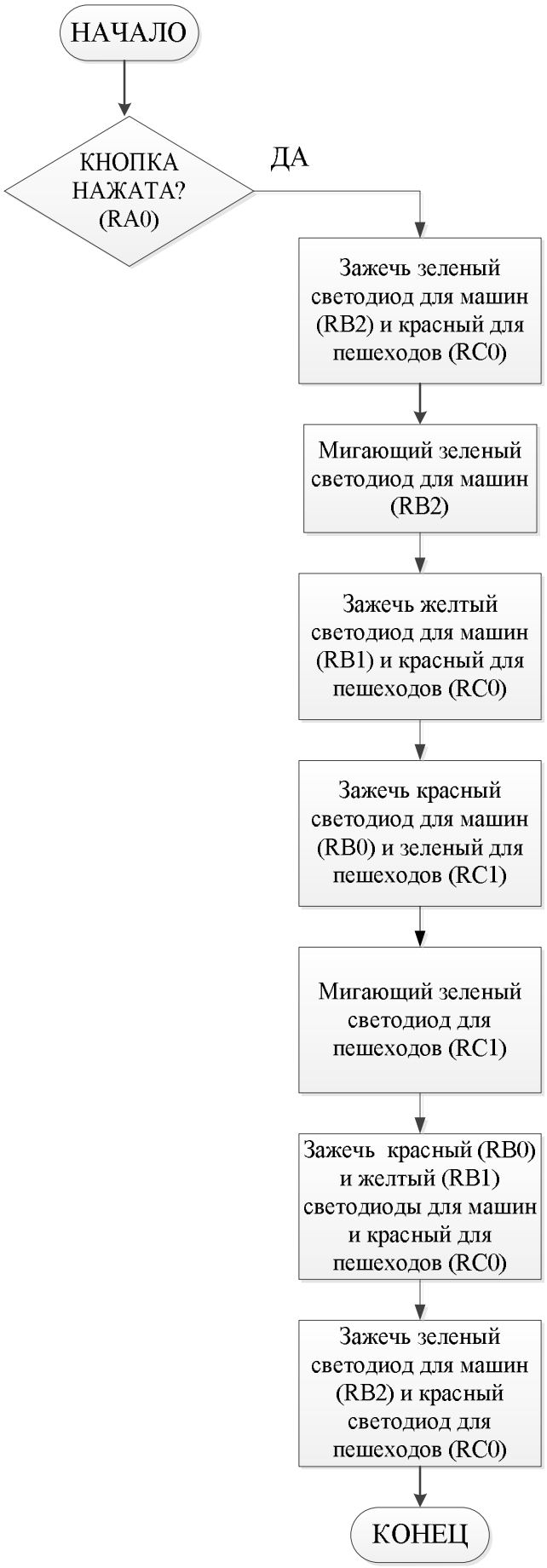


Рисунок 5 - Фрагмент алгоритма работы светофора

3. Затем необходимо написать код программы. Рассмотрим основные части программы, представленные на рис. 6 и 7.

```
START
     CLRWDT
     BTFSC SB_CALL     ; ЕСЛИ КНОПКА НЕ НАЖАТА
     GOTO STEP         ; ПЕРЕЙТИ К МЕТКЕ STEP
     GOTO MOVE         ; ПЕРЕЙТИ К МЕТКЕ MOVE
     GOTO START        ; ПЕРЕЙТИ К МЕТКЕ STOP

STEP
    BSF GREEN          ; ЗАЖЕЧЬ ЗЕЛЕНЫЙ СВЕТОДИОД ДЛЯ МАШИН
    BSF STOP           ; ЗАЖЕЧЬ КРАСНЫЙ СВЕТОДИОД ДЛЯ ПЕШЕХОДОВ
    GOTO START         ; ПЕРЕЙТИ К МЕТКЕ START

MOVE
    MOVLW b'00000100' ; ЗАЖЕЧЬ ЗЕЛЕННЫЙ
    MOVWF PORTB       ; СВЕТОДИОД ДЛЯ МАШИН
    MOVLW b'00000001' ; ЗАЖЕЧЬ КРАСНЫЙ
    MOVWF PORTC       ; СВЕТОДИОД ДЛЯ ПЕШЕХОДОВ
    CALL PAUSE        ; ВЫЗОВ ПОДПРОГРАММЫ PAUSE
    CALL PAUSE        ; ВЫЗОВ ПОДПРОГРАММЫ PAUSE
    CALL PAUSE        ; ВЫЗОВ ПОДПРОГРАММЫ PAUSE
```

Рисунок 6 – Фрагмент кода программы

В начале программы осуществляется опрос кнопки. В случае, если она не нажата, то зажигаются зеленый светодиод у светофора для машин и красный светодиод светофора для пешеходов. Если кнопка нажата, то выполняется подпрограмма для светофора для машин.

```
PAUSE
     MOVLW .9          ; ВЫГРУЗИТЬ В АККУМУЛЯТОР ЧИСЛО 9
     MOVWF REG_3       ; СОХРАНИТЬ В REG_3

P4
  MOVLW .218           ; ВЫГРУЗИТЬ В АКУУМУЛЯТОР ЧИСЛО 218
  MOVWF REG_2          ; СОХРАНИТЬ В REG_2

P3
  MOVLW .255           ; ВЫНРУЗИТЬ В АККУМЛЯТОР ЧИСЛО 255
  MOVWF REG_1          ; СОХРАНИТЬ В REG_1

P2
  DECFSZ REG_1,1       ; УМЕНЬШИТЬ REG_1 НА ЕДИНИЦУ, ЕСЛИ REG_1=0 ПРОПУСТИТЬ СЛЕДУЮЩУЮ КОМАНДУ
  GOTO P2              ; ПЕРЕЙТИ К МЕТКЕ P2
  DECFSZ REG_2,1       ; УМЕНЬШИТЬ REG_2 НА ЕДИНИЦУ, ЕСЛИ REG_2=0 ПРОПУСТИТЬ СЛЕДУЮЩУЮ КОМАНДУ
  GOTO P3              ; ПЕРЕЙТИ К МЕТКЕ P3
  DECFSZ REG_3,1       ; УМЕНЬШИТЬ REG_3 НА ЕДИНИЦУ, ЕСЛИ REG_3=0 ПРОПУСТИТЬ СЛЕДУЮЩУЮ КОМАНДУ
  GOTO P4              ; ПЕРЕЙТИ К МЕТКЕ P3
  RETURN               ; ВЫХОД ИЗ ПОДПРОГРАММЫ

 END
```

Рисунок 7 – Фрагмент кода программы

Для паузы используются три регистра, в которые при входе в подпрограмму PAUSE записываются числа. После обнуления регистра REG_1, декрементируется регистр REG_2, если он неравен нулю, снова записываем число в регистр REG_1, и заново начинаем круговой цикл декремента REG_1. После того как обнулится регистр REG_1 декрементируется регистр REG_3. Таким образом, внутренняя конструкция декремента REG_1 и REG_2 повториться еще 9 раз.

4. Далее необходимо выполнить компиляцию кода программы и загрузить его в микроконтроллер. После чего проверить правильность работы светофора (см. рис. 8).

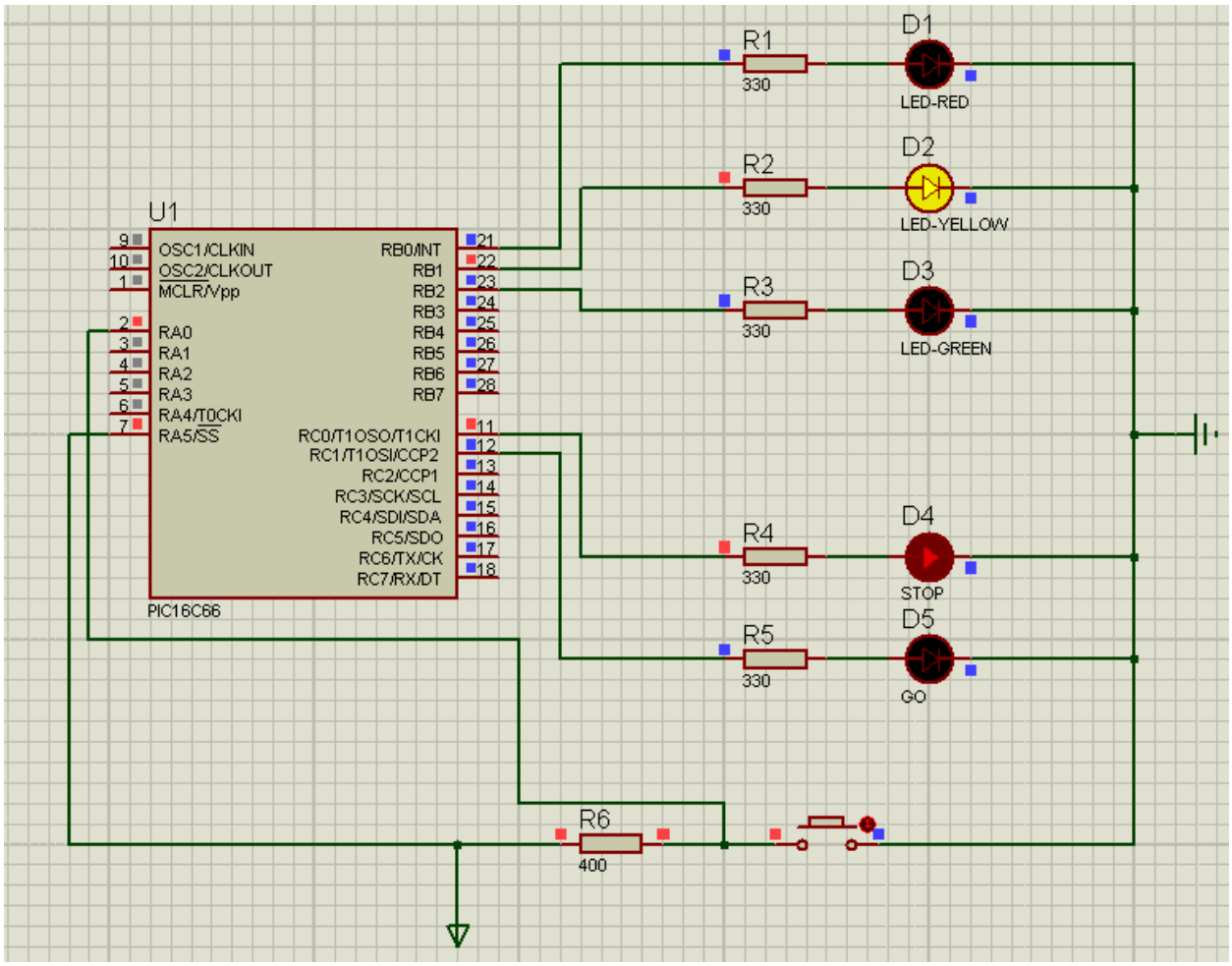


Рисунок 8 – Работа светофора

Задания для самостоятельного выполнения

1. Изменить подключения кнопок и дисплеев (полярность поменять).
2. Добавить в схему звуковое сопровождение.

3. Добавить в схему запрет на повторное нажатие кнопки при смене режима.

4. Использовать для работы аппаратный таймер.

5. Реализовать гирлянду.

6. Реализовать дисплей.

РАБОТА № 3

РАЗРАБОТКА ПРОГРАММЫ РАБОТЫ С МАТРИЧНОЙ КЛАВИАТУРОЙ

Цель работы:

1. Изучение способа подключения матричной клавиатуры.
2. Исследование программы, реализующую работу матричной клавиатуры.

Краткие теоретические сведения

Для тех случаев, когда микроконтроллер используется в панелях управления, или еще где-либо, часто возникает необходимость в оперативном вводе либо коррекции некоторой информации. Именно для этих случаев использование клавиатуры бывает очень удобно.

Вариантов клавиатур существует огромное количество. Если клавиатура состоит из нескольких клавиш, то они могут быть подключены к микроконтроллеру как отдельные кнопки, то есть каждая через свой порт. Но если кнопок много, то сам собой напрашивается вопрос о применении некоторого кодировщика, для того, чтобы сократить количество цепей подключения к микроконтроллеру. Для этого можно воспользоваться различными микросхемами-шифраторами, а можно и использовать так называемую матричную клавиатуру.

Этот прием в настоящее время является самым распространенным. Что же собой представляет матричная клавиатура? Это клавиатура, у которой клавиши находятся на пересечении строк и столбцов матрицы.

На рисунке 1 показана схема построения такой клавиатуры.

При нажатии на одну из кнопок произойдет замыкание (соединение) определенного столбца с определенной строкой.

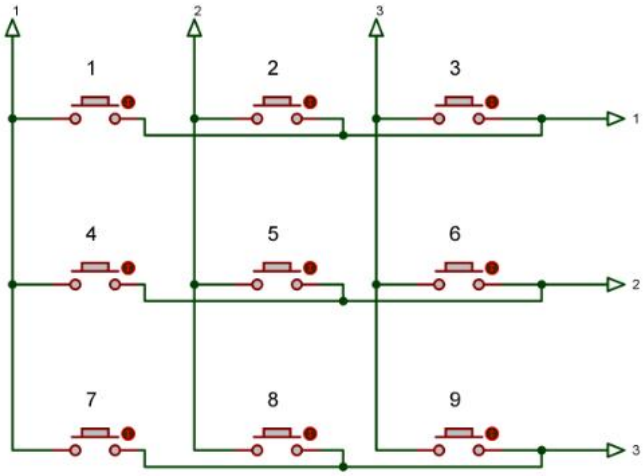


Рисунок 1 – Схема построения матричной клавиатуры

Если строки и столбцы такой клавиатуры подключить к портам микроконтроллера и опрашивать эти порты в определенном порядке, то всегда можно определить, когда и какая кнопка была нажата. Например, если по порядку подавать на каждую цепь строки высокий логический уровень и при этом, опрашивая поочередно цепи столбцов, выяснить, где этот высокий уровень обнаружен, можно определить, какая кнопка была нажата.

1. Подключение матричной клавиатуры

Теперь давайте рассмотрим способ подключения 9–и кнопочной клавиатуры. На рисунке 2 показана схема подключения к микроконтроллеру *PIC16F876A* 9 кнопочной клавиатуры и *7-сегментного дисплея* для вывода информации о нажатой клавише. В PIC-микроконтроллерах для управления матричной клавиатурой целесообразно использовать выводы порта *PORTB.* Входы этого порта снабжены внутренними резисторами, подключаемыми к напряжению питания, а выходы портов в схемном отношении аналогичны транзисторным ключам с открытым стоком.

2. Разработка программы

На рисунке 3 представлен алгоритм программы. Для того, чтобы идентифицировать цифру, нужно поочередно опрашивать столбцы клавиатуры “бегущим нулем”.

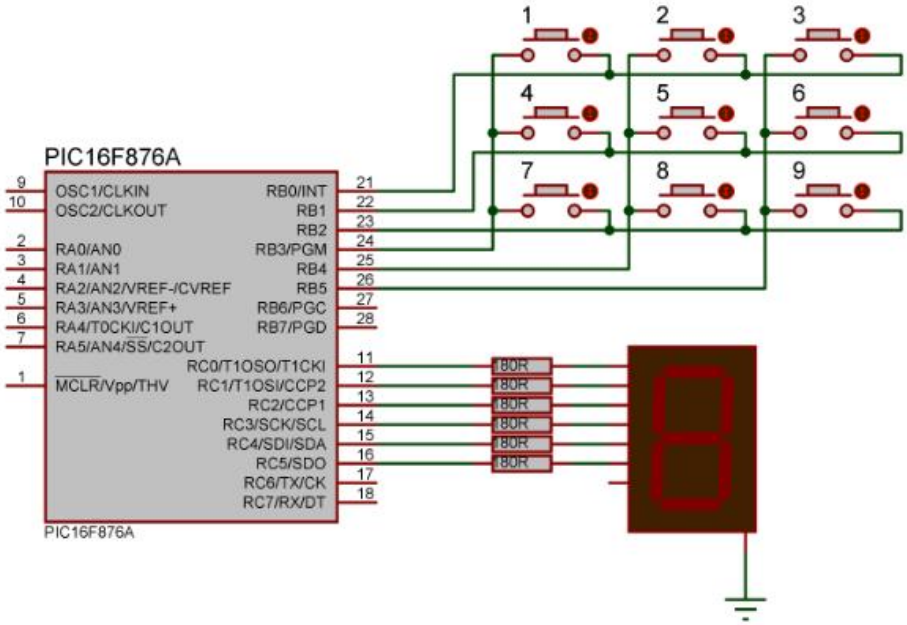


Рисунок 2 – Схема подключения 9-кнопочной клавиатуры

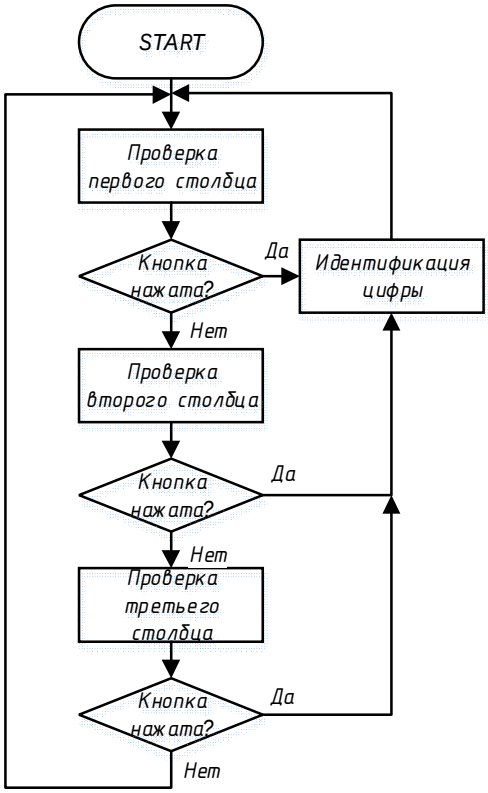


Рисунок 3 – Алгоритм программы

Для начала составим “шапку” для нашей программы (Рисунок 4). Здесь указывается – микроконтроллер и конфигурации, а также назначаем регистры. Регистр *OPTION_REG* необходим для включения “подтягивающих

резисторов” на пинах порта B. Регистр *CON_WORD (CONFIGURATION WORD)* предназначен для пина RB3. Сброс 7 бита в этом регистре, настраивает RB3 как обычный регистр вход/выход, иначе этот регистр работает как PGM т.е. переводится в режим программирования. При эмуляции в программе *Proteus* иногда пин RB3 срабатывает в режиме PGM, что мешает нормальной работе МК. Команды *NOP* выполняют функцию небольшой задержки.

```
;*****************************************************
; ПРОГРАММА "МАТРИЧНАЯ КЛАВИАТУРА" К ЛАБ.РАБОТЕ №1
; ШАПКА ПРОГРАММЫ
;*****************************************************
        LIST        P=PIC16F876A
        __CONFIG    0x3F79
W           EQU         0
F           EQU         1
C           EQU         0
Z           EQU         2
PC          EQU         0x02
OPTION_REG  EQU         0x81
CON_WORD    EQU         0x2007
STATUS      EQU         0x03
PORTA       EQU         0x05
PORTB       EQU         0x06
PORTC       EQU         0x07
TRISA       EQU         0x05
TRISB       EQU         0x06
TRISC       EQU         0x07
Reg_1       EQU         0x0C
Reg_2       EQU         0x0D
            ORG         0           ; НАЧАЛО ПРОГРАММЫ
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
```

Рисунок 4 – Шапка программы

После того как регистры подготовлены, необходимо инициализировать порты для ввода и вывода, включить подтягивающие регистры и сбросить бит режима программирования МК (Рисунок 5).

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; ИНИЦИАЛИЗАЦИЯ ПОРТОВ. 0-ВЫВОД, 1-ВВОД
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        BSF         STATUS,5    ; ПЕРЕХОД В БАНК 1
        MOVLW       b'00000111' ; ПИНЫ 012-ВВОД
        MOVWF       TRISB       ; ПИНЫ 345-ВЫВОД
        CLRF        TRISC       ; ПОРТ С НА ВЫВОД
        BCF         OPTION_REG,7; ВКЛ. ПОДТЯГИВАЮЩИХ
                                ; РЕЗИСТОРОВ
        BCF         CON_WORD,7
        BCF         STATUS,5    ; ПЕРЕХОД В БАНК 0
        CLRF        PORTB       ; ОЧИСТКА ПОРТА В
        CLRF        PORTC       ; ОЧИСТКА ПОРТА С
```

Рисунок 5 – Подготовительные моменты

Фрагмент основной программы, для кнопок 1, 2 и 7 представлен на рисунке 6. Следуем алгоритму представленном на рисунке 3. Записываем в аккумулятор *00110000*, подчеркнутый 0 означает, что будет опрошена первая столбца клавиатуры. На этом этапе опрашиваем ножку RB0, если на 0, то переход на метку PR1. Далее в аккумулятор нужно загрузить *00101000*, что означает опрос 2 столбца.

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; ОСНОВНАЯ ПРОГРАММА.
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
START

        MOVLW       b'00110000' ; ЗАГРУЗКА В АККУМ
        MOVWF       PORTB       ; ПЕРЕСЫЛАЕМ В ПОРТ В
        CALL        PAUSE       ; ВЫЗЫВАЕМ ПОДПРОГРАММУ
        BTFSS       PORTB,0 ; ОПРАШИВАЕМ НОЖКУ RB0
        GOTO        PR1     ; ЕСЛИ 0 ПЕРЕХОД НА "PR1"
        BTFSS       PORTB,1 ; ОПРАШИВАЕМ НОЖКУ RB1
        GOTO        PR4     ; ЕСЛИ 0 ПЕРЕХОД НА "PR4"
        BTFSS       PORTB,2 ; ОПРАШИВАЕМ НОЖКУ RB2
        GOTO        PR7     ; ЕСЛИ 0 ПЕРЕХОД НА "PR7"
```

Рисунок 6 – Основная программа

Если кнопка была нажата, то начинается процесс идентификации, фрагмент которой представлен на рисунке 7. Метка PR1 соответствует идентификации числа 1. Метка PR2 – число 2 и т.д. В зависимости от нажатой кнопки осуществляется переход на нужную метку, после чего в аккумулятор загружается число. Число выводится на дисплей, подключенный к порту С.

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; ИНДЕТИФИКАЦИЯ ЦИФР
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
PR1
        MOVLW       b'00000110' ; ЗАГРУЗКА В АККУМ
        MOVWF       PORTC       ; ПЕРЕСЫЛКА В ПОРТ С
        GOTO        START       ; ПЕРЕХОД НА "START"
PR2
        MOVLW       b'01011011'
        MOVWF       PORTC
        GOTO        START
PR3
        MOVLW       b'01001111'
        MOVWF       PORTC
        GOTO        START
```

Рисунок 7 – Идентификация числа

В программе используется программная задержка, код которой представлен на рисунке 8.

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
; ЗАДЕРЖКА
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
PAUSE       MOVLW       .248
            MOVWF       Reg_1
            MOVLW       .26
            MOVWF       Reg_2
METKA1      DECFSZ      Reg_1, F
            GOTO        METKA1
            DECFSZ      Reg_2, F
            GOTO        METKA1
            NOP
            RETURN
            END
```

Рисунок 8 – Программная задержка

Задания для самостоятельного выполнения

Необходимо составить программу на основе МК серии PIC16, выполняющую функции вывода числа на дисплей с использованием 9-кнопочной матричной клавиатуры. Смоделировать устройство в программе Proteus.

РАБОТА № 4

ДИНАМИЧЕСКАЯ ИНДИКАЦИЯ

Цель работы:

1. Изучение динамической индикации

2. Создание программ для вывода информации на семисегментные светодиоды и светодиодные матрицы

Краткие теоретические сведения

Динамическая индикация – это метод отображения целостной картины через быстрое последовательное отображение отдельных элементов этой картины. Причем, «целостность» восприятия получается благодаря инерционности человеческого зрения.

На этом принципе построен вывод изображения на экраны мониторов, дисплеев и прочих знаковыводящих и графических элементов с большим числом светоизлучающих элементов.

Рассмотрим динамическую индикацию на примере отображения трех числе на трех семисегментных светодиодах (рисунок 1).

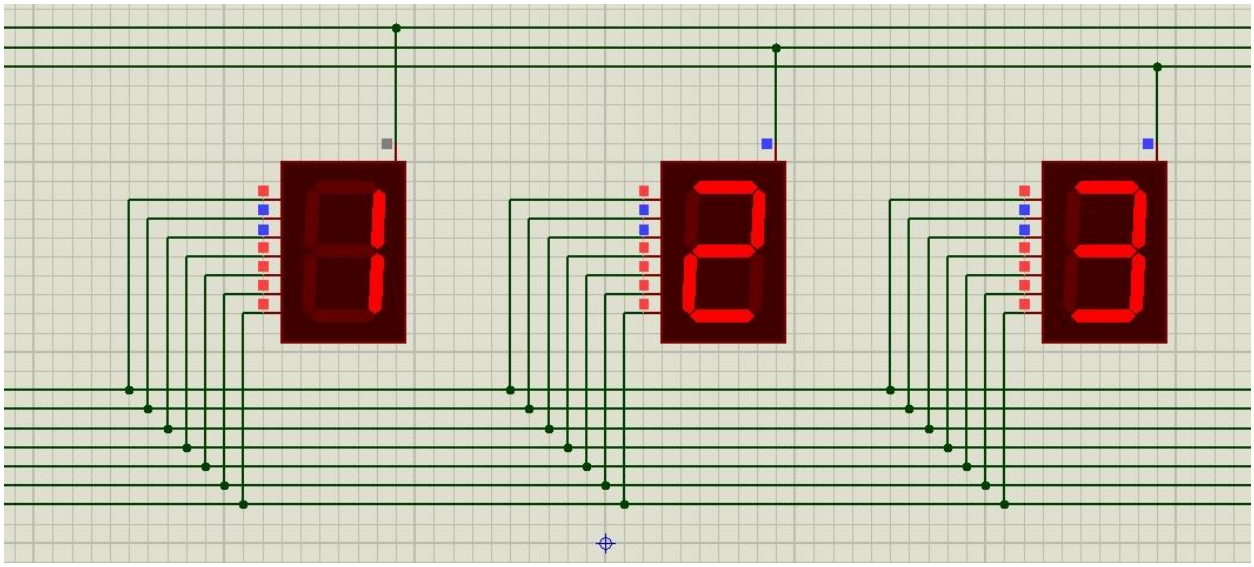


Рисунок 1 – Вывод трех чисел на трех семисегментных светодиодах

Для того что бы решить эту задачу потребуется всего два порта микроконтроллера, один порт отвечает за выбор светоизлучающего элемента, а другой, информационный, за данные, которая будет отображаться на этих элементах. Число каналов в линии выбора соответствует число используемых светоизлучающих элементов. Число подключенных элементов в данном случае, при использовании всего двух портов, будет равно семи. Но также для экономии портов и линий для выбора светоизлучающих элементов допустимо использовать дешифраторы (рисунок 2). Как видно из рисунка, используя всего два вывода одного порта, к контроллеру можно подключить четыре светоизлучающих устройства. В этом случае максимальное число семисегментных светодиодных дисплея, подключенных к двум портам будет равно $2^8 = 256$.

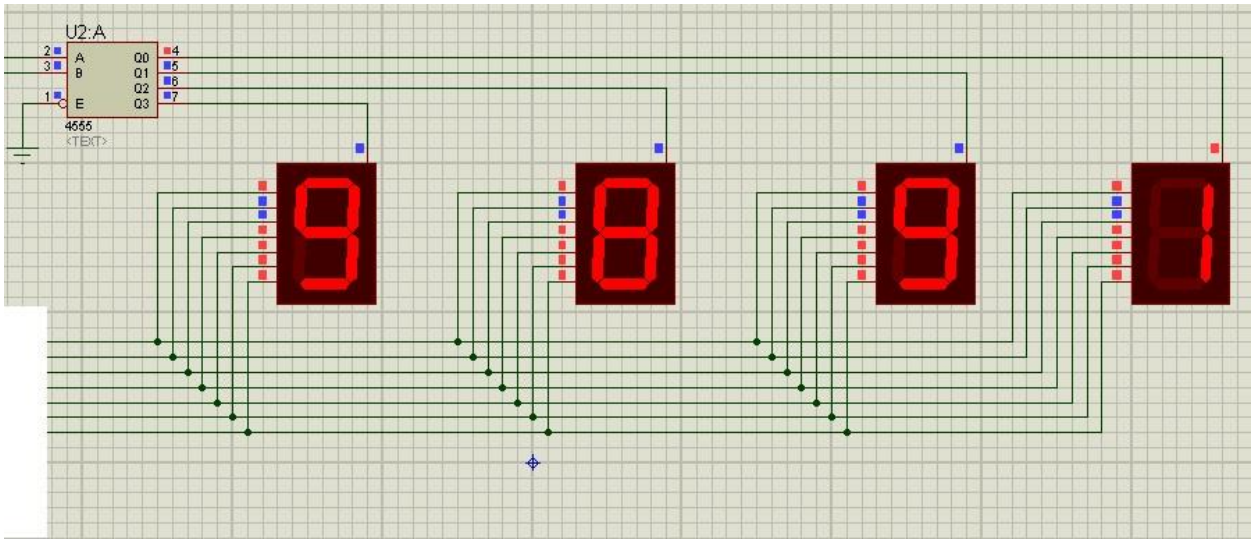


Рисунок 2 – Подключение семисегментных светодиодных дисплея к контроллеру через дешифратор

Итак, преимущества использования динамической индикации заключается в экономии портов ввода/вывода контроллера, что позволяет использовать микроконтроллеры с меньшим размером, что экономит место и уменьшает габариты используемого оборудования, и, следовательно, менее дорогим, что позволяет сэкономить бюджет.

Принцип динамической индикации заключается в том, что

1) На линию выбора мы выводим информацию о выбираемом контроллере (рисунок 3) (логический ноль, если элементы с общим катодом, или логическую единицу, если элементы с общим анодом), Один из выводов, соответствующий выбранному элементу, включен (или отключен) относительно остальных. В случае использования дешифратора необходимо на порт вывести адрес выбираемого элемента.

ВНИМАНИЕ!!! Во время этой операции биты информационной линии должны быть установлены в ноль, если элементы с общим катодом, или в единицу, если элементы с общим анодом. Это позволит избежать отображения «ложной» информации в процессе выбора необходимого элемента.

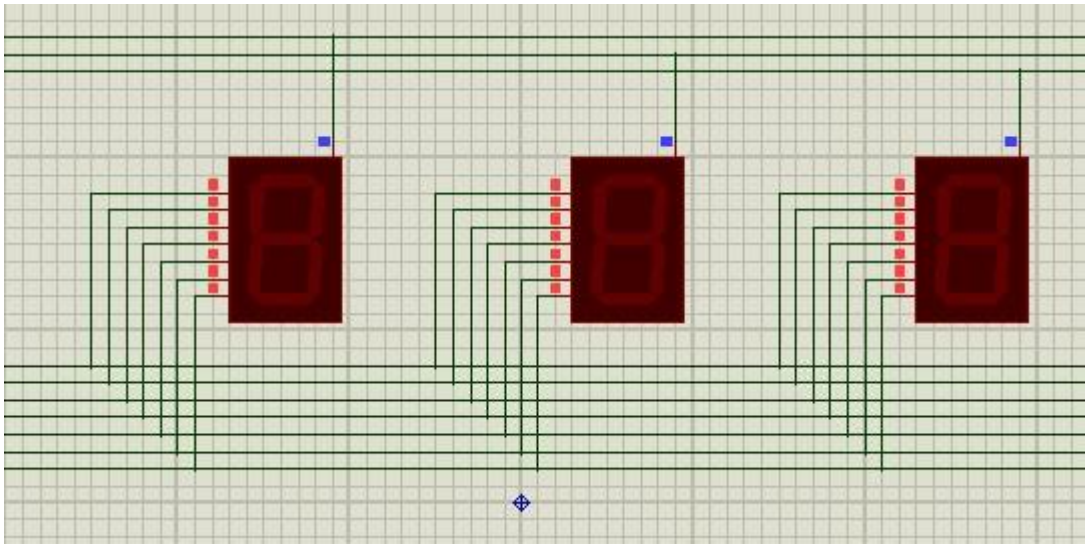


Рисунок 3 – Выбор элемента

2) После выбора элемента, на информационную линию подается необходимые данные, которые пользователь или программа решила вывести на выбранный элемент (рисунок 4).

3) Созданное состояния для выбранного элемента должно продлиться некоторое время, так как полное влечение светодиода происходит за несколько десятков микросекунд, а микроконтроллер с частотой работы 1 МГц держит состояние портов по времени неизменными 1 мкс.

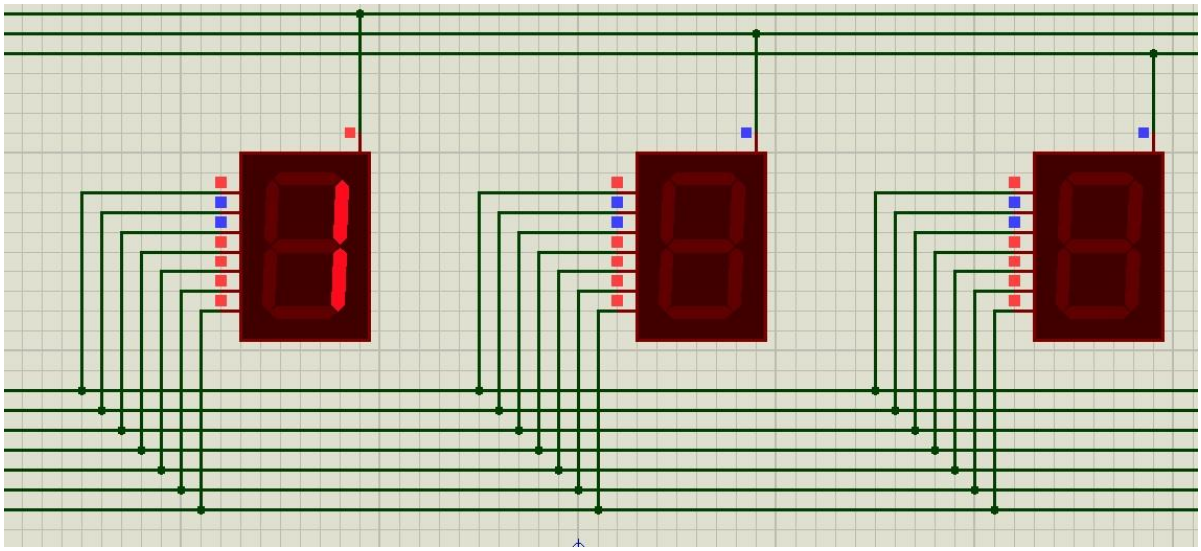


Рисунок 4 – Вывод информации на элемент

Если не предусмотреть данной выдержки времени для портов, то изображение не будет отображено на дисплее, либо будет отображено, но довольно тускло.

4) После вывода на элемент необходимой информации сбрасываем информационную линию в ноль или устанавливаем в единицу (рисунок 5) (для чего смотрите выше).

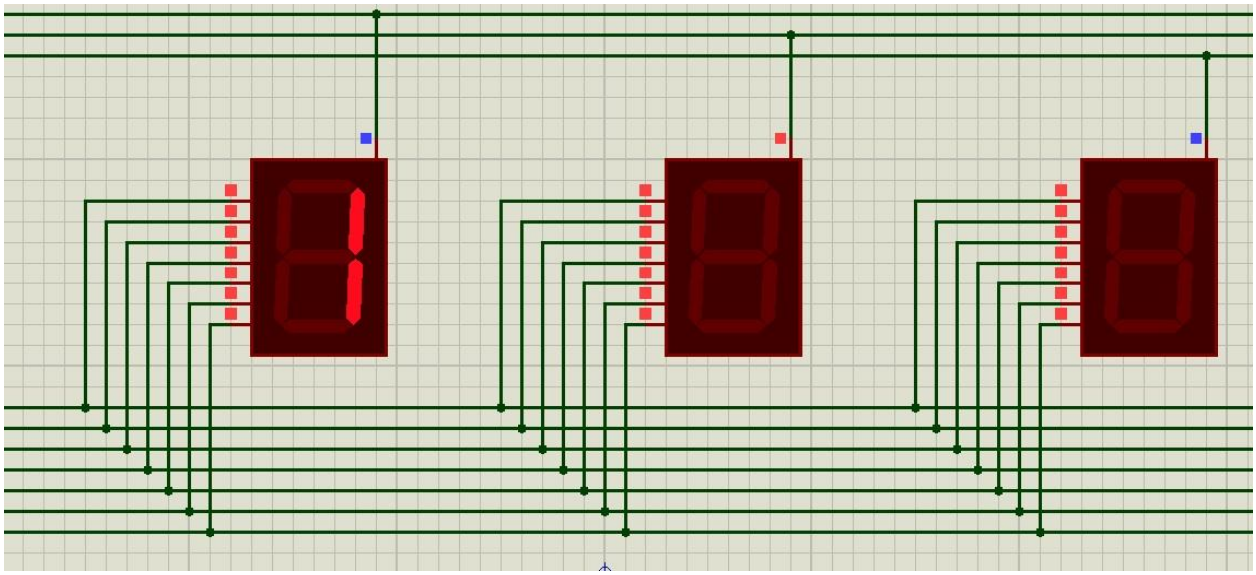


Рисунок 5 – Установка битов информационной линии

5) По линии выбора выбираем следующий необходимый элемент и повторяем все вышеизложенные операции заново (рисунок 6).

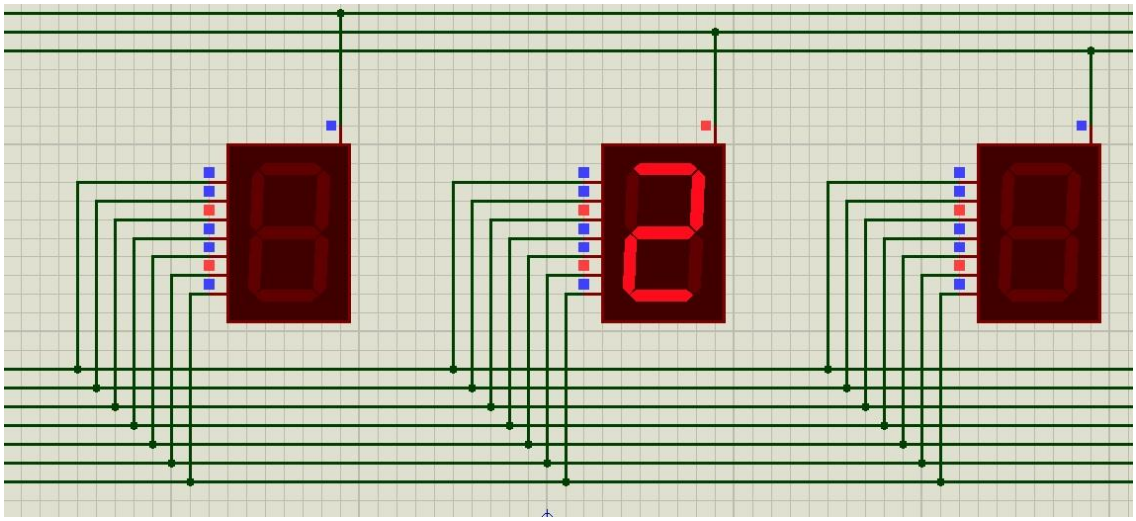


Рисунок 6 – Вывод информации на следующий элемент

Если этот процесс будет происходить с большой частотой, то человеческий глаз не успеет отследить включение и отключение светодиодных элементов и будет воспринимать целостную картину (рисунок 7)

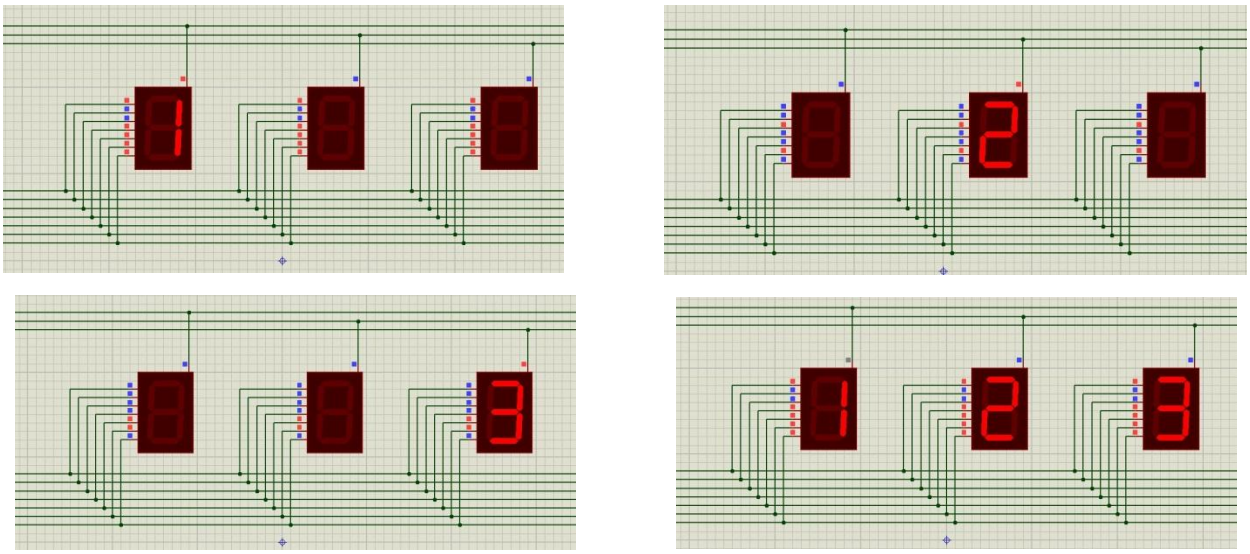


Рисунок 7 – Результат работы динамической индикации

Задания для самостоятельного выполнения

Создадим программу для реализации динамической индикации для схемы, изображенной на рисунке 8. Схема реализована на микроконтроллере PIC16F887. Для вывода информации взять дисплей с четырьмя

семисегментными светодиодными дисплеями с общим анодом. В качестве токоограничивающих резисторов взят блок резисторов RN1.

На микроконтроллере, в качестве порта вывода взят порт A, подключённый к выводам 1,2,3,4 дисплея. Высокий уровень на одном из этих вывод соответствует выбору одного из элементов. Порт B информационный, на него подается информация для вывода на выбранный элемент.

По алгоритму, описанному выше составим алгоритм работы программы для данной схемы для вывода на дисплей чисел (слева направо) 1,2,3,4 (рисунок 9).

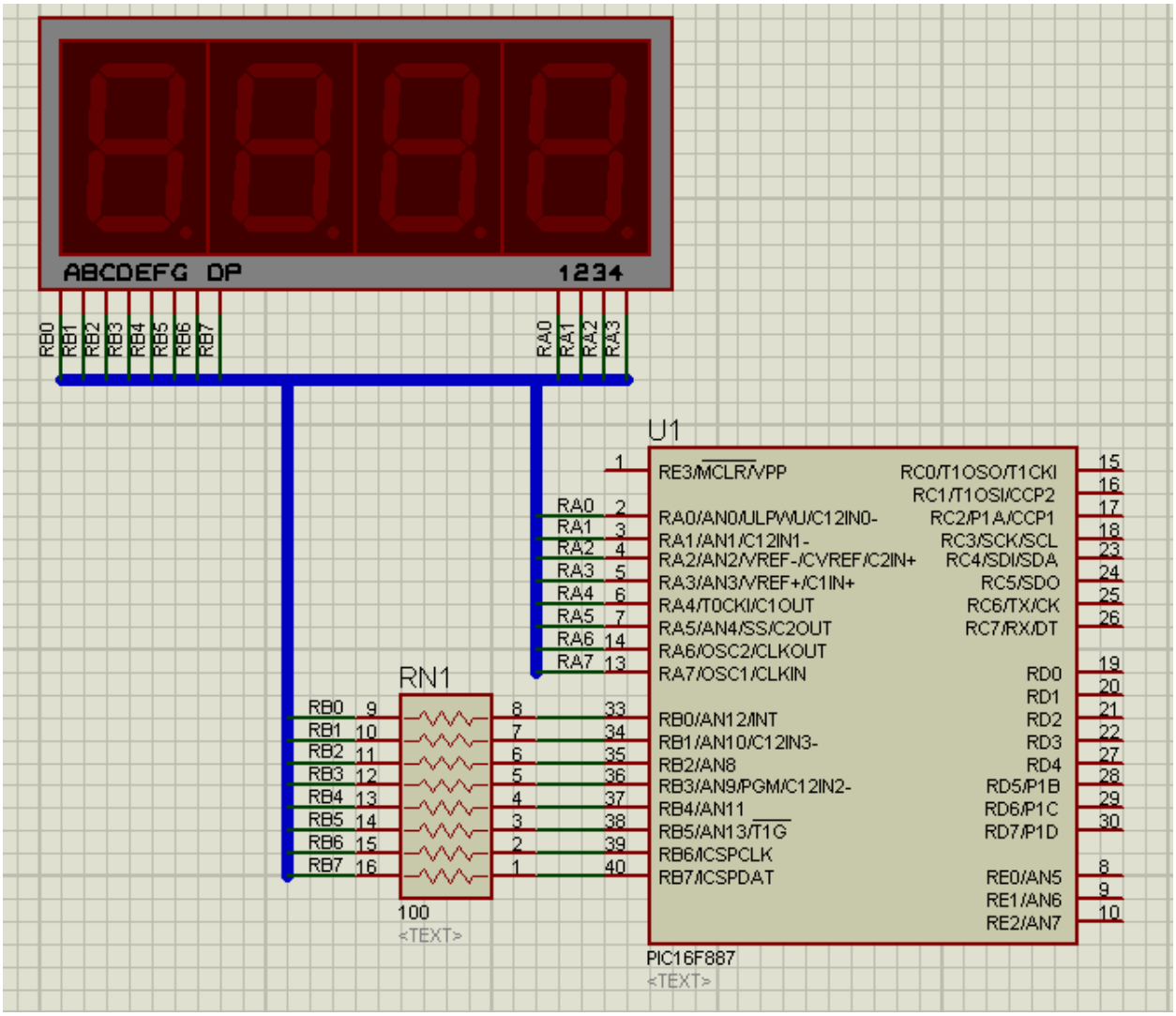


Рисунок 8 – Исследуемая схема

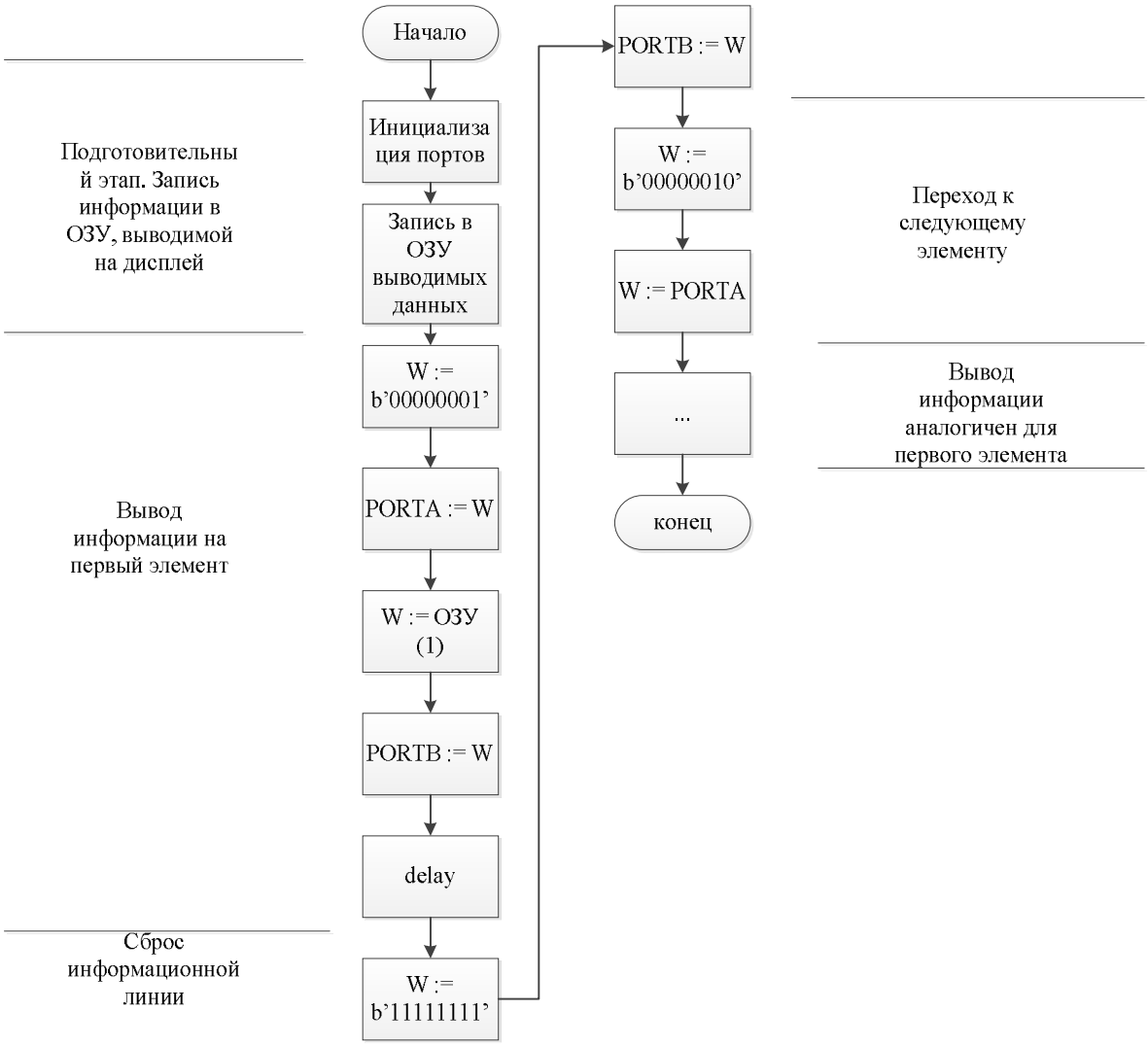


Рисунок 9 – Алгоритм динамической индикации для представленной схемы

Используя выше представленный алгоритм, напишем программу для его реализации.

ЛИСТИНГ 1

```
LIST   P=16f887, R=hex              ; Дириктива, определяющая тип процессора
                                    ; и систему счисления - шестнадцатеричная
;*****************************************************************

;*****************************************************************
;------------------------Прикрепленные файлы-------------------------------
;*****************************************************************
```

```
        INCLUDE P16F887.INC  ;  Файл  с  описаниями  регистров  и  флагов  и
                                                    ;конфигурации
;*****************************************************************

; Значения для инициализации портов
; 0 - бит вывода
; 1 - бит ввода
INITA EQU                       B'00000000'
INITB EQU B'00000000'
;*****************************************************************
;-------------------------Инициализация портов-----------------------------
;*****************************************************************
; Выбор банка 1 для инициализации портов. Определяется сочетанием битов
; регистра STATUS. RP0 = 1, RP1 = 0
        BSF STATUS,RP0
        BCF STATUS,RP1
        MOVLW INITA
        MOVWF TRISA
        MOVLW INITB
        MOVWF TRISB
;*****************************************************************

;*****************************************************************
;---------------Динамическая индикация на цифровом дисплее-------------------
;*****************************************************************
; Переходим к банку 0, в котором храниться информация по работе с портами
; Комбинация флагов регистра STATUS: RP0 = 0, RP1 = 0
        BCF STATUS,RP0
        BCF STATUS,RP1
        ;Значения выводимые на дисплей
        movlw b'11111001' ;1
        movwf 0x30
        movlw b'10100100' ;2
        movwf 0x31
        movlw b'10110000' ;3
```

```
        movwf 0x32
        movlw b'10011001';4
        movwf 0x33
        ; вывод значений
label:
   ; Первый элемент
        movlw b'00000001'
        movwf PORTA
        movfw 0x30
        movwf PORTB
        call delay ; Подпрограмма выдержки времени
        call sbros ; Подпрограмма сброса линий
        ; Второй элемент
        movlw b'00000010'
        movwf PORTA
        movfw 0x31
        movwf PORTB
        call delay  ;
        call sbros
        ; Третий элемент
        movlw b'00000100'
        movwf PORTA
        movfw 0x32
        movwf PORTB
        call delay
        call sbros
        ; Четвертый элемент
        movlw b'000001000'
        movwf PORTA
        movfw 0x33
        movwf PORTB
        call delay
        call sbros
        goto label
;*****************************************************************
```

```
; Подпрограмма выдержки времени
;*****************************************************************
; Выдерживает время для полного включения светодиодов
delay:
        MOVLW .1
        MOVWF 0x20
TIME_3
        MOVLW .1
        MOVWF 0x21
TIME_2
        MOVLW .20
        MOVWF 0x22
TIME_1
        DECF 0x23,1
        BNZ TIME_1
        DECF 0x21,1
        BNZ TIME_2
        DECF 0x20,1
        BNZ TIME_3
        RETURN
; *****************************************************************
; Подпрограмма сброса линий
;*****************************************************************
; Подпрограмма сбрасывает линии для того, что бы избежать вывода ложной ; информации
sbros:
        movlw b'00000000'
        movwf PORTA
        movlw b'11111111'
        movwf PORTB
        call delay
        return
;*****************************************************************
        END
```

РАБОТА № 5

РАБОТА ТАЙМЕРОВ МИКРОКОНТРОЛЛЕРОВ PIC

Цель работы:

1. Изучение принципа работы таймеров микроконтроллеров среднего семейства PIC.

2. Создание программы, обеспечивающей работу таймера.

Краткие теоретические сведения

При написании программ для микроконтроллеров достаточно часто требуется выполнение определенных действий спустя некоторое время после определенного события, для этого может подойти «задержка» небольшая подпрограмма выполняющая бессмысленные действия, не наносящие вреда работе основной программы. Но в это время микроконтроллер занят выполнением бессмысленного кода ради задержки во времени. Бывает так что помимо отсчета времени нам нужно чтобы выполнялись еще какие-либо действия, для этого подойдет таймер, пока выполняются определенные команды таймер выполняет отсчет не загружая работу контроллера только своими действиями.

1. Изучение работы таймера TMR0 на основе микроконтроллера pic16f627a

TMR0 – таймер/счетчик, имеет следующие особенности:

- 8-разрядный таймер/счетчик;
- возможность чтения и записи текущего значения счетчика;
- 8-разрядный программируемый предделитель;
- внутренний или внешний источник тактового сигнала;
- выбор активного фронта внешнего тактового сигнала;
- прерывания при переполнении (переход от FFh к 00h).

Блок схема модуля TMR0 показана на рисунке 1.

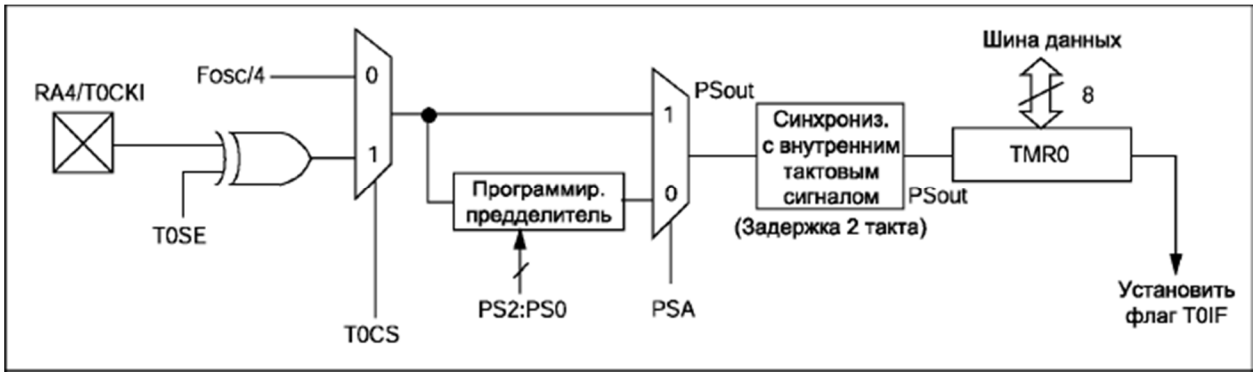


Рисунок 1 – Блок схема модуля TMR0

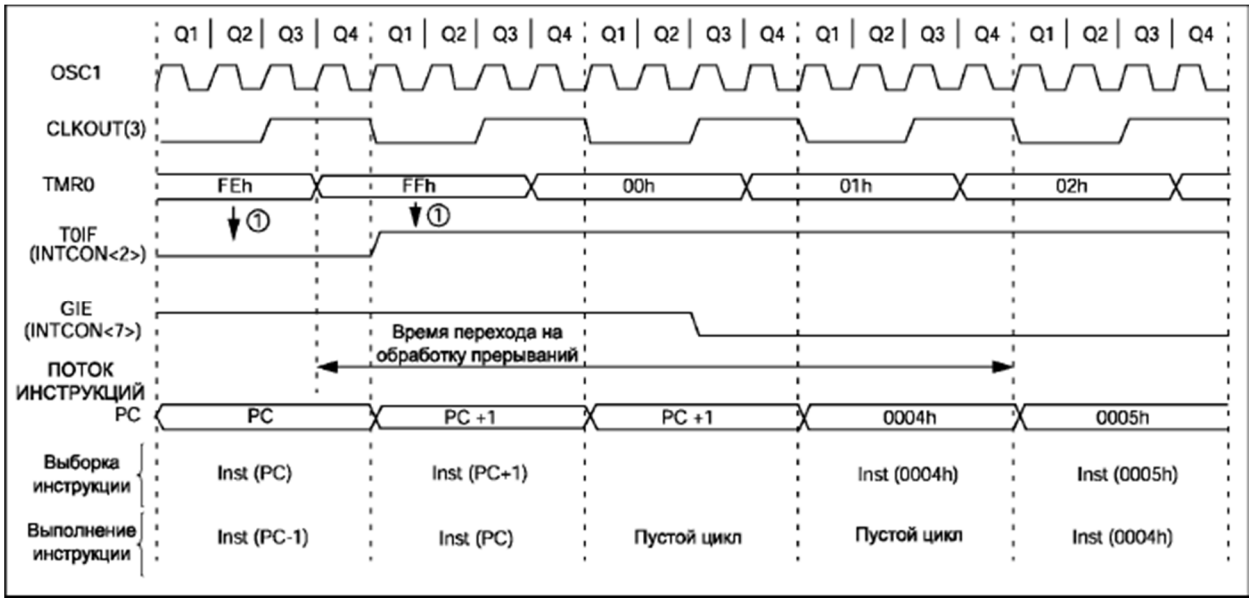
Биты управления T0CS, T0SE, PS2, PS1, PS0, PSA расположены в регистре OPTION.

Когда бит TOCS сброшен в ‘0’ (OPTION<5>), TMR0 работает от внутреннего тактового сигнала. Приращение счетчика TMR0 происходит в каждом машинном цикле (если предделитель отключен). После записи в TMR0 приращение счетчика запрещено два следующих. Пользователь должен предусмотреть эту задержку перед записью нового значения в TMR0. Если бит TOCS установлен в ‘1’ (OPTION<5>), TMR0 работает от внешнего источника тактового сигнала с входа RA4/T0CKI. Активный фронт внешнего тактового сигнала выбирается битом T0SE в регистре OPTION<4> (T0SE=0 – активным является передний фронт сигнала). Работа модуля TMR0 с внешним источником тактового сигнала будет рассмотрена ниже. Предделитель может быть включен перед WDT (сторожевой таймер) или TMR0, в зависимости от состояния бита PSA в регистре OPTION<3>. Если бит PSA сброшен в ‘0’, то предделитель включен перед TMR0. Нельзя прочитать или записать новое значение в предделитель. Когда предделитель включен перед TMR0, можно выбрать его коэффициент деления 1:2, 1:4, …, 1:256.

2. Прерывания от TMR0

Прерывания от TMR0 возникают при переполнении счетчика, т.е. при переходе его значения от FFh к 00h. При возникновении прерывания устанавливается в ‘1’ бит T0IF. Само прерывание может быть

разрешено/запрещено установкой/сбросом бита T0IE в регистре INTCON<5>. Флаг прерывания от TMR0 T0IF (INTCON<2>) должен быть сброшен в



подпрограмме обработки прерываний. В SLEEP режиме микроконтроллера модуль TMR0 выключен и не может генерировать прерывания. На рисунке 2 показана временная диаграмма возникновения прерывания от TMR0.

Рисунок 2 – Временная диаграмма возникновения прерывания от TMR0

Проверка установки флага T0IF выполняется в каждом цикле Q1. Время перехода на обработку прерывания $3T_{CY}$, где T_{CY} – машинный цикл. CLKOUT доступен только ER и INTRC режиме тактового генератора.

3. Использование внешнего источника тактового сигнала для TMR0

При использовании внешнего тактового сигнала для TMR0 необходимо учитывать некоторые детали работы таймера. Активный фронт внешнего тактового сигнала синхронизируется с внутренней тактовой частотой микроконтроллера, из-за чего возникает задержка от получения активного фронта сигнала до приращения TMR0.

3.1 Синхронизация внешнего сигнала

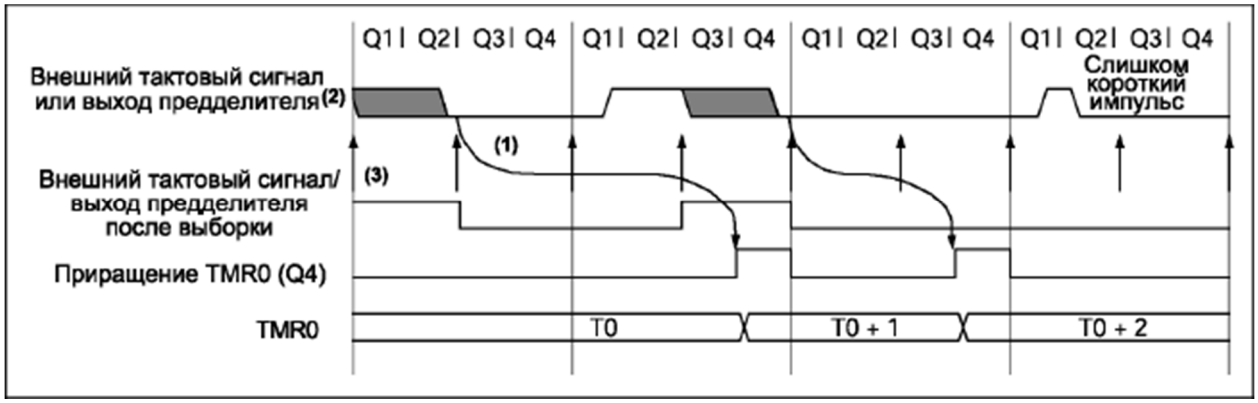
Если предделитель не используется, внешний тактовый сигнал поступает непосредственно на синхронизатор. Синхронизация T0CKI с таковым сигналом микроконтроллера усложняется из-за опроса выхода синхронизатора в машинные циклы Q2 и Q4 (см. рисунок 6-5). Поэтому

длительность высокого или низкого логического уровня внешнего сигнала должна быть не меньше 2T OSC (плюс небольшая задержка внутренней RC цепи 20нс). Дополнительную информацию смотрите в разделе электрических характеристик.

Если предделитель включен перед TMR0, то на вход синхронизатора поступает сигнал с асинхронного предделителя. Период сигнала T0CKI должен быть не менее 4T OSC (плюс небольшая задержка внутренней RC цепи 40нс) деленное на коэффициент предделителя. Дополнительное требование, высокий и низкий логический уровень внешнего сигнала должен быть не менее 10нс. Смотрите параметры 40, 41 и 42 в разделе электрических характеристик.

3.2 Задержка приращения TMR0

Поскольку сигнал с выхода предделителя синхронизируется с внутренним тактовым сигналом микроконтроллера, возникает задержка от



получения активного фронта сигнала до приращения TMR0 (см. рисунок 3).

Рисунок 3 – Временная диаграмма работы таймера TMR0 с внешним источником тактового сигнала

Задержка от активного фронта тактового сигнала до приращения TMR0 от $3T_{OSC}$ до $7T_{OSC}$. Следовательно, максимальная ошибка измерения интервала между двумя активными фронтами тактового сигнала $4T_{OSC}$. Если предделитель выключен, на вход синхронизатора поступает внешний тактовый сигнал. Стрелками указаны точки выборки уровня сигнала.

4. Предделитель

8-разрядный счетчик может работать как предделитель TMR0 или выходной делитель WDT (см. рисунок 6-6). Для простоты описания этот счетчик всегда будем называть «предделитель». Обратите внимание, что существует только один предделитель, который может быть включен перед TMR0 или WDT. Использование предделителя пред TMR0 означает, что WDT работает без предделителя, и наоборот.

Коэффициент деления предделителя определяется битами PSA и PS2:PS0 в регистре OPTION<3:0>.

Если предделитель включен перед TMR0, любые команды записи в TMR0 (например, CLRF 1, MOVWF 1, BSF 1,x и т.д.) сбрасывают предделитель. Когда предделитель подключен к WDT, команда CLRWDT сбросит предделитель вместе с WDT. Предделитель также очищается при сбросе микроконтроллера.

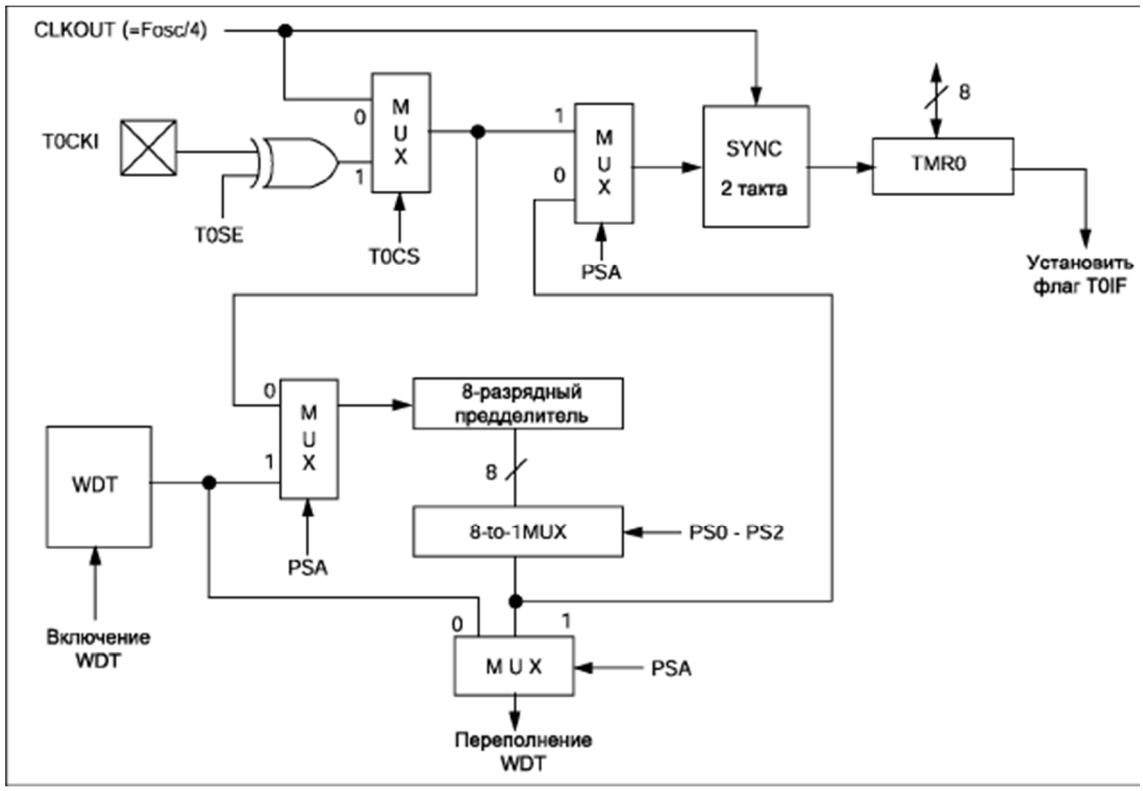


Рисунок 4 – Структурная схема предделителя

Биты управления T0CS, T0SE, PS2, PS1, PS0, PSA расположены в регистре OPTION.

4.1. Переключение предделителя

Переключение предделителя выполняется программным способом, т.е. переключение можно сделать во время выполнения программы. В рисунке 1 показана рекомендуемая последовательность инструкций переключения предделителя от TMR0 на WDT для предотвращения неожиданного сброса микроконтроллера.

Переключение предделителя от WDT на TMR0 показано в рисунке 2. Меры осторожности должны применяться, даже если сторожевой таймер WDT выключен.

```
BCF     STSTUS, PRO     ; Выбрать банк 0
CLRWDT                  ; Сбросить WDT
CLRF    TMR0            ; Сбросить TMR0 и предделитель
BSF     STATUS, PRO     ; Выбрать банк 1
MOVLW   b'00101111'     ; Три строки (5, 6. 7) должны быть включены в
MOVWF   OPTION_REG      ; текст программы только, если биты
CLRWDT                  ; PS<2:0> равны значению 000 или 001

MOVLW   b'00101xxx'     ; Переключить предделитель на WDT,
MOVWF   OPTION_REG      ; выбирать коэффициент деления
BCF     STSTUS, PRO     ; Выбрать банк 0
```

Рисунок 5 – Переключение от TMR0 к WDT

```
CLRWDT                  ; Сбросить WDT и предделитель
BSF     STATUS, PRO     ; Выбрать банк 1
MOVLW   b'xxxx0xxx'     ; Включить предделитель перед TMR0 и
MOVWF   OPTION_REG      ; выбрать новое значение коэффициента деления
BCF     STSTUS, PRO     ; Выбрать банк 0
```

Рисунок 6 - Переключения предделителя от WDT к TMR0

Адрес	Имя	Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0	Сброс POR	Другие сбросы
01h	TMR0	Регистр таймера 0								xxxx xxxx	uuuu uuuu
0Bh/8Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
81h	OPTION	-RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
85h	TRISA	TRISA7	TRISA6	-	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	11-1 1111	11-1 1111

Таблица 1 – Регистры и биты связанные с работой TMR0

Обозначения: - – не используется, читается как 0; u – не изменяется; x – не известно; q – зависит от условий.

Примечание. Затемненные биты не влияют на работу TMR0.

Ниже приведены побитные описания основных регистров связанных с работой таймера TMR0.

Регистр TMR0 осуществляет непосредственно счет, биты этого регистра не несут в себе особенной информации кроме цифры (сколько таймер отсчитал). В этот регистр можно как записывать число, так и считывать из него.

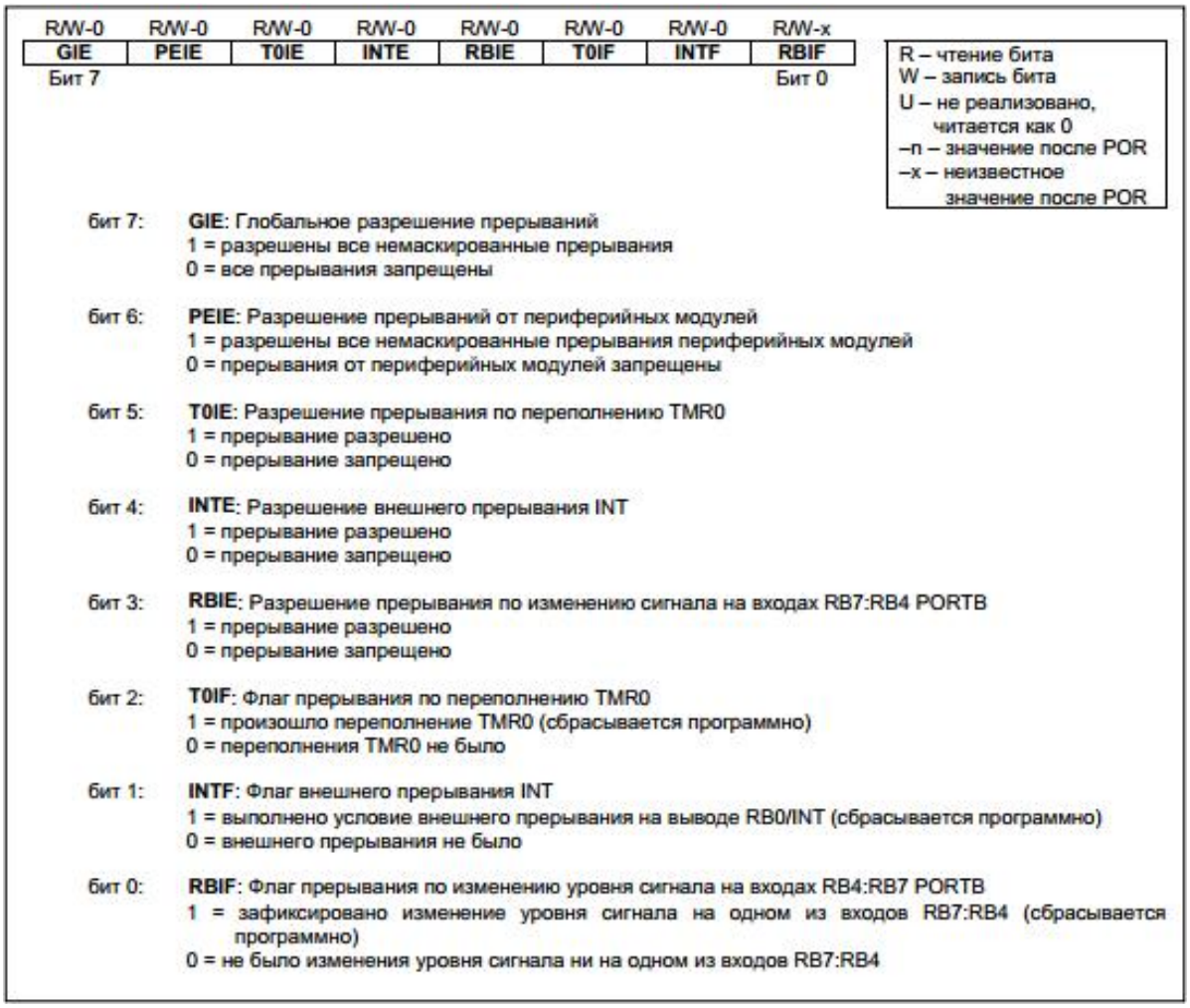


Рисунок 5 – Побитное описание регистра INTCON

5. Работа таймера TMR2

По принципу работы таймер TMR2 не сильно отличается от TMR0.Отличия состоят в том, что его можно включить, есть различие в принципе работы предделителя и ему можно задать максимальное значение счета. Таймер TMR0 включить нельзя и начало его отсчета можно обозначить записью в него нуля.

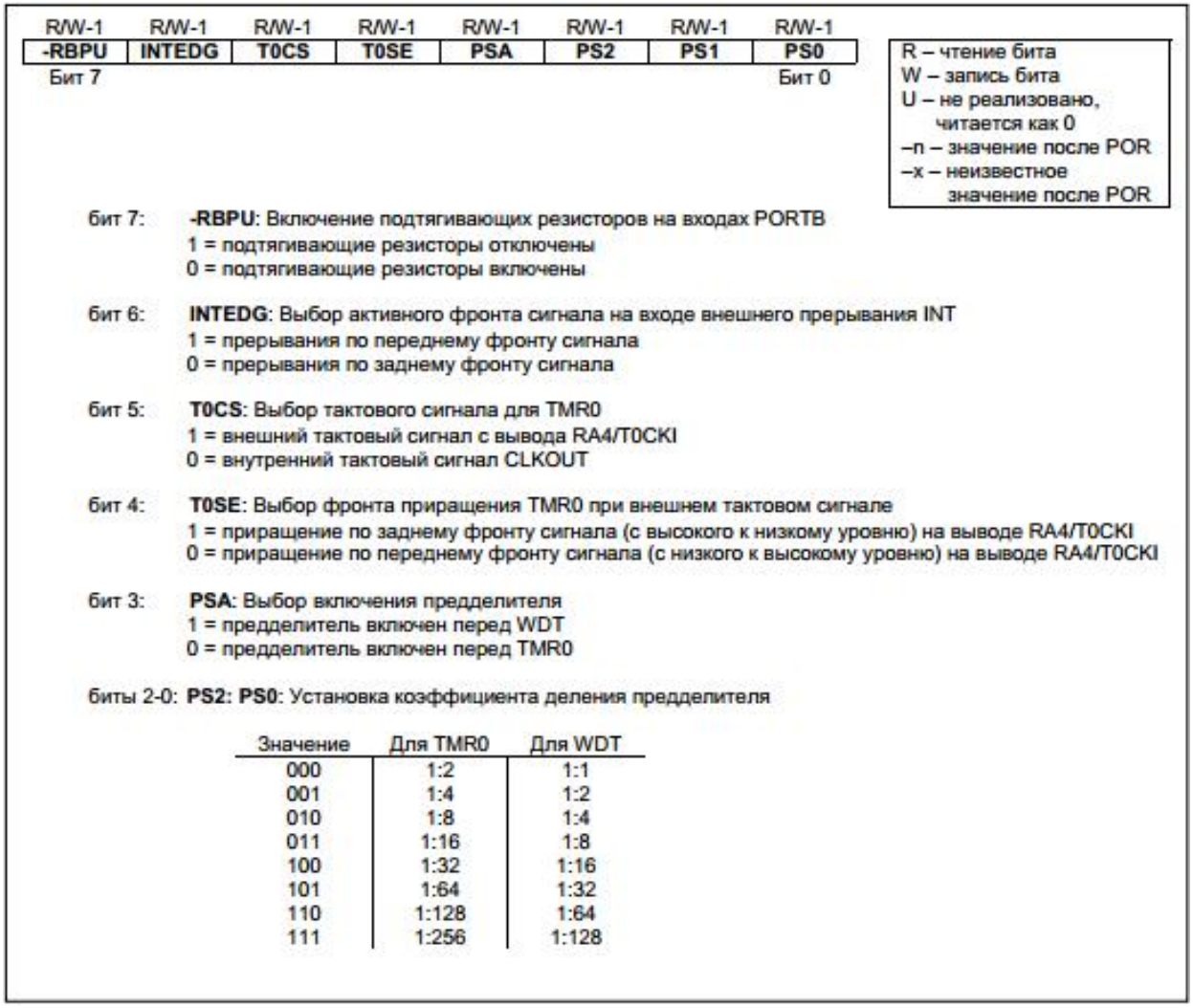


Рисунок 6 – Побитное описание регистра OPTION

6. Предделитель и входной делитель таймера TMR2

Делитель счета таймера здесь разделен на 2. Предделитель 1:1 ,1:2 …, 1:16 и входной делитель 1:1, 1:4 и 1:16.

Счетчик предделителя сбрасывается в случае записи в регистры TMR2, T2CON и любого вида сброса микроконтроллера (POR, BOR, сброс WDT или активный сигнал -MCLR). Регистр TMR2 не очищается при записи в T2CON.

7. Регистры, связанные с работой таймера TMR2

Побитное описание регистра INTCON представлено выше.

Регистры PIR1 и PIE1 содержат по одному регистру, связанному с работой таймера TMR2.

Таблица 2 – регистры, связанные с работой таймера TMR2

Адрес	Имя	Бит 7	Бит 6	Бит 5	Бит 4	Бит 3	Бит 2	Бит 1	Бит 0	Сброс POR	Другие сбросы
0Bh/8Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	EEIF	CMIF	RCIF	TXIF	-	CCP1F	TMR2IF	TMR1IF	0000 -000	0000 -000
8Ch	PIE1	EEIE	CMIE	RCIE	TXIE	-	CCP1E	TMR2IE	TMR1IE	0000 -000	0000 -000
11h	TMR2	Регистр таймера 2								0000 0000	0000 0000
12h	T2CON	-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-uuu uuuu
92h	PR2	Регистр периода таймера 2								1111 1111	1111 1111

TMR2IF – Флаг прерывания по переполнению таймера TMR2:

1 – Произошло переполнение TMR2 (сбрасывается программно);

0 – Переполнения TMR2 не было;

TMR2IE – Разрешение прерывания по переполнению TMR2:

1 – прерывание разрешено;

0 – прерывание запрещено;

Регистр TMR2 осуществляет непосредственно счет, биты этого регистра не несут в себе особенной информации кроме цифры (сколько таймер отсчитал). В этот регистр можно как записывать число, так и считывать из него.

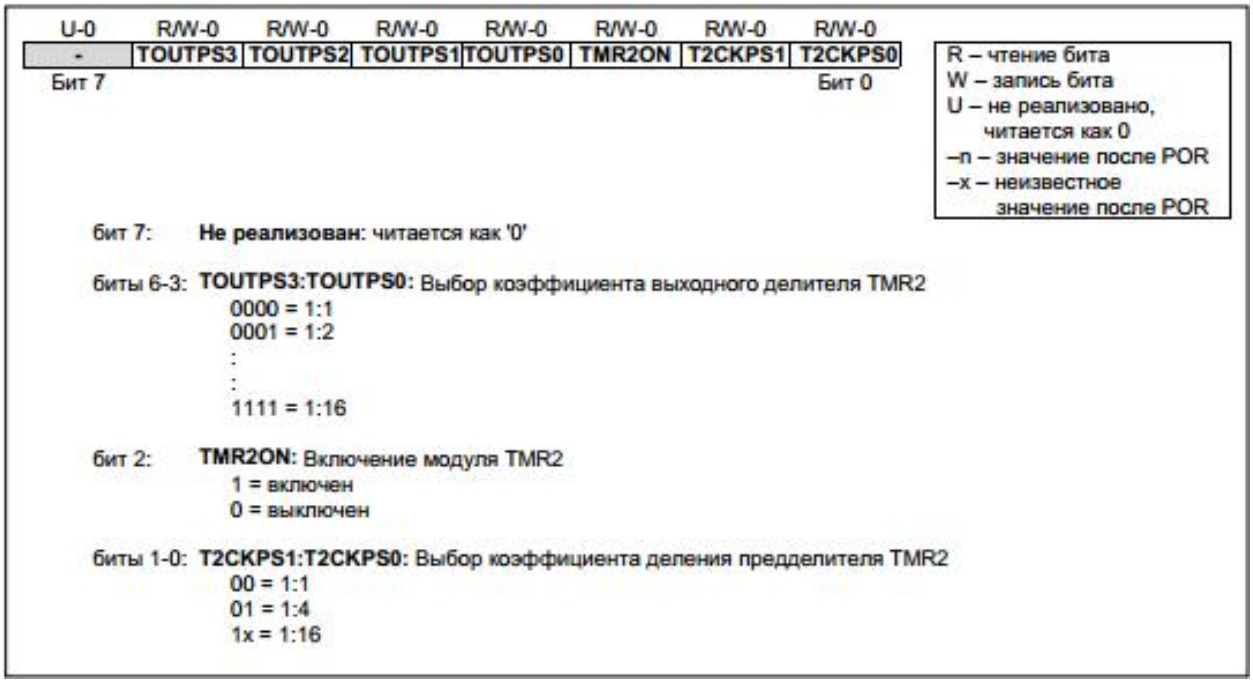


Рисунок 7 – Побитное описание регистра T2CON

Регистр PR2 устанавливает максимальное значение счета, по умолчанию, когда PR2=0 тогда максимальное значение счета 0xFF или 256. Если же PR2≠0 он становится максимумом счета регистра TMR2, при

увеличении регистра TMR2 он постоянно сравнивается с PR2 и когда TMR2=PR2 устанавливается флаг переполнения, или, если прерывания разрешены, происходит прерывание.

Порядок работы

Задание: На основе микроконтроллера PIC16F627A разработать программу демонстрирующую работу таймера TMR0.

Для расчета времени нужно определиться с частотой работы микроконтроллера. Частоту возьмем 100 КГц эта частота не сильно большая, не придется использовать несколько буферных регистров для сохранения количества переполнений таймера, данной частоты достаточно для выполнения нашей задачи и многих других. Приращение счетчика происходит каждый машинный цикл, а машинный цикл выполняется за 4 такта, следовательно, для того чтобы таймер зафиксировал время прохождения 1 секунды нужно чтобы он увеличился 100000/4=25000 раз. Регистр таймера TMR0 восьми разрядный, следовательно, максимальное число которое в него можно поместить 256. Тогда для отсчета 1 секунды нужно чтобы таймер переполнился 25000/256=97 раз. У таймера есть еще предделитель с коэффициентом деления 1:2, 1:4, …, 1:256. Это значит, чтобы таймер счетчик увеличился на 1 нужно 2, 4, …, 256 машинных циклов соответственно. Теперь включая предделитель 1:128 получается 250000/128=196 это значит, что таймер должен достигать до 196 для того чтобы прошла 1 секунда. После просчета предделителя можно приступить к созданию модели и далее уже непосредственно к разработке программного кода.

Для демонстрации работы таймера воспользуемся светодиодом, который будет моргать с заданной частотой, для примера возьмем частоту в 1 секунду.

Для создания модели воспользуемся программой Proteus.

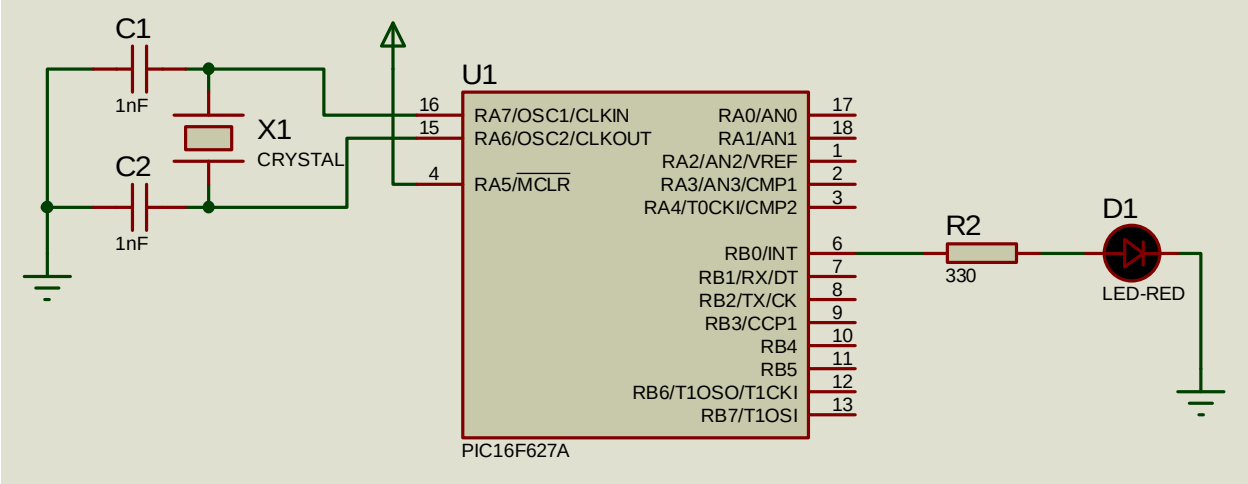


Рисунок 8 – Модель устройства демонстрирующего работу таймера

Далее перейдем к разработке алгоритма будущей программы.

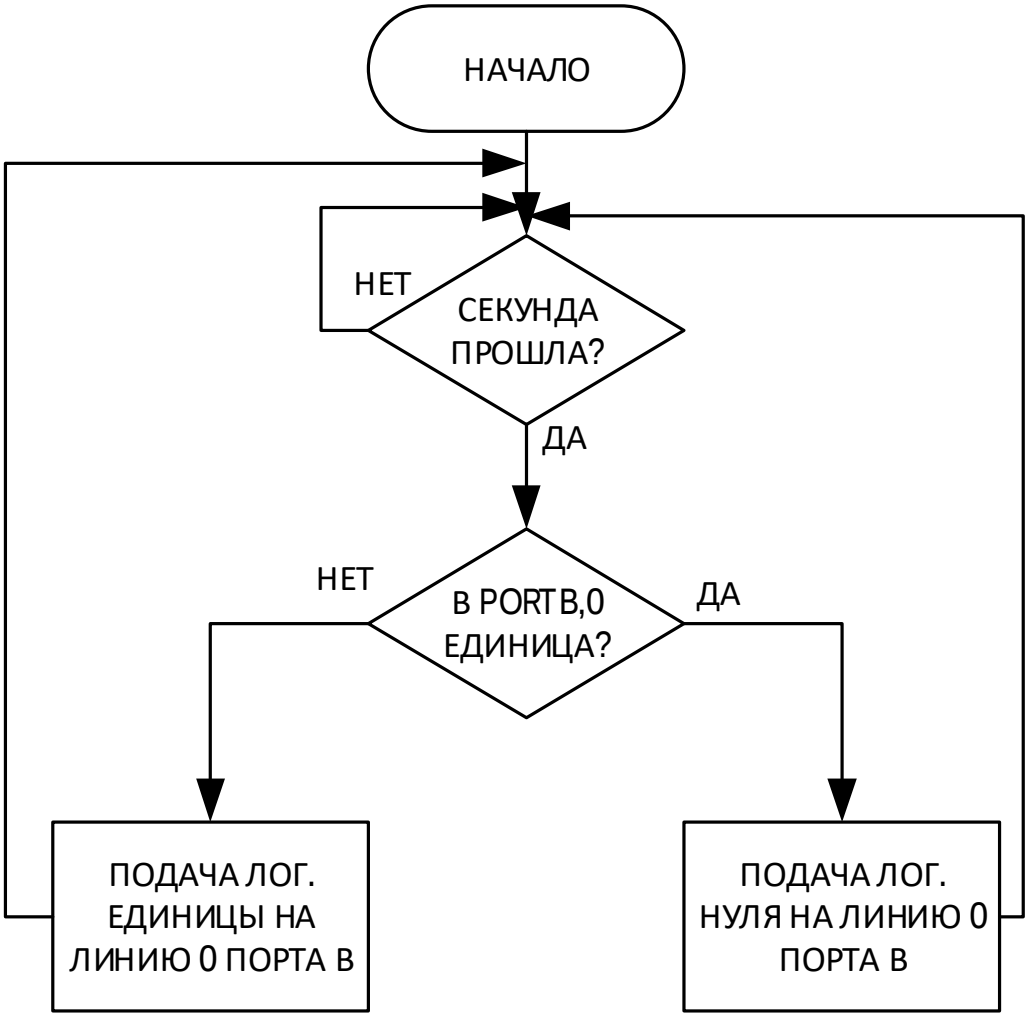


Рисунок 9 – Алгоритм работы программы

В начале определяем модель контроллера и слово конфигурации затем происходит инициализация портов, регистров общего и специального назначения.

```
            LIST        P=PIC16f627A, R=HEX
            __CONFIG    0x3F71
;=================================================================================
;ОПРЕДЕЛЕНИЕ РЕГИСТРОВ СПЕЦИАЛНОГО НАЗНАЧЕНИЯ
;=================================================================================
TMR0            EQU         01H
OPTION_REG      EQU         01H
STATUS          EQU         03H
RP0             EQU         5
PORTA           EQU         05H
TRISA           EQU         05H
PORTB           EQU         06H
TRISB           EQU         06H
W               EQU         0
F               EQU         1
C               EQU         0
;=================================================================================
;ОПРЕДЕЛЕНИЕ РЕГИСТРОВ ОБЩЕГО НАЗНАЧЕНИЯ
;=================================================================================
SRAVN           EQU         0x20            ;СЛУЖИТ ДЛЯ ХРАНЕНИЯ  МАКСИМАЛЬНОГО ЧИСЛА
;==============НАЧАЛО ПРОГРАММЫ=================================================
                ORG         0x00            ;УСТАНОВКА НАЧАЛЬНОГО АДРЕСА ПО СБРОСУ
;=================================================================================
;ПОДГОТОВИТЕЛЬНЫЕ МОМЕНТЫ (ИНИЦИАЛИЗАЦИЯ ПОРТОВ УСТАНОВКА ПРЕДДЕЛИТЕЛЯ И Т.Д.)
;=================================================================================
                BCF         STATUS,RP0      ; ПЕРЕХОД В БАНК 0
                CLRF        PORTA           ; ОЧИСТКА ПОРТА А
                CLRF        PORTB           ; ОЧИСТКА ПОРТА В
                BSF         STATUS,RP0      ; ПЕРЕХОД В БАНК 1
                MOVLW       0xFF            ; ОТПРАВИТЬ КОНСТАНТУ В АККУМУЛЯТОР
                MOVWF       TRISA           ; ОТПРАВИТЬ СОДЕРЖИМОЕ W В РЕГИСТР
                MOVLW       0x00            ; ОТПРАВИТЬ КОНСТАНТУ К АККУМУЛЯТОР
                MOVWF       TRISB           ; ОТПРАВИТЬ СОДЕРЖИМОЕ W В РЕГИСТР
                MOVLW       0x06            ; ОТПРАВИТЬ ЧИСЛО "00000110" В АККУМУЛЯТОР
                MOVWF       OPTION_REG      ; ОТПРАВИТЬ ИЗ АКК В РЕГИСТР
                BCF         STATUS,RP0      ; ПЕРЕХОД В БАНК 0
                CLRF        TMR0            ;
;=================================================================================
```

Рисунок 10 – Начало программы

Далее начинается основная программа где проверяется число, которое отсчитывает таймер и сравнивается с числом 194, мы вычислили что для того чтобы прошла секунда должно пройти 196 машинных циклов, от этого числа мы отняли 2 машинных цикла которые выжидаются после обнуления таймера.

Эта подпрограмма выполняется циклично пока таймер не достигнет 196. Здесь же можно было бы добавить опрос кнопки для выполнение какой-либо задачи, пока таймер делает свое дело.

```
;=================================================================================
;ОСНОВНАЯ ПОДПРОГРАММА
;=================================================================================
START           MOVLW       0xC2            ;ЗАГРУЖАЕМ В АКК 194(ПОЧЕМУ НЕ 196?)
                MOVWF       SRAVN           ;ИЗ АКК В РЕГИСТР
                MOVF        TMR0,W          ;СЧИТЫВАЕМ ЗНАЧЕНИЕ В ТАЙМЕРЕ
                SUBWF       SRAVN,W         ;АКК=SRAVN-АКК
                BTFSS       STATUS,C        ;ПРОВЕРКА ФЛАГА С РЕГИСТРА СТАТУС НА ЕДИНИЦУ
                CALL        DIODE           ;ПЕРЕХОД ПО МЕТКЕ С ВОЗВРАТОМ
                GOTO        START           ;ЗАЦИКЛИВАНИЕ ПОДПРОГРАММЫ
```

Рисунок 11 – Основная программа

Последняя часть программы заключается в проверке порта Б линии 0 и загрузки в него противоположного сигнала, для индикации работы таймера.

```
;=================================================================================
;ПОДПРОГРАММА ПРОВЕРКИ ГОРЕНИЯ ДИОДА И ВЫВОДА ПРОТИВОПОЛОЖНОГО СИГНАЛА
;=================================================================================
DIODE           BTFSC       PORTB,0         ;ПРОВЕРКА ЛИНИИ 0 ПОРТА В НА 0
                GOTO        VIKL            ;ПЕРЕХОД НА ВЫКЛ. СВЕТОДИОДА
                GOTO        VKL             ;ПЕРЕХОД НА ВКЛ. СВЕТОДИОДА
VIKL            BCF         PORTB,0         ;ОБНУЛЕНИЕ ЛИНИИ 0 ПОРТА В
                GOTO        OBNYL           ;ПЕРЕХОД ПО МЕТКЕ
VKL             BSF         PORTB,0         ;УСТАНОВКА ЛОГ. ЕДИНИЦЫ НА ЛИНИИ 0 ПОРТА В
                GOTO        OBNYL           ;ПЕРЕХОД ПО МЕТКЕ
OBNYL           CLRF        TMR0            ;ОБНУЛЕНИЕ ТАЙМЕРА ДЛЯ ОТСЧЕТА ВРЕМЕНИ
                RETURN                      ;ВОЗВРАТ ИЗ ПОДПРОГРАММЫ
;==============КОНЕЦ ПРОГРАММЫ=================================================
                END
```

Рисунок 12 – Подпрограмма реализующая переключение диода

Задания для самостоятельного выполнения

1. Сделать вывод чисел на семи сегментный индикатор посекундно.
2. Добавить в схему кнопки начала отсчета и сброса
3. Использовать в работе другой таймер
4. Использовать в работе прерывания от таймера/ов.

5 Продемонстрировать параллельную работу контроллера. (помимо мигания диода сделать включение и выключение другого диода по нажатию кнопки).

6 Сделать программу светофор для пешеходов с использованием таймера.

РАБОТА № 6

РАБОТА С LCD ДИСПЛЕЕМ

Цель работы:

1. Изучение принципа работы и правила подключения LCD дисплея.
2. Создание программ, обеспечивающих вывод информации на LCD дисплей.

Краткие теоретические сведения

ЖКИ или ЖК монитор часто используются для вывода информации в удобной для восприятия человеком форме (см. рисунок 1).

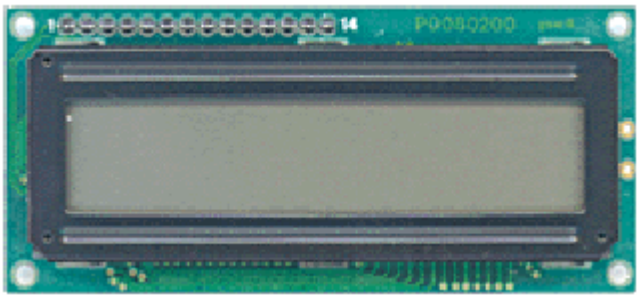


Рисунок 1 - Пример Жидкокристаллического индикатора

Существует большое количество различных типов ЖКИ, но в основном (в 99%) распространены индикаторы с интерфейсом Hitachi 44780. Что же представляют собой эти индикаторы? Большинство таких индикаторов имеет 14 выводов. Назначение этих выводов приведено на рисунке 2.

ВЫВОД	ОБОЗНАЧЕНИЕ	НАЗНАЧЕНИЕ
1	Vdd	Общий
2	Vcc	+ Питание
3	Contrast (Vee)	Регулировка контрастности
4	R/S	_Команда/Выбор регистра
5	R/W	Чтение/_Запись
6	E	Тактовые импульсы
7-14	Data	Данные: D0 -7,…,D7 - 14

Рисунок 2 – Назначение выводов LCD дисплея

Запись информации происходит в параллельном коде по фронту тактовых импульсов E. Можно не только записывать данные в регистры ЖКИ, но и читать их. Для этого используется цепь R/W. Для подключения шины данных можно использовать 2 режима: Это режим 8-разрядной шины, когда данные в контроллер ЖКИ передаются байтами, или режим 4-разрядной шины, когда вначале передаются старшие 4 разряда байта, а затем младшие 4 разряда. Хотя понятно, что при 4-разрядной шине количество соединений меньше и загруженность портов меньше, но при этом и скорость обмена меньше. Цепь R/S предназначена для указания типа информации, которая в данный момент подается на выводы данных. Если на линии R/S установлена логическая 1, то на шине данных можно устанавливать ASCII код, символ которого будет отображаться в текущей позиции курсора. Если же на линии R/S будет установлен логический 0, а в цепи R/W логическая 1, то можно считать код символа, отображаемого в данный момент на экране. Хотя в большинстве конструкций вывод R/W подключается к общей шине питания, так как нет необходимости использовать режим чтения. На выполнение команд ЖКИ затрачивается разное время. Например, для стирания экрана или на перемещение курсора в начальную позицию требуется около 4,1 см, а на другие команды достаточно 160 мксм.

Запись данных производится так же, как и запись команд, только в первом случае по цепи R/S должна быть подана 1. Но использование компилятора PicBasicPro избавляет вас от необходимости изучения интерфейса с такими ЖКИ. Он все сделает сам. Для регулировки контрастности изображения вы можете воспользоваться схемой, приведенной на рисунке 3.

Все возможные символы, которые можно выводить на LCD индикатор, представлены на рисунке 4.

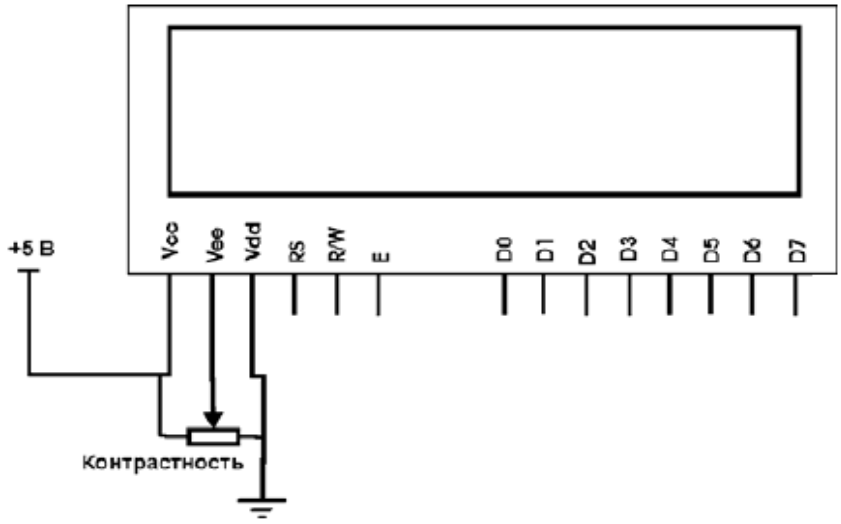


Рисунок 3 - Схема подключения регулятора контрастности к LCD

Рисунок 4 – Символы для LCD

Порядок работы

Задание: На основе микроконтроллера разработать устройство, которое выводит на индикатор “HELLO 041”.

Схема, собранная в программе – эмуляторе PROTEUS показана на рисунке 5.

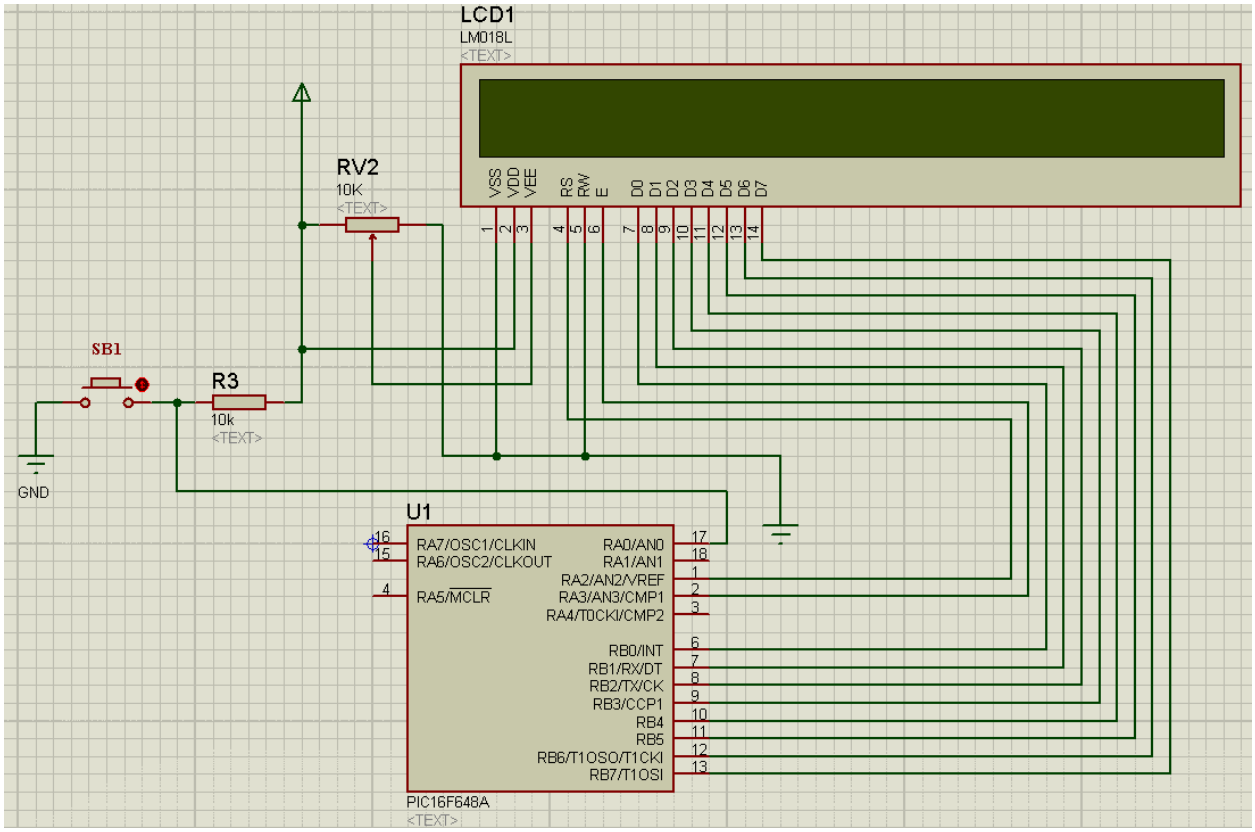


Рисунок 5 – Схема в программе – эмуляторе PROTEUS

Разработка программного кода

На рисунке 6 представлена подпрограмма «включения дисплея».

```
add
                MOVLW b'00001111'   ;Включается дисплей и курсор
                movwf PORTB         ;порт B
                bsf    PORTA,3       ; E = 1 (разрешение)
                bcf    PORTA,3       ; E = 0
                return      ;возрат подпрограммы
```

Рисунок 6 – Подпрограмма включения дисплея

Индикатор включается подачей на шину данных константы b'00001111', после чего необходимо подать импульс E. Индикатор включится и появится мигающий курсор (см. рисунок 7).

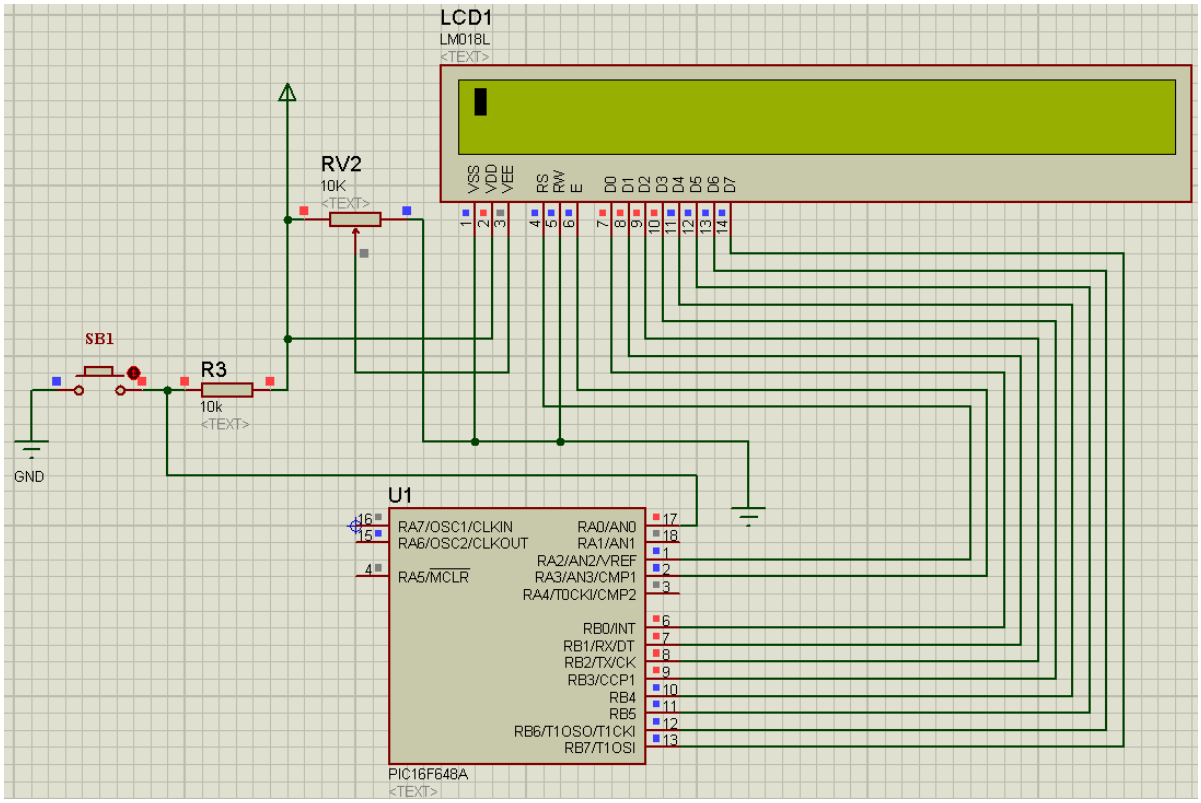


Рисунок 7 – Схема в программе – эмуляторе PROTEUS

На рисунке 8 представлена подпрограмма опроса кнопки и вывода буквы “H” на индикатор.

```
opros
                btfss PORTA,0
                goto  Sb1
                goto opros
Sb1
                 MOVLW  0x48
                 movwf  TRISB          ;пересылка из W в порт B
                 bsf    PORTA,2        ; RS=1
                 bsf    PORTA,3        ; E=1 (разрешение)
                 bcf    PORTA,3        ; E=0
                 call   Pause1
```

Рисунок 8 – Подпрограмма вывода числа

Из рисунка 4 мы находим букву “H”, ей соответствует номер 48. Эту константу мы отправляем по шине данных на индикатор. Затем подаем логическую единицу на вывод RS, для того чтобы указать тип информации, которая в данный момент подается на выводы данных. То есть на шине данных можно устанавливать ASCII код. После чего подаем импульс на вывод E, после чего на дисплее появится буква “H”. Далее следует задержка.

Таким образом, будем выводить все остальные буквы поочерёдно через определенную задержку. Полный код представлен в приложении А.

Задания для самостоятельного выполнения

1. Пользуясь таблицей ASC нарисовать символ по варианту…
2. Ввести вторую кнопку и сделать так, чтобы на дисплей выводился номер нажатой кнопки.
3. Организовать считывание информации с дисплея и вывод его на семисегментный дисплей.
4. Изменить точку начала вывода информации
5. Ввести в рассмотрение кнопку, удаляющую информацию.
6. Измените выводимое слово на свою фамилию, имя и отчество.

```
;*************************************************************
;*листинг исходной программы
;*************************************************************
        LIST P=PIC16F648A, R=HEX;деректива, определяющая тип
                        ;процессора и систему счисления
                        ;по умолчанию
;*************************************************************
;*описание используемых переменных и назначение адресов
;*ячеек для хранения переменных пользователя
;*************************************************************
;------Registr Files--------------------------------------
INDF        EQU   H'0000' ;доступ к памяти через FSR
TMR0        EQU   H'0001' ;таймер
PCL         EQU   H'0002' ;счетчик команд
STATUS      EQU   H'0003' ;регистр состояния алу.
FSR         EQU   H'0004' ;регистр косвенной адресации.
PORTA       EQU   H'0005' ;порт А в/выв.
PORTB       EQU   H'0006' ;порт В в/выв.
EEDATA      EQU   H'0008' ;Данных ПЗУ
EEADR       EQU   H'0009' ;Адрес ПЗУ
INTCON      EQU   H'000B' ;регистр флагов прерыванИЙ.
OPTION_REG  EQU   H'0081'
TRISA       EQU   H'0005' ;направление данных порта А.
TRISB       EQU   H'0006' ;направление данных порта В.
EECON1      EQU   H'0088' ;регистры чтения-записи
EECON2      EQU   H'0089' ;ПЗУ
W           EQU   H'0000' ;аккамулятор
F           EQU   H'0001' ;
Reg_1       EQU   0x0C    ; регистры простого пользования
Reg_2       EQU   0x0D    ;
Reg_3       EQU   0x0E    ;
Reg_4       EQU   0x0F    ;
Reg_5       EQU   0x12    ;
```

```
Reg_6       EQU   0x31    ;
Reg_7       EQU   0x41    ;
Reg_8       EQU   0x71    ;
Reg_9       EQU   0x61    ;
Reg_10      EQU   0x19    ;
;------STATUS Bits--------------------------------------
IRP         EQU   H'0007'
RP1         EQU   H'0006'
RP0         EQU   H'0005'
NOT_TO      EQU   H'0004'
NOT_PD      EQU   H'0003'
Z           EQU   H'0002'
DC          EQU   H'0001'
C           EQU   H'0000'
;------INTCON Bits--------------------------------------
GIE         EQU   H'0007'
EEIE        EQU   H'0006'
T0IE        EQU   H'0005'
INTE        EQU   H'0004'
RBIE        EQU   H'0003'
TOIE        EQU   H'0002'
INTF        EQU   H'0001'
RBIF        EQU   H'0000'
;*****************************************************
;*определение меток замены текста
;*****************************************************
 #DEFINE     z     STATUS,2    ;Бит нулевого результата
 #DEFINE     SB1   PORTA,0     ;определение кнопки 1
 #DEFINE     SB2   PORTA,1     ;определение кнопки 2
 #DEFINE     SBsrav PORTA,4    ;определение кнопки сравнения
 #DEFINE     LCD_DREG PORTB    ;Определям порт, к которому
                    ;подключены цепи данных.
 #DEFINE     LCD_DBIT 0        ;Определяем первый вывод, к ;которому подключена ШД.
```

```
 #DEFINE     LCD_RSREG PORTA   ;Опр.порт, к которому подкл.RS
 #DEFINE     LCD_RSBIT 2       ;Опр.вывод, к которму подк.RS
 #DEFINE     LCD_EREG  PORTA   ;Опр.порт, подк.E
 #DEFINE     LCD_EBIT  3       ;Опр.вывод, подк.E
 #DEFINE     LCD_BITS  8       ;Опр.режим 8-разрядной шины
 #DEFINE     LCD_LINES 2       ;Опр.тип ЖКИ
 #DEFINE     LCD_COMMANDUS 2000 ;Опр.время задержки М/У командами
 #DEFINE     LCD_DATAUS 50     ;Опр.время задержки М/У посылками данных
;=================================================================
          ORG   0    ;начало
          call Int   ;объявление портов ввода/вывода
          call add   ;включение дисплея
          call opros    ;опрос кнопок
Int
          MOVLW 0x00         ;обнуление портов
          MOVWF PORTA        ;порта A
          MOVWF PORTB        ;порта B
          BSF   STATUS,5     ;переход в банк 1
          MOVLW b'10010011'   ;настройка линий RA0 RA1
          MOVWF TRISA        ;и RA7 на ввод, а остольные - на вывод
          MOVLW b'00000000'   ;настройка порт B
          MOVWF TRISB        ;весь порт B на вывод
          BCF   STATUS,RP0    ;возврат в банк 0
          return    ;возврат подпрограммы
add
          MOVLW b'00001111'   ;Включается дисплей и курсор
          movwf PORTB        ;порт B
         bsf    PORTA,3                ; E = 1 (разрешение)
         bcf    PORTA,3      ; E = 0
          return     ;возрат подпрограммы
opros
          btfss PORTA,0
          goto  Sb1
```

```
        goto opros
Sb1
        MOVLW  0x48
        movwf  TRISB       ;пересылка из W в порт B
        bsf    PORTA,2     ; RS=1
        bsf    PORTA,3             ; E=1 (разрешение)
        bcf    PORTA,3     ; E=0
        call   Pause1

        MOVLW  0x45
        movwf  TRISB       ;пересылка из W в порт B
        bsf    PORTA,2     ; RS=1
        bsf    PORTA,3             ; E=1 (разрешение)
        bcf    PORTA,3     ; E=0
        call   Pause1
        MOVLW  0x4c
        movwf  TRISB       ;пересылка из W в порт B
        bsf    PORTA,2     ; RS=1
        bsf    PORTA,3             ; E=1 (разрешение)
        bcf    PORTA,3     ; E=0
        call   Pause1
        MOVLW  0x4c
        movwf  TRISB       ;пересылка из W в порт B
        bsf    PORTA,2     ; RS=1
        bsf    PORTA,3             ; E=1 (разрешение)
        bcf    PORTA,3     ; E=0
        call   Pause1
        MOVLW  0x4f
        movwf  TRISB       ;пересылка из W в порт B
        bsf    PORTA,2     ; RS=1
        bsf    PORTA,3             ; E=1 (разрешение)
        bcf    PORTA,3     ; E=0
        call   Pause1
```

```
          MOVLW  0x00
          movwf  TRISB       ;пересылка из W в порт B
          bsf    PORTA,2     ; RS=1
          bsf    PORTA,3             ; E=1 (разрешение)
          bcf    PORTA,3     ; E=0
          call   Pause1
          MOVLW  0x30
          movwf  TRISB       ;пересылка из W в порт B
          bsf    PORTA,2     ; RS=1
          bsf    PORTA,3             ; E=1 (разрешение)
          bcf    PORTA,3     ; E=0
          call   Pause1
          MOVLW  0x34
          movwf  TRISB       ;пересылка из W в порт B
          bsf    PORTA,2     ; RS=1
          bsf    PORTA,3             ; E=1 (разрешение)
          bcf    PORTA,3     ; E=0
          call   Pause1
          MOVLW  0x31
          movwf  TRISB       ;пересылка из W в порт B
          bsf    PORTA,2     ; RS=1
          bsf    PORTA,3             ; E=1 (разрешение)
          bcf    PORTA,3     ; E=0
          call   Pause1
          goto   opros
Pause1    movlw     .221
          movwf     Reg_1
          movlw     .130
          movwf     Reg_3
wr1       decfsz    Reg_1, F
          goto      wr1
          decfsz    Reg_3, F
          goto      wr1
          nop
```

```
nop
return
  end
```

РАБОТА № 7

РАБОТА С МОДУЛЕМ АЦП

Цель работы: разобраться с внутренним устройством, принципом работы и настройки аналого-цифрового преобразователя (АЦП).

Краткие теоретические сведения

Что такое АЦП?

Аналого-цифровой преобразователь (АЦП, англ. Analog-to-digital converter, ADC) - устройство, преобразующее входной аналоговый сигнал в дискретный код (цифровой сигнал). Обратное преобразование осуществляется при помощи ЦАП (цифро-аналогового преобразователя, DAC). Как правило, АЦП - электронное устройство, преобразующее напряжение в двоичный цифровой код. Тем не менее, некоторые неэлектронные устройства с цифровым выходом, следует также относить к АЦП, например, некоторые типы преобразователей угол-код. Простейшим одноразрядным двоичным АЦП является компаратор.

Разрядность АЦП

Разрядность АЦП характеризует количество дискретных значений, которые преобразователь может выдать на выходе. В двоичных АЦП разрядность измеряется в битах. Разрядностью АЦП определяется и его разрешение – минимальное изменение величины входного аналогового сигнала, которое может быть зафиксировано данным АЦП. АЦП преобразовывает сигнал (напряжение) находящийся в диапазоне измеряемых сигналов. Нижняя и верхняя граница этого диапазона определяется напряжениями, поданными на соответствующие выводы.

Для МК со встроенным АЦП, нижняя граница – это уровень GND (0 В), а верхняя – подается на отдельный вывод (AREF) или используются внутренние источники опорных напряжений. При диапазоне входных

напряжений от 0 В до 5 В и использовании 10-битного АЦП мы имеем следующее разрешение АЦП Δ (см. Рисунок 1).

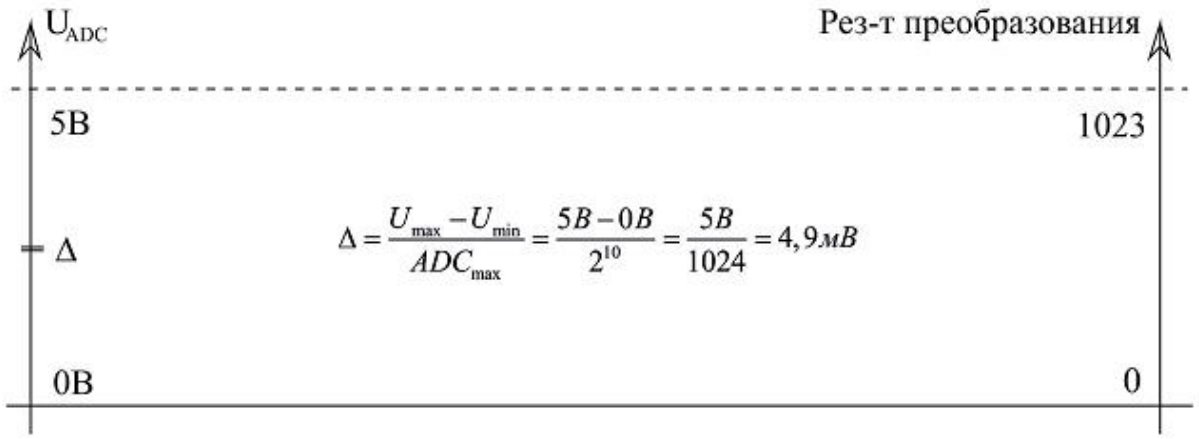


Рисунок 1 – Разрешение 10-битного АЦП

Для того, чтобы компьютер мог выполнить обработку сигнала необходимо выполнить преобразование сигнала из аналоговой формы в цифровую. После обработки выполняется обратное преобразование, поскольку большинство бытовых устройств управляются аналоговыми сигналами. Структурная схема цифровой обработки сигнала в общем виде выглядит следующим образом (рисунок 2):

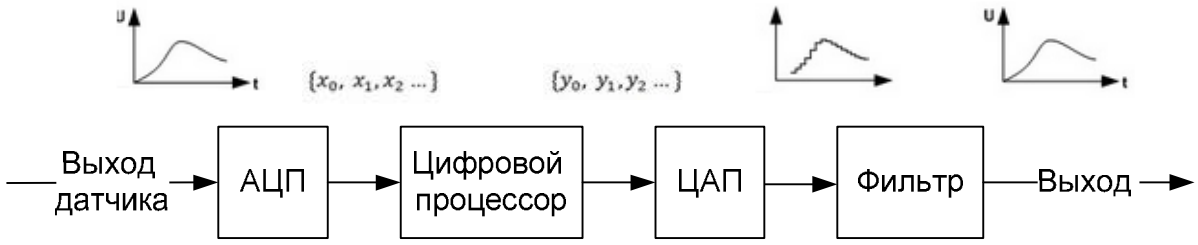


Рисунок 2 - Аналого-цифровое преобразование сигнала

Аналого-цифровое преобразование сигнала включает в себя два этапа:

1. Дискретизация сигнала (во времени или пространстве)
2. Квантование по уровню

На этапе дискретизации берутся отсчёты сигнала с некоторым периодом дискретизации (T). Частоту дискретизации можно определить по формуле. Процесс получения отсчёта входного сигнала должен занимать

очень малую часть периода дискретизации, чтобы снизить динамические ошибки преобразования, обусловленные изменением сигнала за время снятия отсчёта.

Применение АЦП

Аналого-цифровое преобразование используется везде, где требуется обрабатывать, хранить или передавать сигнал в цифровой форме:

• АЦП являются составной частью систем сбора данных;

• быстрые видео АЦП используются, например, в ТВ-тюнерах;

• медленные встроенные 8, 10, 12 или 16-битные АЦП часто входят в состав микроконтроллеров;

• очень быстрые АЦП необходимы в цифровых осциллографах;

• современные весы используют АЦП с разрядностью до 24 бит, преобразующие сигнал непосредственно от тензометрического датчика;

• АЦП входят в состав радиомодемов и других устройств радиопередачи данных, где используются совместно с процессором цифровой обработки сигналов в качестве демодулятора;

• сверхбыстрые АЦП используются в антенных системах базовых станций (в так называемых SMART-антеннах) и в антенных решетках радиолокационных станций.

Принцип работы

Принцип работы АЦП заключается в зарядке внутреннего конденсатора C_{HOLD} и преобразование в соответствующий 10 разрядный цифровой код методом последовательного приближения. Результат преобразования сохраняется в регистрах ADRESH, ADRESL (старший и младший регистр). Хотя каналов и несколько, но преобразование производится через коммутатор (некий переключатель каналов), соответственно в момент времени может преобразоваться только один канал и по завершению обработки результата можно приступать к следующему (рисунок 3).

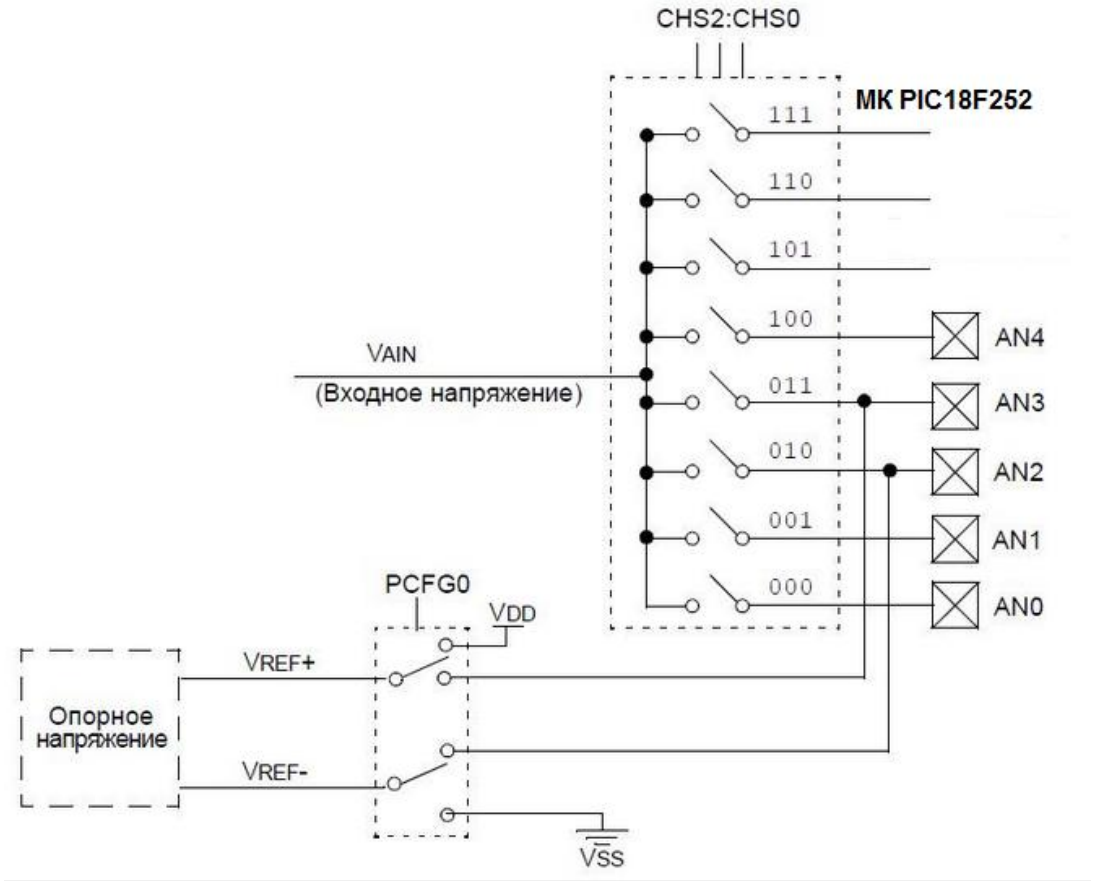


Рисунок 3 – АЦП последовательного приближения

Для настройки модуля аналого-цифрового преобразования в МК PIC18xx2 предусмотрено два регистра специального назначения: ADCON0 и ADCON1.

Примечание: в других микроконтроллерах семейства PIC18XXXX регистров управление может быть больше, это зависит от количества каналов АЦП. К примеру, в МК PIC18F25X20 12 каналов и соответственно для их управления добавлен регистр ADCON2. По аналогии легко понять его назначение.

Настройка модуля АЦП:

• Настроить выводы как аналоговые входы, входы
VREF или цифровые каналы ввода/вывода (ADCON1)

• Выбрать входной канал АЦП (ADCON0)

• Выбрать источник тактовых импульсов для АЦП (ADCON0)

• Включить модуль АЦП (ADCON0)

Для начала процесса аналого-цифрового преобразования (далее АЦП) выбираем в регистре ADCON1 линию порта (или несколько линий), на которую будем подавать измеряемый сигнал и выбираем опорное напряжение, оно может быть, как внутренним (напряжение питания микроконтроллера, выводы Vcc, Vdd), так и внешним, подключенным к выводам порта «А» (AN3/Vref+, AN2/Vref-). Выбор производится битами 3-0 **PCFG3:PCFG0**: При этом, если выводы настраиваются как аналоговые входы, то необходимо установить соответствующие биты регистров TRIS..,Т.е. настроить на вход. Включаем модуль АЦП установкой бита **ADON** регистра ADCON0.

Порядок работы

Задание: разработать и отладить программу с использованием модуля АЦП в среде программирования MPLAB на языке ассемблер.

Приведем схему, демонстрирующую работу модуля АЦП, собранную в протеусе (рисунок 4). Программный код реализован в среде MPLAB, и приведен в приложении А.

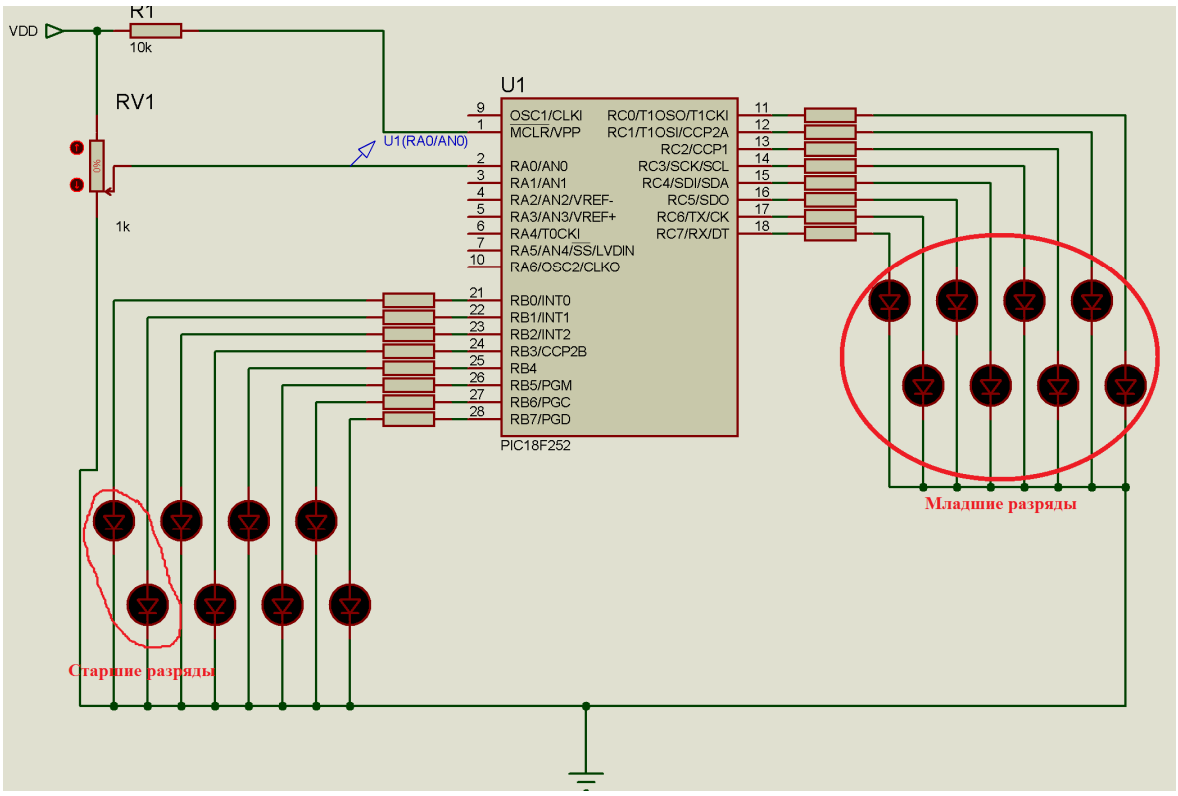


Рисунок 4 – Схема работы АЦП

На схеме на каждый порт приведено по 8 диодов, это сделано для наглядности, чтобы можно было менять выравнивание. В этой лабораторной работе используется правое выравнивание. Значит, порт B отвечает за старшие разряды, а порт C – за младшие (рисунок 4).

Далее запустим проект, при увеличении напряжения видим, что диоды определенным образом загораются, а именно (рисунок 5, 6).

Это можно проследить при каждом увеличении напряжения на потенциометре, а именно 0В соответствует 0 в двоичном коде, а 4,9 В – 1023(рисунок 1). 10-разрядный двоичный код образуют светодиоды, где два светодиода – старшие разряды (порт B), младшие разряды – порт C.

На рисунке 5 число $10_{10} – 1010_2$, что говорит о том, что 1% от 4.9 В это 0,24 В.

На рисунке 6 число $358_{10} – 101100110_2$, это говорит о том, что 35% от 4.9 В, это 1,715 В.

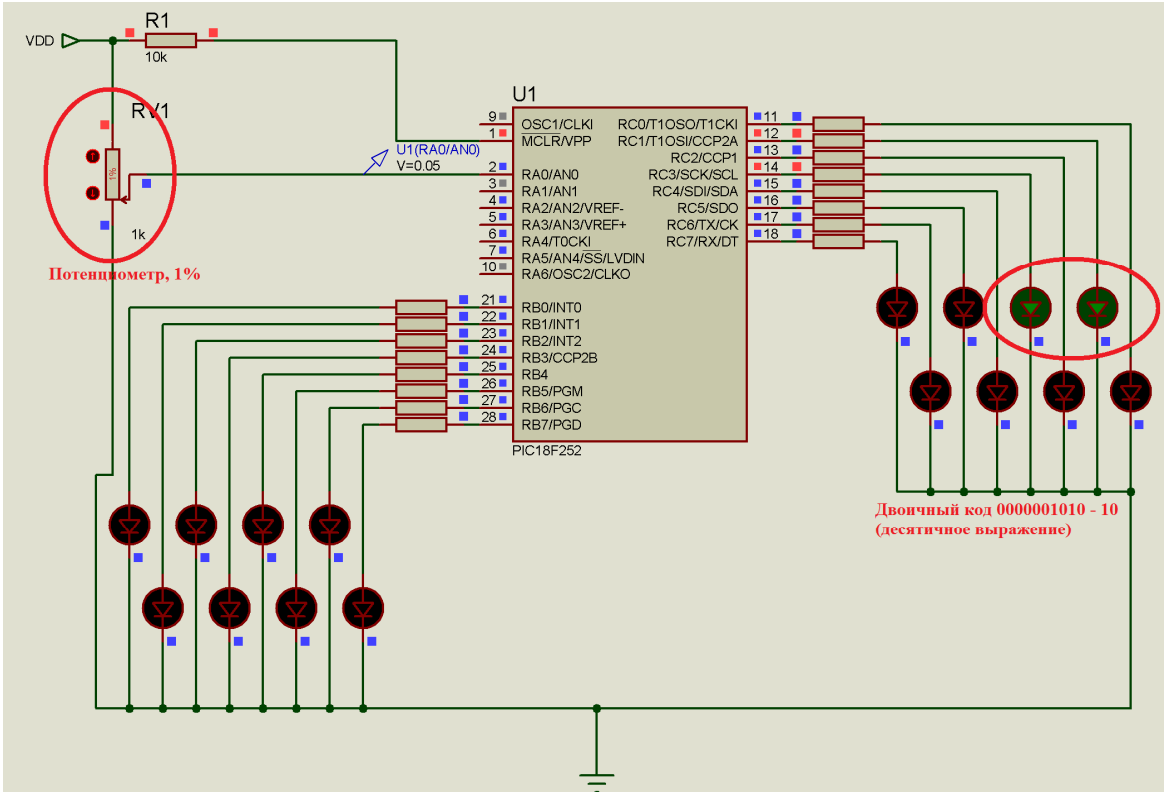


Рисунок 5 – Схема работы АЦП

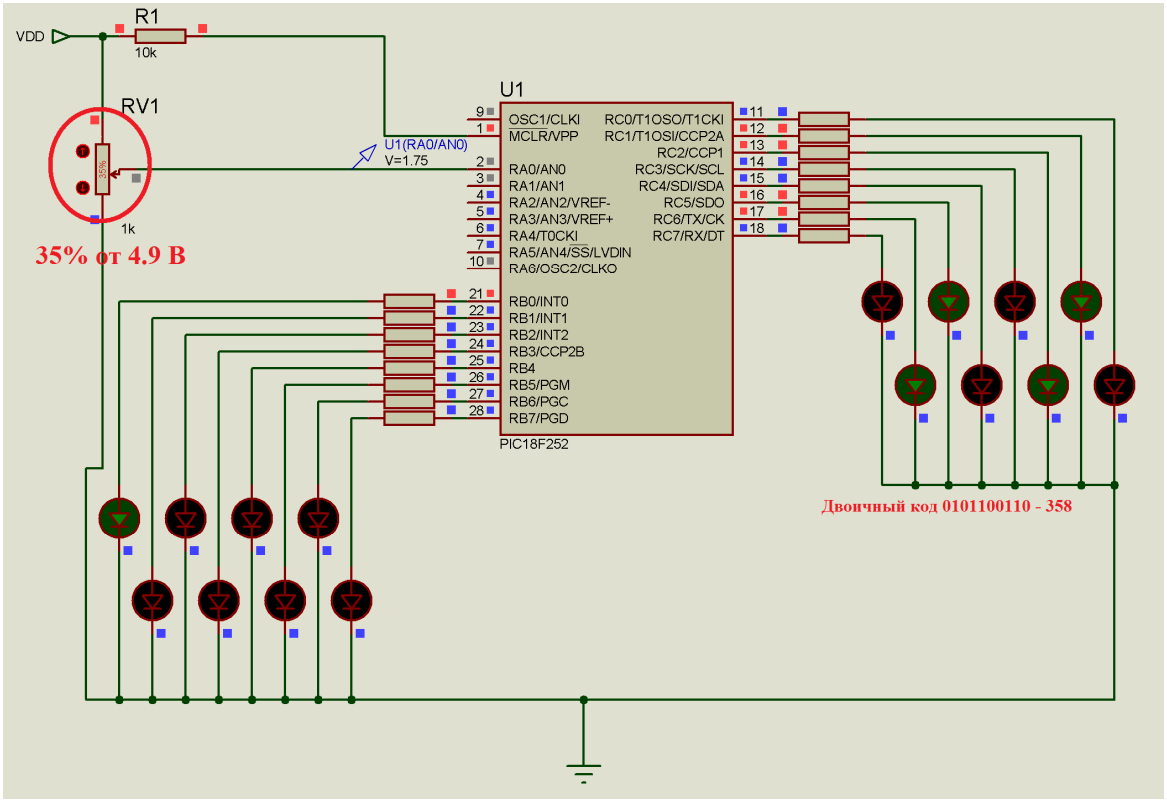


Рисунок 6 – Схема работы АЦП

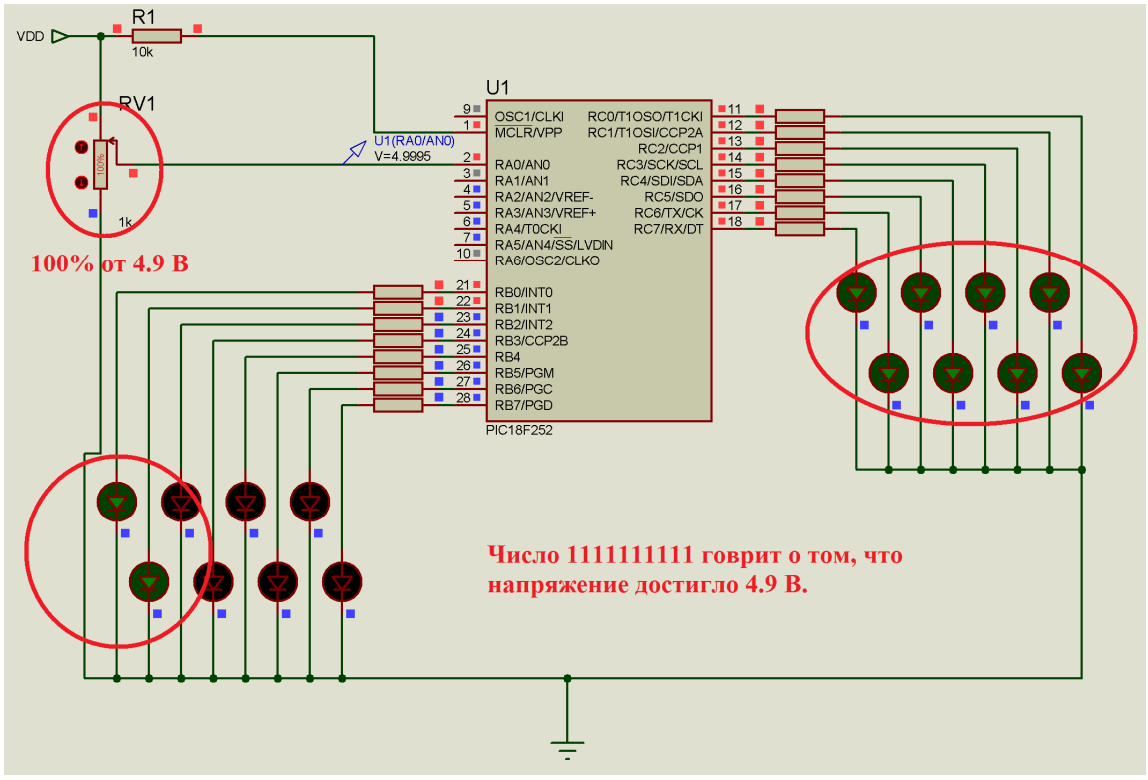


Рисунок 7 - Схема работы АЦП

На рисунке 7 видно, что на потенциометре выставили 100 % напряжения, что говорит о том, что оно достигло максимального, т е 4.9 В,

диоды, отслеживающие динамику изменения напряжения выдают двоичный код 1111111111_2 , что соответствует числу $1023_{10.}$

Задания для самостоятельного выполнения

1. Ввести в схему дешифратор и организовать вывод информации на семисегментый дисплей в виде десятых от процентов.
2. Изменить работу схемы так, что бы при увеличении напряжения на входе АЦП горел сначала один диод, потом два и т.д.
3. Изменить задание так, чтобы на АЦП высчитывалась разница от двух сигналов.
4. Изменить направление вывода информации (сделать так что бы в левой группе светодиодов выводились младшие разряды, а в правой - старшие).
5. Проработать вопрос изменения разрядности АЦП (по вариантам).
6. Изменить структуру системы и программу так, чтобы система работала с контроллером без встроенного АЦП (ввести дополнительный модуль внешнего АЦП).

```
 list        p=18F252 ; Используется микроконтроллер PIC18F252.
#include   p18F252.inc ; Подключение INC-файла PIC18F252.
; Конфигурирование.
        CONFIG      OSC=HSPLL ;  HS+PLL.
        CONFIG       BOR=OFF ; Сброс по снижению питания выключен
        CONFIG       WDT=OFF ; WDT выключен.
        CONFIG       LVP=OFF ; Режим низковольтного програм. выкл.
;   назначение
        CBLOCK
        Reg_1
        Reg_2
        Reg_3
        ENDC
        org        0x000
        goto       START
START movlw       b'00000011'; A0,A1 на вход, остальные на выход
        movwf          TRISA ;
        clrf           TRISB ;
        clrf           TRISC ;
        movlw      b'11001110'       ;  правое выравнивание,FOSC/64,
        movwf          ADCON1        ; A0-аналоговые входы,=
        movlw      b'10000000'       ; FOSC/64,выбран канал A0,
        movwf          ADCON0
Povt    bsf       ADCON0,ADON      ; включить модуль АЦП
        call         Pause_14           ; пауза для зарядки Hold
        bsf          ADCON0,2           ; начать преобразование
        btfsc        ADCON0,2 ;
        bra             $-.2              ; ожидать окончания преобр.
        movff     ADRESH,PORTB ; результат старший в PORTB
        movff     ADRESL,PORTC ; результат младший в PORTC
        bcf       ADCON0,ADON   ; выключить модуль АЦП
;-----------------------------------
 ; если есть необходимость в следующем цикле
 ; АЦП. то необходимо выдержать паузу в данном случае 4 мкс
```

```
;-----------------------------------
        call    Pause_4 ; необходимая задержка для следующего цикла.
        bra     Povt ; следующий цикл
;-----------------------------------
Pause_14    movlw   .46
        movwf   Reg_1
        decfsz  Reg_1
        bra     $-2
        return
;-----------------------------------
Pause_4     movlw   .13
        movwf   Reg_1
        decfsz  Reg_1
        bra     $-2
        return
end
```

РАБОТА № 8

ИЗУЧЕНИЕ МОДУЛЯ USART

Цель работы:

1. Изучение принципа работы модуля последовательного ввода вывода – USART.
2. Выполнение простейших программ, связанных с передачей данных между микроконтроллерами.

Краткие теоретические сведения

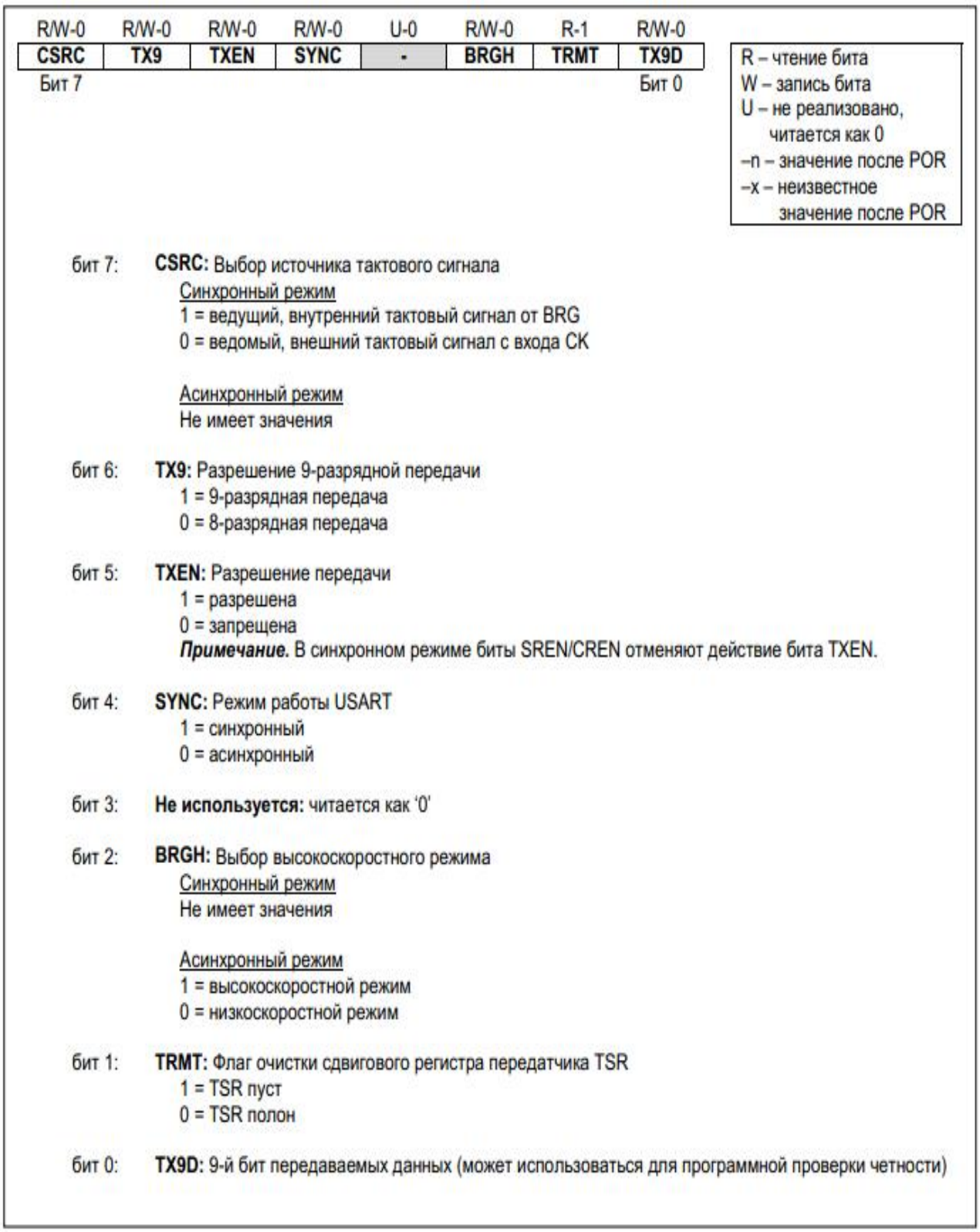
1. Универсальный синхронно – асинхронный приемопередатчик (USART)

USART - это модуль последовательного ввода/вывода, который может работать в полнодуплексном асинхронном режиме для связи с терминалами, персональными компьютерами, или синхронном полудуплексном режиме для связи с микросхемами ЦАП, АЦП, последовательными EEPROM и т.д. USART может работать в трех режимах:

- Асинхронный, полный дуплекс:
- Ведущий синхронный, полудуплекс;
- Ведомый синхронный, полудуплекс.

Биты SPEN (RCSTA<7>) и TRISC<7:6> должны быть установлены в ‘1’ для использования выводов RC6/TX/CK и RC7/RX/DT в качестве портов универсального синхронно-асинхронного приемопередатчика. Модуль USART поддерживает режим детектирования 9-разрядного адреса для работы в сетевом режиме.

На рисунке 1 и рисунке 2 показаны регистры управления и статуса приемника и передатчика.



R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	**TX9**	**TXEN**	**SYNC**	**-**	**BRGH**	**TRMT**	**TX9D**
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

бит 7: **CSRC:** Выбор источника тактового сигнала
Синхронный режим
1 = ведущий, внутренний тактовый сигнал от BRG
0 = ведомый, внешний тактовый сигнал с входа CK

Асинхронный режим
Не имеет значения

бит 6: **TX9:** Разрешение 9-разрядной передачи
1 = 9-разрядная передача
0 = 8-разрядная передача

бит 5: **TXEN:** Разрешение передачи
1 = разрешена
0 = запрещена
Примечание. В синхронном режиме биты SREN/CREN отменяют действие бита TXEN.

бит 4: **SYNC:** Режим работы USART
1 = синхронный
0 = асинхронный

бит 3: **Не используется:** читается как '0'

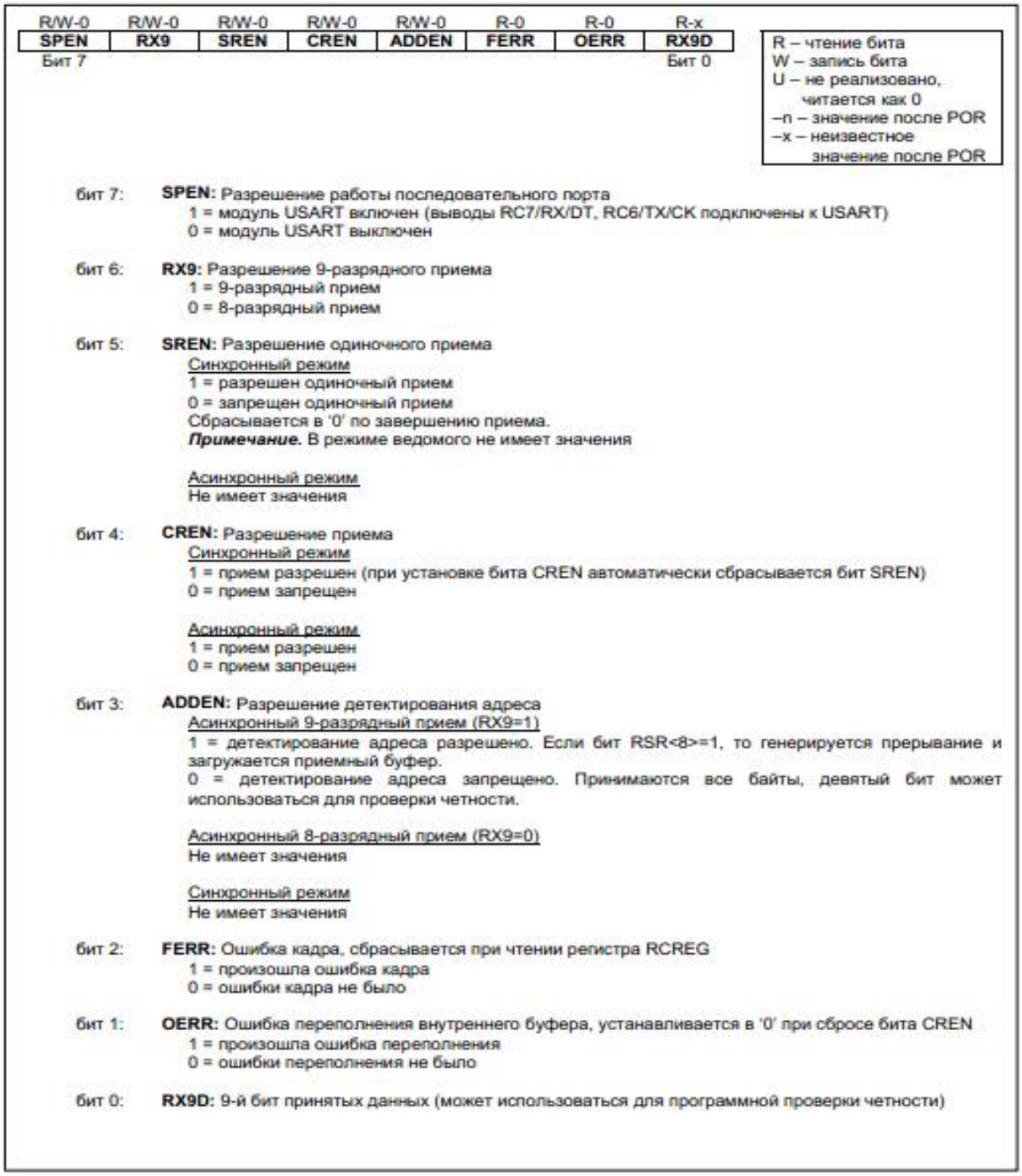
бит 2: **BRGH:** Выбор высокоскоростного режима
Синхронный режим
Не имеет значения

Асинхронный режим
1 = высокоскоростной режим
0 = низкоскоростной режим

бит 1: **TRMT:** Флаг очистки сдвигового регистра передатчика TSR
1 = TSR пуст
0 = TSR полон

бит 0: **TX9D:** 9-й бит передаваемых данных (может использоваться для программной проверки четности)

Рисунок 1 - TXSTA (адрес 98h) Регистр управления и статуса передатчика



R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	**RX9**	**SREN**	**CREN**	**ADDEN**	**FERR**	**OERR**	**RX9D**
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

бит 7: **SPEN:** Разрешение работы последовательного порта
1 = модуль USART включен (выводы RC7/RX/DT, RC6/TX/CK подключены к USART)
0 = модуль USART выключен

бит 6: **RX9:** Разрешение 9-разрядного приема
1 = 9-разрядный прием
0 = 8-разрядный прием

бит 5: **SREN:** Разрешение одиночного приема
Синхронный режим
1 = разрешен одиночный прием
0 = запрещен одиночный прием
Сбрасывается в '0' по завершению приема.
Примечание. В режиме ведомого не имеет значения

Асинхронный режим
Не имеет значения

бит 4: **CREN:** Разрешение приема
Синхронный режим
1 = прием разрешен (при установке бита CREN автоматически сбрасывается бит SREN)
0 = прием запрещен

Асинхронный режим
1 = прием разрешен
0 = прием запрещен

бит 3: **ADDEN:** Разрешение детектирования адреса
Асинхронный 9-разрядный прием (RX9=1)
1 = детектирование адреса разрешено. Если бит RSR<8>=1, то генерируется прерывание и загружается приемный буфер.
0 = детектирование адреса запрещено. Принимаются все байты, девятый бит может использоваться для проверки четности.

Асинхронный 8-разрядный прием (RX9=0)
Не имеет значения

Синхронный режим
Не имеет значения

бит 2: **FERR:** Ошибка кадра, сбрасывается при чтении регистра RCREG
1 = произошла ошибка кадра
0 = ошибки кадра не было

бит 1: **OERR:** Ошибка переполнения внутреннего буфера, устанавливается в '0' при сбросе бита CREN
1 = произошла ошибка переполнения
0 = ошибки переполнения не было

бит 0: **RX9D:** 9-й бит принятых данных (может использоваться для программной проверки четности)

Рисунок 2 - RSTA (адрес 18h) Регистр управления и статуса приемника

2. Генератор частоты обмена USART BRG

BRG используется для работы USART в синхронном ведущем и асинхронном режимах. BRG представляет собой отдельный 8-разрядный генератор скорости обмена в бодах, период которого определяется значением в регистре SPBRG. В асинхронном режиме бит BRGH (TXSTA<2>) тоже

влияет на скорость обмена (в синхронном режиме бит BRGH игнорируется). В таблице 1 указаны формулы для вычисления скорости обмена в бодах при различных режимах работы модуля USART (относительно внутреннего тактового сигнала микроконтроллера).

Учитывая требуемую скорость и Fosc, выбирается самое близкое целое значение для записи в регистр SPBRG, рассчитанное по формулам, приведенным в таблице 1. Затем рассчитывается ошибка скорости обмена.

Таблица 1 Формулы расчета скорости обмена данными

SYNC	BRGH = 0	BRGH = 1
0	(Асинхронный) Скорость обмена = Fosc / (64 (X + 1))	(Асинхронный) Скорость обмена = Fosc / (16 (X + 1))
1	(Синхронный) Скорость обмена = Fosc / (4 (X + 1))	(Синхронный) Скорость обмена = Fosc / (4 (X + 1))

X = значение регистра SPBRG (от 0 до 255)

3. Асинхронный режим USART

В этом режиме USART использует стандартный формат NRZ: один стартовый бит, восемь или девять битов данных и один стоповый бит. Наиболее часто встречается 8-разрядный формат передачи данных. Интегрированный 8-разрядный генератор BRG позволяет получить стандартные скорости передачи данных. Генератор скорости обмена может работать в одном из двух режимов: высокоскоростной (x16 BRGH= 1 TXSTA<2>), низкоскоростной (x64 BRGH=0 TXSTA<2>). Приемник и передатчик последовательного порта работают независимо друг от друга, но используют один и тот же формат данных и одинаковую скорость обмена. Бит четности аппаратно не поддерживается, но может быть реализован программно, применяя 9-разрядный формат данных. Все данные передаются младшим битом вперед. В SLEEP режиме микроконтроллера модуль USART

(асинхронный режим) выключен. Выбор асинхронного режима USART выполняется сбросом бита SYNC в 0 (TXSTA<4>).

Модуль USART в асинхронном режиме состоит из следующих элементов.

• Генератор скорости обмена;

• Цепь опроса;

• Асинхронный передатчик;

• Асинхронный приемник.

4. Асинхронный передатчик USART

Структурная схема асинхронного передатчика USART показана на рисунке 3. Главным в передатчике является сдвиговый регистр TSR, который получает данные из буфера передатчика TXREG. Данные в регистр TXREG загружаются программно. После передачи стопового бита предыдущего байта, в последнем машинном такте цикла BRG, TSR загружается новым значением из TXREG (если оно присутствует), после чего устанавливается флаг прерывания TXIF (PIR1<4>). Прерывание может быть разрешено или запрещено битом TXIE (P1E1<4>). Флаг TX1F устанавливается независимо от состояния бита TXIE и не может быть сброшен в 0 программно. Очистка флага TX1F происходит только после загрузки новых данных в регистр TXREG. Аналогичным образом бит TRMT (TXSTA<1>) отображает состояние регистра TSR. Бит TRMT доступен только на чтение и не может вызвать генерацию прерывания.

Примечания:

1. Регистр TSR не отображается на память и не доступен для чтения.

2. Флаг TXIF устанавливается в 1 только, когда бит TXEN=1 и сбрасывается автоматически в 0 после загрузки новых данных в регистр TXREG.

Для разрешения передачи необходимо установить бит TXEN (TXSTA<5>) в 1. Передача данных не начнется до тех пор, пока в TXREG не будут загружены новые данные: не придет очередной тактовый импульс от

генератора BRG (см рисунок 4). Можно сначала загрузить данные в TXREG. а затем установить бит TXEN. Как правило, после разрешения передачи регистр TSR пуст, таким образом, данные записываемые в TXREG сразу передаются в TSR, а TXREG остается пустым. Это позволяет реализовать слитную передачу данных. Сброс бита TXEN в '0' вызовет немедленное прекращение передачи, сброс передатчика и перевод вывода RC6/TX/CK в третье состояние.

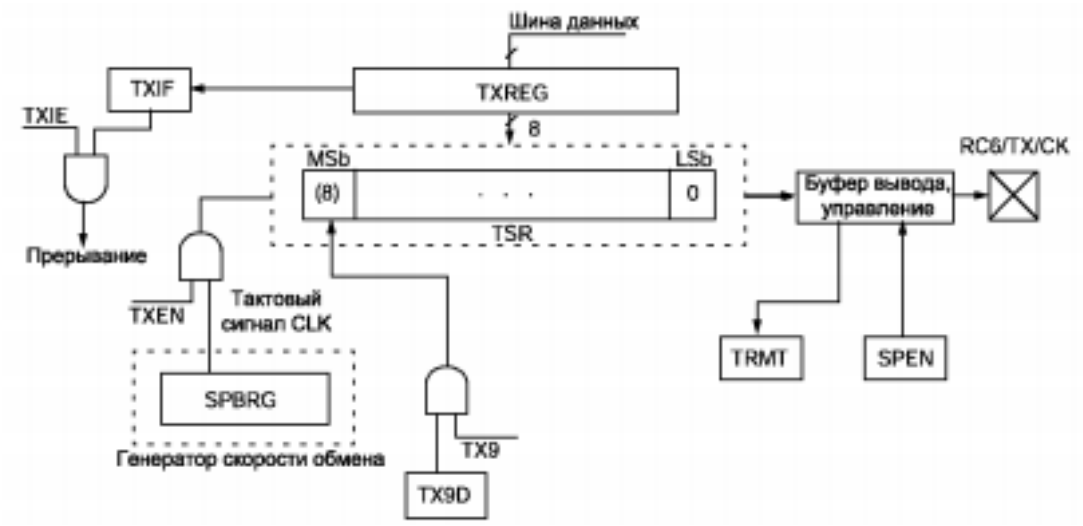


Рисунок 3 – Структурная схема асинхронного передатчика USART

Рекомендованная последовательность действий для передачи данных в асинхронном режиме:

1. Установить требуемую скорость передачи с помощью регистра SPBRG и бита BRGH.
2. Выбрать асинхронный режим сбросом бита SYNC в 0 и установкой бита SPEN в 1.
3. Если необходимо, разрешить прерывания установкой бита TXIE в 1.
4. Если передача 9-разрядная. установить бит TX9 в 1.
5. Разрешить передачу установкой бита TXEN в 1, автоматически устанавливается флаг TXIF.
6. Если передача 9-разрядная, записать 9-й бит данных в TX9D.
7. Записать данные в регистр TXREG.

8. Если используются прерывания, то биты GIE и PEIE в регистре INTCON должны быть установлены в 1.

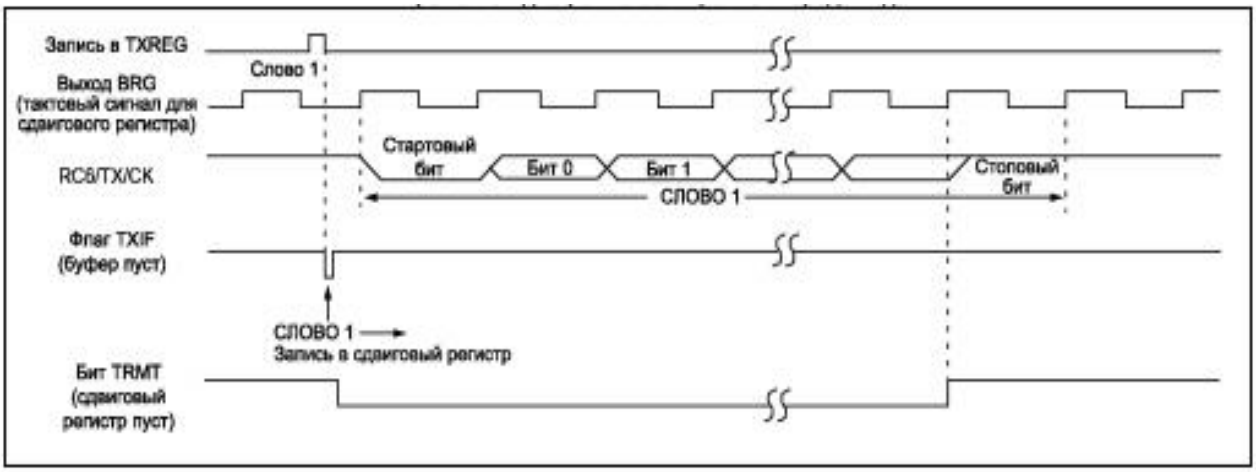


Рисунок 4 – временная диаграмма асинхронной передачи данных

5. Асинхронный приемник USART

Структурная схема асинхронного приемника USART показана на рисунке 5. Данные подаются на вход RC7/RX/DT в блок восстановления данных, представляющий собой скоростной сдвиговый регистр, работающий на частоте в 16 раз превышающей скорость передачи или F_{osc}.

Включение приемника производится установкой бита CREN регистра RCSTA в 1.

Главным в приемнике является сдвиговый регистр RSR. После получения стопового бита данные переписываются в регистр RCREG. если он пуст. После записи в регистр RCREG выставляется флаг прерывания RCIF (PIR1<5>). Прерывание можно разрешить/запретить установкой/сбросом бита RCIE (P1E1<5>). Флаг RCIF доступен только на чтение, сбрасывается аппаратно при чтении из регистра RCREG. Регистр RCREG имеет двойную буферизацию, т.е. представляет собой двухуровневый буфер FIFO. Поэтому можно принять 2 байта данных в FIFO RCREG и третий в регистр RSR. Если FIFO заполнен и обнаружен стоповый бит третьего байта, устанавливается бит переполнения приемника OERR (RCSTA<1>). Байт, принятый в RSR. будет потерян. Для извлечения двух байт из FIFO необходимо дважды

прочитать регистр RCREG. Бит OERR нужно программно очистить сбросом бита CREN. т.е. запрещением приема. В любом случае, если бит OERR установлен, логика приемника выключена.

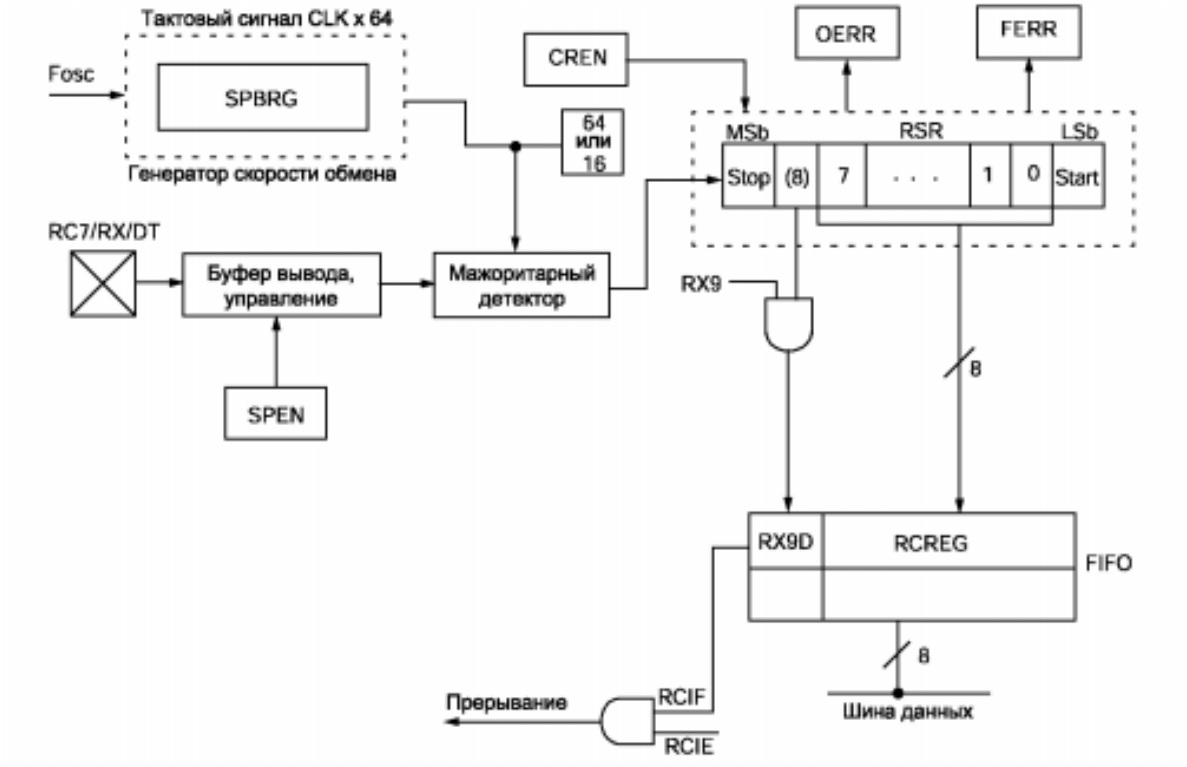


Рисунок 5 – Структурная схема асинхронного приемника USART

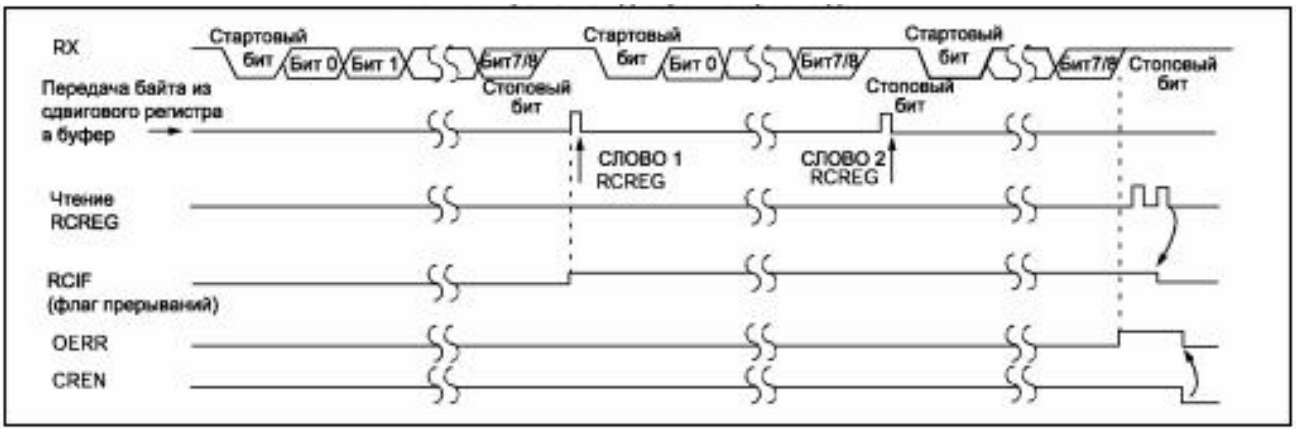


Рисунок 6 – Временная диаграмма приема данных

Примечание:

На временной диаграмме показан последовательный прием трех байт. Регистр RCREG (приемный буфер) читается после приема трех байт, поэтому устанавливается бит OERR в 1.

Рекомендованные действия при приеме данных в асинхронном режиме:

1. Установить требуемую скорость передачи с помощью регистра SPBRG и бита BRGH.

2. Выбрать асинхронный режим сбросом бита SYNC в 0 и установкой бита SPEN в 1.

3. Если необходимо, разрешить прерывания установкой бита RCIE в 1.

4. Если прием 9-разрядный. установить бит RX9 в 1.

5. Разрешить прием установкой бита CREN в 1.

6. Ожидать установку бита RCIF. или прерывание, если оно разрешено битом RCIE.

7. Считать 9-й бит данных (если разрешен 9-разрядный прием) из регистра RCSTA и проверить возникновение ошибки.

8. Считать 8 бит данных из регистра RCREG.

9. При возникновении ошибки переполнения сбросить бит CREN в 0'.

10. Если используются прерывания, то биты GIE и PEIE в регистре INTCON должны быть установлены в 1.

Порядок работы

Задание: На основе микроконтроллера разработать устройство, выполняющее следующие функции:

Передача информации о номере нажатой кнопки. Устройство реализовать на 2 микроконтроллерах, связывающихся между собой. Первый микроконтроллер определяет номер нажатой кнопки (1, 2 или 3) и передает его второму микроконтроллеру, который в свою очередь выводит номер нажатой кнопки на семисегментный дисплей.

Схема, собранная в программе – эмуляторе PROTEUS показана на рисунке 7.

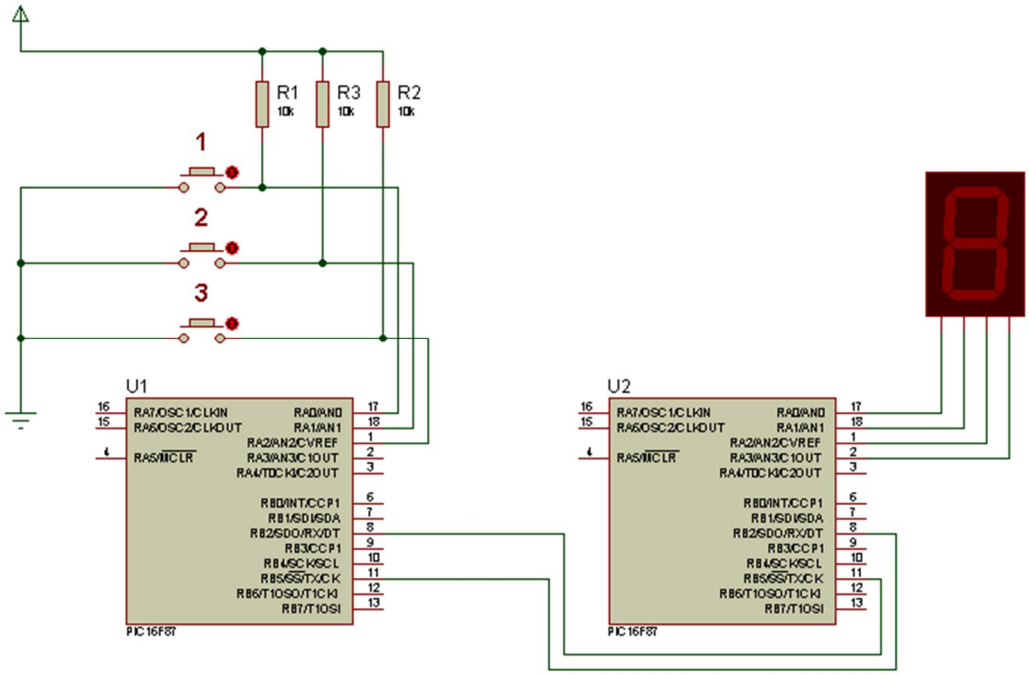


Рисунок 7 – Схема в программе – эмуляторе PROTEUS

В качестве портов универсального синхронно-асинхронного приемопередатчика используются выводы RC6/TX/CK и RC7/RX/DT

Разработка полного алгоритма программы

На рисунке 8 представлен алгоритм работы программы.

Представленный алгоритм программы показывает, каким образом работает программа. Составление алгоритма помогает в дальнейшем написании программного кода и в представлении законченного рабочего устройства.

На рисунке 9 показана подпрограмма «передача», используемая в программном коде для МК1.

Здесь байт, который необходимо передать, предварительно должен быть записан в регистр Reg_1.

На рисунке 10 показана подпрограмма «прием», используемая в программном коде для МК2.

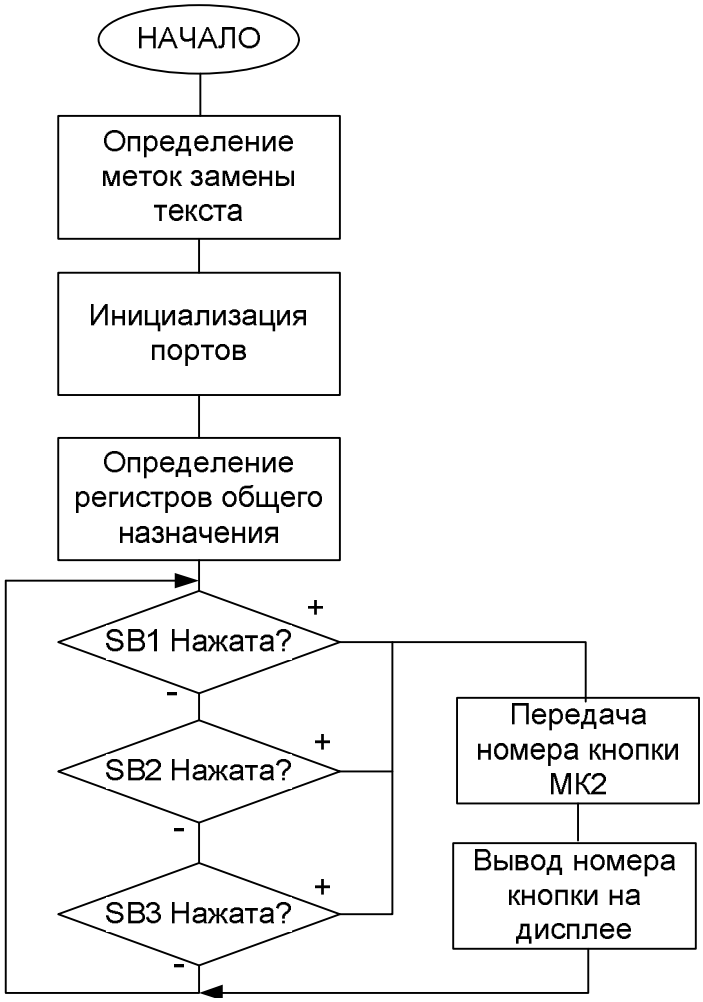


Рисунок 8 – алгоритм работы программы.

```
;=================================================
            ;ПОДПРОГРАММА ПЕРЕДАЧА
;=================================================
peredacha
            bsf         STATUS,5
            BSF         TXSTA,2
            movlw       0x01
            movwf       SPBRG
            movlw       b'00100000'
            movwf       TXSTA
            bcf         STATUS,5
            movlw       b'10010000'
            movwf       RCSTA
PUTCHAR
            btfss       PIR1,4
            goto        PUTCHAR
            movf        Reg_1,W
            movwf       TXREG
            bsf         STATUS,5
M1
            BTFSS       TXSTA,1
            GOTO        M1
            bcf         STATUS,5
            return
            end
```

Рисунок 9 – Подпрограмма передача

```
;=========================================
            ;ПОДПРОГРАММА ПРИЕМ
;=========================================
priem
            bcf         STATUS,5
            bcf         RCSTA,0
            bsf         STATUS,5
            bsf         PIE1,5
            BSF         TXSTA,2
            movlw       0x01
            movwf       SPBRG
            movlw       b'00000000'
            movwf       TXSTA
            bcf         STATUS,5
            movlw       b'10010000'
            movwf       RCSTA
GETCHAR
            btfss       PIR1,5
            goto        GETCHAR
            movf        RCREG,w
            movwf       Reg_1
            return
            end
```

Рисунок 10 – Подпрограмма прием

Байт, принятый в данной подпрограмме, сохраняется в регистре Reg_1. Подпрограммы составлены аналогично пунктам 4 и 5 краткой теории. Полный текст программных кодов предоставлен в приложении А.

Задания для самостоятельного выполнения

Использовать другой интерфейс.

Выполнить то же задания, но с передачей информации как с одного, так и с другого МК.

Добавить еще один МК.

В первый МК записываются два числа, второй выводит их сумму.

Организовать подтверждение передачи.

Организовать запоминание первым МК номеров нажатых кнопок и их последовательное отображение на дисплее.

Программный код для МК1:

```
        LIST       P=PIC16F87, R=HEX


STATUS     EQU        H'0003'
PORTA      EQU        H'0005'
PORTB      EQU        H'0006'
TRISB      EQU        H'0006'
TRISA      EQU        H'0005'
Reg_1      EQU                H'0027'
Reg_2      EQU                H'0028'
RP0        EQU        H'0005'
Z          EQU        2
C          EQU        0
RCSTA      EQU        H'0018'
TXSTA      EQU        H'0098'
TXREG      EQU        H'0019'
RCREG      EQU        H'001A'
SPBRG      equ        H'0099'
PIR1       EQU                H'000C'
PIR2       EQU                H'000D'
PIE1       EQU        H'008C'
                                org        0
;=================================================================
        ;НАСТРОЙКА ВЫВОДОВ
;=================================================================
                                bsf        STATUS,5
            MOVLW      0xFF
                                MOVWF      TRISA

                                MOVLW      0X04
                                MOVWF      TRISB
                                bcf        STATUS,5
;=================================================================
```

```
        ;ОСНОВНАЯ ПРОГРАММА
;=============================================================
m1
                                BTFSC           PORTA,0
                                GOTO            m2
                                movlw           b'00001000'
                                movwf           Reg_1
                                call            peredacha

m2
                                BTFSC           PORTA,1
                                GOTO            m3
                                movlw           b'00000100'
                                movwf           Reg_1
                                call            peredacha
m3
                                BTFSC           PORTA,2
                                GOTO            m1
                                movlw           b'00001100'
                                movwf           Reg_1
                                call            peredacha
                                goto            m1
;=============================================================
        ;ПОДПРОГРАММА ПЕРЕДАЧА
;=============================================================
peredacha
                                bsf     STATUS,5
                            BSF             TXSTA,2
                                movlw       0x01
                                movwf       SPBRG
        movlw   b'00100000'
                                movwf       TXSTA
                                bcf         STATUS,5
                                movlw       b'10010000'
                                movwf       RCSTA
```

```
PUTCHAR
                    btfss   PIR1,4
                    goto    PUTCHAR
                    movf    Reg_1,W
                    movwf   TXREG
                    bsf     STATUS,5
M1
                    BTFSS   TXSTA,1
                    GOTO    M1
                    bcf     STATUS,5
                    return
                    end
```

Программный код для МК2:

```
        LIST    P=PIC16F87, R=HEX


STATUS  EQU     H'0003'
PORTA   EQU     H'0005'
PORTB   EQU     H'0006'
TRISB   EQU     H'0006'
TRISA   EQU     H'0005'
Reg_1   EQU     H'0027'
Reg_2   EQU             H'0028'
Z       EQU     2
C       EQU     0
RCSTA   EQU     H'0018'
TXSTA   EQU     H'0098'
TXREG   EQU     H'0019'
RCREG   EQU     H'001A'
SPBRG   equ     H'0019'
PIR1    EQU             H'000C'
PIR2    EQU             H'000D'
PIE1    EQU     H'008C'
```

```
        org     0
;=============================================================
        ;НАСТРОЙКА ВЫВОДОВ
;=============================================================
        bsf     STATUS,5
        MOVLW   0x00
                                MOVWF           TRISA

                                MOVLW           0X04
                                MOVWF           TRISB
                                bcf         STATUS,5
;=============================================================
        ;ОСНОВНАЯ ПРОГРАММА
;=============================================================
m1
                                call            priem
                                movf            Reg_1,W
                                movwf           PORTA
                                goto            m1
;=============================================================
        ;ПОДПРОГРАММА ПРИЕМ
;=============================================================
priem
                                bcf     STATUS,5
                                bcf             RCSTA,0
                                bsf     STATUS,5
                                bsf             PIE1,5
                            BSF                 TXSTA,2
                                movlw           0x01
                                movwf           SPBRG
          movlw             b'00000000'
                                movwf           TXSTA
                                bcf             STATUS,5
                                movlw           b'10010000'
                                movwf           RCSTA
```

```
GETCHAR
        btfss   PIR1,5
        goto    GETCHAR
        movf    RCREG,w
        movwf       Reg_1
        return
        end
```