

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з дисципліни

«МІКРОКОНТРОЛЕРИ В БІОМЕДИЧНИХ АПАРАТАХ»

для студентів усіх форм навчання
спеціальності 7.05140201, 8.05140201 «Біомедична інженерія»

Електронне видання

ЗАТВЕРДЖЕНО

кафедрою «Біомедична інженерія».

Протокол № 6 від 18.01.2014 р.

ХАРКІВ 2014

Методичні вказівки до лабораторних робіт з дисциплін «Мікроконтролери в біомедичних апаратах» для студентів усіх форм навчання спеціальності 7.05140201, 8.05140201 «Біомедична інженерія» [Електронне видання] / Упоряд. О.Г. Аврунін, О.Я. Крук, В.В. Семенець – Харків: ХНУРЕ, 2014. – 104 с.

Упорядники: О.Г. Аврунін,
 О.Я. Крук,
 В.В. Семенець

Рецензент: Т.В. Жемчужкіна, канд. техн. наук, доцент кафедри БМІ

ЗМІСТ

	ВВЕДЕНИЕ	5
1	ИЗУЧЕНИЕ СИСТЕМЫ КОМАНД И ОСНОВНЫХ ПРИНЦИПОВ ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРОВ НА ПРИМЕРЕ УПРАВЛЕНИЯ БЛОКОМ СВЕТОДИОДОВ.....	6
	1.1 Указания по организации самостоятельной работы.....	6
	1.2 Описание лабораторной установки.....	23
	1.3 Порядок проведения работы и указания по ее выполнению.....	33
	1.4 Содержание отчета.....	37
	1.5 Контрольные вопросы и задания.....	37
2	ИЗУЧЕНИЕ РЕЖИМА ПРОГРАММНОГО ОПРОСА КЛАВИАТУРЫ.....	38
	2.1 Указания по организации самостоятельной работы.....	38
	2.2 Описание лабораторной установки.....	41
	2.3 Порядок проведения работы и указания по ее выполнению.....	43
	2.4 Содержание отчета.....	48
	2.5 Контрольные вопросы и задания.....	48
3	ИЗУЧЕНИЕ ПРИНЦИПОВ ПРОГРАММНОГО УПРАВЛЕНИЯ ВНЕШНИМИ УСТРОЙСТВАМИ НА ПРИМЕРЕ ВЫВОДА ИНФОРМАЦИИ НА ЦИФРОВОЙ ИНДИКАТОР.....	49
	3.1 Указания по организации самостоятельной работы.....	49
	3.2 Описание лабораторной установки.....	52
	3.3 Порядок проведения работы и указания по ее выполнению.....	53
	3.4 Содержание отчета.....	58
	3.5 Контрольные вопросы и задания.....	58
4	ИЗУЧЕНИЕ ПРИНЦИПОВ ОБРАБОТКИ ПРЕРЫВАНИЙ НА ПРИМЕРЕ УПРАВЛЕНИЯ ВСТРОЕННЫМИ В МИКРОКОНТРОЛЛЕР ТАЙМЕРАМИ–СЧЕТЧИКАМИ.....	59
	4.1 Указания по организации самостоятельной работы.....	59
	4.2 Описание лабораторной установки.....	70
	4.3 Порядок проведения работы и указания по ее выполнению.....	70
	4.4 Содержание отчета.....	74
	4.5 Контрольные вопросы и задания.....	74
5	ИЗУЧЕНИЕ ПРИНЦИПОВ ОРГАНИЗАЦИИ ОБМЕНА ДАННЫМИ ПО ПОСЛЕДОВАТЕЛЬНОМУ ИНТЕРФЕЙСУ RS-232C МЕЖДУ МИКРОКОНТРОЛЛЕРОМ AVR ATMEGA128 И ПЭВМ.....	

	75
5.1 Указания по организации самостоятельной работы.....	75
5.2 Описание лабораторной установки.....	80
5.3 Порядок проведения работы и указания по ее выполнению.....	81
5.4 Содержание отчета.....	85
5.5 Контрольные вопросы и задания.....	85
6 ИЗУЧЕНИЕ ПРИНЦИПОВ РАБОТЫ СО ВСТРОЕННЫМ В МИКРОКОНТРОЛЛЕР АНАЛОГО-ЦИФРОВЫМ ПРЕОБРАЗОВА- ТЕЛЕМ НА ПРИМЕРЕ ИЗМЕРЕНИЯ ТЕМПЕРАТУРЫ С ПОМОЩЬЮ АНАЛОГОВОГО ТЕРМОДАТЧИКА.....	86
6.1 Указания по организации самостоятельной работы.....	86
6.2 Описание лабораторной установки.....	89
6.3 Порядок проведения работы и указания по ее выполнению.....	90
6.4 Содержание отчета.....	95
6.5 Контрольные вопросы и задания.....	95
РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА	96
ПРИЛОЖЕНИЕ 1. РАСПОЛОЖЕНИЕ ВЫВОДОВ МИКРОКОНТРОЛ- ЛЕРА AVR ATMEGA 128.....	97
ПРИЛОЖЕНИЕ 2. СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРОВ AVR.....	98

ВВЕДЕНИЕ

Лабораторный практикум по дисциплинам “Микроконтроллеры в бытовых электронных аппаратах”, “Вычислительные и микропроцессорные средства в электронных аппаратах”, “Сопряжение микропроцессорных систем с внешними устройствами”, “Применение микропроцессоров в биомедицинской аппаратуре” предназначен для оказания помощи студентам в усвоении и закреплении знаний по соответствующим дисциплинам, приобретения практических навыков в процессе самостоятельного решения ряда задач, в которых используется микропроцессорная техника или проектируются компоненты систем управления.

Выполнение лабораторных работ предусматривает наличие у студента знаний по курсам: «Основы программирования», «Цифровая схемотехника», «Микропроцессорная техника». Дисциплины, связанные с изучением микроконтроллеров, являются логическим продолжением последних и основой для дисциплин, направленных на изучение способов регистрации внешних сигналов, их автоматизированной обработки и анализа.

Подготовка к лабораторным работам, в результате которой проводится составление предварительных вариантов программных листингов, является одним из основных этапов самостоятельной работы студентов под руководством преподавателя.

Перед началом выполнения практической части лабораторной работы проводится экспресс–контроль знаний, необходимых для выполнения работы.

Студенты допускаются к рабочим местам для выполнения практической части лабораторной работы при условии успешного прохождения экспресс–тестирования и наличия листингов программ для исследования.

Курс лабораторных работ по изучению сопряжения микропроцессорных систем с внешними устройствами поставлен на базе применения многофункциональных лабораторных макетов на основе микроконтроллера AVR ATMEGA128 с использованием ПЭВМ типа IBM PC/AT с соответствующим программным обеспечением в качестве программатора. Микроконтроллеры AVR семейства MEGA – это универсальные 8-ми разрядные высокопроизводительные микроконтроллеры, основанные на RISC ядре, и предназначенные для решения широкого спектра задач для встроенных систем управления.

Описание лабораторной работы включает краткие теоретические сведения, необходимые для успешного выполнения задания, описание программно–аппаратного обеспечения перечень расчетных и экспериментальных заданий, методику выполнения экспериментальной части работы, контрольные вопросы и задания.

1 ИЗУЧЕНИЕ СИСТЕМЫ КОМАНД И ОСНОВНЫХ ПРИНЦИПОВ ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРОВ НА ПРИМЕРЕ УПРАВЛЕНИЯ БЛОКОМ СВЕТОДИОДОВ

Цель работы: изучить архитектуру и принципы программирования микроконтроллера AVR ATMEGA128 на примере разработки программы для управления блоком светодиодов; структурную организацию, состав и возможности компонентов лабораторного макета, освоить пользовательский интерфейс среды программирования C Code Vision AVR.

1.1 Указания по организации самостоятельной работы

Перед работой необходимо проработать теоретический материал по литературе [1, 2] и конспект лекций, ознакомиться со структурой и принципами функционирования микроконтроллера AVR ATMEGA128, системой команд и основами программирования на языках Assembler и C. При подготовке к лабораторной работе необходимо составить предварительные варианты листингов программ, указываемых в пунктах практического выполнения работы.

1.1.1. Основные характеристики микроконтроллера AVR ATMEGA128. AVR-архитектура объединяет высокопроизводительный RISC-процессор с отдельным доступом к памяти программ и данных, 32 регистра общего назначения, каждый из которых может работать как регистр-аккумулятор, и развитую систему команд с фиксированной (16-бит) длиной. Конвейерная архитектура с одновременным исполнением текущей и выборкой следующей команды позволяет выполнять большинство команд за один машинный цикл, что обеспечивает производительность до 1 MIPS на каждый МГц тактовой частоты.

Ниже приводятся основные характеристики микроконтроллера AVR ATMEGA128:

- производство по КМОП-технологии с низким энергопотреблением;
- тактовая частота может изменяться в широких пределах от 0 до 16 МГц (полностью статическая архитектура);
- ядро микроконтроллера основано на RISC архитектуре с двухступенчатым конвейером, обеспечивающим выполнение одной команды за один машинный цикл;
- гарвардская архитектура с отдельной памятью программ и данных;
- регистровый файл содержит 32 регистра общего назначения;
- все регистры общего назначения непосредственно подключены к АЛУ;
- совмещенная архитектура ввода/вывода (регистры общего назначения и порты ввода/вывода находятся в адресном пространстве ОЗУ данных);
- наличие программного стека;

наличие в составе АЛУ аппаратного умножителя;
19 источников внутренних прерываний, 8 источников внешних прерываний;
Объем FLASH-памяти программ: 128 кБт;
Объем статической оперативной памяти (ОЗУ) : 4 кБт
Объем памяти данных на основе электрически-стираемого ПЗУ (EEPROM): 4 кБт;
Интерфейсы программирования: SPI и JTAG;
Напряжение питания: 4.5–5.5 В;
Периферийные устройства:
8-разрядные параллельные порты ввода/вывода;
8 и 16 разрядные таймеры-счетчики;
шиотно-импульсные модуляторы;
аналоговые компараторы,
10-разрядный 8-канальный АЦП,
Встроенный универсальный асинхронный приемопередатчик (USART).

Высокая производительность, наличие развитой подсистемы ввода/вывода и широкого спектра встроенных периферийных устройств позволяют отнести микроконтроллеры AVR ATMEGA128 к классу наиболее функциональных микроконтроллеров для встроенных систем управления, применяемых в бытовой и офисной технике, мобильных телефонах, контроллерах периферийного оборудования (принтеры, сканеры, приводы CD-ROM), портативных медицинских приборах, интеллектуальных датчиках (охранных, пожарных) и др.

1.1.2. Программная модель микроконтроллера AVR MEGA128. Механизм работы с регистрами, памятью и портами ввода/вывода.

В микроконтроллере AVR ATMEGA128 реализована гарвардская архитектура, в соответствии с которой адресные пространства памяти программ и данных физически разделены (доступ к этим областям памяти осуществляется по отдельным шинам). Такая организация позволяет ядру процессора одновременно работать с памятью программ и данных, что повышает быстродействие. Карта распределения памяти в микроконтроллере AVR ATMEGA128 приведена на рисунке 1.1. Память программ представляет собой электрически стираемое перепрограммируемое постоянное запоминающее устройство ППЗУ объемом 128 кБт, выполненное по технологии FLASH – памяти, и предназначена для хранения команд, управляющих функционированием микроконтроллера, а также для хранения констант, не меняющих своих значений в ходе выполнения программы. Так, как длина команды составляет 16 бит, то память программ имеет 16-разрядную организацию. Для адресации памяти программ используется 16-разрядный регистр – программный счетчик PC (Program Counter). Программа исполняется последовательно. Для управления ходом выполнения

программы существуют команды перехода, изменяющие соответствующим образом значение РС.

Память данных организована по принципу совмещенной архитектуры ввода/вывода и разделена на 3 части: регистровая память, память портов (регистров) ввода/вывода и статическое ОЗУ (SRAM), расположенные в едином адресном пространстве.

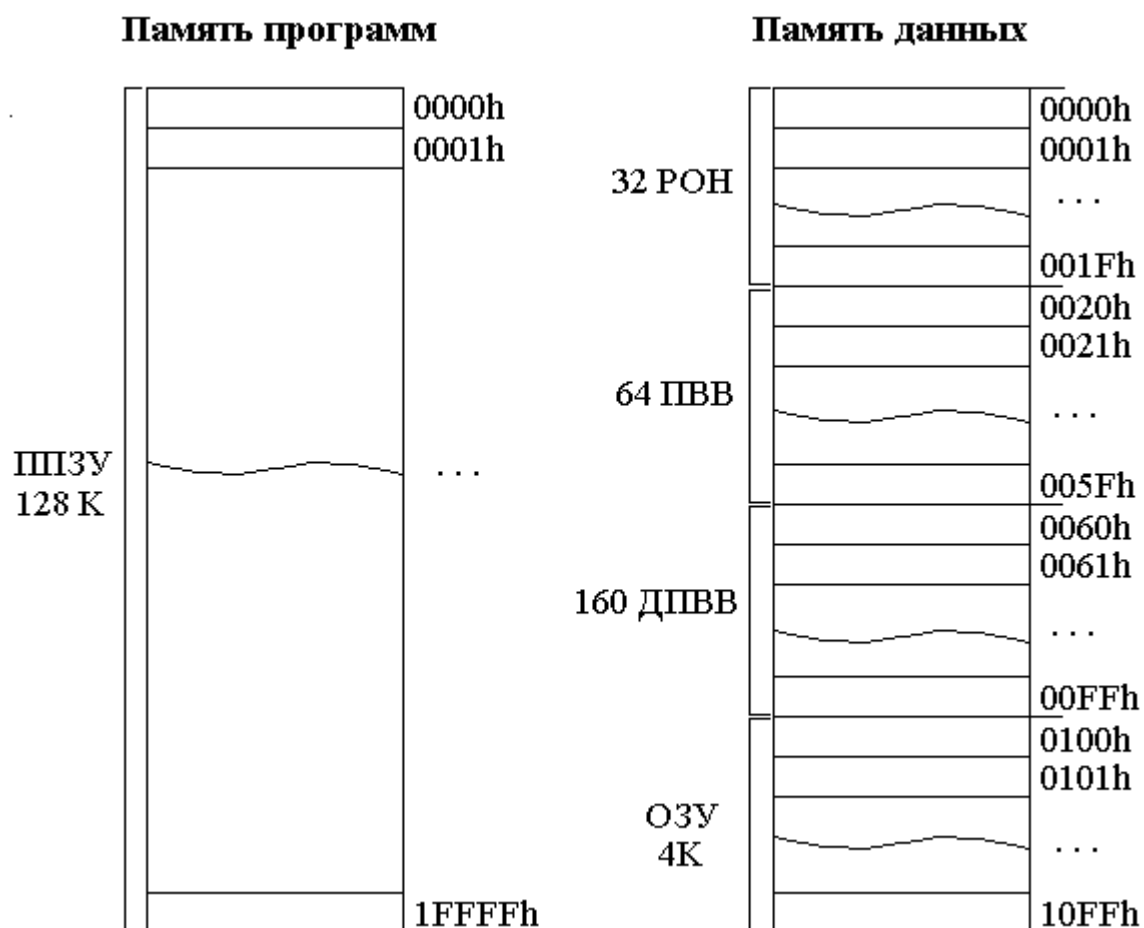


Рисунок 1.1 – Распределение памяти в микроконтроллере AVR ATMEGA128

Регистровая память (см. рисунок 1.2) включает 32 8-разрядных регистра общего назначения (R0 - R31), объединенных в регистровый файл. Каждый из регистров общего назначения непосредственно связан с АЛУ. АЛУ поддерживает арифметические и логические операции с регистрами, между регистром и константой или непосредственно с регистром. При выполнении арифметической или логической команды за один такт из регистрового файла выбираются два операнда, выполняется действие, и результат возвращается в регистровый файл. Регистровый файл отображается на младшие 32 адреса 0000h-001Fh памяти данных и к его регистрам можно обращаться как к ячейкам памяти. Шесть 8 - разрядных регистров (R26 - R31) могут использоваться как три 16-разрядных регистра-указателя для косвенной адресации (см. рисунок 1.3).

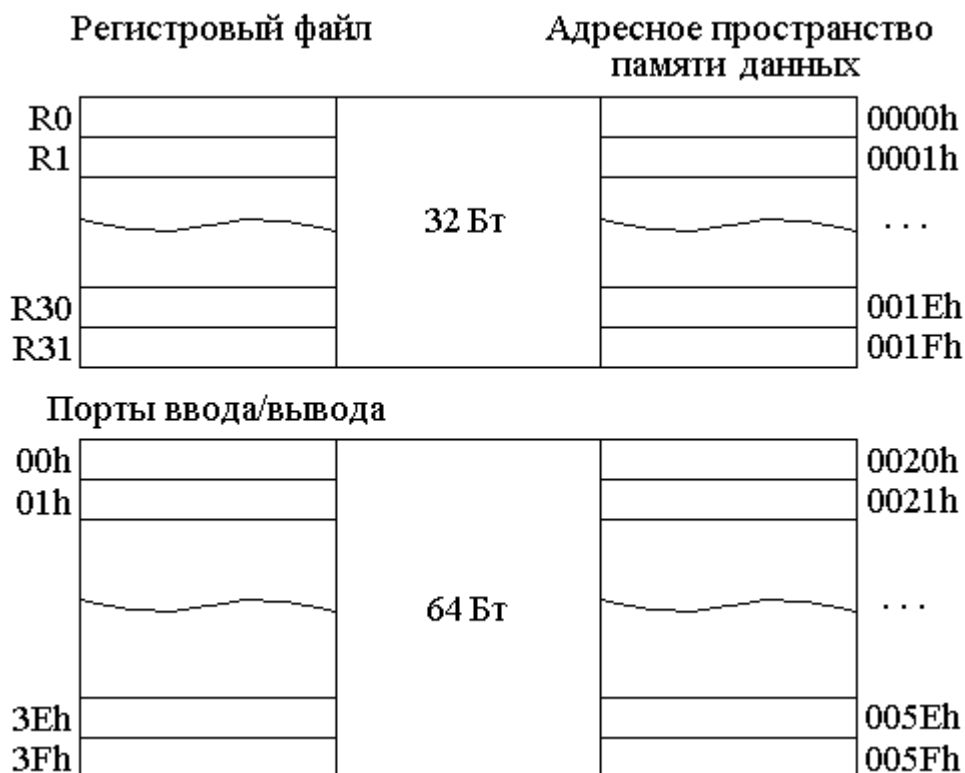


Рисунок 1.2 – Иллюстрация отображения регистров общего назначения и портов ввода/вывода на адресное пространство памяти данных

X		Y		Z	
27	26	29	28	31	30

Рисунок 1.3 – 16-разрядные регистры X, Y, Z, использующиеся для косвенной адресации памяти.

Пространство ввода/вывода состоит из 64 адресов портов 0000h-003Fh, предназначенных для взаимодействия с внутренними и внешними устройствами по отношению к микроконтроллеру. Порты ввода/вывода отображаются на область памяти данных с адресами 0020h-005Fh и допускают возможность обращения к ним как к ячейкам памяти. При доступе к порту ввода/вывода как к ячейке памяти к адресу порта необходимо добавить 20h. Адрес порта ввода/вывода в пространстве памяти часто указывается в скобках после адреса в пространстве портов ввода/вывода. Ввиду того, что основной функцией микроконтроллера является управление внешними устройствами, в таблице 1.1. приводятся названия и адреса (в пространстве портов ввода/вывода) основных интерфейсных портов с указанием режима работы и функций отдельных регистров.

По адресам памяти 0060h-00FFh расположены 160 дополнительных

регистров ввода/вывода.

Непосредственно память данных представляет собой статическое ОЗУ (SRAM) объемом 4 кБт, занимающее диапазон адресов 0100h-10FFh.

Таблица 1.1 – Порты ввода/вывода микроконтроллера AVR MEGA128 для подключения внешних устройств

Название порта ввода/вывода	Идентификаторы отдельных регистров	Адрес	Режим / функция
PORTA	PINA	19h	IN
	DDRA	1Ah	OUT / DIRECTION
	PORTA	1Bh	OUT
PORTB	PINB	16h	IN
	DDRB	17h	OUT / DIRECTION
	PORTB	18h	OUT
PORTC	PINC	13h	IN
	DDRC	14h	OUT / DIRECTION
	PORTC	15h	OUT
PORTD	PIND	10h	IN
	DDRD	11h	OUT / DIRECTION
	PORTD	12h	OUT
PORTE	PINE	01h	IN
	DDRE	02h	OUT / DIRECTION
	PORTE	03h	OUT
PORTF	PINF	00h	IN
	DDRF	61h	OUT / DIRECTION
	PORTF	62h	OUT
PORTG	PING	63h	IN
	DDRG	64h	OUT / DIRECTION
	PORTG	65h	OUT

Регистр состояния SREG расположен в области ввода/вывода по адресу 3Fh (5Fh) и содержит информацию о текущем состоянии микроконтроллера. Расположение флаговых битов регистра состояния приведено на рисунке 1.4.

№ бита	7	6	5	4	3	2	1	0
3Fh (5Fh)	I	T	H	S	V	N	Z	C

Рисунок 1.4 – Регистр состояния SREG.

Назначение отдельных битов регистра состояния приведено ниже:

- **C** – флаг переноса, устанавливается в 1 при наличии переноса в

- арифметических операциях;
- **Z** – флаг нуля, устанавливается в 1, если результат операции равен 0;
- **N** – флаг отрицательного результата, устанавливается в 1 при получении отрицательного результата;
- **V** – флаг переполнения, фиксирует выход результата за пределы допустимого диапазона значений;
- **S** – флаг знака, $S = N \text{ xor } V$;
- **H** – флаг дополнительного переноса (из младшей тетрады байта в старшую);
- **T** – флаг для временного хранения бита из регистров общего назначения;
- **I** – управляющий флаг разрешения прерываний, разрешает (1) или запрещает (0) процессору реагировать на аппаратные прерывания.

1.1.3 Система команд микроконтроллера AVR MEGA128.

Базовый набор команд языка ASSEMBLER для микроконтроллеров AVR содержит 120 инструкций, которые можно разделить на 4 группы: команды пересылки данных; арифметические и логические команды; инструкции для работы с битами; команды управления ходом исполнения программы.

Команды пересылки данных. Группа команд пересылки данных включает в себя инструкции по загрузке значений констант, пересылки данных типа регистр – регистр, регистр – память, регистр – порт ввода/вывода. Команды данной группы являются двух-операндными, причем первым операндом является приемник данных, а вторым – источник данных.

Команда загрузки констант **ldi R, K** применяется для записи непосредственного значения K в регистр – приемник R. В качестве регистра – приемника могут использоваться регистры общего назначения R16 – R31. Если константа представлена в двоичной или шестнадцатеричной системах счисления, то перед значением константы K необходимо указать спецификатор системы счисления **0b** – для двоичной, **0x** – для шестнадцатеричной соответственно. Примеры:

ldi R16, 125 загрузка в R16 десятичного числа 125;
ldi R20, 0xFF загрузка в R20 шестнадцатеричной константы FFh;
ldi R23, 0b11011001 загрузка в R23 двоичной константы 11011001;

Команда пересылки данных между регистрами **mov Rd, Rs** используется для пересылки значения из регистра-источника Rs в регистр-приемник Rd. Операнды в команде являются исключительно регистрами общего назначения R0 – R31.

Примеры:

mov R16, R0 загрузка в R16 значения из регистра R0;
mov R17, R20 загрузка в R17 значения из регистра R20;

В командах пересылки данных между регистром и ячейкой памяти используется механизм косвенной адресации, при котором адрес ячейки памяти заносится в один из 16-разрядных регистров X,Y,Z (см. рисунок 1.3).
Форматы команд:

ld R₈, (R₁₆) – загрузка данных из ячейки памяти, адрес которой находится в 16-разрядном регистре R₁₆, в регистр общего назначения R₈

st (R₁₆), R₈ – загрузка данных из регистра общего назначения R₈ в ячейку памяти, адрес которой находится в 16-разрядном регистре R₁₆,

ldd R₈, (R₁₆+Q) – загрузка данных в регистр общего назначения R₈ из ячейки памяти, адрес которой находится как сумма значения, находящегося в 16-разрядном регистре R₁₆, и смещения Q.

std (R₁₆+Q), R₈ – загрузка данных из регистра общего назначения R₈ в ячейку памяти, адрес которой находится как сумма значения, находящегося в 16-разрядном регистре R₁₆, и смещения Q.

ld R₈, (R₁₆) – загрузка в регистр общего назначения R₈ данных из ячейки памяти, адрес которой находится в 16-разрядном регистре R₁₆;

Примеры:

ld R2, X загрузка в R2 значения из памяти по адресу, указанному в X;

st Y, R5 загрузка значения из регистра R5 в память по адресу, указанному в Y.

ldd R5, Z+1 загрузка в R5 байта из памяти по адресу Z+1;

std Y+4, R7 загрузка байта из регистра R7 в память по адресу Y+4.

Для обращения к портам ввода/вывода в микропроцессоре предусмотрены специальные команды **in** и **out**:

in R, P ввод данных из порта с адресом P в регистр общего назначения R;

out P, R вывод данных из регистра общего назначения R в порт с адресом P;

Примеры:

in R10, 0x15 ввод данных из порта с адресом 15h в регистр общего назначения R10;

out 0x2F, R8 вывод данных из регистра общего назначения R8 в порт с адресом 2Fh;

Арифметические и логические команды. Для работы с целыми двоичными числами целочисленное АЛУ микроконтроллера AVR MEGA128 поддерживает более десятка арифметических и логических команд.

Основными арифметическими командами являются инструкции сложения, вычитания и умножения. Операндами в командах данной группы могут быть только регистры общего назначения. Результат операции (кроме умножения) записывается по адресу первого операнда.

Основные команды для выполнения операций сложения, вычитания и умножения (для чисел без знака):

add Rd, Rs команда сложения (addition), действие: $Rd=Rd+Rs$;
sub Rd, Rs команда вычитания (subtraction), действие: $Rd=Rd-Rs$.
mul Rd, Rs команда умножения (multipl.), действие: $R1, R0=Rd*Rs$;

Команды изменяют флаги переноса **C**, переполнения **V**, знака **N**, **S**, и нуля **Z**. При выполнении операции умножения **n**-значных чисел местонахождение результата разрядностью $2n$ фиксировано и не указывается в команде: при умножении двух байтов результат размером в слово заносится в регистровую пару (R1, R0), в R0 – младшее слово, в R1 – старшее слово.

Примеры:

add R10, R15 $R10 = R10 + R15$;
sub R2, R7 $R2 = R2 - R7$;
mul R5, R16 $R1, R0 = R5 * R16$.

Команды положительного и отрицательного приращения (инкремента и декремента):

inc R инкремент (increment), действие $R=R + 1$;
dec R декремент (decrement), действие $R=R - 1$;

В качестве операнда в этих командах допускается использовать только регистр общего назначения.

Примеры:

inc R20 действие $R20 = R20 + 1$;
dec R16 действие $R16 = R16 - 1$;

Основными логическими командами микроконтроллера AVR MEGA128 являются:

or Rd, Rs логическое “или”; действие: $Rd = Rd \text{ or } Rs$;
and Rd, Rs логическое “и”; действие: $Rd = Rd \text{ and } Rs$;
eor Rd, Rs “исключающее или”; действие: $Rd = Rd \text{ eor } Rs$.

Данные команды выполняют операции поразрядного логического “или”, логического “и”, “исключающего или” (см. таблицу 1.2) над операндами, находящимися в регистрах общего назначения, причем результат записывается по адресу первого операнда. Примеры:

or R7, R11 действие: $R7 = R7 \text{ or } R11$;
and R10, R11 действие: $R10 = R10 \text{ and } R11$;
eor R25, R30 действие: $R25 = R25 \text{ eor } R30$.

Таблица 1.2 – Таблицы истинности логических операций or, and, eor

or (или)			and (и)			eor (исключ. или)		
Вход		Выход	Вход		Выход	Вход		Выход
A	B	Q	A	B	Q	A	B	Q
0	0	0	0	0	0	0	0	0
0	1	1	0	1	0	0	1	1
1	0	1	1	0	0	1	0	1

1	1	1	1	1	1	1	1	0
---	---	---	---	---	---	---	---	---

Указанные команды используются для выполнения операций поразрядного маскирования: **or** – для установки единиц в заданных разрядах, **and** – для установки нулей, **eor** – для выяснения совпадений значений битов первого операнда с маской. Команды изменяют флаги нуля, **Z**, знака **N** и переполнения **V**. Примеры поразрядных логических операций, иллюстрирующие применение механизма маскирования битов, приводятся в таблице 1.3.

Таблица 1.3 – Примеры поразрядных логических операций

Пример поразрядного маскирования or		Пример поразрядного маскирования and		Пример поразрядного маскирования eor	
Rd	xxxxxxx	Rd	xxxxxxx	Rd	10100110
Rs	00010010	Rs	10110101	Rs	00010010
Rd=Rd or Rs	xxx1xx1x	Rd=Rd and Rs	x0xx0x0x	Rd=Rd eor Rs	10110100

Команда поразрядного инвертирования:

com R логическое отрицание; действие: $R = 0b11111111 - R$, выполняет изменение значений двоичных разрядов операнда (регистр общего назначения) на противоположные.

Пример:

com R3 действие: $R3 = 0b11111111 - R3$.

Полный перечень арифметических и логических команд микроконтроллера AVR MEGA128 приводится в Приложении 2.

Команды для работы с битами. Дополняют совокупность логических операций команды сброса, установки и проверки значений отдельных битов.

Команды сброса **cbi P, n** и установки **sbi P, n** битов предназначены для присваивания значений 0 и 1 отдельным битам портов ввода/вывода соответственно. Первым операндом в этих командах является адрес порта ввода/вывода, вторым – номер бита (от 0 до 7).

Примеры:

cbi 0x17, 5 действие: $0x17_5 = 0$;

sbi 0x40, 1 действие: $0x40_1 = 1$.

Команда логического сдвига **lsl R** осуществляет сдвиг влево на одну позицию всех битов операнда, а в младший разряд добавляется нуль. Старший бит операнда поступает в флаг переноса **C**. В качестве операнда могут использоваться только регистры общего назначения. Команда **lsr R** выполняет сдвиг вправо на одну позицию всех битов операнда, а в старший разряд добавляется нуль. Младший бит операнда поступает в флаг переноса **C**. Механизм работы и синтаксис аналогичен команде **lsl**. Примеры

использования команд логического сдвига:

lsl R17 выполнить логический сдвиг влево всех разрядов в R17;

lsr R9 выполнить логический сдвиг вправо всех разрядов в R9.

Поменять местами младшую и старшую тетрады байта, загруженного в регистр общего назначения, можно с помощью команды **swap R**. Следующий фрагмент иллюстрирует действие команды **swap**:

ldi R19, 0b01001101 Загрузить константу 0b01001101 в регистр R19;

swap R19 В результате исполнения команды **swap** в регистре R19 будет сохранено значение 0b11010100.

Дополняют перечень команд для работы с битами инструкции для сброса/установки значений флаговых разрядов в регистре статуса SREG, описание которых приводится в Приложении 2.

Команды сравнения, условного и безусловного перехода. Команда сравнения **cp Rd, Rs** – осуществляет действие **Rd–Rs** и устанавливает флаги нуля **Z**, отрицательного результата **N**, переполнения **V**, переноса **C** и дополнительного переноса **H**. Результат не сохраняется по адресу первого операнда, а только формируются флаги. Операндами могут быть только регистры общего назначения.

Команды условного перехода вызываются сразу после команд сравнения (или других инструкций, вызывающих изменения битов регистра состояния SREG) и на основе анализа флагов осуществляют переход по указанному адресу (метке) в памяти команд.

Наиболее распространенными среди команд этой группы являются:

breq M переход на M, если равно;

brne M переход на M, если неравно;

brlo M переход на M, если меньше;

brsh M переход на M, если больше или равно.

Пример совместного использования команд сравнения и условного перехода:

cp R1, R5 сравнить значения в регистрах R1 и R5;

breq lbl1 выполнить переход на метку lbl1, если значения в регистрах R1 и R5 равны (R1–R5=0).

Команда **rjmp M** осуществляют безусловный переход по указанному 8-разрядному адресу (метке, label) в памяти команд. Пример:

rjmp lbl2 безусловный переход на метку lbl2.

Команда **jmp M** осуществляют безусловный переход по указанному 16-разрядному адресу (метке, label) в памяти команд. Пример:

rjmp lbl3 безусловный переход на метку lbl3.

Полный перечень команд сравнения и перехода приводится в Приложении 2.

1.1.4. Синтаксис и основные операторы языка C.

Сложные программные проекты можно более компактно описывать (по сравнению с языком Assembler) с помощью языка программирования C, который обладает возможностями языков низкого и высокого уровня, а также большой библиотекой функций.

Алфавит языка C состоит из строчных и заглавных букв латинского алфавита, цифр (0 – 9) и специальных символов. Причем, при записи идентификаторов и ключевых слов необходимо учитывать регистр символов. Так, идентификаторы `sysreg` и `Sysreg` не являются одинаковыми. Все ключевые слова должны быть набраны строчными буквами. Разделителем между операторами является символ `;`; Закомментированные строки начинаются с идущих подряд двух символов `//`.

Константы в языке C декларируются с помощью директивы **#define** в соответствии с синтаксисом:

#define имя константы значение;

При работе с аппаратными средствами удобно записывать константы в двоичной и шестнадцатеричной формах. Перед значением констант ставятся символы **0b** и **0x** для двоичного и шестнадцатеричного представлений соответственно. Регистр символов при записи шестнадцатеричных констант не имеет значения. Примеры объявления констант:

```
#define k 25;      // объявлена десятичная константа k=25;
#define KX 0xF5; // объявлена шестнадцатеричная константа KX=F5h;
#define k2 0x6e;  // объявлена шестнадцатеричная константа k2=6Eh;
#define KB 0b1010; // объявлена двоичная константа KB=0b1010;
```

Базовыми целыми типами данных в языке C являются: **char** (размер 1 байт, диапазон значений – 128 ÷ 127) и **int** (размер 2 байта, диапазон значений – 32768 ÷ 32767). Модификатор **unsigned**, записываемый перед именем базового типа, позволяет интерпретировать значения приведенных выше типов данных как числа без знака: **unsigned char** (размер 1 байт, диапазон значений 0 ÷ 255), **unsigned int** (размер 2 байта, диапазон значений 0 ÷ 65535). При этом старший разряд является битом данных, а не знаковым битом числа.

Все переменные должны быть декларированы до их использования в программе. При записи имен переменных необходимо учитывать регистр символов. Формат объявления переменной:

тип данных имя переменной [=начальное значение];

Если несколько переменных имеют одинаковый тип данных, то при объявлении их идентификаторы можно перечислить через запятую. Начальное значение переменной можно не указывать. Примеры:

```
char A=10;
int B, C, D;
unsigned int E, F;
```


Рассмотрим основные операторы языка С.

Оператор присваивания имеет следующий синтаксис:

идентификатор = выражение;

В выражениях над операндами могут использоваться следующие арифметические и логические операции языка С:

Арифметические операции:

+ сложение;

– вычитание;

* умножение,

/ деление.

Логические операции:

|| логическое ИЛИ;

&& логическое И;

! логическое НЕ;

| побитовая операция ИЛИ;

& побитовая операция И;

^ побитовая операция исключающее ИЛИ;

~ побитовая операция НЕ;

<< логический сдвиг влево;

>> логический сдвиг вправо.

Язык С относится к строго типизированным языкам программирования: переменным одного типа нельзя непосредственно присваивать значения другого типа данных. Для однозначного определения приоритета операций в выражениях необходимо использовать круглые скобки (). Пример оператора присваивания:

$$A = (B+C)*D;$$

Оператор условия **if/else** позволяет выполнять одно из двух действий в зависимости от условия. Синтаксис оператора:

if (условие) выражение1

[**else** выражение2];

Условие представляет собой выражение, заданное с помощью операций отношения:

== равно;

!= не равно;

< меньше;

<= меньше или равно;

> больше;

>= больше или равно.

Если условие истинно, то выполняется выражение1, если ложно – то выполняется выражение2. Часть **else** может отсутствовать. Если, в

зависимости от условия необходимо выполнить фрагмент программы, состоящий из нескольких операторов, то их необходимо поместить в фигурные скобки { }. Пример использования оператора условия:

```
if (A==B) {C=D+E; I=N+5}
else {C=D-E; I=N-5;}
```

Оператор выбора **switch/case** позволяет избирательно выполнить фрагмент программного кода, в зависимости от значения выражения. Формат оператора:

```
switch (целочисленное выражение) {
case константа1: выражение1;
break;
case константа2: выражение2;
break;
. . .
case константаN: выражениеN;
break;
[ default: действия по умолчанию;] }
```

Оператор **break** должен находиться во всех ветвях, в противном случае нарушится выборочное выполнение команд в ветвях после **case**. Ветвь **default** можно не указывать. Пример использования оператора выбора:

```
switch (num) {
case 0: A=B+C; C=D+2;
break;
case 1: A=B-C; C=D+10;
break;
case 5: A=B*C; C=D+15;
break; }
```

Оператор цикла с параметром **for** используется в тех случаях, когда заранее известно количество итераций цикла. Синтаксис оператора цикла **for** приведен ниже:

```
for (инициализирующее выражение; условное выражение; модифицирующее выражение)
{
операторы тела цикла;
}
```

Рассмотрим работу цикла **for**: инициализирующее выражение при первом запуске цикла присваивает начальное значение счетчику цикла; затем анализируется условное выражение (цикл выполняется пока условие истинно). Каждый раз после всех строк тела цикла выполняется модифицирующее выражение, в котором происходит изменение счетчика цикла. Выход из цикла произойдет, как только условное выражение получит значение **false**. Пример оператора цикла:

```
s=0;
```

```

m = 1;
for (i=1; i<=10; i++)
{ s=s+i;
  m=m*i; }

```

Оператор цикла с предусловием **while** применяется, когда число повторений неизвестно, но необходимо выполнить некоторое условие. Формат оператора приведен ниже:

```

while (условное выражение)
{
  операторы тела цикла;
}

```

Оператор начинается с ключевого слова **while**, за которым следует логическое выражение, возвращающее значения false или true. Операторы, заключенные в фигурных скобках, образуют тело цикла. Пример использования оператора цикла **while**:

```

a=0;
while (a<10)
{ b=(c+d)*a-f;
  a=a+2; }

```

Подпрограммы на языке C оформляются в виде функций. Описание функции имеет следующий синтаксис:

```

[Тип возвращаемого значения] имя функции ([список параметров])
{
  [декларации локальных переменных]
  операторы тела функции;
  [return выражение;]
}

```

Если функция не возвращает значения, то тип возвращаемого значения не указывается и секция return не используется. При отсутствии параметров после имени функции обязательно указываются пустые круглые скобки (). Тело функции заключается в фигурные скобки { }. Переменные, объявленные в теле функции, являются локальными (видимыми только в пределах тела функции). Функции, возвращающие значения, можно использовать в правой части операторов присваивания.

Рассмотрим пример декларации функции, возвращающей, в зависимости от значения аргумента C, сумму или разность двух целых чисел:

```

int sum (int A, int B, char C)
{
  if (C>=0) return A+B;
  else return A-B;
}

```

Пример функции, не имеющей аргументов и не возвращающей значения:

```
init_data ( ) {  
  A=10;  
  B=100+A; }  
}
```

При программировании микроконтроллера часто возникает необходимость использовать ассемблерный код в программе, написанной на языке C. В этом случае фрагмент программы, составленный из ассемблерных операторов, помещается в операторные скобки:

```
#asm  
операторы языка ассемблер  
#endasm ;
```

Пример фрагмента программы на языке Assembler:

```
#asm  
mov r1,r5  
ldi r17,0xf5  
#endasm ;
```

Одиночный ассемблерный оператор в теле C – программы, может быть записан в виде директивы **#asm("Оператор языка Assembler")**;

Пример:

```
#asm("out 0x12,r16"); // выполнить команду out 0x12,r16.
```

Функции, написанные на ассемблере, возвращают значения через регистр **R30** для типов **char** и **unsigned char**, и регистровую пару (**R31**, **R30**) для типов **int** и **unsigned int** (в **R31** – старший, а в **R30** – младший байты). Параметры функции передаются через стек, на вершину которого указывает регистр **Y**, причем старший байт слова записывается по старшему адресу. Механизмы вызова и возврата из функции осуществляются средствами компилятора C. Директива **#pragma warn-** запрещает компилятору генерировать предупреждения о том, что функция не возвращает результат стандартным способом (с помощью оператора **return**).

В качестве примера рассмотрим ассемблерную функцию, выполняющую суммирование двух целых чисел и возвращающую значение типа **int** (пояснения к ассемблерному коду прилагаются):

```
int summa (int A, int B)  
{  
  #asm  
  ldd R27, Y+3      загрузить старший байт параметра A;  
  ldd R26, Y+2      загрузить младший байт параметра A;  
  ldd R25, Y+1      загрузить старший байт параметра B;  
  ld R24, Y;        загрузить младший байт параметра B;  
  add R24, R26      выполнить суммирование младших байтов A и B;
```

```

adc R25, R27      выполнить суммирование старших байтов А и В
                  с учетом флага переноса С;
mov R30, R24     записать младший байт суммы в регистр R30;
mov R31, R25     занести старший байт суммы в регистр R31;
#endasm
}

```

Вызов данной функции может осуществляться следующим образом:

```

...
int C;
C=summa (10, 15);
...

```

Часто в ассемблерных подпрограммах возникает необходимость получить доступ к значениям переменных, объявленных в С–программе. Размещение переменных (в регистрах процессора или памяти данных) можно определить из файла с расширением *.map, имя которого совпадает с именем файла исходного кода программы. Данный файл генерируется компилятором и находится в одном каталоге с исходным модулем программы.

Рассмотрим структуру программы на С. Текст исходного модуля программы начинается с директив препроцессора **#include** <имя файла.h>, с помощью которых в программный код проекта подгружаются файлы заголовков, содержащие объявления различных констант, идентификаторов и прототипы функций. Пример использования данной директивы, подгружающей заголовочный файл mega128.h:

```
#include <mega128.h>
```

Далее следуют объявления констант (с помощью директив **#define**) и глобальных переменных.

Затем записываются декларации функций, используемых в тексте главной программы, расположенной в теле функции **main()**, с операторов которой начинается выполнение программы. Тело функции **main()** расположено внутри фигурных скобок { } .

1.1.5. Принципы программного управления светодиодами, подключенными к внешним выводам портов ввода/вывода микроконтроллера AVR ATMEGA128.

Согласно технической документации на выходных линиях микроконтроллеров **AVR** при уровне напряжения, соответствующем “логическому нулю”, ток нагрузки составляет порядка 20 мА. Стандартные светодиоды потребляют ток в пределах 3÷20 мА при рабочем напряжении порядка 1,5÷4 В. Это позволяет непосредственно подключать светодиод к выходной линии порта последовательно с ограничивающим ток резистором (см. рисунок 1.5). Второй вывод цепи необходимо подсоединить к положительному полюсу источника питания (+5 В).

Для управления светодиодом необходимо подавать на соответствующий вывод микроконтроллера уровни “логического нуля” или “логической единицы”. При появлении на выводе микроконтроллера, к которому подключен светодиод, уровня «логического нуля», падение напряжения на светодиоде будет достаточным для свечения. При формировании на соответствующем выводе микроконтроллера напряжения “логической единицы” падения напряжения на светодиоде не будет, и он будет погашен.

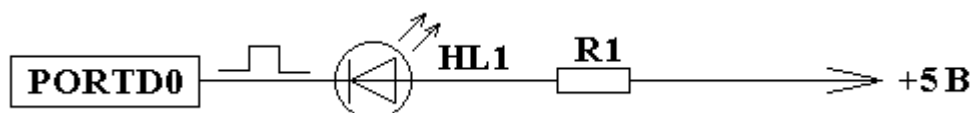


Рисунок 1.5 – Схема подключения светодиода к выходу порта ввода/вывода микроконтроллера AVR

Для управления уровнями напряжения на выходных линиях микроконтроллера можно применять алгоритмы маскирования или, непосредственно, команды для работы с битами (**cbi**, **sbi**), описанные в подразделе 1.1.3.3. Прямое обращение к регистрам портов ввода/вывода на языке Assembler обеспечивается с помощью команд **in** и **out** (см. подраздел 1.1.3.1). Компилятор языка C допускает использование идентификаторов регистров ввода/вывода: **DDRX** – регистр управлением направлением передачи данных, **PORTX** регистр вывода данных, **PINX** – регистр ввода данных, где **X** – обозначение порта ввода/вывода. Например, программно доступные регистры порта A обозначаются как: **DDRA**, **PORTA**, **PINA** (см. таблицу 1.1). Если линия порта ввода/вывода **X** программируется на вывод, то в соответствующем бите регистра управлением направлением передачи данных **DDRX** должна быть установлена **1**, если на ввод – то **0**. Рассмотрим примеры:

unsigned char a, b;	декларация переменных a и b размером в байт;
DDRB=0b01011110;	линии 0, 5 и 7 порта B запрограммированы на вывод данных, остальные (1,2,3,6) – на ввод.
DDRD=0xFF;	установка всех линий порта D в режим вывода данных;
a=0x12;	присвоить переменной a значение 12h;
PORTD = a;	вывести данные (значение переменной a) в порт вывода D .
DDRC=0;	установка всех линий порта C в режим ввода данных;
b=PINC;	записать данные из порта ввода C в переменную b .

Для работы с отдельными разрядами регистров портов ввода/вывода на языке С можно использовать конструкции: **PORTX.N** и **PINX.N**, где **N** – номер бита. Например,

PORTB.2=0; второй бит порта D сбросить в 0;
PORTB.4=1; четвертый бит порта D установить в 1;
PORTD.5=PINF.3; пятому биту порта D присвоить значение третьего бита порта F;

Наиболее универсальным способом работы с отдельными разрядами регистров портов ввода/вывода является использование масок и поразрядных логических операций.

1.2 Описание лабораторной установки

Лабораторная работа выполняется в индивидуальном порядке. На каждом рабочем месте должны быть установлены: многофункциональный лабораторный макет на базе микроконтроллера AVR ATMEGA 128, ПЭВМ типа IBM PC/AT с установленным программным обеспечением: операционной системой MS–WINDOWS v. 9x, 2000, XP и программатором на основе кросс-компилятора языка программирования C CodeVision AVR.

1.2.1. Описание лабораторного макета.

Лабораторный макет представляет собой универсальное устройство на базе 8-ми разрядного микроконтроллера AVR ATMEGA 128 ($f_T=11,0592$ МГц), в состав которого входят (см рисунок 1.6): микроконтроллер AVR ATMEGA128, графический ЖК-дисплей Toshiba T6963C, блок светодиодов, клавиатуры: 3×4 и 3×1 (количество столбцов × количество строк), АЦП, внешний аналоговый термодатчик, пьезоизлучатель, последовательный интерфейс RS-232C. Сопряжение лабораторного макета и программатора (ПЭВМ) обеспечивается с помощью последовательного интерфейса SPI, конструктивно использующего стандартный разъем Centronics DB-25. Общий вид передней панели лабораторного макета приведен на рисунке 1.7. Управление встроенным в макет графическим ЖК-дисплеем осуществляется через порты А и С. Блок из восьми светодиодов подключен к микроконтроллеру через порт D. Сопряжение клавиатуры 3×4 осуществляется с помощью порта E. Через порт F, линии которого являются входами АЦП, к микроконтроллеру подключаются: пьезоизлучатель, термодатчик и клавиатура 3×1. Порт В микроконтроллера свободен и предназначен для подключения внешних устройств, таких как цифровые ЖК-индикаторы, датчики и т.д. Расположение выводов микроконтроллера AVR ATMEGA 128 приводится в Приложении 1.

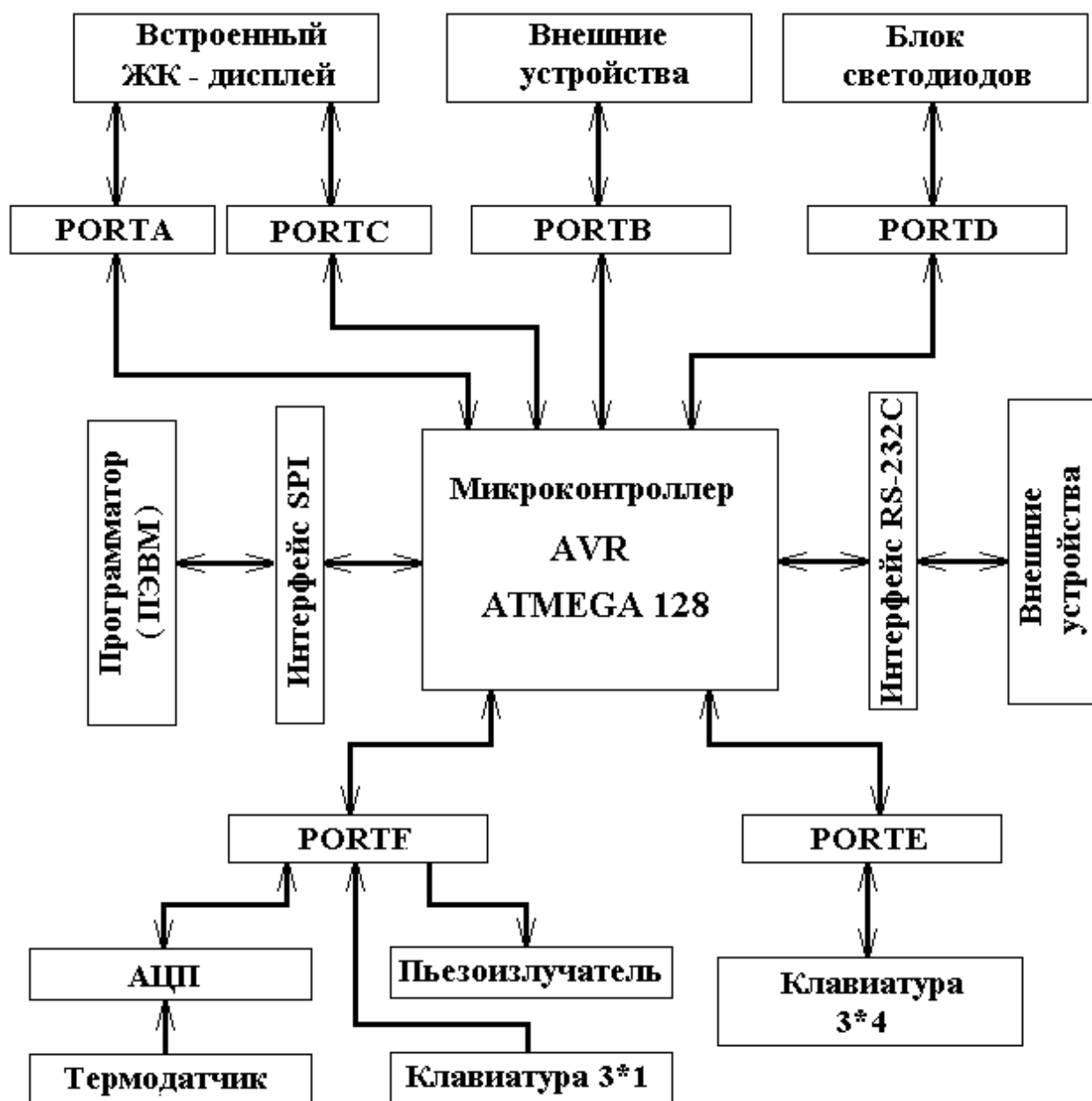


Рисунок 1.6 – Структурная схема лабораторного макета на базе микроконтроллера AVR ATMEGA 128

Интерфейс порта В конструктивно выполнен в виде унифицированного разъема DB-25, расположенного на задней панели корпуса макета (см. рисунок 1.8). Для линий двух встроенных в микроконтроллер последовательных интерфейсов RS-232C (USART0, USART1) предназначены разъемы DB-9, расположенные на задней панели макета, через прорезь в которой проходят сигнальные шлейфы интерфейса SPI и аналогового термодатчика. На задней панели находится также разъем для подключения внешнего источника питания (12 В) и выключатель электропитания. Индикатор электропитания расположен на лицевой панели лабораторного макета.

Внимание: во время выполнения лабораторной работы соблюдать правила техники безопасности при работе с ПЭВМ.



Рисунок 1.7 – Общий вид передней панели лабораторного макета.

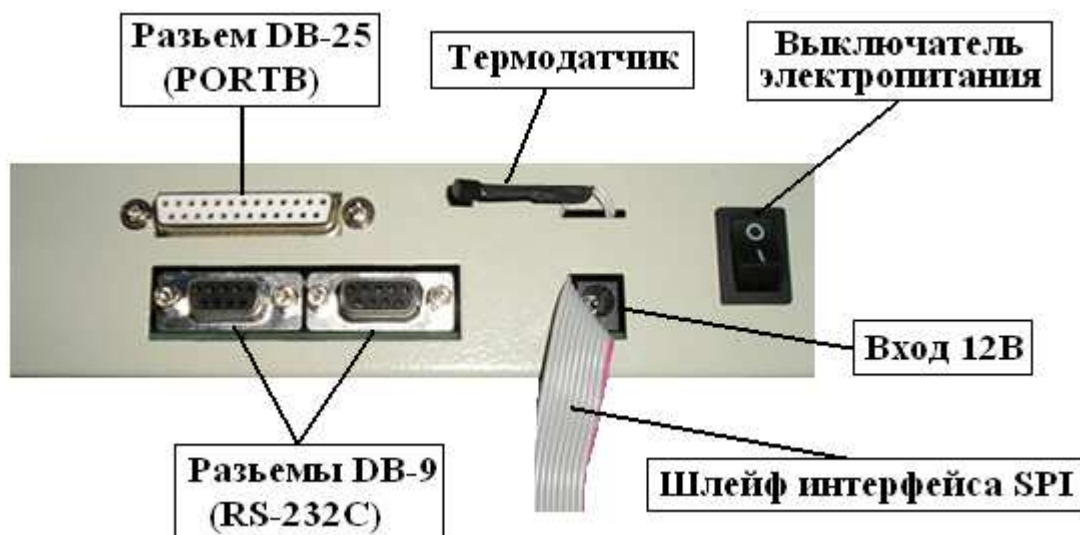


Рисунок 1.8 – Расположение разъемов на задней панели лабораторного макета

1.2.2. Описание блока светодиодов лабораторного макета.

В состав макета входит блок индикации, состоящий из 8 светодиодов (см. рисунок 1.7), подключенных к микроконтроллеру через порт D в соответствии со схемой, приведенной на рисунке 1.9. На аноды светодиодов (HL1-HL8) подается напряжение + 5 В, катоды каждого светодиода подключаются к соответствующим выходам порта ввода/вывода D. В ветвь каждого светодиода последовательно подключается токоограничивающий резистор номиналом 390 Ом (на схеме обозначены R1-R8). Программное управление свечением светодиодов осуществляется путем подачи уровней “логического нуля” и “логической единицы” на соответствующие разряды порта D, который должен функционировать как порт вывода.

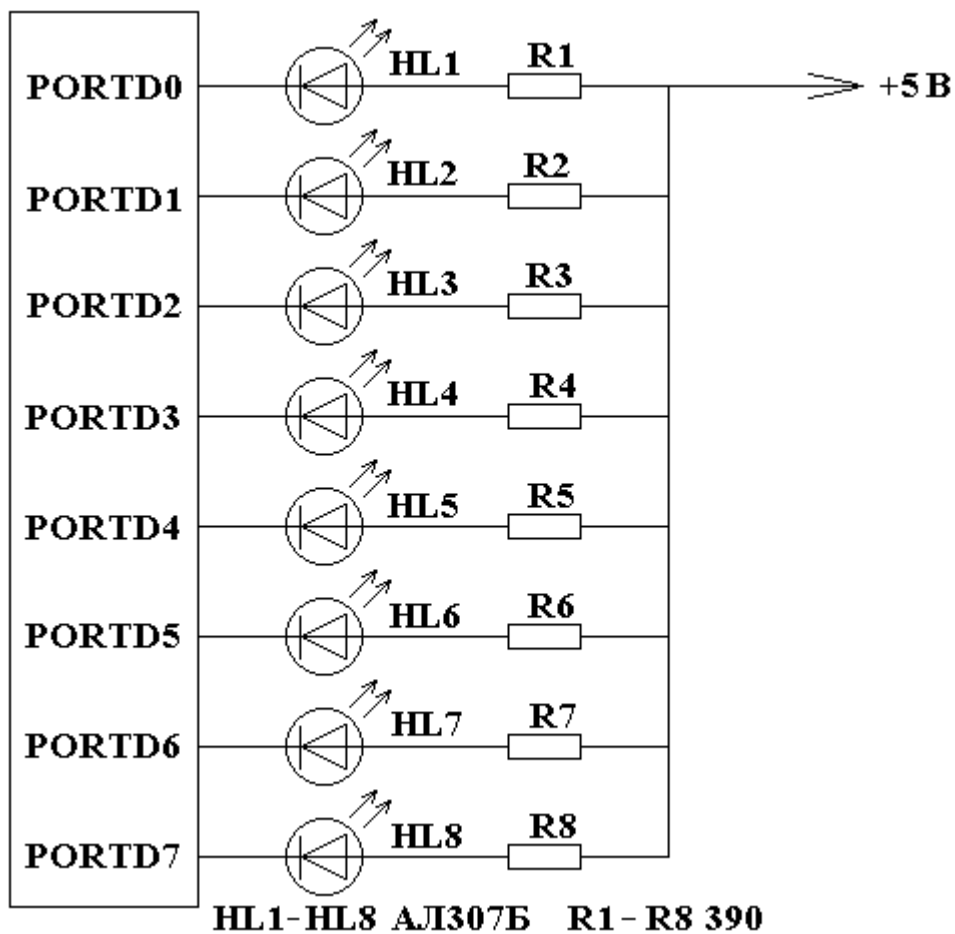


Рисунок 1.9 – Принципиальная схема подключения блока светодиодов к выводам порта D

1.2.3. Описание интерфейса компилятора языка C CodeVision AVR.

Интегрированная среда CodeVision AVR представляет собой кросс-компилятор языка C ориентированный на семейство микроконтроллеров AVR и содержит: графическую оболочку для управления ресурсами проекта; текстовый редактор исходного модуля программы; кросс-компилятор;

отладчик, программатор; автоматический генератор программного кода; терминал для работы с последовательным интерфейсом RS232C (USART). Полученный в результате компиляции исходного кода программы на языке C исполняемый модуль (файл прошивки) может быть непосредственно записан в память программ микроконтроллера.

В среде CodeVision AVR каждая программа для микроконтроллера должна оформляться в виде проекта, представляющего собой совокупность файлов, содержащих исчерпывающую информацию для программатора. Файлы каждого проекта желательно сохранять в отдельном подкаталоге.

При создании нового проекта следует выполнить следующую последовательность действий:

- создать новый каталог для файлов проекта;
- запустить программный модуль CodeVision AVR;
- в появившемся главном рабочем окне программы, используя верхнее меню, выполнить команду **File** ⇒ **New**;
- в открывшемся диалоговом окне (см. рисунок 1.10), выбрать пункт **Project**;

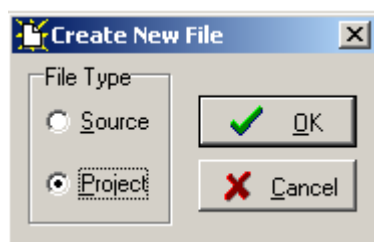


Рисунок 1.10 – Диалоговое окно для выбора типа создаваемого ресурса при создании проекта.

- В появившемся диалоговом окне **Confirm** (см. рисунок 1.11), необходимо отказаться от использования автоматического генератора программного кода, нажав кнопку **No**;

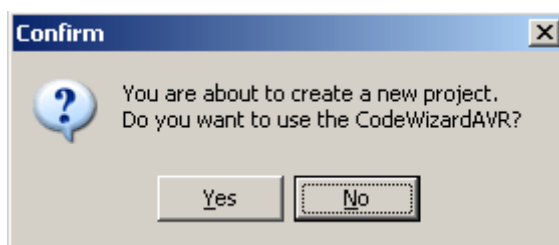


Рисунок 1.11 – Диалоговое окно для выбора автоматического генератора программного кода.

- в появившемся окне для сохранения файла проекта ввести имя файла и нажать ОК (см. рисунок 1.11).

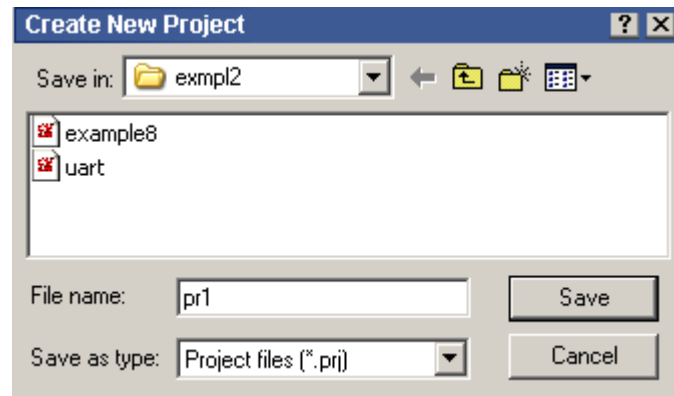


Рисунок 1.12 – Диалоговое окно для сохранения файла проекта.

- далее (см. рисунок 1.13) открывается окно конфигурации проекта (вкладка Files), в котором необходимо активизировать вкладку **C Compiler**;

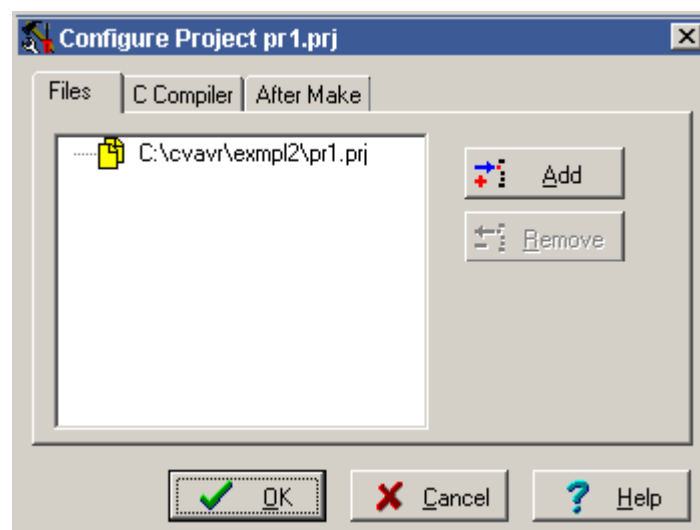


Рисунок 1.13 – Вкладка **Files** окна конфигурации проекта.

- на вкладке **C Compiler** (см. рисунок 1.14) окна конфигурации проекта необходимо задать тип и тактовую частоту микроконтроллера (Chip: **Atmega128**, Clock: **11.059200** MHz).

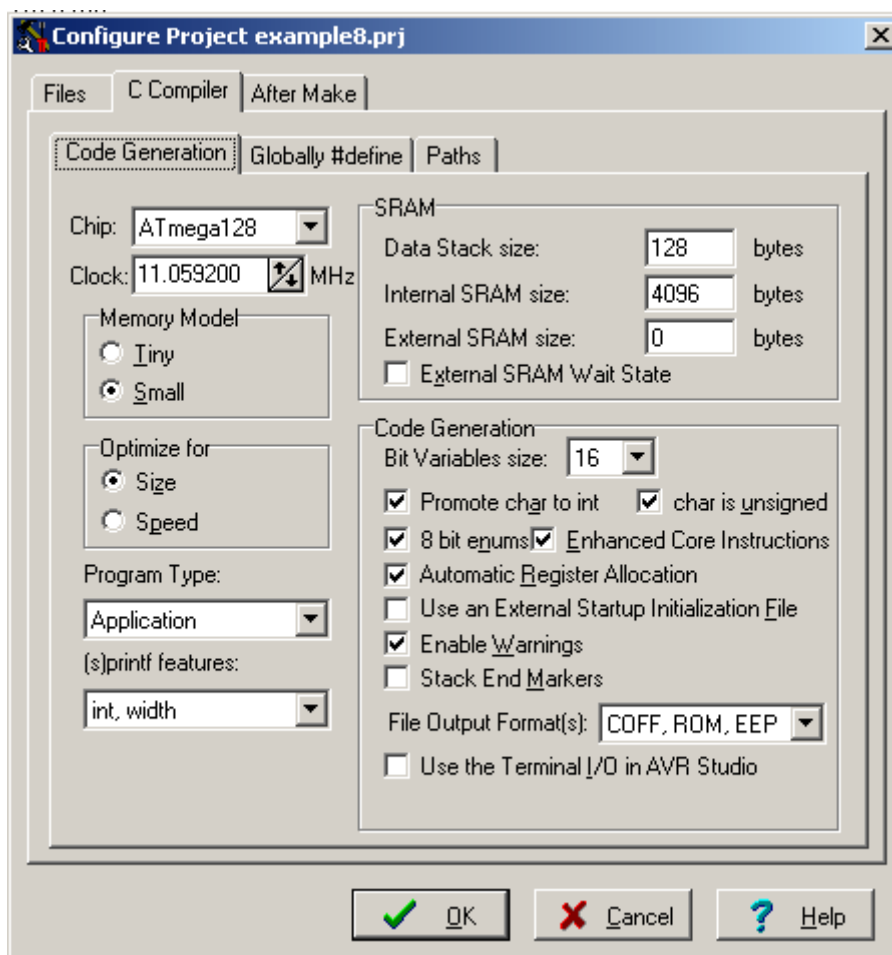


Рисунок 1.14 – Вкладка **C Compiler** окна конфигурации проекта

- на вкладке **After Make** окна конфигурации проекта необходимо активизировать опцию **Program the chip** и нажать клавишу **OK**. В результате будет создан пустой проект и на экране появится главное рабочее окно программы **CodeVision AVR**, имеющее классическую компоновку для интегрированных средств разработки приложений и CAD систем (см. рисунок 1.15). В верхней части главного рабочего окна находятся текстовое меню и пиктограммы для быстрого запуска отдельных команд. Назначения пиктограмм, отвечающих за выполнение специализированных функций, приводится на рисунке 1.16. В левой части главного окна располагается информация о ресурсах проекта, в правой – ресурс, являющийся, в данный момент, активным. В нижней части главного окна располагается строка сообщений (**Messages**).

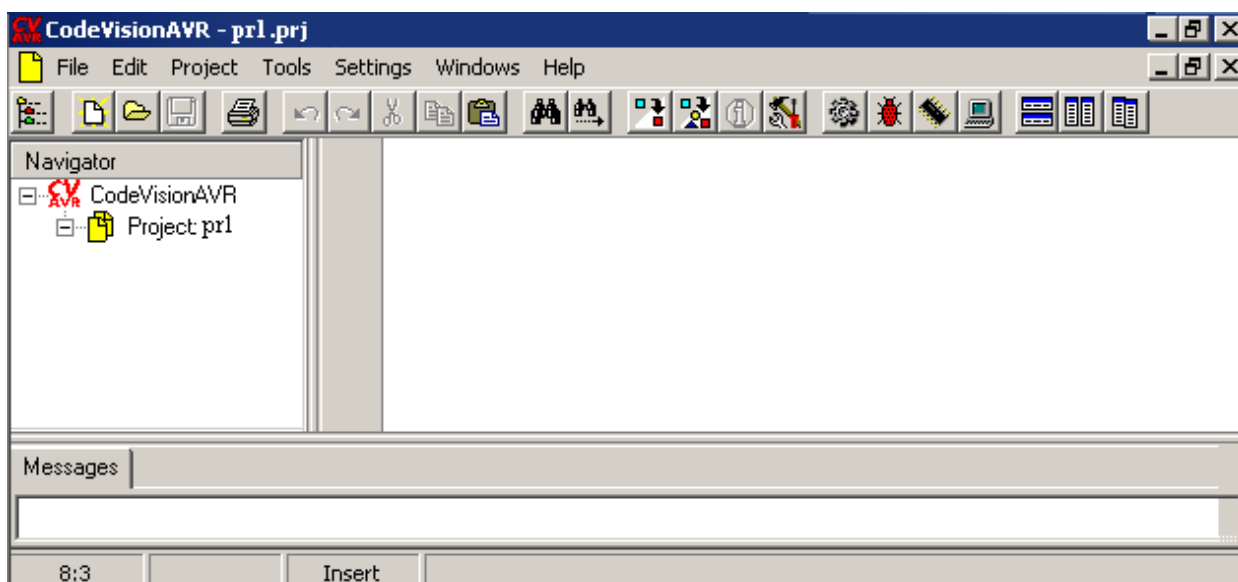


Рисунок 1.15 – Интерфейс главного рабочего окна программы **CodeVision**



Рисунок 1.16 – Назначения пиктограмм, отвечающих за выполнение специализированных функций

Исходный текст программы для микроконтроллера на языке C записывается в отдельном текстовом файле, для создания которого необходимо выполнить следующие действия:

- в главном рабочем окне программы, используя верхнее меню, выполнить команду **File ⇒ New**;
- в открывшемся диалоговом окне (см. рисунок 1.17), выбрать пункт **Source**;

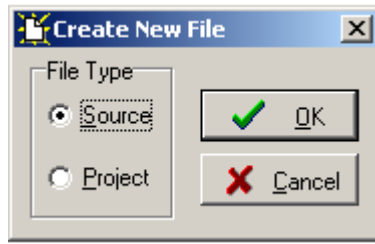


Рисунок 1.17 – Диалоговое окно для выбора типа создаваемого ресурса при создании текстового файла с кодом программы

- в правой части вновь появившегося главного рабочего окна программы будет отображаться текстовый редактор для ввода текста в созданный файл ресурса, для сохранения которого необходимо, используя верхнее меню, выполнить команду **File** ⇒ **Save As ...** и ввести уникальное имя файла;
- файл ресурса необходимо включить в состав проекта. Для этого необходимо с помощью основного или пиктограммного меню (см. рисунок 1.16) вызвать окно конфигурации проекта (**Configure Project**), активизировать вкладку **Files** (см. рисунок 1.13), и, нажав на кнопку **Add**, указать в появившемся диалоговом окне имя файла-ресурса;

После того, как исходный текст программы для микроконтроллера будет набран, необходимо выполнить компиляцию проекта и, непосредственно, загрузку кода и данных программы в память микроконтроллера:

- перед выполнением процедуры программирования микроконтроллера необходимо произвести настройку параметров интерфейса программатора (**Programmer Settings**) с помощью команды **Programmer** из подпункта главного меню **Setting** в соответствии с данными, приведенными на рисунке 1.18. Параметры интерфейса программатора устанавливаются только один раз перед началом работы с микроконтроллером и при корректной работе устройства не требуют изменений.

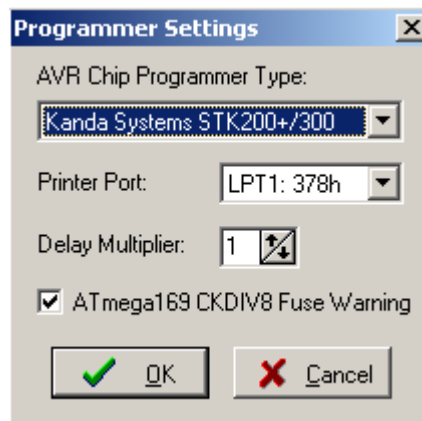


Рисунок 1.18 – Окно настройки параметров интерфейса программатора

- далее необходимо выполнить компиляцию проекта – создать исполняемый программный модуль, пригодный для исполнения микроконтроллером. Для этого необходимо нажать клавишу **F9** или выбрать подпункт **Компиляция проекта (Compile)** из пиктограммного меню (см. рисунок 1.16). В большинстве случаев целесообразно сразу после компиляции произвести программирование микроконтроллера, для чего необходимо нажать комбинацию клавиш **Shift+F9** или выбрать подпункт **Программирование микроконтроллера (Make)** из пиктограммного меню (см. рисунок 1.16). При завершении этапа компиляции активизируется окно **Information** (см. рисунок 1.19), содержащее информацию о скомпилированном проекте. Если компиляция прошла успешно, то для записи программы в память микроконтроллера необходимо нажать кнопку **Program**, расположенную в нижней части окна **Information**.

Выявленные в результате компиляции сообщения об ошибках отображаются в левой части главного рабочего окна, где располагаются данные о ресурсах проекта. При активизации сообщения об ошибке компилятор выводит подробные сведения о локализации и возможных причинах ошибки.

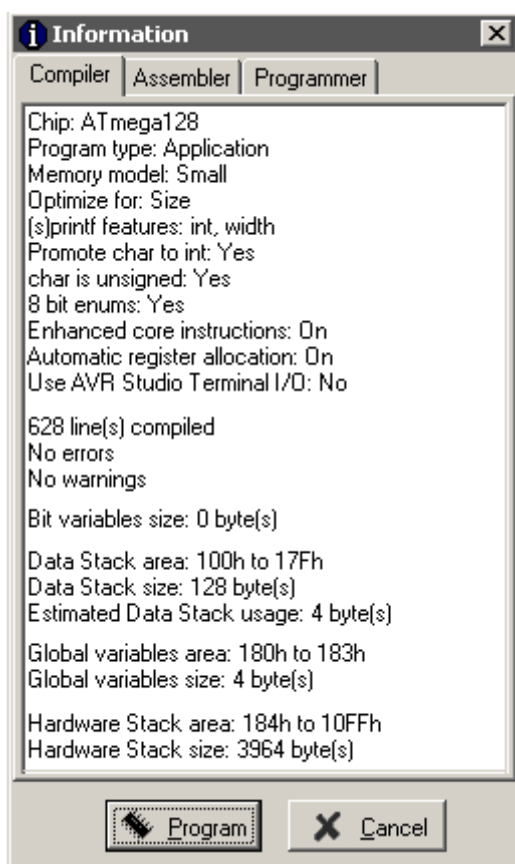


Рисунок 1.19 – Окно отображения информации о результатах компиляции проекта

В некоторых случаях для проверки работоспособности загруженной программы необходимо выполнить сброс микроконтроллера (**Reset Chip**) или удалить программу из памяти (**Erase Chip**). Эти функции (см. рисунок 1.20) становятся доступными при выборе подпункта **Настройки Программатора** из пиктограммного меню (см. рисунок 1.16).

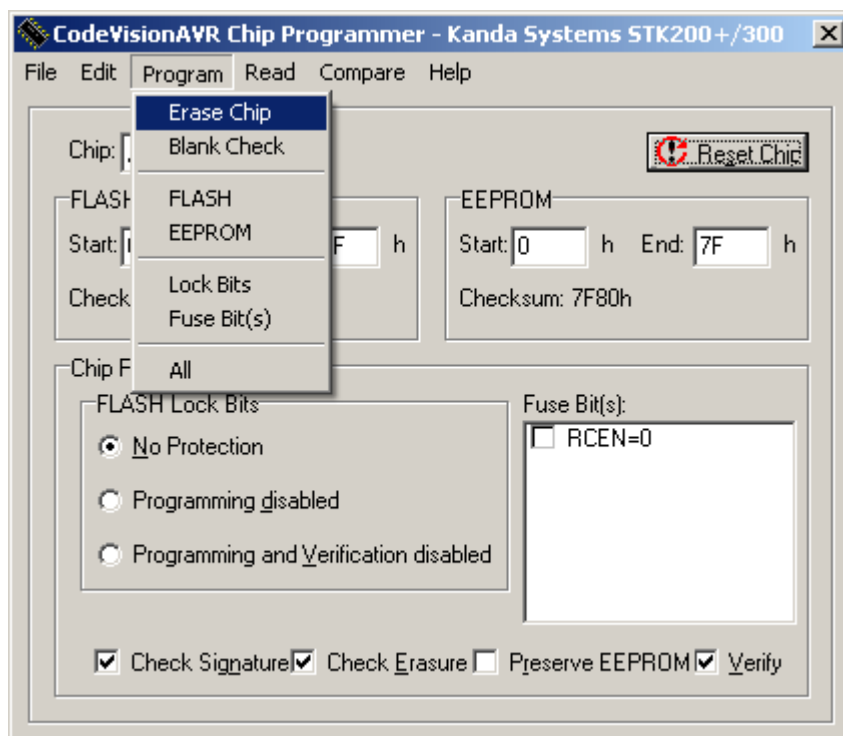


Рисунок 1.20 – Окно отображения настроек программатора

1.3 Порядок проведения работы и указания по ее выполнению

Перед началом выполнения практической части лабораторной работы проводится экспресс–контроль знаний по принципам функционирования микроконтроллера AVR ATMEGA 128, системе команд и возможностям программного управления светодиодами, которые непосредственно подключаются к внешним линиям порта ввода/вывода. При подготовке к лабораторной работе необходимо составить предварительный вариант листинга программы, в соответствии с индивидуальным заданием (см. таблицу 1.4).

Задание 1. Разработать в среде программирования Code Vision AVR программу на языке C для микроконтроллера AVR ATMEGA 128, управляющую блоком из восьми светодиодов. Варианты индивидуальных заданий представлены в таблице 1.4.

Порядок выполнения задания:

1. Включить лабораторный макет (установить выключатель электропитания в положение I, и убедиться в свечении индикатора электропитания красным цветом).
2. Запустить компилятор Code Vision AVR.
3. Создать пустой проект.
4. Создать файл ресурса для кода программы и подключить его к проекту.
5. Ввести код исходного модуля программы управления светодиодами в соответствие с вариантом задания, указанном в таблице 1.4.
6. Выполнить компиляцию (нажав клавишу **F9**) исходного модуля программы и устранить ошибки, полученные на данном этапе.
7. Настроить параметры программатора.
8. Создать загрузочный модуль программы (нажав комбинацию клавиш **Shift+F9**) и выполнить программирование микроконтроллера.
9. Проверить работоспособность загруженной в микроконтроллер программы и показать результаты работы преподавателю.
10. В случае некорректной работы разработанной программы, выполнить аппаратный сброс микроконтроллера, провести отладку исходного модуля программы и заново проверить функционирование программы, повторив выполнение пункта 9.

Пример выполнения задания. Разработать программу, выполняющую в бесконечном цикле параллельное включение и выключение 1–го, 3–го, 6–го и 8–го светодиодов с длительностью свечения 2 с и временем нахождения в погашенном состоянии 1 с.

Решение. В лабораторном макете блок, состоящий из 8-ми светодиодов, подключен к порту D микроконтроллера в соответствии принципиальной схемой, приведенной на рисунке 1.9. При этом необходимо учитывать, что нумерация светодиодов начинается с 1, а не с 0. Программное управление светодиодами можно обеспечить, записывая в соответствующие разряды регистра PORTD порта D уровни “логического нуля” (зажечь светодиод) или “логической единицы” (погасить светодиод) согласно алгоритму, приведенному на рисунке 1.21. Полный текст исходного модуля программы с подробными комментариями приводится ниже:

```
#include <mega128.h> Подключить заголовочный файл mega128.h;
#include <delay.h>   подключить заголовочный файл delay.h;
main() {             основная часть программы;
  DDRD=0xFF;        настроить порт D на вывод данных;
  while (1) {       создать цикл с бесконечным числом итераций;
    PORTD=0b1111111; погасить все светодиоды;
    delay_ms(1000);  установить временную задержку 1 с;
    PORTD=0b01010101; включить 1–й, 3–й, 6–й и 8–й светодиоды;
    delay_ms(2000); } установить временную задержку 2 с;
  }                 завершающая операторная скобка программы;
```

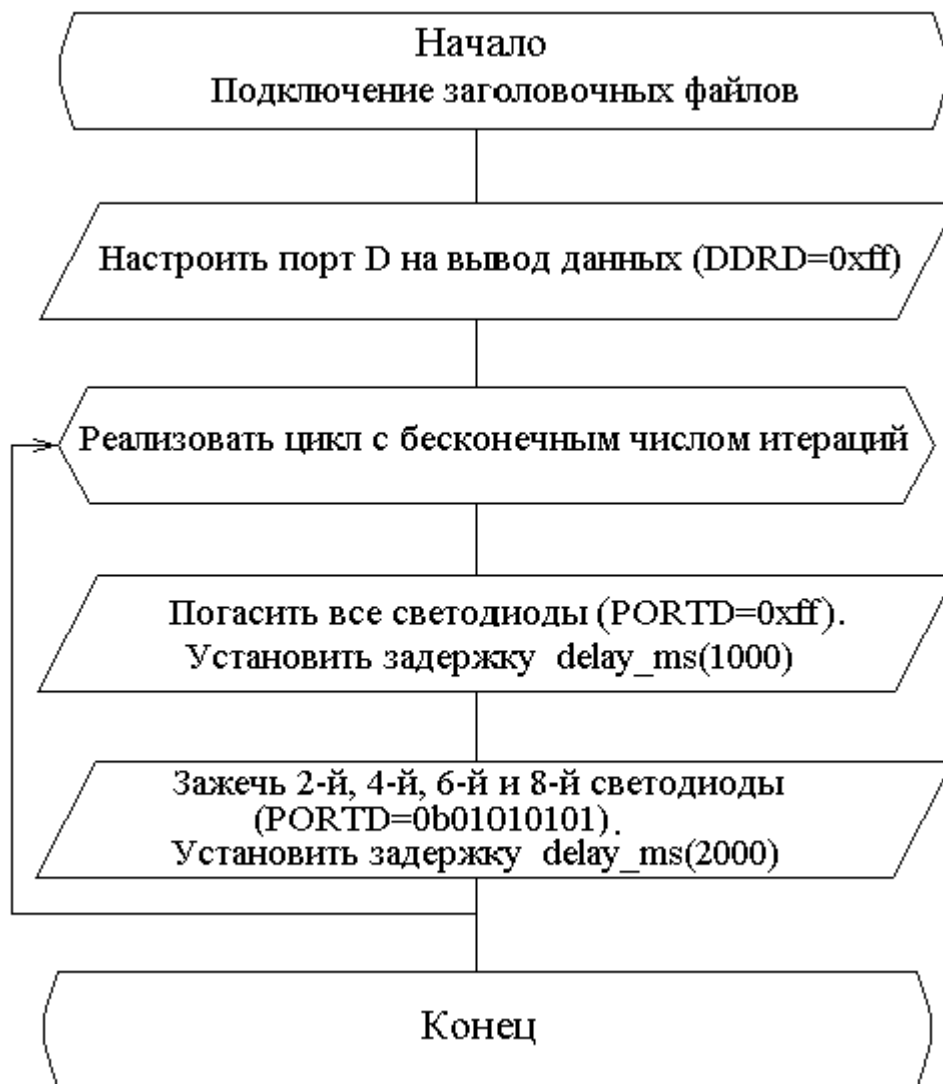


Рисунок 1.21 – Алгоритм программы управления светодиодами

Задание 2. Выполнить предыдущее задание с использованием команд пересылки данных языка Assembler.

Порядок выполнения задания совпадает с последовательностью действий, указанной в пункте 1.3.1. Адреса регистров порта D приведены в таблице 1.1.

Пример выполнения задания, рассмотренного в пункте 1.3.1, с использованием операторов языка Assembler приводится ниже:

#include <mega128.h>	Подключить заголовочный файл mega128.h;
#include <delay.h>	подключить заголовочный файл delay.h;
main() {	основная часть программы;
#asm	начало кода ассемблерной вставки;
ldi R16,0b11111111;	загрузить в регистр R16 константу 0b11111111;
ldi R17,0b01010101;	загрузить в регистр R17 константу 0b11111111;
out 0x11,R16	настроить порт D на вывод данных;
#endasm	завершение кода ассемблерной вставки;
while (1) {	установить цикл с бесконечным числом итераций;

```

#asm("out 0x12,R16");    погасить все светодиоды;
delay_ms (1000);        установить временную задержку 1 с;
#asm ("out 0x12,R17");   включить 1–й, 3–й, 6–й и 8–й светодиоды;
delay_ms(2000); }       установить временную задержку 2 с;
}                         завершающая операторная скобка программы;

```

Таблица 1.4 – Варианты индивидуальных заданий

№ п.п.	Задание
1	Разработать программу, выполняющую в бесконечном цикле последовательное включение/выключение 1 – го, 3 – го и 6–го светодиодов с интервалом 2 с.
2	Разработать программу, выполняющую 5 раз подряд последовательное включение 2 – го, 4 – го и 6–го светодиодов с интервалом 1,5 с.
3	Разработать программу, выполняющую в бесконечном цикле последовательное включение/выключение 1 – го, 2 – го, 3 – го и 4–го светодиодов с интервалом 1 с.
4	Разработать программу, выполняющую в бесконечном цикле параллельное включение 1–го, 2–го, 7–го и 8–го светодиодов с длительностью свечения 2 с и временем нахождения в погашенном состоянии 1 с.
5	Разработать программу, выполняющую в бесконечном цикле параллельное включение 1–го, 2–го и 8–го светодиодов с длительностью свечения 1 с и временем нахождения в погашенном состоянии 2 с.
6	Разработать программу, выполняющую в бесконечном цикле последовательное включение/выключение всех светодиодов (с 1–го по 8–й) с интервалом 1,4 с.
7	Разработать программу, выполняющую в бесконечном цикле последовательное включение всех светодиодов (с 1–го по 8–й) с интервалом 0,5 с.
8	Разработать программу, выполняющую в бесконечном цикле последовательное включение/выключение 3–го, 4–го и 5–го светодиодов с интервалом 3 с.
9	Разработать программу, выполняющую в бесконечном цикле последовательное включение 5–го, 6–го, 7–го и 8–го светодиодов с интервалом 2.5 с.
10	Разработать программу, выполняющую в бесконечном цикле параллельное включение и выключение блока из 8-ми светодиодов с длительностью свечения 2 с и временем нахождения в погашенном состоянии 4 с.

1.4 Содержание отчета

В отчете необходимо привести следующее:
характеристики лабораторной вычислительной системы;
исходные модули разработанных программ;
анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности программного управления блоком светодиодов с помощью микроконтроллера AVR ATMEGA 128.

1.5 Контрольные вопросы и задания

1. Поясните основные особенности архитектуры микроконтроллера AVR ATMEGA 128.
2. Поясните принципы распределения адресных пространств памяти, регистров общего назначения и портов ввода/вывода в микроконтроллере AVR ATMEGA 128.
3. Каково назначение отдельных битов регистра состояния **SREG** ?
4. Поясните реализацию в микроконтроллере AVR ATMEGA 128 Гарвардской архитектуры и принципа конвейерной обработки команд.
5. Каким образом реализуется вызов операторов языка Assembler из C – программы ? Приведите примеры.
6. Дайте характеристику основным командам микроконтроллера AVR ATMEGA 128 при обращении к памяти и портам ввода/вывода.

2 ИЗУЧЕНИЕ РЕЖИМА ПРОГРАММНОГО ОПРОСА КЛАВИАТУРЫ

Цель работы: изучить принципы функционирования и подключения к микроконтроллеру матричной клавиатуры 3*4 лабораторного макета, разработать алгоритм и программу для реализации режима опроса клавиатуры без использования прерываний.

2.1 Указания по организации самостоятельной работы

Перед работой необходимо проработать теоретический материал по литературе [1,2] и конспект лекций, ознакомиться со структурной схемой и принципами функционирования клавиатуры лабораторного макета, возможностями реализации режима программного опроса клавиатуры лабораторного макета на базе микроконтроллера AVR ATMEGA128. При подготовке к лабораторной работе необходимо составить предварительные варианты листингов программ, указываемых в пунктах практического выполнения работы.

2.1.1. Принципы анализа нажатия стандартных кнопок с помощью микроконтроллера AVR MEGA128.

Параллельные входы микроконтроллеров часто используются для подключения различных коммутационных элементов: переключателей, кнопок, контактных блоков, которые служат для управления внешними устройствами. В простейшем случае кнопка подключается одним выводом к общему проводу, а другим – ко входной линии порта ввода/вывода, работающего в режиме ввода данных, и через резистор (сопротивлением порядка 10 кОм) с положительным полюсом источника электропитания (см. рисунок 2.1). При разомкнутых контактах кнопки на входной линии микроконтроллера установится уровень “логической единицы”, при замкнутых – “логического нуля”.

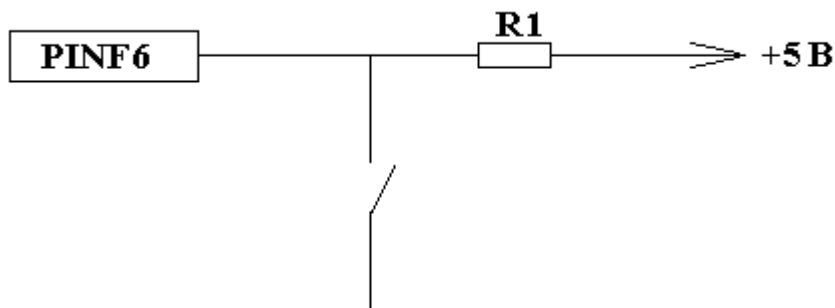


Рисунок 2.1 – Принципиальная схема подключения кнопки к микроконтроллеру

Принцип проверки состояния нажатия клавиши заключается в периодическом программном опросе входной линии, к которой подключен один из выводов клавиши, и анализе значения соответствующего бита. Рассмотрим фрагмент ассемблерного кода процедуры контроля состояния нажатия кнопки, подключенной к 6-й линии порта ввода/вывода F (адрес регистра ввода 0) согласно схеме на рисунке 2.1:

ldi R16,0b01000000	Загрузить константу для маскирования в R16;
lbl:	метка перехода для повторения опроса;
in R17,0	считать данные из регистра ввода порта F;
and R17,R16	обнулить все биты регистра R17, кроме 6-го.
cpi R17,0	сравнить значение в регистре R17 с нулем.
brne lbl	если клавиша не нажата – то выполнить переход для повторения опроса, иначе – выполнять дальнейшие действия.

2.1.2. Принципы считывания данных с матричной клавиатуры с помощью микроконтроллера AVR ATMEGA128 в режиме программного опроса.

При использовании большого количества кнопок управления целесообразно применить матричную схему подключения клавиатуры, сходную с приведенной на рисунке 2.2. В данной схеме 16-ти клавишная клавиатура 4×4 соединяется выводами с портом ввода/вывода E. Причем, линии 4 – 7 порта E настроены как выходные и обозначаются соответственно PORTE.4 – PORTE.7, а линии 0 – 3 – как входные (PINE.0 – PINE.3). Горизонтальные линии матрицы через токоограничительные резисторы подключены к положительному полюсу источника питания (+5 В).

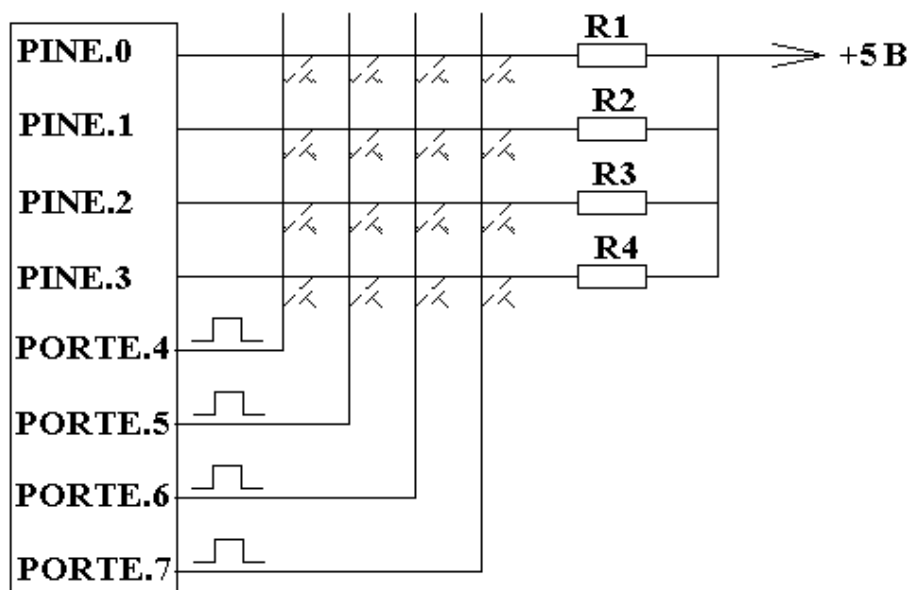


Рисунок 2.2 – Принципиальная схема подключения матричной клавиатуры к микроконтроллеру

Нажатие одной из клавиш замыкает в соответствующей позиции горизонтальную и вертикальную сигнальную линии. Если на вертикальную линию был подан уровень напряжения, соответствующий “логическому нулю”, то при нажатии клавиши на горизонтальной линии также установится низкий уровень напряжения. Алгоритм опроса нажатия клавиши сводится к поочередной установке низких уровней напряжения на вертикальных линиях (PORTE.4 – PORTE.7) матрицы (см. временные диаграммы управляющих сигналов на рисунке 2.3) и считывании информации об уровне сигнала на горизонтальных линиях (PINE.0 – PINE.3).

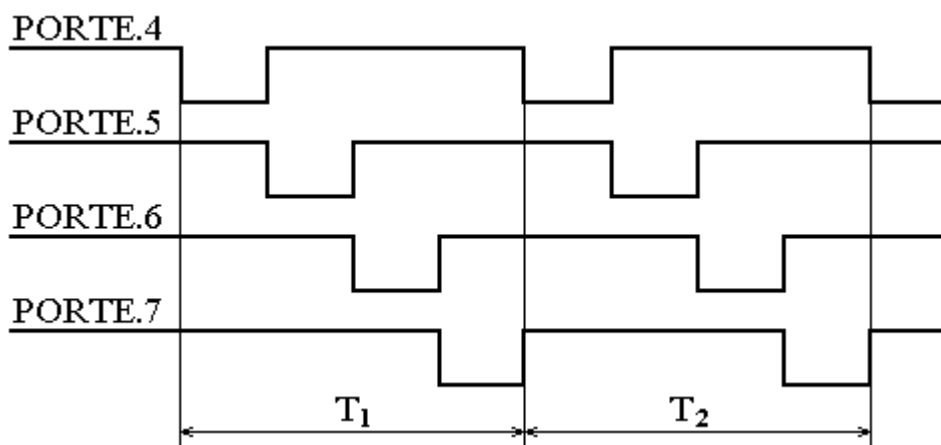


Рисунок 2.3 – Временные диаграммы сигналов на выходных линиях (PORTE.4 – PORTE.7) порта E при опросе матричной клавиатуры 4*4

Рассмотрим пример проверки нажатия одной из клавиш первого и второго (крайних справа) столбцов матричной клавиатуры, схема подключения которой приведена на рисунке 2.2 (адреса портов указаны в таблице 1.1):

```

...
ldi R19,0b11110000   Загрузить константу F0h в R19;
ldi R20,0b11100000   Загрузить константу E0h в R20;
ldi R21,0b11010000   загрузить константу D0h в R21;
ldi R22,0b00001111   загрузить константу 0Fh для маскирования в R16;
out 02,R19           настройка линий E0-3 на ввод, а E4-7 – на вывод;
lbl1:                метка перехода для повторения опроса;
out 03,R20           установить в 0 уровень напряжения на линии PORTE.7,
                    соответствующей первому столбцу клавиатуры;
nop                 установить задержку в один такт;
in R17,01           считать данные из регистра PINE в R18;
out 03,R21           установить в 0 уровень напряжения на линии PORTE.6,
                    соответствующей второму столбцу клавиатуры;
nop                 установить задержку в один такт;

```


in R18,01	считать данные из регистра PINE в R18;
and R17,R22	обнулить неинформативную старшую тетраду в R17;
and R18,R22	обнулить неинформативную старшую тетраду в R18;
cp R17,R22	сравнить значение в регистре R17 со значением 0Fh;
brne lbl2	выполнить переход на метку lbl2, если R18 \neq 0Fh (одна или несколько клавиш первого столбца клавиатуры нажаты);
cp R18,R22	сравнить значение в регистре R18 со значением 0Fh.
brne lbl2	выполнить переход на метку lbl2, если R18 \neq 0Fh (одна или несколько клавиш второго столбца клавиатуры нажаты);
rjmp lbl1	переход на метку lbl1 для повторения процедуры опроса;
lbl2:	метка выхода из процедуры опроса.
...	

Если в результате процедуры опроса в битах младших тетрад регистров R17 и/или R18 будут находиться 0, то это будет свидетельствовать о нажатии клавиш, позиции которых можно определить исходя из схемы, изображенной на рисунке 2.1, и номеров обнуленных разрядов в тетрадах.

Процедуру опроса матричной клавиатуры 3×4 необходимо выполнять только с использованием ассемблерных команд, которые (в отличие от их эквивалентов на языке C) не преобразуют значения битов в старших тетрадах порта E.

2.2 Описание лабораторной установки

Лабораторная работа выполняется в индивидуальном порядке. На каждом рабочем месте должны быть установлены: многофункциональный лабораторный макет на базе микроконтроллера AVR ATMEGA 128, ПЭВМ типа IBM PC/AT с установленным программным обеспечением: операционной системой MS–WINDOWS v. 9x, 2000, XP и программатором на основе кросс-компилятора языка программирования C CodeVision AVR. Задания выполняются на лабораторном макете на базе 8-ми разрядного микроконтроллера AVR ATMEGA 128. Подробное описание лабораторного макета приведено в пункте 1.2 лабораторной работы № 1.

В данной работе основным используемым периферийным оборудованием лабораторного макета будут 3-х кнопочная клавиатура 3×1 и матричная клавиатура 3×4 (см. рисунок 2.4), подключаемые к микроконтроллеру через порты F и E соответственно. Для удобства на рисунке 2.4. проведено обозначение номеров клавиш, а так же индексов столбцов $C_1 - C_3$ и строк $R_1 - R_3$ (для матричной клавиатуры 3×4). Принципиальные схемы подключения клавиатур 3×1 и 3×4 приведены на рисунках 2.5 и 2.6 соответственно.

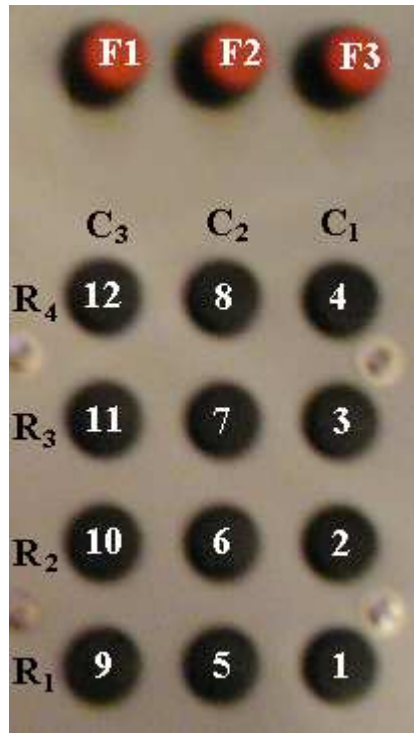


Рисунок 2.4 – Общий вид 3-х кнопочной 3×1 (вверху) и 12-и кнопочной 3×4 матричной (внизу) клавиатуры лабораторного макета

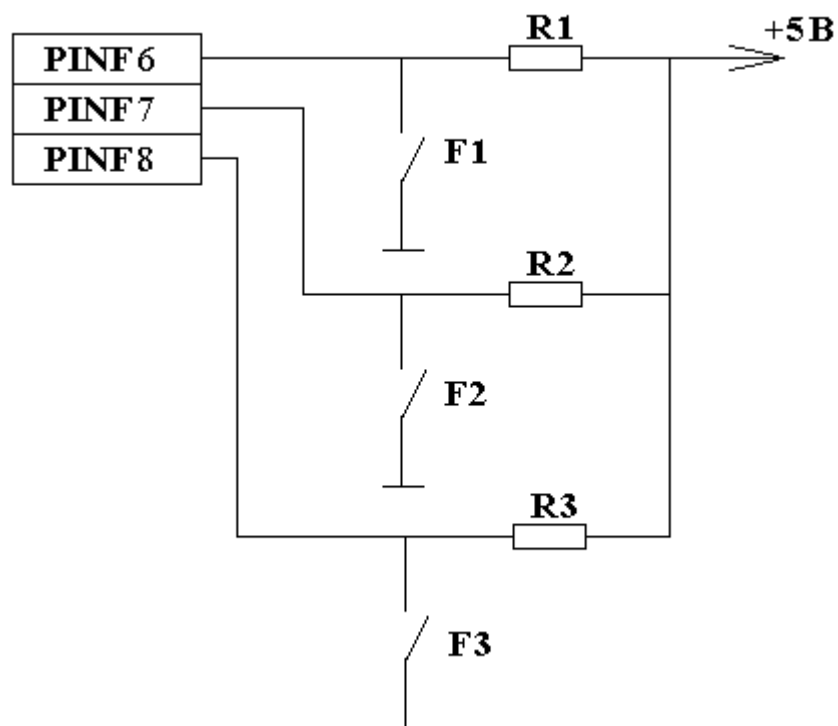


Рисунок 2.5 – Принципиальная схема подключения 3-х кнопочной (3×1) клавиатуры ко входам 6-8 порта F микроконтроллера AVR MEGA 128

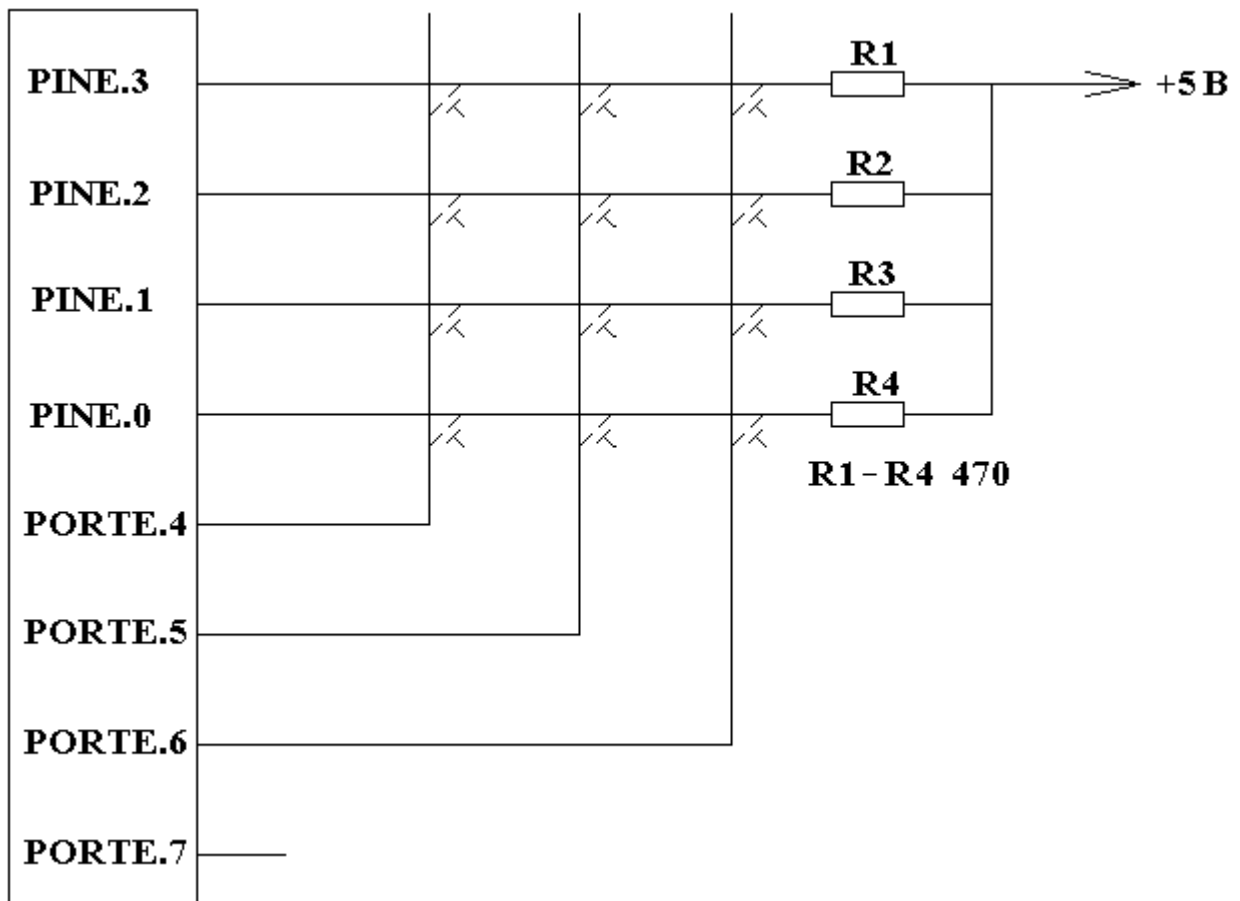


Рисунок 2.6 – Принципиальная схема подключения 12-и кнопочной (3×4) клавиатуры к микроконтроллеру AVR ATMEGA 128 через порт ввода/вывода E

2.3 Порядок проведения работы и указания по ее выполнению.

Перед началом выполнения практической части лабораторной работы проводится экспресс-контроль знаний по принципам функционирования микроконтроллера AVR ATMEGA 128, системе команд и возможностям организации программного опроса клавиатуры лабораторного макета.

При подготовке к лабораторной работе необходимо составить предварительный вариант листинга программы, в соответствие с индивидуальным заданием (см. таблицу 2.1).

Задание. Разработать в среде программирования Code Vision AVR программу для микроконтроллера AVR ATMEGA 128, которая выполняет опрос клавиатуры лабораторного макета и выводит информацию о нажатых клавишах с помощью блока светодиодов. Обозначения клавиш приведены на рисунке 2.4. Варианты индивидуальных заданий представлены в таблице 2.1.

Порядок выполнения задания:

1. Включить лабораторный макет (установить выключатель электропитания в положение I, и убедиться в свечении индикатора электропитания красным цветом).
2. Запустить компилятор Code Vision AVR.
3. Создать пустой проект.
4. Создать файл ресурса для кода программы и подключить его к проекту.
5. Ввести код исходного модуля программы считывания данных с клавиатуры лабораторного макета в соответствие с вариантом задания, указанном в таблице 2.1.
6. Выполнить компиляцию (нажав клавишу **F9**) исходного модуля программы и устранить ошибки, полученные на данном этапе.
7. Настроить параметры программатора.
8. Создать загрузочный модуль программы (нажав комбинацию клавиш **Shift+F9**) и выполнить программирование микроконтроллера.
9. Проверить работоспособность загруженной в микроконтроллер программы и показать результаты работы преподавателю.
10. В случае некорректной работы разработанной программы, выполнить аппаратный сброс микроконтроллера, провести отладку исходного модуля программы и заново проверить функционирование программы, повторив выполнение пункта 9.

Пример выполнения задания. Разработать программу, выводящую информацию об индексах нажатых клавиш первого и второго столбцов (C_1 и C_2) матричной клавиатуры 3×4 (зажигается светодиод, соответствующий номеру нажатой клавиши), и осуществляющую выход из режима опроса при нажатии кнопки F_3 клавиатуры 3×1 .

Решение. В лабораторном макете матричная клавиатура 3×4 подключена к порту E микроконтроллера. Линии старшей тетрады порта E настраиваются на вывод данных, а линии младшей тетрады – на ввод. Кнопки F_1 , F_2 , F_3 к 6-му, 7-му и 8-му разрядам порта F, который не нужно специально настраивать на ввод данных. Блок светодиодов подключен к порту D микроконтроллера и настраивается на вывод данных. Цикл опроса состоит в последовательном считывании данных с 1-го и 2-го столбцов матричной клавиатуры 3×4 при соответствующих управляющих сигналах, вывода информации о позиции нажатой клавиши на блок светодиодов и проверки состояния нажатия кнопки F_1 , при нажатии на которую процедура опроса клавиатуры завершается. Алгоритм программы приведен на рисунке 2.7. Полный текст исходного модуля программы с подробными комментариями приводится ниже:

```
#include <mega128.h> Подключить заголовочный файл mega128.h;
main() {           основная часть программы;
#asm              начало кода ассемблерной вставки;
```

ldi R18,0b11100000	запись маскирующих констант в регистры;
ldi R19,0b11010000	
ldi R20,0b11110000	
ldi R21,0b00001111	
ldi R22,0b10000000	
ldi R23,0b11111111	
ldi R24,0b00000000	
out 02,R20	настройка линий E ₀₋₃ на ввод, а E ₄₋₇ – на вывод;
out 0x11,R23	установить режим вывода для порта D;
out 0x12, R23	погасить все светодиоды;
loop1:	стартовая метка цикла опроса;
out 03,R18	обнулить бит E ₄ для опроса клавиш 1-го столбца;
nop	установить задержку в один такт;
in R16,01	считать данные из регистра PINE в R16;
out 03,R19	обнулить бит E ₅ для опроса клавиш 2-го столбца;
nop	установить задержку в один такт;
in R17,01	считать данные из регистра PINE в R16;
or R16,R20	установить в 1 биты старшей тетрады R16;
or R17,R20	установить в 1 биты старшей тетрады R17;
swap R17	поменять местами старшую и младшую тетрады в R17;
and R17,R16	объединить информацию о нажатии клавиш в R17;
out 0x12,R17	вывести данные о номерах нажатых клавиш в порт D;
in R26,0	считать данные из регистра ввода порта F;
and R26,R22	обнулить все биты регистра R26, кроме 7-го.
cpi R26,0	сравнить значение в регистре R26 с нулем.
brne loop1	если клавиша не нажата – то выполнить переход для повторения опроса, иначе –
out 0x12,r24	выполнить команду включить все светодиоды;
#endasm	завершение кода ассемблерной вставки;
}	завершающая операторная скобка программы.

Для установки и обнуления битов регистров ввода/вывода можно использовать команды `sbi` и `cbi` соответственно. Однако эти команды необходимо будет использовать попарно, что ухудшает наглядность исходного кода программы.



Рисунок 2.7 – Алгоритм программы считывания данных с клавиатуры

Таблица 2.1 – Варианты индивидуальных заданий

№ п.п.	Задание
1	Разработать программу, фиксирующую нажатия клавиш 1, 6 и 12 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши F1.
2	Разработать программу, фиксирующую нажатия клавиш 3, 7 и 11 матричной клавиатуры включением светодиодов 5, 6 и 7 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши F2.
3	Разработать программу, фиксирующую нажатия клавиш 4, 7 и 10 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши F1.
4	Разработать программу, фиксирующую нажатия клавиш 1, 2 и 3 матричной клавиатуры включением светодиодов 4, 5 и 6 соответственно. Выход из цикла опроса осуществляется при одновременном нажатии клавиш F1+F3.
5	Разработать программу, фиксирующую нажатия клавиш 2, 5 и 9 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при одновременном нажатии клавиш F1+F2.
6	Разработать программу, фиксирующую нажатия клавиш 4, 8 и 11 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши F3.
7	Разработать программу, фиксирующую нажатия клавиш 3, 7 и 10 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши F1.
8	Разработать программу, фиксирующую нажатия клавиш 5, 6 и 7 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при одновременном нажатии клавиш F2+F3.
9	Разработать программу, фиксирующую нажатия клавиш 1, 6 и 8 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши F2.
10	Разработать программу, фиксирующую нажатия клавиш 9, 10 и 11 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши F1.

2.4 Содержание отчета

В отчете необходимо привести следующее:

характеристики лабораторной вычислительной системы;

исходный модуль разработанной программы;

анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности считывания данных с матричной клавиатуры в режиме программного обмена с помощью микроконтроллера AVR ATMEGA 128.

2.5 Контрольные вопросы и задания

1. Поясните принцип функционирования матричной клавиатуры 3*4, входящей в состав лабораторного макета.
2. Поясните алгоритм программного опроса клавиатуры.
3. Возможно ли обращение к портам ввода/вывода как к ячейкам памяти в микроконтроллере AVR ATMEGA ?
4. Поясните принцип работы клавиатуры 1*3, входящей в состав лабораторного макета.
5. В чем заключаются преимущества и недостатки режима программного опроса клавиатуры ?
6. Перечислите и поясните основные команды и приемы для логического анализа двоичных данных. AVR ATMEGA 128 при обращении к памяти и портам ввода/вывода.

3 ИЗУЧЕНИЕ ПРИНЦИПОВ ПРОГРАММНОГО УПРАВЛЕНИЯ ВНЕШНИМИ УСТРОЙСТВАМИ НА ПРИМЕРЕ ВЫВОДА ИНФОРМАЦИИ НА ЦИФРОВОЙ ИНДИКАТОР

Цель работы: изучить принципы функционирования и возможности программного управления цифровым индикатором, разработать алгоритм и программу для вывода информации на цифровой индикатор.

3.1 Указания по организации самостоятельной работы

Перед работой необходимо проработать теоретический материал по литературе [1,2] и конспект лекций, ознакомиться с принципами функционирования и возможностями программного управления выводом символов на экран 10 позиционного цифрового ЖК-индикатора LCM-10 с использованием микроконтроллера AVR ATMEGA128.

Принципы программного управления выводом символов на экран цифрового индикатора с помощью микроконтроллера AVR ATMEGA128. В лабораторной работе используется 10-ти позиционный жидкокристаллический (LED) цифровой индикатор на базе контроллера HT1613 фирмы Holtek (см. рисунок 3.1), для подключения которого используется порт ввода/вывода В (адреса регистров порта приводятся в таблице 1.1). Выходной сигнал с 0-го бита порта В (PORTB.0) управляет тактовым входом **SK** цифрового индикатора, а сигнал с 1-го бита порта В (PORTB.1) – входом данных **DI** (см. рисунок 3.2).



Рисунок 3.1 – Схематическое изображение передней панели 10-позиционного цифрового индикатора

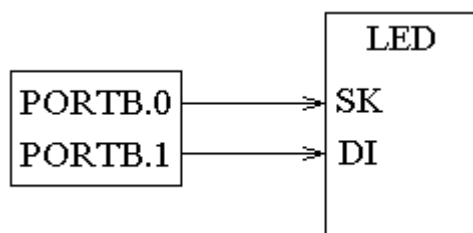


Рисунок 3.2 – Схема подключения информационных входов цифрового индикатора к лабораторному макету

ЖК-индикатор позволяет осуществлять вывод цифровых и специальных символов, кодируемых 4-мя разрядами (см. таблицу 3.1). Код символа передается последовательно в виде пакета, состоящего из 4-х битов b_3 , b_2 , b_1 , b_0 (первым – старший, последним – младший биты). Сигналы данных в пакете подаются на вход **DI** цифрового индикатора и защелкиваются по заднему фронту тактирующих импульсов на тактовом входе **SK**. При этом временной интервал T_a с момента установки стабильного уровня сигнала данных до появления заднего фронта тактирующего сигнала не должен быть меньше 50 нс, временной интервал T_b между задним и передним фронтами тактирующих импульсов должен быть не менее 60 нс, а временной интервал T_c между пакетами данных – не менее 80 нс при длительности пакета порядка 500 нс (см. рисунок 3.3).

Символы выводятся в крайнюю правую позицию экрана индикатора. При загрузке следующего символа остальные сдвигаются на одну позицию влево, при этом крайний слева символ стирается.

Таблица 3.1 – Кодировка символов, выводимых на экран цифрового индикатора

Биты данных				Символ
b_3	b_2	b_1	b_0	
0	0	0	0	табуляция
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	0
1	0	1	1	F
1	1	0	0	┐
1	1	0	1	└
1	1	1	0	P
1	1	1	1	–

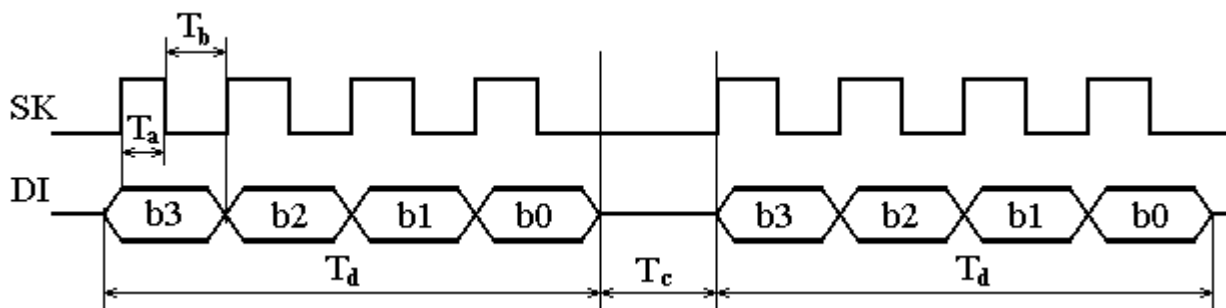


Рисунок 3.3 – Временные диаграммы пакетного цикла передачи кода символа для вывода на экран цифрового индикатора

В качестве примера рассмотрим фрагмент программы вывода на экран цифрового индикатора символа **6**, 4-х битовый код которого (**0110**) приведен в 7-й строке таблицы 3.1.

...

#asm

ldi R18,0xFF настроить порт В на вывод данных;
out 0x17, R18

cbi 0x18,1 вывести значение 3-го бита данных в 1-й разряд порта В;
sbi 0x18,0 установить в 1 уровень тактирующего сигнала;
cbi 0x18,0 сбросить в 0 уровень тактирующего сигнала;

sbi 0x18,1 вывести значение 2-го бита данных в 1-й разряд порта В;
sbi 0x18,0 установить в 1 уровень тактирующего сигнала;
cbi 0x18,0 сбросить в 0 уровень тактирующего сигнала;

sbi 0x18,1 вывести значение 1-го бита данных в 1-й разряд порта В;
sbi 0x18,0 установить в 1 уровень тактирующего сигнала;
cbi 0x18,0 сбросить в 0 уровень тактирующего сигнала;

cbi 0x18,1 вывести значение 0-го бита данных в 1-й разряд порта В;
sbi 0x18,0 установить в 1 уровень тактирующего сигнала;
cbi 0x18,0 сбросить в 0 уровень тактирующего сигнала;

#endasm

...

Приведенный фрагмент программы показывает, что при выводе символа на экран цифрового индикатора необходимо выполнить передачу битов данных по линии, подключенной к 1-му разряду порта В, при подаче тактирующих сигналов по линии, подключенной к 0-му разряду порта В. Так, как тактовая частота микроконтроллера ATMEGA 128, входящего в состав лабораторного макета, равна 11 МГц, то длительность цикла команды,

выполняемой за один такт, равна 90 нс, что позволяет выполнять передачу данных без введения дополнительных задержек.

3.2 Описание лабораторной установки

Лабораторная работа выполняется в индивидуальном порядке. На каждом рабочем месте должны быть установлены: многофункциональный лабораторный макет на базе микроконтроллера AVR ATMEGA 128, ПЭВМ типа IBM PC/AT с установленным программным обеспечением: операционной системой MS-WINDOWS v. 9x, 2000, XP и программатором на основе кросс-компилятора языка программирования C CodeVision AVR. Задания выполняются на лабораторном макете на базе 8-ми разрядного микроконтроллера AVR ATMEGA 128. Подробное описание лабораторного макета приведено в пункте 1.2 лабораторной работы № 1.

В данной работе основным используемым периферийным оборудованием лабораторного макета будут матричная клавиатура 3×4 (см. рисунок 2.4), подключаемая к микроконтроллеру через порт E, и цифровой 10-ти позиционный жидкокристаллический индикатор (см рисунок 3.4), управляющая часть которого выполнена на основе контроллера Holtek HT1613. Индикатор подключается к лабораторному макету с помощью 4-х проводного кабеля с разъемом DB-25. Тактовый **SK** и информационный **DI** входы индикатора подключены к 0-му и 1-му выходам порта ввода/вывода В микроконтроллера соответственно; две остальные линии подключаются к источнику питания +5 В и общему проводу.

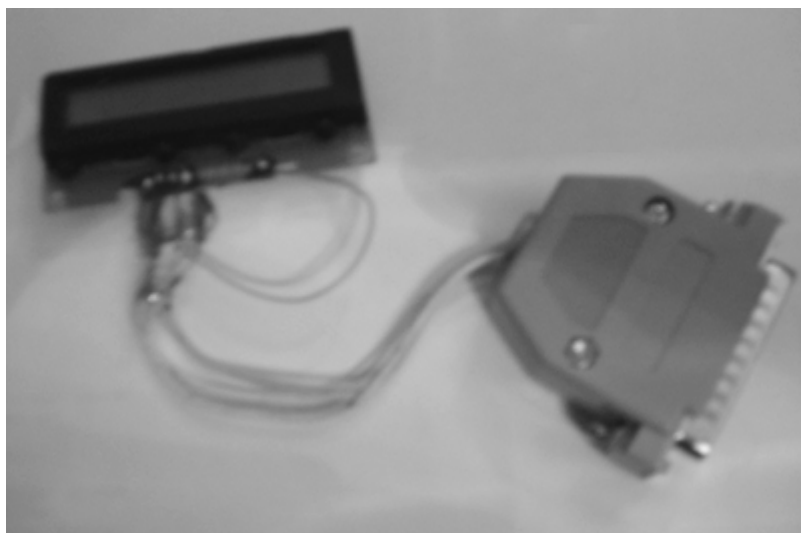


Рисунок 3.4 – Общий вид панели цифрового индикатора и интерфейсного разъема

3.3 Порядок проведения работы и указания по ее выполнению

Перед началом выполнения практической части лабораторной работы проводится экспресс–контроль знаний по принципам функционирования микроконтроллера AVR ATMEGA 128, системе команд и возможностям организации программного управления выводом символов на экран цифрового индикатора.

При подготовке к лабораторной работе необходимо составить предварительный вариант листинга программы, в соответствие с индивидуальным заданием (см. таблицу 3.2).

Задание. Разработать в среде программирования Code Vision AVR программу для микроконтроллера AVR ATMEGA 128, которая выполняет опрос клавиатуры лабораторного макета и выводит информацию о нажатых клавишах (или комбинациях клавиш) на экран цифрового индикатора. Обозначения клавиш приведены на рисунке 2.4. Варианты индивидуальных заданий представлены в таблице 2.1.

Порядок выполнения задания:

1. Включить лабораторный макет (установить выключатель электропитания в положение I, и убедиться в свечении индикатора электропитания красным цветом).
2. Запустить компилятор Code Vision AVR.
3. Создать пустой проект.
4. Создать файл ресурса для кода программы и подключить его к проекту.
5. Ввести код исходного модуля программы вывода информации на экран цифрового индикатора в соответствии с кодом нажатой клавиши согласно варианту задания, указанному в таблице 3.2.
6. Выполнить компиляцию (нажав клавишу **F9**) исходного модуля программы и устранить ошибки, полученные на данном этапе.
7. Настроить параметры программатора.
8. Создать загрузочный модуль программы (нажав комбинацию клавиш **Shift+F9**) и выполнить программирование микроконтроллера.
9. Проверить работоспособность загруженной в микроконтроллер программы и показать результаты работы преподавателю.
10. В случае некорректной работы разработанной программы, выполнить аппаратный сброс микроконтроллера, провести отладку исходного модуля программы и заново проверить функционирование программы, повторив выполнение пункта 9.

Пример выполнения задания. Разработать программу, выводящую на экран цифрового индикатора символ 1 при нажатии на клавишу **1**, и символ 2 при нажатии комбинации клавиш **1+2**. Нумерация клавиш матричной клавиатуры 3×4 приведена на рисунке 2.4.

Решение: в лабораторном макете матричная клавиатура 3×4 подключена к порту E микроконтроллера. Линии старшей тетрады порта E настраиваются на вывод данных, а линии младшей тетрады – на ввод. Входы цифрового индикатора подключены к выходам порта В микроконтроллера в соответствие со схемой на рисунке 3.2. Порт В настраивается на вывод данных. Основная часть программы в соответствии с алгоритмом, изображенным на рисунке 3.5, содержит цикл опроса клавиатуры, проверку нажатия клавиши 1 или комбинации клавиш 1+2 и процедуры вывода символов 1 и 2 на индикатор.

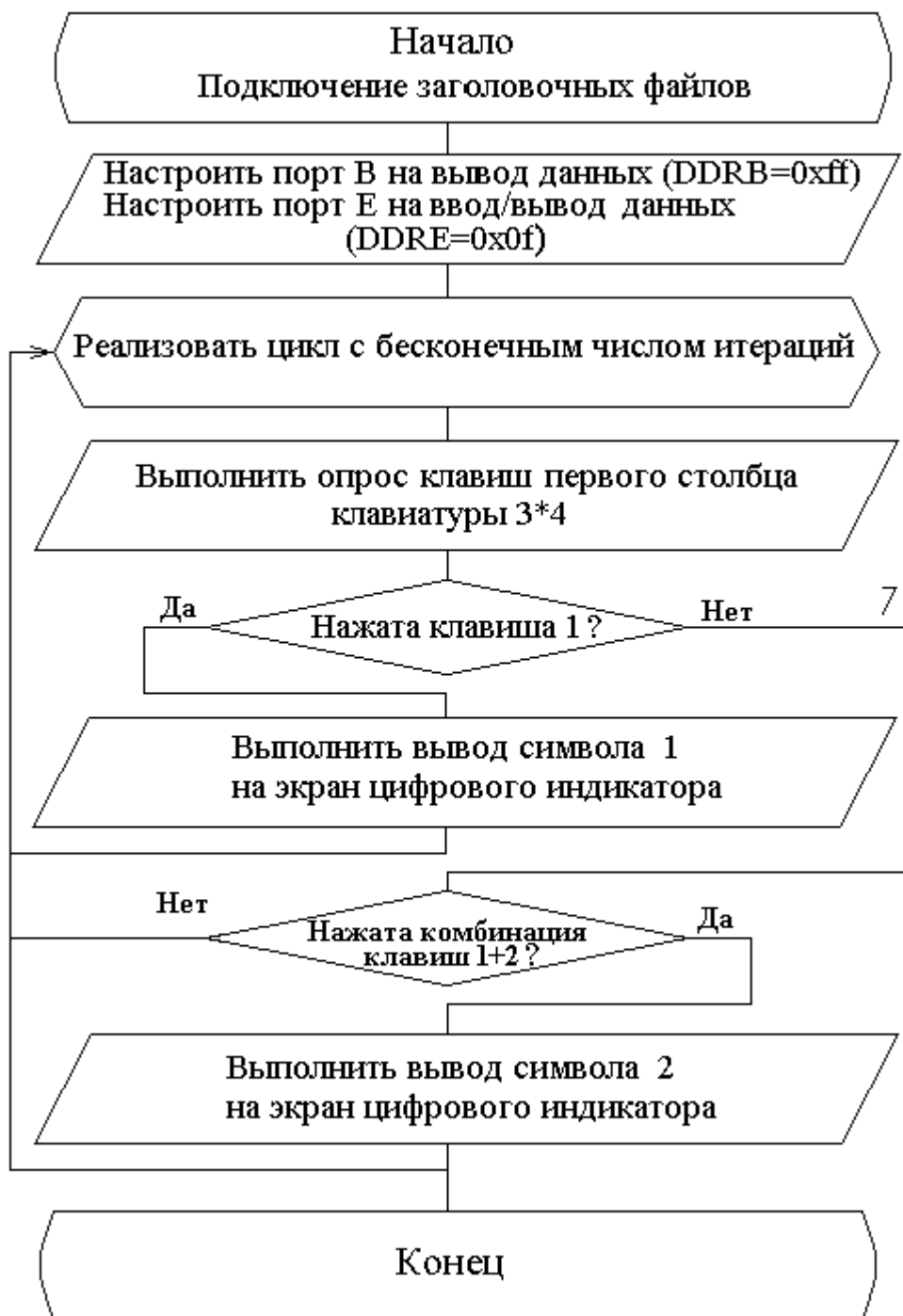


Рисунок 3.5 – Алгоритм программы вывода символа на экран цифрового

индикатора в зависимости от данных, принятых с клавиатуры

Для того, чтобы символы каждый раз заносились в крайнюю правую позицию экрана индикатора, а введенные ранее символы не отображались, необходимо непосредственно перед выводом информационного символа выполнить вывод девяти символов табуляции. Полный текст исходного модуля программы с подробными комментариями приводится ниже:

```
#include <mega128.h> Подключить заголовочный файл mega128.h;  
main() { основная часть программы;  
#asm начало кода ассемблерной вставки;  
ldi R18,0b11100000 запись маскирующих констант в регистры;  
ldi R19,0b11111111  
ldi R20,0b11110000  
out 02,R20 настройка линий E0-3 на ввод, а E4-7 – на вывод;  
out 0x17, R19 установить режим вывода для порта В;  
out 03,r18 обнулить бит E4 для опроса клавиш 1-го столбца;  
loop1: стартовая метка цикла опроса;  
in r16,01 считать данные из регистра PINE в R16;  
or r16,r20 применить маску для выделения данных в младшей тетраде регистра R16;  
cp R16,0b11111110 проверить нажатие клавиши 1;  
brne lbl если клавиша 1 не нажата, то переход на метку lbl для продолжения процедуры проверки;  
ldi R27,9 организовать цикл с параметром в R27 из 9 итераций;  
lblc: стартовая метка цикла с параметром;  
вывод символа табуляции с кодом 0000;  
cbi 0x18,1 бит b3;  
sbi 0x18,0  
cbi 0x18,0  
cbi 0x18,1 бит b2;  
sbi 0x18,0  
cbi 0x18,0  
cbi 0x18,1 бит b1;  
sbi 0x18,0  
cbi 0x18,0  
cbi 0x18,1 бит b0;  
sbi 0x18,0  
cbi 0x18,0  
dec R27 завершающий код цикла из 9 итераций;  
cp R27,0  
brne lblc:  
вывод символа 1 на экран цифрового индикатора;  
cbi 0x18,1 бит b3;
```

sbi 0x18,0	
cbi 0x18,0	
cbi 0x18,1	бит b2;
sbi 0x18,0	
cbi 0x18,0	
cbi 0x18,1	бит b1;
sbi 0x18,0	
cbi 0x18,0	
sbi 0x18,1	бит b0;
sbi 0x18,0	
cbi 0x18,0	
rjmp vihod	выполнить переход для завершения текущей итерации цикла;
lbl:	метка фрагмента проверки нажатия комбинации клавиш 1+2;
cpi R16, 0b11111100	выполнить проверку нажатия комбинации клавиш 1+2;
brne vihod	если комбинация клавиш 1+2 не нажата, то переход на метку vihod для выхода из текущей итерации цикла;
ldi R27,9	организовать цикл с параметром в R27 из 9 итераций;
lbc2:	стартовая метка цикла с параметром;
вывод символа табуляции с кодом 0000;	
cbi 0x18,1	бит b3;
sbi 0x18,0	
cbi 0x18,0	
cbi 0x18,1	бит b2;
sbi 0x18,0	
cbi 0x18,0	
cbi 0x18,1	бит b1;
sbi 0x18,0	
cbi 0x18,0	
cbi 0x18,1	бит b0;
sbi 0x18,0	
cbi 0x18,0	
dec R27	завершающий код цикла из 9 итераций;
cpi R27,0	
brne lbc2:	
вывод символа 2 на экран цифрового индикатора;	
cbi 0x18,1	бит b3;
sbi 0x18,0	
cbi 0x18,0	
cbi 0x18,1	бит b2;
sbi 0x18,0	
cbi 0x18,0	
sbi 0x18,1	бит b1;


```
sbi 0x18,0
cbi 0x18,0
cbi 0x18,1
sbi 0x18,0
cbi 0x18,0
```

бит b0;

```
vihod:
rjmp loop1
#endasm
}
```

метка завершения кода текущей итерации цикла опроса;
 выполнить переход к новой итерации цикла опроса;
 завершение кода ассемблерной вставки;
 завершающая операторная скобка программы;

Таблица 3.2 – Варианты индивидуальных заданий*

№ п.п.	Задание
1	Разработать программу, выводящую на экран цифрового индикатора символ 8 при нажатии на клавишу 8 , и символ 9 при нажатии комбинации клавиш 5+6 .
2	Разработать программу, выводящую на экран цифрового индикатора символы 2 и 3 при нажатии на клавиши 2 и 3 соответственно.
3	Разработать программу, выводящую на экран цифрового индикатора символ 5 при нажатии на клавишу 12 , и символ 6 при нажатии комбинации клавиш 9+10+11 .
4	Разработать программу, выводящую на экран цифрового индикатора символы 3 и 4 при нажатии на клавиши 3 и 4 соответственно.
5	Разработать программу, выводящую на экран цифрового индикатора символ 8 при нажатии на клавишу 11 , и символ 7 при нажатии комбинации клавиш 9+10 .
6	Разработать программу, выводящую на экран цифрового индикатора символ 0 при нажатии на клавишу 1 , и символ 1 при нажатии комбинации клавиш 1+2+3+4 .
7	Разработать программу, выводящую на экран цифрового индикатора символы 6 и 7 при нажатии на клавиши 6 и 7 соответственно.
8	Разработать программу, выводящую на экран цифрового индикатора символ 5 при нажатии на клавишу 5 , и символ 6 при нажатии комбинации клавиш 5+6+7 .
9	Разработать программу, выводящую на экран цифрового индикатора символ 7 при нажатии на клавишу 7 , и символ 8 при нажатии комбинации клавиш 5+6 .
10	Разработать программу, выводящую на экран цифрового индикатора символ 4 при нажатии на клавишу 12 , и символ 5 при нажатии комбинации клавиш 11+12 .

* Нумерация клавиш матричной клавиатуры 3×4 приведена на рисунке 2.4.

3.4 Содержание отчета

В отчете необходимо привести следующее:
характеристики лабораторной вычислительной системы;
исходный модуль разработанной программы;
анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности управления выводом данных на экран цифрового индикатора с помощью микроконтроллера AVR ATMEGA 128.

3.5 Контрольные вопросы и задания

1. Поясните принцип функционирования цифрового индикатора, подключаемого к лабораторному макету.
2. Поясните алгоритм программного управления контроллером цифрового индикатора.
3. Каким образом можно осуществлять вывод информации на цифровой индикатор в фиксированные позиции ?
4. Поясните принципы использования команд установки и сброса отдельных битов; приведите примеры.
5. Приведите алгоритм универсальной программы управления цифровым индикатором.
6. Каким образом можно формировать сигналы заданной длительности на выходных линиях портов ввода/вывода ?

4 ИЗУЧЕНИЕ ПРИНЦИПОВ ОБРАБОТКИ ПРЕРЫВАНИЙ НА ПРИМЕРЕ УПРАВЛЕНИЯ ВСТРОЕННЫМИ В МИКРОКОНТРОЛЛЕР ТАЙМЕРАМИ–СЧЕТЧИКАМИ

Цель работы: изучить принципы разработки процедур обработки прерываний в микроконтроллере AVR ATMEGA128, ознакомиться с принципами функционирования встроенных в микроконтроллер 8 и 16 – разрядных таймеров – счетчиков.

4.1 Указания по организации самостоятельной работы

Перед работой необходимо проработать теоретический материал по литературе [1 – 3] и конспект лекций, изучить принципы разработки процедур обработки прерываний в микроконтроллере AVR ATMEGA 128, ознакомиться с возможностями функционирования встроенных в микроконтроллер 8 и 16 - разрядных таймеров – счетчиков, изучить алгоритмы формирования временных задержек с помощью прерываний от таймеров – счетчиков.

4.1.1. Система прерываний в микроконтроллере AVR ATMEGA 128.

Прерывание – процесс, нарушающий выполнение нормального хода программы. Прерывания инициируются внутренними или внешними событиями микроконтроллера. При возникновении прерывания микроконтроллер сохраняет в стеке содержимого счетчика команд PC и загружает в него адрес соответствующего вектора прерывания, в котором, как правило, содержится команда безусловного перехода к подпрограмме обработки прерывания. Последней командой подпрограммы – обработчика прерываний должна быть команда **reti**, которая обеспечивает возврат в основную программу путем восстановления значения предварительно сохраненного счетчика команд.

Вектор прерывания представляет собой адрес процедуры обработки прерывания. Вектора прерываний от разных источников объединены структуру, называемую таблицей векторов прерываний. В микроконтроллере AVR ATMEGA 128 таблица векторов прерываний находится, начиная с адреса 0002h. Положение вектора в таблице прерываний определяет приоритет соответствующего прерывания, который уменьшается с увеличением адреса в таблице прерывания (чем меньше адрес – тем выше приоритет). Размещение векторов прерываний микроконтроллера AVR ATMEGA 128 приводится в таблице 4.1. Регистр состояния SREG аппаратно не обрабатывается процессором, как при вызове подпрограмм, так и при обслуживании прерываний. Если программа требует сохранения SREG, то это должно производиться программой пользователя.

Таблица 4.1 – Номера векторов прерываний и идентификаторы процедур-обработчиков прерываний микроконтроллера AVR ATMEGA 128.

Номер вектора	Адрес	Источник прерывания	Идентификатор прерывания	Описание прерывания
1	0000h	RESET		Вывод сброса, отключение электропитания, сброс от сторожевого таймера
2	0002h	INT0	EXT_INT0	Внешнее прерывание по линии запроса 0
3	0004h	INT1	EXT_INT1	Внешнее прерывание по линии запроса 1
4	0006h	INT2	EXT_INT2	Внешнее прерывание по линии запроса 2
5	0008h	INT3	EXT_INT3	Внешнее прерывание по линии запроса 3
6	000Ah	INT4	EXT_INT4	Внешнее прерывание по линии запроса 4
7	000Ch	INT5	EXT_INT5	Внешнее прерывание по линии запроса 5
8	000Eh	INT6	EXT_INT6	Внешнее прерывание по линии запроса 6
9	0010h	INT7	EXT_INT7	Внешнее прерывание по линии запроса 7
10	0012h	TIMER2 COMP	TIM2_COMP	Совпадение таймера/счетчика T2
11	0014h	TIMER2_OVF	TIM2_OVF	Переполнение таймера/счетчика T2
12	0016h	TIMER1 CAPT1	TIM1_CAPT	Захват таймера/счетчика T1
13	0018h	TIMER1 COMP1	TIM1_COMPA	Совпадение «А» таймера/счетчика T1
14	001Ah	TIMER1 COMPB	TIM1_COMPB	Совпадение «В» таймера/счетчика T1
15	001Ch	TIMER1 OVF	TIM1_OVF	Переполнение таймера/счетчика T1
16	001Eh	TIMER0 COMP	TIM0_COMP	Совпадение таймера/счетчика T0
17	0020h	TIMER0 OVF	TIM0_OVF	Переполнение таймера/счетчика T0
18	0022h	SPI STC	SPI_STC	Передача данных по SPI закончена
19	0024h	USART0 RXC	USART0_RXC	Прием по интерфейсу UART0 завершен
20	0026h	USART0 DRE	USART0_DRE	Регистр данных UART0 пуст
21	0028h	USART0 TXC	USART0_TXC	Передача по USART0 завершена
22	002Ah	ADC INT	ADC_INT	Преобразование АЦП завершено
23	002Ch	EE RDY	EE_RDY	Прерывание при готовности EEPROM
24	002Eh	ANA COMP	ANA_COMP	Прерывание от аналогового компаратора

25	0030h	TIMER1 COMPC	TIM1_COMPC	Совпадение «С» таймера-счетчика T1
26	0032h	TIMER3 CAPT	TIM3_CAPT	Захват таймера-счетчика T3
27	0034h	TIMER3_COMPA	TIM3_COMPA	Совпадение «А» таймера-счетчика T3
28	0036h	TIMER3_COMPB	TIM3_COMPB	Совпадение «В» таймера-счетчика T3
29	0038h	TIMER3_COMPC	TIM3_COMPC	Совпадение «В» таймера-счетчика T3
30	003Ah	TIMER3_OVF	TIM3_OVF	Переполнение таймера-счетчика T3
31	003Ch	USART1_RXC	USART1_RXC	Прием по интерфейсу UART1 завершен
32	003Eh	USART1_DRE	USART1_DRE	Регистр данных UART1 пуст
33	0040h	USART1_TXC	USART1_TXC	Передача по USART1 завершена
34	0042h	TWI	TWI	Прерывание от модуля TWI
35	0044h	SPM_RDY	SPM_RDY	Готовность SPM

Функция – обработчик прерывания записывается в компиляторе Code Vision AVR C в соответствии со следующими правилами:

interrupt [*идентификатор прерывания*]

тип возвращаемого значения *имя функции (список аргументов)*

{

тело функции обработки прерывания

}

Пример декларации обработчика прерывания ADC_INT по вектору 002Ah, которое вырабатывается при завершении преобразования АЦП:

interrupt [ADC_INT] void *adc_interrupt* (void)

{

тело функции обработки прерывания

}

Минимальное время реакции прерывание составляет 4 периода тактовой частоты, в результате которых значение программного счетчика записывается в стек, а указатель стека уменьшается на 2. После этого выполняется относительный переход на подпрограмму (функцию), обрабатывающую данное прерывание. Если прерывание происходит во время выполнения команды длящейся несколько циклов, перед вызовом прерывания завершается выполнение этой команды. Выход из программы обслуживания прерывания занимает 4 периода тактовой частоты, во время которых из стека восстанавливается значение программного счетчика. После выхода из

прерывания процессор всегда выполняет еще одну команду, прежде чем обслужить любое отложенное прерывание.

4.1.2. Принципы функционирования аппаратных таймеров-счетчиков, входящих в состав микроконтроллера AVR ATMEGA 128.

В микроконтроллере AVR ATMEGA 128 содержатся 4 аппаратных таймера/счетчика (T0 – T3), из которых T0 и T2 являются 8 разрядными, а T1 и T3 – 16 разрядными. В состав таймеров/счетчиков входят 3 основных регистра ввода/вывода: счетный регистр **TCNTx**, регистр управления **TCCRx** и регистр сравнения **OCRx**. Дополнительно для управления прерываниями от таймеров/счетчиков используются регистры **TIMSK** и **TIFR**.

В регистр **TIMSK** записываются управляющие (маскирующие) биты (см. рисунок 4.1).

№ бита	7	6	5	4	3	2	1	0
37h(57h)	OSIE2	TOIE2	TICIE1	OSIE1A	OSIE1B	TOIE1	OSIE0	TOIE0

Рисунок 4.1 – Регистр маски прерываний от таймеров/счетчиков – **TIMSK**.

Описание управляющих битов регистра **TIMSK** приводится ниже:

- Бит 7 – **OSIE2**: разрешение прерывания по совпадению 8 разрядного таймера/счетчика T2;
- Бит 6 – **TOIE2**: разрешение прерывания по переполнению 8 разрядного таймера/счетчика T2;
- Бит 5 – **TICIE1**: разрешение прерывания по событию «захват» 16 разрядного таймера/счетчика T1;
- Бит 4 – **OSIE1A**: разрешение прерывания по совпадению «А» 16 разрядного таймера/счетчика T1;
- Бит 3 – **OSIE1B**: разрешение прерывания по совпадению «В» 16 разрядного таймера/счетчика T1;
- Бит 2 – **TOIE1**: разрешение прерывания по переполнению 16 разрядного таймера/счетчика T1;
- Бит 1 – **OSIE0**: разрешение прерывания по совпадению 8 разрядного таймера/счетчика T0;
- Бит 0 – **TOIE0**: разрешение прерывания по переполнению 8 разрядного таймера/счетчика T0;

В регистре **TIFR** формируются соответствующие флаги (см. рисунок 4.2).

№ бита	7	6	5	4	3	2	1	0
36h (56h)	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0

Рисунок 4.2 – Регистр флагов прерываний от таймеров/счетчиков – **TIFR**.

Описание флагов регистра **TIFR** приводится ниже:

- Бит 7 – OCF2: флаг прерывания по совпадению 8 разрядного таймера/счетчика T2;
- Бит 6 – TOF2: флаг прерывания по переполнению 8 разрядного таймера/счетчика T2;
- Бит 5 – ICF1: флаг прерывания по событию «захват» 16 разрядного таймера счетчика T1;
- Бит 4 – OCF1A: флаг совпадения «А» 16 разрядного таймера/счетчика T1;
- Бит 3 – OCF1B: флаг совпадения «В» 16 разрядного таймера/счетчика T1;
- Бит 2 – TOV1: флаг прерывания по переполнению 16 разрядного таймера/счетчика T1;
- Бит 1 – OCF0: флаг прерывания по совпадению 8 разрядного таймера/счетчика T0;
- Бит 0 – TOV0: флаг прерывания по переполнению 8 разрядного таймера/счетчика T0;

Рассмотрим подробно принципы функционирования 8-ми разрядного таймера/счетчика T0, структурная схема которого представлена на рисунке 4.3. В состав таймера/счетчика T0 входят 3 8-разрядных регистра ввода/вывода:

- счетный регистр **TCNT0**;
- регистр сравнения **OCR0**;
- регистр управления **TCCR0**.

Дополнительно для задания режимов работы и управления прерываниями используются регистры **ASSR**, **TIMSK** и **TIFR**. Адреса основных регистров таймера/счетчика T0 с указанием их идентификаторов приводятся в таблице 4.2.

Таблица 4.2 – Адреса* и аббревиатуры регистров, использующихся при работе с таймером/счетчиком T0

Адрес*	Аббревиатура	Адрес*	Аббревиатура	Адрес*	Аббревиатура
032h	TCNT0	033h	TCCR0	037h	TIMSK
031h	OCR0	030h	ASSR	036h	TIFR

*Адреса регистров указываются в адресном пространстве ввода/вывода.

Регистр **ASSR** (см. рисунок 4.4) позволяет задать режим работы таймера/счетчика T0. Сигнал AS0 подается на адресный вход мультиплексора MUX1 и определяет входной сигнал для предварительного делителя Prsc частоты. Результирующая частота f_{p0} таймера/счетчика T0 (частота обновления значений в счетном регистре **TCNT0**, период – T_{p0}) определяется значениями битов CS00, CS01, CS02 регистра TCCR0 (см. рисунок 4.5), с помощью которого устанавливаются необходимые параметры работы таймера. В зависимости от режима работы, значения, содержащиеся в

счетном регистре **TCNT0**, либо инкрементируются, либо декрементируются (изначально в **TCNT0** содержится 0). Флаг переполнения таймера находится в регистре **TIFR** (см. рисунок 4.2). Разрешение и запрещение прерываний от таймера устанавливается в соответствующих битах регистра **TIMSK** (см. рисунок 4.1).

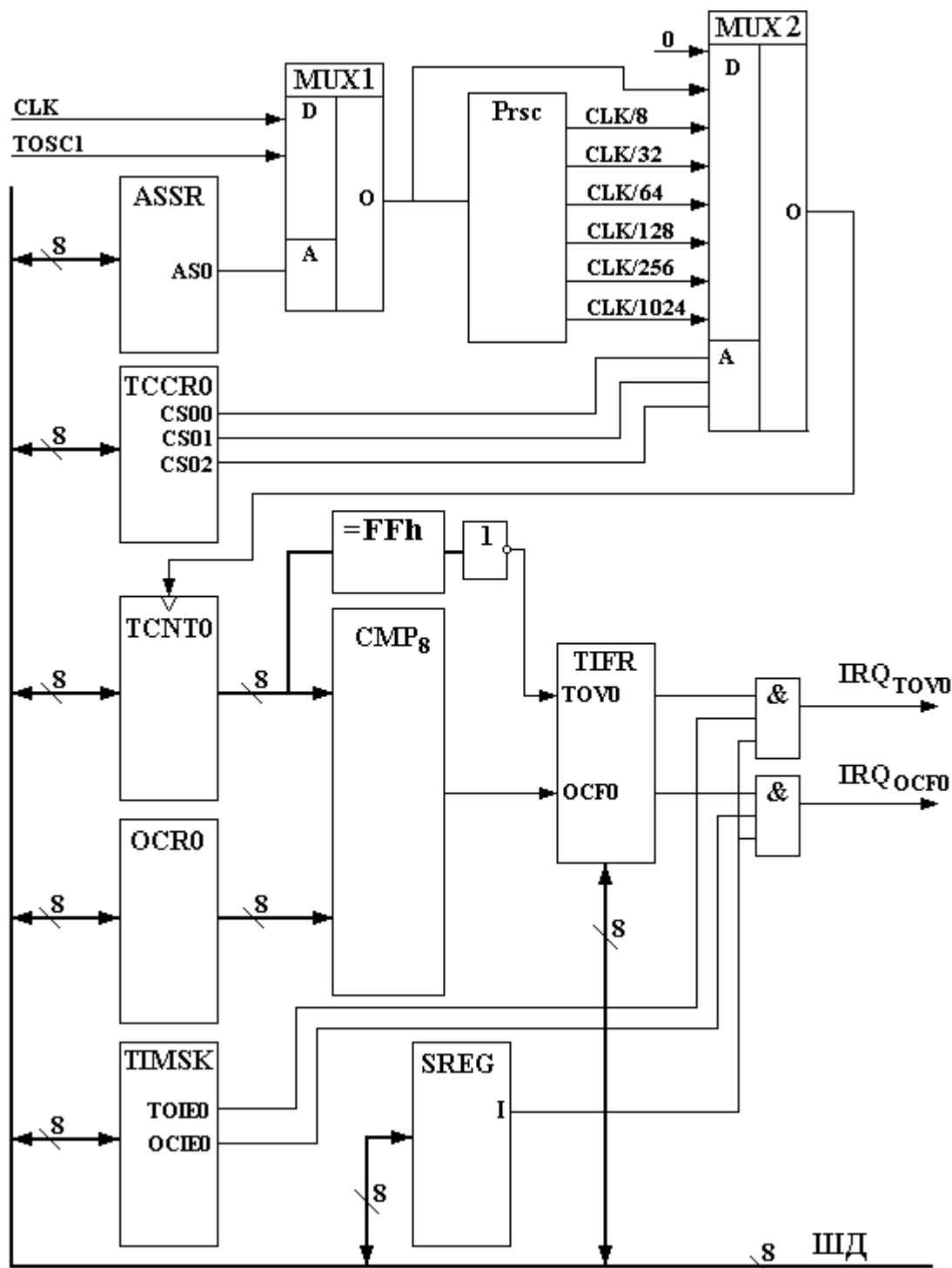


Рисунок 4.3 – Структурная схема 8-ми разрядного таймера/счетчика T0

Таймер/счетчик T0 можно использовать как счетчик с высоким разрешением, так и для точных применений с низким коэффициентом деления тактовой частоты. Более высокие коэффициенты деления можно указывать для работы с медленно изменяющимися функциями или при измерении длительности временных интервалов между редкими событиями. При переполнении счетного регистра TCNT0 устанавливается флаг TOV0. В каждом цикле работы таймера/счетчика T0 происходит сравнение значений, находящихся в счетном регистре TCNT0 и регистре сравнения OCR0. В случае равенства содержимого этих регистров устанавливается флаг OCF0 регистра TIFR (см. рисунок 4.2). Если в регистре TIMSK (см. рисунок 4.1) установлены управляющие биты TOIE0 и OCIE0, то при возникновении событий TCNT0>255 (переполнение) и TCNT0=OCR0 (совпадение) генерируются соответствующие прерывания, обработчики которых описываются как:

interrupt [TIM0_OVF] void timer0_overflow (void) // переполнение;

interrupt [TIM0_COMP] void timer0_compare (void) // совпадение.

Прерывания обрабатываются только при установленном флаге I разрешения прерываний регистра состояния SREG.

№ бита	7	6	5	4	3	2	1	0
30h (50h)	X	X	X	X	AS0	TCN0UB	TCN0UB	TCN0UB

Рисунок 4.4 – Регистр управления таймером/счетчиком T0 в асинхронном режиме – ASSR

Описание управляющих битов регистра ASSR приводится ниже:

Биты 7..4 – зарезервированы и всегда читаются как 0.

Бит 3 – AS0: бит переключения режима работы 8 разрядного таймера/счетчика T0: если AS0=0, то на вход предварительного делителя частоты поступает внутренний тактовый сигнал микроконтроллера с частотой CLK (синхронный режим), если AS0=1, то на вход предварительного делителя частоты поступает внешний тактовый сигнал по линии TOSC1 (асинхронный режим);

Бит 2 – TCNT0UB: флаг обновления счетного регистра TCNT0: в начале операции записи нового значения в TCNT0 флаг устанавливается в 1, а при завершении – в 0;

Бит 1 – OCR0UB: флаг обновления регистра сравнения OCR0: в начале операции записи нового значения в OCR0 флаг устанавливается в 1, а при завершении – в 0;

Бит 0 – TCR0UB: флаг обновления управляющего регистра TCCR0: в начале операции записи нового значения в TCCR0 флаг устанавливается в 1, а при завершении – в 0;

№ бита	7	6	5	4	3	2	1	0
33h (53h)	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

Рисунок 4.5 – Регистр управления таймером/счетчиком T0 - **TCCR0**

Описание управляющих битов регистра **TCCR0** приводится ниже:

Бит 7 – FOC0: бит установки принудительного изменения состояния вывода OC0: FOC0=0 в режимах “Быстродействующий ШИМ” и “ШИМ с точной фазой”, если FOC0=1, то состояние вывода OC0 изменяется в соответствии с установками разрядов COM00 и COM01;

Биты 6,3 – WGM00, WGM01: определяют режим работы таймера счетчика T0 (см. таблицу 4.3);

Биты 5,4 – COM01, COM00: определяют поведение вывода OC0 при появлении события «совпадение» (см. таблицу 4.4) ;

Биты 2,1,0 – CS02, CS01, CS00: определяют частоту тактового сигнала таймера/счетчика T0 (см. таблицу 4.5);

Таблица 4.3 – Режимы работы таймера/счетчика T0

WGM01	WGM00	Описание режима работы таймера/счетчика T0
0	0	Стандартный (Normal)
0	1	ШИМ с точной фазой (Phase correct PWM)
1	0	Сброс при совпадении (CTC)
1	1	Быстродействующий ШИМ Fast PWM

Таблица 4.4 – Режимы управления выводом OC0 таймера/счетчика T0

COM01	COM00	Описание режима управления выводом OC0
0	0	Таймер/счетчик T0 отключен от вывода OC0
0	1	Инверсия сигнала на выводе OC0
1	0	Вывод OC0 сбрасывается в 0
1	1	Вывод OC0 устанавливается в 1

При работе таймера/счетчика T0 от внешнего сигнала, внешний сигнал синхронизируется с сигналом CLK тактового генератора микроконтроллера. Для корректной обработки внешнего сигнала, минимальный интервал между соседними импульсами внешнего сигнала должен превышать период тактовой частоты процессора. Сигнал внешнего источника обрабатывается по спадающему фронту тактовой частоты процессора.

Таблица 4.5 – Значения коэффициентов для предварительного делителя частоты таймера/счетчика T0

CS02	CS01	CS00	Описание	f_{p0} , кГц	T_{p0} , мкс
0	0	0	Таймер/счетчик остановлен	0	–
0	0	1	CLK	11000	0,09
0	1	0	CLK/8	1375	0,72
0	1	1	CLK/32	344	2,88
1	0	0	CLK/64	172	5,76
1	0	1	CLK/128	86	11,52
1	1	0	CLK/256	43	23,04
1	1	1	CLK/1024	11,7	92,16

В состав микроконтроллера AVR MEGA 128 входят два 16-разрядных таймера/счетчика T1 и T3. Данные устройства, за исключением некоторых специфических функций (например, режим “захвата”), работают аналогично 8-разрядным таймерам счетчикам и используются для сходных целей, однако позволяют выполнять счет со значительно меньшими частотами.

Рассмотрим подробно функционирование таймера/счетчика T1. В состав таймера/счетчика T1 входят следующие регистры:

- 16-разрядный счетный регистр **TCNT1**;
- три 16-разрядных регистра сравнения **OCR1A, OCR1B, OCR1C**;
- три 16-разрядных регистра управления **TCCR1A, TCCR1B, TCCR1C**;
- 16-разрядный регистр захвата **ICR1**.

Дополнительно для задания режимов работы и управления прерываниями используются регистры **TIMSK, ETIMSK, TIFR, ETIFR, ICR1**. Адреса основных регистров таймера/счетчика T1 с указанием их идентификаторов приводятся в таблице 4.6.

Каждый 16-разрядный регистр физически размещается в двух 8-разрядных регистрах ввода/вывода, аббревиатуры которых получаются путем добавления к названию регистра букв **H** (High, старший байт) **L** (Low – младший байт). Например, 16 разрядный счетный регистр **TCNT1** физически находится в двух 8-разрядных регистрах **TCNT1H : TCNT1L**. Для того, чтобы запись или чтение обоих байт, входящих в состав 16-разрядного регистра, происходила одновременно, в составе таймеров/счетчиков T1 и T3 имеется специальный (программно недоступный) 8-разрядный регистр **TEMP**, предназначенный для хранения старшего байта значения.

Таблица 4.6 – Адреса* и аббревиатуры регистров, использующихся при работе с таймером/счетчиком T1.

Адрес*	Аббревиатура	Адрес*	Аббревиатура
02Ch	TCNT1L	02Fh	TCCR1A
02Dh	TCNT1H	02Eh	TCCR1B
02Ah	OCR1AL	07Ah	TCCR1C
02Bh	OCR1AH	036h	TIFR
028h	OCR1BL	07Ch	ETIFR
029h	OCR1BH	037h	TIMSK
078h	OCR1CL	07Dh	ETIMSK
079h	OCR1CH		

*Адреса регистров указываются в адресном пространстве ввода/вывода.

При записи 16-разрядного значения сначала должен быть загружен старший байт (в регистр **TEMP**), а затем младший байт с регистром **TEMP** записываются в одном такте. Прерывания при записи данных в 16-разрядные регистры прерывания должны быть запрещены. В режиме чтения сначала должен быть считан младший байт 16-разрядного регистра. Счетный регистр **TCNT1** используется для хранения значений реверсивного счетчика, данные в котором инкрементируются или декрементируются по фронту тактового сигнала таймера/счетчика T1. Если установлен флаг **TOIE1** в регистре **TIMSK** (см. рисунок 4.1), то при переполнении **TCNT1** генерируется прерывание и устанавливается флаг **TOV1** в регистре **TIFR** (см. рисунок 4.2). Заголовок процедуры обработки прерывания по переполнению регистра-счетчика **TCNT1** может выглядеть следующим образом:

interrupt [TIM1_OVF] void timer1_overflow (void).

Блок сравнения таймера/счетчика T1 содержит 3 16-разрядных регистра сравнения **OCR1A**, **OCR1B**, **OCR1C**. Во время работы таймера/счетчика в каждом машинном цикле выполняется непрерывное сравнение данных этих регистрах со значением в **TCNT1**. При равенстве значений этих регистров генерируются прерывания по сравнению (при установленных флагах **TOIE1A**, **TOIE1B**, **TOIE1C** в регистрах **TIMSK/ETIMSK**) и устанавливаются соответствующие флаги **OCF1A**, **OCF1B**, **OCF1C** в регистрах **TIFR/ETIFR**. Описания заголовков процедур обработки прерываний по сравнению может выглядеть следующим образом:

interrupt [TIM1_COMPA] void timer1_compareA (void);

interrupt [TIM1_COMPB] void timer1_compareB (void);

interrupt [TIM1_COMPC] void timer1_compareC (void).

В стандартном режиме работы все биты управляющих регистров **TCCR1A** и **TCCR1C** устанавливаются в **0** (подробное описание данных

регистров приводится в [1]). Формирование тактового сигнала таймера/счетчика осуществляется с помощью предделителя. Результирующая частота f_{p1} таймера/счетчика T1 (частота обновления значений в счетном регистре **TCNT1**, период – T_{p1}) определяется значениями битов CS00, CS01, CS02 регистра **TCCR1B** (см. рисунок 4.6), с помощью которого устанавливаются необходимые параметры работы таймера (см. таблицу 4.7). Автоматический сброс счетного регистра **TCNT1** при совпадении со значением регистра **OCR1A** программируется путем установки 3-го бита регистра **TCCR1B** в 1.

№ бита	7	6	5	4	3	2	1	0
2Eh (4Eh)	FOC0	WGM00	COM01	WGM13	WGM12	CS02	CS01	CS00

Рисунок 4.6 – Регистр управления таймером/счетчиком T1 – **TCCR1B**

Описание управляющих битов регистра **TCCR1B** сходно с описанием битов регистра **TCCR0** (см. рисунок 4.5).

Таблица 4.6 – Значения коэффициентов для предварительного делителя частоты таймера/счетчика T1

CS02	CS01	CS00	Описание	f_{p1} , кГц	T_{p1} , мкс
0	0	0	Таймер/счетчик остановлен	0	–
0	0	1	CLK	11000	0,09
0	1	0	CLK/8	1375	0,72
0	1	1	CLK/64	172	5,76
1	0	0	CLK/256	43	23,04
1	0	1	CLK/1024	11,7	92,16
1	1	0	Счет осуществляется по заднему фронту импульсов на линии T1	–	–
1	1	1	Счет осуществляется по переднему фронту импульсов на линии T1	–	–

Пример С–программы инициализации таймера–счетчика T0 для генерирования прерываний по переполнению, работающего в стандартном режиме с частотой CLK/1024 приводится ниже:

ASSR = 0; задание синхронного режима работы;
TCCR0 = 7; задание параметров предделителя;
TIMSK = 1; установка бита TOIE0 для разрешения прерывания по переполнению таймера-счетчика T0;
#asm("sei"); разрешить прерывания в системе.

4.2 Описание лабораторной установки

Лабораторная работа выполняется в индивидуальном порядке. На каждом рабочем месте должны быть установлены: многофункциональный лабораторный макет на базе микроконтроллера AVR ATMEGA 128, ПЭВМ типа IBM PC/AT с установленным программным обеспечением: операционной системой MS–WINDOWS v. 9x, 2000, XP и программатором на основе кросс-компилятора языка программирования C CodeVision AVR. Задания выполняются на лабораторном макете на базе 8-ми разрядного микроконтроллера AVR ATMEGA 128. Исследуемые в работе таймеры - счетчики T0 и T1 входят в состав микроконтроллера. Для индикации режимов работы таймеров дополнительно используется блок из 8 светодиодов (см. рисунок 1.7), подключенных к микроконтроллеру через порт D в соответствии со схемой, приведенной на рисунке 1.9 (катоды светодиодов подключены к выводам порта D, на аноды светодиодов подано положительное напряжение). Подробное описание лабораторного макета приведено в пункте 1.2 лабораторной работы № 1.

4.3 Порядок проведения работы и указания по ее выполнению

Перед началом выполнения практической части лабораторной работы проводится экспресс–контроль знаний по принципам функционирования таймеров/счетчиков, входящих в состав микроконтроллера AVR ATMEGA 128, а также по способам задания и измерения временных интервалов. При подготовке к лабораторной работе необходимо составить предварительный вариант листинга программы, в соответствии с индивидуальным заданием (см. таблицу 4.7).

Задание. Разработать в среде программирования Code Vision AVR программу на языке C для микроконтроллера AVR ATMEGA 128, управляющую блоком из 8-ми светодиодов. Временные параметры индикации задаются с помощью установок таймеров – счетчиков T1 или T0. Варианты индивидуальных заданий представлены в таблице 4.7.

Порядок выполнения задания:

1. Включить лабораторный макет (установить выключатель электропитания в положение I, и убедиться в свечении индикатора электропитания красным цветом).

2. Запустить компилятор Code Vision AVR.
3. Создать пустой проект.
4. Создать файл ресурса для кода программы и подключить его к проекту.
5. Ввести код исходного модуля программы управления блоком светодиодов в соответствии с вариантом задания, указанным в таблице 4.7.
6. Выполнить компиляцию (нажав клавишу **F9**) исходного модуля программы и устранить ошибки, полученные на данном этапе.
7. Настроить параметры программатора.
8. Создать загрузочный модуль программы (нажав комбинацию клавиш **Shift+F9**) и выполнить программирование микроконтроллера.
9. Проверить работоспособность загруженной в микроконтроллер программы и показать результаты работы преподавателю.
10. В случае некорректной работы разработанной программы, выполнить аппаратный сброс микроконтроллера, провести отладку исходного модуля программы и заново проверить функционирование программы, повторив выполнение пункта 9.

Пример выполнения задания: разработать программу, выполняющую в бесконечном цикле управление блоком светодиодов в режиме “бегущий огонь” (с последовательным включением/выключением светодиодов блока индикации). Задание временных интервалов выполнить с помощью следующих настроек таймера T1: управляющее прерывание **TIM1_OVF**, частота счета $f_{p1} = 43$ КГц.

Решение: Программное управление блоком светодиодов в режиме “бегущий огонь” можно обеспечить, записывая во все, кроме нулевого, разряды регистра PORTD порта D уровни “логической единицы” (погасить все светодиоды кроме нулевого), а затем выполняя циклическое перемещение нулевого уровня по линиям порта D. Данные действия необходимо разметить в обработчике прерывания **TIM1_OVF** по переполнению от таймера – счетчика T1.

Частота изменения значений в счетном регистре **TCNT1** таймера – счетчика T1 равна 43 КГц (см. таблицу 4.6):

$$f_{p1} = \text{CLK}/256 = 11 \text{ МГц}/256 = 43 \text{ КГц}$$

Соответственно значение, заносимое в регистр **TCCR1B**, равняется 4.

Период изменения значений в счетном регистре **TCNT1**:

$$T_{p1} = 1/f_{p1} = 23 \text{ мкс.}$$

Соответственно длительность одного полного цикла счета таймера T1 равна:

$$t_1 = 23 \text{ мкс} \cdot 65536 \approx 1,5 \text{ с.}$$

Таким образом, с интервалом 1,5 с будет осуществляться циклическое перемещение зажженного светодиода (согласно алгоритму, приведенному на рисунке 4.7).

Полный текст исходного модуля программы на языке C с подробными комментариями приводится ниже:

```
#include <mega128.h> Подключить заголовочный файл mega128.h;  
unsigned char ld_stat=0xFE; описание глобальной переменной ld_stat  
с присвоением начального значения 0xfe;  
  
interrupt [TIM1_OVF] void timer1_overflow(void) процедура обработки  
{ прерывания по переполнению от таймера-счетчика T1;  
led_stat<<=1; выполнить логический сдвиг на 1 разряд значения в ld_stat;  
ld_stat|=1; выполнить операцию логического ИЛИ над младшим битом  
переменной ld_stat;  
if (ld_stat==0xFF) ld_stat=0xFE; при переносе нулевого значения из 7-го  
бита (за пределы левой границы блока светодиодов)  
загрузить в переменную ld_stat начальное значение 0xFE;  
PORTD=ld_stat; вывести значение led_status в PORTD для индикации;  
} завершение процедуры обработки прерывания;  
  
main() { основная часть программы;  
DDRD=0xFF; настроить порт D на вывод данных;  
PORTD=0xFF; погасить все светодиоды;  
TCCR1A=0; задание режимов работы для таймера-счетчика T1;  
TCCR1B=4; задание частоты работы для таймера-счетчика T1;  
TIMSK=4; разрешить прерывание TIM1_OVF по переполнению  
16 разрядного таймера/счетчика T1;  
#asm("sei"); установить бит I общего разрешения прерываний;  
while (1) { } установить цикл с бесконечным числом итераций;  
} завершающая операторная скобка программы.
```

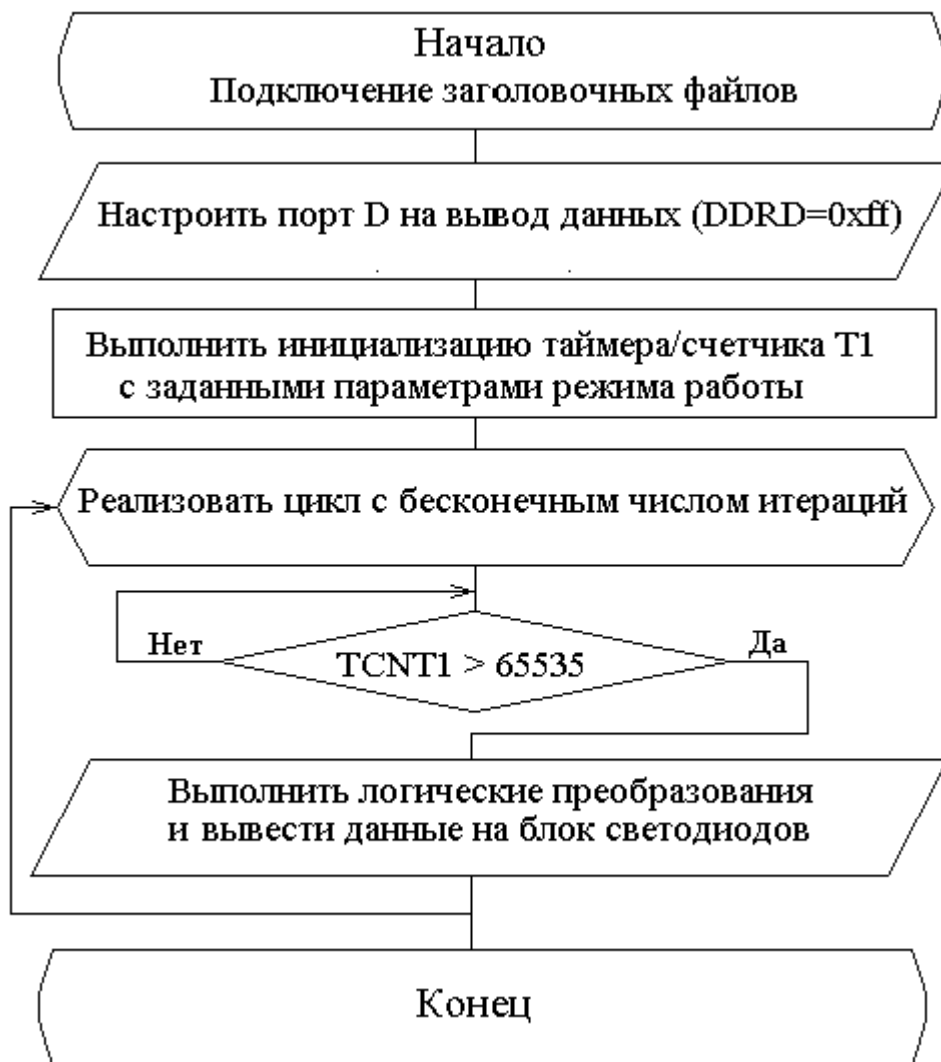



Рисунок 4.7 – Алгоритм программы управления блоком светодиодов в режиме “бегущий огонь”.

Таблица 4.7 – Варианты индивидуальных заданий

№ п.п.	Задание
1	Разработать программу, выполняющую в бесконечном цикле включение/выключение 1–го светодиода: управляющее прерывание TIM1_OVF , частота счета $f_{p1} = 43$ КГц.
2	Разработать программу, выполняющую в бесконечном цикле включение/выключение 2–го светодиода: управляющее прерывание TIM0_OVF , частота счета $f_{p1} = 11,7$ КГц.
3	Разработать программу, выполняющую в бесконечном цикле включение/выключение 3–го светодиода: управляющее прерывание TIM1_COMPА , $f_{p1} = 43$ КГц, OCR1A=40000 .
4	Разработать программу, выполняющую в бесконечном цикле включение/выключение 0–го и 7–го светодиодов: управляющее прерывание TIM1_COMPВ , $f_{p1} = 43$ КГц, OCR1B=50000 .

5	Разработать программу, выполняющую в бесконечном цикле включение/выключение 4–го и 5–го светодиодов: управляющее прерывание TIM1_COMPB , $f_{p1} = 43$ КГц, OCR1B=30000 .
6	Разработать программу, выполняющую в бесконечном цикле включение/выключение 3–го и 6–го светодиодов: управляющее прерывание TIM0_OVF , частота счета $f_{p1} = 11$ КГц, OCR0=200 .
7	Разработать программу, выполняющую в бесконечном цикле включение/выключение 2–го светодиода: управляющее прерывание TIM1_COMPA , $f_{p1} = 11$ КГц, OCR1A=10000 .
8	Разработать программу, выполняющую в бесконечном цикле включение/выключение 3–го и 6–го светодиодов: управляющее прерывание TIM1_OVF , частота счета $f_{p1} = 86$ КГц.
9	Разработать программу, выполняющую в бесконечном цикле включение/выключение 1–го и 2–го светодиодов: управляющее прерывание TIM1_OVF , частота счета $f_{p1} = 11$ КГц.
10	Разработать программу, выполняющую в бесконечном цикле включение/выключение 0–го и 5–го светодиодов: управляющее прерывание TIM1_COMPB , $f_{p1} = 43$ КГц, OCR1B=30000 .

* задания повышенной сложности.

4.4 Содержание отчета

В отчете необходимо привести следующее:

характеристики лабораторной вычислительной системы;

исходный модуль разработанной программы;

анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности использования встроенных таймеров микроконтроллера для формирования аппаратно-независимых временных интервалов.

4.5 Контрольные вопросы и задания

1. В чем преимущества обмена по прерываниям по сравнению с другими известными вам способами обмена информацией ?
2. Что включает в себя понятия системы прерываний ?
3. Поясните понятия вектора прерываний и таблицы векторов прерываний.
4. Какие действия выполняет микроконтроллер при переходе на процедуру обработки прерывания ?
5. Поясните принципы формирования временных интервалов с помощью 8–разрядного таймера–счетчика.
6. Поясните принципы формирования временных интервалов с помощью 16–разрядного таймера–счетчика.

5 ИЗУЧЕНИЕ ПРИНЦИПОВ ОРГАНИЗАЦИИ ОБМЕНА ДАННЫМИ ПО ПОСЛЕДОВАТЕЛЬНОМУ ИНТЕРФЕЙСУ RS-232C МЕЖДУ МИКРОКОНТРОЛЛЕРОМ AVR ATMEGA128 И ПЭВМ

Цель работы: Изучить возможности сопряжения лабораторного макета на базе микроконтроллера AVR ATMEGA128 и ПЭВМ с помощью последовательного интерфейса RS-232C, принципы программного управления двунаправленным обменом данными по последовательному интерфейсу RS-232C.

5.1 Указания по организации самостоятельной работы.

Перед работой необходимо проработать теоретический материал по литературе [1–4] и конспект лекций, ознакомиться с основными возможностями и принципами функционирования последовательного интерфейса RS-232C, возможностями сопряжения лабораторного макета на базе микроконтроллера AVR ATMEGA128 и ПЭВМ с помощью последовательного интерфейса RS-232C, принципами программного управления двунаправленным обменом данными по последовательному интерфейсу RS-232C.

5.1.1. Принципы обмена данными по последовательному интерфейсу RS-232C.

Интерфейс RS-232C предназначен для соединения двух устройств (см. рисунок 5.1), находящихся на расстоянии до 15 м с предельной скоростью обмена данными около 10 кБайт/с. Линия TxD передачи первого устройства через преобразователь уровней RS-232C/ТТЛ соединяется с линией RxD приема второго и наоборот (режим обмена full duplex). Дополнительно используются общий и экранирующий сигналы интерфейса. Для управления соединенными устройствами применяется программное подтверждение (введение в поток передаваемых данных соответствующих управляющих символов). Возможна организация аппаратного подтверждения путем введения в протокол обмена дополнительных сигналов интерфейса для обеспечения функций определения статуса и управления.

Данные по интерфейсу RS-232C передаются в последовательном коде по кадрам – порциям данных, обрамленных служебной информацией (см. рисунок 5.2). Когда обмена данными нет, на линиях RxD или TxD присутствует высокий уровень сигнала. Кадр начинается со стартового бита (сигнальные линии RxD или TxD переводятся в состояние логического нуля), за которым следует младший бит слова данных, состоящего из 5-9 информационных разрядов). Далее (в зависимости от режима) может

следовать бит четности (паритета). Завершают кадр один или два стоповых бита. Получив стартовый бит, приемник выбирает из линии биты данных через определенные интервалы времени. Очень важно, чтобы тактовые частоты приемника и передатчика были одинаковыми (допустимое расхождение - не более 10%).



Рисунок 5.1 – Обобщенная функционально-структурная схема соединения двух устройств с помощью интерфейса RS-232C

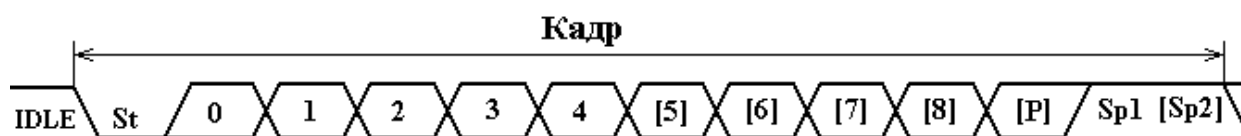


Рисунок 5.2 – Формат кадра при обмене данными по интерфейсу RS-232C

5.1.2 Организация модулей USART в микроконтроллере AVR ATMEGA128. Все микроконтроллеры AVR семейства MEGA имеют в своем составе модули универсального синхронно/асинхронного приемопередатчика USART (Universal Synchronous and Asynchronous serial Receiver and Transmitter). В микроконтроллер AVR ATMEGA128 встроены два таких модуля USART0 и USART1. Каждый модуль USART состоит из трех частей: блока тактирования, блока передатчика и блока приемника. Блок тактирования включает в себя устройство синхронизации (при работе в синхронном режиме) и контроллер скорости передачи данных. Блок передатчика включает одноуровневый буфер, регистр сдвига, схему формирования четности (для синхронного режима) и схему управления. Блок приемника состоит из схемы восстановления тактового сигнала и данных, схемы контроля четности (для синхронного режима), буферного и сдвигового регистров, а так же схемы управления. Для упрощения описания далее будут рассматриваться ресурсы только модуля USART1 микроконтроллера AVR ATMEGA128. Работа с модулем USART0 будет проводиться аналогично.

Модуль USART1 имеет следующие программно-доступные регистры:

UDR1 – регистр данных;

UCSR1A, UCSR1B, UCSR1C – регистры управления/статуса ;
UBRR1H, UBRR1L – регистры скорости передачи данных.

Рассмотрим их функциональное назначение.

В режиме передатчика запись данных в регистр **UDR1**, расположенный по адресу 9Ch(BCh), инициирует передачу данных (данные из регистра **UDR1** пересылаются в регистр сдвига и подаются на линию TxD побитово). В режиме приемника считывание полученных данных осуществляется из регистра **UDR1**. Инициализация и контролирование режимов работы модуля UART1 происходит с помощью регистров управления/статуса **UCSR1A** (см. рисунок 5.3), **UCSR1B** (см. рисунок 5.4), **UCSR1C** (см. рисунок 5.5).

№ бита	7	6	5	4	3	2	1	0
9Bh(BBh)	RXC1	TXC1	UDRE1	FE1	DOR1	UPE1	U2X1	MPCM1

Рисунок 5.3 – Регистр управления/статуса **UCSR1A**.

Описание битов регистра управления/статуса **UCSR1A** приведено ниже:
 Бит 7 – **RXC1** – устанавливается, если в буфере UDR есть непрочитанные данные, и сбрасывается, когда приемный буфер пуст (нет непрочитанных данных). Если прием запрещен, этот бит всегда читается как 0. Этот бит может использоваться для вызова прерывания по приему данных.
 Бит 6 – **TXC1** – устанавливается, когда буфер передачи UDR пуст.
 Бит 5 – **UDRE1** – устанавливается, когда регистр данных пуст.
 Бит 4 – **FE1** – признак ошибки кадра.
 Бит 3 – **DOR1** – переполнение приемного буфера.
 Бит 2 – **UPE1** – ошибка бита четности.
 Бит 1 – **U2X1** – бит управления скоростью передачи (0 – стандартная, 1 – удвоенная скорость).
 Бит 0 – **MPCM1** – 0 – стандартный, 1 – мультипроцессорный режим работы.

№ бита	7	6	5	4	3	2	1	0
9Ah(BAh)	RXCIE1	TXCIE1	UDRIE1	RXEN1	TXEN1	UCSZ12	RXB81	TXB81

Рисунок 5.4 – Регистр управления/статуса **UCSR1B**

Описание битов регистра управления/статуса **UCSR1B** приведено ниже:
 Бит 7 – **RXCIE1** – разрешение прерывания по завершению приема (при установке бита **RXC1** в регистре **UCSR1A**).
 Бит 6 – **TXCIE1** – разрешение прерывания по завершению передачи (при установке бита **TXC1** в регистре **UCSR1A**).
 Бит 5 – **UDRIE1** – разрешение прерываний при очистке регистра данных **UDR1** и установке флага **UDRE1** регистре **UCSR1A**.
 Бит 4 – **RXEN1** – установка этого бита разрешает работу приемника USART.

Бит 3 – TXEN1 – установка этого бита разрешает работу передатчика USART.

Бит 2 – UCSZ12 – в сочетании с битами UCSZ11:UCSZ10 регистра UCSRC устанавливает размер кадра данных (см. таблицу 5.1).

Таблица 5.1. – Размер кадра данных

UCSZ12	UCSZ11	UCSZ10	Размер данных
0	0	0	5 бит
0	0	1	6 бит
0	1	0	7 бит
0	1	1	8 бит
1	1	1	9 бит

Бит 1 – RXB81 – формат принимаемых данных 9 бит.

Бит 0 – TXB81 – формат передаваемых данных 9 бит.

№ бита	7	6	5	4	3	2	1	0
9Dh(BDh)	-	UMSEL1	UPM11	UPM10	USBS1	UCSZ11	UCSZ10	UCPOL1

Рисунок 5.5 – Регистр управления/статуса UCSR1C.

Описание битов регистра управления/статуса UCSR1C приведено ниже:

Бит 6 – UMSEL – бит выбора режима работы USART (0-асинхронный. 1-синхронный).

Биты 5, 4 – UPM11, UPM10 – определяют режим проверки четности при приеме и при передаче данных (см. таблицу 5.2).

Таблица 5.2 – Установки режима четности

UPM1	UPM0	Режим четности
0	0	отключен
0	1	не используется
1	0	проверка четности
1	1	проверка нечетности

Бит 3 – USBS1 – определяет количество стоповых битов для передатчика. Приемник игнорирует этот бит (0 – один стоповый бит, 1 – два стоповых бита).

Биты 2,1 – UCSZ11, UCSZ10 – в сочетании с битом UCSZ12 регистра UCSR1B устанавливают размер кадра данных (см. таблицу 5.1);

Бит 0 – UCPOL1 – в асинхронном режиме должен быть равен 0.

Скорость передачи данных V_{BAUD} (в бодах, бит/с) определяется из выражения 5.1 и задается путем записи 12-разрядного значения в регистры

UBRR1L 98h(B8h) и **UBRR1H** 99h(B9h). В регистре **UBRR1H** используются только младшие 4 разряда.

$$V_{BAUD} = \frac{f_{clk}}{16(UBRR_{H:L} + 1)}. \quad (5.1)$$

Значение, записываемое в регистры **UBRR1H:UBRR1L**, будет соответственно определяться по формуле 5.2:

$$UBRR_{H:L} = \frac{f_{clk}}{16 \cdot V_{BAUD}} - 1. \quad (5.2)$$

Стандартные значения делителя $UBRR_{H:L}$, соответствующие стандартным скоростям передачи данных, для тактовой частоты $f_{clk} = 11,0592$ МГц микроконтроллера, входящего в состав лабораторного макета, приводятся в таблице 5.3. При этом необходимо учитывать значение – бита U2X1 управления скоростью передачи (0 – стандартная, 1 – удвоенная скорость), расположенного в первом разряде регистра **UCSR1A**.

Таблица 5.3 – Значения делителей частоты модуля USART для различных значений скорости передачи данных при нулевой погрешности установки скорости

$UBRR_{H:L}$, бит/с	$f_{clk} = 11,0592$ МГц	
	U2X1=0	U2X1=1
2400	287	575
4800	143	287
9600	71	143
14400	47	95
19200	35	71
28800	23	47
38400	17	35
57600	11	23
115200	5	11

Пример программного кода инициализации модуля USART1 для режима асинхронного считывания данных на скорости 144400 бит/с (в формате 8 бит без бита четности) на языке C приводится ниже:

UCSR1A=0x00; установка стандартного режима задания скорости передачи данных;

UCSR1B=0x90; установка 7-го и 4-го битов регистра **UCSR1B** для

инициализации USART1 в режиме приемника и разрешения прерывания по завершению приема кадра;
UCSR1C=0x06; установка формата кадра: 8 бит данных с отключенным режимом четности;
UBRR1H=0x00; установка значения делителя (47) соответствующего
UBRR1L=47; скорости приема данных 14400 бит/с.

5.2 Описание лабораторной установки

Лабораторная работа выполняется в индивидуальном порядке. На каждом рабочем месте должны быть установлены: многофункциональный лабораторный макет на базе микроконтроллера AVR ATMEGA 128, ПЭВМ типа IBM PC/AT с инсталлированным программным обеспечением: операционной системой MS-WINDOWS v. 9x, 2000, XP, программатором на основе кросс-компилятора языка программирования C CodeVision AVR, утилитой Terminal для работы с последовательным интерфейсом RS232C. Задания выполняются на лабораторном макете на базе 8-ми разрядного микроконтроллера AVR ATMEGA 128. Дополнительно в работе используется кабель с 9-контактными разъемами DB-9 (см. рисунок 1.8) для соединения лабораторного макета с ПЭВМ через последовательный интерфейс RS232C.

Назначение сигналов интерфейса RS232C следующее:

FG – защитное заземление (экран);

–**TxD** – данные, передаваемые компьютером в последовательном коде (логика отрицательная);

–**RxD** – данные, принимаемые компьютером в последовательном коде (логика отрицательная);

RTS – сигнал запроса передачи. Активен во все время передачи;

CTS – сигнал сброса (очистки) для передачи. Активен во все время передачи. Говорит о готовности приемника;

DSR – готовность данных. Используется для задания режима модема;

SG – сигнальное заземление, нулевой провод;

DCD – линия детектирование принимаемого сигнала;

DTR – готовность выходных данных;

RI – индикатор вызова. Говорит о приеме модемом сигнала вызова по телефонной сети;

Для трехпроводной двунаправленной линии связи используются сигналы RxD, TxD и SG. Все 10 сигналов интерфейса задействуются только при работе с модемом. Модули USART0 и USART1 входят в состав микроконтроллера. Дополнительно в лабораторном макете содержится блок преобразования уровней RS232/ТТЛ. Для связи с ПЭВМ через COM – порт используется только асинхронный режим работы интерфейса RS232C.

Подробное описание лабораторного макета приведено в пункте 1.2 лабораторной работы № 1.

Работа с программой Terminal (см. рисунок 5.6) выполняется путем настройки соответствующих параметров протокола обмена в верхней части рабочего окна и ввода отправляемых (в области Transmit) или наблюдения принимаемых (в области Receive) данных в десятичной, шестнадцатеричной или двоичной кодировке.

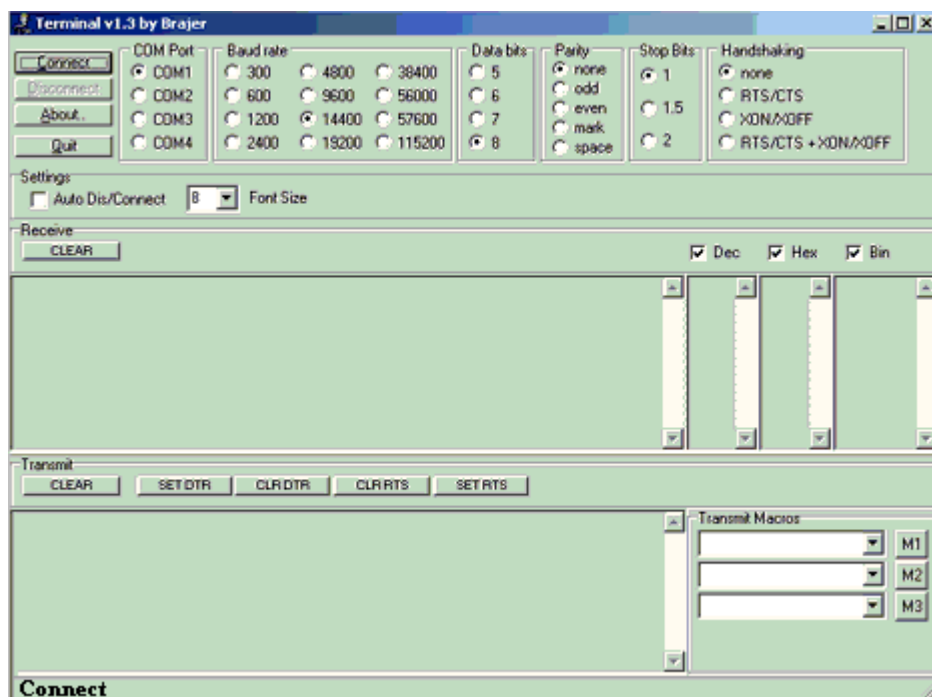


Рисунок 5.6 – Рабочее окно программы Terminal

5.3 Порядок проведения работы и указания по ее выполнению

Перед началом выполнения практической части лабораторной работы проводится экспресс–контроль знаний по принципам функционирования модулей USART, входящих в состав микроконтроллера AVR ATMEGA 128, а также по протоколу обмена данными по интерфейсу RS232C. При подготовке к лабораторной работе необходимо составить предварительный вариант листинга программы, в соответствие с индивидуальным заданием (см. таблицу 5.4).

Задание. Разработать в среде программирования Code Vision AVR программу на языке C для связи микроконтроллера AVR ATMEGA 128 с ПЭВМ по интерфейсу RS232C в соответствие с параметрами протокола обмена, приведенными в таблице 5.4.

Порядок выполнения задания:

1. Включить лабораторный макет (установить выключатель электропитания в положение I, и убедиться в свечении индикатора электропитания красным цветом).
2. Запустить компилятор Code Vision AVR.
3. Создать пустой проект.
4. Создать файл ресурса для кода программы и подключить его к проекту.
5. Ввести код исходного модуля программы обмена данными между микроконтроллером AVR ATMEGA 128 с ПЭВМ по интерфейсу RS232C в соответствии с индивидуальным заданием, приведенным в таблице 5.4.
6. Выполнить компиляцию (нажав клавишу **F9**) исходного модуля программы и устранить ошибки, полученные на данном этапе.
7. Настроить параметры программатора.
8. Проверить правильность подключения интерфейсного кабеля RS232 к разъемам лабораторного макета и ПЭВМ.
9. Запустить на ПЭВМ программу Terminal, установить необходимые параметры протокола обмена данными, выбрать номер последовательного порта (COM1 или COM2), к внешнему разъему которого подключен кабель микроконтроллера, и нажать на кнопку **Connect** в верхнем левом углу рабочего окна программы.
10. Создать загрузочный модуль программы (нажав комбинацию клавиш **Shift+F9**) и выполнить программирование микроконтроллера.
11. Проверить работоспособность загруженной в микроконтроллер программы и показать результаты работы преподавателю.
12. В случае некорректной работы разработанной программы, выполнить аппаратный сброс микроконтроллера, провести отладку исходного модуля программы и заново проверить функционирование программы, повторив выполнение пунктов 7 – 12.

Пример выполнения задания. Разработать программу для передачи 20 чисел (от 0 до 19) из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу RS232C в соответствии с протоколом: модуль USART1, скорость обмена данными 19200 бит/с, режим обмена асинхронный, 7 битов данных без бита четности.

Решение. Исходя из параметров обмена необходимо настроить регистры управления/статуса и скорости передачи данных модуля USART1 а затем в цикле вывести данные в регистр **UDR1**. Алгоритм программы приводится на рисунке 5.7, полный текст исходного модуля программы на языке C с подробными комментариями приводится ниже:

```
#include <mega128.h> Подключить заголовочный файл mega128.h;
#include <delay.h>    подключить заголовочный файл delay.h;
char i;              описание глобальной переменной i;
main() {             основная часть программы;
```

```

UCSR1A=0x00;      установка стандартного режима задания скорости
                    передачи данных;
UCSR1B=0x08;      установка 3-го бита регистра UCSR1B для
                    инициализации USART1 в режиме передатчика;
UCSR1C=0x04;      установка формата кадра: 7 бит данных с
                    отключенным режимом четности;
UBRR1H=0x00;      установка значения делителя (35) соответствующего
UBRR1L=35;        скорости приема данных 19200 бит/с;
for (i=0; i<=19; i++) { организовать цикл на 20 итераций;
delay_ms(20);      установить временную задержку 20 мс;
UDR1=i; }         выполнить передачу значения параметра цикла i;
                    }                                       завершающая операторная скобка программы;

```

Таким образом, 20 числовых значений (от 0 до 19) будут переданы в ПЭВМ и будут зафиксированы предварительно запущенной программой Terminal с соответствующими настройками протокола обмена данными.

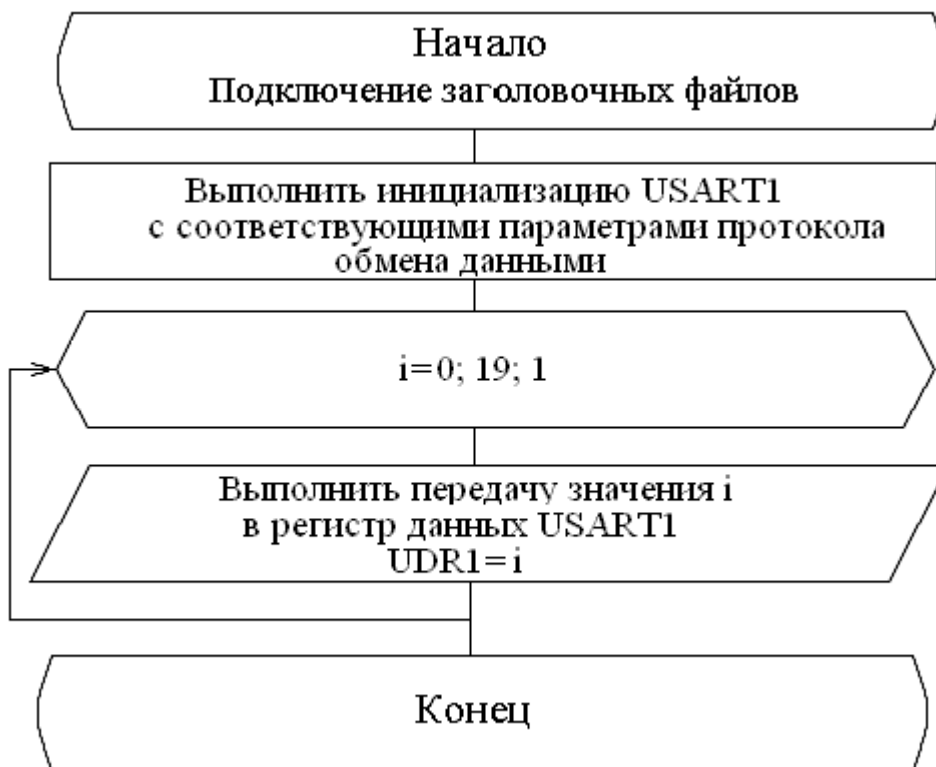


Рисунок 5.7 – Алгоритм программы передачи данных из микроконтроллера T AVR ATMEGA 128 в ПЭВМ по интерфейсу RS232C

Таблица 5.4 – Варианты индивидуальных заданий

№ п.п.	Задание
1	Разработать программу передачи 100 чисел (от 0 до 99) из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу

	RS232C в соответствии с протоколом: модуль USART1, скорость обмена данными 19200 бит/с, режим обмена асинхронный, 8 битов данных без бита четности.
2	Разработать программу передачи 50 чисел (от 20 до 69) из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу RS232C в соответствии с протоколом: модуль USART1, скорость обмена данными 38400 бит/с, режим обмена асинхронный, 8 битов данных без бита четности.
3	Разработать программу передачи 20 чисел (от 10 до 29) из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу RS232C в соответствии с протоколом: модуль USART1, скорость обмена данными 57600 бит/с, режим обмена асинхронный, 7 битов данных без бита четности.
4	Разработать программу передачи 10 чисел (от 0 до 9) из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу RS232C в соответствии с протоколом: модуль USART1, скорость обмена данными 14400 бит/с, режим обмена асинхронный, 6 битов данных без бита четности.
5	Разработать программу передачи 20 чисел (от 10 до 29) из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу RS232C в соответствии с протоколом: модуль USART1, скорость обмена данными 19200 бит/с, режим обмена асинхронный, 7 битов данных без бита четности, данные передаются через каждую секунду.*
6	Разработать программу передачи 50 чисел (от 10 до 59) из ПЭВМ в микроконтроллер AVR ATMEGA 128 по интерфейсу RS232C в соответствии с протоколом: модуль USART1, скорость обмена данными 14400 бит/с, режим обмена асинхронный, 7 битов данных без бита четности. При получении последнего информационного кадра выдать сигнал завершения приема на блок светодиодной индикации.
7	Разработать программу передачи 200 чисел (от 0 до 199) из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу RS232C в соответствии с протоколом: модуль USART1, скорость обмена данными 38400 бит/с, режим обмена асинхронный, 8 битов данных без бита четности.
8	Разработать программу передачи номера нажатой клавиши 3-х кнопочной клавиатуры (см. лабораторную работу №2) из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу RS232C в соответствии с протоколом: модуль USART1, скорость обмена данными 19200 бит/с, режим обмена асинхронный, 8 битов данных без бита четности.*
9	Разработать программу передачи номера нажатой клавиши

	матричной клавиатуры (см. лабораторную работу №2) из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу RS232C в соответствии с протоколом: модуль USART1, скорость обмена данными 14400 бит/с, режим обмена асинхронный, 8 битов данных без бита четности.*
10	Разработать программу передачи 10 чисел (от 0 до 9) из ПЭВМ в микроконтроллер AVR ATMEGA 128 по интерфейсу RS232C в соответствии с протоколом: модуль USART1, скорость обмена данными 19200 бит/с, режим обмена асинхронный, 8 битов данных без бита четности. Выполнить индикацию принятых данных на экране цифрового индикатора.*

* задания повышенной сложности.

5.4 Содержание отчета

В отчете необходимо привести следующее:
 характеристики лабораторной вычислительной системы;
 исходный модуль разработанной программы;
 анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности использования встроенных в микроконтроллер модулей USART при реализации обмена данными между лабораторным макетом и ПЭВМ.

5.5 Контрольные вопросы и задания

1. Поясните принципы передачи информации по последовательным и параллельным интерфейсам.
2. Назовите современные универсальные интерфейсы и приведите их основные характеристики.
3. Поясните принципы обмена данными по интерфейсу RS232C.
4. Какие регистры используются для настройки параметров передачи данных с помощью встроенного в микроконтроллер AVR MEGA128 блока USART ?
5. Какие сигналы прерываний могут генерироваться блоком USART ?
6. Поясните формат кадра при обмене данными по интерфейсу RS-232C.

6 ИЗУЧЕНИЕ ПРИНЦИПОВ РАБОТЫ СО ВСТРОЕННЫМ В МИКРОКОНТРОЛЛЕР АНАЛОГО-ЦИФРОВЫМ ПРЕОБРАЗОВАТЕЛЕМ НА ПРИМЕРЕ ИЗМЕРЕНИЯ ТЕМПЕРАТУРЫ С ПОМОЩЬЮ АНАЛОГОВОГО ТЕРМОДАТЧИКА.

Цель работы: изучить принципы функционирования встроенного в микроконтроллер AVR ATMEGA128 АЦП и методику измерения температуры с помощью аналогового термодатчика.

6.1 Указания по организации самостоятельной работы

Перед работой необходимо проработать теоретический материал по литературе [1, 4] и конспект лекций, ознакомиться принципами функционирования и возможностями программирования встроенного в микроконтроллер AVR ATMEGA 128 АЦП, изучить методику измерения температуры с помощью аналогового термодатчика.

В состав микроконтроллера AVR ATMEGA 128 входит 10-разрядный аналого-цифровой преобразователь (АЦП), реализующий принцип последовательного приближения (см. рисунок 6.1). На входе модуля АЦП имеется 8-ми канальный аналоговый мультиплексор, управляющий переключением данных, поступающих с восьми каналов с несимметричными входами. В качестве источника опорного напряжения может выступать напряжение от внутреннего источника U_{REF} (2.56 В), напряжение питания микроконтроллера U_{CC} , или внешнего источника. Наибольшая точность преобразования достигается при тактовой частоте модуля АЦП порядка $50 \div 200$ кГц. Модуль АЦП может функционировать в режимах одиночного и непрерывного преобразований. Стандартное преобразование аналогового значения в цифровое с помощью встроенного АЦП выполняется за 13 тактов, одиночное – за 25 тактов. Результат U_{ADC} преобразования для каналов с несимметричным входом определяется из выражения:

$$U_{ADC} = \frac{U_{IN} \cdot 1024}{U_{REF}},$$

где U_{IN} – значение входного напряжения, U_{REF} – величина опорного напряжения.

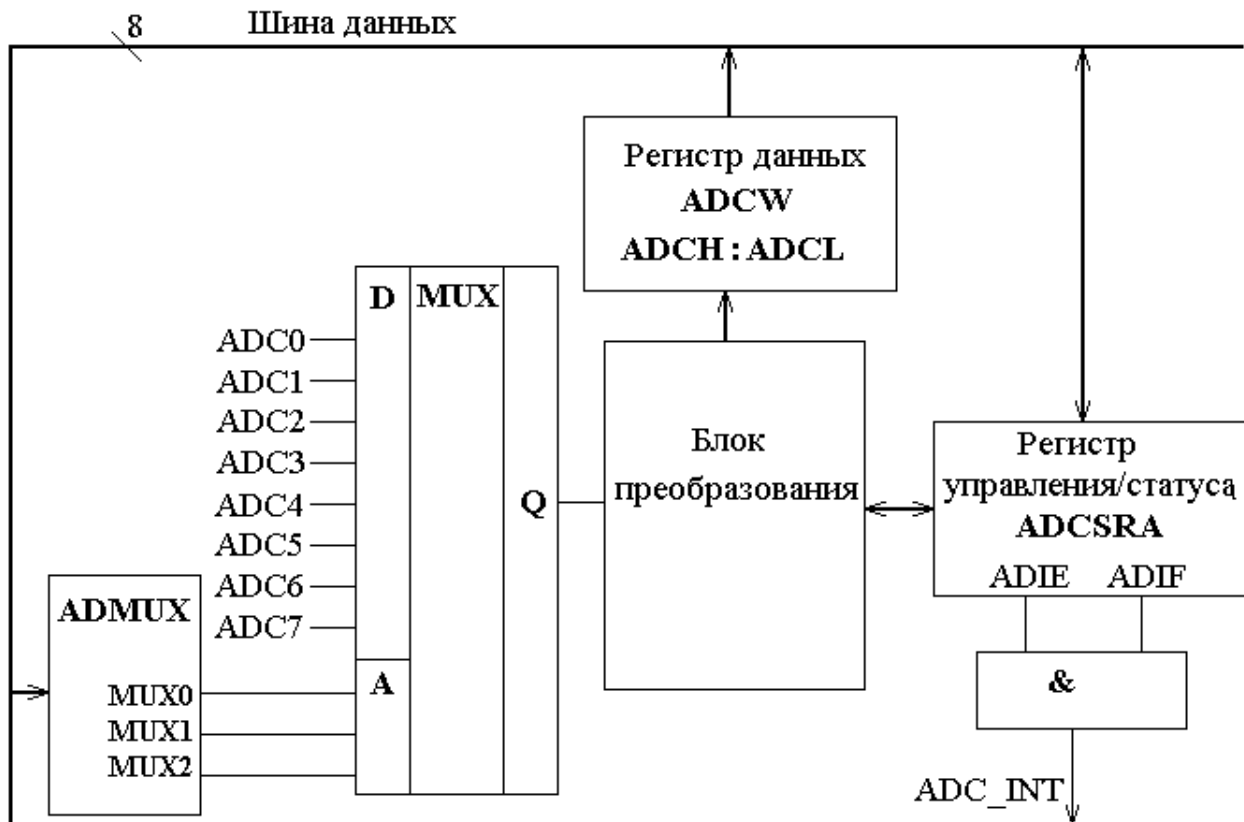


Рисунок 6.1 – Обобщенная структурная схема модуля АЦП в микроконтроллере AVR ATMEGA 128

Для работы с модулем АЦП используются следующие регистры:

ADCW (ADCH, ADCL) – 16-разрядный регистр данных, расположенный по адресу 4h/24h. В регистре используется 10 разрядов, выровненных по левой или правой границе.

ADCSRA – регистр управления/статуса.

ADMUX – регистр управления мультиплексором входных каналов.

Форматы и описания отдельных битов регистров **ADCSRA** и **ADMUX** приводится на рисунках 6.2, 6.3 и в таблицах 6.1, 6.2 соответственно.

№ бита	7	6	5	4	3	2	1	0
06h (26h)	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0

Рисунок 6.2 – Регистр состояния ADCSRA

Таблица 6.1 – Описание значений управляющих битов регистра **ADCSRA**

Разряд	Обозначение	Описание			
7	ADEN	Разрешение работы АЦП: 0 – выкл.; 1- вкл.			
6	ADSC	Запуск преобразования (1 – начать преобразование)			
5	ADFR	Выбор режима работы АЦП: 1 – режим непрерывного преобразования, 0 – режим одиночного преобразования			
4	ADIF	Флаг прерывания (1 – произошло прерывание от АЦП). Сбрасывается аппаратно при переходе на процедуру обработки прерывания.			
3	ADIE	Обмен по прерыванию (1–разрешение прерывания)			
2 – 0	ADPS2- ADPS0	Управление предделителем тактовой частоты АЦП			
		ADPS2	ADPS1	ADPS0	k
		0	0	0	2
		0	0	1	2
		0	1	0	4
		0	1	1	8
		1	0	0	16
		1	0	1	32
		1	1	0	64
1	1	1	128		

№ бита	7	6	5	4	3	2	1	0
07h (27h)	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

Рисунок 6.3 – Регистр состояния ADMUX

Таблица 6.2 – Описание значений управляющих битов регистра ADMUX.

Разряд	Обозначение	Описание		
7, 6	REFS1, REFS0	Выбор источника опорного напряжения (ИОН)		
		REFS1	REFS0	ИОН
		0	0	Внешний, AREF
		0	1	Напряжение питания U_{CC}
		1	0	Зарезервировано
1	1	Внутренний (2,56 В), U_{REF}		
5	ADLAR	Выравнивание 10-битового результата по границе слова: 0 – по правой, 1 – по левой.		
4,3,2,1,0	MUX4 – MUX0	Управление мультиплексором входных каналов: при использовании несимметричных входов код на линиях MUX2–MUX0 соответствует номеру канала.		

Согласно формату регистров **ADCSRA** и **ADMUX** для работы с модулем АЦП в режиме одиночного преобразования необходимо при каждом чтении данных из регистра **ADCW** выполнить следующий инициализирующий код на языке C:

ADMUX=0b11000000; используется внутренний источник опорного напряжения U_{REF} (2.56 В), оцифровывание данных, поступающих по 0-му входному каналу;

ADCSRA=0b11000000; запуск модуля АЦП в режиме одиночного преобразования.

Если используется режим, в котором модуль АЦП генерирует прерывание ADC INT (см. таблицу 4.1) по окончанию преобразования, то необходимо обязательно описать процедуру обработки данного прерывания.

6.2 Описание лабораторной установки

Лабораторная работа выполняется в индивидуальном порядке. На каждом рабочем месте должны быть установлены: многофункциональный лабораторный макет на базе микроконтроллера AVR ATMEGA 128, ПЭВМ типа IBM PC/AT с установленным программным обеспечением: операционной системой MS-WINDOWS v. 9x, 2000, XP, программатором на основе кросс-компилятора языка программирования C CodeVision AVR, утилитой Terminal для работы с последовательным интерфейсом RS232C. Задания выполняются на лабораторном макете на базе 8-ми разрядного микроконтроллера AVR ATMEGA 128, к 0-му выводу порта F которого подключен аналоговый датчик температуры Analog Devices TMP-35. Данный термодатчик обладает следующими характеристиками:

Диапазон рабочих напряжений: 2,7 – 5,5 В.

Диапазон измеряемых температур: - 40°C - +125°C.

Погрешность измерений: $\pm 2^\circ\text{C}$

Масштабирующий коэффициент: 10 мВ/1°C.

Для функционирования термодатчика используется схема подключения, изображенная на рисунке 6.4. Уровень логического нуля на входе \overline{SD} переводит выход термодатчика в высокоимпедансное состояние. Если отключение термодатчика в процессе работы системы не предусматривается, то вход \overline{SD} подключается к выводу V_{REF} .

Дополнительно в работе используется кабель с 9-контактными разъемами DB-9 (см. рисунок 1.8) для соединения лабораторного макета с ПЭВМ через последовательный интерфейс RS232C.

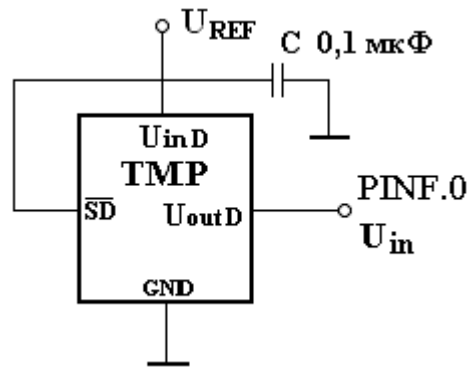


Рисунок 6.4 – Схема подключения датчика температуры TMP–35 к лабораторному макету

Измерение температуры (в °С) с помощью термодатчика TMP–35 выполняется путем преобразования оцифрованного значения входного напряжения (по отношению к лабораторному макету) по следующей формуле:

$$T = \frac{U_{IN} \cdot V_{REF} - V_{OFF}}{1024 \cdot 10}, \quad (6.1)$$

где U_{IN} – значение входного напряжения, U_{REF} – величина опорного напряжения; U_{OFF} – напряжение смещения (500 мВ).

6.3 Порядок проведения работы и указания по ее выполнению

Перед началом выполнения практической части лабораторной работы проводится экспресс–контроль знаний по принципам функционирования модуля АЦП, входящего в состав микроконтроллера AVR ATMEGA 128. При подготовке к лабораторной работе необходимо составить предварительный вариант листинга программы, в соответствии с индивидуальным заданием (см. таблицу 6.3).

Задание: разработать в среде программирования Code Vision AVR программу на языке С для измерения значений температуры с помощью термодатчика Analog Devices TMP–35 в соответствии с параметрами режима работы, приведенными в таблице 6.3.

Порядок выполнения задания:

1. Включить лабораторный макет (установить выключатель электропитания в положение I, и убедиться в свечении индикатора электропитания красным цветом).
2. Запустить компилятор Code Vision AVR.
3. Создать пустой проект.
4. Создать файл ресурса для кода программы и подключить его к проекту.

5. Ввести код исходного модуля программы для считывания данных с модуля АЦП.
6. Выполнить компиляцию (нажав клавишу **F9**) исходного модуля программы и устранить ошибки, полученные на данном этапе.
7. Настроить параметры программатора.
8. Создать загрузочный модуль программы (нажав комбинацию клавиш **Shift+F9**) и выполнить программирование микроконтроллера.
9. Проверить работоспособность загруженной в микроконтроллер программы и показать результаты работы преподавателю.
10. В случае некорректной работы разработанной программы, выполнить аппаратный сброс микроконтроллера, провести отладку исходного модуля программы и заново проверить функционирование программы, повторив выполнение пункта 9.

Пример выполнения задания: разработать программу для передачи данных о температуре в ПЕВМ по интерфейсу RS232C: режим работы АЦП – непрерывное преобразование, значение делителя частоты – 128; если величина температуры превысит 27°C , то выдать сигнал предупреждения с помощью включения светодиода.

Решение: исходя из параметров режима работы модуля АЦП необходимо настроить регистры **ADCSRA** и **ADMUX**, настроить порт **D** на вывод данных и погасить все светодиоды, инициализировать модуль **USART1**, в цикле считывать данные из регистра **ADCW**, вычислять значения температуры по формуле 6.1, выводить их в регистр **UDR1** и анализировать величину температуры для формирования сигнала предупреждения. Алгоритм программы приводится на рисунке 6.5, полный текст исходного модуля программы на языке **C** с подробными комментариями приводится ниже:

```

#include <mega128.h> Подключить заголовочный файл mega128.h;
#include <delay.h> подключить заголовочный файл delay.h;
#define VREF 2560L задание константы  $U_{REF}$ ;
#define OFFSET 500L задание константы  $U_{OFF}$ ;
int a; описание глобальной переменной a;
int read_adc() { описание подпрограммы считывания данных
int result; из модуля АЦП и вычисления значения
result = ADCW; температуры по формуле 6.1;
result=((result*VREF)/1024 -OFFSET)/10;
return result; }

main() { основная часть программы;
DDRD=0xFF; установка всех линий порта D в режим вывода дан-
PORTD=0xFF; ных и выключение всех светодиодов;

```

инициализация модуля АЦП

ADMUX=0b11000000; используется внутренний источник опорного напряжения U_{REF} (2.56 В), оцифровывание данных, поступающих по 0-му входному каналу;

ADCSRA=0b11101111; запуск модуля АЦП в режиме непрерывного преобразования, делитель частоты равен 128;

инициализация модуля USART1:

UCSR1A=0x00; установка стандартного режима задания скорости передачи данных;

UCSR1B=0x08; установка 3-го бита регистра **UCSR1B** для инициализации USART1 в режиме передатчика;

UCSR1C=0x06; установка формата кадра: 8 бит данных с отключенным режимом четности;

UBRR1H=0x00; установка значения делителя (35) соответствующего

UBRR1L=35; скорости приема данных 19200 бит/с;

while (1){ организовать цикл с бесконечным числом итераций;

a=read_adc(); получить новое значение температуры;

delay_ms(100); установить временную задержку 100 мс;

UDR1=a; выполнить передачу значения температуры по интерфейсу RS232C в ПЭВМ;

if (a>27) PORTD.1=0; зажечь 1-й светодиод блока индикации если значение

else PORTD.1=1; } температуры превышает 27°C;

} завершающая операторная скобка программы;

Таким образом, программа будет осуществлять измерение температуры и передавать данные в ПЭВМ по интерфейсу RS232C, а так же формировать сигнал предупреждения о повышении температуры на величину более 27°C. Данные, передаваемые в ПЭВМ, будут фиксироваться предварительно запущенной программой Terminal с соответствующими настройками протокола обмена данными.

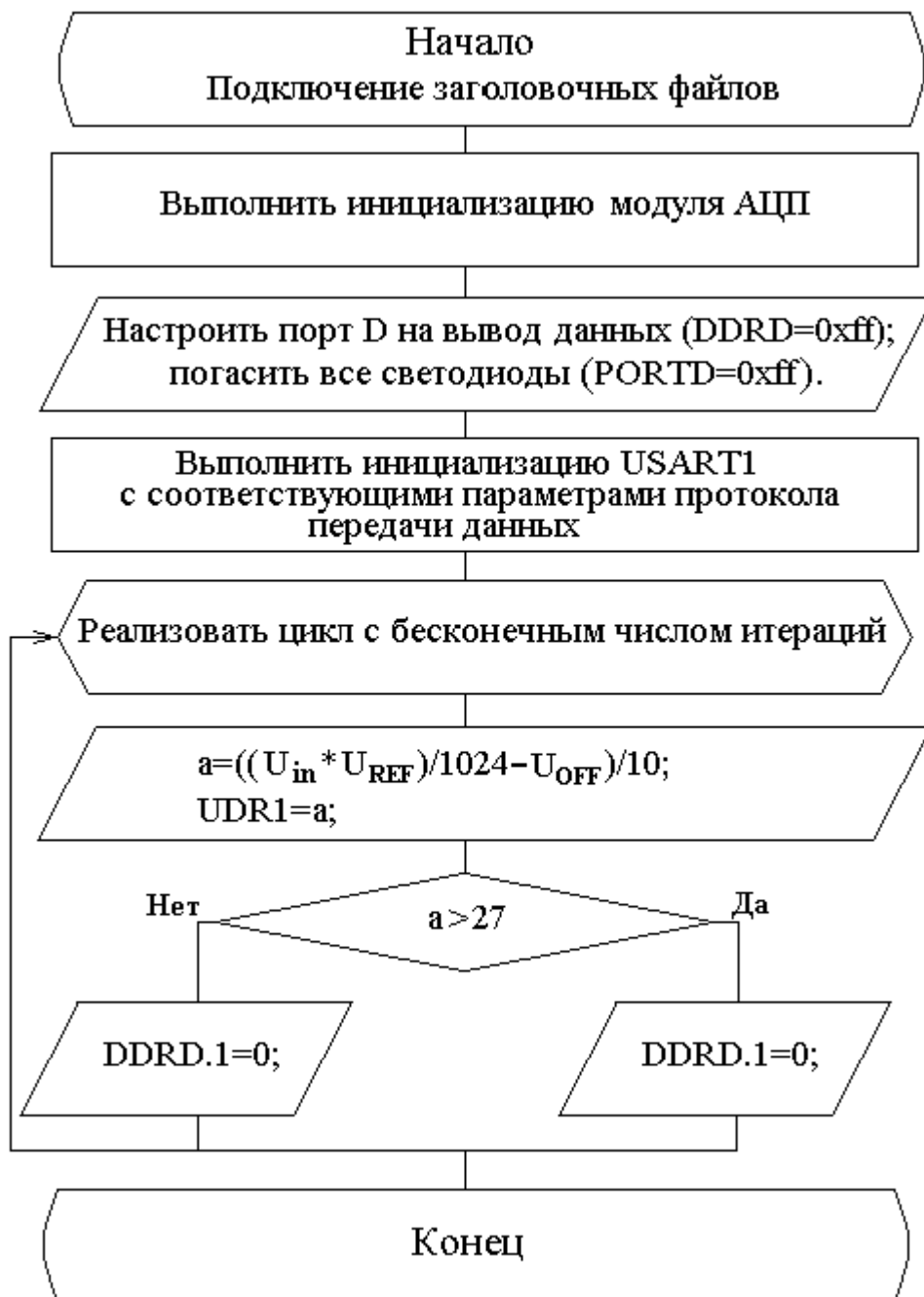


Рисунок 6.5 – Алгоритм программы передачи данных о температуре из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу RS232C с формированием сигнала предупреждения при повышении температуры

Таблица 6.4 – Варианты индивидуальных заданий

№ п.п.	Задание
1	Разработать программу, выполняющую измерение температуры в режиме одиночного преобразования (делитель частоты равен 2) и формирующую сигнал предупреждения с помощью блока светодиодов, если значение температуры превысит 30°C.

2	Разработать программу, выполняющую измерение температуры в режиме непрерывного преобразования (делитель частоты равен 8) и формирующую сигнал предупреждения с помощью блока светодиодов, если значение температуры превысит 40°C.
3	Разработать программу, выполняющую измерение температуры в режиме одиночного преобразования (делитель частоты равен 64) и формирующую сигналы предупреждения с помощью блока светодиодов, если значение температуры выходит за рамки диапазона $25^{\circ}\text{C} < T < 30^{\circ}\text{C}$.
4	Разработать программу, выполняющую измерение температуры в режиме непрерывного преобразования (делитель частоты равен 128) и формирующую сигналы предупреждения с помощью блока светодиодов, если значение температуры выходит за рамки диапазона $30^{\circ}\text{C} < T < 40^{\circ}\text{C}$.
5	Разработать программу, выполняющую измерение температуры в режиме непрерывного преобразования (делитель частоты равен 4) и формирующую сигналы предупреждения с помощью блока светодиодов, если значение температуры выходит за рамки диапазонов $25^{\circ}\text{C} < T < 30^{\circ}\text{C}$ и $40^{\circ}\text{C} < T < 50^{\circ}\text{C}$.
6	Разработать программу, выполняющую измерение температуры в режиме одиночного преобразования (делитель частоты равен 32) и формирующую сигналы предупреждения с помощью блока светодиодов, если значение температуры выходит за рамки диапазонов $20^{\circ}\text{C} < T < 25^{\circ}\text{C}$ и $35^{\circ}\text{C} < T < 40^{\circ}\text{C}$.
7	Разработать программу, выполняющую измерение температуры в режиме непрерывного преобразования (делитель частоты равен 16) и формирующую сигналы предупреждения с помощью цифрового индикатора Holtek HT1613, если значение температуры выходит за рамки диапазона $25^{\circ}\text{C} < T < 35^{\circ}\text{C}$.
8	Разработать программу, выполняющую измерение температуры в режиме непрерывного преобразования (делитель частоты равен 4) и формирующую сигналы предупреждения с помощью блока светодиодов, если значение температуры выходит за рамки диапазона $25^{\circ}\text{C} < T < 40^{\circ}\text{C}$. Обработка данных происходит по прерыванию ADC_INT*.
9	Разработать программу, выполняющую измерение температуры в режиме непрерывного преобразования (делитель частоты равен 8) и выводящую значения температуры на цифровой индикатор Holtek HT1613* .
10	Разработать программу, выполняющую измерение температуры в режиме непрерывного преобразования (делитель частоты равен 32) и передающую значения температуры в ПЭВМ с помощью последовательного интерфейса RS232C.*

* задания повышенной сложности.

6.4 Содержание отчета

В отчете необходимо привести следующее:
характеристики лабораторной вычислительной системы;
исходный модуль разработанной программы;
анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности использования встроенного в микроконтроллер модуля АЦП для измерения аналоговых величин.

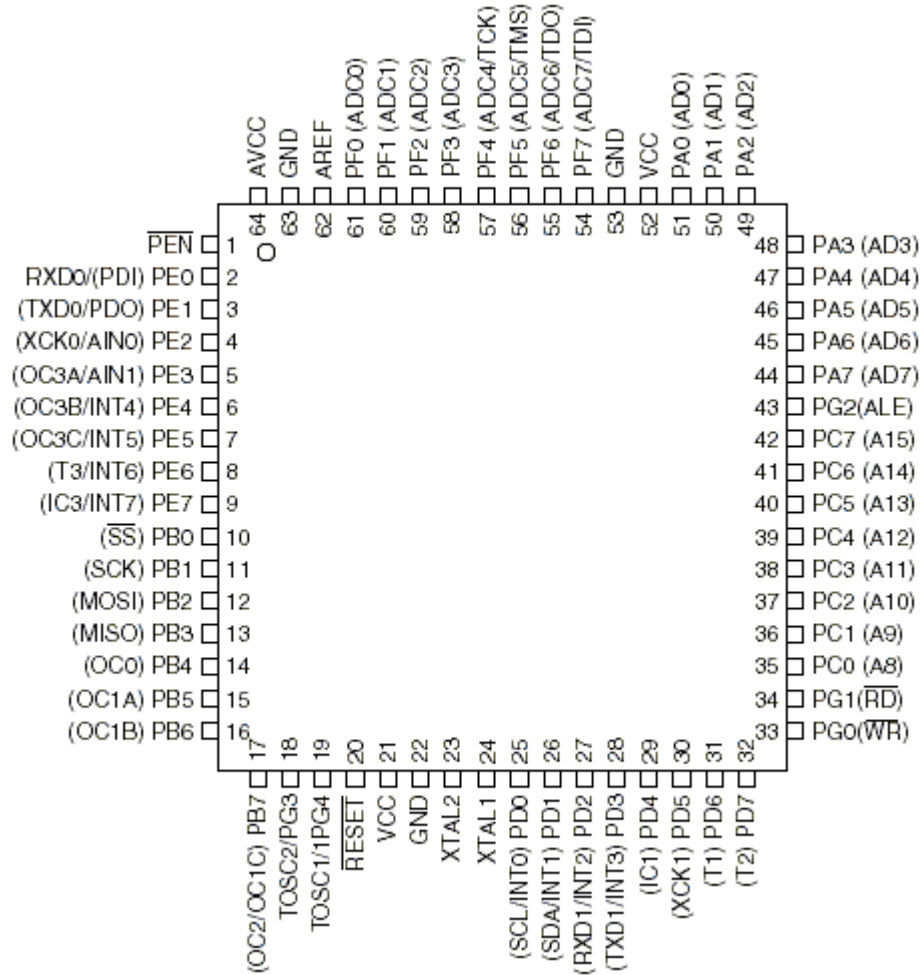
6.5 Контрольные вопросы и задания

1. Поясните принцип работы встроенного в микроконтроллер 10-разрядного АЦП.
2. Назовите основные управляющие регистры АЦП, встроенного в микроконтроллер, и поясните их функции.
3. Поясните принцип измерения температуры с помощью термодатчика.
4. Для чего применяется выравнивание результата по левой или правой границе слова ?
5. Поясните возможности передачи данных о температуре в ПЭВМ.
6. Поясните возможность отображения данных о температуре на экране цифрового индикатора Holtek HT1613, рассмотренного в лабораторной работе № 3.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы ATMEL. – М.: Изд. дом Додэка-XXI, 2004. – 560 с.
2. Голубцов М.С., Кириченко А.В. Микроконтроллеры AVR: от простого к сложному.- М.: СОЛОН-Пресс, 2004. – 304 с.
3. Баранов В.Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы.- М.: Издательский дом “Додэка-XXI”, 2004. – 288 с.
4. <http://www.atmel.ru/> – описание микроконтроллеров фирмы ATMEL на русском языке.

ПРИЛОЖЕНИЕ 1
 РАСПОЛОЖЕНИЕ ВЫВОДОВ МИКРОКОНТРОЛЛЕРА
 AVR ATMEGA 128



ПРИЛОЖЕНИЕ 2
ИНСТРУКЦИИ ПРОЦЕССОРОВ AVR

Таблица П.2.1 – Арифметические и логические команды.

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
ADD	Rd,Rr	Суммирование без переноса	$Rd = Rd + Rr$	Z,C,N,V, H,S	1
ADC	Rd,Rr	Суммирование с переносом	$Rd = Rd + Rr + C$	Z,C,N,V, H,S	1
SUB	Rd,Rr	Вычитание без переноса	$Rd = Rd - Rr$	Z,C,N,V, H,S	1
SUBI	Rd,K8	Вычитание константы	$Rd = Rd - K8$	Z,C,N,V, H,S	1
SBC	Rd,Rr	Вычитание с переносом	$Rd = Rd - Rr - C$	Z,C,N,V, H,S	1
SBCI	Rd,K8	Вычитание константы с переносом	$Rd = Rd - K8 - C$	Z,C,N,V, H,S	1
AND	Rd,Rr	Логическое И	$Rd = Rd \cdot Rr$	Z,N,V,S	1
ANDI	Rd,K8	Логическое И с константой	$Rd = Rd \cdot K8$	Z,N,V,S	1
OR	Rd,Rr	Логическое ИЛИ	$Rd = Rd \vee Rr$	Z,N,V,S	1
ORI	Rd,K8	Логическое ИЛИ с константой	$Rd = Rd \vee K8$	Z,N,V,S	1
EOR	Rd,Rr	Логическое исключающее ИЛИ	$Rd = Rd \oplus Rr$	Z,N,V,S	1
COM	Rd	Побитная Инверсия	$Rd = \$FF - Rd$	Z,C,N,V,S	1
NEG	Rd	Изменение знака (Доп. код)	$Rd = \$00 - Rd$	Z,C,N,V, H,S	1
SBR	Rd,K8	Установить бит (биты) в регистре	$Rd = Rd \vee K8$	Z,C,N,V,S	1
CBR	Rd,K8	Сбросить бит (биты) в регистре	$Rd = Rd \cdot (\$FF - K8)$	Z,C,N,V,S	1
INC	Rd	Инкрементировать значение регистра	$Rd = Rd + 1$	Z,N,V,S	1
DEC	Rd	Декрементировать значение регистра	$Rd = Rd - 1$	Z,N,V,S	1
TST	Rd	Проверка на ноль либо отрицательность	$Rd = Rd \cdot Rd$	Z,C,N,V,S	1
CLR	Rd	Очистить регистр	$Rd = 0$	Z,C,N,V,S	1
SER	Rd	Установить регистр	$Rd = \$FF$	None	1
ADIW	Rd1,K6	Сложить константу и слово	$Rdh:Rdl = Rdh:Rdl + K6$	Z,C,N,V,S	2
SBIW	Rd1,K6	Вычесть константу из слова	$Rdh:Rdl = Rdh:Rdl - K6$	Z,C,N,V,S	2
MUL	Rd,Rr	Умножение чисел без знака	$R1:R0 = Rd * Rr$	Z,C	2
MULS	Rd,Rr	Умножение чисел со знаком	$R1:R0 = Rd * Rr$	Z,C	2
MULSU	Rd,Rr	Умножение числа со знаком с числом без знака	$R1:R0 = Rd * Rr$	Z,C	2
FMUL	Rd,Rr	Умножение дробных чисел без знака	$R1:R0 = (Rd * Rr) \ll 1$	Z,C	2
FMULS	Rd,Rr	Умножение дробных чисел со знаком	$R1:R0 = (Rd * Rr) \ll 1$	Z,C	2
FMULSU	Rd,Rr	Умножение дробного числа со знаком с числом без знака	$R1:R0 = (Rd * Rr) \ll 1$	Z,C	2

Таблица П.2.2. – Команды ветвления

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
RJMP	k	Относительный переход	$PC = PC + k + 1$	None	2
IJMP	Нет	Косвенный переход на (Z)	$PC = Z$	None	2
EIJMP	Нет	Расширенный косвенный переход на (Z)	$STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND$	None	2
JMP	k	Переход	$PC = k$	None	3
RCALL	k	Относительный вызов подпрограммы	$STACK = PC+1, PC = PC+k+1$	None	3/4*
ICALL	Нет	Косвенный вызов (Z)	$STACK = PC+1, PC = Z$	None	3/4*
EICALL	Нет	Расширенный косвенный вызов (Z)	$STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND$	None	4*
CALL	k	Вызов подпрограммы	$STACK = PC+2, PC = k$	None	4/5*
RET	Нет	Возврат из подпрограммы	$PC = STACK$	None	4/5*
RETI	Нет	Возврат из прерывания	$PC = STACK$	I	4/5*
CPSE	Rd,Rr	Сравнить, пропустить если равны	$if (Rd == Rr) PC = PC + 2 \text{ or } 3$	None	1/2/3
CP	Rd,Rr	Сравнить	$Rd - Rr$	Z,C,N, V,H,S	1
CPC	Rd,Rr	Сравнить с переносом	$Rd - Rr - C$	Z,C,N, V,H,S	1
CPI	Rd,K ₈	Сравнить с константой	$Rd - K$	Z,C,N, V,H,S	1
SBRC	Rr,b	Пропустить если бит в регистре очищен	$if(Rr(b)==0) PC = PC+2 \text{ or } 3$	None	1/2/3
SBRS	Rr,b	Пропустить если бит в регистре установлен	$if(Rr(b)==1) PC = PC+2 \text{ or } 3$	None	1/2/3
SBIC	P,b	Пропустить если бит в порту очищен	$if(I/O(P,b)==0) PC = PC+ \text{ or } 3$	None	1/2/3
SBIS	P,b	Пропустить если бит в порту установлен	$if(I/O(P,b)==1) PC = PC+2 \text{ or } 3$	None	1/2/3
BRBC	s,k	Перейти если флаг в SREG очищен	$if(SREG(s)==0) PC = PC+k+1$	None	1/2
BRBS	s,k	Перейти если флаг в SREG установлен	$if(SREG(s)==1) PC = PC+k+1$	None	1/2
BREQ	k	Перейти если равно	$if(Z==1) PC = PC + k + 1$	None	1/2
BRNE	k	Перейти если не равно	$if(Z==0) PC = PC + k + 1$	None	1/2
BRCS	k	Перейти если перенос установлен	$if(C==1) PC = PC + k + 1$	None	1/2
BRCC	k	Перейти если перенос очищен	$if(C==0) PC = PC + k + 1$	None	1/2
BRSH	k	Перейти если равно или больше	$if(C==0) PC = PC + k + 1$	None	1/2
BRLO	k	Перейти если меньше	$if(C==1) PC = PC + k + 1$	None	1/2
BRMI	k	Перейти если минус	$if(N==1) PC = PC + k + 1$	None	1/2
BRPL	k	Перейти если плюс	$if(N==0) PC = PC + k + 1$	None	1/2
BRGE	k	Перейти если больше или равно (со знаком)	$if(S==0) PC = PC + k + 1$	None	1/2
BRLT	k	Перейти если меньше (со знаком)	$if(S==1) PC = PC + k + 1$	None	1/2

BRHS	k	Перейти если флаг внутреннего переноса установлен	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Перейти если флаг внутреннего переноса очищен	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Перейти если флаг T установлен	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Перейти если флаг T очищен	if(T==0) PC = PC + k + 1	None	1/2
BRVS	k	Перейти если флаг переполнения установлен	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Перейти если флаг переполнения очищен	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Перейти если прерывания разрешены	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Перейти если прерывания запрещены	if(I==0) PC = PC + k + 1	None	1/2

* Для операций доступа к данным количество циклов указано при условии доступа к внутренней памяти данных, и не корректно при работе с внешним ОЗУ. Для инструкций CALL, ICALL, EICALL, RCALL, RET и RETI, необходимо добавить три цикла плюс по два цикла для каждого ожидания в контроллерах с PC меньшим 16 бит (128KB памяти программ). Для устройств с памятью программ свыше 128KB, добавьте пять циклов плюс по три цикла на каждое ожидание.

Таблица П.2.3 – Команды пересылки данных

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
MOV	Rd,Rr	Скопировать регистр	$Rd = Rr$	–	1
MOVW	Rd,Rr	Скопировать пару регистров	$Rd+1:Rd = Rr+1:Rr$, $r,d \text{ even}$	–	1
LDI	Rd,K8	Загрузить константу	$Rd = K$	–	1
LDS	Rd,k	Прямая загрузка	$Rd = (k)$	–	2*
LD	Rd,X	Косвенная загрузка	$Rd = (X)$	–	2*
LD	Rd,X+	Косвенная загрузка с пост-инкрементом	$Rd = (X)$, $X=X+1$	–	2*
LD	Rd,-X	Косвенная загрузка с пре-декрементом	$X=X-1$, $Rd = (X)$	–	2*
LD	Rd,Y	Косвенная загрузка	$Rd = (Y)$	–	2*
LD	Rd,Y+	Косвенная загрузка с пост-инкрементом	$Rd = (Y)$, $Y=Y+1$	–	2*
LD	Rd,-Y	Косвенная загрузка с пре-декрементом	$Y=Y-1$, $Rd = (Y)$	–	2*
LDD	Rd,Y+q	Косвенная загрузка с замещением	$Rd = (Y+q)$	–	2*
LD	Rd,Z	Косвенная загрузка	$Rd = (Z)$	–	2*
LD	Rd,Z+	Косвенная загрузка с пост-инкрементом	$Rd = (Z)$, $Z=Z+1$	–	2*
LD	Rd,-Z	Косвенная загрузка с пре-декрементом	$Z=Z-1$, $Rd = (Z)$	–	2*
LDD	Rd,Z+q	Косвенная загрузка с замещением	$Rd = (Z+q)$	–	2*
STS	k,Rr	Прямое сохранение	$(k) = Rr$	–	2*
ST	X,Rr	Косвенное сохранение	$(X) = Rr$	–	2*
ST	X+,Rr	Косвенное сохранение с пост-инкрементом	$(X) = Rr$, $X=X+1$	–	2*
ST	-X,Rr	Косвенное сохранение с пре-декрементом	$X=X-1$, $(X)=Rr$	–	2*
ST	Y,Rr	Косвенное сохранение	$(Y) = Rr$	–	2*
ST	Y+,Rr	Косвенное сохранение с пост-инкрементом	$(Y) = Rr$, $Y=Y+1$	–	2
ST	-Y,Rr	Косвенное сохранение с пре-декрементом	$Y=Y-1$, $(Y) = Rr$	–	2
ST	Y+q,Rr	Косвенное сохранение с замещением	$(Y+q) = Rr$	–	2
ST	Z,Rr	Косвенное сохранение	$(Z) = Rr$	–	2
ST	Z+,Rr	Косвенное сохранение с пост-инкрементом	$(Z) = Rr$, $Z=Z+1$	–	2
ST	-Z,Rr	Косвенное сохранение с пре-декрементом	$Z=Z-1$, $(Z) = Rr$	–	2
ST	Z+q,Rr	Косвенное сохранение с замещением	$(Z+q) = Rr$	–	2
LPM	Нет	Загрузка из программной памяти	$R0 = (Z)$	–	3
LPM	Rd,Z	Загрузка из программной памяти	$Rd = (Z)$	–	3
LPM	Rd,Z+	Загрузка из программной памяти с пост-инкрементом	$Rd = (Z)$, $Z=Z+1$	–	3
ELPM	Нет	Расширенная загрузка из памяти программ	$R0 = (RAMPZ:Z)$	–	3
ELPM	Rd,Z	Расширенная загрузка из памяти программ	$Rd = (RAMPZ:Z)$	–	3
ELPM	Rd,Z+	Расширенная загрузка из программной памяти с пост-инкрементом	$Rd = (RAMPZ:Z)$, $Z = Z+1$	–	3
SPM	Нет	Сохранение в программной памяти	$(Z) = R1:R0$	–	-
ESPM	Нет	Расширенное сохранение в памяти программ	$(RAMPZ:Z) = R1:R0$	–	-
IN	Rd,P	Чтение порта	$Rd = P$	–	1
OUT	P,Rr	Запись в порт	$P = Rr$	–	1
PUSH	Rr	Занесение регистра в стек	$STACK = Rr$	–	2
POP	Rd	Извлечение регистра из стека	$Rd = STACK$	–	2

* Для операций доступа к данным количество циклов указано при условии доступа к внутренней памяти данных, и не корректно при работе с внешним ОЗУ. Для инструкций LD, ST, LDD, STD, LDS, STS, PUSH и POP, необходимо добавить один цикл.

Таблица П.2.4 – Команды для работы с битами.

Мнемоника	Операнды	Описание	Операция	Флаги	Циклы
LSL	Rd	Логический сдвиг влево	$Rd(n+1)=Rd(n)$, $Rd(0)=0$, $C=Rd(7)$	Z,C,N,V, H,S	1
LSR	Rd	Логический сдвиг вправо	$Rd(n)=Rd(n+1)$, $Rd(7)=0$, $C=Rd(0)$	Z,C,N,V, S	1
ROL	Rd	Циклический сдвиг влево через C	$Rd(0)=C$, $Rd(n+1)=Rd(n)$, $C=Rd(7)$	Z,C,N,V, H,S	1
ROR	Rd	Циклический сдвиг вправо через C	$Rd(7)=C$, $Rd(n)=Rd(n+1)$, $C=Rd(0)$	Z,C,N,V, S	1
ASR	Rd	Арифметический сдвиг вправо	$Rd(n)=Rd(n+1)$, $n=0,\dots,6$	Z,C,N,V, S	1
SWAP	Rd	Перестановка тетрад	$Rd(3..0) = Rd(7..4)$, $Rd(7..4) = Rd(3..0)$	None	1
BSET	s	Установка флага	$SREG(s) = 1$	$SREG(s)$	1
BCLR	s	Очистка флага	$SREG(s) = 0$	$SREG(s)$	1
SBI	P,b	Установить бит в порту	$I/O(P,b) = 1$	None	2
CBI	P,b	Очистить бит в порту	$I/O(P,b) = 0$	None	2
BST	Rr,b	Сохранить бит из регистра в T	$T = Rr(b)$	T	1
BLD	Rd,b	Загрузить бит из T в регистр	$Rd(b) = T$	None	1
SEC	Нет	Установить флаг переноса	$C = 1$	C	1
CLC	Нет	Очистить флаг переноса	$C = 0$	C	1
SEN	Нет	Установить флаг отрицательного числа	$N = 1$	N	1
CLN	Нет	Очистить флаг отрицательного числа	$N = 0$	N	1
SEZ	Нет	Установить флаг нуля	$Z = 1$	Z	1
CLZ	Нет	Очистить флаг нуля	$Z = 0$	Z	1
SEI	Нет	Установить флаг прерываний	$I = 1$	I	1
CLI	Нет	Очистить флаг прерываний	$I = 0$	I	1
SES	Нет	Установить флаг числа со знаком	$S = 1$	S	1
CLN	Нет	Очистить флаг числа со знаком	$S = 0$	S	1
SEV	Нет	Установить флаг переполнения	$V = 1$	V	1
CLV	Нет	Очистить флаг переполнения	$V = 0$	V	1
SET	Нет	Установить флаг T	$T = 1$	T	1
CLT	Нет	Очистить флаг T	$T = 0$	T	1
SEN	Нет	Установить флаг внутреннего переноса	$H = 1$	H	1
CLH	Нет	Очистить флаг внутреннего переноса	$H = 0$	H	1
NOP	Нет	Нет операции	Нет	None	1
SLEEP	Нет	Спать (уменьшить энергопотребление)	Смотрите описание инструкции	None	1
WDR	Нет	Сброс сторожевого таймера	Смотрите описание инструкции	None	1

В языке Assembler нет различия в регистре символов.

Операнды могут быть следующих видов:

- Rd: Результирующий (и исходный) регистр в регистровом файле;
- Rr: Исходный регистр в регистровом файле;
- b: Константа (3 бита), или константное выражение;
- s: Константа (3 бита), или константное выражение;
- P: Константа (5-6 бит), или константное выражение;
- K6; Константа (6 бит), или константное выражение;
- K8: Константа (8 бит), или константное выражение;
- k: Константа (размер зависит от инструкции), или константное выражение;
- q: Константа (6 бит), или константное выражение;
- Rdl: R24, R26, R28, R30. Для инструкций ADIW и SBIW ;
- X,Y,Z: Регистры косвенной адресации (X=R27:R26, Y=R29:R28, Z=R31:R30).

Електронне навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторних робіт

з дисципліни
«МІКРОКОНТРОЛЕРИ В БІОМЕДИЧНИХ АПАРАТАХ»

для студентів усіх форм навчання
спеціальності 7.05140201, 8.05140201 «Біомедична інженерія»

Упорядники АВРУНІН Олег Григорович
КРУК Олег Ярославович
СЕМЕНЕЦЬ Валерій Васильович

Відповідальний випусковий А.І. Бих

Авторська редакція