Министерство образования и науки Украины

НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
"ХАРЬКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ"

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

по курсу

"ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА"

УТВЕРЖДЕНО
Кафедрой АУТС.
Протокол № от 2010

Харьков 2010

Методические указания к лабораторным работам по курсу "Вычислительная техника" для студентов всех специальностей / Составитель Зуев А.А. – НТУ "ХПИ", 2010. – 114 с.

Составитель: А.А. Зуев

В настоящее время микроконтроллеры принимают все большее распространение в автоматических системах управления различными техническими объектами. В связи с этим возникает задача по обучению практическим вопросам программированию микроконтроллеров и построению устройств на их основе и самостоятельной разработки законченных проектов.

В методических указаниях рассмотрены архитектура, аппаратные и программные средства микроконтроллеров фирмы ATMEL семейства AVR MEGA, на примере модели AVR ATMEGA 128.

Анализ методики программирования работы различных устройств в интегрированной среде разработки Code Vision AVR C позволил объединить теоретические понятия и практические рекомендации в процессе обучения разработке микроконтроллерных систем управления.

В курсе лабораторных работ рассматривается реализация девяти лабораторных работ. Он охватывает типовые задачи, возникающие перед разработчиком микроконтроллерных систем: управление блоком светодиодных индикаторов, считывание данных с клавиатуры и задание интервалов времени, вывод информации на графический индикатор, сопряжение микроконтроллера с ПЭВМ по последовательному интерфейсу RS232-C.

В приложениях приведены справочные данные о системе команд микроконтроллера AVR ATMEGA 128, таблице прерываний и устройстве таймера/счетчика.

СОДЕРЖАНИЕ

1	**ОБЩИЕ СВЕДЕНИЯ О МАКЕТЕ МЛ-1**		5
	1.1	Основные характеристики микроконтроллера AVR ATMEGA 128	5
	1.2	Программная модель микроконтроллера AVR MEGA128 и механизм работы с регистрами, памятью и портами ввода/вывода	7
	1.3	Синтаксис и основные операторы языка С	11
2	**НАСТРОЙКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**		17
	2.1	Настройка проекта в среде CodeVision	17
	2.2	Программирование микроконтроллера	21
3	**УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ**		24
	3.1	Ход лабораторной работы	24
	3.2	Описание блока светодиодов лабораторного макета	25
	3.3	Принципы анализа нажатия стандартных кнопок с помощью микроконтроллера AVR MEGA 128	26
	3.4	Принципы считывания данных с матричной клавиатуры с помощью микроконтроллера AVR ATMEGA 128 в режиме программного опроса.	27
	3.5	Библиотека для работы с ЖКИ лабораторного макета	29
	3.6	Работа с функцией sprintf стандартной библиотеки ввода-вывода	30
	3.7	Система прерываний микроконтроллера AVR ATMEGA 128	31
	3.7.1	Принципы функционирования аппаратных таймеров-счетчиков, входящих в состав микроконтроллера AVR ATMEGA 128	32
	3.8	Работа со стандартной математической библиотекой	39
	3.9	Последовательный интерфейс RS-232C	40
	3.9.1	Организация модулей USART в микроконтроллере AVR ATMEGA 128	41
4	**ЗАДАНИЯ НА ЛАБОРАТОРНЫЕ РАБОТЫ**		45
	4.1	Лабораторная работа №1 – Создание простейшей программы на языке С	45
	4.2	Лабораторная работа №2 – Циклическое управление группой светодиодов	49

	4.3	Лабораторная работа №3 – Обработка нажатий на клавиши	53
	4.4	Лабораторная работа №4 – Работа с матричной клавиатурой	57
	4.5	Лабораторная работа №5 – Синтез звука заданной частоты	62
	4.6	Лабораторная работа №6 – Вывод текстовых и числовых данных на индикатор	66
	4.7	Лабораторная работа №7 – Изучение принципов обработки прерываний	70
	4.8	Лабораторная работа №8 (повышенной сложности) – Вывод графической информации на ЖКИ	74
	4.9	Лабораторная работа №9 (повышенной сложности) – Обмен данными по интерфейсу RS-232C между микроконтроллером и ПЭВМ	78
А	**ПРИЛОЖЕНИЕ. Система команд микроконтроллера AVR MEGA128**		83
Б	**ПРИЛОЖЕНИЕ. Векторы прерываний микроконтроллера AVR ATMEGA 128**		92
В	**ПРИЛОЖЕНИЕ. Аппаратные таймеры/счетчики, входящие в состав микроконтроллера AVR ATMEGA 128**		99
Г	**ПРИЛОЖЕНИЕ. Исходные коды библиотеки для работы с ЖКИ**		100
РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА			107

1 ОБЩИЕ СВЕДЕНИЯ О МАКЕТЕ МЛ-1

1.1 Основные характеристики микроконтроллера AVR ATMEGA 128

AVR-архитектура объединяет высокопроизводительный RISC-процессор с раздельным 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.

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 раздельной 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 Кб;
- Объем статической оперативной памяти (ОЗУ) : 4 Кб;
- Объем памяти данных на основе электрически-стираемого ПЗУ (EEPROM): 4 Кб;
- 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–счётчики;
- широтно-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.

Структурная схема макета приведена на рис. 1.1.

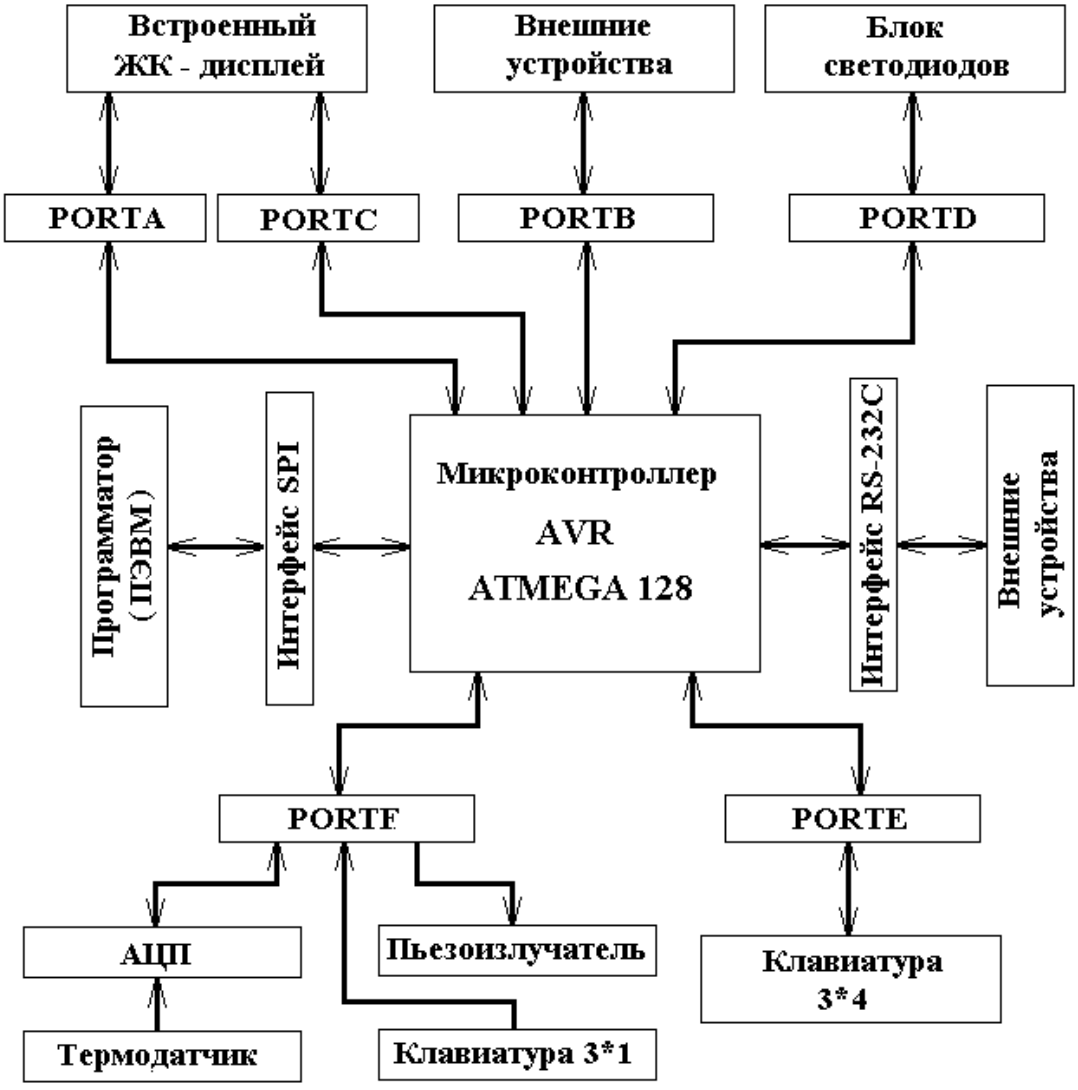


Рис. 1.1. – Структурная схема лабораторного макета на базе микроконтроллера AVR ATMEGA 128

1.2 Программная модель микроконтроллера AVR MEGA128 и механизм работы с регистрами, памятью и портами ввода/вывода

В микроконтроллере AVR ATMEGA128 реализована гарвардская архитектура, в соответствие с которой адресные пространства памяти программ и данных физически разделены (доступ к этим областям памяти осуществляется по раздельным 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 рис. 1.2. Память программ представляет собой электрически стираемое перепрограммируемое постоянное запоминающее устройство ППЗУ объемом 128 Кб, 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.

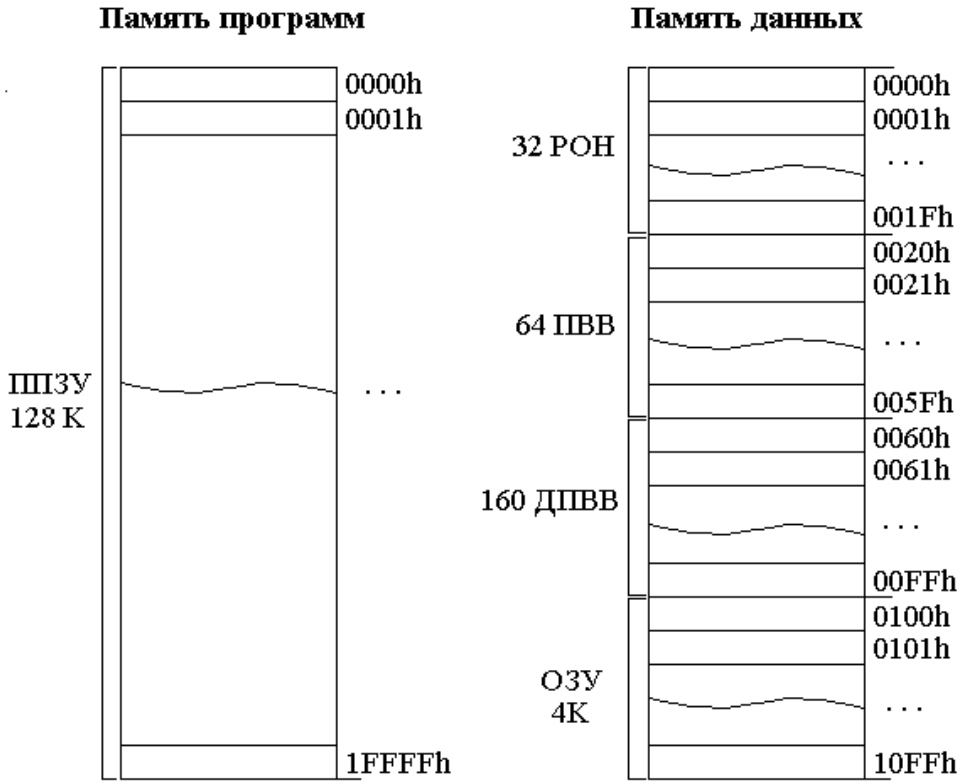


Рис. 1.2. – Распределение памяти в микроконтроллере AVR ATMEGA128

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 PC.

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. Регистровая память (см. рис. 1.3) включает 32 8-разрядных регистра общего назначения (R0 - R31), объединенных в регистровый файл. Каждый из регистров общего назначения непосредственно связан с АЛУ.

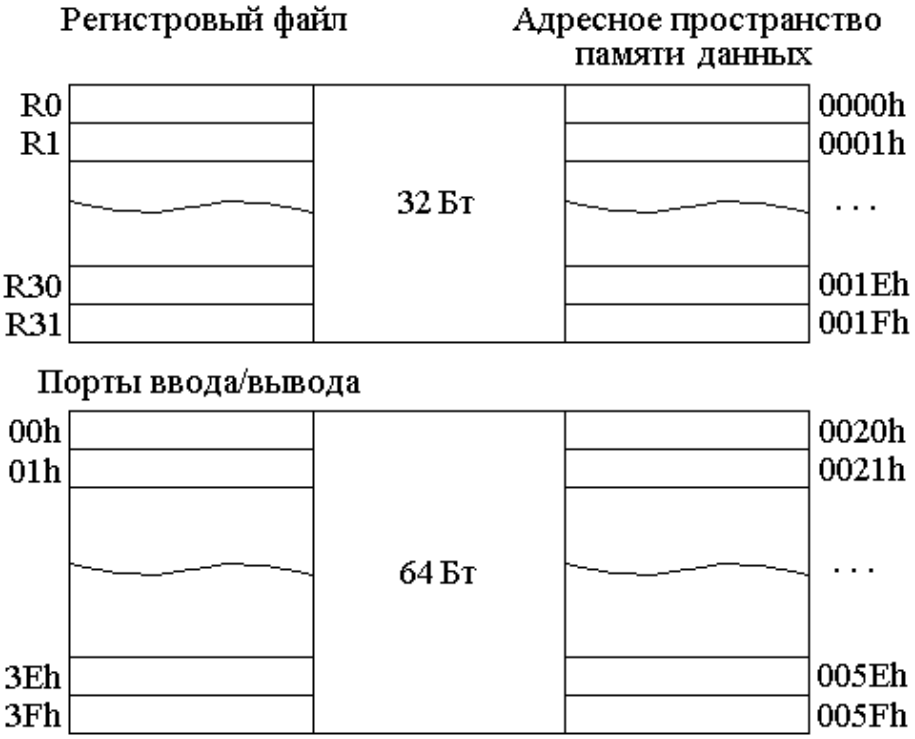


Рис. 1.3. – Иллюстрация отображения регистров общего назначения и портов ввода/вывода на адресное пространство памяти данных

АЛУ поддерживает арифметические и логические операции с регистрами, между регистром и константой или непосредственно с регистром. При исполнении 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. рис. 1.4).

X		Y		Z	
27	26	29	28	31	30

Рис. 1.4. – 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 (таблица 1.1).

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

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 Кб, занимающее диапазон адресов 0100h-10FFh.

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 рис. 1.5. Система команд микропроцессора описана в приложении А.

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

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 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.

Рис. 1.5. – Регистр состояния SREG

1.3 Синтаксис и основные операторы языка С

Сложные программные проекты можно реализовывать при помощи языка программирования С, который обладает возможностями языков низкого и высокого уровня, а так же большой библиотекой функций.

Алфавит языка С 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 С декларируются с помощью директивы #define в соответствие 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=0b1001.
```

Базовыми целыми типами данных в языке С являются: char (размер 1 байт, диапазон значений – 128 ÷ 127) и int (размер 2 байта, диапазон значений – 32768 ÷ 32767). Модификатор unsigned, записывыемый 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;
```

Для объявления массивов в языке С используется оператор [] с указанием размера массива. В фигурных скобках задаются значения элементов массива. Ключевое слово const используется для задания констант. Примеры:

```
char  A[3]= {10, 3, 7};
int  B[5];
const unsigned int E[2] = { 0, 1 };
```

Рассмотрим основные операторы языка С. Оператор присваивания имеет следующий синтаксис:

<идентификатор> = <выражение>;

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

Арифметические операции: + сложение, – вычитание, * умножение, / деление, % остаток от деления.

Логические операции: || логическое ИЛИ, && логическое И, ! логическое НЕ.

Операции с битами: | побитовая операция ИЛИ, & побитовая операция И; ^ побитовая операция исключающее ИЛИ, ~ побитовая операция НЕ;
<< битовый сдвиг влево, >> битовый сдвиг вправо.

Язык С относится к языкам программирования со строгой типизацией: 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(<инициализация >;  <условие >;  <модификация>)
        { <операторы тела цикла>; }
```

Рассмотрим работу цикла for: в блоке инициализации при первом запуске цикла присваивается начальное значение счетчику цикла; затем

анализируется условие (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( <условие> ) { <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 в языке С оформляются в виде функций. Описание функции имеет следующий синтаксис:

```
[<тип возвращаемого значения>]  <имя функции> ([<параметры>])
{
        [<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 С, 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, можно использовать ассемблерный код в программе, написанной на языке С. 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 С – программмы, 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 С. Директива #pragma warn - запрещает компилятору генерировать предупреждения о том, что функция не возвращает результат стандартным способом (с помощью оператора return).

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 {} .

2 НАСТРОЙКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Основной программой которая будет использоваться для работы с макетом является CodeVision – позволяет создавать и компилировать программы микроконтроллера на языке С, а также создает файлы с расширением .COFF и .OBJ необходимые для последующей отладки в среде AVR Studio.

2.1 Настройка проекта в среде CodeVision

После установки среды проектирования CodeVision в дирректорию C:\cvavr, необходимо произвести настройку среды для работы с макетом МЛ-1. Основной настройкой является выбор типа программатора, который будет работать совместно с отладочным комплексом МЛ-1. В меню Settings необходимо выбрать пункт Programmer и после того как появится диалоговое меню с различными типами аппаратных устройств для программирования, необходимо выбрать Kanda Systems STK200+/300. Это программатор работающий через LPT порт, поэтому в графе Printer Port необходимо выбрать номер LPT порта , к котрому подключен программатор отладочного компекса. На рис. 2.1. показан выбор типа программатора.

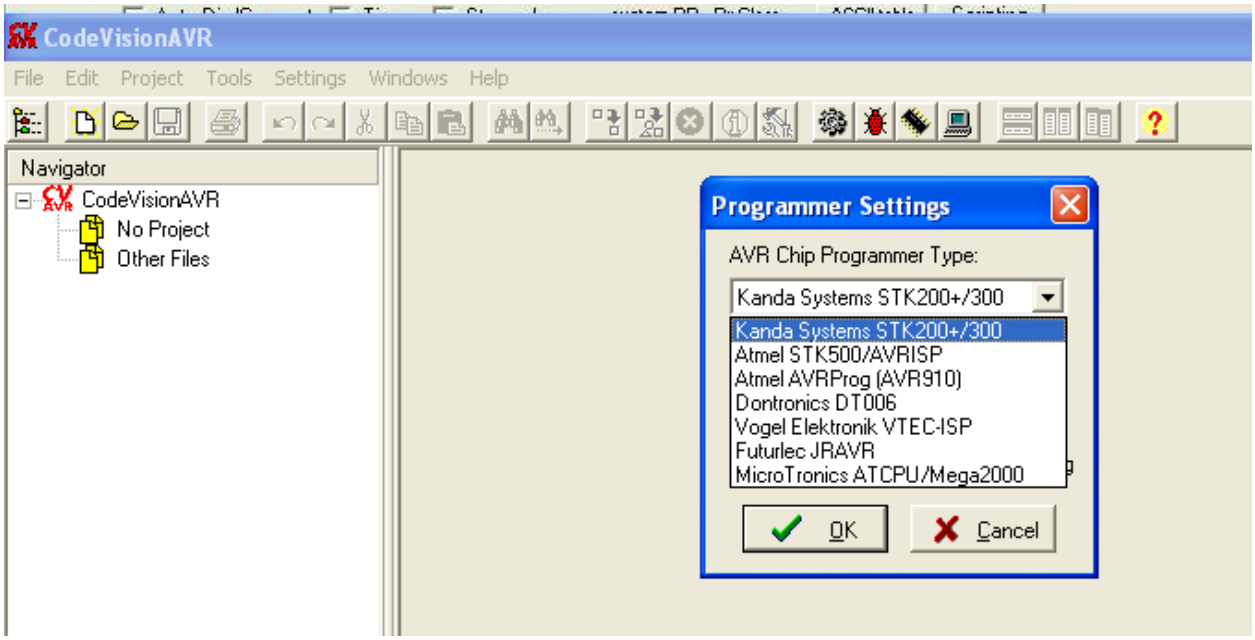


Рис. 2.1. - Выбор типа программатора

Далее нажимаем ОК и переходим к настройке отладчика AVRStudio, который будет вызываться из среды CodeVision. Для этого необходимо в меню Settings выбрать пункт Debugger и после того как появится диалоговое меню, необходимо выбрать Atmel AVRStudio4, и указать путь к этой

программе, по умолчанию путь следующий: C:\Program Files\Atmel\AVR Tools\AvrStudio4. На рис. 2.2 .показаны установки отладчика.

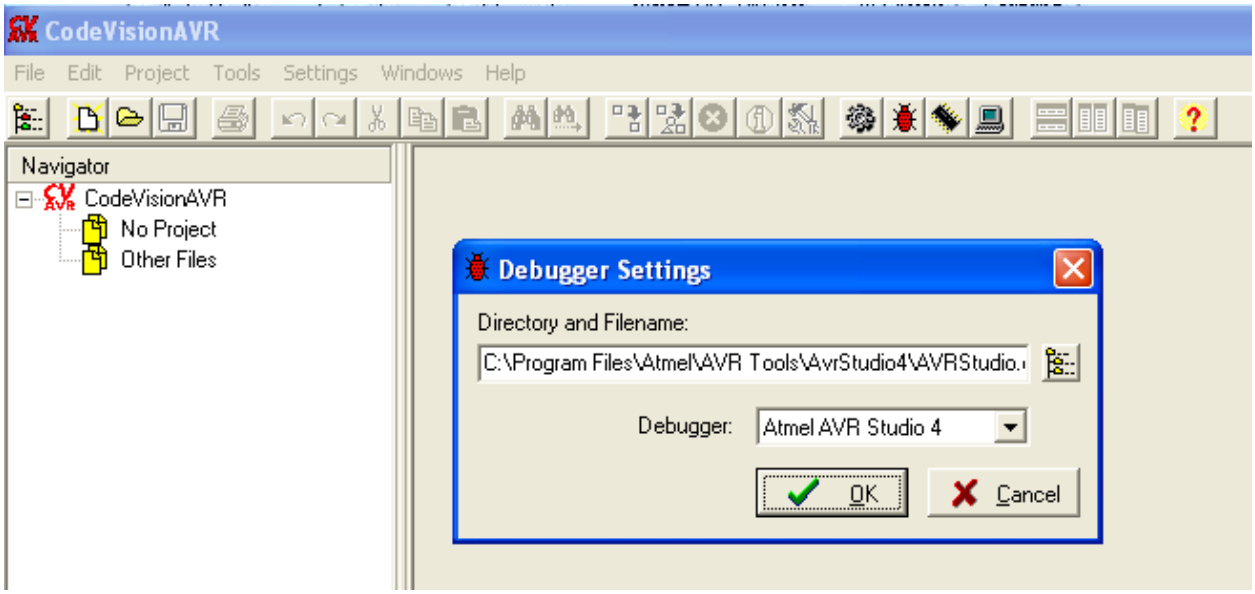


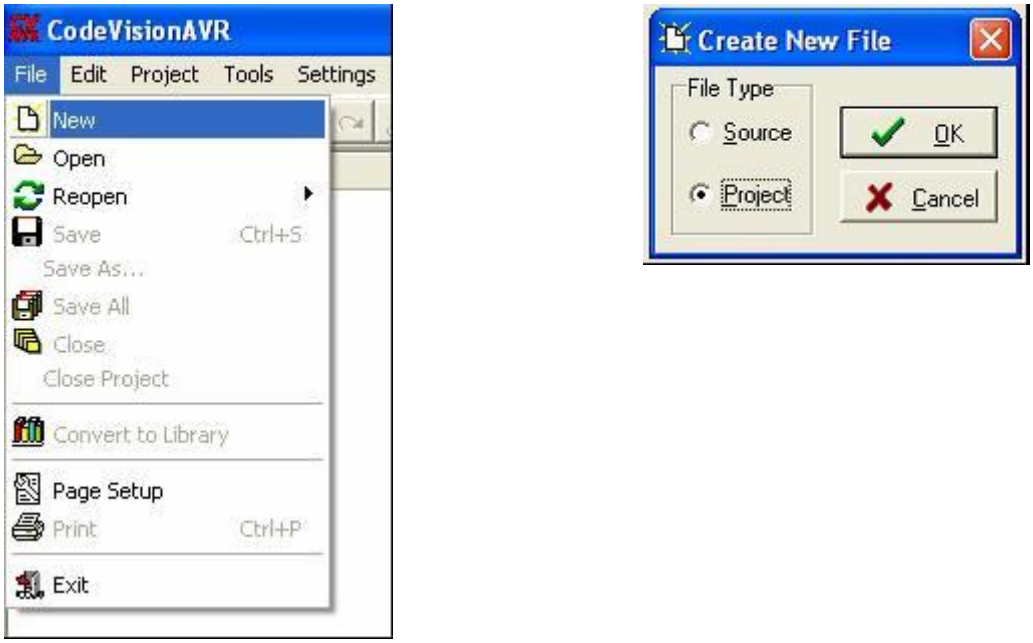
Рис. 2.2. - Установки запуска отладчика из среды CodeVision

На этом настройки аппаратной части программной среды CodeVision можно считать законченными.

Теперь напишем небольшой проект, откомпилируем его и прошьем микроконтроллер. Запускаем среду CodeVision и в меню File выбираем пункт New (рис. 2.3а) после чего появится диалоговое сообщение о том, что нам предлагается создать файл или проект (рис.2.3б). Выбираем создание проекта (Project) и нажимаем ОК, после чего появляется диалоговое сообщение, в котором среда проектирования предлагает нам воспользоваться CodeWizardAVR. Это средство является дополнительной функцией среды CodeVision, позволяющее при помощи графического меню создать основу проекта. Нажмем YES и воспользуемся визардом. В результате должно появиться диалоговое окно, показанное на рис. 2.4а.

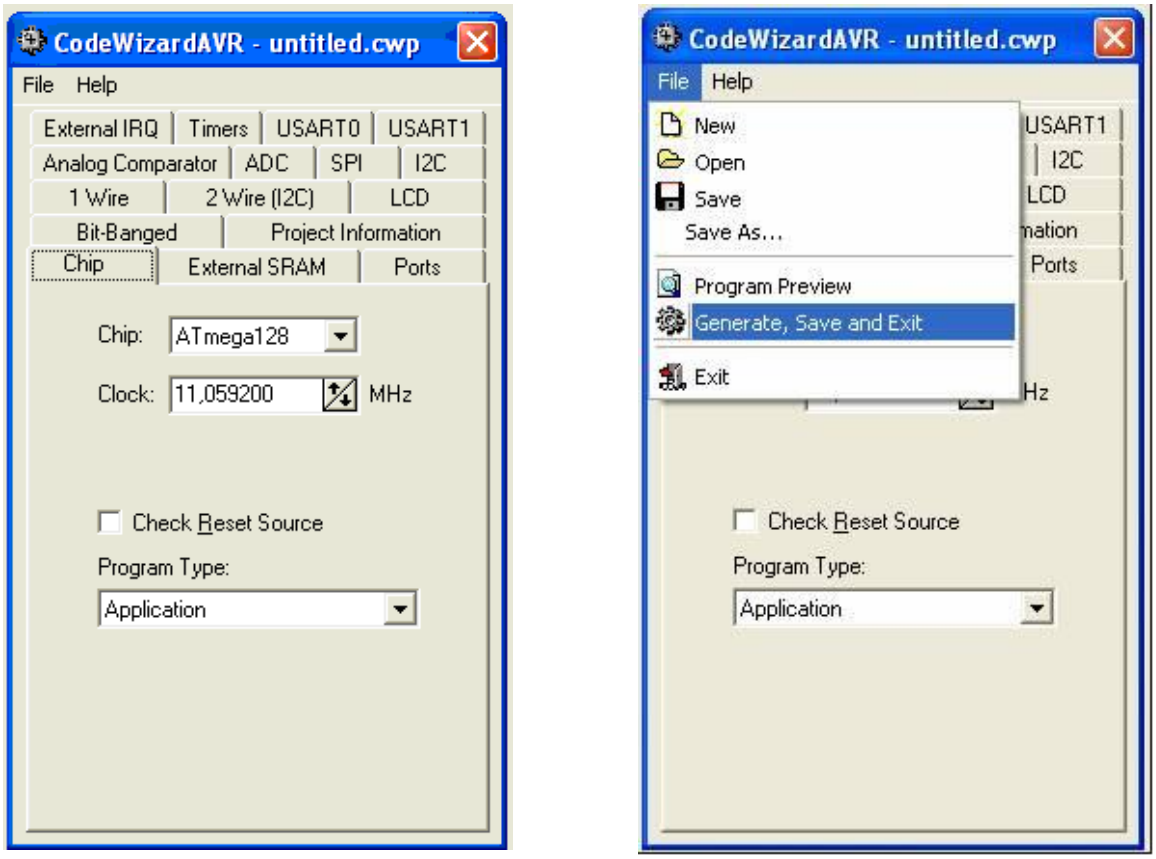
В выпадающем меню с надписью Chip: необходимо выбрать ATmega128, а в меню с надписью Clock: выставить частоту задающего кварцевого резонатора установленного на плате отладочного комплекса. В нашем случае частота равна 11,0592 MHZ. После этого в меню визарда выбираем пункт File, а в нем Generate, Save and Exit (рис. 2.4б).

После чего нужно сохранить проект и все его файлы и можно приступать непосредственно к написанию программы. Болванка создаваемая с помощью визарда содержит все регистры, которые могут и не использоваться в нашей программе, их нужно удалить.



(а) (б)

Рис. 2.3. – Создание нового проекта



(а) (б)

Рис. 2.4. – Диалоговое окно настроек проекта

На рис. 2.5. показано назначение пиктограмм в среде CodeVision.

Рис. 2.5. – Назначения пиктограмм, отвечающих за выполнение специализированных функций

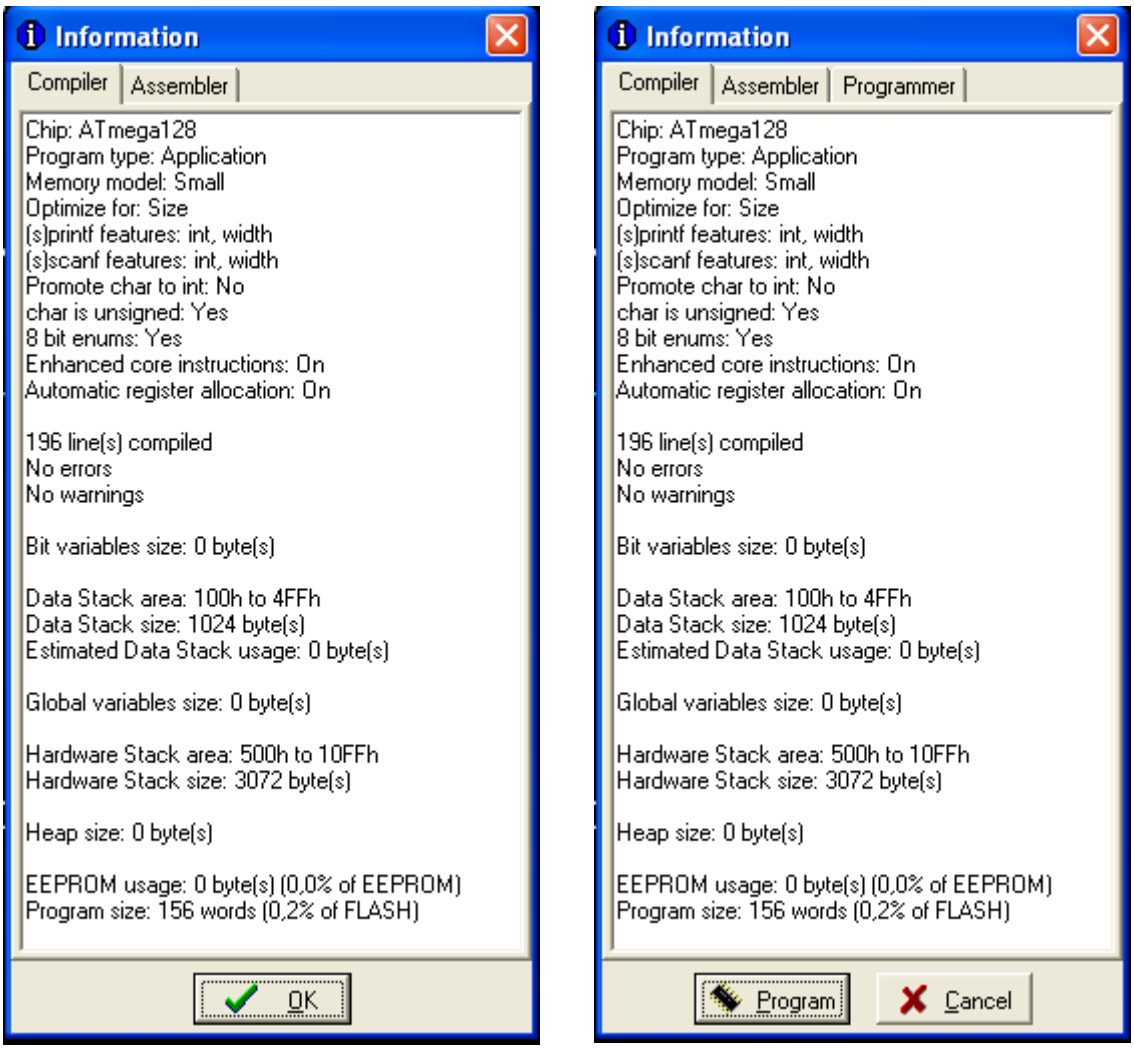
2.2 Программирование микроконтроллера

После того как код программы набран в среде, можно произвести компиляцию проекта, для этого необходимо нажать комбинацию клавиш Shist+F9, после чего CodeVision выдаст сообщение о том, что проект скомпилирован и укажет в процентах, какое количество памяти займет откомпилированный код при программировании микроконтроллера.

Отличительной особенностью программной среды CodeVision является наличие программатора, чтобы его задействовать в конкретном проекте, необходимо в опциях этого проекта указать, что после компиляции запускается программатор. Делается это следующим образом. В меню необходимо выбрать пункт Project -> Configure и здесь выбрать вкладку After Make. В этом меню необходимо установить флажок напротив надписи Program the Chip. При этом становятся доступными все опции программирования и аппаратной настройки микроконтроллера при программировании и сообщение после компиляции нашего проекта будет выглядеть так же как и в предыдущем случае за исключением того, что появится кнопка Program. С ее помощью можно запрограммировать микроконтроллер непосредственно из среды CodeVision. При этом будет произведена прошивка откомпилированного HEX файла совместно с установочными фьюзами и дополнительными опциями аппаратной инициализации, которые позволяют включать или отключать периферийные устройства микроконтроллера.

Также кроме файла прошивки с расширением .HEX компилятор создаст файлы с расширением .COFF и .OBJ, которые в дальнейшем используются при пошаговой отладке проекта в среде проектирования AVRStudio. Несмотря на то, что AVRStudio позволяет писать программы только на языке ассемблер, эта среда имеет возможность вести отладку как на ассемблере, так и на Си.

Если при программировании микроконтроллера не было выдано никаких сообщений и результат выполнения программы виден на светодиодной индикации, значит все подключения и настройки выполнены верно и комплекс готов к работе.



(а) (б)

Рис. 2.6. – Компиляция программы

Перед выполнением процедуры программирования микроконтроллера необходимо произвести настройку параметров интерфейса программатора (Programmer Settings) с помощью команды Programmer из подпункта главного меню Setting в соответствие с данными, приведенными на рис. 2.7.

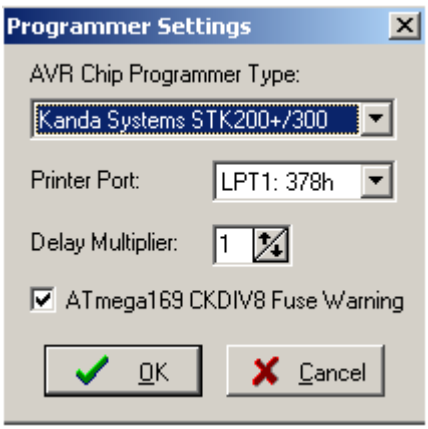


Рис. 2.7. – Окно настройки параметров интерфейса программатора

Параметры интерфейса программатора устанавливаются только один раз перед началом работы с микроконтроллером и при корректной работе устройства не требуют изменений.

В некоторых случаях для проверки работоспособности загруженной программы необходимо выполнить сброс микроконтроллера (Reset Chip) или удалить программу из памяти (Erase Chip). Эти функции (см. рис. 2.8) становятся доступными при выборе подпункта Настройки Программатора из пиктограммного меню (см. рис. 2.5).

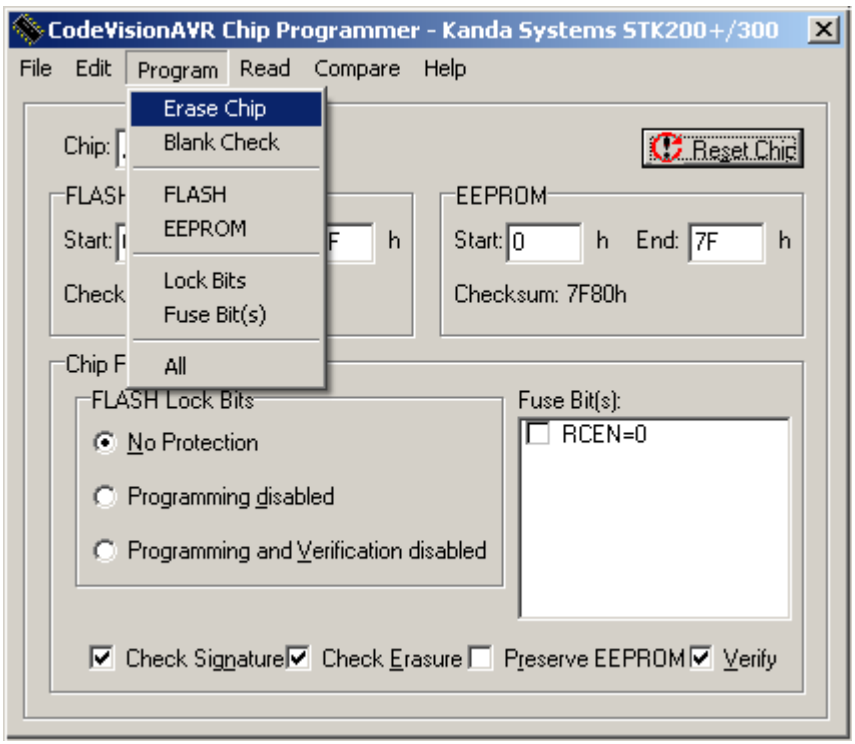


Рис. 2.8. – Окно отображения настроек программатора

3 УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

3.1 Ход лабораторной работы

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 программы в соответствии с вариантом задания.

6) Выполнить компиляцию (нажав клавишу F9) 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.

3.2 Описание блока светодиодов лабораторного макета

В состав макета входит блок индикации, состоящий из 8 светодиодов, подключенных к микроконтроллеру через порт D в соответствие со схемой, приведенной на рис. 3.1. На аноды светодиодов (HL1-HL8) подается напряжение + 5В, 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.

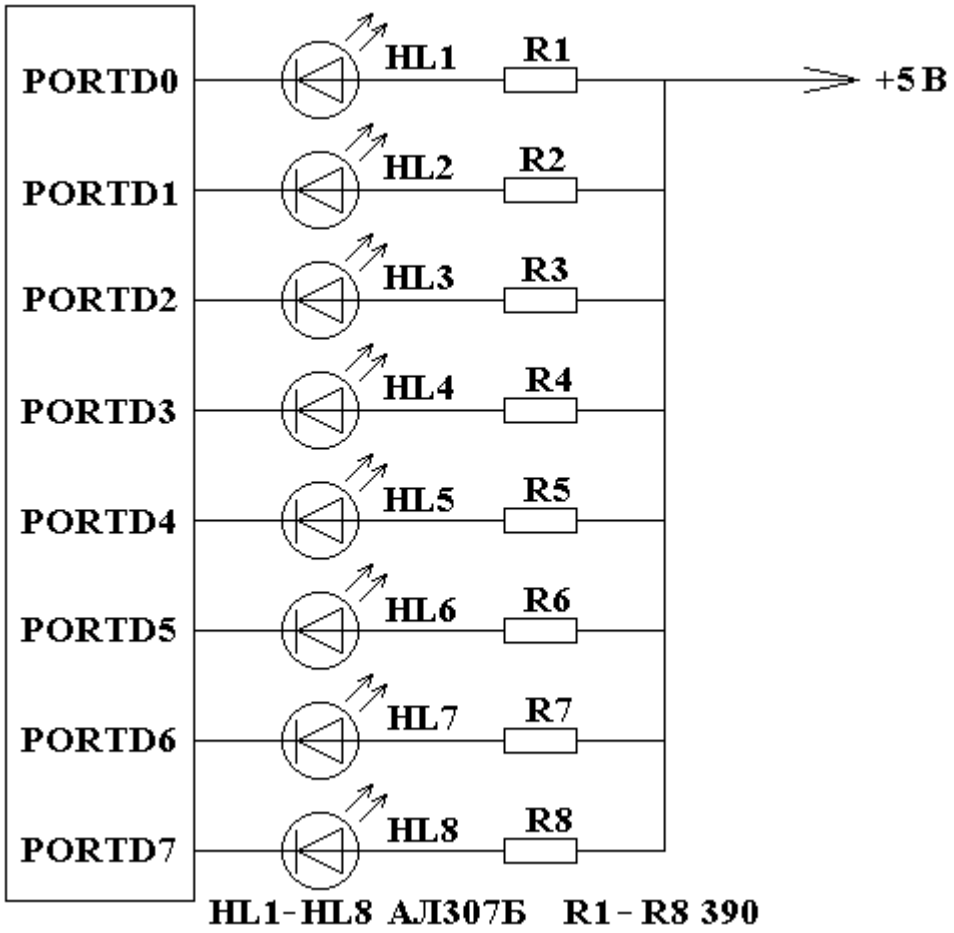


Рис. 3.1. – Принципиальная схема подключения блока светодиодов к выводам порта D

3.3 Принципы анализа нажатия стандартных кнопок с помощью микроконтроллера AVR MEGA 128

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 B, работающего в режиме ввода данных, и через резистор (сопротивлением порядка 10 кОм) с положительным полюсом источника электропитания (см. рис. 3.2). При разомкнутых контактах кнопки на входной линии микроконтроллера установится уровень “логической единицы”, при замкнутых – “логического нуля”.

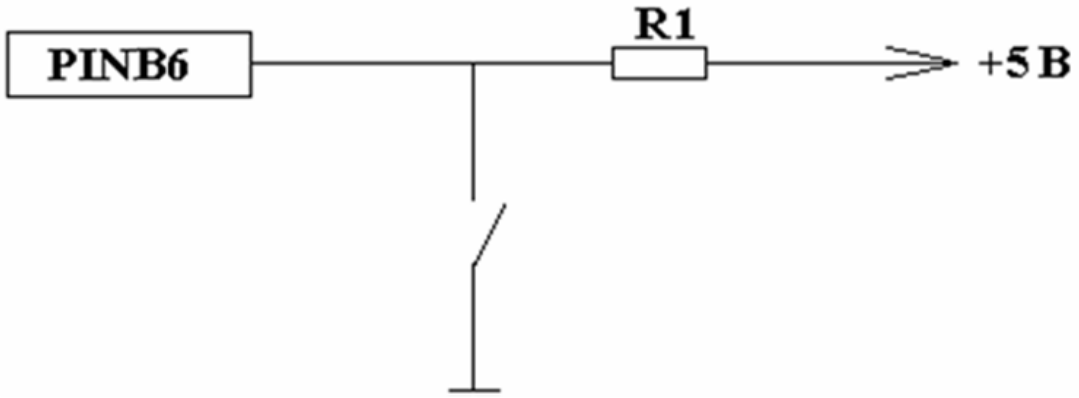


Рис. 3.2. – Принципиальная схема подключения кнопки к микроконтроллеру

Принцип проверки состояния нажатия клавиши заключается в периодическом программном опросе входной линии, к которой подключен один из выводов клавиши, и анализе значения соответствующего бита.

3.4 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 рис. 3.3. 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).

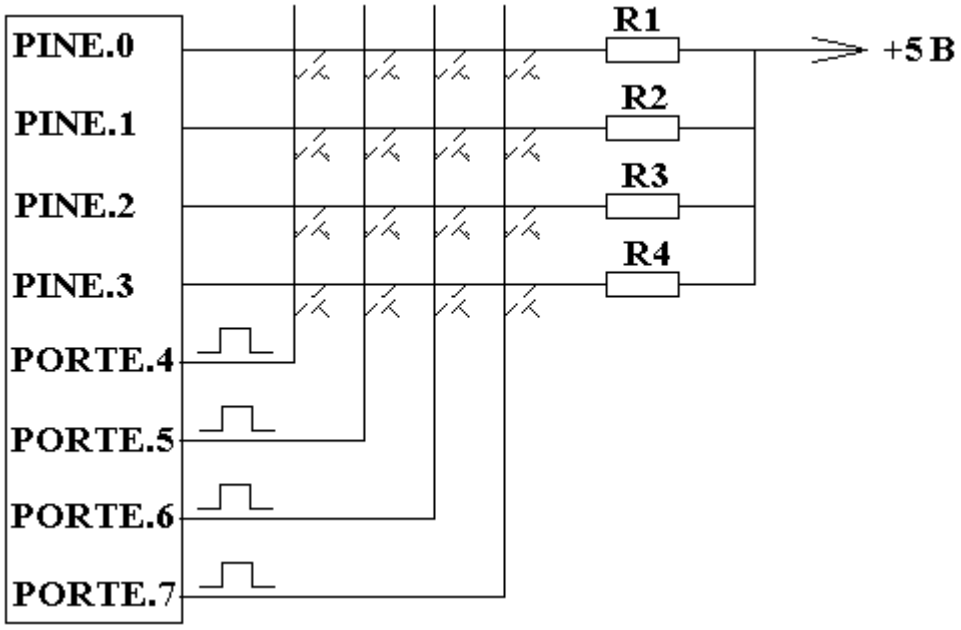


Рис. 3.3. – Принципиальная схема подключения матричной клавиатуры к микроконтроллеру

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 рис. 3.4) и считывании информации об уровне сигнала на горизонтальных линиях (PINE.0 – PINE.3).

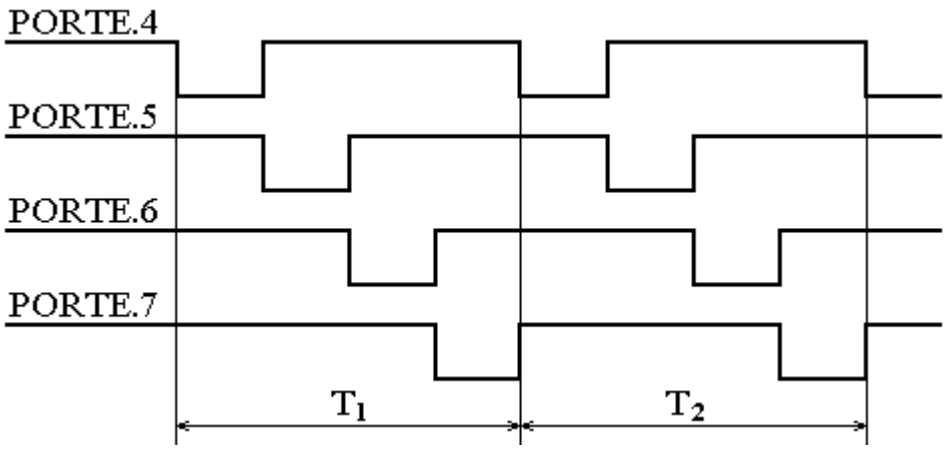


Рис. 3.4. – Временные диаграммы сигналов на выходных линиях (PORTE.4 – PORTE.7) порта E при опросе матричной клавиатуры

3.5 Библиотека для работы с ЖКИ лабораторного макета

Библиотека находится в файле lcd.c, который необходимо подключить директивой #include. ЖКИ позволяет отображать как буквенно-цифровую так и графическую информацию, для чего в состав контроллера ЖКИ входит модуль памяти. Для хранения кодов символов отображающихся на экране используется область памяти, начиная с адреса 0x0000, а для хранения графики с 0x0281.

Размер экрана в символах – 30x16, в пикселях - 240x128. Текст и графика могут отображаться совместно. Для начала работы с библиотекой необходимо инициализировать ЖКИ при помощи функции gl_init(). А затем очистить экран функцией gl_clear().

В библиотеке объявлены два дополнительных типа данных: t_byte и t_uint, для представления беззнаковых целых чисел (координат на экране).
Для вывода строки символов используется функция gl_string(t_byte x, t_byte y, const char* str), в которую передаются координаты первого выводимого символа по горизонтали и вертикали (x, y) и указатель на строку символов оканчивающуюся 0.

Для очистки только графического экрана используется функция gl_clear_graphics(), а для очистки текстового gl_clear_text().
Для вывода точки на экран используется функция gl_pixel_s(t_byte x, t_byte y, t_byte data), где x и y задают координаты, а data цвет пикселя: 0 – белый, 1 – черный.

Для изменения цвета точки на противоположный используется функция gl_pixel_xor_s(t_byte x, t_byte y), где x и y задают координаты точки.

Для получения цвета точки по координатам x и y используется функция gl_peek_pixel_s(t_byte x, t_byte y) которая возвращает цвет пикселя (0 или 1).

Для рисования линий используются функции gl_line(int sx, int sy, int ex, int ey, t_byte mask), gl_line_xor(int sx, int sy, int ex, int ey), где sx и sy задают координаты начальной точки, а ex и ey координаты конечной, mask (1 или 0) задает цвет линии.

3.6 Работа с функцией sprintf стандартной библиотеки ввода-вывода

Библиотека находится в файле stdio.h, который необходимо подключить директивой #include.

Формат вызова функции выглядит следующим образом:

sprintf(<буфер>, "< формат>", <параметр1>, <параметр2>, …, <параметрn>);

Для работы с функцией необходимо объявить буфер, в котором будет формироваться строка, например, так: char str_buf[32]. Буфер объявляется с некоторым запасом. Необходимо убедиться, что сформированная строка не будет содержать количество символов, превышающее размер буфера (с учетом последнего 0-го символа)!

Строка формата может содержать как символы, непосредственно выводящиеся в буфер, так и указатели на параметры, начинающиеся с символа % после которого идет описание формата вывода и типа параметра. Знак % можно вывести, записав последовательно %%.

Для вывода целого числа в 10-й системе исчисления используется параметр %i, для вывода в 16-й системе исчисления %x. Если после знака % идет символ 0, а затем цифры – число выводится с ведущими нулями таким образом, чтобы общее число цифр было не менее указанного перед форматом. Например, %02i – выведет целое число из 2-х или более цифр, если же цифр меньше 2 то перед числом будет записан 0.

Для вывода символа используется параметр %c, для вывода беззнакового целого числа %u.

Пример вывода часов, минут и секунд:

```
sprintf( str_buf, "%2u:%02u:%02u", hours, mins, secs );
```

3.7 Система прерываний микроконтроллера AVR ATMEGA 128

Прерывание – процесс, нарушающий выполнение нормального хода программы. Прерывания происходят при возникновении внутреннего или внешнего события микроконтроллера. При этом микроконтроллер сохраняет текущий счетчик команд и загружает в него адрес соответствующего вектора прерывания, в котором, содержится команда 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 Б.1 (в приложении Б). 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.

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

```
interrupt [<идентификатор прерывания>]
<тип возвращаемого значения> <имя функции> ( <аргументы>)
{   <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 актовой 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.

3.7.1 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. рис. 3.5), а в регистре TIFR формируются соответствующие флаги (см. рис. 3.6).

№ бита	7	6	5	4	3	2	1	0
37h/57h	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0

Бит 7 – OCIE2: 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 – OCIE1A: разрешение прерывания по совпадению «A» 16 разрядного таймера/счетчика T1;
Бит 3 – OCIE1B: разрешение прерывания по совпадению «B» 16 разрядного таймера/счетчика T1;
Бит 2 – TOIE1: разрешение прерывания по переполнению 16 разрядного таймера/счетчика T1;
Бит 1 – OCIE0: разрешение прерывания по совпадению 8 разрядного таймера/счетчика T0;
Бит 0 – TOIE0: разрешение прерывания по переполнению 8 разрядного таймера/счетчика T0;

Рис. 3.5. – Регистр маски прерываний от таймеров/счетчиков – TIMSK

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

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 «A» 16 разрядного таймера/счетчика T1;
Бит 3 – OCF1B: флаг совпадения «B» 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;

Рис. 3.6. – Регистр флагов прерываний от таймеров/счетчиков – TIFR

Рассмотрим подробно принципы функционирования 8-ми разрядного таймера/счетчика T0, структурная схема которого представлена на рис. B.1 (в приложении B). В состав таймера/счетчика T0 входят три 8-разрядных регистра ввода/вывода: счетный регистр TCNT0, регистр сравнения OCR0 и 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 3.1.

Таблица 3.1

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. рис. 3.7) 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 fp0 таймера/счетчика T0 (частота обновления значений в счетном регистре TCNT0, период – Tp0) 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. рис. 3.6). Разрешение и запрещение прерываний от таймера устанавливается в соответствующих битах регистра TIMSK (см. рис. 3.5).

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. рис. 3.6). Если в регистре TIMSK (см. рис. 3.5) 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) для обработки переполнения;
interrupt [TIM0_COMP] void timer0_compare (void) для обработки совпадения.

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

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.

Рис. 3.7. – Регистр управления таймером/счетчиком T0 в асинхронном режиме – ASSR

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

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 OCN0 изменяется в соответствие с установками разрядов COM00-01;
Биты 6,3 – WGM00, WGM01: определяют режим работы таймера счетчика T0 (см. таблицу 3.2);
Биты 5,4 – COM01, COM00: определяют поведение вывода OC0 при появлении события «совпадение» (см. таблицу 3.3) ;
Биты 2,1,0 – CS02, CS01, CS00: определяют частоту тактового сигнала таймера/счетчика T0 (см. таблицу 3.4).

Рис. 3.8 – Регистр управления таймером/счетчиком T0 - TCCR0

Таблица 3.2

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 3.3

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 3.4

Значения коэффициентов для предварительного делителя частоты таймера/счетчика T0

CS02	CS01	CS00	Описание	f , КГц	T , мкс
0	0	0	Таймер/счетчик остановлен	0,0	0,00
0	0	1	CLK	11000,0	0,09
0	1	0	CLK/8	1375,0	0,72
0	1	1	CLK/32	344,0	2,88
1	0	0	CLK/64	172,0	5,76
1	0	1	CLK/128	86,0	11,52
1	1	0	CLK/256	43,0	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 3.5.

Таблица 3.5

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.

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. 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 инкрементируется или декрементируется по фронту тактового сигнала таймера/счетчика T1.

Если установлен флаг TOIE1 в регистре TIMSK (см. рис. 3.5), то при переполнении TCNT1 генерируется прерывание и устанавливается флаг TOV1 в регистре TIFR (см. рис. 3.6). 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, OCF1A, OCF1A 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 таймера/счетчика T1 (частота обновления значений в счетном регистре TCNT1, период – T) определяется значениями битов CS00-CS03 регистра TCCR1B (см. рис. 3.9), с помощью которого устанавливаются необходимые параметры работы таймера (см. таблицу 3.6). 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

Описание управляющих битов регистра TCCR1B сходно с описанием битов регистра TCCR0 (см. рис. B.5).

Рис. 3.9. – Регистр управления таймером/счетчиком T1 – TCCR1B

Пример программы на языке С для 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 предварительного делителя
TIMSK = 1;                 установка бита TOIE0 для разрешения прерывания по переполнению таймера-счетчика T0
#asm("sei");               разрешить прерывания микропроцессора
```

Таблица 3.6

Значения коэффициентов для предварительного делителя частоты таймера/счетчика T1

CS02	CS01	CS00	Описание	f , КГц	T , мкс
0	0	0	Таймер/счетчик остановлен	0,0	0,00
0	0	1	CLK	11000,0	0,09
0	1	0	CLK/8	1375,0	0,72
0	1	1	CLK/64	172,0	5,76
1	0	0	CLK/256	43,0	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	_	_

3.8 Работа со стандартной математической библиотекой

Библиотека находится в файле math.h, который необходимо подключить директивой #include. В библиотеке находятся основные математические функции, основные из которых приведены в таблице 3.1.

Таблица 3.1

Математические функции стандартной библиотеки языка С

Функция	Математическая запись	Назначение
fabs(x)	$\|x\|$	модуль x
fmax(x, y)	$\max(x, y)$	максимум из x и y
fmin(x, y)	$\min(x, y)$	минимум из x и y
fsign(x)	$\mathrm{sgn}(x)$	знак числа x = -1 или 1
sqrt(x)	$\sqrt{x}$	квадратный корень из x
exp(x)	e^x	e в степени x
log(x)	$\log_2(x)$	двоичный логарифм x
log10(x)	$\log_{10}(x)$	десятичный логарифм x
pow(x, y)	x^y	x в степени y
sin(x)	$\sin(x)$	синус x
cos(x)	$\cos(x)$	косинус x
tan(x)	$\mathrm{tg}(x)$	тангенс x
asin(x)	$\arcsin(x)$	арксинус x
acos(x)	$\arccos(x)$	арккосинус x
atan(x)	$\mathrm{arctg}(x)$	арктангенс x

Все функции (кроме функции fsign) возвращают значение типа float. Значения аргументов x и y должны иметь тип float. Для приведения значений типа float к типу int необходимо использовать операцию приведения: <значение типа int> = (int) <значение типа float>.

Для задания констант типа float используется точка которая разделяет целую и дробную части. Например: 3.14 или 345.246. В библиотеке уже объявлена константа π, которая называется PI.

Пример программы вычисляющей квадратный корень числа и сохраняющей целую часть вычисленного значения в переменной типа int:

```
float y, x = 1234.0;    переменные хранящие значения с плавающей запятой
int iy;                 переменная в которую будет помещена целая часть
                        вычисленного квадратного корня
y = sqrt( x );          вычисляем квадратный корень из x
iy = (int) y;           записываем целую часть в переменную типа int
```

3.9 Последовательный интерфейс RS-232C

Интерфейс RS-232C предназначен для соединения двух устройств (см. рис. 3.10), 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. рис. 3.11). 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%).

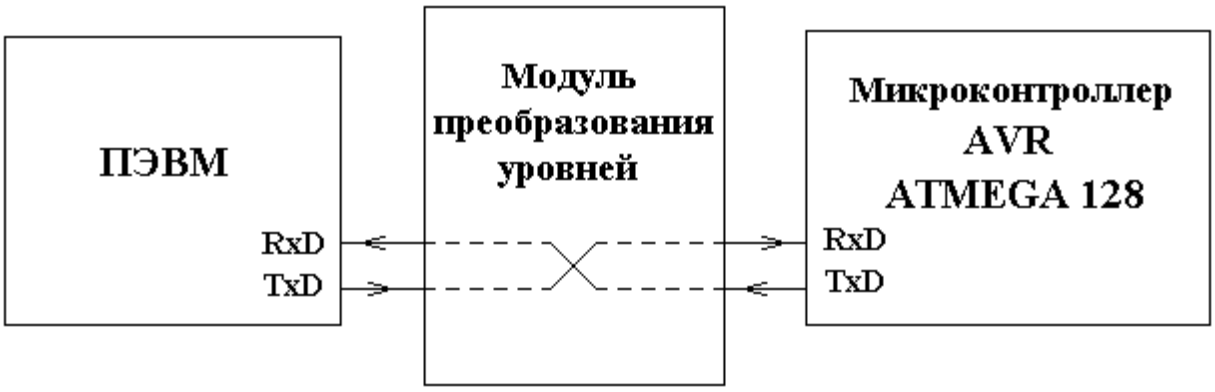


Рисунок 3.10. – Обобщенная функционально-структурная схема соединения двух устройств с помощью интерфейса RS-232C

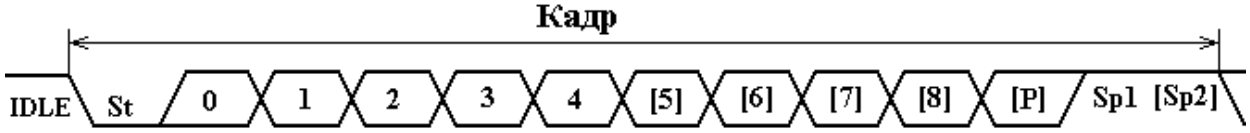


Рис. 3.11. – Формат кадра при обмене данными по интерфейсу RS-232C

3.9.1 Организация модулей USART в микроконтроллере AVR ATMEGA128. Все микроконтроллеры AVR семейства MEGA имеют в своем составе модули универсального синхронно/асинхронного приемо-передатчика 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. рис. 3.12), UCSR1B (см. рис. 3.13), UCSR1C (см. рис. 3.14).

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

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.

Рис. 3.12. – Регистр управления/статуса UCSR1A

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

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 3.2).
Бит 1 – RXB81 – формат принимаемых данных 9 бит.
Бит 0 – TXB81 – формат передаваемых данных 9 бит.

Рисунок 3.13. – Регистр управления/статуса UCSR1B

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

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 3.3).
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 3.2);
Бит 0 – UCPOL1 – в асинхронном режиме должен быть равен 0.

Рисунок 3.14. – Регистр управления/статуса UCSR1C

Таблица 3.2

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

Таблица 3.3

Установки режима четности

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

Скорость передачи данных VBAUD (в бодах, бит/с) определяется из выражения 3.1 и задается путем записи 12-разрядного значения в регистры UBRR1L 98h(B8h) и UBRR1H 99h(B9h). В регистре UBRR1H используются только младшие 4 разряда.

$$v_{baud} = \frac{f_{clk}}{16 \cdot (UBRR_{H:L} + 1)}. \quad (3.1)$$

Значение, записываемое в регистры UBRR1H:UBRR1L, будет соответственно определяться по формуле 3.2:

$$UBRR_{H:L} = \frac{f_{clk}}{16 \cdot v_{baud}} - 1. \quad (3.2)$$

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 3.4. При этом необходимо учитывать значение – бита U2X1 управления скоростью передачи (0 – стандартная, 1 – удвоенная скорость), расположенного в первом разряде регистра UCSR1A.

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 С 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; UBRR1L=47; установка значения делителя (47) соответствующего скорости приема данных 14400 бит/с.

Таблица 3.4

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

4 ЗАДАНИЯ НА ЛАБОРАТОРНЫЕ РАБОТЫ

4.1 Лабораторная работа №1 – Создание простейшей программы на языке С

При подготовке к лабораторной работе необходимо составить предварительный вариант текста программы на языке С и блок-схему алгоритма, в соответствии с индивидуальным заданием (см. таблицу 4.1).

Цель. Изучить принципы функционирования лабораторного макета, разработать алгоритм и программу на языке С.

Задание. Разработать в среде программирования Code Vision AVR программу на языке С для микроконтроллера AVR ATMEGA 128 согласно заданному варианту. Порядок выполнения задания показан в разделе 3.1. Указания к лабораторной работе показаны в разделе 3.2.

Настройка порта D на вывод производится при помощи записи в "регистр" DDRD единицы в соответствующем разряде. Вывод данных в порт производится при помощи записи байта данных в "регистр" PORTD при помощи оператора присваивания. Для задания задержки используется процедура delay_ms(), в которую передается величина задержки в миллисекундах. Для использования этой функции необходимо подключить библиотеку delay.h:

```
#include <delay.h>
```

В отчете необходимо привести следующее:

- характеристики лабораторной вычислительной системы;
- исходный код разработанной программы;
- краткие выводы по работе, в которых необходимо отразить особенности программного управления светодиодом с помощью микроконтроллера ATMEGA 128.

Контрольные вопросы и задания:

1) Поясните основные особенности архитектуры микроконтроллера ATMEGA 128.
2) Поясните принципы распределения адресных пространств памяти, регистров общего назначения и портов ввода/вывода в микроконтроллере ATMEGA 128.
3) Принцип управления портами ввода-вывода из программы на языке С.
4) Перечислите основные типы данных в языке С для ATMEGA 128.
5) Оператор присваивания в языке С.
6) Структура программы на языке С.
7) Назначение и подключение библиотек на языке С.

Таблица 4.1

Задания на лабораторную работу №1

Вариант	Задание: разработать программу, выполняющую…
1	… включить 2-й светодиод на 8 секунд
2	… включить 8-й светодиод на 3 секунды, а затем 3-й светодиод на 7 секунд
3	… включить 5-й светодиод на 12 секунд, а затем 6-й светодиод на 5 секунд
4	… включить 7-й светодиод на 18 секунд
5	… включить 3-й светодиод на 15 секунд, а затем 8-й светодиод на 3 секунды
6	… включить 8-й светодиод на 3 секунды
7	… включить 1-й светодиод на 3000 миллисекунд
8	… включить 4-й светодиод на 7 секунд, а затем 6-й светодиод на 8 секунд

Пример задания. Разработать программу, включающую 1-й светодиод на 10 секунд.

Решение. В лабораторном макете блок, состоящий из 8-ми светодиодов, подключен к порту D микроконтроллера в соответствии с принципиальной схемой, приведенной на рис. 3.1. 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 рис. 4.1.

Полный текст программы приводится ниже:

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

main()
{
        DDRD=0b11111111;
        PORTD=0b11111110;

        delay_ms(10000);

        PORTD=0b11111111;
}
```

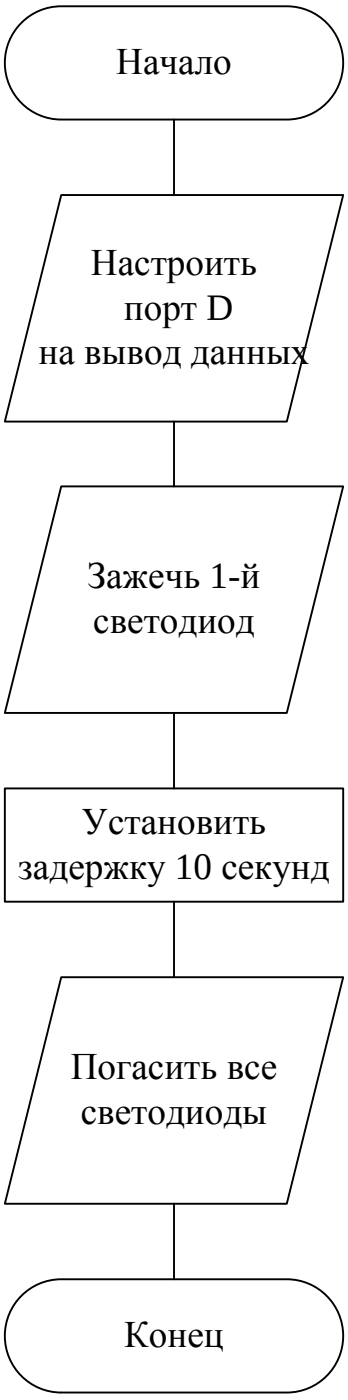


Рис. 4.1. – Алгоритм управления светодиодом

4.2 Лабораторная работа №2 – Циклическое управление группой светодиодов

При подготовке к лабораторной работе необходимо составить предварительный вариант текста программы на языке С и блок-схему алгоритма, в соответствии с индивидуальным заданием (см. таблицу 4.2).

Цель. Изучить принципы функционирования и подключения к микроконтроллеру блока светодиодов, разработать алгоритм и программу для реализации циклического переключения состояния светодиодов.

Задание. Разработать в среде программирования Code Vision AVR программу на языке С для микроконтроллера ATMEGA 128, управляющую блоком из восьми светодиодов. Порядок выполнения задания показан в разделе 3.1. Указания к лабораторной работе показаны в разделе 3.2.

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

В отчете необходимо привести следующее:

- характеристики лабораторной вычислительной системы;
- исходный код разработанной программы;
- анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности программного управления блоком светодиодов с помощью микроконтроллера AVR ATMEGA 128.

Контрольные вопросы и задания:

1) Поясните реализацию в микроконтроллере ATMEGA 128 Гарвардской архитектуры и принципа конвейерной обработки команд.

2) Циклы в языке С.

3) Задание констант в различных системах счисления.

4) Принцип работы цикла while.

5) Принцип работы цикла for.

6) Поясните принципы распределения адресных пространств памяти, регистров общего назначения и портов ввода/вывода в микроконтроллере ATMEGA 128.

7) Принцип управления портами ввода-вывода из программы на языке С.

Таблица 4.2

Задания на лабораторную работу №2

Вариант	Задание: разработать программу, выполняющую…
1	… в бесконечном цикле параллельное включение и выключение 1–го, 2–го, 3–го и 8–го светодиодов с длительностью свечения 1,5 с и временем нахождения в погашенном состоянии 2 с.
2	… параллельное включение и выключение 1–го, 4–го и 7–го светодиодов с длительностью свечения 0,5 с и временем нахождения в погашенном состоянии 1,5 с, в течении ~30 секунд.
3	… в бесконечном цикле параллельное включение и выключение в противофазе четных и нечетных светодиодов с длительностью свечения 0,5 с и временем нахождения в погашенном состоянии 3 с.
4	… параллельное включение и выключение 1–го, 2–го, 3–го и 5–го светодиодов с длительностью свечения 2 с и временем нахождения в погашенном состоянии 5 с, в течении ~30 секунд.
5	… параллельное включение и выключение 5–го, 6–го, и 8–го светодиодов с длительностью свечения 5 с и временем нахождения в погашенном состоянии 3 с. Переключение производится 15 раз.
6	… в бесконечном цикле параллельное включение и выключение в противофазе 1-4-го и 5-8-го светодиодов с длительностью свечения 1 с и временем нахождения в погашенном состоянии 1 с.
7	… параллельное включение и выключение в противофазе нечетных и четных светодиодов с длительностью свечения 1 с и временем нахождения в погашенном состоянии 1 с. Переключение производится 20 раз.
8	… параллельное включение и выключение в противофазе 1-4-го и 5-8-го светодиодов с длительностью свечения 5 с и временем нахождения в погашенном состоянии 2 с, в течении ~45 секунд.

Пример 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 с принципиальной схемой, приведенной на рис. 3.1. 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 рис. 4.2. Для обеспечения непрерывной работы программы используется бесконечный цикл, для чего в операторе while записывается константа всегда принимающая значение "истина".

Полный текст программы приводится ниже:

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

main()
{

      DDRD=0xff;

       while (1)
      {

            PORTD=0xff;
            delay_ms(1000);

            PORTD=0x55;
            delay_ms(2000);

       }

}
```

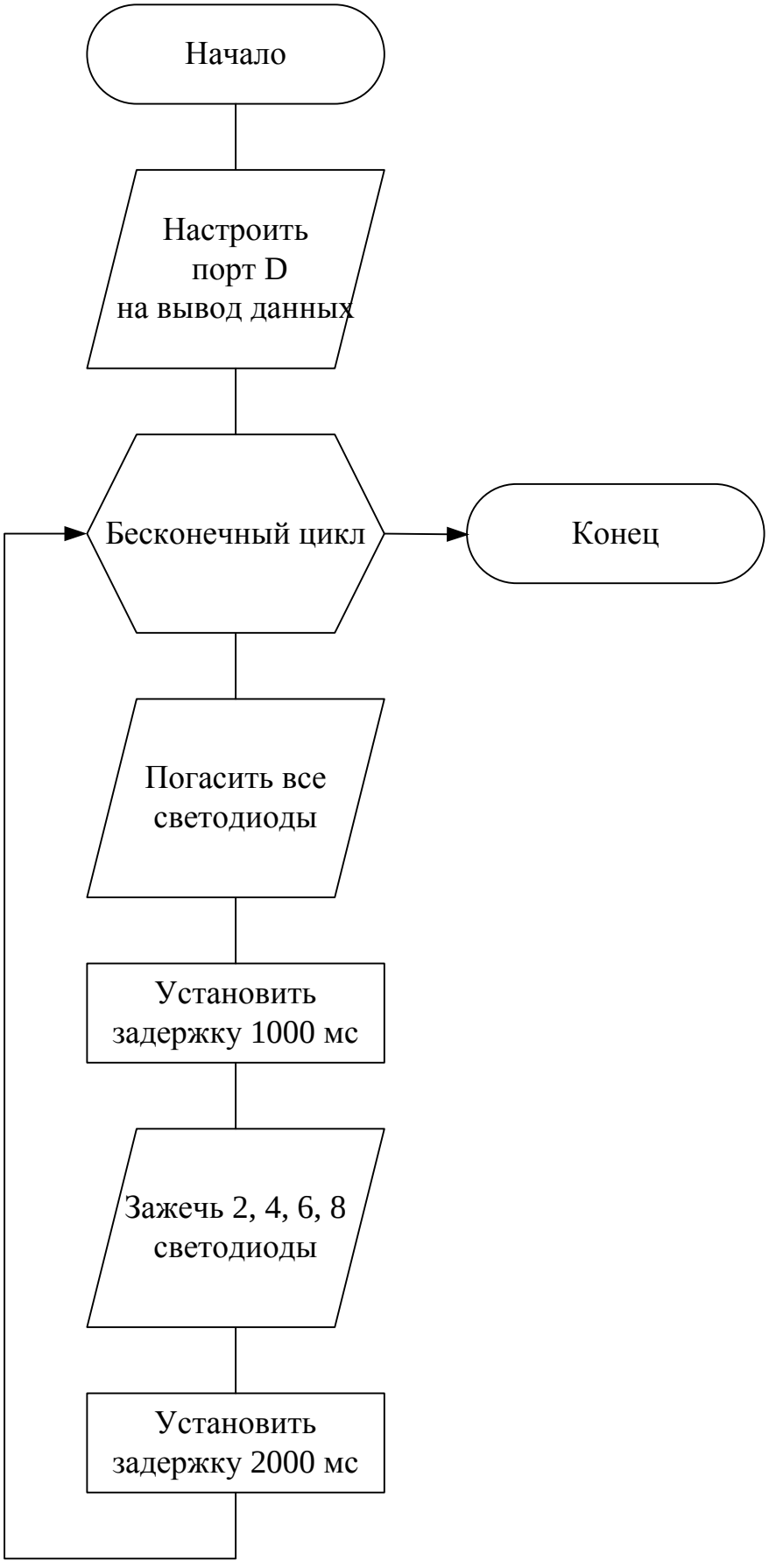


Рис. 4.2. – Алгоритм циклического управления светодиодами

4.3 Лабораторная работа №3 – Обработка нажатий на клавиши

При подготовке к лабораторной работе необходимо составить предварительный вариант текста программы на языке С и блок-схему алгоритма, в соответствии с индивидуальным заданием (см. таблицу 4.3).

Цель. Изучить принципы функционирования и подключения к микроконтроллеру клавиатуры лабораторного макета, разработать алгоритм и программу для реализации режима опроса клавиатуры.

Задание. Разработать в среде программирования Code Vision AVR программу на языке С для микроконтроллера ATMEGA 128, обрабатывающую нажатия на клавиши.

Для опроса трехкнопочной клавиатуры используются линии порта B начиная с 6-й и заканчивая 8-й. Настройка порта B на вывод производится при помощи записи в "регистр" DDRB нулей в соответствующих разрядах. Порядок выполнения задания показан в разделе 3.1. Указания к лабораторной работе показаны в разделе 3.3.

В отчете необходимо привести следующее:

- характеристики лабораторной вычислительной системы;
- исходный код разработанной программы;
- анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности программного опроса клавиатуры с помощью микроконтроллера AVR ATMEGA 128.

Контрольные вопросы и задания:

1) Циклы в языке С.
2) Объявление и инициализация переменных.
3) Битовые операции в языке С.
4) Типы данных в языке С.
5) Принцип управления портами ввода-вывода из программы на языке С.
6) Операции битового сдвига.
7) Условный оператор.
8) Логические операции.
9) Принцип подключения и работы трехкнопочной клавиатуры.

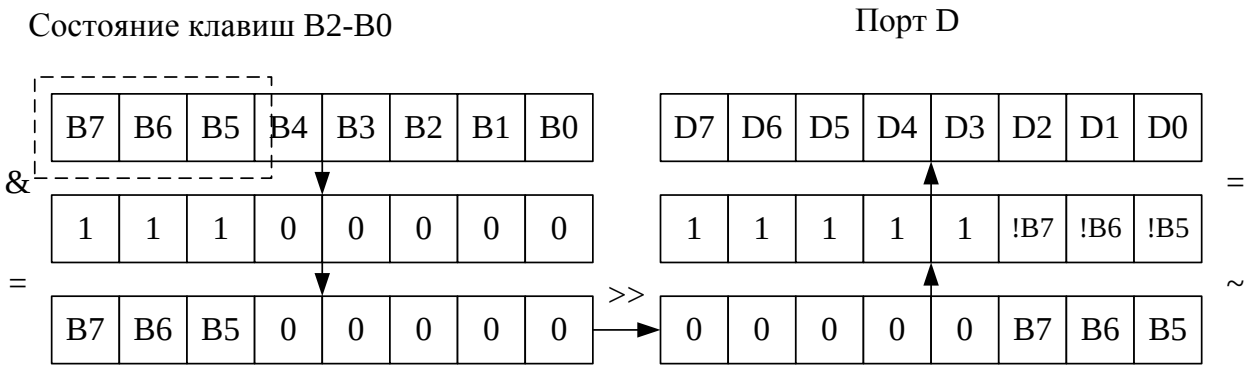
Таблица 4.3

Задания на лабораторную работу №3

Вариант	Задание: разработать программу, выполняющую…
1	… в бесконечном цикле индицирующую состояние нажатия клавиш на трехкнопочной клавиатуре, начиная с 1-го светодиода.
2	… в бесконечном цикле индицирующую состояние нажатия клавиш на трехкнопочной клавиатуре, начиная с 4-го светодиода. При одновременном нажатии трех клавиш программа должна завершаться с включением всех нечетных светодиодов.
3	… в бесконечном цикле индицирующую состояние нажатия клавиш на трехкнопочной клавиатуре, начиная с 4-го светодиода.
4	… в бесконечном цикле индицирующую состояние нажатия клавиш на трехкнопочной клавиатуре. При нажатии на клавишу №1, должны загораться 1, 2, 3 светодиоды; на клавишу №2 – 4, 5 светодиоды; на клавишу №3 – 7, 8 светодиоды.
5	… в бесконечном цикле индицирующую состояние нажатия клавиш на трехкнопочной клавиатуре, начиная со 2-го светодиода, при нажатии на клавишу №3 должен происходить выход из программы.
6	… в бесконечном цикле индицирующую состояние отжатия клавиш на трехкнопочной клавиатуре, начиная с 3-го светодиода.
7	… в бесконечном цикле индицирующую состояние нажатия клавиш на трехкнопочной клавиатуре. При нажатии на клавишу №1, должны загораться 1 и 8 светодиоды; на клавишу №3 – 2 и 7 светодиоды. При нажатии на клавишу №2 должен происходит выход из программы с включением всех светодиодов.
8	… в бесконечном цикле индицирующую состояние отжатия клавиш на трехкнопочной клавиатуре, начиная со 5-го светодиода, при нажатии на клавишу №1 должен происходить выход из программы с включением всех светодиодов.

Пример задания. Разработать программу, циклически опрашивающую клавиши трехкнопочной клавиатуры и индицирующую их состояние на светодиодном индикаторе.

Решение. В лабораторном макете блок из 3-х клавиш подключен к порту B микроконтроллера (начиная с 5-го бита), в соответствии с принципиальной схемой, приведенной на рис. 3.2. Опрос клавиатуры можно провести, считывая линии PINB с 6-й по 8-ю, согласно алгоритму, приведенному на рис. 4.3. Полученные биты состояния клавиатуры ("0" – клавиша нажата, "1" – клавиша отжата) сохраняются в переменной k, маскируются операцией побитового логического умножения (обнуляются младшие 5-ть бит). После чего сдвигаются в младшие биты и выводятся на индикацию в порт D, как показано на рисунке:



Полный текст программы приводится ниже:

```
#include <mega128.h>

main()
{
        char k;
        DDRD = 0xff;
        DDRB = 0;

        while (1)
        {
                k = PINB;
                k = ( k & 0xE0 ) >> 5;

                PORTD = ~k;
        }
}
```

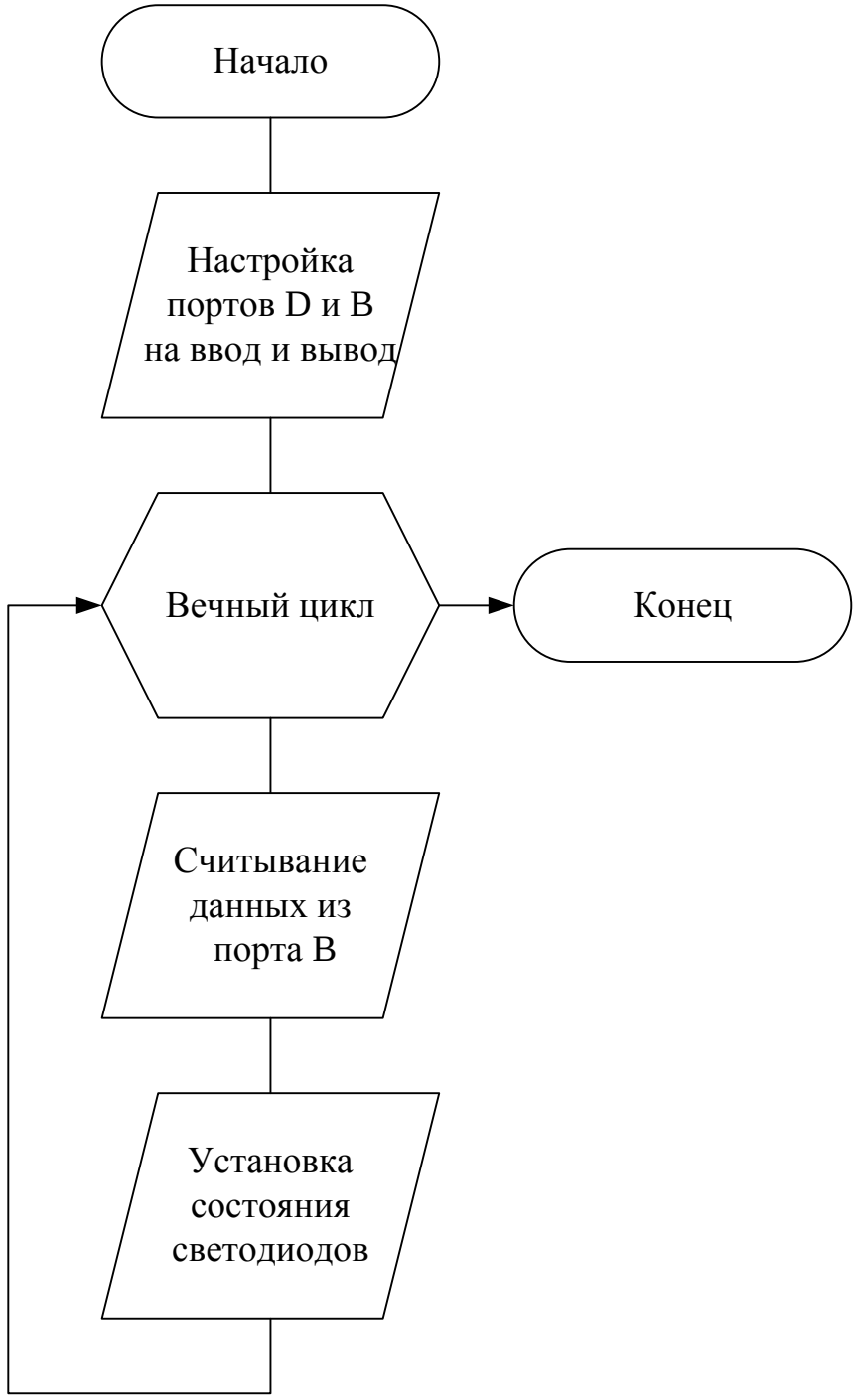


Рис. 4.3. – Алгоритм циклического опроса клавиатуры

4.4 Лабораторная работа №4 – Работа с матричной клавиатурой

При подготовке к лабораторной работе необходимо составить предварительный вариант текста программы на языке С и блок-схему алгоритма, в соответствии с индивидуальным заданием (см. таблицу 4.4).

Цель. 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.

Задание. Разработать в среде программирования Code Vision AVR программу на языке С для микроконтроллера ATMEGA 128, обрабатывающую нажатия на клавиши матричной клавиатуры.

Для опроса матричной клавиатуры используются линии порта E. Настройка порта E на вывод/вывод производится при помощи записи в "регистр" DDRE единиц/нулей в соответствующих разрядах соответственно. Порядок выполнения задания показан в разделе 3.1. Указания к лабораторной работе показаны в разделе 3.4.

В отчете необходимо привести следующее:

- характеристики лабораторной вычислительной системы;
- исходный код разработанной программы;
- анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности программного опроса матричной клавиатуры с помощью микроконтроллера AVR ATMEGA 128.

Контрольные вопросы и задания:

1) Цикл for.
2) Объявление массивов и констант в языке С.
3) Принцип опроса матричной клавиатуры.
4) Принцип управления портами ввода-вывода из программы на языке С.
5) Ассемблерные вставки.
6) Процедуры и функции в языке С.
7) Принцип устройства матричной клавиатуры.

Таблица 4.4

Задания на лабораторную работу №4

Вариант	Задание: разработать программу, фиксирующую …
1	… нажатия клавиш 1, 6 и 12 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши B0.
2	… нажатия клавиш 4, 7 и 10 матричной клавиатуры включением светодиодов 8, 5 и 3 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши B0.
3	… нажатия клавиш 4, 8 и 11 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши B2.
4	… нажатия клавиш 9, 10 и 11 матричной клавиатуры включением светодиодов 6, 7 и 8 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши B0.
5	… нажатия клавиш 1, 2 и 3 матричной клавиатуры включением светодиодов 4, 5 и 6 соответственно. Выход из цикла опроса осуществляется при одновременном нажатии клавиш B0+B2.
6	… нажатия клавиш 3, 7 и 11 матричной клавиатуры включением светодиодов 5, 6 и 7 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши B1.
7	… нажатия клавиш 5, 6 и 7 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при одновременном нажатии клавиш B1+B2.
8	… нажатия клавиш 1, 6 и 8 матричной клавиатуры включением светодиодов 8, 7 и 5 соответственно. Выход из цикла опроса осуществляется при нажатии трех клавиш трехкнопочной клавиатуры одновременно.

Пример задания. Разработать программу, выводящую информацию об индексах нажатых клавиш первого и второго столбцов (С1 и С2) матричной клавиатуры 3×4 (зажигается светодиод, соответствующий номеру нажатой клавиши).

Решение. В лабораторном макете матричная клавиатура 3×4 подключена к порту Е микроконтроллера. Линии старшей тетрады порта Е настраиваются на вывод данных, а линии младшей тетрады – на ввод. Цикл опроса состоит в последовательном считывании данных от 1-го до 3-го столбцов матричной клавиатуры 3×4 при соответствующих управляющих сигналах, вывода информации о позиции нажатой клавиши на блок светодиодов. Алгоритм программы приведен на рис. 4.4.

В программе используется два цикла, один бесконечный с использованием оператора while, который обеспечивает непрерывное выполнение программы, а второй с использованием оператора for, в котором последовательно опрашиваются столбцы матричной клавиатуры. результат опроса последовательно помещается в переменную k, начиная с младших разрядов. На рисунке показан процесс сохранения битов 2-го столбца матричной клавиатуры:

Текст программы приводится ниже:

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

void delay1( void ) { #asm( "nop" ); }

const char out_tab[3] = { 0xe0, 0xd0, 0xb0 };

main()
{
        int k, val;
        char i, s;

        DDRD = 0xff;
        DDRE = 0xf0;

        while (1)
        {
         s = 0, k = 0;

         for( i = 0; i < 3; ++i, s += 4 )
         {
                PORTE = out_tab[i];
                delay1();

                val = PINE & 0xf;

                k |= ( val << s );
         }

         PORTD = ~k;
        }

}
```

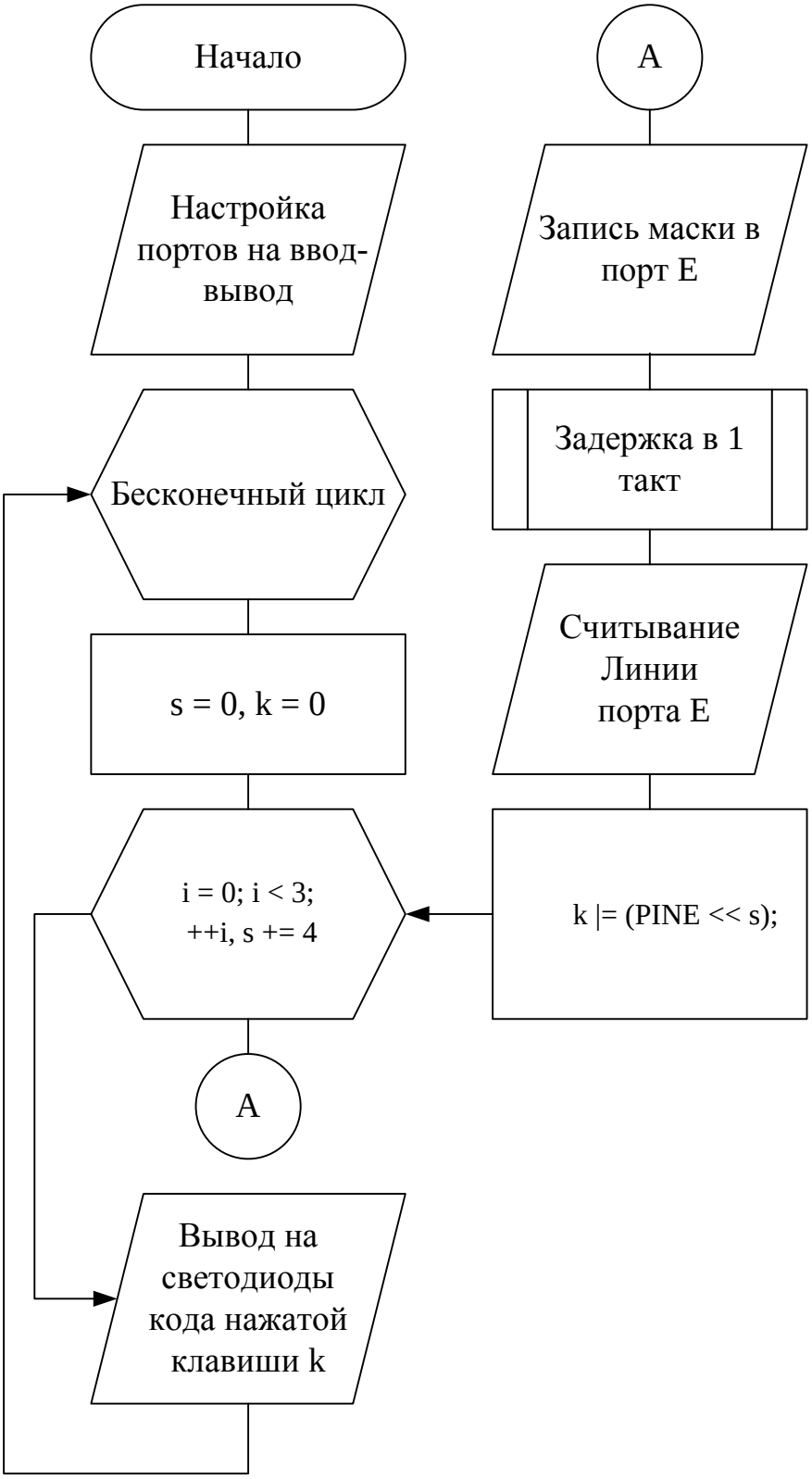


Рис. 4.4. – Алгоритм циклического опроса матричной клавиатуры

4.5 Лабораторная работа №5 – Синтез звука заданной частоты

При подготовке к лабораторной работе необходимо составить предварительный вариант текста программы на языке С и блок-схему алгоритма, в соответствии с индивидуальным заданием (см. таблицу 4.5).

Цель. Изучить принципы формирования звука заданной частоты при помощи лабораторного макета.

Задание. Разработать в среде программирования Code Vision AVR программу на языке С для микроконтроллера ATMEGA 128, синтезирующую звук согласно варианту.

Для выполнения задания необходимо оформить функцию синтеза в виде отдельной функции play_sound принимающую в качестве аргумента величину задержки. Для задания задержки необходимо подключить библиотеку delay.h. Для задания задержки в микросекундах используется функция delay_us (работает только с константными данными), а для задания задержки в миллисекундах delay_ms. Для формирования звука необходимо менять 4 линию порта B (PORTB.4) с заданной частотой. Предварительно эту линию необходимо настроить на вывод.

Порядок выполнения задания показан в разделе 3.1.

В отчете необходимо привести следующее:

- характеристики лабораторной вычислительной системы;
- исходный код разработанной программы;
- анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности работы с ЖКИ лабораторного макета.

Контрольные вопросы и задания:

1) Цикл while.
2) Принцип синтеза звука на лабораторном макете.
3) Принцип управления портами ввода-вывода из программы на языке С.
4) Операции битового сдвига.
5) Условный оператор.
6) Процедуры и функции в языке С.

Таблица 4.5

Задания на лабораторную работу №5

Вариант	Задание: Разработать программу, синтезирующую звук с частотой …
1	… 2 КГц, если не нажата ни одна клавиша 3-х кнопочной клавиатуры и звук с частотой 500 Гц, если нажата клавиша B0, и не издавать звук если нажаты другие клавиши.
2	… 10 КГц, если нажаты все клавиши 3-х кнопочной клавиатуры и звук с частотой 2 КГц, если нажата одна из клавиш. Состояние клавиш индицировать на светодиодах.
3	… 1, 2, 3 КГц, если нажаты клавиши B0, B1, B2 соответственно, в противном случае не издавать звук.
4	… 0.5, 2, 4 КГц, если нажаты клавиши B2, B1, B0 соответственно, в противном случае издавать звук с частотой 10КГц.
5	… 0.5 + 1 * n КГц, где n – количество нажатых клавиш на 3х кнопочной клавиатуре.
6	… 0.5, 1, 2, 3, 4 КГц последовательно, звук каждой частоты звучит в течении 1 секунды.
7	… 0.2 или 1 КГц, плавно изменяя частоту (в течении 2-х секунд) вверх если нажата клавиша B2 или вниз, если нажата клавиша B0. Изначально звук синтезируется с частотой 0.5 КГц.
8	… 1 КГц, если нажаты любые две клавиши 3-х кнопочной клавиатуры и звук с частотой 0.5 КГц, если нажата одна из клавиш. Состояние клавиш индицировать на светодиодах.

Пример задания. Разработать программу, синтезирующую звук с частотой 1 КГц, если не нажата ни одна клавиша 3-х кнопочной клавиатуры и звук с частотой 2 КГц, если нажата любая из клавиш.

Решение. Создаем две функции, одна из которых синтезирует звук с частотой f = 1 КГц, осуществляя задержку на T=1/f = 1/1000 = 1 мс, а вторая синтезирует звук с частотой 2 КГц (используя функцию delay_us), при этом задержка составляет 500 мкс. В бесконечном цикле опрашиваем 3-х кнопочную клавиатуру и при помощи условного оператора запускаем одну из двух функций. Алгоритм программы приведен на рис. 4.5.

Текст программы приводится ниже:

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

void play_sound_2khz()
{
        PORTB.4 = 1;
        delay_us( 500 );
        PORTB.4 = 0;
}

void play_sound_1khz()
{
        PORTB.4 = 1;
        delay_ms( 1 );
        PORTB.4 = 0;
}

main()
{
        DDRB = 0x1f;
        while(1)
        {
                k = ( PINB >> 5 ) & 0x7;
                if( k != 0x7 )
                   play_sound_2khz();
                else
                   play_sound_1khz();
        }
                                                }
```

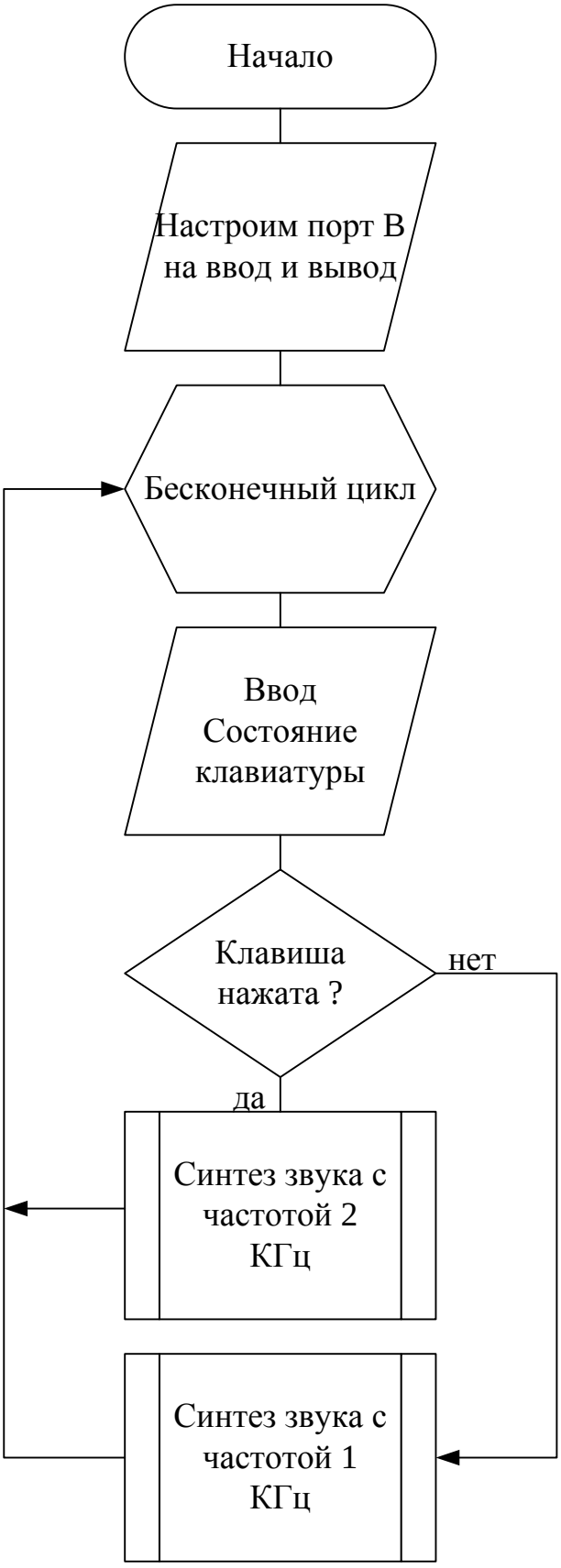


Рис. 4.5. – Алгоритм синтеза звука заданной частоты

4.6 Лабораторная работа №6 – Вывод текстовых и числовых данных на индикатор

При подготовке к лабораторной работе необходимо составить предварительный вариант текста программы на языке С и блок-схему алгоритма, в соответствии с индивидуальным заданием (см. таблицу 4.6).

Цель. Изучить принципы функционирования графико-цифрового ЖК индикатора лабораторного макета, разработать алгоритм и программу для вывода цифровой и текстовой информации.

Задание. Разработать в среде программирования Code Vision AVR программу на языке С для микроконтроллера ATMEGA 128, выводящую на экран текстовую информацию согласно варианту.

Для вывода текстовой информации необходимо подключить библиотеку для работы с ЖКИ "lcd.c" и стандартную библиотеку ввода-вывода <stdio.h>.

Порядок выполнения задания показан в разделе 3.1. Указания к лабораторной работе показаны в разделах 3.5 – 3.6.

В отчете необходимо привести следующее:

- характеристики лабораторной вычислительной системы;
- исходный код разработанной программы;
- анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности работы с ЖКИ лабораторного макета.

Контрольные вопросы и задания:

1) Цикл for.
2) Объявление массивов и констант в языке С.
3) Функции для работы с ЖКИ.
4) Формирование строки при помощи функции sprintf.
5) Процедуры и функции в языке С.
6) Типы данных в языке С.

Таблица 4.6

Задания на лабораторную работу №6

Вариант	Задание: разработать программу, выводящую …
1	… в правом верхнем углу ЖКИ фамилию и группу студента, затем через 2 секунды, по центру справа последовательно выводящую целые числа от -99 до 99 с интервалом в 0,5 секунды.
2	… в правом нижнем углу ЖКИ группу и имя студента, затем через 1 секунду, внизу по центру последовательно выводящую целые числа от 99 до -100 с интервалом в 0,5 секунды.
3	… в правом нижнем углу ЖКИ группу и имя студента, затем через после нажатия кнопки B1, внизу по центру последовательно выводящую вещественные числа от 0.0 до 1.0 с шагом 0.05 и интервалом в 0,5 секунды.
4	… в сверху по центру ЖКИ группу и фамилию студента, затем через после нажатия кнопок B0 и B2, по центру последовательно выводящую целые числа от -10 до 0 с шагом 2 и интервалом в 0,5 секунды.
5	… в правом нижнем углу ЖКИ группу и имя студента, затем через 1 секунду, в бесконечном цикле выводящую в 16-ричной системе по центру экрана код нажатых клавиш B0-B2.
6	… в правом нижнем углу ЖКИ группу и имя студента, затем через 1 секунду, в бесконечном цикле выводящую по центру экрана время в секундах и минутах с начала вывода (в формате MM:CC).
7	… в левом нижнем углу ЖКИ группу и имя студента, затем через 2 секунды, внизу по центру последовательно выводящую целые числа от 100 до -100 с интервалом в 0,5 секунды (в 16-ричной системе исчисления).
8	… в левом верхнем углу ЖКИ группу и фамилию студента, затем через 1 секунду, в бесконечном цикле выводящую по центру экрана время в секундах и миллисекундах с начала вывода (в формате CC:MM).

Пример задания. Разработать программу, выводящую в левом верхнем углу ЖКИ фамилию и группу студента, затем через 1 секунду, слева по центру последовательно выводящую целые числа от 0 до 99 с интервалом в 0,5 секунды.

Решение. Для работы с ЖКИ лабораторного макета используется библиотека "lcd.c". Для формирования строки используется стандартная библиотека ввода-вывода <stdio.h>. Их необходимо подключить в начале программы. Перед началом работы с ЖКИ необходимо инициализировать (настроить ЖКИ) функцией gl_init() библиотеки, а затем очистить экран функцией gl_clear() (см. раздел 3.5). Дальше формируется строка с фамилией и номером группы при помощи функции sprintf (см. раздел 3.6) и производится вывод на экран сформированной строки при помощи функции gl_string по координатам (0,0). После этого в цикле с задержкой, аналогичным образом выводятся числа от 0 до 99. Алгоритм программы приведен на рис. 4.6.

Текст программы приводится ниже:

```
#include <mega128.h>
#include <delay.h>
#include <stdio.h>
#include "lcd.c"

char str[32];

main()
{
        int i;
        gl_init();
        gl_clear();

        sprintf( str, "Ivanov AP-15A" );

        gl_string( 0, 0, str );
        delay_ms( 1000 );

        for( i = 0; i < 100; ++i )
        {
                sprintf( str, "%i", i );
                gl_string( 0, 7, str );
                delay_ms( 500 );
        }
        gl_clear();
}
```

Рис. 4.6. – Алгоритм вывода буквенно-цифровой информации на ЖКИ

4.7 Лабораторная работа №7 – Изучение принципов обработки прерываний

При подготовке к лабораторной работе необходимо составить предварительный вариант текста программы на языке С и блок-схему алгоритма, в соответствии с индивидуальным заданием (см. таблицу 4.7).

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

Задание. Разработать в среде программирования Code Vision AVR программу на языке С для микроконтроллера ATMEGA 128, обрабатывающую прерывания от таймеров счетчиков согласно варианту.

Порядок выполнения задания показан в разделе 3.1. Указания к лабораторной работе показаны в разделе 3.7 и приложениях Б и В.

В отчете необходимо привести следующее:

- характеристики лабораторной вычислительной системы;
- исходный код разработанной программы;
- анализ полученных результатов и краткие выводы по работе, в в которых необходимо отразить особенности использования встроенных таймеров микроконтроллера для формирования аппаратно-независимых временных интервалов.

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.

Таблица 4.7

Задания на лабораторную работу №7

Вариант	Задание: разработать программу, выполняющую в бесконечном цикле …
1	… включение/выключение 1–го светодиода: управляющее прерывание TIM1_OVF, частота счета f = 43 КГц.
2	… включение/выключение 3–го светодиода: управляющее прерывание TIM1_COMPA, f = 43 КГц, OCR1A=40000.
3	… включение/выключение 0–го и 7–го светодиодов: управляющее прерывание TIM1_COMPB, f = 43 КГц, OCR1B=50000.
4	… включение/выключение 3–го и 6–го светодиодов: управляющее прерывание TIM0_OVF, частота счета f = 11 КГц, OCR0=200.
5	… включение/выключение 2–го светодиода: управляющее прерывание TIM1_COMPA, f = 11 КГц, OCR1A=10000.
6	… включение/выключение 1–го и 2–го светодиодов: управляющее прерывание TIM1_OVF, частота счета f = 11 КГц.
7	… включение/выключение 0–го и 5–го светодиодов: управляющее прерывание TIM1_COMPB, f = 43 КГц, OCR1B=30000.
8	… включение/выключение 2–го светодиода: управляющее прерывание TIM0_OVF, частота счета f = 11,7 КГц.

Пример 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 $f = 43\text{КГц}$.

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 3.4):

$$f = \text{CLK}/256 = 11\text{МГц}/256 = 43\text{КГц}.$$

Соответственно значение, заносимое в регистр TCCR1B, равняется 4. Период изменения значений в счетном регистре TCNT1: $T = 1/f = 23\text{мкс}$.

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 1,5с будет осуществляться циклическое перемещение зажженного светодиода. Алгоритм программы приведен на рис. 4.7. Текст программы приводится ниже:

Обработчик прерывания

```
#include <mega128.h>

unsigned char led = 0x0;

interrupt [TIM1_OVF]
void timer1_overflow(void)
{
  led <<= 1;
  if( led == 0 ) ++led;
   PORTD = ~led;
}
```

Основная программа

```
main()
{
  DDRD = 0xff;
  PORTD = 0xff;

  TCCR1A = 0;
  TCCR1B = 4;
  TIMSK = 4;

  #asm( "sei" );
  while( 1 ) {}
}
```

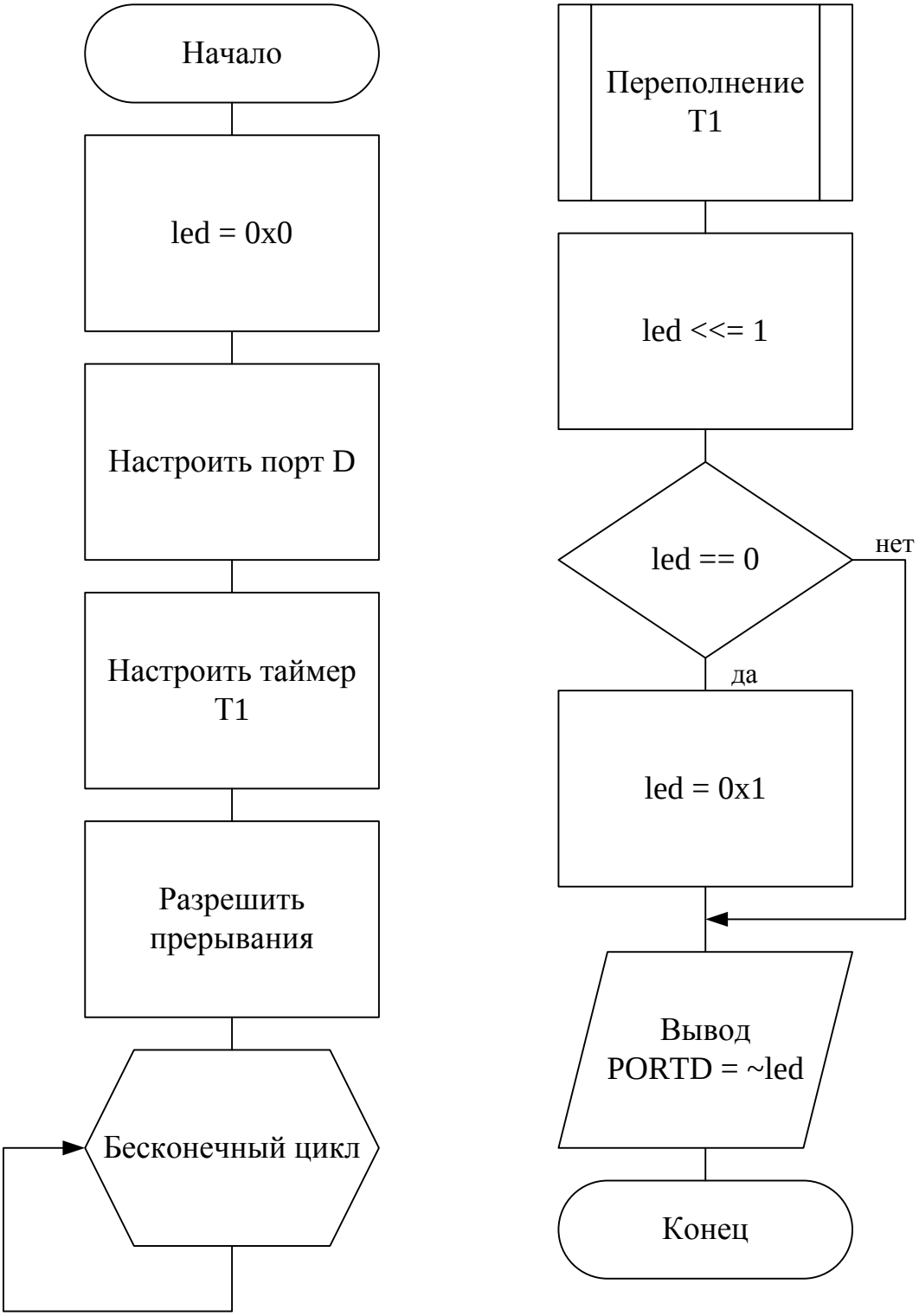


Рис. 4.7. – Алгоритм обработки прерываний

4.8 Лабораторная работа №8 (повышенной сложности) – Вывод графической информации на ЖКИ

При подготовке к лабораторной работе необходимо составить предварительный вариант текста программы на языке С и блок-схему алгоритма, в соответствии с индивидуальным заданием (см. таблицу 4.8).

Цель. Изучить принципы формирования векторных изображений на ЖКИ лабораторного макета, разработать алгоритм и программу для вывода графической информации.

Задание. Разработать в среде программирования Code Vision AVR программу на языке С для микроконтроллера ATMEGA 128, выводящую на экран графическую информацию согласно варианту.

Для вывода графической информации необходимо подключить библиотеку для работы с ЖКИ "lcd.c" и стандартную математическую библиотеку <math.h>.

Порядок выполнения задания показан в разделе 3.1. Указания к лабораторной работе показаны в разделах 3.5 и 3.8.

В отчете необходимо привести следующее:

- характеристики лабораторной вычислительной системы;
- исходный код разработанной программы;
- анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности работы с векторной графикой на ЖКИ лабораторного макета.

Контрольные вопросы и задания:

1) Цикл for.
2) Типы данных в языке С, работа с числами с плавающей запятой.
3) Функции для работы с ЖКИ.
4) Процедуры и функции в языке С.
5) Принципы вывода векторной информации на ЖКИ.
6) Функции математической библиотеки и работа с ними.

Таблица 4.8

Задания на лабораторную работу №8

Вариант	Задание: разработать программу, рисующую…
1	… график функции $\cos(x)$ размахом в 110 пикселей по вертикали и периодом 4π.
2	… график функции $\log_2(x)$ размахом в 120 пикселей по вертикали и 150 по горизонтали, где $x = \overline{1 \div 3}$.
3	… график функции $\sqrt{x}$ размахом в 90 пикселей по вертикали и 200 по горизонтали, где $x = \overline{0 \div 10}$.
4	… график функции $\sin(x) \cdot \cos(x)$ размахом в 110 пикселей по вертикали и периодом 7π.
5	… график функции $\left\|\cos^3(x)\right\|$ размахом в 130 пикселей по вертикали и периодом 5π.
6	… график функции $(\lg(x))^2$ размахом в 100 пикселей по вертикали и 120 по горизонтали, где $x = \overline{1 \div 2}$.
7	… график функции x^2 размахом в 90 пикселей по вертикали и 180 по горизонтали, где $x = \overline{1 \div 2}$.
8	… график функции e^x размахом в 120 пикселей по вертикали и 200 по горизонтали, где $x = \overline{0 \div 1}$.

Пример задания. Разработать программу, строящую график функции $\sin(x)$ размахом в 120 пикселей по вертикали и периодом 8π.

Решение. Для работы с ЖКИ лабораторного макета используется библиотека "lcd.c". Для работы с математическими функциями используется стандартная библиотека <math.h>. Их необходимо подключить в начале программы. Перед началом работы с ЖКИ необходимо инициализировать (настроить ЖКИ) функцией gl_init() библиотеки, а затем очистить экран функцией gl_clear() (см. раздел 3.5).

Дальше в цикле вычисляется значение функции $\sin(x)$ для текущего угла x (который зависит от переменной цикла и постоянно возрастает), с учетом периодичности. Линией соединяется крайняя уже построенная точка графика $A(i-1, py)$ с вновь вычисленной $B(i, iy)$, после чего запоминается вычисленные координаты точки B, как крайняя точка.

При этом производится вычисление вертикальной координаты iy путем масштабирования значения функции $\sin(x)$ с учетом заданного размаха в пикселях.

Алгоритм программы приведен на рис. 4.8. Текст программы приводится ниже:

```
#include <mega128.h>
#include <delay.h>
#include <math.h>
#include "lcd.c"

#define UP_Y 60
#define SCALE 4.0

main()
{
        int i, yi, py = UP_Y;
        float x, y;

        gl_init();
        gl_clear();

        for( i = 1; i < 180; ++i )
        {       x = i * PI / 180.0 * SCALE;
                y = sin( x );
                yi = (int) ( y * UP_Y + UP_Y );
                gl_line( i – 1, py, i, yi, 1 );
                py = yi;
        }
}
```

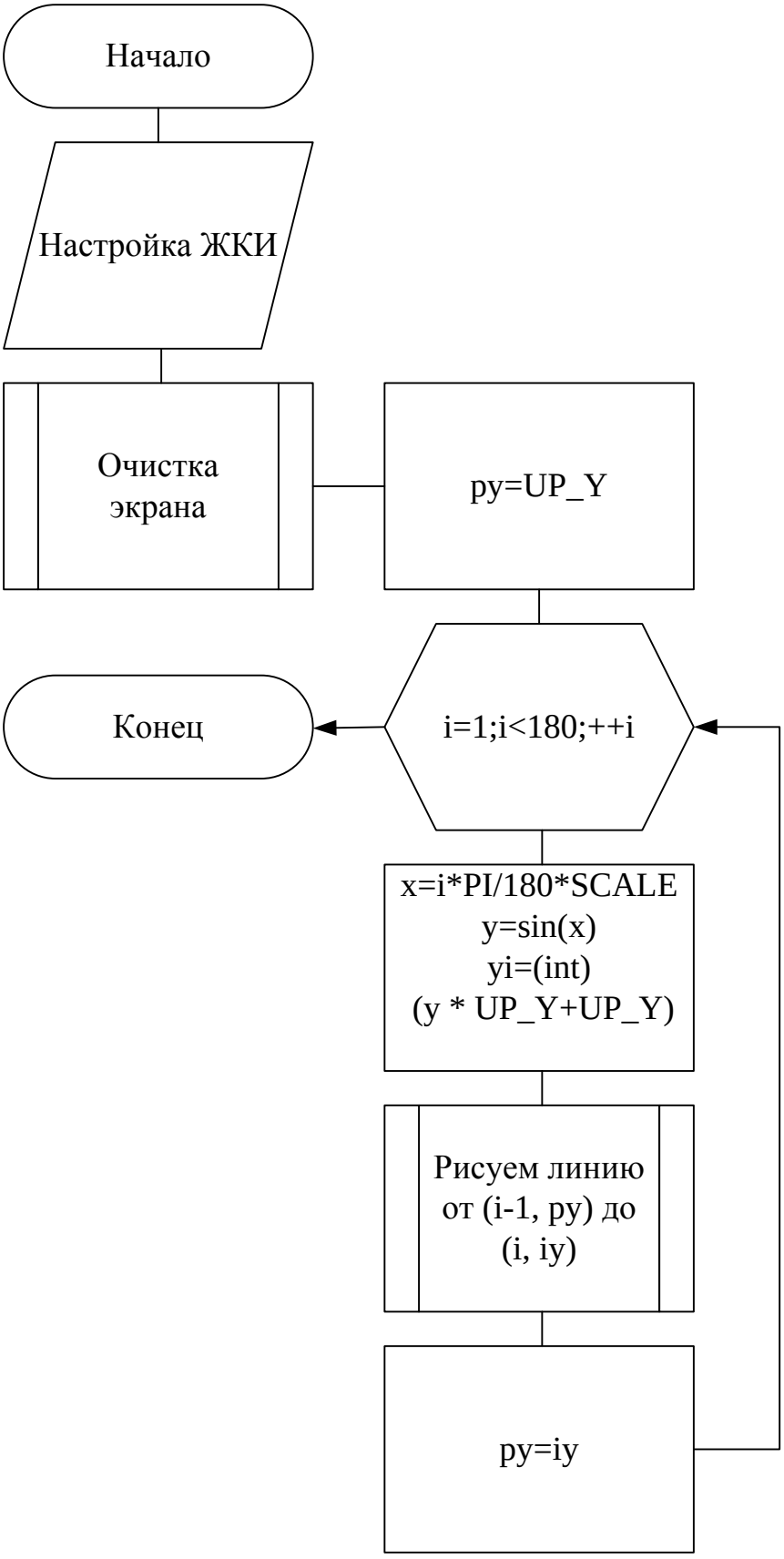


Рис. 4.8. – Алгоритм вывода графика функции $\sin(x)$ на ЖКИ

4.9 Лабораторная работа №9 (повышенной сложности) – Обмен данными по интерфейсу RS-232C между микроконтроллером и ПЭВМ

При подготовке к лабораторной работе необходимо составить предварительный вариант текста программы на языке С и блок-схему алгоритма, в соответствии с индивидуальным заданием (см. таблицу 4.9).

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

Задание. Разработать в среде программирования Code Vision AVR программу на языке С для микроконтроллера ATMEGA 128, передающую и принимающую информацию от ПЭВМ согласно варианту.
Порядок выполнения задания показан в разделе 3.1. Указания к лабораторной работе показаны в разделе 3.9.

Для обмена данными со стороны ПЭВМ необходимо использовать утилиту Terminal (см. рис. 4.9). Дополнительно в работе используется кабель с 9-контактными разъемами DB-9 для соединения лабораторного макета с ПЭВМ через последовательный интерфейс RS232C. Для трехпроводной двунаправленной линии связи используются сигналы RxD, TxD и SG.

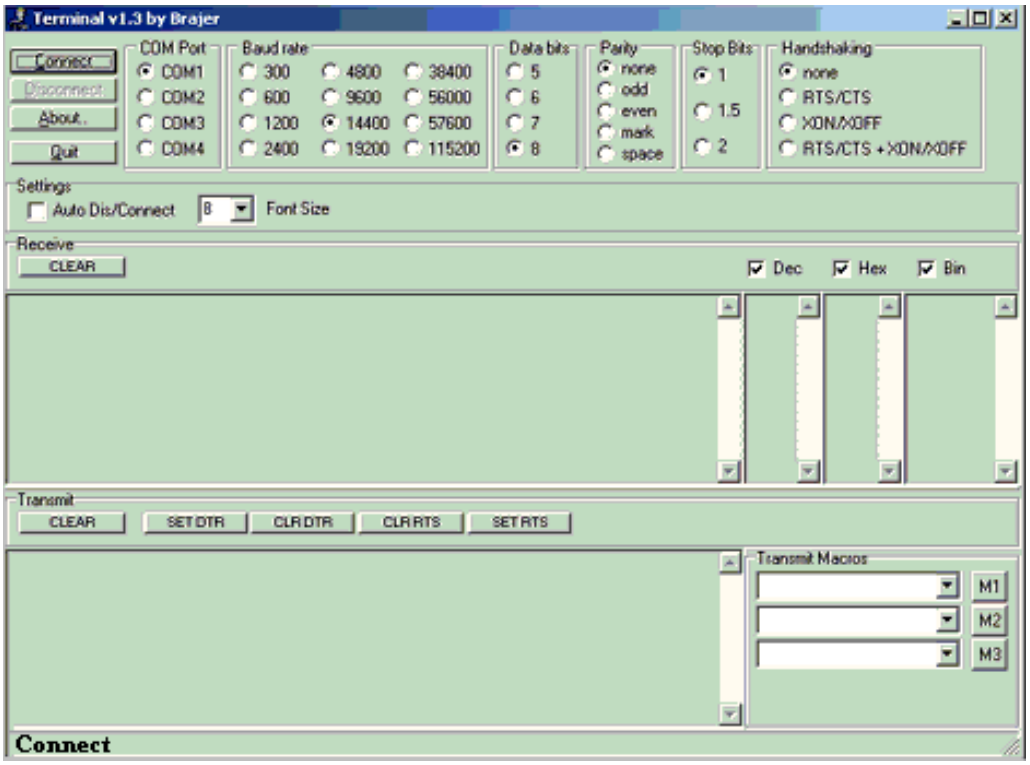


Рис. 4.9. – Рабочее окно программы Terminal

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.

Работа с программой Terminal 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.

В отчете необходимо привести следующее:

- характеристики лабораторной вычислительной системы;
- исходный код 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.

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 данных по интерфейсу RS-232C.

Таблица 4.9

Задания на лабораторную работу №9

Вариант	Задание: разработать программу передачи (модуль USART1) …
1	… 100 чисел (от 0 до 99) из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу RS232C в соответствие с протоколом: скорость обмена данными 19200 бит/с, режим обмена асинхронный, 8 битов данных без бита четности.
2	… 20 чисел (от 10 до 29) из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу RS232C в соответствие с протоколом: скорость обмена данными 57600 бит/с, режим обмена асинхронный, 7 битов данных без бита четности.
3	… 10 чисел (от 0 до 9) из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу RS232C в соответствие с протоколом: скорость обмена данными 14400 бит/с, режим обмена асинхронный, 6 битов данных без бита четности.
4	… номера нажатой клавиши 3-х кнопочной клавиатуры (см. лабораторную работу №3) из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу RS232C в соответствие с протоколом: скорость обмена данными 19200 бит/с, режим обмена асинхронный, 8 битов данных без бита четности.
5	… 50 чисел (от 10 до 59) из ПЭВМ в микроконтроллер AVR ATMEGA 128 по интерфейсу RS232C в соответствие с протоколом: 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.
6	… 200 чисел (от 0 до 199) из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу RS232C в соответствие с протоколом: скорость обмена данными 38400 бит/с, режим обмена асинхронный, 8 битов данных без бита четности.
7	… 10 чисел (от 0 до 9) из ПЭВМ в микроконтроллер AVR ATMEGA 128 по интерфейсу RS232C в соответствие с протоколом: скорость обмена данными 19200 бит/с, режим обмена асинхронный, 8 битов данных без бита четности. Выполнить индикацию принятых данных на экране ЖКИ.
8	… номера нажатой клавиши матричной клавиатуры (см. лабораторную работу №4) из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу RS232C в соответствие с протоколом: скорость обмена данными 14400 бит/с, режим обмена асинхронный, 8 битов данных без бита четности.

Пример задания. Разработать программу для передачи 20 чисел (от 0 до 19) из микроконтроллера AVR ATMEGA 128 в ПЭВМ по интерфейсу RS-232C в соответствии с протоколом: модуль USART1, скорость обмена данными 19200 бит/с, режим обмена асинхронный, 8 бит 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 приведен на рис. 4.10. Текст программы приводится ниже:

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

main()
{
        int i;

        UCSR1A=0x00;
        UCSR1B=0x08;
        UCSR1C=0x04;

        UBRR1H=0x00;
        UBRR1L=35;

        for( i = 0; i < 20; ++i )
        {
                delay_ms(20);
                UDR1=i;
        }

}
```

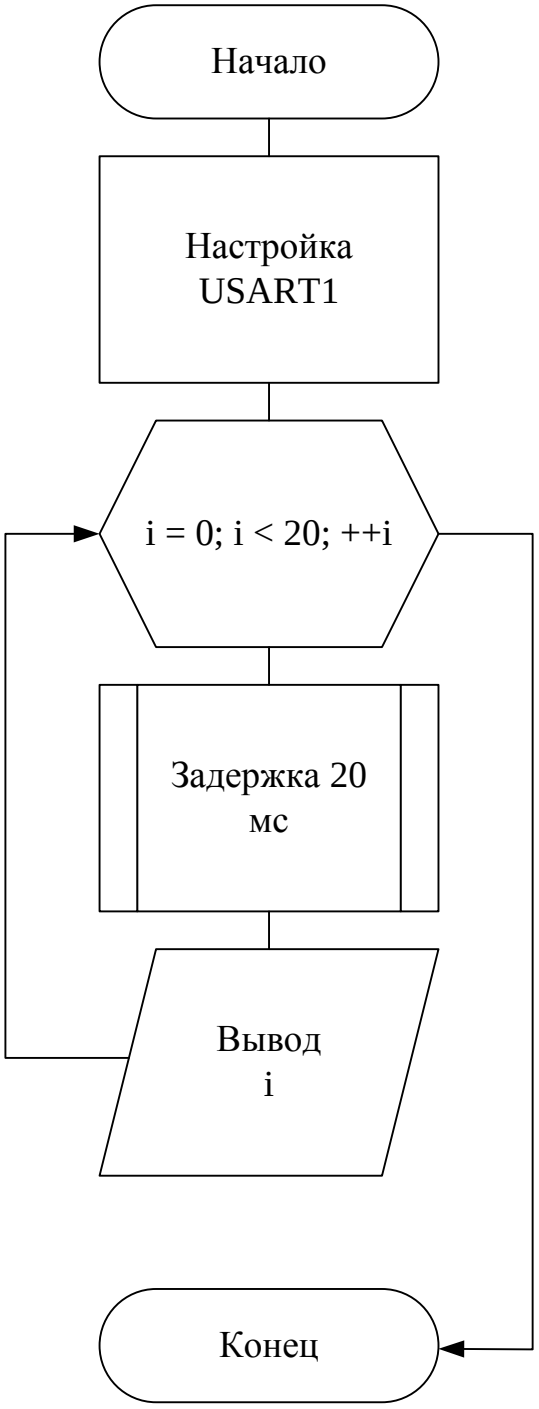


Рис. 4.10. – Алгоритм передачи 20 чисел на ПЭВМ

ПРИЛОЖЕНИЕ А
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 R88, (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:

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:

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 or Rs;
and Rd, Rs логическое “и”; действие: Rd = Rd and Rs;
eor Rd, Rs “исключающее или”; действие: Rd = Rd eor Rs.

Данные команды выполняют операции поразрядного логического “или”, логического “и”, “исключающего или” (см. таблицу А.1) над операндами, находящимися в регистрах общего назначения, причем результат записывается по адресу первого операнда. Примеры:

or R7, R11 действие: R7 = R7 or R11;
and R10, R11 действие: R10 = R10 and R11;
eor R25, R30 действие: R25 = R25 eor R30.

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.

Таблица А.1

Таблицы истинности логических операций or, end, xor

or (или)			and (и)			xor (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

Команды изменяют флаги нуля, Z, знака N и переполнения V. Примеры поразрядных логических операций, иллюстрирующие применение механизма маскирования битов, приводятся в таблице А.2.

Таблица А.2

Примеры поразрядных логических операций

Пример поразрядного маскирования or		Пример поразрядного маскирования and		Пример поразрядного маскирования eor	
Rd	xxxxxxxx	Rd	xxxxxxxx	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.

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: 0x175 = 0;

sbi 0x40, 1 — действие: 0x401 = 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 разрядов в регистре статуса SREG.

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 осуществлянт безусловный переход по указанному 8-разрядному адресу (метке, label) в памяти команд. Пример:

rjmp lbl2	безусловный переход на метку lbl2.

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

rjmp lbl3	безусловный переход на метку lbl3.

Список инструкций микроконтроллера приведен в таблице А.3.

Таблица А.3

Список инструкций микроконтроллера AVR ATMEGA 128

Мнемо-ника	Опе-ранды	Описание	Операция	Флаги	Цик-лы
1	2	3	4	5	6
Арифметические и логические команды					
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 · Rr	Z,N,V,S	1
ANDI	Rd,K8	Логическое И с константой	Rd = Rd · K8	Z,N,V,S	1
OR	Rd,Rr	Логическое ИЛИ	Rd = Rd V Rr	Z,N,V,S	1
ORI	Rd,K8	Логическое ИЛИ с константой	Rd = Rd V K8	Z,N,V,S	1
EOR	Rd,Rr	Логическое исключающее ИЛИ	Rd = Rd EOR 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 = $00 - Rd	Z,C,N,V,H,S	1
SBR	Rd,K8	Установить бит (биты) в регистре	Rd = Rd V K8	Z,C,N,V,S	1
CBR	Rd,K8	Сбросить бит (биты) в регистре	Rd = Rd · ($FF - K8)	Z,C,N,V,S	1

Таблица А.3 (продолжение 1)

Список инструкций микроконтроллера AVR ATMEGA 128
(арифметические и логические операции)

1	2	3	4	5	6
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 · 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	Rdl,K6	Сложить константу и слово	Rdh:Rdl = Rdh:Rdl + K6	Z,C,N,V,S	2
SBIW	Rdl,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) << 1	Z,C	2
FMULS	Rd,Rr	Умножение дробных чисел со знаком	R1:R0 = (Rd *Rr) << 1	Z,C	2
FMULSU	Rd,Rr	Умножение дробного числа со знаком с числом без знака	R1:R0 = (Rd * Rr) << 1	Z,C	2

Таблица А.3 (продолжение 2)

Список инструкций микроконтроллера AVR ATMEGA 128

1	2	3	4	5	6
Команды ветвления					
RJMP	k	Относительный переход	PC = PC + k +1		2
IJMP	Нет	Косвенный переход на (Z)	PC = Z		2
EIJMP	Нет	Расширенный косвенный переход на (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND		2
JMP	k	Переход	PC = k		3
RCALL	k	Относительный вызов подпрограммы	STACK = PC+1, PC = PC+k+1		3/4*
ICALL	Нет	Косвенный вызов (Z)	STACK = PC+1, PC = Z		3/4*
EICALL	Нет	Расширенный косвенный вызов (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) =EIND		4*
CALL	k	Вызов подпрограммы	STACK = PC+2, PC = k		4/5*
RET	Нет	Возврат из подпрограммы	PC = STACK		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 or 3		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,K8	Сравнить с константой	Rd - K	Z,C, N,V, H,S	1

Таблица А.3 (продолжение 3)

Список инструкций микроконтроллера AVR ATMEGA 128
(Команды ветвления)

1	2	3	4	5	6
SBRC	Rr,b	Пропустить если бит в регистре очищен	if(Rr(b)==0) PC = PC+2 or 3		1/2/3
SBRS	Rr,b	Пропустить если бит в регистре установлен	if(Rr(b)==1) PC = PC+2 or 3		1/2/3
SBIC	P,b	Пропустить если бит в порту очищен	if(I/O(P,b)==0) PC = PC+ or 3		1/2/3
SBIS	P,b	Пропустить если бит в порту установлен	if(I/O(P,b)==1) PC = PC+2 or 3		1/2/3
BRBC	s,k	Перейти если флаг в SREG очищен	if(SREG(s)==0) PC = PC+k+1		½
BRBS	s,k	Перейти если флаг в SREG установлен	if(SREG(s)==1) PC = PC+k+1		½
BREQ	k	Перейти если равно	if(Z==1) PC = PC + k + 1		½
BRNE	k	Перейти если не равно	if(Z==0) PC = PC + k + 1		½
BRCS	k	Перейти если перенос установлен	if(C==1) PC = PC + k + 1		½
BRCC	k	Перейти если перенос очищен	if(C==0) PC = PC + k + 1		½
BRSH	k	Перейти если равно или больше	if(C==0) PC = PC + k + 1		½
BRLO	k	Перейти если меньше	if(C==1) PC = PC + k + 1		1/2
BRMI	k	Перейти если минус	if(N==1) PC = PC + k + 1		1/2
BRPL	k	Перейти если плюс	if(N==0) PC = PC + k + 1		1/2

Таблица А.3 (продолжение 4)

Список инструкций микроконтроллера AVR ATMEGA 128
(Команды ветвления)

1	2	3	4	5	6
BRGE	k	Перейти если больше или равно (со знаком)	if(S==0) PC = PC + k + 1		1/2
BRLT	k	Перейти если меньше (со знаком)	if(S==1) PC = PC + k + 1		1/2
BRHS	k	Перейти если флаг внутреннего переноса установлен	if(H==1) PC = PC + k + 1		1/2
BRHC	k	Перейти если флаг внутреннего переноса очищен	if(H==0) PC = PC + k + 1		1/2
BRTS	k	Перейти если флаг T установлен	if(T==1) PC = PC + k + 1		1/2
BRTC	k	Перейти если флаг T очищен	if(T==0) PC = PC + k + 1		1/2
BRVS	k	Перейти если флаг переполнения установлен	if(V==1) PC = PC + k + 1		1/2
BRVC	k	Перейти если флаг переполнения очищен	if(V==0) PC = PC + k + 1		1/2
BRIE	k	Перейти если прерывания разрешены	if(I==1) PC = PC + k + 1		1/2
BRID	k	Перейти если прерывания запрещены	if(I==0) PC = PC + k + 1		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 А.3 (продолжение 5)

Список инструкций микроконтроллера AVR ATMEGA 128

1	2	3	4	5	6
Команды пересылки данных					
MOV	Rd,Rr	Скопировать регистр	Rd = Rr		1
MOVW	Rd,Rr	Скопировать пару регистров	Rd+1:Rd = Rr+1:Rr, r,d 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

Таблица А.3 (продолжение 6)

Список инструкций микроконтроллера AVR ATMEGA 128
(Команды пересылки данных)

1	2	3	4	5	6
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 А.3 (продолжение 7)

Список инструкций микроконтроллера AVR ATMEGA 128

1	2	3	4	5	6
Команды для работы с битами					
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,...,6	Z,C,N, V,S	1
SWAP	Rd	Перестановка тетрад	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)		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		2
CBI	P,b	Очистить бит в порту	I/O(P,b) = 0		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		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

Таблица А.3 (продолжение 8)
Список инструкций микроконтроллера AVR ATMEGA 128
(Команды для работы с битами)

1	2	3	4	5	6
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
SEH	Нет	Установить флаг внутреннего переноса	H = 1	H	1
CLH	Нет	Очистить флаг внутреннего переноса	H = 0	H	1
NOP	Нет	Нет операции	Нет		1
SLEEP	Нет	Спать (уменьшить энергопотребление)	Смотрите описание инструкции		1
WDR	Нет	Сброс сторожевого таймера	Смотрите описание инструкции		1

В языке Assembler нет различия в регистре символов, а 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).

ПРИЛОЖЕНИЕ Б
Векторы прерываний микроконтроллера AVR ATMEGA 128

Таблица Б.1

Номера векторов прерываний и идентификаторы процедур-обработчиков прерываний микроконтроллера AVR ATMEGA 128

№	Адрес	Источник прерывания	Идентификатор прерывания	Описание прерывания
1	2	3	4	5
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 «A» таймера/счетчика T1
14	001Ah	TIMER1 COMPB	TIM1_COMPB	Совпадение «B» таймера/счетчика T1
15	001Ch	TIMER1 OVF	TIM1_OVF	Переполнение таймера/счетчика T1
16	001Eh	TIMER0 COMP	TIM0_COMP	Совпадение таймера/счетчика T0

Таблица Б.1 (продолжение)

Номера векторов прерываний и идентификаторы процедур-обработчиков прерываний микроконтроллера AVR ATMEGA 128

1	2	3	4	5
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 «C» таймера-счетчика T1
26	0032h	TIMER3 CAPT	TIM3_CAPT	Захват таймера-счетчика T3
27	0034h	TIMER3_COMPA	TIM3_COMPA	Совпадение «A» таймера-счетчика T3
28	0036h	TIMER3_COMPB	TIM3_COMPB	Совпадение «B» таймера-счетчика T3
29	0038h	TIMER3_COMPC	TIM3_COMPC	Совпадение «B» 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

ПРИЛОЖЕНИЕ В
Аппаратные таймеры/счетчики, входящие в состав микроконтроллера AVR ATMEGA 128

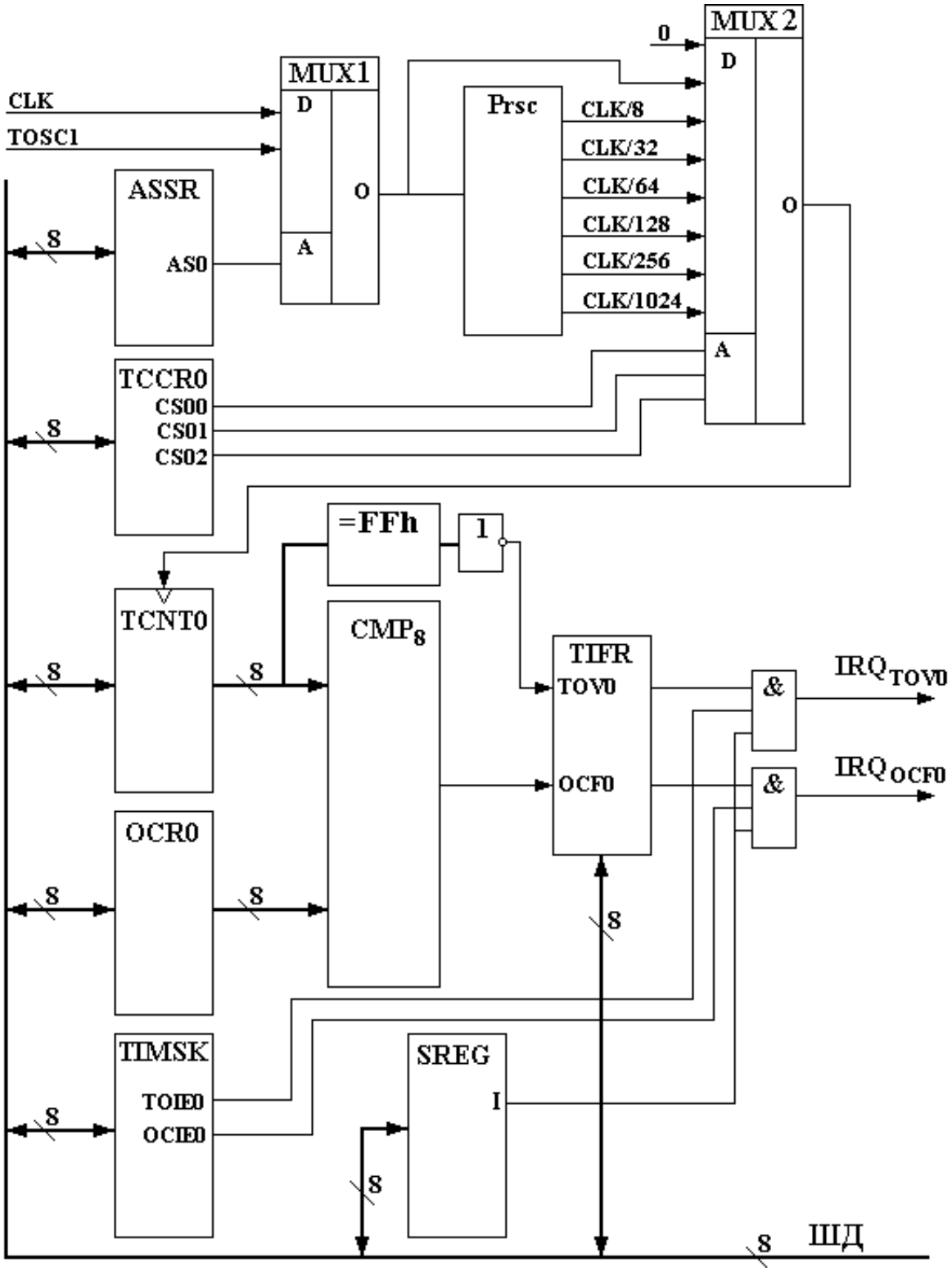


Рис. В.1. – Структурная схема 8-ми разрядного таймера/счетчика Т0

ПРИЛОЖЕНИЕ Г
Исходные коды библиотеки для работы с ЖКИ

```
typedef unsigned char t_byte;
typedef unsigned int t_uint;


#define	GRAPHICS_HOME0  0x281
#define	GRAPHICS_WIDTH  40
#define	GRAPHICS_LINES  128
#define	GRAPHICS_ROWS 240

#define	TEXT_HOME0  0x0000
#define	TEXT_WIDTH  40
#define	TEXT_LINES  16
#define MUL

#define EXTERNAL_CG_HOME	0x1400


#define LCD_DATA	(*((volatile t_byte*)(0x8000)))
#define LCD_COMMAND  (*((volatile t_byte*)(0x8101)))

void gl_wait() { while( ((LCD_COMMAND) & 3) != 3 );   }
void gl_command( t_byte com ) { LCD_COMMAND = com;  gl_wait();   }
void gl_data( t_byte data ) { LCD_DATA = data; gl_wait();  }

t_byte gl_peek_data( void )
{
	gl_command( 0xC1 );
	gl_wait();

	delay_us( 5 );
	return LCD_DATA;
}

void gl_cdata( t_byte com, t_uint address )
{
	t_byte data;
	data = address & 0xFF;
	gl_data( data );
	data = ( address >> 8 ) & 0xFF;
	gl_data( data );
	gl_command( com );
}
```

```
void gl_ptr( t_uint address ) { gl_cdata( 0x24, address );   }
void gl_text_window( t_uint address ) { gl_cdata( 0x40, address ); }

void gl_graphic_window( t_uint address ) { gl_cdata( 0x42, address ); }

void gl_copy_mem( t_uint src, t_uint dest, t_uint len )
{
        t_uint i;
        t_byte data;

        for( i = 0; i < len; ++i )
        {
                gl_ptr( src + i );
                data = gl_peek_data();
                gl_ptr( dest + i );
                gl_data( data );
                gl_command( 0xC0 );
        }// for( i = 0; i < len; ++i )
}

 void gl_init( void )
{
        DDRF = 0x00;
        PORTF = 0x00;
        MCUCR = 0x8a;
        XMCRA = 0xff;
        SFIOR = 0x10;

        gl_command( 0xB2 );
        gl_graphic_window( GRAPHICS_HOME0 );
        gl_data( GRAPHICS_WIDTH );
        gl_data( 0x00 );
        gl_command( 0x43 );
        gl_text_window( TEXT_HOME0 );

        gl_data( TEXT_WIDTH );
        gl_data( 0x00 );
        gl_command( 0x41 );

        gl_command( 0b10000000 );
        gl_command( 0b10011100 );
        gl_command( 0xA3 );
        gl_command( 0x01 );
```

```
        gl_command( 0x00 );
        gl_command( 0x21 );
 }

void gl_blank( t_byte x, t_byte y, t_byte w, t_byte h )
{
        t_byte xi;
        t_uint address;

        address = GRAPHICS_HOME0 + (t_uint) y * GRAPHICS_WIDTH + x;
        gl_ptr( address );

        while( h-- )
        {
                xi = w;
                while( xi-- )
                {
                  gl_data(0);
                  gl_command( 0xC0 );
                }// while( xi-- )

                address += GRAPHICS_WIDTH;
                gl_ptr( address );
        }// while( h-- )
}

void gl_clear_text( void )
{
        t_uint i;
        gl_ptr( TEXT_HOME0 );

        for( i = 0; i < ( TEXT_WIDTH * TEXT_LINES ); ++i )
        {
                gl_data( 0 );
                gl_command( 0xC0 );
        }// for( i = 0; i < TEXT_WIDTH * TEXT_LINES; ++i )
}

void gl_clear_graphics( void )
{
        gl_blank( 0, 0, GRAPHICS_WIDTH, GRAPHICS_LINES );
}
```

```
void gl_pixel( t_byte x, t_byte y, t_byte data )
{
	t_byte bit_pos;
	bit_pos = x % 8; x /= 8;
	gl_ptr( GRAPHICS_HOME0 + ( (t_uint) y * GRAPHICS_WIDTH ) + x );
	gl_command( 0xF0 | data << 3 | ( 7 - bit_pos ) );
}

void gl_pixel_s( t_byte x, t_byte y, t_byte data )
{
	t_byte bit_pos;
	if( x > GRAPHICS_ROWS || y > GRAPHICS_LINES ) return;
	bit_pos = x % 8; x /= 8;
	gl_ptr( GRAPHICS_HOME0 + ( (t_uint) y * GRAPHICS_WIDTH ) + x );
	gl_command( 0xF0 | data << 3 | ( 7 - bit_pos ) );
}

void gl_pixel_xor_s( t_byte x, t_byte y )
{
	t_uint address;
	t_byte bit_pos, prev, data = 1;
	if( x > GRAPHICS_ROWS || y > GRAPHICS_LINES ) return;
	bit_pos = x % 8; x /= 8;
	address = GRAPHICS_HOME0 + ( (t_uint) y * GRAPHICS_WIDTH ) + x;
	gl_ptr( address );
	prev = gl_peek_data();
	if( ( prev >> ( 7 - bit_pos ) ) & 0x1 ) data = 0;
	gl_ptr( address );
	gl_command( 0xF0 | data << 3 | ( 7 - bit_pos ) );
}

t_byte gl_peek_pixel_s( t_byte x, t_byte y )
{
	t_uint address;
	t_byte bit_pos, prev;
	if( x > GRAPHICS_ROWS || y > GRAPHICS_LINES ) return 0;

	bit_pos = x % 8; x /= 8;
	address = GRAPHICS_HOME0 + ( (t_uint) y * GRAPHICS_WIDTH ) + x;
	gl_ptr( address );
	prev = gl_peek_data();
	return ( ( prev >> ( 7 - bit_pos ) ) & 0x1 ) ? 1 : 0;
}
```

```
void gl_string( t_byte x, t_byte y, char* ptr )
{
	gl_ptr( TEXT_HOME0 + ( (t_uint) y * TEXT_WIDTH ) + x );
	while( *ptr )
	{
		gl_data( *ptr - 0x20 );
		gl_command( 0xC0 );
		ptr++;

	}// while( *ptr )
}

void gl_string_f( t_byte x, t_byte y, char flash* ptr )
{
	gl_ptr( TEXT_HOME0 + ( (t_uint) y * TEXT_WIDTH ) + x );

	while( *ptr )
	{
		gl_data( *ptr - 0x20 );
		gl_command( 0xC0 );
		ptr++;

	}// while( *ptr )
}

void gl_clear( void )
{
	t_uint y;
	t_byte d = 0;

	gl_cdata( 0x24, 0 );
	gl_command( 0xb0 );

	for( y = 0; y < 8192; ++y )
		LCD_DATA = d;

	gl_command( 0xb2 );
	gl_cdata( 0x24, 0 );
	gl_cdata( 0x21, 0 );

	gl_clear_text();
	gl_clear_graphics();
}
```

```
void gl_row( t_byte data )
{
	t_uint address;
	t_byte low_byte, high_byte;
	address = ( (t_uint) data * 40 );
	low_byte = address & 0xFF;
	high_byte = ( address >> 8 ) & 0xFF;

	LCD_DATA = low_byte;
	LCD_DATA = high_byte;
	LCD_COMMAND = 0x24;
}

void gl_line( int startx, int starty, int endx, int endy, t_byte mask )
{
	int t, distance;
	int xerr = 0, yerr = 0, delta_x, delta_y;
	int incx, incy;

	delta_x = endx - startx;
	delta_y = endy - starty;

	if( delta_x > 0 ) incx = 1;
	else if( delta_x == 0 ) incx = 0; else incx = -1;
	if( delta_y > 0 ) incy = 1;
	else if( delta_y == 0 ) incy = 0; else incy = -1;

	if( delta_x < 0 ) delta_x =- delta_x;
	if( delta_y < 0 ) delta_y =- delta_y;
	if( delta_x > delta_y ) distance = delta_x;
	else distance = delta_y;

	for( t = 0; t <= distance + 1; ++t )
	{
		gl_pixel_s( startx, starty, mask );
		xerr += delta_x;
		yerr += delta_y;
		if( xerr > distance ) { xerr -= distance; startx += incx; }
		if( yerr > distance ) { yerr -= distance; starty += incy; }
	}// for( t = 0; t <= distance + 1; ++t )
}
```

```
void gl_line_xor( int startx, int starty, int endx, int endy )
{
	int t, distance;
	int xerr = 0, yerr = 0, delta_x, delta_y;
	int incx, incy;

	delta_x = endx - startx;
	delta_y = endy - starty;

	if( delta_x > 0 ) incx = 1;
	else if( delta_x == 0 ) incx = 0; else incx = -1;
	if( delta_y > 0 ) incy = 1;
	else if( delta_y == 0 ) incy = 0; else incy = -1;

	if( delta_x < 0 ) delta_x =- delta_x;
	if( delta_y < 0 ) delta_y =- delta_y;
	if( delta_x > delta_y ) distance = delta_x;
	else distance = delta_y;

	for( t = 0; t <= distance + 1; ++t )
	{
		gl_pixel_xor_s( startx, starty );
		xerr += delta_x;
		yerr += delta_y;
		if( xerr > distance ) { xerr -= distance; startx += incx; }
		if( yerr > distance ) { yerr -= distance; starty += incy; }

	}// for( t = 0; t <= distance + 1; ++t )
}
```

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., Кириченкова 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.