МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра электронных приборов

Методические указания для лабораторных работ по дисциплине «Применение микроконтроллеров»

Микроконтроллер ATmega128A

Версия: 1.0
Для аппаратной версии 4.20

Составили: Микерин В.А.
Чипурнов С.А.

Новосибирск 2012

Содержание

ВВЕДЕНИЕ

Данный лабораторный практикум предназначен для закрепления знаний, полученных в лекционном курсе «Применение микроконтроллеров» и приобретения опыта программирования микроконтроллеров на языке ассемблер. Практикум разработан для лабораторного стенда с микроконтроллером ATmega128A.

1 ОБЩИЕ СВЕДЕНИЯ О МИКРОКОНТРОЛЛЕРЕ ATmega128A

В данном цикле лабораторных работ изучаются AVR микроконтроллеры (МК) на примере МК ATmega128A фирмы Atmel. Ниже дан краткий теоретический и справочный материал, необходимый для выполнения данных лабораторных работ. Более подробную информацию можно получить в конспектах лекций и учебниках, а также в документации на данный микроконтроллер.

Процессорное ядро AVR

МК семейства Mega используют разработанное фирмой Atmel процессорное ядро AVR, которое проектировалось с учетом особенностей компилятора языка Си и относится к RISC ядрам. С целью увеличения производительности используется гарвардская архитектура с раздельными шинами данных и команд. Для разработки программного обеспечения может использоваться поставляемый Atmel с AVRStudio ассемблеры, а также компиляторы и ассемблеры третьих фирм, таких как IAR Systems, ImageCraft Inc. и свободно распространяемый пакет GNU Compilers Collection (GCC). Ядро содержит арифметико-логическое устройство, которое способно выполнять большинство команд за один такт, 32 8-битных регистра общего назначения, три пары которых могут составлять 16-битные регистры X, Y и Z.

Система команд и тактирование

Система команд ATmega128A включает 133 инструкции, большая часть которых выполняется за один такт, что обеспечивается одноуровневым конвейером. Пока одна инструкция выполняется, другая загружается из памяти и декодируется. На рис. 1 представлена параллельная загрузка и выполнение команд. Система команд AVR приведена в **приложении А**, которое содержит мнемонику, количество байтов и количество тактовых циклов для каждой команды.

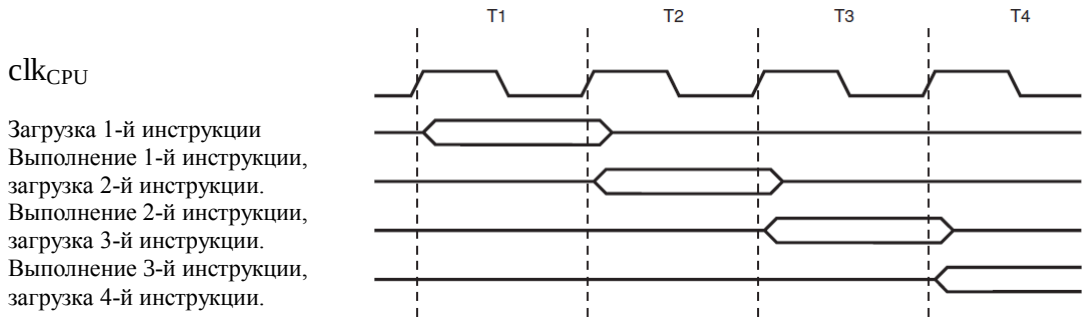


Рисунок 1 — Параллельная загрузка и выполнение команд

Благодаря конвейерной архитектуре AVR, количество тактовых циклов, требуемых для выполнения большинства команд, равно количеству слов в команде. Команды условных переходов требуют для завершения на один цикл меньше, если переход не происходит (по сравнению с тем случаем, когда переход происходит).

Организация памяти

Организация памяти МК с ядром AVR соответствует гарвардской архитектуре. Имеется две отдельных области памяти: память программ и память данных, которые имеют собственное адресное пространство, доступ к ним осуществляется командами различного типа. Объем памяти данных зависит от модели МК, для ATmega128A он составляет 4 Кбайт. Адресное пространство внутренней памяти программ ATmega128A составляет 128 Кбайт. Организация памяти ATmega128A показана ниже на рисунке 2.

1.3.1 Память программ

Поскольку все инструкции AVR являются 16 или 32 битными, Flash-память программ имеет организацию 64 К x 16 и разделена на две области: область загрузчика и область прикладной программы.

Память программ может использоваться как для чтения, так и для записи. Эта возможность позволяет МК ATmega128A обновлять программный код и использовать память программ для долговременного хранения данных.

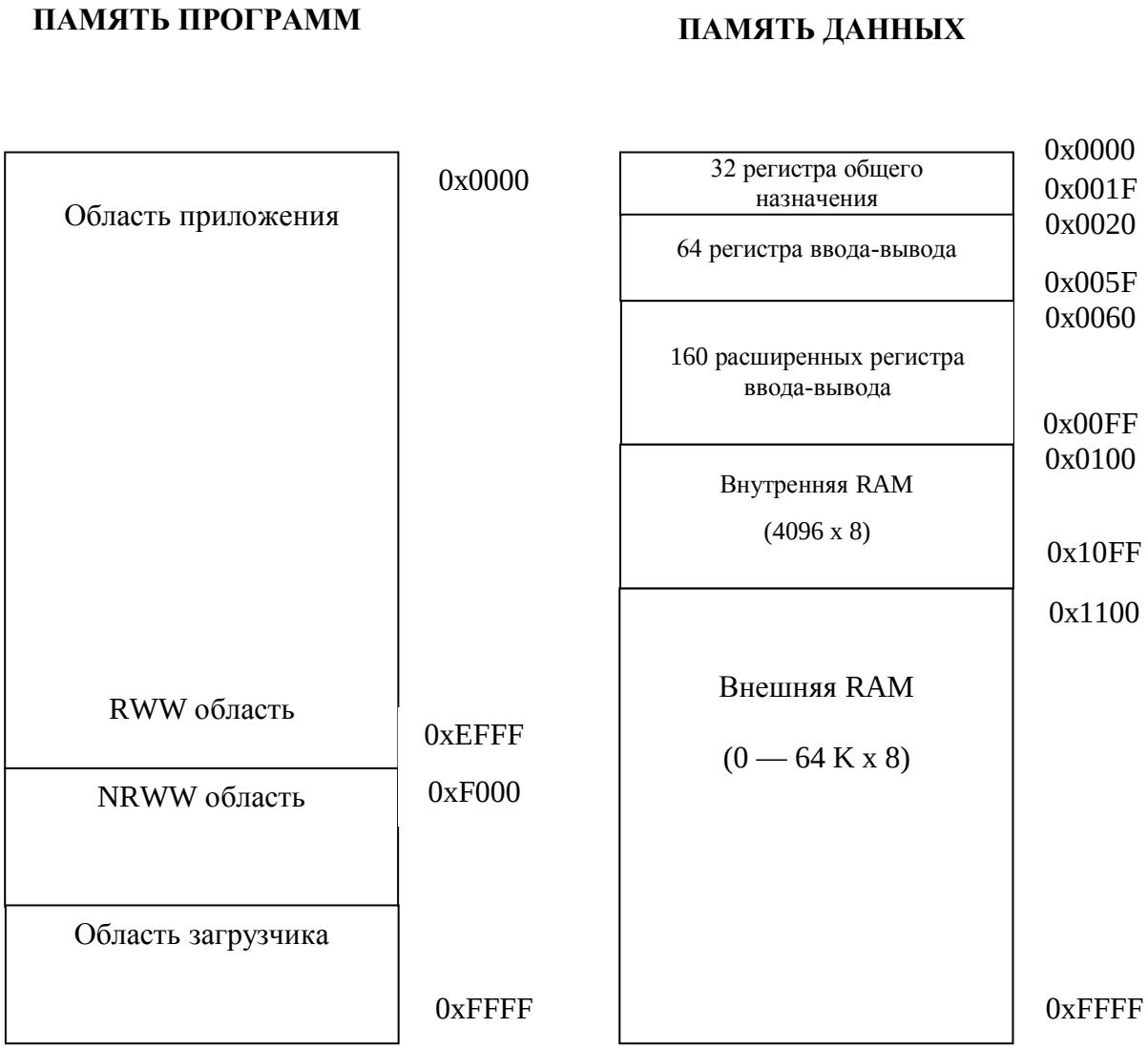


Рисунок 2 — Карта распределения памяти

1.3.2 Память данных

Физически реализовано 4 Кбайт внутреннего ОЗУ, которое поддерживает две конфигурации памяти: стандартный режим и режим совместимости с ATmega103. Далее будет описываться стандартная конфигурация. Младшие 32 байта памяти данных используются для регистров общего назначения (РОН). Следующие 64 байта адресного пространства используются для обращения к регистрам ввода-вывода за которыми следуют дополнительные 160 регистров ввода-вывода.

Старшие 160 регистров ввода-вывода доступны только при использовании инструкций работы с памятью, тогда как для работы с младшими 64 регистрами могут также использоваться специальные инструкции с побитовой адресацией.

1.3.3 Стек

Программный стек может быть размещен в любом месте 4096-байтной памяти данных. Область стека определяется с использованием указателя стека (Stack Pointer), который занимает два 8-битных регистра SPL и SPH, расположенных в пространстве регистров ввода-вывода. SP всегда указывает на верх стека. Значение, загружаемое в стек, размещается по адресу SP и затем SP декрементируется. После сброса SP имеет значение 0, тогда как для нормальной работы SP должен иметь значение больше 0x60. Поэтому перед использованием стека SP необходимо инициализировать. Как правило, стек инициализируется старшим адресом оперативной памяти.

1.3.4 Регистры общего назначения

Младшие 32 адреса общего адресного пространства данных занимают 32 регистра общего назначения, обозначаемых R0-R31. Инструкции с непосредственной адресацией можно использовать только с регистрами R16-R31. Регистры R26-R31 образуют три 16-битных регистра X, Y и Z, которые могут использоваться в качестве индексных регистров.

1.3.5 Побитовые операции

Кроме прямого (побайтного) доступа к регистрам ввода-вывода 0x20 - 0x5F, с этими регистрами возможны побитовые операции. Каждый бит имеет битовый адрес от 0x00 до 0x7F. Бит 0 байта 0x20 имеет битовый адрес 0x00, а бит 7 байта 0x20 имеет битовый адрес 0x07. Бит 7 байта 0x2F имеет битовый адрес 0x7F. Битовый доступ можно отличить от байтового доступа по типу используемой команды (операнды исходных данных и результата в первом случае являются битами, во втором – байтами).

1.3.6 Запись в память программ

МК ATmega128A поддерживает программирование памяти программ, что главным образом используется для обновления программного обеспечения, но также подходит для долговременного сохранения данных[1]. Запись в память программ может осуществляться только при выполнении кода из специальной области, называемой секцией загрузчика (Boot Loader Section - BSL). Размер области загрузчика устанавливается при помощи специальных битов конфигурации BOOTSZ. Кроме деления памяти программ на области загрузчика и приложения, существуют также фиксированные области чтения во время записи RWW (Read-While-Write) и невозможности чтения во время записи NRWW (No Read-While-Write). При записи RWW области ядро может осуществлять чтение из области NRWW, тогда как при записи в область NRWW ядро останавливается. Важно, что попытка чтения области RWW при записи в эту область может привести к непредсказуемому результату, поэтому все прерывания, обработчики которых находятся в области RWW, должны быть запрещены.

Запись в память программ осуществляется постранично. Перед записью страницу необходимо выполнить её очистку. Управление процессом записи и стирания осуществляется при помощи регистра SPMCSR. Перед выполнением записи страницы необходимо заполнить данными временный буфер записи.

[1] Следует отметить, что у МК с ядром AVR, как правило, имеется память EEPROM, которая предназначена специально для энергонезависимого, долговременного хранения различных данных.

2 Написание и отладка программ на лабораторном стенде.

Основы ассемблера

Наиболее приближенным к микроконтроллеру является язык Ассемблер. Как правило, каждая инструкция этого языка программирования соответствует инструкции ядра микроконтроллера, что позволяет максимально использовать особенности архитектуры. Познакомиться с основными принципами программирования на языке Ассемблер.

Как и любая программа, хорошая программа на языке Ассемблер начинается с комментария. Начиная с AVRStudio 4.11 в комплект поставки этой интегрированной среды разработки входит Ассемблер второй версии, который поддерживает как синтаксис первой версии, в котором комментарии начинаются с символа ‘;’, так комментарии в стиле Си. Причем поддерживаются как однострочные комментарии, начинающиеся с //, так и многострочные /* */. Далее следуют директивы с определениями регистров, констант, макросов. Собственно текст программы состоит из строк, в каждой из которых может быть написана только одна команда. Строку можно условно разделить на четыре поля, которые отделяются друг от друга символами пробела или табуляции. Первое поле – поле метки. Метка может состоять из набора латинских букв и цифр, первым символом метки должна быть буква или символ подчеркивания. Метка должна завершаться двоеточием. Во втором поле пишется мнемоника команды или директива Ассемблера. В третьем поле указываются операнды команды. Если операндов несколько, они разделяются запятыми. Последнее четвертое поле – поле комментариев. Все эти поля являются опциональными: строка может содержать только инструкцию, метку с комментариями, только комментарии или вовсе ничего не содержать.

Пример:

```
;************************************
;* Пример простой программы.
;* Автор: С.А. Чипурнов
```

```
;* Версия: 1.0
;* Дата: 5.05.2011
;* Описание:
;* Пример для иллюстрации к описанию структуры
;* программы на языке Ассемблер.
;************************************
.include "m128adef.inc"

; Определения регистров.
.def A   =r16
.def B   =r17

; Текст программы
     ldi  A, 2          ; Запись 2 в операнд A.
     ldi  B, 3          ; Запись 3 в операнд B.
     add  A, B          ; Сложение A и B.
infinite_loop:          ; Вечный цикл.
     rjmp    infinite_loop
```

Константы в ассемблере могут быть указаны в разных системах счисления. В зависимости от конкретной реализации по умолчанию используется десятичная или шестнадцатеричная система счисления.

Методика работы со стендом

Есть много программ, предназначенных для написания и отладки программ для микроконтроллеров. Для примера: IAR, AVR Studio. Какие-то из них разработаны фирмами-производителями микроконтроллеров для поддержки своей продукции и распространяются бесплатно, какие-то пишутся сторонними фирмами и поддерживают большой набор контроллеров различных фирм, но являются коммерческими продуктами. Большинство коммерческих продуктов имеют пробные

версии, которые ограничены функционально, но распространяются бесплатно. Для выполнения данного курса работ воспользуемся средой разработки AVR Studio. С её помощью отладку программы можно производить и без реального устройства в режиме симуляции. Это поможет Вам готовиться дома. В терминальном классе при наличии учебного стенда лучше производить отладку программы в реальном микроконтроллере. Для этого стенд подключается через специальное устройство (назовём его программатором) к компьютеру. Подключение производится следующим образом:

1. Подключите программатор к стенду.
2. Подключите программатор к порту USB компьютера.
3. Подключите блок питания к стенду.
4. Подключите блок питания к сети 220В.

Переключите стенд в режим «работа с микроконтроллером» DIP-переключателями согласно рисунку 3.

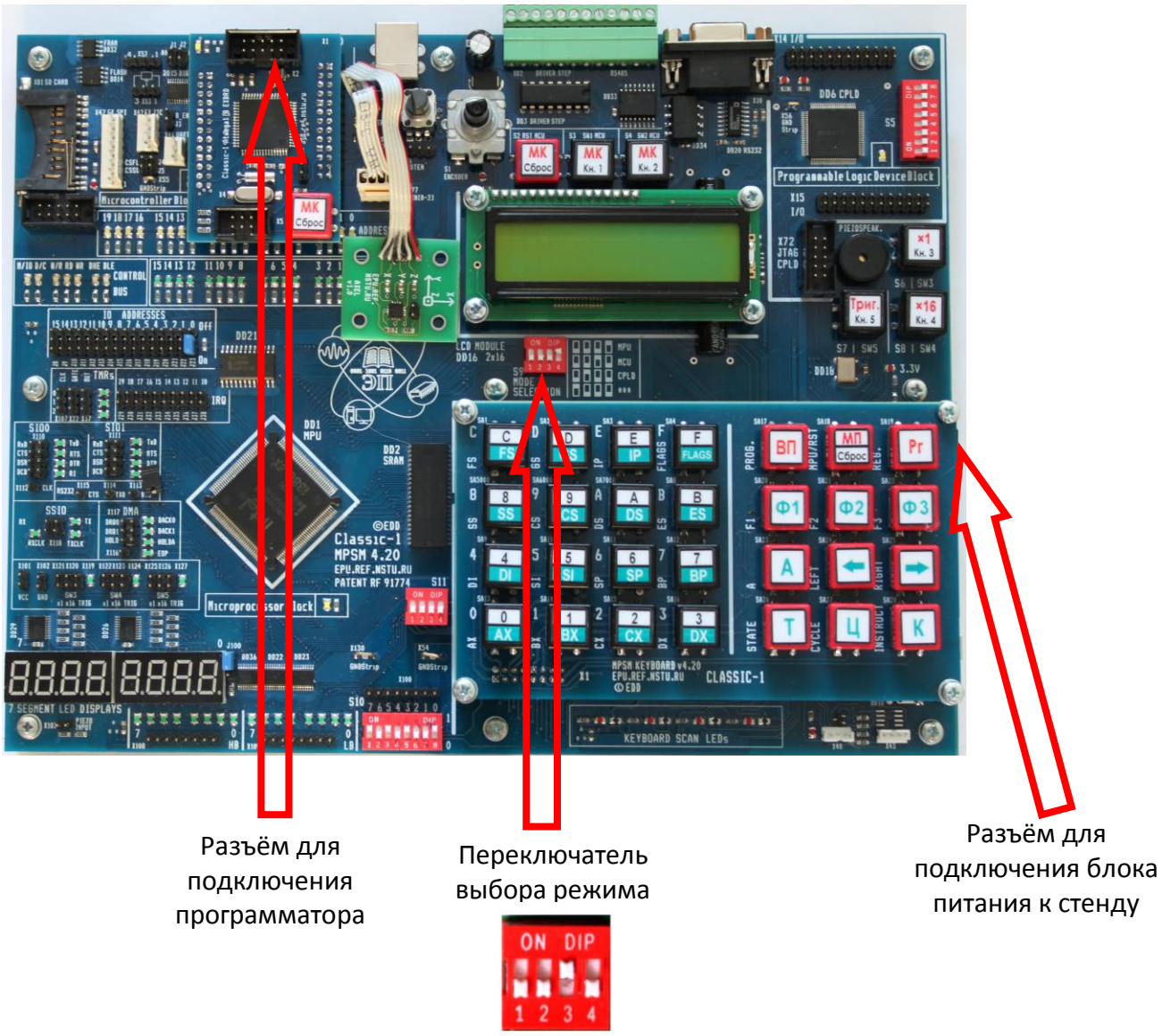


Рисунок 3 — Общий вид стенда

3. ЗАДАНИЕ И ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

3.1 Домашняя подготовка к работе и оформление отчета

При домашней подготовке к лабораторной работе следует повторить необходимый материал, пользуясь конспектом лекций и рекомендованной литературой. В ходе домашней подготовки к каждой работе необходимо придерживаться следующего порядка:

Ознакомьтесь с контрольными вопросами и ответьте на них.

Изучите задание по работе и продумайте пути выполнения всех пунктов. Составьте алгоритмы выполнения и соответствующие им программы на ассемблере.

Сделайте заготовку отчета. В общем виде отчёт должен содержать следующие пункты:

1 Титульный лист.

2 Цель работы.

3 Постановка задачи.

4 Схема устройства, для которого разрабатывается программа.

5 Описание работы программы с наличием карты памяти и необходимых расчётов.

6 Блок-схема.

7 Исходный текст программы.

3.2 Выполнение лабораторной работы

Перед началом каждой лабораторной работы студент предъявляет преподавателю, ведущему занятие, отчет с подготовленным заданием. После

предварительного опроса к выполнению работы допускаются студенты, подготовившие отчет и четко представляющие цель и методику проведения предстоящей работы.

При выполнении лабораторных работ каждый студент должен строго соблюдать правила техники безопасности, а также указания преподавателя.

4 КОМПЛЕКС ЛАБОРАТОРНЫХ РАБОТ

Лабораторная работа 1. «Изучение системы команд и основных принципов программирования микроконтроллеров»

4.1.1 Цель работы

Знакомство со средой разработки для написания и отладки программного обеспечения микроконтроллеров.

Знакомство с основами языка Ассемблер изучаемого микроконтроллера.

Изучение принципов программирования микроконтроллеров.

Изучение основ организации и функционирования ядра микроконтроллера.

4.1.2 Теоретические сведения

Теоретические сведения для выполнения данной работы даны выше. Дополнительные справочные материалы можно получить из описания контроллера.

4.1.3 Задание к работе

Ниже приведен возможный перечень заданий для выполнения. Конкретные параметры (адреса, количество пересылаемых байт и т.д.) получаются у преподавателя индивидуально.

1 Написать программу складывающую два числа, находящихся в памяти данных, и помещающую результат вычисления в ячейку памяти следующую за аргументами.

2 Написать программу складывающую массив чисел находящихся во внутренней памяти данных, и помещающую результат (двухбайтный) после массива.

3 Написать программу складывающую два массива чисел, расположенных в памяти данных, и заносящую результат в третий массив.

4 Написать программу нахождения контрольной суммы содержимого памяти данных.

5 Написать программу, находящую среднее арифметическое содержимого памяти программ.

После реализации каждого пункта и демонстрации его работы преподавателю, студент получает индивидуальное задание на модификацию программ.

4.1.4 Контрольные вопросы к лабораторной работе № 1

1 Опишите способы адресации AVR.

2 Особенности булева процессора AVR.

3 Назначение и примеры использования регистров.

4 Обращение к памяти данных.

5 Обращение к памяти программ.

6 Особенности выполнения команд условных переходов.

7 Особенности команд BREQ, BRNE.

8 Отличие различных команд JMP (IJMP, RJMP, JMP).

9 Команды умножения.

Лабораторная работа 2. «Работа с внешними устройствами»

4.2.1 Цель работы:

1. Изучения портов ввода-вывода микроконтроллеров.
2. Приобретения практических навыков работы с внешними устройствами.

4.2.2 Теоретические сведения

Порты ввода-вывода являются основой любого микроконтроллера, позволяя ему общаться с внешним миром. В ATmega128A реализовано шесть восьмибитных порта ввода-вывода и один шестибитный с возможностью побитной адресации. Познакомимся вкратце с основами организации портов этого контроллера, необходимыми для выполнения данной лабораторной работы.

Во всех современных контроллерах кроме портов ввода-вывода общего назначения есть большое количество аналоговых и цифровых устройств, которым тоже необходимо обмениваться информацией с внешним миром. С целью экономии количества выводов микросхемы все эти устройства используют одни и те же выводы микроконтроллера. Чтобы не было конфликта при обращении нескольких устройств к одному выводу, выводы конфигурируют с помощью регистров конфигурации соответствующей периферии. Поэтому в один момент времени к выводу может обращаться только одно внутреннее устройство контроллера. Если регистры конфигурации выводов не программировать, то в силу вступает принцип «по умолчанию». В большинстве контроллеров, как правило, по умолчанию к выводам подключены цифровые порты ввода-вывода общего назначения настроенные на ввод. Т.е. вводить информацию в цифровом виде можно сразу после старта контроллера.

Для вывода информации уже необходимо провести определенные настройки порта. В частности, необходимо установить соответствующие биты в регистре DDRx. Более подробно о настройки портов ввода-вывода Вы можете узнать из документации на данный микроконтроллер, прилагаемой в комплекте поставки

стенда или скачанной с сайта производителя. В качестве внешних устройств на лабораторном стенде представлены:

1. Светодиоды (8 штук);
2. Восемь семисегментных индикаторов;
3. Динамическая клавиатура, состоящая из 16 клавиш, и клавиатура со сканированием через дешифратор, содержащая 12 клавиш;
4. Переменный резистор;
5. Пьезоизлучатель;
6. Шаговый двигатель;
7. Две кнопки;
8. ЖК-индикатор;
9. Энкодер;
10. Последовательные интерфейсы;
11. DIP-переключатели;

Однако, при использовании данного лабораторного стенда для изучения микроконтроллеров, необходимо учитывать определённые схемотехнические решения подключения внешних устройств.

В данном стенде подключение внешних устройств имеет классическую шинную архитектуру. К порту PA подключена шина данных. К ней также подключены входы данных всех регистров ввода-вывода. А к выходам регистров уже подключены непосредственно внешние устройства. Такая структура выбрана потому, что внешними устройствами могут управлять микроконтроллер, микропроцессор и ПЛИС.

Ниже на рисунке 6 изображена структурная схема подключения внешних устройств. В качестве источника управляющих сигналов, защелкивающих данные в соответствующий регистр, используется порт PC, биты которого подключены к входам записи регистров. Таким образом, для того, чтобы в определенное устройство ввода-вывода вывести информацию, необходимо в порт PA записать выводимый байт, а на определенном выводе порта PC сформировать положительный импульс (сначала вывести «1», потом «0»). Соответствие выводов порта PC регистрам ввода-вывода наглядно видно на схеме (рис. 4).

При этом надо учитывать, что на шине присутствуют не только устройства вывода, но и устройства ввода, которые тоже могут выставлять информацию на шину данных. Чтобы не было конфликтов на шине, необходимо управляющие сигналы устройств ввода удерживать в неактивном состоянии. Одним из таких устройств является ЖК-индикатор. Он управляется тремя старшими битами порта PD. Поэтому, чтобы он не мешал обмену, его надо отключить с помощью «0» на PD7. Другими устройствами ввода являются буферы клавиатуры и DIP-переключателей. Они соответственно управляются выводами портов PC4 и PC5. Причем данные сигналы инверсные. Поэтому, чтобы отключить буферы надо подать на эти порты «1».

Давайте рассмотрим пример простой программы иллюстрирующей всё вышесказанное и выводящей на светодиодную линейку информацию с DIP-переключателей в бесконечном цикле:

```
;************************************
;* Инициализация шины.
;************************************
.include "m128adef.inc"

; Линий порта C.
.equ  L_LEDS         =PC0
.equ  L_DISPDIG      =PC1
.equ  L_DISPSEG      =PC2
.equ  L_KBDOUT       =PC3
.equ  OE_KBDIN       =PC4
.equ  OE_SWITCH      =PC5
; Линий порта D.
.equ  LCD_E          =PD7
.equ  LCD_RW         =PD5
.equ  LCD_RS         =PD6

; Определения регистров.
.def temp  =r16
```

```
.def constFF    =r17
.def const00    =r18

; Текст программы
; Настраиваем порт C.
      ldi  temp, (1<< OE_KBDIN) | (1<< OE_SWITCH)
      out  PORTC, temp     ; Высокий уровень на
      ldi  temp, 0xFF      ; OE_KBDIN и OE_SWITCH.
      out  DDRC, temp      ; PC0-PC7 – вывод.
 ; Настраиваем порт D.
      sbi  DDRD, LCD_E     ; LCD_E – вывод.
 ; Подготавливаем регистры для переключения шины.
      ser constFF
      clr const00
 ; Начинаем работу с устройствами.
 loop:
      cbi PORTC, OE_SWITCH     ;Активируем буфер DIP-переключателей
      in temp, PINA            ;Считываем информацию с шины.
      sbi PORTC, OE_SWITCH     ;Отключаем буфер DIP-переключателей.
      out DDRA, constFF        ;Переключаем шину на вывод.
      out PORTA, temp          ;Выводим данные на шину.
      sbi PORTC, L_LEDS        ;Выводим строб записи на регистр
                               ;светодиодов.
      cbi PORTC, L_LEDS        ;Снимаем строб записи с регистра
                               ;светодиодов.
      out DDRA, const00        ;Переключаем шину на ввод.
      rjmp loop                ;Повторяем снова.
```

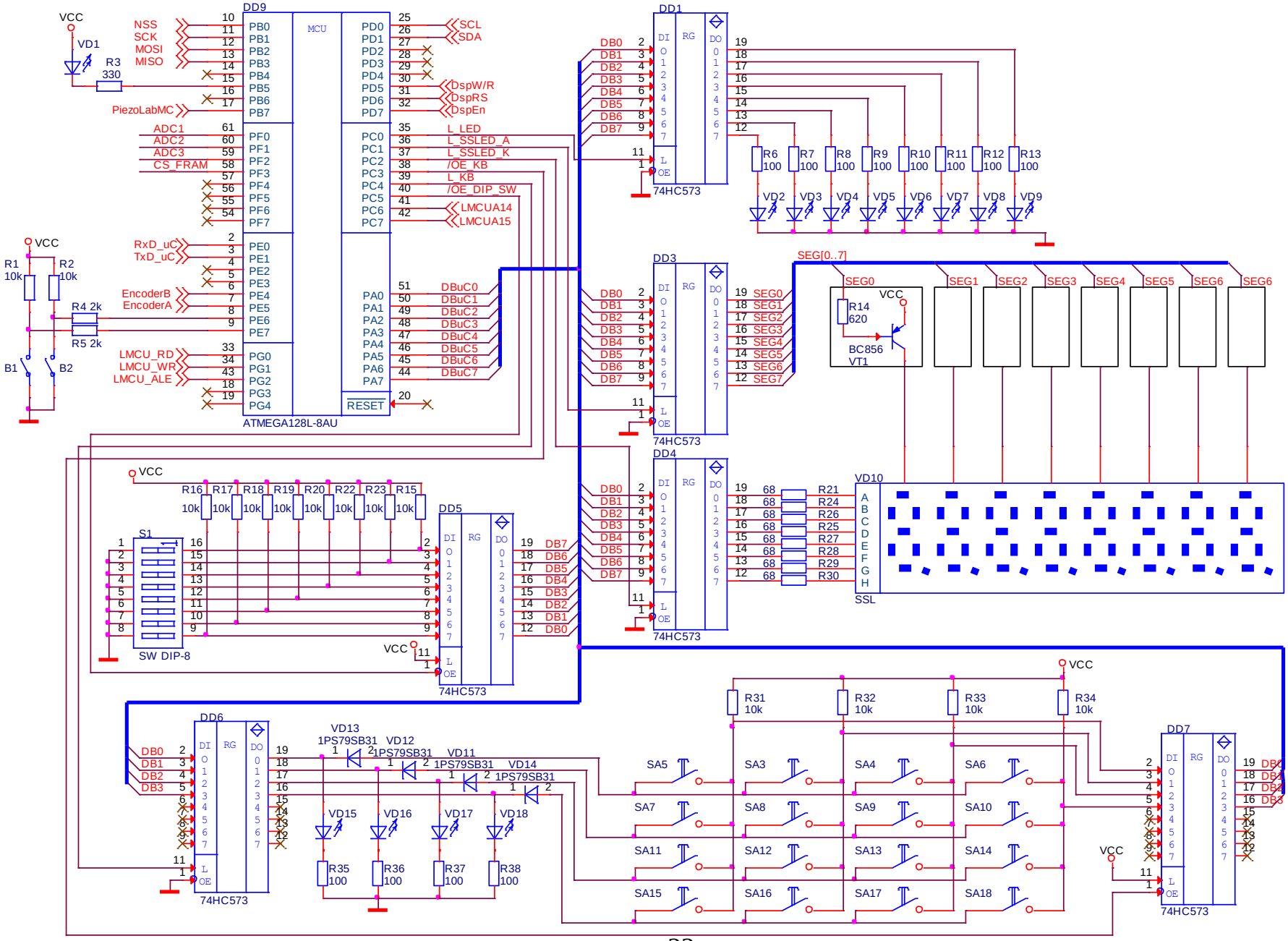


Рисунок 4 — Схема подключения внешних устройств

4.2.3 Задание к работе

Основными устройствами в данной работе являются светодиодная линейка и DIP-переключатели. В качестве дополнительных могут быть задействованы семисегментные индикаторы, две кнопки и пьезоизлучатель.

В ходе лабораторной работы нужно имитировать работу различных цифровых схем.

Возможные варианты заданий для данной лабораторной работы (конкретные детали реализации задания для каждого студента необходимо получить у преподавателя):

1. Мультиплексор 4->1
2. триггер 155ТМ2
3. Двоичный реверсивный счётчик
4. Сдвиговый регистр К155ИР1
5. «Бегущие огни»
6. JK-триггер К155ТВ1
7. Шифратор-дешифратор
8. генератор звука
9. 4 элемента «исключающее ИЛИ»

4.2.4 Контрольные вопросы к лабораторной работе № 2

1. Особенности организации портов AVR.
2. Альтернативные функции портов.
3. Особенности выполнения команд по принципу «чтение-модификация-запись».
4. Работа с памятью данных и программ.
5. Организация выходов и нагрузочная способность портов AVR микроконтроллеров.

6. Модификация отдельных бит при побитном и байтовом обращении к порту.

7. Влияние параметров нагрузки на формирование выходных сигналов.

4.3 Лабораторная работа 3. «Работа с таймерами и системой прерываний»

4.3.1 Цель работы:

1. Изучения таймеров и особенностей их работы.
2. Приобретения навыков написания программ обработки прерываний.

4.3.2 Теоретические сведения

В любом современном микроконтроллере имеются контроллер прерываний и несколько таймеров. Программное формирование временных последовательностей и измерение времени имеет довольно много недостатков, поэтому в реальных системах для этих целей используют аппаратные возможности в виде таймеров-счетчиков. По сути дела таймеры - это устройства, очень напоминающие секундомеры. Их можно запустить в определенный момент времени, их можно приостановить, продолжить счёт далее или сбросить в ноль. Инкрементируются таймеры в каждом машинном цикле или через кратные ему интервалы времени, когда для тактирования таймера используется делитель частоты. Максимальное значение, до которого может считать таймер, определяется его разрядностью, которая обычно составляет 8 или 16 бит. Но кроме этого таймер можно заставить инкрементироваться и импульсами, поступающими на внешний вход контроллера. Таким образом, таймер превращается в счётчик событий, которые в виде импульсов поступают на вход. Отсюда и происходит их название – таймер-счётчик

(Timer/Counter). Более подробное описание реализации таймеров ATmega128A приведено ниже и может использоваться Вами как необходимая и достаточная информация для выполнения данной работы.

4.3.2.1 Таймеры микроконтроллера ATmega128A

МК имеет четыре таймера-счетчика (ТС): по два 8-и и 16-и разрядных. ТС могут использоваться для измерения временных интервалов, подсчета внешних событий, генерации периодических запросов прерываний, в качестве ШИМ генераторов (каналы сравнения) и для регистрации времени возникновения события (каналы захвата). Один из 8-и разрядных ТС имеет возможность асинхронного тактирования от собственного генератора, что в сочетании с часовым кварцевым резонатором позволяет использовать его в качестве часов реального времени. Шестнадцатиразрядные таймеры абсолютно идентичны, тогда как восьмиразрядные имеют некоторые отличия, связанные с тем, что ТС0 может асинхронного тактирования, а ТС2 – нет. ТС0 имеет дополнительные коэффициенты деления тактовой частоты, но не имеет возможности считать внешние события.

Таймер	Разрядность	Число каналов сравнения	Число каналов захвата	Делители тактовой частоты
0	8	1	0	1, 8, 32, 64, 128, 256, 1024
1	16	3	1	1, 8, 64, 256, 1024
2	8	1	0	1, 8, 64, 256, 1024
3	16	3	1	1, 8, 64, 256, 1024

Когда ТС функционирует как таймер, регистры ТС инкрементируются по каждому такту внутреннего сигнала тактирования ТС. Частота внутреннего сигнала тактирования ТС равна системной тактовой частоте, деленной на один из возможных делителей этого ТС. Делители выбираются в зависимости от требуемых интервалов времени. У каждого счетчика свой независимый делитель системной частоты.

Когда ТС функционирует как счетчик, регистры ТС инкрементируются под воздействием перехода из 1 в 0 или из 0 в 1 внешнего сигнала на выбранном входном выводе T1, T2 или T3. Могут подсчитываться импульсы с частотой до 1/2 системной тактовой частоты. Входной сигнал не обязательно должен быть периодическим, но для его гарантированного прочтения он должен удерживаться на заданном уровне как минимум в течение одного полного системного такта.

4.3.2.2 Таймер-счетчик TC0

Для доступа к TC0 и управления ими используются регистры ввода-вывода.

ТС реализован в виде 8-разрядных регистров счетчика TCNT0 и регистра сравнения OCR0. Регистр управления TCCR0 используется для задания режима работы, которых у TC0 четыре, выбора источника тактирования (делителя), задания режима работы выхода OCR0 и принудительного совпадения сравнения модуля генерации выходного сигнала. С помощью регистра TIFR можно узнать состояние флагов прерываний, которых у TC0 два: переполнение таймера и совпадения. Прерывания могут быть запрещены или разрешены индивидуально с помощью регистра TIMSK. Кроме этого для управления TC0 есть регистр асинхронного состояния ASSR и два бита в регистре специального назначения SFIOR. Далее кратко рассмотрим регистры управления и состояния. Более полная информация может быть получена из руководства, предоставляемого производителем МК.

4.3.2.3 Регистр управления TCCR0

Как было отмечено выше, регистр TCCR0 предназначен для управления TC0.

Бит	0	1	2	3	4	5	6	7
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Чтение/запись	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Нач. значение	0	0	0	0	0	0	0	0

FOC0 – бит только для записи. При работе TC0 в режимах, отличных от ШИМ, используется для принудительного перевода генератора выходного сигнала в состояние, которое возникает при совпадении регистра счетчика TCNT0 и регистра сравнения OCR0.

WGM00, WGM01 – биты задания режима работы TC0.

№	WGM01	WGM00	Режим	Считает до	Обновление OCR0	Установка переполнения
0	0	0	Обычный	0xFF	Сразу	В максимуме
1	0	1	ШИМ с точной фазовой	0xFF	В максимуме	В минимуме
2	1	0	Автоматическая перегрузка	OCR0	Сразу	В максимуме
3	1	1	Быстрый ШИМ	0xFF	В минимуме	В максимуме

COM00, COM01 – управление режимом выхода OCR0. Поведение этого выхода зависит от режима работы TC0.

COM01	COM00	Не ШИМ	ШИМ с точной фазой	Быстрый ШИМ
0	0	Обычная работа порта, OCR0 отсоединен	Обычная работа порта, OCR0 отсоединен	Обычная работа порта, OCR0 отсоединен
0	1	Изменение состояния выхода OCR0 при совпадении	Зарезервирован	Зарезервирован
1	0	Вывод низкого уровня при совпадении	Сброс при совпадении, когда счетчик инкрементируется, установка при совпадении, когда счетчик декрементируется	Сброс при совпадении, установка в максимуме
1	1	Вывод высокого уровня при совпадении	Установка при совпадении, когда счетчик инкрементируется, сброс при совпадении, когда	Установка при совпадении, сброс в максимуме

			счетчик декрементируется	

CS02-CS00 – биты выбора источника тактирования. С помощью этих битов можно остановить TC0 или запустить его с делением тактовой частоты на заданный коэффициент.

CS02	CS01	CS00	Описание
0	0	0	TC0 остановлен
0	0	1	Тактирование с частотой ядра
0	1	0	Тактирование с частотой ядра, деленной на 8
0	1	1	Тактирование с частотой ядра, деленной на 32
1	0	0	Тактирование с частотой ядра, деленной на 64
1	0	1	Тактирование с частотой ядра, деленной на 128
1	1	0	Тактирование с частотой ядра, деленной на 256
1	1	1	Тактирование с частотой ядра, деленной на 1024

4.3.2.4 Регистр флагов прерываний TIFR

Бит	0	1	2	3	4	5	6	7
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Чтение/запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Нач. значение	0	0	0	0	0	0	0	0

TC0 использует два бита в регистре TIFR.

OCF0 – флаг выставляется при совпадении значений регистров TCNT0 и OCR0.

TOV0 – флаг переполнения TC0.

Флаги сбрасываются при переходе в прерывание или записью единицы в соответствующий бит регистра.

4.3.2.5 Регистр масок TIMSK

Бит	0	1	2	3	4	5	6	7
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
Чтение/запись	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Нач. значение	0	0	0	0	0	0	0	0

Аналогично регистру TIFR, в регистре масок TIMSK TC0 используется два младших бита.

OCIE0 – установкой единицы разрешается прерывание при переполнении TC0.

TOIE0 – установкой единицы разрешается прерывание при совпадении значений регистров TCNT0 и OCR0.

4.3.2.6 Таймер-счетчик TC2

TC2 аналогичен TC2 за исключением значений битов CS регистра TCCR2.

CS22	CS21	CS20	Описание
0	0	0	TC2 остановлен
0	0	1	Тактирование с частотой ядра
0	1	0	Тактирование с частотой ядра, деленной на 8
0	1	1	Тактирование с частотой ядра, деленной на 64
1	0	0	Тактирование с частотой ядра, деленной на 256
1	0	1	Тактирование с частотой ядра, деленной на 1024
1	1	0	Тактирование по спаду на входе T2
1	1	1	Тактирование по фронту на входе T2

Как видно из таблицы, у TC2 имеется возможность подсчета внешних событий подаваемых на вход T2 тогда как у TC0 есть дополнительные делители системной частоты.

4.3.2.7 Таймеры-счетчики TC1 и TC3

TC1 и TC3 являются 16-битными, у них по три канала сравнения и один канал захвата, у них не по одному, а по три регистра управления, пятнадцать режимов работы, большее количество прерываний. Для получения полной информации о работе этих таймеров следует обратиться к документации на МК ATmega128A.

4.3.2.8 Система прерываний

МК ATmega128A имеет в своем арсенале 35 прерываний, из которых внешние прерывания INT0 – INT7 могут быть вызваны либо уровнем, либо изменением состояния сигнала на входах МК в зависимости от значений управляющих бит в регистрах ECRA и ECRB. Флаги активности прерываний содержатся в регистре EIFR, разрешаются прерывания установкой соответствующего бита регистра EIMSK в единицу.

4.3.3 Возможные варианты заданий к работе:

1 Частотомер(измерение частоты входных импульсов).

2 Часы (вывод секунд, минут часов в коде BCD на порты).

3 ШИМ-регулятор.

4 Управляемый генератор.

5 Календарь.

6 Трехфазный генератор.

7 Программа подсчёта числа импульсов между двумя событиями.

4.3.4 Контрольные вопросы к лабораторной работе № 3:

1 Сравнение режимов работы таймеров.

2 Временные характеристики таймеров.

3 Работа с таймерами с помощью прерываний.

4 Последовательность обслуживания прерываний.

5 Размещение подпрограмм обслуживания прерываний в памяти в программах на ассемблере. Директива ORG.

6 Время реакции на запрос прерывания.

7 Особенности обслуживание подпрограмм обработки прерываний в отличии от обычных подпрограмм.

8 Управление работой таймера посредством внешних сигналов.

9 Тактирование последовательного порта с помощью таймера.

4.4 Лабораторная работа 4. «Работа с клавиатурой и семисегментными индикаторами»

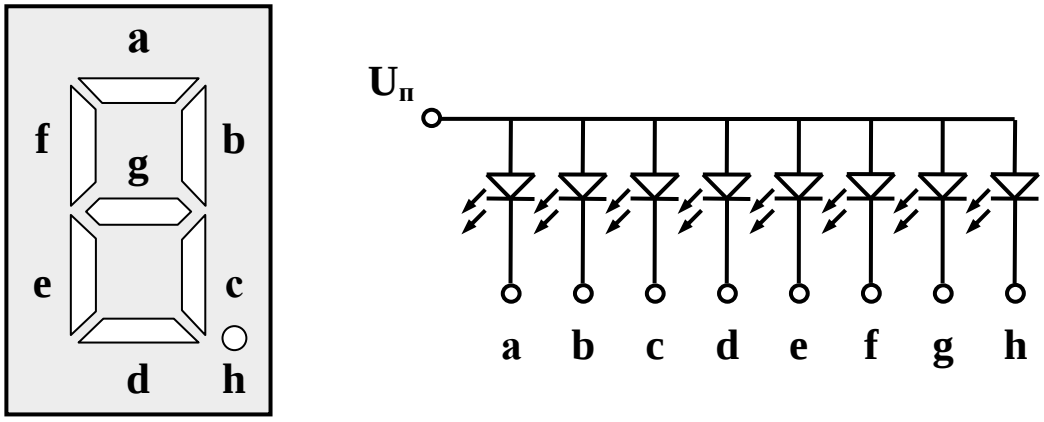
4.4.1 Цель работы:

Изучение принципов подключения и работы дисплея и клавиатуры с динамическим сканированием.

4.4.2 Теоретические сведения

4.4.2.1 Схема подключения 7-сегментного индикатора

В качестве устройства вывода информации, удобного для восприятия, часто используется 7-сегментный индикатор. Рассмотрим подключение двухразрядного 7-сегментного индикатора. Каждый индикатор представляет собой восемь светодиодов с общим анодом: семь светодиодов для отображения сегментов цифр, а восьмой светодиод отображает десятичную точку. Внешний вид и схема 7-сегментной светодиодной матрицы представлена на рисунке 5. Индикатор может отображать цифры от 0 до 9, а также некоторые буквы. Буквенное обозначение сегментов и схема светодиодной матрицы представлены на рисунке 5.



а) б)

Рисунок 5 — Внешний вид (а) и схема 7-сегмнетной светодиодной матрицы (б)

На рисунке 6 показан пример подключения семисегментных индикаторов.

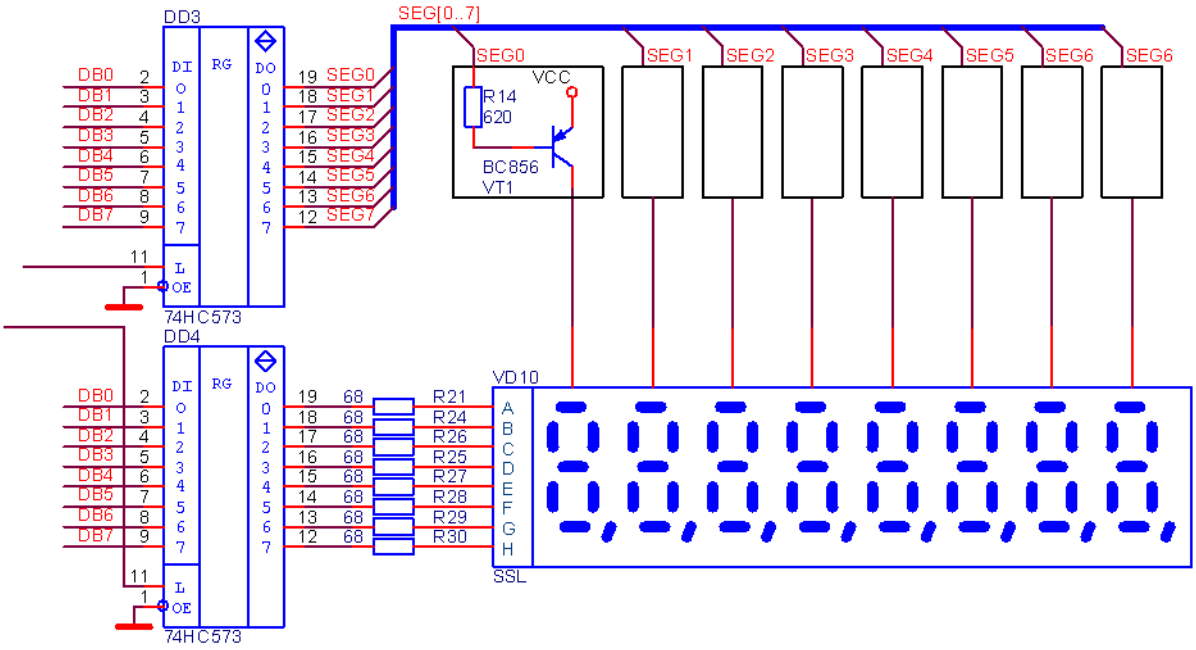


Рисунок 6 — Схема подключения 7-сегментного индикатора

Общие аноды каждого разряда подключаются к выходам регистра разрядов. А выводы сегментов всех индикаторов соединённые параллельно подключаются к выходам регистра сегментов. Если вывести разрешающий сигнал на один из выходов регистра разрядов, то на аноды светодиодов соответствующего индикатора будет подано напряжение питания. Если теперь на регистр сегментов вывести

определенную комбинацию 0 и 1, то в этом разряде будет светится определенный символ. Теперь делаем определенную задержку, и повторяем описанный алгоритм для другого разряда, сменяя комбинацию 0 и 1 на регистре сегментов на соответствующую тому символу, который должен отображаться в этом разряде. Пройдя до последнего разряда, начинаем эту процедуру сначала. Таким образом, в один момент времени на индикаторах будет отображаться только один разряд. Но при достаточно большой скорости сканирования глаза не будут замечать мерцания и будет воспринимать изображение как статическое. Частота, при которой изображение перестает «мерцать», равна 50 Герцам. Поэтому сканирование всех индикаторов должно происходить за время менее 20 мс.

Теперь рассмотрим принцип динамического сканирования клавиатуры.

Пример подключения клавиатуры по этому принципу показан на рисунке 7. Кнопки в такой клавиатуре организованы в виде матрицы. Одним выводом кнопки подключены к «столбцам», другим выводом к «строкам». «Строки» подключаются к выходам регистра сканирования. «Столбцы» подключаются ко входам регистра «возвратных линий». На одну из строк выставляется 1. После чего считывается состояние регистра возвратных линий. Если во всех столбцах считаны 0, то мы смещаем выставленную 1 на следующую строку, и повторяем опрос столбцов. Дойдя до последней строки, начинаем всё сначала. Если при очередном считывании, мы обнаружим на одном из столбцов 1, то значит мы зарегистрировали факт нажатия клавиши. Произведение номера сканируемой строки на номер столбца, в котором обнаружена 1, и будет номером нажатой клавиши. Сканирование всей клавиатуры необходимо осуществлять с периодом примерно 5 мс., чтобы не «проморгать» факт нажатия клавиши. Также необходимо помнить о «дребезге контактов».

4.4.3 Возможные варианты заданий к работе:

1. Часы.

2. Конвертер DEC-> HEX.
3. Бегущая цифровая строка.
4. Простой калькулятор.
5. Таймер.
6. Измеритель длительности импульса.

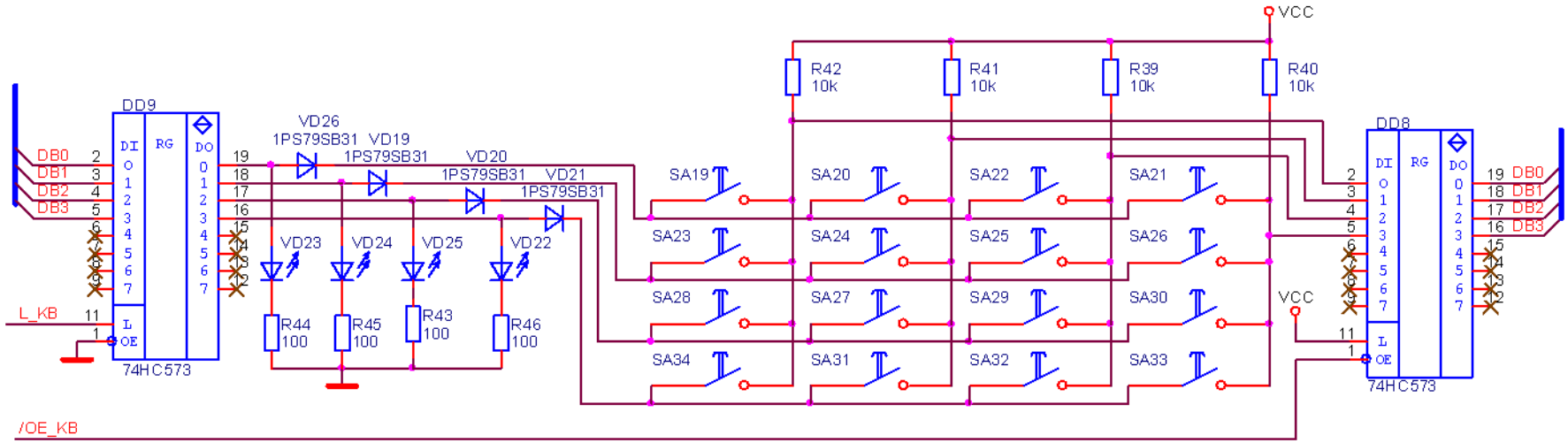


Рисунок 7 — Схема подключения клавиатуры

4.4.4 Контрольные вопросы к лабораторной работе № 4

1. Способы вычисления номера нажатой клавиши.
2. Способы подавления дребезга кнопок.
3. Требования к временным интервалам динамического сканирования клавиатуры и дисплея.
4. Использование табличного вычисления функций при работе с клавиатурой и семисегментным дисплеем.
5. Различные способы преобразования BIN->BCD.
6. Вывод на индикаторы вещественных чисел.
7. Программная регулировка яркости семисегментных индикаторов.

Приложение А. Система команд МК ATmega128A

Мнемокод	Операнды	Описание	Действие	Флаги	Кол. машинных циклов
Арифметические и логические инструкции					
ADD1	Rd, Rr	Сложить два регистра	Rd <— Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Сложить два регистра с переносом	Rd <— Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Сложить слово с константой	Rdh:Rdl <— Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Вычесть два регистра	Rd <— Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Вычесть константу из регистра	Rd <— Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Вычесть два регистра с учетом переноса	Rd <— Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Вычесть константу из регистра с учетом переноса	Rd <— Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Вычесть константу из слова	Rdh:Rdl <— Rdh:Rdl-K	Z,C,N,V,S	2
AND	Rd, Rr	Логическое И между регистрами	Rd <— Rd · Rr	Z,N,V	1
ANDI	Rd, K	Логическое И между регистром и константой	Rd <— Rd · K	Z,N,V	1
OR	Rd, Rr	Логическое ИЛИ между регистрами	Rd <— Rd v Rr	Z,N,V	1
ORI	Rd, K	Логическое ИЛИ между регистром и константой	Rd <— Rd v K	Z,N,V	1
EOR	Rd, Rr	Искл. ИЛИ между регистрами	Rd <— Rd ⊕ Rr	Z,N,V	1
COM	Rd	Дополнение до 0b11111111 ($FF), инверсия	Rd <— $FF - Rd	Z,C,N,V	1
NEG	Rd	Дополнение до 0b00000000 ($00)	Rd <— $00 - Rd	Z,C,N,V,H	1
SBR	Rd, K	Установка бит (бита) в регистре	Rd <— Rd v K	Z,N,V	1
CBR	Rd, K	Сброс бит (бита) в регистре	Rd <— Rd · ($FF - K)	Z,N,V	1
INC	Rd	Инкремент	Rd <— Rd + 1	Z,N,V	1
DEC	Rd	Декремент	Rd <— Rd - 1	Z,N,V	1
TST	Rd	Проверка на ноль или минус	Rd <— Rd . Rd	Z,N,V	1
CLR	Rd	Сброс регистра	Rd <— Rd ⊕ Rd	Z,N,V	1
SER	Rd	Установка регистра	Rd <— $FF	Нет	1
MUL	Rd, Rr	Умножение без знака	R1:R0 <— RdxRr	Z, C	2
MULS	Rd, Rr	Умножение со знаком	R1:R0 <— RdxRr	Z, C	2
MULSU	Rd, Rr	Умножение знакового с беззнаковым числом	R1:R0 <— Rd x Rr	Z, C	2
FMUL	Rd. Rr	Дробное умножение без знака	R1:R0 <— (RdxRr) << 1	Z, C	2
FMULS	Rd, Rr	Дробное умножение со знаком	R1:R0 <— (RdxRr) << 1	Z, C	2
FMULSU	Rd, Rr	Дробное умножение знакового с беззнаковым числом	R1:R0 <— (RdxRr) << 1	Z, C	2
Инструкции перехода					
RJMP	k	Относительный переход	PC <— PC + k +1	Нет	2
IJMP		Косвенный переход по указателю (Z)	PC <— Z	Нет	2
JMP	k	Безусловный переход	PC <— k	Нет	3
RCALL	k	Относительный вызов процедуры	PC <— PC + k + 1	Нет	3
ICALL		Косвенный вызов процедуры по указателю (Z)	PC <— Z	Нет	3

CALL	k	Безусловный вызов процедуры	PC <— k	Нет	4
RET		Возврат из подпрограммы	PC <— STACK	Нет	4
RETI		Возврат из прерывания	PC <— STACK	I	4
CPSE	Rd,Rr	Сравнение и пропуск, если равно if (Rd = Rr)	PC <— PC + 2 или 3	Нет	1/2/3
CP	Rd,Rr	Сравнение	Rd-Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Сравнение с учетом переноса	Rd - Rr-C	Z, N,V,C,H	1
CPI	Rd,K	Сравнение регистра с константой	Rd-K	Z, N,V,C,H	1
SBRC	Rr,b	Пропуск, если бит в регистре сброшен	if (Rr(b)=O)PC <— PC + 2 или 3	Нет	1 /2/3
SBRS	Rr, b	Пропуск, если бит в регистре установлен	if (Rr(b)=1)PC <— PC+ 2 или 3	Нет	1/2/3
SBIC	P, b	Пропуск, если бит в регистре ввода-вывода сброшен	if (P(b)=O)PC <— PC + 2 или 3	Нет	1 /2/3
SBIS	P, b	Пропуск, если бит в регистре ввода-вывода установлен	if (P(b)=1)PC <— PC + 2 или 3	Нет	1 /2/3
BRBS	s, k	Переход, если флаг состояния установлен	if (SREG(s) = 1) then PC <— PC+k + 1	Нет	1/2
BRBC	s, k	Переход, если флаг состояния сброшен	if (SREG(s) = 0) then PC <— PC+k + 1	Нет	1 /2
BREQ	k	Переход, если равно	if (Z = 1) then PC <— PC + k + 1	Нет	1 /2
BRNE	k	Переход, если не равно	if (Z = 0) then PC <— PC + k + 1	Нет	1 /2
BRCS	k	Переход, если перенос установлен	if (C = 1)then PC <— PC + k+ 1	Нет	1 /2
BRCC	k	Переход, если перенос сброшен	if (C = 0) then PC <— PC + k + 1	Нет	1 /2
BRSH	k	Переход, если больше или равно	if (C = 0) then PC <— PC + k + 1	Нет	1 /2
BRLO	k	Переход, если меньше	if (C = 1) then PC <— PC + k+ 1	Нет	1 /2
BRMI	k	Переход, если минус	if (N = 1)then PC <— PC + k + 1	Нет	1 /2
BRPL	k	Переход, если плюс	if (N = 0) then PC <— PC + k + 1	Нет	1 /2
BRGE	k	Переход, если больше или равно с учетом знака	if (N e V= 0) then PC <— PC + k + 1	Нет	1 /2
BRLT	k	Переход, если меньше нуля с учетом знака	if (N e V= 1) then PC <— PC + k + 1	Нет	1 /2
BRHS	k	Переход, если флаг H установлен	if (H = 1)then PC <— PC + k + 1	Нет	1 /2
BRHC	k	Переход, если флаг H сброшен	if (H = 0) then PC <— PC + k + 1	Нет	1 /2
BRTS	k	Переход, если флаг T установлен	if (T = 1)then PC <— PC + k +1	Нет	1 /2
BRTC	k	Переход, если флаг T сброшен	if (T = 0) then PC <— PC + k + 1	Нет	1 /2
BRVS	k	Переход, если флаг V установлен	if (V = 1)then PC <— PC + k+ 1	Нет	1 /2
BRVC	k	Переход, если флаг V сброшен	if (V = 0) then PC <— PC + k + 1	Нет	1 /2

BRIE	k	Переход, если прерывания разрешены	if (I = 1)then PC <— PC + k + 1	Нет	1 /2
BRID	k	Переход, если прерывания запрещены	if (I = 0) then PC <— PC + k + 1	Нет	1 /2
Инструкции передачи данных					
MOV	Rd, Rr	Запись из регистра в регистр	Rd <— Rr	Нет	1
MOVW	Rd, Rr	Перезапись слова между регистрами	Rd+1:Rd <— Rr+1:Rr	Нет	1
LDI	Rd, K	Запись константы в регистр	Rd <— K	Нет	1
LD	Rd, X	Косвенное считывание из памяти в регистр	Rd <— (X)	Нет	2
LD	Rd, X+	Косвенное считывание из памяти в регистр и инкр.	Rd <— (X), X <— X + 1	Нет	2
LD	Rd,-X	Предварительный декремент, а затем косвенное считывание из памяти в регистр	X <— X - 1, Rd <— (X)	Нет	2
LD	Rd,Y	Косвенное считывание из памяти в регистр	Rd <— (Y)	Нет	2
LD	Rd, Y+	Косвенное считывание из памяти в регистр и инкр.	Rd <— (Y), Y <— Y + 1	Нет	2
LD	Rd,-Y	Предварительный декремент, а затем косвенное считывание из памяти в регистр	Y <— Y - 1, Rd <— (Y)	Нет	2
LDD	Rd,Y+q	Косвенное считывание из памяти в регистр со смещением	Rd <— (Y + q)	Нет	2
LD	Rd, Z	Косвенное считывание из памяти в регистр	Rd <— (Z)	Нет	2
LD	Rd, Z+	Косвенное считывание из памяти в регистр и инкр.	Rd <— (Z),Z <— Z+1	Нет	2
LD	Rd.-Z	Предварительный декремент, а затем косвенное считывание из памяти в регистр	Z <— Z - 1, Rd <— (Z)	Нет	2
LDD	Rd, Z+q	Косвенное считывание из памяти в регистр со смещением	Rd <— (Z + q)	Нет	2
LDS	Rd, k	Непосредственное чтение из ОЗУ в регистр	Rd <— (k)	Нет	2
ST	X, Rr	Косвенная запись	(X) <— Rr	Нет	2
ST	X+, Rr	Косвенная запись и послед. инкремент	(X) <— Rr, X <— X + 1	Нет	2
ST	-X, Rr	Предв. декремент и косвенная запись	X <— X - 1, (X) <— Rr	Нет	2
ST	Y, Rr	Косвенная запись	(Y) <— Rr	Нет	2
ST	Y+, Rr	Косвенная запись и послед. инкремент	(Y) <— Rr, Y <— Y + 1	Нет	2
ST	-Y, Rr	Предв. декремент и косвенная запись	Y <— Y - 1, (Y) <— Rr	Нет	2
STD	Y+q,Rr	Косвенная запись со смещением	(Y + q) <— Rr	Нет	2
ST	Z, Rr	Косвенная запись	(Z) <— Rr	Нет	2
ST	Z+, Rr	Косвенная запись и послед. инкремент	(Z) <— Rr, Z <— Z + 1	Нет	2
ST	-Z. Rr	Предв. декремент и косвенная запись	Z <— Z - 1, (Z) <— Rr	Нет	2
STD	Z+q,Rr	Косвенная запись со смещением	(Z + q) <— Rr	Нет	2
STS	k, Rr	Непосредственная запись в ОЗУ	(k) <— Rr	Нет	2
LPM		Чтение из памяти программ	R0 <— (Z)	Нет	3

LPM	Rd, Z	Чтение из памяти программ	Rd <— (Z)	Нет	3
LPM	Rd, Z+	Чтение из памяти программ и последующий инкремент	Rd <— (Z), Z <— Z+1	Нет	3
ELPM		Расширенное чтение из памяти программ	R0 <— (RAMPZ:Z)	Нет	3
ELPM	Rd, Z	Расширенное чтение из памяти программ	Rd <— (RAMPZ:Z)	Нет	3
ELPM	Rd, Z+	Расширенное чтение из памяти программ и последующие инкремент	Rd <— (RAMPZ:Z), RAMPZ:Z <— RAMPZ:Z+1	Нет	3
SPM		Запись в память программ	(Z) <— R1 :R0	Нет	-
IN	Rd, P	Считывание из порта ввода-вывода в регистр	Rd <— P	Нет	1
OUT	P, Rr	Запись из регистра в порт ввода-вывода	P <— Rd	Нет	1
PUSH	Rr	Помещение содержимого регистра в стек	STACK <— Rr	Нет	2
POP	Rd	Извлечение из стека в регистр	Rd <— STACK	Нет	2
Битовые инструкции и инструкции тестирования бит					
SBI	P,b	Установка бита в регистре ввода-вывода	I/O(P,b) <— 1	Нет	2
CBI	P,b	Сброс бита в регистре ввода-вывода	I/O(P,b) <— 0	Нет	2
LSL	Rd	Логический сдвиг влево	Rd(n+1) <— Rd(n), Rd(0) <— 0	Z,C,N,V	1
LSR	Rd	Логический сдвиг вправо	Rd(n) <— Rd(n+1), Rd(7) <— 0	Z,C,N,V	1
ROL	Rd	Вращение влево через перенос	Rd(0)<— C,Rd(n+1)<— Rd(n),C <— Rd(7)	Z,C,N,V	1
ROR	Rd	Вращение вправо через перенос	Rd(7) <— C,Rd(n)<— Rd(n+1),C <— Rd(0)	Z,C,N,V	1
ASR	Rd	Арифметический сдвиг вправо	Rd(n)<— Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Обмен тетрадами	Rd(3..0) <— Rd(7..4), Rd(7..4) <— Rd(3..0)	Нет	1
BSET	s	Установка флага регистра SREG	SREG(s) <— 1	SREG(s)	1
BCLR	s	Сброс флага регистра SREG	SREG(s) <— 0	SREG(s)	1
BST	Rr, b	Запись бита регистра в T	T <— Rr(b)	T	1
BLD	Rd, b	Чтение из T в бит регистра	Rd(b) <— T	Нет	1
SEC		Установка переноса	C <— 1	C	1
CLC		Сброс переноса	C <— 0	C	1
SEN		Установка флага N	N <— 1	N	1
CLN		Сброс флага N	N <— 0	N	1
SEZ		Установка флага нуля Z	Z <— 1	Z	1
CLZ		Сброс флага нуля Z	Z <— 0	Z	1
SEI		Общее разрешение прерываний	I<— 1	I	1
CLI		Общий запрет прерываний	I <— 0	I	1
SES		Установка флага S	S <— 1	S	1
CLS		Сброс флага S	S <— 0	S	1
SEV		Установка флага V в регистре SREG	V <— 1	V	1
CLV		Сброс флага V в регистре SREG	V <— 0	V	1
SET		Установка флага T в регистре SREG	T <— 1	T	1

CLT		Сброс флага T в регистре SREG	T <— 0	T	1
SEH		Установка флага H в регистре SREG	H <— 1	H	1
CLH		Сброс флага H в регистре SREG	H <— 0	H	1
Инструкции управления микроконтроллером					
NOP		Нет операции		Нет	1
SLEEP		Перевод в режим сна	(см. подробное описание режима сна)	Нет	1
WDR		Сброс сторожевого таймера	(см. подробное описание сторожевого таймера)	Нет	1
BREAK		Прерывание	Только для встроенной отладки	Нет	-