

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ**

Ю.В. Китаев

**ПРОГРАММИРОВАНИЕ МК НА
АССЕМБЛЕРЕ ASM-51 и AVR Pascal**

Учебное пособие



Санкт-Петербург

2011

Китаев Ю.В. “Программирование МК на ассемблере ASM-51 и AVR Pascal”. Учебное пособие: СПб: СПбГУ ИТМО, 2011. 90 с.

Приведены лабораторные работы по проектированию и программированию некоторых типовых устройств ввода-вывода для МК семейства MCS-51 и Atmel AVR.

Для студентов, обучающихся по направлениям “Приборостроение”, “Телекоммуникации” и “Оптотехника”: 210401 Физика и технология элементов систем оптической связи, 200600.62 Фотоника и оптоинформатика, 20020104 Лазерная технология

Рекомендовано к печати Советом ИФФ от 05 октября 2010г., протокол №2.



В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена Программа развития государственного образовательного учреждения высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» на 2009–2018 годы.

© Санкт-Петербургский государственный университет информационных технологий, механики и оптики, 2011

© Ю.В. Китаев, 2011

ОГЛАВЛЕНИЕ

ЛАБОРАТОРНАЯ РАБОТА № 18 “РАЗРАБОТКА ПРОСТОГО ПРИБОРА НА МК АТМЕГА128”	5
1.1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ	5
1.2 ЦЕЛЬ РАБОТЫ	5
1.3 ЗАДАЧИ	5
1.4 ОБЩИЕ СВЕДЕНИЯ О МИКРОКОНТРОЛЛЕРЕ	5
ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ.....	6
2.1 СОЗДАНИЕ ШАБЛОНА ПРОГРАММЫ.....	6
2.2 РАЗРАБОТКА И ОТЛАДКА ПРОГРАММЫ.....	8
2.2.1 НАСТРОЙКА ПОРТОВ.....	9
2.2.2 ЗАГРУЗКА И ВЫПОЛНЕНИЕ ПРОГРАММЫ.....	12
2.2.3 ПРОГРАММИРОВАНИЕ ТАЙМЕРА.....	13
2.2.4 ПРОГРАММИРОВАНИЕ 8-МИ СЕГМЕНТНОГО ИНДИКАТОРА.....	20
2.2.5 ВСПОМОГАТЕЛЬНЫЕ ПОДПРОГРАММЫ.....	24
2.2.6 ПРОГРАММИРОВАНИЕ АЦП.....	27
2.2.7 ПРОГРАММИРОВАНИЕ EEPROM.....	34
2.2.8 САМОСТОЯТЕЛЬНОЕ ЗАДАНИЕ.....	35
ПРИЛОЖЕНИЯ	35
Приложение 1. Синхронизация состояния линии порта PORTE.7. ...	35
Приложение 2. Подпрограммы для работы с клавиатурой	36
Приложение 3. ШИНА VS ПОРТ	38
ВОПРОСЫ ДЛЯ ЗАЩИТЫ РАБОТЫ И ЭКЗАМЕНА	39
ВАРИАНТЫ ЗАДАНИЙ.....	40
ЛАБОРАТОРНАЯ РАБОТА № 32 “ЖК ДИСПЛЕЙ”	42
ТЕХНИЧЕСКОЕ ЗАДАНИЕ	45
РАСЧЕТ АДРЕСОВ РЕГИСТРА УПРАВЛЕНИЯ И РЕГИСТРА ДАННЫХ ЖК СИМВОЛЬНОГО ДИСПЛЕЯ	46
РАСЧЕТ АДРЕСА РЕГИСТРА RG2.	47
РАЗРАБОТКА ПРОГРАММЫ	48
I) СОЗДАНИЕ ШАБЛОНА ПРОГРАММЫ НА АССЕМБЛЕРЕ.....	48
II). РАЗРАБОТКА ПРОГРАММЫ УПРАВЛЕНИЯ ЖК ДИСПЛЕЕМ И ЦАП-АЦП. .	51
II-1). РАЗРАБОТКА ПРОГРАММЫ, ОТОБРАЖАЮЩЕЙ НА ЖК ДИСПЛЕЕ ОДИН СИМВОЛ.....	56
III) ПРОБНЫЙ ЗАПУСК ПРОГРАММЫ.	62
IV). РАЗРАБОТКА ШАБЛОНА ПРОГРАММЫ С МЕНЮ ВЫБОРА ДЕЙСТВИЙ	64
V). РАЗРАБОТКА МОДУЛЯ ПРОСТЕЙШЕГО ЦИФРОВОГО ВОЛЬТМЕТРА С ИСПОЛЬЗОВАНИЕМ ЦАП И АЦП.....	69

V-1). ВСТРОЕННЫЕ ЦАП и АЦП	71
V-2). ПРОГРАММИРОВАНИЕ ВСТРОЕННЫХ ЦАП и АЦП	73
V-3) ОКОНЧАТЕЛЬНАЯ ПРОВЕРКА РАБОТЫ ПРОГРАММЫ	80
ПРИЛОЖЕНИЕ №1. НЕКОТОРЫЕ КОМАНДЫ И ДИРЕКТИВЫ АССЕМБЛЕРА MCS-51.....	82
ВАРИАНТЫ ТЕХНИЧЕСКОГО ЗАДАНИЯ	83
ВОПРОСЫ ДЛЯ ЗАЩИТЫ И ЭКЗАМЕНА	83
ЛИТЕРАТУРА	86

ЛАБОРАТОРНАЯ РАБОТА № 18 “РАЗРАБОТКА ПРОСТОГО ПРИБОРА НА МК ATMega128”

1.1 Техническое задание

- Разработать систему слежения за углом поворота призмы оптического прибора, с записью данных в ПЗУ (EEPROM)
- Вариант – система слежения за углом поворота штурвала с записью данных в “черный ящик” (EEPROM)
- Вариант – измерение напряжения от датчика температуры с записью данных во флэш-накопитель и подачей сигнала тревоги при превышении допустимого значения.
- и т. д.

1.2 Цель работы

Изучить:

- работу с 8-ми сегментным дисплеем,
- подачу звуковых сигналов,
- работу с встроенным АЦП,
- асинхронную работу таймера0 от кварцевого резонатора 32768Гц,
- программирование клавиатуры,
- запись/чтение в/из EEPROM

1.3 Задачи

- Сформировать заданный временной интервал с подачей короткого звукового сигнала и переключением светодиода.
- Обеспечить непрерывное измерение напряжения (угла поворота, температуры и т. д.) с одновременным отображением на 8-ми сегментном дисплее.
- При нажатии на цифровые клавиши высвечивать их код и подавать звуковой сигнал.
- Переключение задач производится с помощью клавиатуры.
- Две клавиши: “*” и “#” использовать в качестве управляющих. Клавиша “#” переводит устройство в режим измерения напряжения (угла поворота, температуры и т. д.). Клавиша “*” для дополнительной задачи.

1.4 Общие сведения о микроконтроллере

Микроконтроллер ATmega128 включает следующие функциональные блоки:


- 8-разрядное арифметическо-логическое устройство (АЛУ);
- внутреннюю flash-память программ объемом 128 Кбайт с возможностью внутрисистемного программирования через последовательный интерфейс;
- 32 регистра общего назначения;
- внутреннюю EEPROM память данных объемом 4 Кбайт;
- внутреннее ОЗУ данных объемом 4 Кбайт;
- 6 параллельных 8-разрядных портов (PORTA..PORTG);
- 4 программируемых таймера-счетчика;
- 10-разрядный 8-канальный АЦП и аналоговый компаратор;
- последовательные интерфейсы UART0, UART0, TWI и SPI;
- блоки прерывания и управления (включая сторожевой таймер).

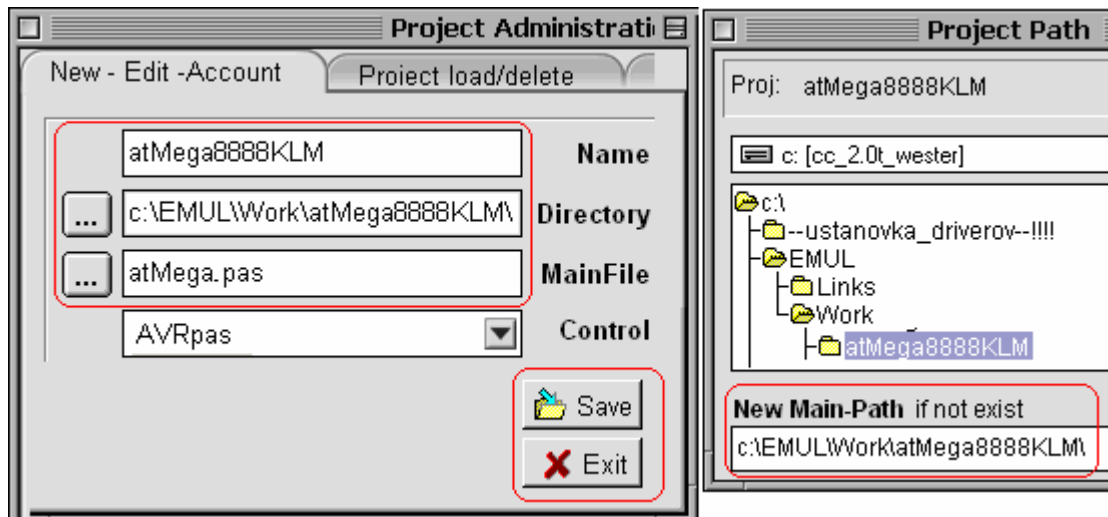
Подробное описание микроконтроллера ATmega128 приведено в лабораторной работе №15:

- на сайте - http://faculty.ifmo.ru/electron/cons/raspisanie_current.htm
- в библиотеке - Китаев Ю.В. Основы программирования микроконтроллеров ATMEGA 128 и 68HC908: [учебное пособие], [Каф. электроники] .— СПб.: СПбГУ ИТМО, 2007

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

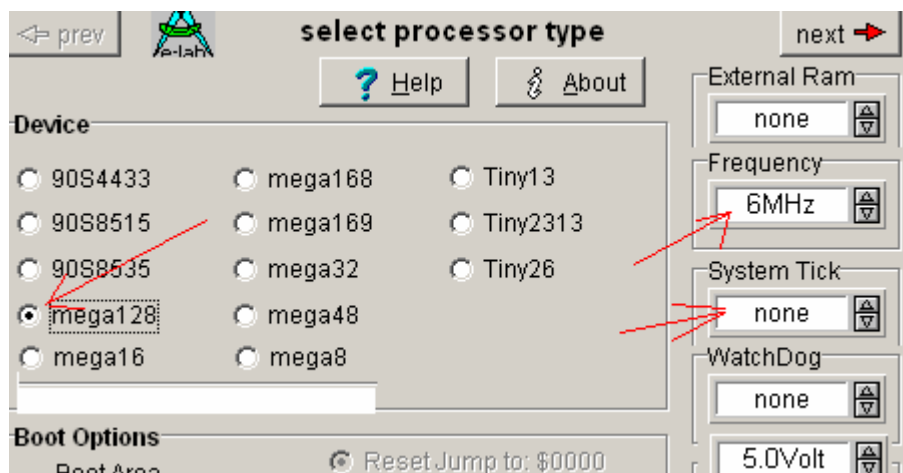
2.1 Создание шаблона программы. Откройте интегрированную среду


разработки E-LAB PED32  для МК семейства AVR. В открывшемся рабочем пространстве из главного меню выберите п. "File | New Project". В появившемся окне диалога на странице "New-Edit-Account" заполните отмеченные стрелками поля (создайте папку с номером СВОЕЙ группы и ИНИЦИАЛАМИ, например atMega8888KLM). Свою папку создавать обязательно в разделе "C:\EMUL\Work\". Инициалы (например KLM) также обязательны. Затем нажмите на кнопки "OK" и "Save" и "Exit".



Если появилось окно “Select Destination Book Page”, оставьте “E” – по умолчанию.

Появится первая страница "мастера или генератора шаблона программы". Выберем тип микроконтроллера "mega128", частоту задающего генератора и отметим отсутствие программного системного таймера "System Tick --> none".



После заполнения полей жмем на кнопку "next" в правом верхнем углу. Проскакиваем все остальные страницы генератора шаблонов с помощью той-же кнопки "next" и только на последней 16-ой странице жмем на кнопку  Build Application и затем на “Store” и “Exit”.

В окне редактора E-LAB PED32 появится сконструированный "Wizard'ом" шаблон нашей программы на языке "Pascal". Выделенную строку, включающую в программу файл с функциями клавиатуры нужно добавить самостоятельно. Оператор “EnableInts” разрешает прерывания.

Loop .. Endloop – бесконечный цикл, предотвращающий выполнение кода за пределами программы.

```


program atMega8888KLM;
{ $BOOTRST $0F000}           {Reset Jump to $0F000}
{$NOSHADOW}
{ $W+ Warnings}              {Warnings off}
Device = mega128, VCC=5;
Import ;
From System Import ;
Define
  ProcClock    = 6000000;      {Hertz}
  StackSize    = $0064, iData;
  FrameSize    = $0064, iData;

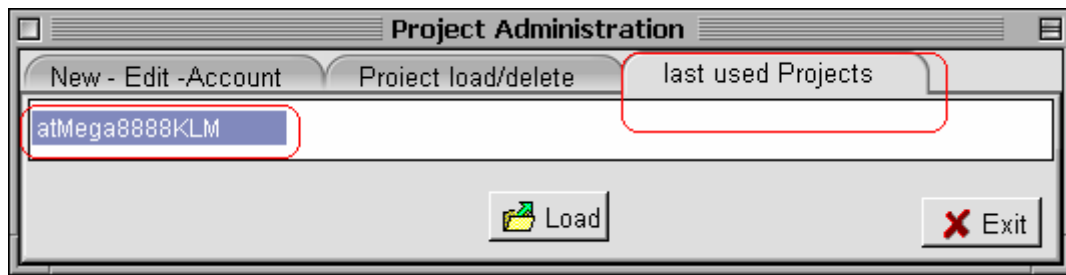
Implementation
{$IDATA}
{-----}
{ Type Declarations }
type
{-----}
{ Const Declarations }
{-----}
{ Var Declarations }
{$IDATA}
{-----}
{ functions }
{$I C:\EMUL\Work\KeyLib4Pascal.pas} //== ДОБАВИТЬ САМОСТОЯТЕЛЬНО
{-----}
{ Main Program }
{$IDATA}
begin
  EnableInts;
  loop
  endloop;
end atMega8888KLM.

```

Теперь необходимо добавить операторы, обеспечивающие функциональность нашей программы в соответствии с техническим заданием.

2.2 Разработка и отладка программы.

Если окно редактора открыто - продолжайте работу. Если нет, снова запустите приложение E-LAB PED32 . В открывшемся рабочем пространстве из главного меню выберите п. "Project | Load Project". В появившемся окне диалога на странице "Project load/delete" или "Last used Projects" выберите проект "atMega8888KLM" и нажмите на кнопку "Load".

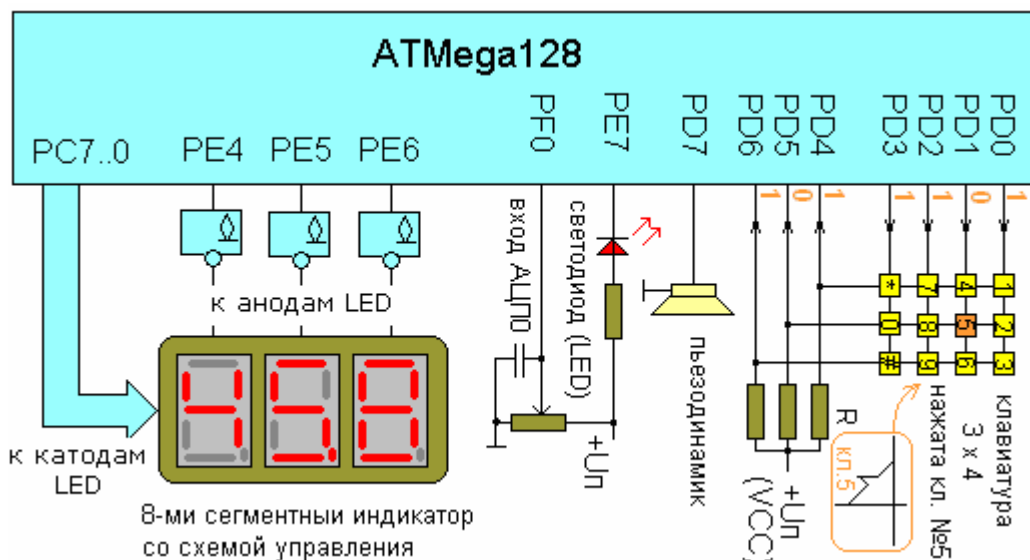


В редакторе должен появиться исходный текст программы. Если текст не появился, выберите п. меню "File | Open MainFile". Теперь текст программы появится.

2.2.1 Настройка портов.

Добавим в программу процедуру "InitPorts", настраивающую выходы портов C,D,E и F на **ввод** или **вывод** в соответствии с **направлением** передачи/приема данных внешними устройствами, подключенными к микроконтроллеру.

Как уже говорилось, за каждым портом закреплен регистр направления передачи данных (Data Direction Register X или **DDRx**, x=A,B..F) если порт/линия порта настроены на **вывод**, то в соответствующий регистр/бит регистра направления нужно записать '1'. Если на **ввод**, то – '0'. Также нужно учесть, что в начальном состоянии в регистрах DDRx записаны нули.





Из схемы на рисунке видно, что порт PC служит для вывода 8-ми сегментных кодов, поэтому в регистр направления DDRC (Data Direction Register C) необходимо записать код \$FF(или %11111111). Остальные биты регистров DDRx рассчитайте самостоятельно.

Четыре старших бита порта PE служат для подачи напряжения на аноды 8-ми сегментного индикатора и катод светодиода, поэтому 4 старших бита регистра направления DDRE д.б. равны Четыре младших бита порта PD предназначены для вывода “бегущего нуля” в 4 строки матрицы клавиатуры (3 вывода PD6..4 служат линиями возврата), а по линии порта PD7 выводится сигнал на пьезодинамик, т.е. в регистр направления DDRD нужно записать Порт PF (АЦП) при включении питания настроен на ввод, поэтому записывать в регистр DDRF нули не обязательно. Соответствующий фрагмент программы написанной на Pascal’е будет выглядеть следующим образом. Добавлены также некоторые переменные и вызов процедуры “Init_Ports”. Также в теле основного блока программы нужно разрешить прерывания записью единицы в бит I регистра состояния (флагов) SREG (В программе на Pascal’е этот бит устанавливается командой “EnableInts”).

```
{ Var Declarations }
{$IDATA}
var s:array[0..2] of byte;key,kn:byte; adccode:word;
{-----}
{ functions }
{$I C:\EMUL\Work\KeyLib4Pascal.pas}//=== ДОБАВИТЬ САМОСТОЯТЕЛЬНО
//========
procedure Init_Ports; //== задаем направления передачи данных
begin //== через порты, а также начальные значения
    DDRC:=$FF;//== или так DDRC:=$11111111;
    DDRD:=$70;//== значение $70 для примера (пересчитать)
    DDRE:=$F;//== значение $F для примера (пересчитать)
    PORTE:=$11111111;//== гасим индикаторы, подавая на
end; //== аноды светодиодов нули через инверторы
//========
.....
    EnableInts;
    Init_ports;
loop
```


ВНИМАНИЕ: В подпрограмме Init_Ports вам необходимо пересчитать значения \$70 и \$F, в соответствии со схемой подключения внешних устройств.
 #####

Компилируем, полученную на этом этапе программу, для чего в редакторе E-LAB PED32 нажмем на кнопку “Make Project” . Если ошибок при наборе программы не было, то компиляция закончится без предупреждающего сообщения и в статусной строке справа внизу появится сообщение- **Errors: 0**.

ВАЖНОЕ ПРИМЕЧАНИЕ! После каждой редакции текста выполняйте сборку проекта кнопкой . Иначе выявить все накопившиеся ошибки будет трудней.

Проверим, правильность расчета значений, загружаемых в регистры DDRD и DDRE. Для этого временно включим в программу файл “testPorts.pas” и сделаем вызов процедуры тестирования в основной программе. Добавьте в указанные места программы 2 строчки.



```
{SI C:\EMUL\Work\KeyLib4Pascal.pas}//=== ДОБАВИТЬ САМОСТОЯТЕЛЬНО
{SI C:\EMUL\Work\testPorts.pas}//=== только для тестирования
```

.....

```
Init_ports;
```

```
TestPorts;
```

```
loop
```

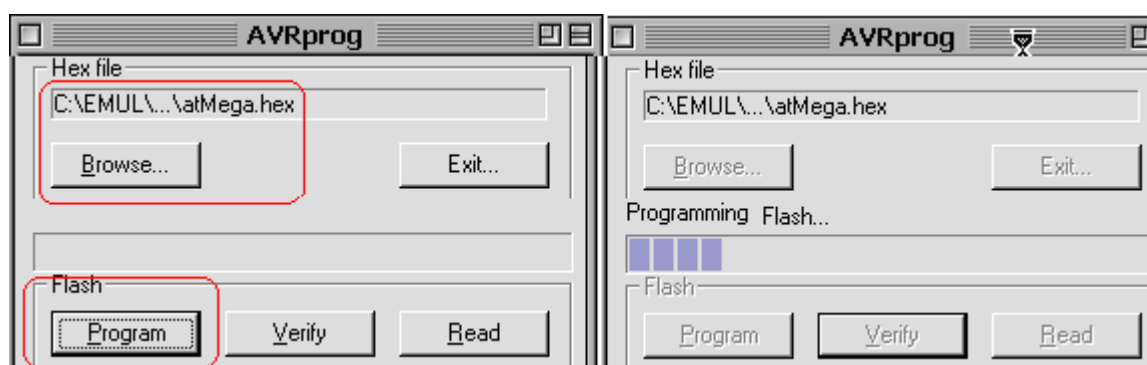
Снова компилируем  программу.  1 На данном этапе она должна иметь следующий вид:



```
program atMega8888KLM;
{ $BOOTRST $0F000}      {Reset Jump to $0F000}
{ $NOSHADOW}
{ $W+ Warnings}          {Warnings off}
Device = mega128, VCC=5;
Import ;
From System Import ;
Define
  ProcClock = 6000000;    {Hertz}
  StackSize = $0064, iData;
  FrameSize = $0064, iData;
Implementation
{$IDATA}
{-----}
{ Type Declarations }
type
{-----}
{ Const Declarations }
{-----}
{ Var Declarations }
{$IDATA}
var s:array[0..2]of byte;key,kn:byte; adccode:word;
{-----}
{ functions }
{SI C:\EMUL\Work\KeyLib4Pascal.pas}//=== ДОБАВИТЬ
САМОСТОЯТЕЛЬНО
{SI C:\EMUL\Work\testPorts.pas}//=== только для тестирования
```

```
//=====
procedure Init_Ports; //== задаем направления передачи данных
begin           //== через порты, а также начальные значения
  DDRC:=$FF; //== или так DDRC:=%11111111;
  DDRD:=$70; //== значение $70 для примера (пересчитать)
  DDRE:=$F; //== значение $F для примера (пересчитать)
  PORTE:=%11111111; //== гасим индикаторы, подавая на
end;           //== аноды светодиодов нули через инверторы
//=====
{-----}
{ Main Program }
{$IDATA}
begin
  EnableInts;
  Init_ports;
  TestPorts;
  loop
  endloop;
end atMega8888KLM.
```

2.2.2 Загрузка и выполнение программы.

Проверим ход выполнения промежуточного варианта программы, для чего запишем ее во флэш память микроконтроллера ATMega128. В процессе компиляции программная оболочка E-LAB PED32 создает файл с выполнимым кодом программы в специальном интеловском HEX-формате с расширением *.hex (в нашем примере это файл - atMega.hex). Этот файл должен быть доставлен на плату с микроконтроллером и записан в его флэш память. Для этой цели служит специальный кабель с программатором и свободно распространяемая программа AVRprog.




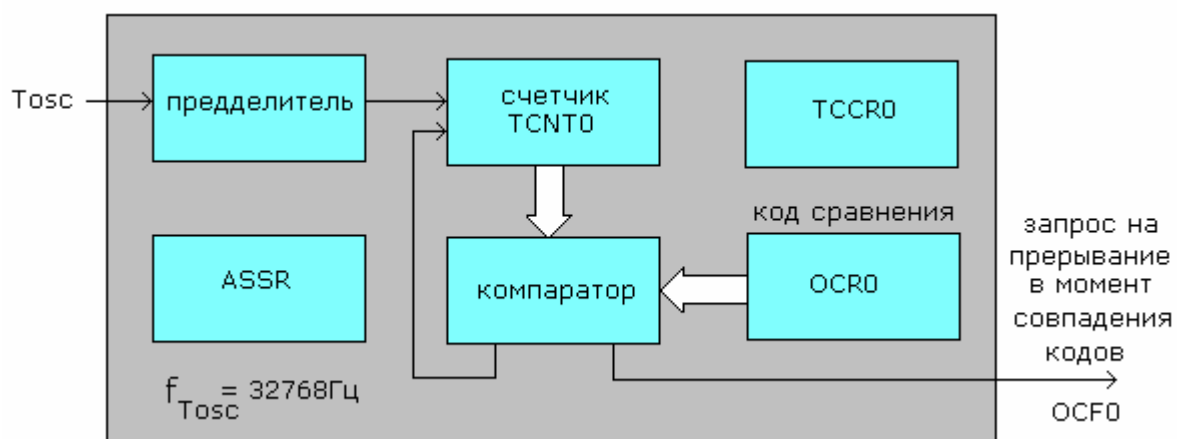
1  Запустите на выполнение программу  . Затем кнопкой “Browse..” найдите загрузочный файл “C:\EMUL\Work\atMega8888KLM\atMega.hex” и нажмите кнопку Flash -> Program. Шкала прогресса покажет момент окончания загрузки (если шкала

прогресса остановилась **НЕ ПРЕРЫВАЙТЕ** процесс программирования. В течение примерно минуты он закончится). Одновременно начнет выполняться записанная во Flash память программа. Нажимая на клавиши, вы увидите: 1) смену кодов высвечиваемых на трех индикаторах, 2) переключение светодиода и 3) звуковые сигналы. Для проверки правильности портов клавиатуры достаточно проверить 3 клавиши по диагонали и еще одну клавишу в оставшейся строке. То есть порты клавиатуры, 8-ми сегментного дисплея, пьезодинамика и светодиода настроены правильно. **Если, что-то из перечисленного не функционирует, пересчитайте значения DDRD и/или DDRE.** Закрывать окно AVRprog не нужно.

Результат покажите преподавателю

2.2.3 Программирование таймера.

 **2** Добавим в программу операторы, настраивающие таймер0 для формирования временных отрезков заданной длительности, например 1сек., и осуществляющие сигнализацию светодиодом. Ниже на рисунке приведена блок-схема Таймера “0” (таймер0). В основе таймера 8-ми битный счетчик. Импульсы на счетный вход поступают с выхода предварительного делителя (предделитель или prescaler). Коэффициент деления предделителя может иметь различные значения. В асинхронном режиме на предделитель импульсы поступают не с выхода генератора общей частоты синхронизации, а с выхода дополнительного генератора Tosc, частота которого задается специальным кварцевым резонатором с частотой $2^{15}=32768$ Гц. Такой “кварц” стоит во всех электронных часах и называется “часовым кварцем”.



В одном из режимов работы таймера0 код загруженный в регистр OCR0 (верхний предел) сравнивается с текущим кодом счетчика. При совпадении **следующий** входной импульс установит флаг сравнения

OCF0, который является запросом на прерывание и одновременно сбрасывает счетчик в исходное нулевое состояние. На рисунке верхний предел или код сравнения = 127, следовательно, импульс OCF0 и соответствующее прерывание будут происходить с частотой в 128 раз меньшей входной частоты счетчика. В нашем примере частота прерываний = $32768 / K_{\text{преддел}} / K_{\text{компар}} = 32768 / 8 / 128 = 32 \text{ Гц}$. $K_{\text{преддел}}$ - коэффициент деления частоты предделителем и $K_{\text{компар}}$ - коэффициент деления частоты, задаваемый компаратором.

Флаг прерывания при совпадении кодов OCF0 сбрасывается аппаратно при переходе на соответствующий вектор прерывания, поэтому в программе мы к нему обращаться не будем.



Временная диаграмма таймер-счетчика0 с предделением на 8 ($f_{\text{Tosc}}/8$) в режиме сброса при совпадении

Настройка таймера0 и контроль за его работой производится с помощью регистров:

- **TCCR0** – TimerCounterControlRegister0 (Регистр управления таймером/счетчиком0)
- **ASSR** – AsynchronousStatusRegister (Регистр состояния асинхронного режима таймера0)
- **TIMSK** – TimerInterruptMask (Регистр масок разрешения прерываний для таймеров/счетчиков)
- **OCR0** – OutputCompareRegister0 (Регистр кода сравнения)
- **TCNT0** – TimerCounter0 (запись начального или чтение текущего значения счетчика)
- **TIFR** - TimerInterruptFlagRegister (регистр флагов прерываний таймеров)

В нашей задаче понадобится явно обращаться к первым 4-м регистрам.

Регистр управления таймером/счетчиком - **TCCR0**

Разряд	7	6	5	4	3	2	1	0
--------	---	---	---	---	---	---	---	---

		WGM00			WGM01	CS02	CS01	CS00
--	--	-------	--	--	-------	------	------	------

Биты **WGM00, WGM01** задают режим работы таймер/счетчика.

Номер режима	WGM01	WGM00	Режим работы таймер/счетчика
0	0	0	Нормальный режим
1	0	1	ШИМ с коррекцией фазы
2	1	0	Сброс при совпадении
3	1	1	Быстрый ШИМ

Поделить частоту счетчиком можно двумя способами: 1) загрузкой начального кода и сбросом в начальный код при переполнении счетчика, 2) загрузкой конечного кода (верхний предел) в компаратор и сбросом в “0” при достижении этого значения (как на временной диаграмме). Первый способ используется в лаб. работе №30, поэтому здесь используем 2-й способ.

2👉 #####
ВНИМАНИЕ: Вам необходимо выбрать значения битов **WGM00** и **WGM01** для способа с компаратором. Эти биты запишите или запомните.

#####

Биты **CS02, CS01, CS00** - выбор коэффициента деления предделителя – Кпреддел (знаменатель в выражении $TOSC1 / K_{преддел}$).

CS02	CS01	CS00	Источник тактирования в зависимости от бита AS0 в регистре ASSR	
			AS0=0	AS0=1
0	0	0	таймер/счетчик T/C0 остановлен	
0	0	1	СК	TOSC1
0	1	0	СК / 8	TOSC1/8
0	1	1	СК / 32	TOSC1/32
1	0	0	СК / 64	TOSC1/64
1	0	1	СК / 128	TOSC1/128
1	1	0	СК / 256	TOSC1/256
1	1	1	СК / 1024	TOSC1/1024

Регистр состояния асинхронного режима таймера/счетчика T/C0 - **ASSR**

Разряд	7	6	5	4	3	2	1	0
	-	-	-	-	AS0			

Бит AS0: Разрешение асинхронного режима таймера/счетчика T/C0. При установленном (= 1) бите на вход предделителя таймера/счетчика 0 поступают импульсы с внешнего кварцевого генератора TOSC.

Теперь нужно разрешить прерывания при совпадении кодов OCR0 и TCNT0 (см. временную диаграмму). Для этого нужно записать 1 в бит OCIE0 (OutputCompareInterruptEnable) регистра масок прерываний TIMSK

Регистр масок прерываний для таймеров/счетчиков – TIMSK:

Разряд	7	6	5	4	3	2	1	0
							OCIE0	

ВНИМАНИЕ: Вам необходимо выбрать значения: 1) битов **CS02**, **CS01**, **CS00** (с учетом заданного значения Кпреддел), 2) бита **AS0** и 3) бита **OCIE0**. Эти значения также запишите или запомните.
 #####

В соответствии с изложенным запишите процедуру инициализации таймера0 и ее вызов:


```
//=====
procedure Init_Timer0_Async;
begin
  TCCR0:=$xx; //== байт xx - замените рассчитанным значением
  OCR0:=$xx; //== байт xx - замените рассчитанным значением
  ASSR.x:=1; //== бит AS0=1 (позицию x - определите из таблицы)
  TIMSK.x:=1; //== бит OCIE0=1 (позицию x - определите из таблицы)
end;
//=====
{-----}
{ Main Program }
```

```
.....
Init_ports; //== настройка линий портов на ввод/вывод
Init_Timer0_Async; //== настройка таймера0 на асинхр. режим
```


ВНИМАНИЕ: В строке OCR0:=\$xx, коэффициент деления “Ккомпар=xx”, задаваемый компаратором рассчитайте из формулы: Частота прерываний = 32768 / Кпреддел / Ккомпар. Причем частота прерываний и Кпреддел даны в задании.
 #####


Запишите процедуру настройки таймера0 в программу и **ЗАМЕНИТЕ XX** и **X** рассчитанными значениями. Заодно закомментируйте или удалите ненужные теперь две строки программы.

```
//({$I C:\EMUL\Work\testPorts.pas}) //== только для тестирования
.....
// TestPorts;
```



Скомпилируйте программу . Загружать ее во флэш-память микроконтроллера пока не нужно.

Далее в программу необходимо добавить процедуру – обработчик прерывания при сравнении кодов счетчика и регистра сравнения. В процедуре с **предопределенным именем TIMER0COMP** предусмотрим управление светодиодом (при каждом вызове состояние светодиода будет инвертироваться).

```
//=====
var b: boolean; //== вспомогательная логическая переменная
interrupt TIMER0COMP; //== обработчик прерывания при совпадении
begin //== кодов TCNT0 и OCR0
    b:=not b; //== меняем состояние
    PORTx.x:= b; //== светодиода
end;
//=====
{-----}
{ Main Program }
```

ЗАМЕНИТЕ X и X значениями, взятыми из схемы (раздел 2.2.1) подключения внешних устройств. Первый ‘x’, естественно наименование порта (A,B,C,D,E или F), а второй номер его линии (0..7), к которой подключен светодиод. Снова скомпилируйте программу  и загрузите ее во флэш-память микроконтроллера. Светодиод будет переключаться с заданной частотой. Если это не происходит – необходимо проверить или пересчитать 6 операндов/обозначений с символическим именем ‘x’.

Результат покажите преподавателю

 **3** В настоящий момент программа должна иметь следующий вид (причем, значения выделенные жирным шрифтом вами уже рассчитаны).

```
program atMega8888KLM;
{ $BOOTRST $0F000}      {Reset Jump to $0F000}
{ $NOSHADOW}
{ $W+ Warnings}         {Warnings off}
Device = mega128, VCC=5;
Import ;
From System Import ;
Define
    ProcClock  = 6000000;    {Hertz}
    StackSize  = $0064, iData;
    FrameSize  = $0064, iData;
Implementation
{ $IDATA}
{-----}
{ Type Declarations }
type
```

```

{-----}
{ Const Declarations }
{-----}
{ Var Declarations }
{$IDATA}
var s:array[0..2]of byte;key,kn:byte; adccode:word;
{-----}
{ functions }
{$I C:\EMUL\Work\KeyLib4PascaL.pas}//=== ДОБАВИТЬ
САМОСТОЯТЕЛЬНО
//========
procedure Init_Ports; //== задаем направления передачи данных
begin
    //== через порты, а также начальные значения
    DDRC:=$FF; //== или так DDRC:=%11111111;
    DDRD:=$70; //== значение $70 для примера (пересчитать)
    DDRE:=$F; //== значение $F для примера (пересчитать)
    PORTE:=%11111111; //== гасим индикаторы, подавая на
end; //== аноды светодиодов нули через инверторы
//========
procedure Init_Timer0_Async;
begin
    TCCR0:=$xx; //== байт xx - замените рассчитанным значением (например
F2)
    OCR0:=$xx; //== байт xx - дан в задании (например 00)
    ASSR.x:=1; //== бит AS0=1 (позицию x - определите из таблицы,
например 4)
    TIMSK.x:=1; //== бит OCIE0=1 (позицию x - определите из таблицы,
например 6)
end;
//========
var b: bool*ean; //== вспомогательная логическая переменная
interrupt TIMER0COMP; //== обработчик прерывания при совпадении
begin //== кодов TCNT0 и OCR0
    b:=not b; //== меняем состояние светодиода
    PORTx.x:= b; имя порта и номер бита рассчитать (например PORTB.0)
end;
//========
{-----}
{ Main Program }
{$IDATA}
begin
    EnableInts; //== снятие запрета с прерываний
    Init_ports;
    Init_Timer0_Async;

```

```

loop
endloop;
end atMega8888KLM.

```

Некоторые пояснения к механизму прерываний. Ниже приведена таблица некоторых векторов прерываний МК ATmega128 и пример расположения вектора прерывания при совпадении кодов таймера0 по адресу 1E и процедуры обработчика с символическим адресом TIMER0COMP.

Vector No.	Program Address	Source	Interrupt Definition
14	\$001A	TIMER1 COMPB	Timer/Counter1 Compare Match B
15	\$001C	TIMER1 OVF	Timer/Counter1 Overflow
16	\$001E	TIMER0 COMP	Timer/Counter0 Compare Match
17	\$0020	TIMER0 OVF	Timer/Counter0 Overflow
18	\$0022	SPI, STC	SPI Serial Transfer Complete
22	\$002A	ADCRDY	ADC Conversion Complete
23	\$002C	EE READY	EEPROM Ready

запрос OCF0
1E
Flash память
вектор
обработчик

TIMER0COMP

В момент совпадения кодов счетчика TCNT0 и регистра OCR0 схема управления компаратора сформирует признак переполнения OCF0 и запрос на прерывание. МК по запросу автоматически обратится к ячейке с адресом 1E, куда программист (или транслятор) заносит команду перехода JMP TIMER0COMP к процедуре обработки прерывания с символическим адресом TIMER0COMP. Осуществив этот переход МК начнет выполнять подпрограмму **interrupt TIMER0COMP....** (см. выше), в которой в нашем примере переключается светодиод. Для чего это нужно? В процессе разработки программы ее размер и относительное положение отдельных ее фрагментов постоянно меняется, поэтому указать сразу окончательное значение физического адреса, соответствующего символическому имени (TIMER0COMP) невозможно. Адрес же ячейки памяти, где будет храниться адрес перехода (или сама команда перехода) определяется структурой МК и известен из справочника. Содержимое такой ячейки(ее) памяти называют вектором прерывания.

address	opcode	memnemonic
00001C	940C04A7	JMP 04A7h
00001E	940C03A9	JMP 03A9h
000020	940C04A7	JMP 04A7h
000022	940C04A7	JMP 04A7h
000024	940C04A7	JMP 04A7h
000026	940C04A7	JMP 04A7h
000028	940C04A7	JMP 04A7h
00002A	940C03B8	JMP 03B8h
00002C	940C04A7	JMP 04A7h
00002E	940C04A7	JMP 04A7h
0003A7	940C0243	JMP 0243h
0003A9	93EF	PUSH R30
0003AA	93FF	PUSH R31

На этом рисунке показан фрагмент кода после очередной трансляции, из которого видно, что в ячейку памяти с адресом 1E транслятор поместил команду перехода JMP 03A9h к подпрограмме TIMER0COMP,

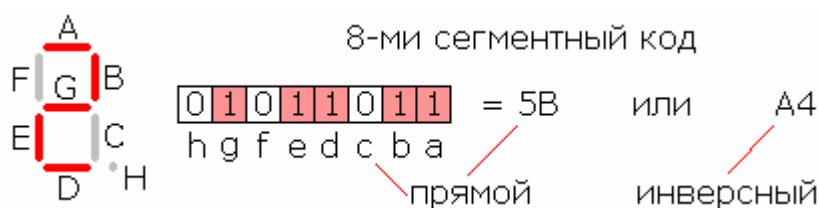
которая на данном этапе разработки программы располагается по адресу 0003A9 (у вас может быть другой адрес). Аналогично действует механизм прерывания и для других источников, в частности для АЦП (вектор по адресу 2A).

2.2.4 Программирование 8-ми сегментного индикатора.

3👉 Следующим усовершенствованием нашей программы будет отображение цифровых кодов в том числе нажатых клавиш, на 8-ми сегментном индикаторе и подача звукового сигнала. Для этого сначала добавим в программу 8-ми сегментные ИНВЕРСНЫЕ коды символов (т.к. через порт РС коды цифр подаются на катоды светодиодов).

```
{ Const Declarations }
const key8segm: array [0..11] of byte = ($F9,$A4,$B0,$99,$92,$82,
//                                     1 2 3 4 5 6
                                     $xx,$80,$xx,$7F,$xx,$B6);
//                                     7 8 9 * 0 #
```

Пример расчета для цифры '2':



Начертания для '*' и '#' – условные, их можно оставить, а можно заменить.

ВНИМАНИЕ: Инверсные коды для цифр '7', '9' и '0' рассчитайте самостоятельно и подставьте их вместо 'xx'.
 #####

Чтобы убедиться в правильности отображения рассчитанных символов и подачи звуковых сигналов добавим в программу вызов процедуры ожидания нажатия на клавишу 'Wait4KeyPressed', стандартный блок выбора действий 'case ... of', которые нам понадобятся для дальнейшей разработки и подпрограмму 'bzzz' формирующую короткий звуковой сигнал.

```

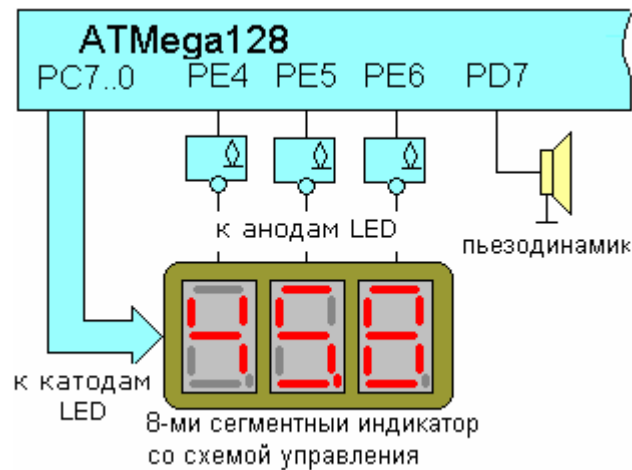
loop//== начало бесконечного цикла
Wait4KeyPressed;//== ждем нажатия на клавишу
case key of //== выполняем действия в соотв-ии с нажатой клавишей
  '0'..'9': //== key - ASCII код клавиши, kn - ее порядковый номер (0..11)
    PORTx:=$xx;//==на анод(ы) заданных. инд-ров подать напр-е (е.г. $40)
    PORTx=key8segm[kn];//== выводим 8-ми сегм. код нажатой клавиши
    bzzz;//== подача звукового сигнала
  | //== этот разделитель обязателен
endcase;//== конец оператора множественного выбора "case .. of"
endloop;//== возврат к началу бесконечного цикла


//=====
var bz:boolean; jj,ii:byte;//== ii,jj - вспом. переменные (счетчики циклов)
procedure bzzz;begin//== подпрограмма подачи звукового сигнала
  ii:=0;
  loop//== начало бесконечного цикла
    bz:=not bz;//== bz - "амплитуда" звука (от 0 до 1) - 01010101.....
    PORTx.x:=bz;//== выводим "полуволну" - 01 или 10 на пьезодинамик
    for jj:=0 to 255 do endfor;//== jj=255 - высота звука (длит-сть "полуволны")
    ii:=ii+1;
    if ii=50 then//== ii=50 - длительность звука
      exitloop;//== заканчиваем подачу сигнала (выход из бескон. цикла)
    endif;
  endloop;//== возврат к началу бесконечного цикла
end;
//=====
{-----}
{ Main Program }

```


ВНИМАНИЕ: Вам опять придется заменить 6 “иксов” значениями, взятыми или рассчитанными из приведенного ниже рисунка-фрагмента схемы для заданных индикаторов и динамика.
 #####

Выводить код нажатой клавиши будем на индикатор(ы), указанный(ые) в задании. Имейте в виду, что для подачи напряжения на анод индикатора (логическая ‘1’) необходимо учесть, что он (анод) подключен к порту через **инвертор**. На остальных выходах этого порта подключенных к оставшимся анодам д.б. **противоположные значения**. На прочих выходах - любые значения. Кстати, во фрагменте программы, в качестве примера, приведено неверное значение \$40, которое вам и нужно пересчитать.



Скомпилируйте программу  и загрузите ее во флэш-память микроконтроллера с помощью “AVRprog”. Нажимая поочередно на **все** клавиши ‘0..9’ убедитесь в правильности 8-ми сегментных кодов, высвечиваемых **на заданных** индикаторах и подаче звуковых сигналов.

ВНИМАНИЕ: В этом месте вы может быть обратили внимание на некорректное переключение светодиода при нажатии на цифровые клавиши. Объяснение и решение этого вопроса будет приведено в следующих разделах.

Результат покажите преподавателю

4 В настоящий момент программа должна иметь следующий вид (значения **x** и **xx** вами только что рассчитаны).

```

program atMega888KLM;
{ $BOOTRST $0F000}      {Reset Jump to $0F000}
{ $NOSHADOW}
{ $W+ Warnings}          {Warnings off}
Device = mega128, VCC=5;
Import ;
From System Import ;
Define
  ProcClock = 6000000;    {Hertz}
  StackSize = $0064, iData;
  FrameSize = $0064, iData;
Implementation
{$IDATA}
{-----}
{ Type Declarations }
type
{-----}
{ Const Declarations }
const key8segm: array [0..11] of byte = ($F9,$A4,$B0,$99,$92,$82,

```

```
//                               1   2   3   4   5   6
//      $xx,$80,$xx,$7F,$xx,$B6);//инв. 8-ми сегм. коды
//      7   8   9   *   0   #
{-----}
{ Var Declarations }
{$IDATA}
var s:array[0..2]of byte;key,kn:byte; adccode:word;
{-----}
{ functions }
{$I C:\EMUL\Work\KeyLib4Pascal.pas}//== ф-ии для работы с клав-ой
//=====================================================
procedure Init_Ports; //== задаем направления передачи данных
begin           //== через порты, а также начальные значения
    DDRC:=$FF;//== или так DDRC:=%11111111;
    DDRD:=$8f;//== значение $70 для примера (пересчитать)
    DDRE:=$f0;//== значение $F для примера (пересчитать)
    PORTE:=%11111111;//== гасим индикаторы, подавая на
end;           //== аноды светодиодов нули через инверторы
//=====================================================
procedure Init_Timer0_Async;
begin
    TCCR0:=$0D;//== биты CS00,02=1(Rпреддел=128),WGM01=1-сброс
    счетчика при совп.
    OCR0:=$FF;//== код совпадения
    ASSR.3:=1; //== бит AS0=1 - асинхронный режим Timer0 от кварца
    32768Гц
    TIMSK.1:=1;//== бит OCIE0:=1 - разрешить прерывания при совпадении
    текущего
end;           //== значения таймера0 с кодом в регистре совпадения OCR0
//=====================================================
var b:boolean;//== вспомогательная логическая переменная
interrupt TIMER0COMP;//== обработчик прерывания при совпадении
begin           //== кодов TCNT0 и OCR0
    b:=not b; //== меняем состояние
    PORTE.7:= b; //== светодиода
end;
//=====================================================
var bz:boolean; jj,ii:byte;//== ii,jj - вспом. переменные (счетчики циклов)
procedure bzzz;begin//== подпрограмма подачи звукового сигнала
    ii:=0;
loop//== начало бесконечного цикла
    bz:=not bz;//== bz - "амплитуда" звука (от 0 до 1) - 01010101.....
    PORTx.x:=bz;//== выводим "полуволну" - 01 или 10 на пьезодинамик
```

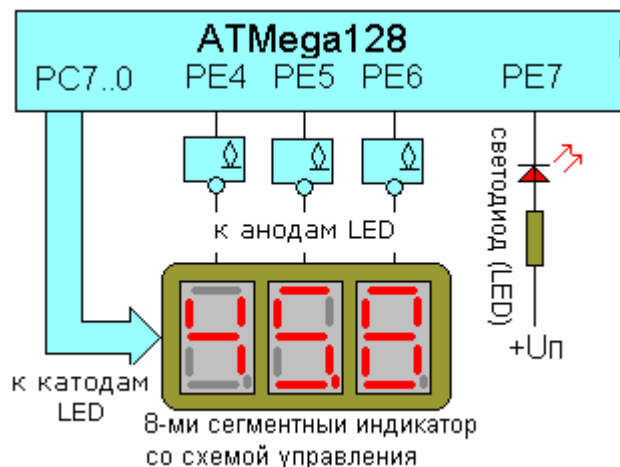
```

for jj:=0 to 255 do endfor;//=== jj=255 - высота звука (длит-сть
"полуволны")
ii:=ii+1;
if ii=50 then//=== ii=50 - длительность звука
  exitloop;//=== заканчиваем подачу сигнала (выход из бескон. цикла)
endif;
endloop;//=== возврат к началу бесконечного цикла
end;
//========
{-----}
{ Main Program }
{$IDATA}
begin
  EnableInts;//=== снятие запрета с прерываний
  Init_ports;//=== настройка линий портов на ввод/вывод
  Init_Timer0_Async;//=== настройка таймера0 на асинхр. режим
  loop//=== начало бесконечного цикла
    Wait4KeyPressed;//=== ждем нажатия на клавишу
    case key of//=== выполняем действия в соотв-ии с нажатой клавишей
      '0'..'9'://=== key - ASCII код клавиши, kn - ее порядковый номер (0..11)
        PORTx:=$xx;//===на анод(ы) заданных. инд-ров подать напр-е (e.g.
$40)
        PORTx:=key8segm[kn];//=== выводим 8-ми сегм. код нажатой клавиши
        bzzz;//=== подача звукового сигнала
        |//=== этот разделитель обязателен
      endcase;//=== конец оператора множественного выбора "case .. of"
    endloop;//=== возврат к началу бесконечного цикла
end atMega8888KLM.

```


2.2.5 Вспомогательные подпрограммы.

Далее нам понадобятся две вспомогательные подпрограммы – функция “Dec2seg” и процедура “Display8seg”. Первая преобразует число




в 8-ми сегментный код, а вторая отображает его на LED дисплее.

Как видно из рисунка через порт “С” выводится 8-ми сегментный код цифры **одновременно** на все катоды светодиодов всех трех индикаторов, а высветится соответствующий символ только на том индикаторе, на общий анод, которого подан высокий

уровень напряжения (лог. 1). Напряжение снимается с выхода **инвертора** (с открытым коллектором). То есть на соответствующем выходе 4,5 или 6 порта “Е” должен быть лог. 0, А на остальных двух выходах этого порта, подключенным к двум другим индикаторам должна быть 1. Сдвигая “0” в соседний разряд, выводим в него следующий 8-ми сегментный код и т.д. Следовательно, в каждый момент времени активен **ТОЛЬКО** один индикатор из трех. Если этот процесс будет повторяться продолжительное время, то на дисплее мы увидим “неподвижное” изображение десятичного числа. Глаз не способен различить кратковременную **последовательную смену** кодов.  Процесс последовательного смещения “0” (в некоторых системах “1”) называется **“бегущим нулем”** (единицей) или сканированием с помощью “бегущего нуля”.

```
//=====
procedure Display8seg; //== однокр. высвечивание на индикаторе 8-ми сегм. кода
var i,j:byte;running0:byte;//== i,j - вспом. переменные (счетчики циклов)
begin
  running0:=%11110111; //== начальное значение "бегущего нуля"
  for i:=2 downto 0 do //== сканируем 3 линии порта Е (PORTE.6 .. PORTE.4)
    running0:=running0 rol 1; //== готовим вывод в следующий индикатор
    if not b then running0:=running0 and $7F; endif; //== нельзя вмешиваться
    //== в состояние LED (PORTE.7) - управляется обработчиком TIMER0COMP !!
    PORTC:=s[i]; //== выводим инверсный 8-ми сегментный код цифры на индикатор
    PORTE:=running0; //== подаем на анод текущего 8-сегм. инд-ра высокий уровень
    for j:=0 to 255 do endfor; //== для увеличения яркости
  endfor;
end;
//=====
{-----}
{ Main Program }
```

 **5** Процедура “Display8seg”, как раз и осуществляет однократное сканирование дисплея с помощью “бегущего нуля”. В операнде “running0”, логич. 0 последовательно смещается на один бит **влево**, обеспечивая переход к следующему индикатору. Циклический сдвиг битов операнда “running0” производится командой “rol”. Далее, блок команд “if not b...” осуществляет синхронизацию значения 7-го бита порта “Е”, к которому подключен светодиод, управляемый обработчиком прерывания TIMER0COMP. Далее следует цикл “for j:=...” стандартная задержка для увеличения времени в течение, которого текущий 8-ми сегментный индикатор, находится во включенном состоянии. 8-ми сегментный код s[i] вычисляется и возвращается подпрограммой-функцией “Dec2seg”.

Назначение этой подпрограммы в преобразовании двоичного числа в трехзначный 8-ми сегментный десятичный код. Алгоритм работы этой функции – стандартный. Число делится (“в столбик”) на основание системы счисления, в которую его необходимо перевести (в нашем случае

10). Остаток записывается в следующую цифру результата, начиная с младшего разряда. Частное снова делится. Процедура завершается, когда частное становится меньше единицы. Параллельно в выходной массив “s” записываются полученные коды цифр исходного числа. Это могут быть ASCII коды, 8-ми сегментные коды или что-то другое. В нашем случае формируется массив из трех цифр, коды которых берутся из таблицы “key8seg”, в составлении которой вы уже поучаствовали.

Еще один момент. При непрерывном отображении изменяющихся чисел, в том случае, когда, например, следующее число имеет меньше значащих цифр, чем предыдущее, в неиспользуемых старших членах массива s[i] остаются предыдущие цифры и выводимый на дисплей результат искажается. Например вместо “_8” или “_48” будет “158” или “748”. Поэтому вначале каждого вызова “Dec2seg” в массив “cifry” записывается неотображаемый код “FF” (вспомним, что он выводится на катоды светодиодов и “гасит” их).

5

```
//=====
function Dec2seg (chislo:word):array;
var cifry:array[0..2] of byte; i:byte; ostatok:word;
begin
  i:=0; cifry:=blank; //== FF FF FF - коды гашения индикаторов
  repeat
    ostatok:=chislo mod 10; //== вычисляем очередную ДЕСЯТИЧНУЮ цифру числа
    chislo:=chislo div 10; //== путем последовательного деления результата на 10
    if ostatok<>0 then //== если остаток не равен 0, то
      cifry[i]:=key8segm[ostatok]; //== извлекаем из таблицы код цифры от 1 до 9
    else //== иначе извлекаем из таблицы код цифры 0
      cifry[i]:=key8segm[ostatok+11]; //== цифра 0 не на "своем" месте на клав.
    endif;
    Inc(i); //== переходим к следующей цифре числа (инкремент)
  until chislo<1; //== до тех пор пока частное не станет < 1
  return(cifry); //== возвращаем массив трех цифр
end;
//=====
procedure Display8seg; //== однокр. высвечивание на индикаторе 8-ми сегм. кода
```


Оба фрагмента введите в программу, в указанные места. Заодно, в раздел объявлений и в основную программу добавим следующие фрагменты:

```
{ Const Declarations }
const blank:array[0..2] of byte=($FF,$FF,$FF); //== код гашения
const key8segm: array [1..12] of byte = .....
```

```


bzzz; //== подача короткого звукового сигнала
| //== этот разделитель обязателен
'#': //== если нажата '#'
loop
  s:=Dec2seg(945); //== преобразуем число в 8-ми сегм десятичный код
  Display8seg; //== отображаем 8-сегм. код на дисплее
  if ScanKeyOnce(kn) then //== если нажата клавиша,
    PORTC:=$FF; //== погасить дисплей
    exitloop; //== и выйти из цикла
  endif;
endloop;
| //== этот разделитель обязателен
endcase; //== конец оператора множественного выбора "case .. of"

```

В последнем фрагменте добавлен код реакции на клавишу “#”, в котором некое число выводится на дисплей. Выход из цикла – при нажатии на любую цифровую клавишу. Скомпилируйте программу  и загрузите ее. Убедитесь в работоспособности добавленного раздела (вывод числа на дисплей при нажатии на клавишу “#”).

Результат покажите преподавателю

2.2.6 Программирование АЦП.

 **6** Введем в программу код, позволяющий измерять напряжение на выходе датчика с помощью АЦП. В качестве датчика используется потенциометр, поэтому код на выходе АЦП пропорционален углу поворота рукоятки и в этом качестве можно считать, что АЦП измеряет не просто напряжение, а угол поворота. ATmega128 содержит 10-разр. АЦП последовательного приближения. АЦП связан с 8-канальным аналоговым мультиплексором. 8 однополярных входов АЦП связаны с линиями порта F. Минимальное значение соответствует уровню GND (0), а максимальное уровню AREF равное **2,56В** в лабораторном стенде. АЦП программируется с помощью регистра управления и – **ADCSRA**. Два разряда не задействованные в работе в таблице не приведены (подробности в лаб. раб. №15).

Разряд	7	6	5	4	3	2	1	0
	ADE N	ADS C			ADI E	ADPS 2	ADPS 1	ADPS 0
Исх. значение	0	0	0	0	0	0	0	0

Бит **ADEN**: Разрешение работы АЦП. Запись в данный бит лог. 1 разрешает работу АЦП. Если в данный бит записать лог. 0, то АЦП отключается, даже если он находился в процессе преобразования.

Бит **ADSC**: Запуск преобразования АЦП. В режиме одиночного преобразования установка данного бита начинает преобразование. В

режиме автоматического перезапуска установкой этого бита инициируется только первое преобразование, а все остальные выполняются автоматически. Первое преобразование после разрешения работы АЦП, инициированное битом ADSC, выполняется по расширенному алгоритму и длится 25 тактов синхронизации АЦП, вместо обычных 13 тактов. Это связано с необходимостью инициализации АЦП. Бит **ADIE**: Разрешение прерывания АЦП. После записи лог. 1 в этот бит, при условии, что установлен бит I в регистре SREG, разрешается прерывание по завершении преобразования АЦП. Биты **ADPS2..0**: Биты управления предделителем (Prescaler) АЦП. Данные биты определяют на какое значение будет поделена тактовая частота МК перед подачей на вход синхронизации АЦП. Если требуется максимальная разрешающая способность (10 разрядов), то частота синхронизации должна быть в диапазоне 50...200 кГц. Если достаточно разрешение менее 10 разрядов, но требуется более высокая частота преобразования, то частота на входе АЦП может быть установлена свыше 200 кГц.

ADPS2	ADPS1	ADPS0	Коэффициент деления
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Работа АЦП разрешается путем установки бита ADEN в ADCSRA. Выбор опорного источника и канала преобразования не возможно выполнить до установки ADEN.

АЦП генерирует 10-разрядный результат, который помещается в пару регистров данных АЦП **ADCH** и **ADCL**. По умолчанию результат преобразования размещается в младших 10-ти разрядах 16-разр. слова (выравнивание справа), но может быть размещен в старших 10-ти разрядах (выравнивание слева) путем установки бита ADLAR в регистре ADMUX.

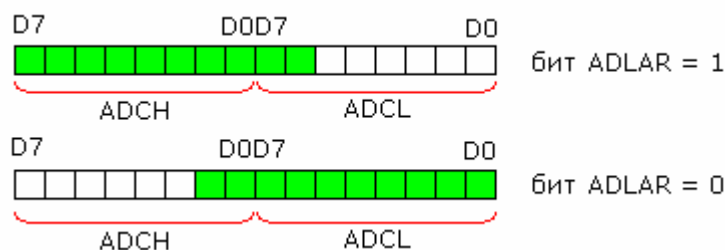
Практическая полезность представления результата с выравниванием слева существует, когда достаточно 8-разрядное разрешение, т.к. в этом случае необходимо считать только регистр ADCH. В другом же случае необходимо **первым** считать содержимое регистра ADCL, а затем ADCH, чем гарантируется, что оба байта являются

результатом одного и того же преобразования. Одиночное преобразование запускается путем записи лог. 1 в бит запуска преобразования АЦП ADSC. Данный бит остается в высоком состоянии в процессе преобразования и сбрасывается по завершении преобразования.

Регистр управления входным мультимплексором, источником опорного напряжения и способом выравнивания выходного кода - **ADMUX**. В таблице приведен бит ADLAR –

Разряд	7	6	5	4	3	2	1	0
			ADLAR					

По завершении преобразования 10-ти битный результат помещается в двух регистрах АЦП - ADCH и ADCL.




Левосторонний формат ADLAR=1 представления результата удобно использовать, если достаточно 8 разрядов. В этом случае 8-разрядный результат хранится в регистре ADCH и, следовательно, чтение регистра ADCL можно не выполнять. При правостороннем формате необходимо сначала считать ADCL, а затем ADCH.

Настройка АЦП. Прежде всего, необходимо разрешить работу АЦП в режиме прерывания, т.е. записать в биты ADEN и ADIE единицу. Для 10-ти битного разрешения АЦП необходимо выбрать значения частоты его синхронизации. Т.к. она должна в этом случае находиться в диапазоне 50..200КГц, то при общей частоте синхронизации МК равной 6МГц коэффициент деления можно взять равным $K_{\text{предделАЦП}} = 64$ ($6000000/64 = 93750\text{Гц}$) и биты ADSP2, ADSP1, ADSP0=....(в вашем задании будет другой). Запускать преобразование при настройке не нужно, поэтому бит ADSC оставим в начальном положении. Таким образом, рассчитываем управляющий байт АЦП - ADCSRA.

Из схемы подключения видно, что источник сигнала (средний вывод потенциометра) подключен к нулевому входу ADC0 (вывод PF0 порта "F"), поэтому биты MUX4..0 регистра управления ADMUX должны быть равны нулю. В этом же регистре положим **ADLAR=1**, т.е. **выравниваем результат влево**. В результате в регистр ADMUX необходимо записать код Запуск преобразования АЦП производится записью 1 в бит ADSC

регистра ADCSRA в теле основного блока программы и в самом обработчике для нового запуска.

6  Процедура настройки АЦП и обработчик прерывания при завершении АЦ преобразования могут выглядеть следующим образом:

```
//=====
procedure Init_ADC; begin //== настройка АЦП
    ADCSRA:=$xx; //== уст. бит разрешения работы АЦП и Кпреддел = 64 (е.г. =$71)
    ADMUX:=$xx; //== биты АЦП выравнены влево и читать можно один ADCH (е.г. =$DF)
end;
//=====
interrupt ADCRDY;begin //== обработчик при окончании АЦ преобразования
    adccode:= word(ADCH); //== читаем только ст. байт кода АЦП (8-ми битный АЦП)
    ADCSRA.6:=1; //== новый запуск АЦП
end;
//=====
{-----}
{ Main Program }
```

#####

ВНИМАНИЕ: Необходимо пересчитать два значения “xx” в соответствии с заданием

#####


В основном модуле добавьте вызов подпрограммы настройки АЦП и команды для начала и остановки преобразований.

```
Init_Timer0_Async; //== настройка таймера0 на асинхр. режим
Init_ADC; //== настройка АЦП (регистры ADCSRA и ADMUX)
loop //== начало бесконечного цикла

.....
'#': //== если нажата '#'
    ADCSRA.6:=1; //== подготов-ный пуск АЦП (остальные в обраб-ке прер-я)
    loop
        s:=Dec2seg(adccode); //== преобразуем код АЦП в 8-ми сегм код чный код
        Display8seg; //== отображаем 8-сегм. код на дисплее
        if ScanKeyOnce(kn) then //== если нажата клавиша,
            ADCSRA.6:=0; //== то остановить АЦП,
            PORTC:=$FF; //== погасить дисплей
            exitloop; //== и выйти из цикла
```

Скомпилируйте программу и загрузите ее кнопкой “Flash -> Program” программатора AVRprog. Затем, нажмите на клавишу “#” и, вращая ручку потенциометра, удостоверьтесь в том, что код соответствующий напряжению (углу поворота) отображается на светодиодном дисплее.

Результат покажите преподавателю

7  На этом этапе программа имеет почти законченный вид:

```
program atMega8888KLM;
{ $BOOTRST $0F000}      {Reset Jump to $0F000}
{ $NOSHADOW}
{ $W+ Warnings}         {Warnings off}
```

```

Device = mega128, VCC=5;
Import ;
From System Import ;
Define
  ProcClock = 6000000;    {Hertz}
  StackSize = $0064, iData;
  FrameSize = $0064, iData;
Implementation
{$IDATA}
{-----}
{ Type Declarations }
type
{-----}
{ Const Declarations }
const blank:array[0..2]of byte=($FF,$FF,$FF);//=== код гашения
const key8segm: array [0..11] of byte =
  ($F9,$A4,$B0,$99,$92,$82,$F8,$80,$90,$7F,$C0,$B6);//=== инв. 8-ми сегм.
коды
// 1 2 3 4 5 6 7 8 9 * 0 #
//          $xx,$80,$xx,$7F,$xx,$B6;
{-----}
{ Var Declarations }
{$IDATA}
var s:array[0..2]of byte; key,kn:byte; adccode:word;
{-----}
{ functions }
{$I C:\EMUL\Work\KeyLib4PascaL.pas}//=== ф-ии для работы с клав-ой
//===
procedure Init_Ports; === задаем направления передачи данных
begin          === через порты, а также начальные значения
  DDRC:=$FF;//=== или так DDRC:=%11111111;
  DDRD:=$8f;//=== значение $70 для примера (пересчитать)
  DDRE:=$f0;//=== значение $F для примера (пересчитать)
  PORTE:=%11111111;//=== гасим индикаторы, подавая на
end;           === аноды светодиодов нули через инверторы
//===
procedure Init_Timer0_Async;
begin
  TCCR0:=$0D;//=== биты CS00,02=1(Rпреддел=128),WGM01=1-сброс
счетчика при совп.
  OCR0:=$FF;//=== код совпадения
  ASSR.3:=1; === бит AS0=1 - асинхронный режим Timer0 от кварца
32768Гц

```



```

TIMSK.1:=1;//=== бит OCIE0:=1 - разрешить прерывания при совпадении
текущего
end;      ///<== значения таймера0 с кодом в регистре совпадения OCR0
//=====================================================
var b: boolean;///<== вспомогательная логическая переменная
interrupt TIMER0COMP;///<== обработчик прерывания при совпадении
begin      ///<== кодов TCNT0 и OCR0
  b:=not b;  ///<== меняем состояние
  PORTE.7:= b; ///<== светодиода
end;
//=====================================================
var bz:boolean; jj,ii:byte;///<== ii,jj - вспом. переменные (счетчики циклов)
procedure bzzz;      ///<== boolean bz - GLOBAL ONLY
begin///<== подпрограмма подачи звукового сигнала
  ii:=0;
  loop///<== начало бесконечного цикла
    bz:=not bz;///<== bz - "амплитуда" звука (от 0 до 1) - 01010101.....
    PORTD.7:=bz;///<== выводим "полуволну" - 01 или 10 на пьезодинамик
    for jj:=0 to 250 do endfor;///<== jj=250 - высота звука (длит-сть "полуволны")
    ii:=ii+1;
    if ii=50 then///<== ii=50 - длительность звука
      exitloop;///<== заканчиваем подачу сигнала (выход из бескон. цикла)
    endif;
  endloop;///<== возврат к началу бесконечного цикла
end;
//=====================================================
function Dec2seg (chislo:word):array;
var cifry:array[0..2]of byte; i:byte; ostatok:word;
begin
  i:=0; cifry:=blank;///<== FF FF FF - коды гашения индикаторов
  repeat
    ostatok:=chislo mod 10;///<== вычисляем очередную ДЕСЯТИЧНУЮ
цифру числа
    chislo:=chislo div 10;///<== путем последовательного деления результата
на 10
    if ostatok<>0 then ///<== если остаток не равен 0, то
      cifry[i]:=key8segm[ostatok-1]; ///<== извлекаем из таблицы код цифры от
1 до 9
    else///<== иначе извлекаем из таблицы код цифры 0
      cifry[i]:=key8segm[ostatok+10];///<== цифра 0 не на "своем" месте на
клавиатуре.
    endif;
    Inc(i);///<== переходим к следующей цифре числа (инкремент)
  until chislo<1;///<== до тех пор пока частное не станет < 1

```



```

    return(cifry); //== возвращаем массив трех цифр
end;
//=====
procedure Display8seg; //== однокр. высвечивание на индикаторе 8-ми
сегм. кода
var i,j:byte;running0:byte; //== i,j - вспом. переменные (счетчики циклов)
begin
    running0:=%11110111; //== начальное значение "бегущего нуля"
    for i:=2 downto 0 do //== сканируем 3 линии порта E (PORTE.6 ..
PORTE.4)
        running0:=running0 rol 1; //== готовим вывод в следующий индикатор
        if not b then running0:=running0 and $7F;endif; //== нельзя вмешиваться
        //== в состояние LED (PORTE.7) - управляется обработчиком
TIMER0COMP !!
        PORTC:=s[i]; //== выводим инверсный 8-ми сегментный код цифры на
индикатор
        PORTE:=running0; //== подаем на анод текущего 8-сегм. инд-ра высокий
уровень
        for j:=0 to 255 do endfor; //== для увеличения яркости
    endfor;
end;
//=====
procedure Init_ADC; begin //== настройка АЦП
    ADCSRA:=$xx; //== уст. бит разрешения работы АЦП и Кпреддел = 64
(e.g. =$71)
    ADMUX:=$xx; //== биты АЦП выровнены влево и читать можно один
ADCH (e.g. =$DF)
end;
//=====
interrupt ADCRDY;begin //== обработчик при окончании АЦ
преобразования
    adcode:= word(ADCH); //== читаем только ст. байт кода АЦП (8-ми
битный АЦП)
    ADCSRA.6:=1; //== новый запуск АЦП
end;
//=====
{-----}
{ Main Program }
{$IDATA}
begin
    EnableInts; //== снятие запрета с прерываний (уст. бита I регистра SREG)
    Init_ports; //== настройка линий портов на ввод/вывод
    Init_Timer0_Async; //== настройка таймера0 на асинхр. режим
    Init_ADC; //== настройка АЦП (регистры ADCSRA и ADMUX)


```

```

loop//== начало бесконечного цикла
Wait4KeyPressed;//== ждем нажатия на клавишу
case key of //== выполняем действия в соотв-ии с нажатой клавишей
'0'..'9': //== key - ASCII код клавиши, kn - ее порядковый номер (0..11)
  PORTE:=$BF;//==на аноды заданных. инд-ров подать напр-е (e.g. $40)
  PORTC:=key8segm[kn];//== выводим 8-ми сегм. код нажатой клавиши
  bzzz;//== подача короткого звукового сигнала
| //== этот разделитель обязателен
'#'://== если нажата '#'
  ADCSRA.6:=1; //== подгот-ный пуск АЦП (остальные в обраб-ке
прер-я)
  loop
    s:=Dec2seg(adccode);//== преобразуем код АЦП в 8-ми сегм код
    Display8seg; //== отображаем 8-сегм. код на дисплее
    if ScanKeyOnce(kn)then //== если нажата клавиша,
      ADCSRA.6:=0; //== то остановить АЦП,
      PORTC:=$FF;//== погасить дисплей
      exitloop; //== и выйти из цикла
    endif;
  endloop;
| //== этот разделитель обязателен
endcase;//== конец оператора множественного выбора "case .. of"
endloop;//== возврат к началу бесконечного цикла
end atMega8888KLM.

```

2.2.7 Программирование EEPROM.

7  Следующая модификации программы дает возможность записывать наши действия в ПЗУ (EEPROM). Сначала резервируем 10 байтов для хранения кодов нажатых клавиш “0..9” (наш “бортовой журнал”) в EEPROM (наш “черный ящик”).

Implementation

```

{$EEPROM} //== раздел данных находится во внутренней EEPROM
var
  e2prom: array[0..9] of byte;//== для долговременного (>10лет) хранения данных
{$IDATA} //== раздел данных находится во внутренней памяти МК

```

В следующем фрагменте фиксируем все действия оператора (только для клавиш “0..9”). Для двух остальных клавиш можно предусмотреть аналогичные действия, изменив верхнюю границу массива e2prom до 11-ти. В следующем фрагменте записываем 8-ми сегментный код в EEPROM (или E2PROM) в ячейку с номером равным разности ASCII кода клавиши (key) и ASCII кода нуля (\$30). Например, для клавиши “8”имеющей код key=\$38, разность равна \$38-\$30=8. То есть код клавиши “8” будет отправлен на хранение в ячейку под номером 8 (e2prom[8]).И еще один

нюанс. Теперь код на индикацию отправляется в порт "С" из EEPROM, чтобы продемонстрировать, 1) что она существует и 2) работает исправно.

Последнее замечание. Вы уже наверно заметили (об этом говорилось ранее), что при нажатии на цифровые клавиши переключение светодиода может происходить некорректно. Строка "if not b..." устраняет этот дефект и предотвращает возможное вмешательство в седьмой бит порта "Е", который управляет переключением светодиода в обработчике прерывания "interrupt TIMER0COMP" с помощью логической переменной "b".

```
'0'..'9': //== key - ASCII код клавиши, kn - ее порядковый номер (0..11)
e2prom[key-'0']:=key8segm[kn+1]; //== пишем в EEPROM (E2PROM)
if not b then PORTE:=$BF;else PORTE:=$3F;endif; //==на анод(ы) заданных
//== индикаторов подать напр-е (не трогая светодиод - PORTE.7 !)
PORTC:=e2prom[key-'0'];//== выводим 8-ми сегм. ТЕПЕРЬ ИЗ EEPROM !
bzzzz; //== подача короткого звукового сигнала
```

Другой, универсальный вариант сохранения бита PORTE.7 для всех случаев, когда производится вывод в порт "Е" приведен в приложении 1.

Скомпилируйте программу и загрузите ее кнопкой "**Flash -> Program**" программатора AVRprog. Затем, нажимая на цифровые клавиши убедитесь в том, что их коды теперь записываются и считываются из памяти EEPROM.

Результат покажите преподавателю

2.2.8 Самостоятельное задание.

Последняя модификация программы, самостоятельная (смотри вариант задания). Новые возможности "повесим" на незадействованную пока клавишу "*".

ПРИМЕЧАНИЕ: В работе не рассматривалась работа клавиатуры и ее программирование, так как эти вопросы подробно исследуются в работе №31 "Программирование клавиатуры на ассемблере ASM-51". В настоящей работе управление клавиатурой производится с помощью вспомогательного модуля "KeyLib4PascaL.pas", который дан в приложении 2.

ПРИЛОЖЕНИЯ

Приложение 1. Синхронизация состояния линии порта PORTE.7.

Вводится дополнительная переменная-маска "mask_PE7", в которой старший бит с помощью "исключающего или - xor" переключается в обработчике прерывания TIMER0COMP **синхронно** с переключением

светодиода. Остальные биты маски равны единице. Теперь, если в других разделах программы производится вывод байта "\$XX" в порт "E", например на 8-ми сегментный дисплей, то вывод должен производиться командой "PORTE:=\$XX and mask_PE7" (вместо "PORTE:=\$XX"), что позволяет обходиться без конструкций "if not b then".

```
{ $IDATA }
var s:array[0..2] of byte;key,kn,mask_PE7:byte; adccode:word;

.....

interrupt TIMEROCOMP; //== обработчик прерывания при совпадении
begin //== кодов TCNT0 и OCR0
  b:=not b; //== смена признака присостояния светодиода
  mask_PE7:=mask_PE7 xor $80; //== и инверсия бита маски
  PORTE.7:= b; //== переключение светодиода
end;

.....

Init_ADC; //== настройка АЦП (регистры ADCSRA и ADMUX)
mask_PE7:=$7F; //== начальное значение маски ++++++++
loop //== начало бесконечного цикла

.....

'0'..'9': //== key - ASCII код клавиши, kn - ее порядковый номер (0..11)
  e2prom[key-'0']:=key8segm[kn+1]; //== пишем в EEPROM (E2PROM)
  //if not b then PORTE:=$BF;else PORTE:=$3F;endif; //== на анод(ы) заданных
  PORTE:=$BF and mask_PE7; //== на анод(ы) заданных
  //== индикаторов подать напр-е (не трогая светодиод - PORTE.7 !)

.....

//if not b then running0:=running0 and $7F;endif; //== нельзя вмещиваться
//== в состояние LED (PORTE.7) - управляется обработчиком TIMEROCOMP !!
PORTC:=s[i]; //== выводим инверсный 8-ми сегментный код цифры на индикатор
PORTE:=running0 and mask_PE7; //== подаем на анод 8-с. инд-ра питание ++++

.....
```

Приложение 2. Подпрограммы для работы с клавиатурой

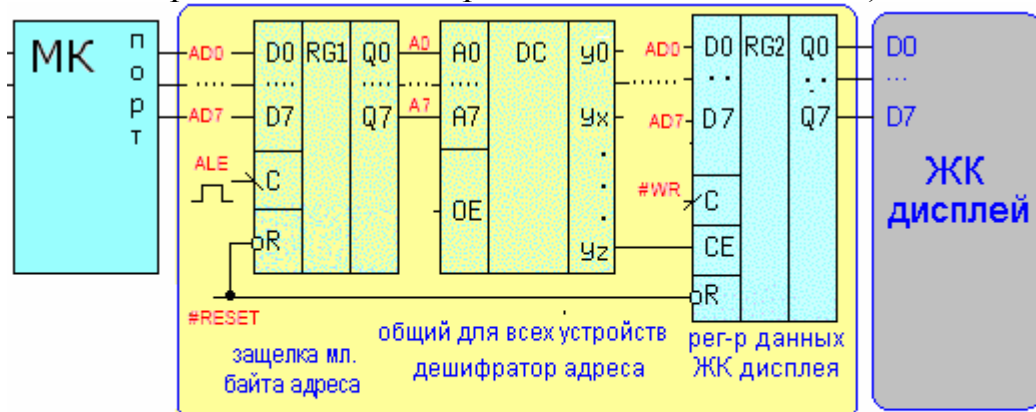
```

//== подпрограммы и функции для работы с клавиатурой 3x4
const keyASCII: string = '123456789*0#'; //== ASCII коды клавиш/цифр
//=====
function ScanKeyOnce (var kn: byte):boolean; //== однокр. сканирование клав-ры
var row,col: byte; //== var kn - параметр переменная № клавиши
var delay:word;
begin
  for col:=0 to 3 do //== сканируем по столбцам матрицы клавиатуры
    PORTD:=not(1 shl col); //== 11111110,11111101,11111011,11110111 бегущий ноль)
    for row:=0 to 2 do //== сканируем по строкам матрицы клавиатуры
      if ((PIND and ($10 shl row))=0) then //== если обнаружен ноль, то
        kn := col*3+row; //== вычисляем порядковый № клавиши (0..11)
      //      for delay:=0 to 5000 do endfor; //== задержка надребезг контактов
      return (true); //== возвращаем бит "обнаружено нажатие клавиши"
    endif;
  endfor;
  return (false); //== возвращаем бит "ни одна из клавиш не нажата"
end;
//=====
procedure Wait4KeyPressed;
begin
  while not ScanKeyOnce(kn) do endwhile; //== ожидаем нажатия на клавишу
  key:=byte(keyASCII[kn+1]); //== +1, т.к. в паскале в 0-м байте -длина строки!
  while ScanKeyOnce(kn) do endwhile; //== ожидаем отпускание клавиши
end;
//=====

```

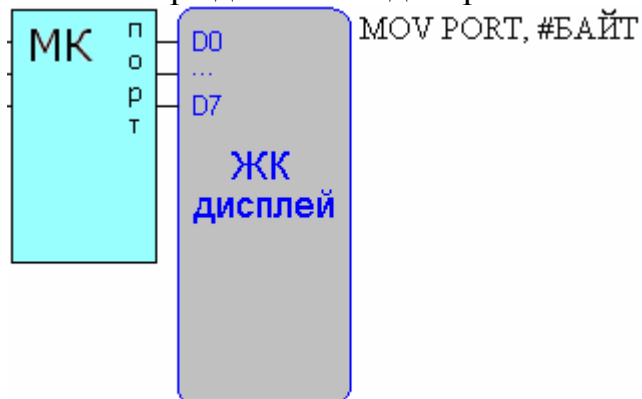
Приложение 3. ШИНА VS ПОРТ

1) Обмен данными с помощью шинного интерфейса (например, вывод байта в порт данных жидкокристаллического дисплея):



```
MOV DPP, #НОМЕР
MOV DPTR, #АДРЕС
MOV A, #БАЙТ
MOV @DPTR, A
```

2) Обмен данными с помощью портов ввода-вывода (например, вывод байта в порт данных жидкокристаллического дисплея):



Вывод в пользу второго варианта напрашивается сам собой. И аппаратных средств меньше и цепочка команд короче. Но не все так просто. Если число требуемых внешних устройств (ЖКД, клавиатура, 8-ми сегментный дисплей, шаговый двигатель, датчики и т. д.) невелико – предпочтительнее вариант с прямым подключением ВУ к портам МК. Если имеющихся портов недостаточно, необходим первый вариант.

-

-
- The diagram illustrates the internal structure of the timer/counter module (TCC0). It includes the following components and connections:
- Inputs:**
 - T_{osc} (oscillator signal) is connected to the **предделитель** (prescaler).
 - $f_{T_{osc}} = 32768 \text{ Гц}$ is noted at the bottom left.
 - Internal Blocks:**
 - предделитель** (prescaler): Receives T_{osc} and outputs to the **счетчик TCNT0**.
 - счетчик TCNT0** (TCNT0 counter): Receives input from the prescaler and outputs to the **компаратор** (comparator).
 - компаратор** (comparator): Receives input from the TCNT0 counter and outputs to the **OCR0** register.
 - OCR0** (Output Compare Register): Receives **код сравнения** (comparison code) and outputs to the **компаратор**.
 - TCCR0** (Timer Counter Control Register): A control register for the module.
 - ASSR** (Asynchronous Status Register): A status register for the module.
 - Outputs:**
 - The **компаратор** outputs a **запрос на прерывание в момент совпадения кодов** (interrupt request at the moment of code coincidence) to the **OCF0** pin.

Регистр управления таймером/счетчиком - TCCR0

Биты **CS02, CS01, CS00** - выбор коэффициента деления предделителя (8..1024)

Биты **WGM00, WGM01** задают режим работы таймер/счетчика (норм., с совп., ШИМ)

Бит **AS0** - выбор источн. имп-ов - системная частота **(0)** или дополн. генератор F=32768 **(1)**

Разряд							OCIE0	
--------	--	--	--	--	--	--	-------	--

Бит **OCIE** – разрешение прерывания при совпадении кодов (OutputCompareInterruptEnable)

В регистр **OCR0** – записывается код, с кот-м сравнивается текущее знач. счетчика.

4. Работа АЦП. Регистр **ADCSRA**:

Разряд	ADEN	ADSC			ADIE	ADPS2	ADPS1	ADPS0
--------	------	------	--	--	------	-------	-------	-------

Бит **ADEN** - разрешение работы АЦП (Enable)

Бит **ADSC** - запуск преобразования АЦП (StartConversion)

Бит **ADIE** - разрешение прерывания АЦП (InterruptEnable)

Биты **ADPS2..0** - биты управления делителем (Prescaler)

Регистр способа выравнивания и выбора входа **ADMUX**:

Разряд			ADLAR				
--------	--	--	-------	--	--	--	--

ADLAR – выравнивание выходного кода влево (LeftArrange)

5. Система прерываний, вектора прерываний. Прерывания **TIMER0COMP** и **ADCRDY**.

6. Программирование 8-ми сегментного дисплея. Сканирование, “бегущий 0”.

7. Программирование EEPROM.

8. Знать и понимать комментарии к окончательной программе (стр. 21-23).


ВАРИАНТЫ ЗАДАНИЙ

<p>Кпреддел = 1024 Тпрерыв = 4 сек</p>  <p>Кпреддел АЦП = 8 Остановить таймер</p>	<p>Кпреддел = 1024 Тпрерыв = 7 сек</p>  <p>Кпреддел АЦП = 16 Запретить прерывания для АЦП</p>	<p>Кпреддел = 128 Тпрерыв = 1 сек</p>  <p>Кпреддел АЦП = 32 Запретить подачу звуковых сигналов</p>
<p>Кпреддел = 256 Тпрерыв = 2 сек</p>  <p>Кпреддел АЦП = 4 Изменить частоту переключ. светодиода</p>	<p>Кпреддел = 64 Тпрерыв = 1/4 сек</p>  <p>Кпреддел АЦП = 128 Прочитать EEPROM</p>	<p>Кпреддел = 128 Тпрерыв = 1/2 сек</p>  <p>Кпреддел АЦП = 64 Остановить таймер</p>

Самостоятельное задание: а) остановить таймер, б) запретить прерывания для АЦП (теперь при вращении ручки потенциометра код не будет изменяться), в) запретить подачу звуковых сигналов (подсказка – направление данных по линии порта), г) изменить частоту переключения системного светодиода (вместо п. а.), д) после выключения питания прочитать ячейки EEPROM и убедиться, что данные не стерты, и т.д...

ЛАБОРАТОРНАЯ РАБОТА № 32 “ЖК ДИСПЛЕЙ”

На рис.1 приведена стандартная схема микроконтроллерной (МК) системы на основе МК ADuC812 семейства MCS-51 (8051). МК или Однокристалльная ЭВМ (ОЭВМ) выполняется в виде большой интегральной микросхемы (БИС), которая содержит необходимое число функциональных устройств для работы в качестве управляющего устройства (контроллера) или ЭВМ. МК содержит процессор, ОЗУ и ПЗУ, набор различных устройств, которые называются периферийными и порты ввода/вывода (ВВ или IO). Стандартный набор периферии (не путать с периферией компьютера, т.е. с устройствами, располагающимися снаружи системного блока) содержит счетчики/таймеры, каналы последовательного обмена данными (например UART), ЦАП, АЦП, компараторы и др. На основе приведенной на рисунке МК системы выполнен универсальный МК комплекс SDK1.1 (в дальнейшем УМК).

 1 Смысл этого значка спросите у преподавателя.

На рис.1 выводы элементов схемы, имеющие одинаковые обозначения соединены вместе. Два регистра защелки адреса RG вместе с совмещенной шиной адрес/данные образуют стандартный шинный интерфейс (системная шина или просто шина), к которому могут подключаться дополнительные устройства, не показанные на рис.1. В УМК к шине подключена программируемая логическая интегральная схема (ПЛИС), а к ней уже подключены клавиатура, светодиоды, жидкокристаллический индикатор, звуковой излучатель и параллельный 16-ти битный двунаправленный порт).

Все МК семейства MCS-51 имеют одинаковую базовую систему памяти:

- внутреннюю память программ (Flash, EPROM или ROM - CODE),
- внутреннюю память данных - DATA,
- внешнюю память программ и/или данных (ВПП - CODE, ВПД - XDATA).

Flash-память ADuC812 равна 8КБ (0..1FFF). Внутренняя программная память может отсутствовать. В большинстве случаев ВПП, ВПД ограничены объемом 64КБ (0..FFFF).

В некоторых МК верхний предел памяти превышает 64КБ, например в ADuC812 объем ВПД может достигать 16МБ (объем ВПП по-прежнему 64 КБ). Механизм обращения к внутренней программной памяти разработчику недоступен.

Обращение к ВПД производится с использованием стандартных управляющих сигналов: стробов чтения/записи ($\sim RD$, $\sim WR$). Выборка команд из ВПП производится с помощью специального строба чтения $\sim PSEN$ (Program Store Enable).

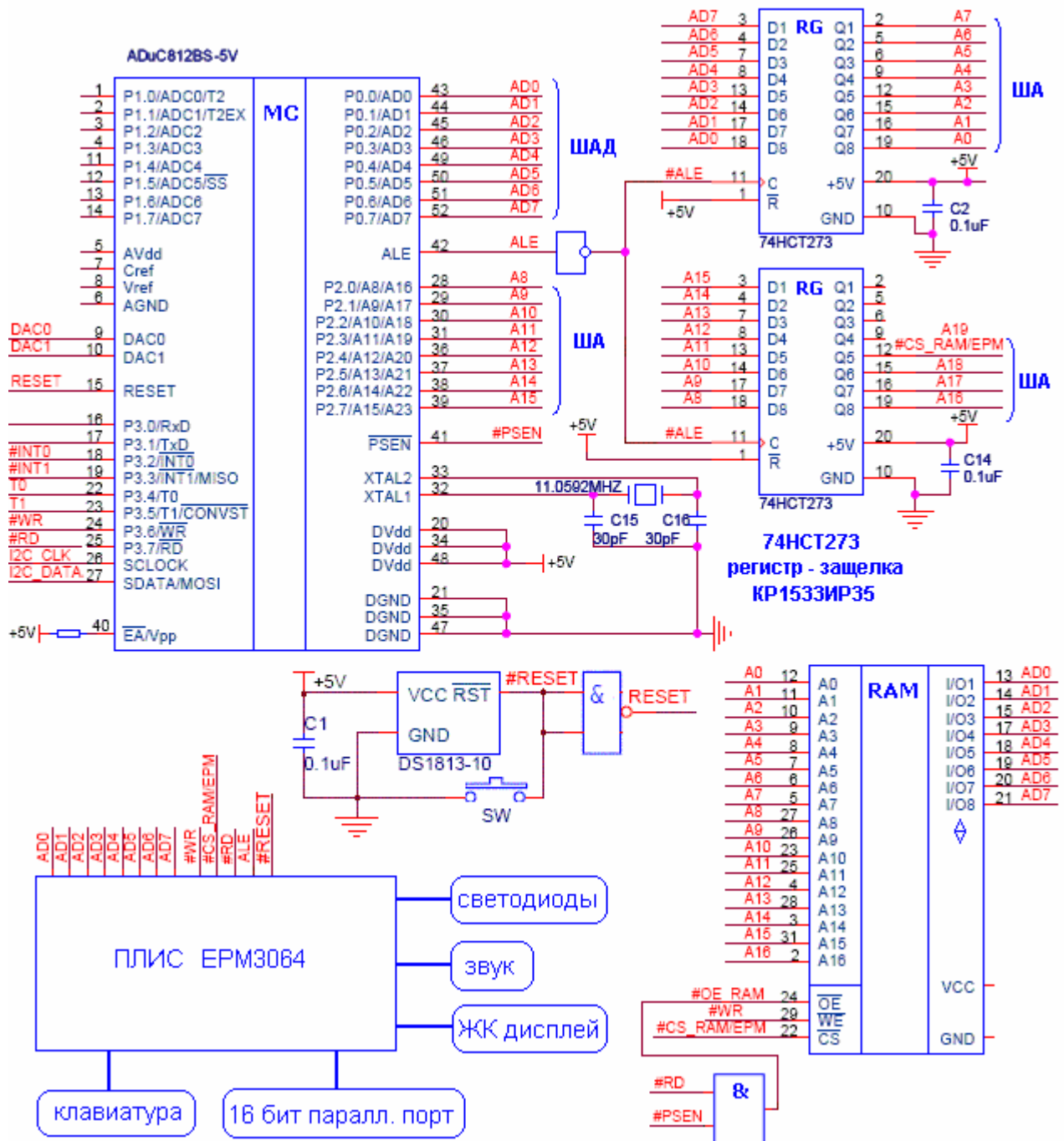


Рис.1 Принципиальная схема УМК

На рис.2 приведена карта стандартного распределения памяти в МК семейства MCS-51.

В тех случаях, когда объема внешней памяти достаточно и для программного кода и для данных, можно использовать совмещенное адресное пространство и заводить на микросхему памяти все три указанных stroba. Однако, у микросхем памяти для этих целей только два входа: \sim WE (Write Enable) для stroba \sim WR и \sim OE (Output Enable) для stroba \sim RD. Стандартным решением является объединение двух strobov чтения (\sim RD и \sim PSEN) по “ИЛИ” с помощью дополнительного логического элемента “И” (см. рис.1). В УМК используется совмещенное адресное пространство ВПП + ВПД (рис.2).

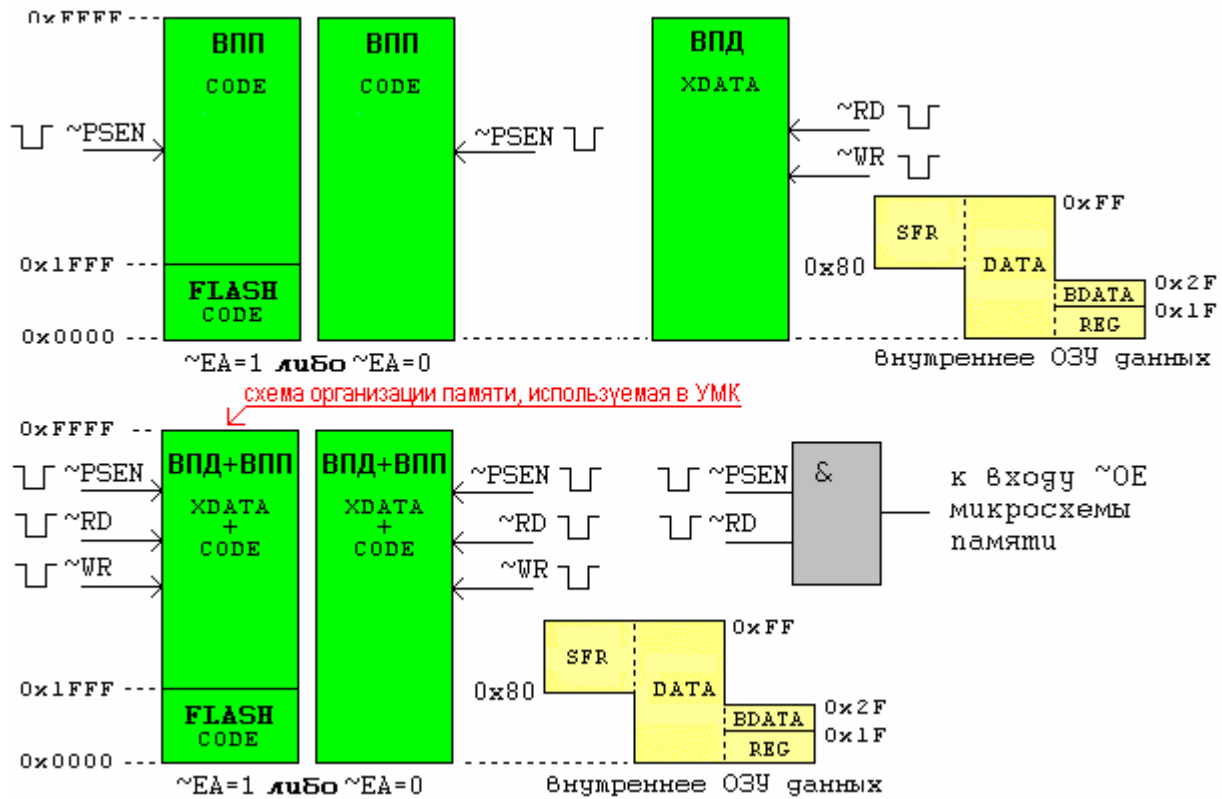


Рис.2

Внутренняя память данных - DATA (256 байт) делится на две равные части. В младших 128 байтах (0..7F) могут располагаться до 4-х 8-ми байтовых банков рабочих регистров – REG и область, в которой можно хранить однобитовые переменные – BDATA. Далее в этой области программист размещает стек, переменные, массивы и другую оперативную информацию. К старшим 128 байтам (80..FF) при необходимости можно обращаться, как к регистрам специальных функций – SFR (Special Function Register). SFR по сути являются регистрами управления и данных многочисленной периферии МК. Если периферия не используется, что маловероятно, то верхнюю половину области DATA можно также использовать для хранения оперативных данных.

По возможности нужно стараться располагать стек и данные в области DATA, так как доступ к ней производится значительно быстрее, чем к области XDATA (eXternal DATA).

Использовать или нет внутреннюю программную память разработчик решает с помощью сигнала (External Access Enable) на инверсном входе ~EA. Если на входе ~EA=1, то МК может выбирать команды, как из внутренней памяти (флэш), так и из внешней памяти программ (ОЗУ или ПЗУ). Если ~EA = 0, внутренняя флэш память МК недоступна.

Отладку и пробные пуски программ удобно вести с использованием совмещения ВПД + ВПП при ~EA=1. Во флэш-памяти в этом случае располагается резидентный загрузчик пользовательской программы, а во

внешнее ОЗУ загружается разрабатываемая пользовательская (целевая) программа. Когда пользовательская программа окончательно написана и отлажена ее можно записать во флэш-память МК вместо резидентного загрузчика и/или во внешнее ПЗУ. В схеме на рис.1 задействовано только внешнее ОЗУ, т.к. УМК в лабораторных работах служит для разработки и отладки пользовательских (целевых) программ.

Помимо внутреннего оперативного ЗУ данных (DATA) и программной флэш-памяти (CODE) в ADuC812 встроена (не оперативная) электрически перепрограммируемая память EEPROM, доступ к которой производится только через регистры SFR.

Кроме того в МК ADuC812 реализован механизм страничной памяти (256 страниц по 64КБ = 16МБ). На рис.2 показана начальная (нулевая) страница (только для ВПП или ВПП + ВПД). Остальные 255 страниц предназначены только для внешней памяти данных (ВПД) или для регистров внешних устройств.

На рис.2-1 приведена структурная схема МК ADuC812. Все выводы портов P0..P3 имеют альтернативные функции, например выводы приемопередатчика - RxD и TxD, входы АЦП – ADCi, стробы чтения/записи - \sim RD и \sim WR, выходы шины адреса – Ai, двунаправленные выводы совмещенной шины адрес/данные – ADi и т.д (см. рис.1).

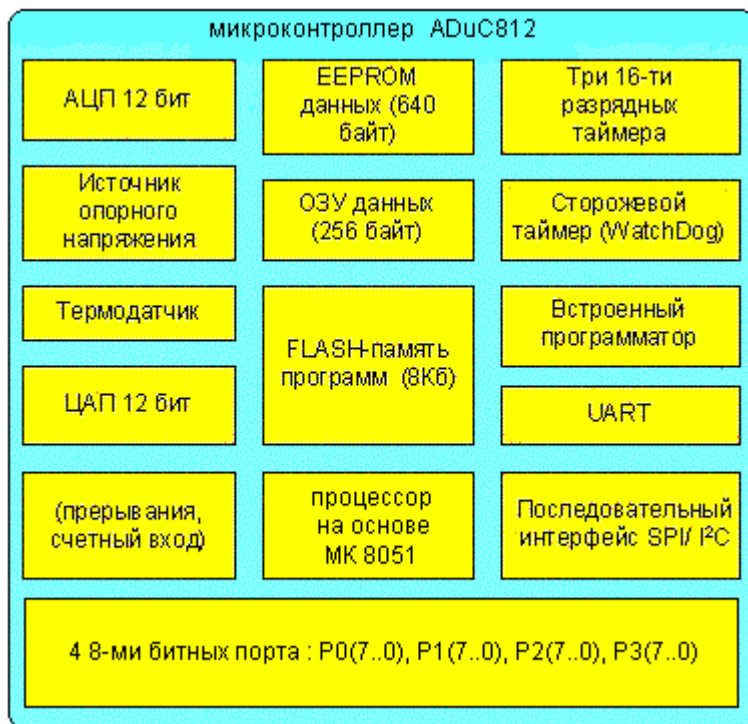


Рис.2-1

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Написать программу, управляющую работой двух стандартных систем ввода-вывода: 1) ЖК символьного дисплея и 2) АЦ и ЦА

преобразователей. На этой базе спроектировать простейший вольтметр.

РАСЧЕТ АДРЕСОВ РЕГИСТРА УПРАВЛЕНИЯ И РЕГИСТРА ДАННЫХ ЖК СИМВОЛЬНОГО ДИСПЛЕЯ

В простых схемах, когда достаточно свободных выходов портов, дисплей на жидких кристаллах (ЖКД или LCD-Liquid Crystall Display) подключается к одному или двум свободным портам. В тех системах (УМК), где число свободных выходов МК ограничено, для подключения дополнительных внешних устройств может использоваться шинный интерфейс.

Например, в УМК подключение дополнительных внешних устройств осуществляется к программируемой логической интегральной схеме (ПЛИС) с требуемым числом выводов. Сама ПЛИС подключается к совмещенной шине адрес/данные ШАД (AD0..AD7) МК системы. На ПЛИС также подаются некоторые стандартные управляющие сигналы (стробы ALE, #WR, #RD, сигнал #RESET и др.). Следует отметить, что в некоторых схемах активный инверсный уровень сигнала в тексте отмечается символом '#' вместо '~'. Ниже на рис.3 приведен фрагмент схемы, запрограммированной в ПЛИС EPM3064.

ВНИМАНИЕ: 1) *Схема скорректирована* для большего числа вариантов заданий. Часть входов элемента "И" – *инверсные*.

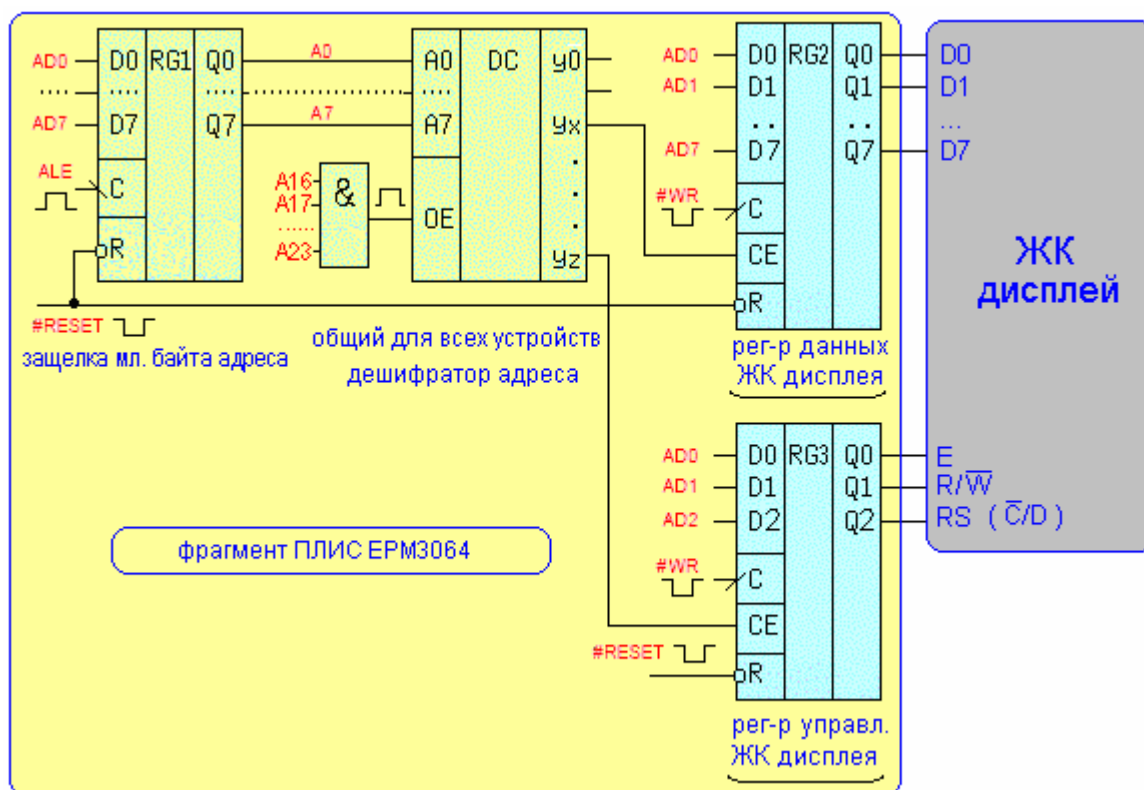



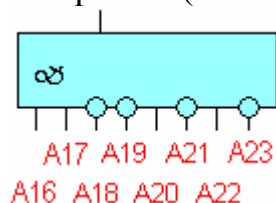
Рис.3

Назначение выводов ЖКД. D7..D0 - информационный порт. E - вход строб-импульса для защелкивания байта данных во внутреннем регистре

данных ЖКД. $R/\sim W$ – признак записи или чтения байта данных ($R/\sim W=1$ – чтение, $R/\sim W=0$ – запись). RS он же $\sim C/D(\sim \text{Control/Data})$ – бит, определяющий, что передается через порт $D7..D0$: байт данных при $\sim C/D=1$ или управляющий работой ЖКД байт при $\sim C/D=0$. Естественно, что при $R/\sim W=0$ и $\sim C/D=0$ считывается информация о состоянии ЖК дисплея.

Необходимо рассчитать адреса, но теперь уже **двух регистров**: регистра данных RG2 и регистра управляющих сигналов RG3 (не путать с регистром управления – регистром специальных функций SFR, находящимся в недрах самого ЖКД).

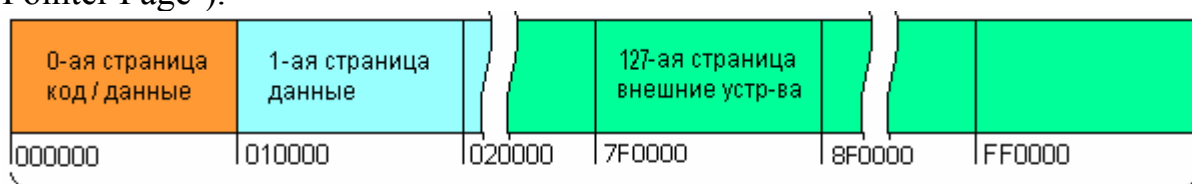
1  **Рассмотрим вариант задания:** 1) Задействован 62-й выход дешифратора, т.е. $Y_x=Y_{62}$, 2) Четыре входа элемента “И” A_{22}, A_{20}, A_{17} и A_{16} – прямые, остальные – инверсные (как показано на рисунке).



Расчет адреса регистра RG2.

Во время выполнения команды ассемблера **“MOVX @DPTR, A”** младший байт адреса ШАД (AD_7, AD_6, \dots, AD_0) записывается в регистре RG1 стробом адреса ALE (A_7, A_6, \dots, A_0). Далее, дешифратор адреса формирует один из 256-х управляющих сигналов ($y_0..y_{255}$), каждый из которых поступает на вход разрешения того или иного устройства (CE, CS и т.д.). На рис.3 сигнал Y_x разрешает выходы по входу CE (Crystall Enable он же Chip Select) регистра RG2. Далее за стробом ALE следует строб записи данных $\#WR$, который защелкивает в RG2 информационный байт $D7..D_0$, поступающий по ШАД.

Рассчитаем адрес для обращения к регистру RG2. 16 младших битов адреса ячейки внешней памяти XRAM или ВУ в пределах одной страницы памяти (64KB) хранятся в двухбайтовом регистре DPTR=(DPH и DPL ‘Data Pointer’) микроконтроллера. Номер текущей страницы (8 старших битов адреса) находится в однобайтовом регистре DPP (‘Data Pointer Page’).



распределение адресного пространства внешней памяти XDATA ADuC812 в УМК
(256 страниц по 64 КБ)

Принимая во внимание, что на выходе ЛЭ “И” единица будет при единичных значениях на прямых входах и нулевых на инверсных входах получим, что $A_{23}, A_{22}, A_{21}, A_{20}, A_{19}, A_{18}, A_{17}, A_{16} = 01010011(\text{BIN}) = 53(\text{HEX})$. Выход Y62 дешифратора будет активирован, когда на его адресных входах будет соответствующий двоичный код ($62_{10} = 00111110_2 = 3E_{16}$). Тогда для нашей конкретной схемы (см. рис.3) и варианта задания запишем адрес регистра RG2:

A23	A22	A21	A20	A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	1	0	1	0	0	1	1	X	X	X	X	X	X	X	X	0	0	1	1	1	1	1	0
53								3E															
регистр DPP (ADuC812)								регистр DPTR (ADuC812)															

Как обычно, неиспользуемые биты ША могут быть любыми, например нулевыми и адрес RG2 будет в этом случае равен 0101 0011 0000 0000 0011 1110 = **53003E(HEX)** или (DPP)=**53h**, (DPTR)=**003E=3E**.

Расчет адреса регистра RG3 (производится аналогично).

#####

ВНИМАНИЕ: Вам необходимо рассчитать **ТОЛЬКО** свои значения **2-х адресов** регистров, загружаемых в DPTR в соответствии с полученным заданием. Адрес страницы памяти, загружаемой в DPP, будет определен во включаемом файле и рассчитывать его **НЕ НУЖНО**. В программе эти 2 значения имеют символические обозначения - **DATA_LCD** и **CNTR_LCD**.



#####

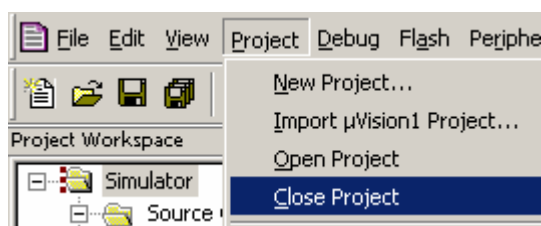
Если вы еще не получили задание - **ТРЕБУЙТЕ** его у преподавателя.

РАЗРАБОТКА ПРОГРАММЫ

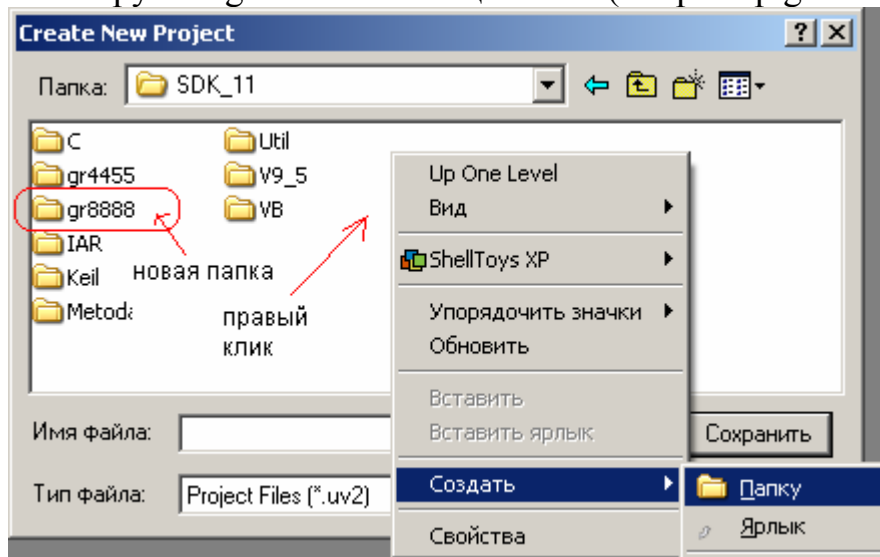
1) Создание шаблона программы на ассемблере.

Запустите интегрированную среду разработки (IDE) для МК семейства

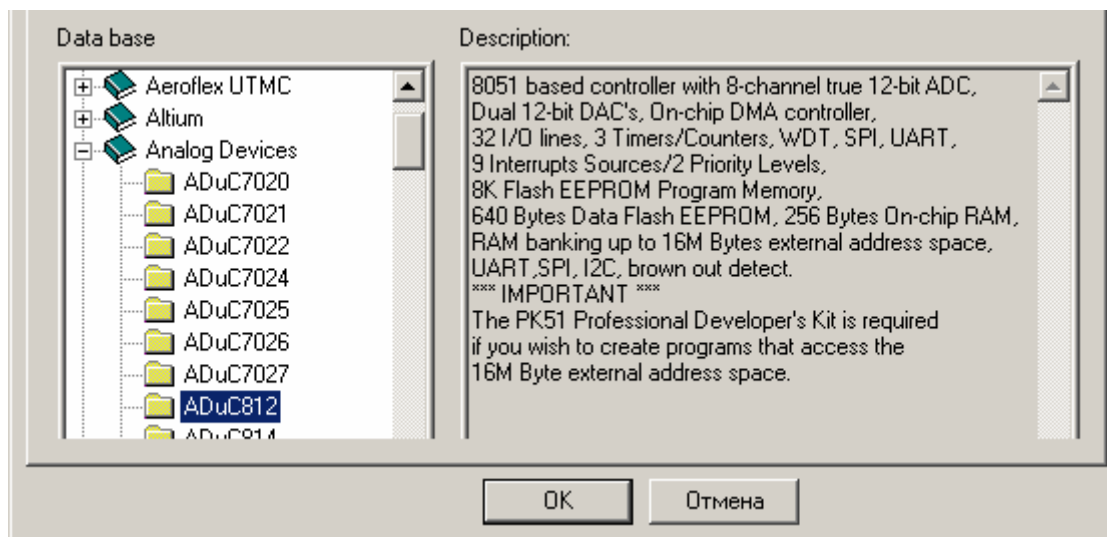
MCS-51 “Keil uVision”  или  в зависимости от версии. Обычно при запуске открывается предыдущий проект, поэтому закройте его из основного меню “Project | Close Project”.



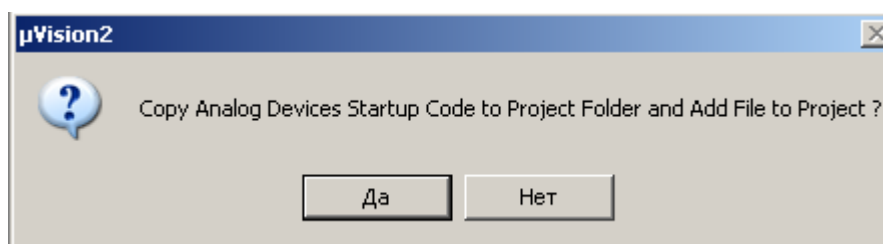
В основном меню выберите п. “Project | New Project...”. Появится диалоговое окно “Create New Project”. Перейдите в папку “C:\EMUL\Work\SDK_11\”, кликните правой кнопкой в пустом поле диалога и в появившемся контекстном меню создайте новую папку с номером своей группы grXXXX и инициалами (например gr8888PAN).



Войдите в созданную папку и введите имя файла проекта, например My_LCD затем кликните по кнопке “Сохранить”. Откроется диалоговое окно “Select Device for Target ...”.



Выбираем МК ADuC812 и жмем на “OK”. На появившееся предложение



даем ОТРИЦАТЕЛЬНЫЙ ответ, т.к. программа будет написана на ассемблере и стартовый модуль для языка “С” не понадобится. Все от начала и до конца программы придется написать самим, что полезно для понимания взаимодействия программных и аппаратных средств, в реальной разработке микроконтроллерной системы.

На этом этапе рабочее поле проекта будет выглядеть примерно, как на рис.4.

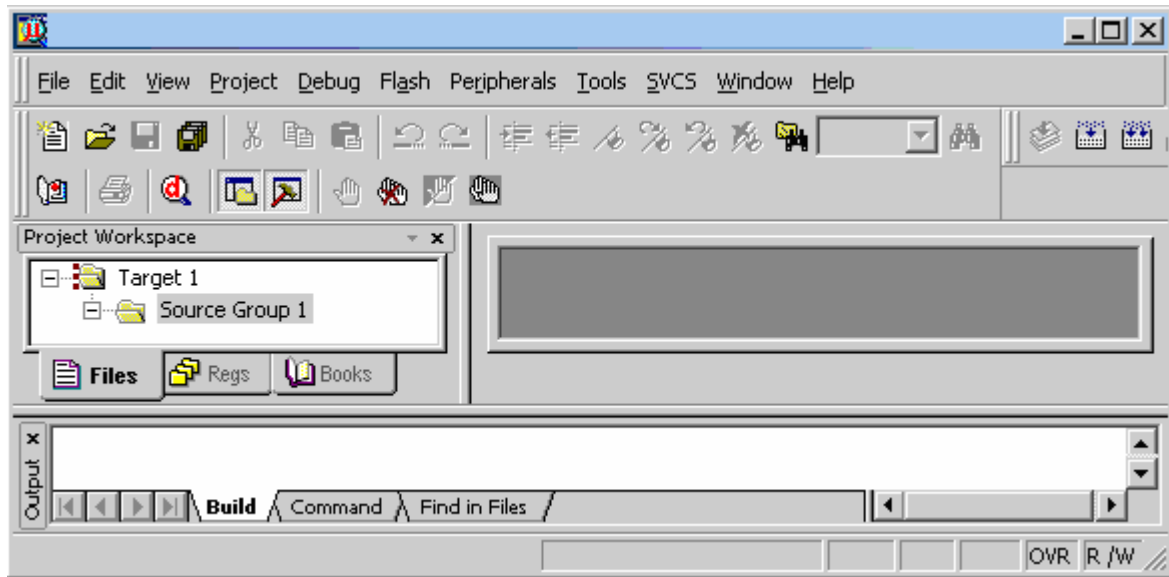
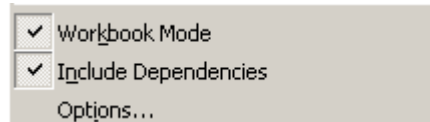


Рис.4


Создадим в проекте основной ассемблерный рабочий файл с расширением *.a51. Для этого из п. основного меню “File | New” создадим текстовый файл с именем “Text1” по умолчанию. Это не то, что нам нужно, поэтому с помощью п. основного меню “File | Save As” в появившемся диалоговом окне записываем имя файла, например “My_LCD.a51” (не пропустите точку и расширение “a51”). Теперь добавим этот файл в проект. Для этого в левом окне на странице “Files” кликаем правой кнопкой сначала по “Source Group 1” и затем в диалоговом окне “Add Files to Group ...” по “My_LCD.a51”.

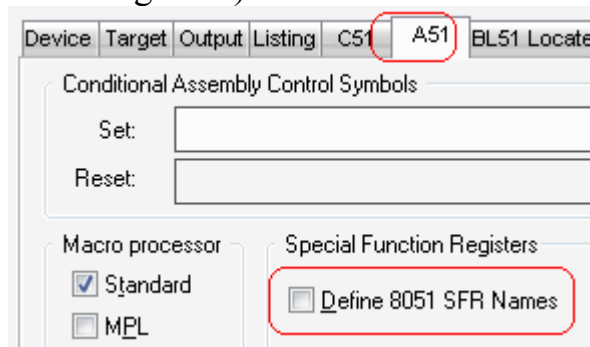
Нажимаем на кнопки “Add” и “Close”. Отметим, что в левой панели в группе файлов сразу же появится новый файл “ My_LCD.a51”, пока пустой.


Теперь можно сохранить все файлы проекта с помощью п. меню “File | Save All”. Делать это желательно, почаще. Для удобной навигации по файлам проекта (для каждого файла своя закладка) убедитесь, что в п. меню “View” отмечен режим “Workbook Mode”




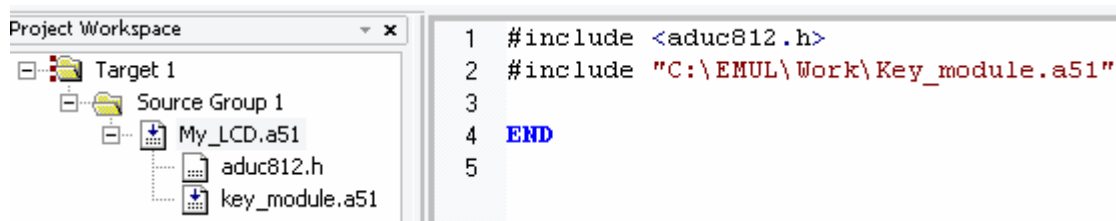
В поздних версиях IDE Keil uVision этот режим устанавливается из п. "Edit | Configuration | Editor".

Далее нам понадобится файл с привязкой внутренних ресурсов МК к внутренним адресам. Сначала отключим файл с базовыми (неполными определениями), сняв "птичку" с "Define 8051...". Кнопка  (или п. меню "Project | Options for Target...").



Затем в рабочий файл "My_LCD.a51" записываем строку: `#include <aduc812.h>` и директиву `END`. Строка `#include....` включает в проект, такой же файл, но с расширенными для МК aduc812 определениями, как показано на рисунке. Теперь нажмите на кнопку  и файл "aduc812.h" будет включен в проект.

Выполнение работы будет завязано на использование клавиатуры и светодиодов, подпрограммы и макроопределения, для которых, разработанные нами в предыдущих лабораторных работах №30 и №31 также включим в проект. Поэтому добавьте еще одну строку в рабочий файл `#include "C:\EMUL\Work\Key_module.a51"`. Еще раз воспользуйтесь кнопкой  и рабочее поле проекта будет иметь следующий вид.



II). Разработка программы управления ЖК дисплеем и ЦАП-АЦП.

Символьные ЖКД служат для отображения символов ASCII (в т.ч. расширенных) с помощью матрицы точек 5x8 или 5x10.

2 ЖКД имеет встроенный контроллер и три блока памяти: **DDRAM** (Data Display RAM) – **рабочая** память данных, куда пользователь записывает коды, отображаемых на дисплее символов, **CGROM** (Character Generator ROM) – ПЗУ знакогенератор, здесь хранятся образы символов (программисту обычно недоступна) и **CGRAM** (Character Generator RAM) – ОЗУ знакогенератор, область, в которую программист может записывать коды дополнительных (отсутствующих в CGROM) символов.

В таблице 1 приведены образы символов размером 5 x 8 точек, хранящиеся в ПЗУ (CGROM) ЖКД, используемого в УМК. Цветом выделены 8 ячеек ОЗУ (CGRAM) ЖКД, в которые пользователь может записать коды дополнительных символов.

Таблица 1

Upper 4 bit Lower 4 bit	LLLL	LLHH	LLHL	LLHH	LHLL	LHLH	LHHL	LHHH	HLLL	HLLH	HLHL	HLHH	HHLL	HHHL	HHHH
LLLL	CG RAM (1)														
LLHH	CG RAM (2)														
LLHL	CG RAM (3)														
LLHH	CG RAM (4)														
LHLL	CG RAM (5)														
LHLH	CG RAM (6)														
LHHL	CG RAM (7)														
LHHH	CG RAM (8)														
HLLL	CG RAM (1)														
HLLH	CG RAM (2)														
HLHL	CG RAM (3)														
HLHH	CG RAM (4)														
HHLL	CG RAM (5)														
HHHL	CG RAM (6)														
HHHL	CG RAM (7)														
HHHH	CG RAM (8)														

ОБРАЗ

КОД

00

1B

02

00

04

11

0E



00

Объем памяти данных DDRAM составляет 80 байт (80 символов). В двухстрочных 16-ти и 20-ти символьных ЖКД адрес крайней левой

позиции верхней строки (знакоместа) равен 00h, адрес крайней левой позиции нижней строки = 40h.

Таблица 2

Таблица 2 (соответствие между адресами DDRAM и позициями ЖКИ)																		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0f	1	1	1	1
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	0	1	2	3
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4f	5	5	5	5
0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	0	1	2	3

CGROM может хранить образы символов размером 5 x 8 и 5 x 10 точек на основе нестандартного 8-ми битного кода. Первые 128 символов (латинские буквы, цифры, знаки препинания, арифметические знаки и некоторые другие) соответствуют стандартным ASCII кодам. Во второй половине русские буквы находятся на местах не соответствующих ни одной из расширенных кодировок. Это значит, что записав в DDRAM код 'F' мы увидим в соответствующей позиции ЖКД ее изображение  (ASCII код = LHLL LNHLL = 0100 0110 = 46h). Буквы русского алфавита находятся "не на своих местах". Например буква 'Б' в таблице CGROM имеет код = HLHL LLLL = 1010 0000 = A0h, который не соответствует ни DOS(CP-866) ни WIN(CP-1251) и никаким другим кодировкам буквы 'Б'. Например в WIN кодировке код 'Б' = c1h. Поэтому при записи в программе вывода на ЖКД буквы 'Б', на дисплее отобразится символ , в соответствии с кодировкой буквы 'ш' в CGROM. В таких случаях, в программе для вывода на дисплей символов с нестандартными кодами, например буквы 'Б' придется записать не символ 'Б', а его CGROM код 'A0'.

Помимо приведенных в таблице CGROM символов, пользователь может "нарисовать" до 8-ми своих собственных и разместить их в CGRAM. Например, для "смайла" (см. таблицу 1) пользователь должен записать 8 указанных байтов (в которых значащими являются только 5 младших битов – матрица 5x8) по одному из адресов CGRAM (0..7)

Большинство ЖКД имеют встроенный контроллер типа 44780 (Hitachi). Команды (управляющие байты) для этого контроллера приведены в таблице 3. В максимальной конфигурации управление производится с помощью 8-ми выводов: RS(~Control/Data – признак управляющего байта или байта данных), RW(Read/~Write – признак чтения/записи) и E (строб), а также порта данных D7..D0. По линии E – подается строб-импульс. RS определяет, чем будут обмениваться контроллер и МК: управляющей информацией при RS=0 или байтом данных RS=1. Сигнал RW или точнее R/~W, естественно определяет, что производится: запись или чтение.

Таблица 3										
RS (~C/D)	R/~W	D7	D6	D5	D4	D3	D2	D1	D0	Команда/функции
0	0	0	0	0	0	0	0	0	1	Очистка дисплея
0	0	0	0	0	0	0	0	1	x	Возвращение курсора в исходное состояние
0	0	0	0	0	0	0	1	ID	S	Задание направления перемещения курсора
0	0	0	0	0	0	1	D	C	B	Разрешение отображения курсора
0	0	0	0	0	1	SC	RL	x	x	Смещение курсора / сдвиг изображения на дисплее
0	0	0	0	1	DL	N	F	x	x	Сброс/задание параметров интерфейса
0	0	0	1	A	A	A	A	A	A	Перевод курсора в CGRAM
0	0	1	A	A	A	A	A	A	A	Перевод курсора на экран по адресу A..A
1	0	H	H	H	H	H	H	H	H	Запись символа в текущую позицию курсора
0	1	BF	x	x	x	x	x	x	x	Проверка признака "занято"
1	1	H	H	H	H	H	H	H	H	Считывание символа, указываемого курсором

- **ID** - автоинкрементирование позиции курсора
- **S** - сдвиг изображения после записанного байта
- **D** - включение / отключение дисплея (1/0)
- **C** - разрешение / запрет использования курсора (1/0)
- **B** - задание / отмена режима мерцания курсора (1/0)
- **SC** - разрешение / запрещение сдвига изображения (1/0)
- **RL** - направление сдвига: вправо / влево(1/0)
- **DL** - разрядность данных: 8/4 бита (1/0)
- **N** - число строк изображения: 1/2 (0/1)
- **F** - размер знакоместа: 5x10/5x8 (1/0)
- **BF** - флаг занятости: равен 1 в процессе обработки данных ЖК дисплеем

- **А** – адрес
- **Н** – данные

Управляющие сигналы RS, RW и E записываются пользователем в разряды D2, D1, D0 регистра RG3 (рис.3 и 5). Байты данных или управляющие байты записываются в регистр RG2 (рис.3 и 5).



Рис.5

На рисунке 6 приведен фрагмент схемы подключения ЖКД к шинному интерфейсу с помощью двух регистров.

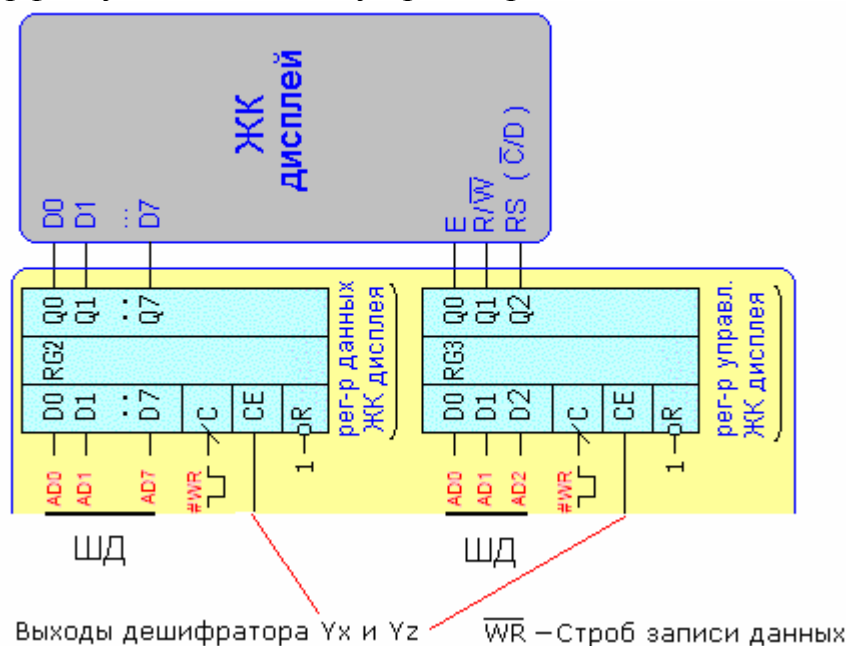


Рис. 6

Вывод байта (байта управления или байта данных) в ЖКД, как видно из рисунка осуществляется через регистр RG2 командой ассемблера “movx @DPTR, a”, в момент выполнения которой адрес @DPTR декодируется дешифратором (рис.3). С выхода дешифратора сигнал Y_x поступает на вход CE (Chip Enable он же Chip Select) и разрешает запись данных.

Следом, положительный фронт строба записи $\sim WR$ записывает (защелкивает) байт из аккумулятора микроконтроллера в регистр. В этот же момент байт с шины данных окажется на входах D7..D0 ЖКД, но будет записан и отображен, только после подачи строба записи на управляющий вход E ЖКД.

II-1). Разработка программы, отображающей на ЖК дисплее ОДИН символ.

2 Начнем с определения вспомогательных программных средств для записи управляющих сигналов E,R/~W и ~C/D. По справочным данным ввод некоторых команд в LCD сопровождается временными задержками. Поэтому сначала введем в текст программы вспомогательное макроопределение временной задержки с регулируемой длительностью и несколько переменных.

Категорически рекомендуется использовать отступления и выделять структурные блоки (иначе выявление ошибок затруднится). На данном этапе текст программы будет выглядеть следующим образом (комментарии писать не обязательно).

```
#include <ADUC812.h>;== файл с определениями адресов регистров МК
#include "C:\EMUL\Work\Key_module.a51";== всё для работы с кл-рой


;=====
Tochka_Vhoda EQU 4000h,== адрес входа в основную программу
;=====

DSEG,== сегмент данных
DL1:      DS 1,== счетчик циклов для макроса Delay
DL2:      DS 1,== еще один
ctrl1:    DS 1,== код на управляющих входах ЖКД
dno_steka: DS 1,== нужен ТОЛЬКО адрес - dno_steka
;=====
Delay MACRO Dmax1,Dmax2,== Dmax1=1(0.0005сек) при Dmax2=229
    LOCAL L1,L2,== объявление локальных меток
    mov DL1,#Dmax1,== константа для внешнего цикла
L1: mov DL2,#Dmax2,== константа для внутреннего цикла (подобрана)
L2: djnz DL2,L2,== (L2:djnz DL2,L2 - вложенный цикл)
    djnz DL1,L1,== (L1:..djnz DL1,L1 - внешний цикл)
    ENDM,== конец макроопределения
;=====

CSEG at 0,== начало сегмента программы
jmp Tochka_Vhoda,== переход к началу программы
ORG Tochka_Vhoda,== с этого адреса располагается код программы
;====main====main====main====main====main====main====
main_prog: ;== точка входа в основную программу
    mov sp,#dno_steka,== записываем адрес начала стека в регистр sp
    Delay 10,229,== задержка на 5 миллисекунд
    nop,== только для точки останова
;====endmain====endmain====endmain====endmain====
END
```

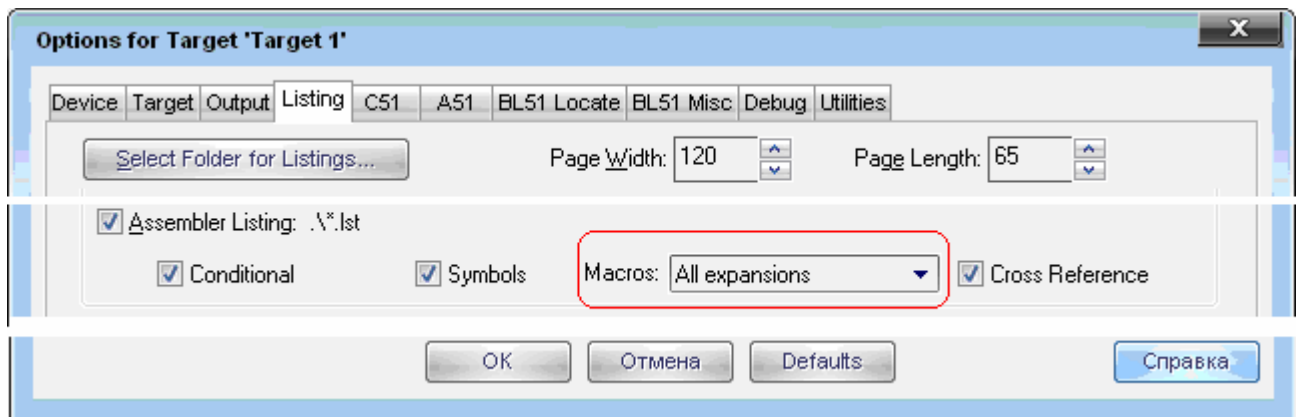
Программные задержки обычно формируются с помощью одного или нескольких циклов. В последнем случае циклы являются вложенными. Задержка равна сумме времени выполнения команд не входящих в циклы (в нашем случае “mov DL1,#Dmax1”) и времени работы самих циклов с учетом числа их повторений.



Подберем значения Dmax1 и Dmax2 для задержки ~5 мсек (в программе уже подобраны). Зная длительность выполнения команд и

число их повторений можно вычислить точное значение задержки. Но проще это сделать в отладчике-симуляторе. Оттранслируйте программу  и убедитесь, что ошибок нет.

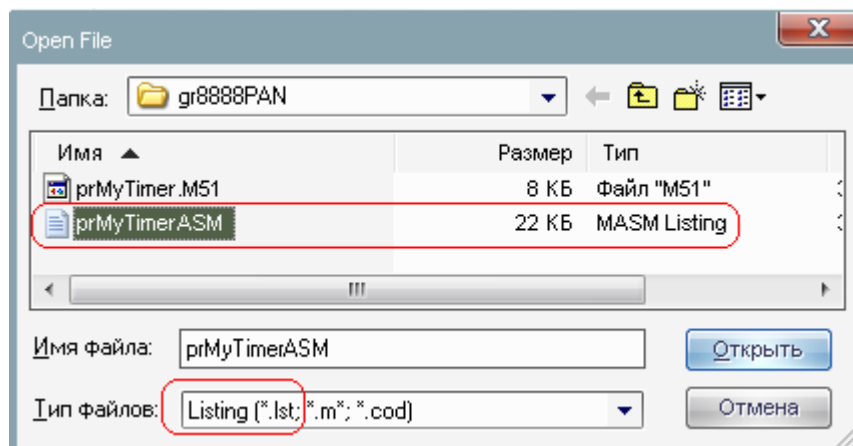
0 Error(s) , 0 Warning(s) .

В противном случае ошибки (особенно если они внутри макросов) проще устранить по описанной уже методике. Отметив в п. меню “Project/Options for Target...” п. “Macros: All Expansions” получим возможность ‘отлавливать’



синтаксические ошибки в листинге программы (файл с расширением *.lst). К сожалению, в исходном тексте точное местоположение ошибки внутри тела макроса транслятор не указывает. Если листинг не обновляется, попробуйте запустить трансляцию программы кнопкой , а не .

После трансляции курсор не укажет на строку с ошибкой в исходном тексте макроса, а только в месте его вызова. Если макросов много и они вложены друг в друга, поиск ошибки будет сильно затруднен. Откройте файл листинга с расширением *.LST из п. меню “File | Open...”.




Найдите строку с ошибкой. Поиск по служебной строке “ERROR” (в примере ошибка в имени DPPP вместо DPP)..

```

4009                               198+1      mov DPPP,#0;== 7fh0bdh7fh3fh
***                               ^
*** ERROR #A45 IN 198 (prMyTimerASM.a51, LINE 18): UNDEFINED SYMBOL

```


Вернитесь к исходному тексту сделайте исправление(я) и СНОВА оттранслируйте программу.

Теперь перейдите в окно отладчика  или по команде “CNTR+F5” или из п. меню “Debugger”. Двойным кликом установите точку прерывания (breakpoint) на команде “nop”, т.е. после выхода из макроса (также можно клавишей “F9” или из п. меню “Debug...”). Номера строк 24 и 25 в вашем случае могут быть и другие.


```

24      Delay 10,229,;== задержка на 5 миллисекунд
25      nop,;== только для точки останова

```

Затем кнопкой  установите курсор на строке Delay... Запомните значение счетчика секунд.

states	4	24	mov sp,#dno_steka;
sec	0.00000434	25	Delay 10,229,;== за
psw	0x00	26	nop,;== только для

Кнопкой  продолжите выполнение. Программа сама доберется до точки останова. Обратите внимание на счетчик секунд.

states	4626	24	mov sp,#dno_steka;
sec	0.00501953	25	Delay 10,229,;== за
psw	0x00	26	nop,;== только для

Разница значений $0.00501953 - 0.00000434 = 0.00501519$ сек есть длительность выполнения макроса задержки “Delay”. Погрешность около 0.25%, что в большинстве случаев достаточно.

Если нужна супер-точность, то используются таймеры (см. лаб. раб. №30) или кропотливая подгонка числа команд и учет их длительности в циклах и вне их.

В программе также понадобится задержка в 15 мсек. Подберите нужное значение параметра Dmax1 и убедитесь в правильности выбора с помощью отладчика. Запишите или запомните это значение.
 #####

Теперь можно записать макрос для вывода управляющих сигналов и строба в порт управления ЖКД.

```

    djnz DL1,L1,==(L1:..djnz DL1,L1 - внешний цикл)
    ENDM,== конец макроопределения
;=====
Strobe MACRO Data1_Cntr0,== бит RS он же D/~C
    mov ctrl, #Data1_Cntr0,== определяемся: данные или управление
    orl ctrl, #0x01,== передний фронт строб-импульса (бит D0=1 "E")
    Write ctrl, CNTR_LCD,== управл. код подаем на управл. входы ЖКД
    anl ctrl, #0xFE,== задний фронт строб-импульса (бит D0=0 "E")
    Write ctrl, CNTR_LCD,== управл. код подаем на управл. входы ЖКД
    Delay 10,229,== задержка на 5 мсек (по справочнику)
    ENDM
;=====
CSEG at 0,== начало сегмента программы

```

Т.к. читать состояние бита занятости ЖКД не требуется (задержки на выполнение операций, только что рассчитаны), то сигнал R/~W = 0 и передавать его в макрос не нужно. Имя параметра “Data1_Cntr0” подсказывает, что при его значении = 1, через порт данных в ЖКД выводится байт **данных**, а при Data1_Cntr0 = 0 через порт данных в ЖКД выводится **управляющий** байт (команда). Снова оттранслируйте программу и убедитесь в отсутствии ошибок. Добавьте в программу новую порцию программного кода.

```

;=====
Tochka_Vhoda EQU 4000h,== адрес входа в основную программу
DATA_LCD EQU 0FEh,== адрес регистра данных RG2 ЖКД (рис.3)
CNTR_LCD EQU 0F9h,== адрес регистра управл. RG3 ЖКД (рис.3)
;=====
DSEG,== начало сегмента данных

```

Адреса регистров RG2 и RG3 - DATA_LCD и CNTR_LCD вами **УЖЕ РАССЧИТАНЫ** и имеют другие значения.

```

    Delay 10,229,== задержка на 5 мсек (по справочнику)
    ENDM
;=====
LCDctrl MACRO ch,== макрос вывода управл. байта в контроллер ЖКД
    Write ch, DATA_LCD,== упр. байт код подаем на входы данных ЖКД
    Strobe 0x00,== строб+управл. код подаем на управл. входы ЖКД
    ENDM,== 00h - бит D2(~C/D=0) признак записи упр. байта
;=====
LCDchar MACRO ch,== макрос вывода расшир. ASCII кода на ЖКД
    Write ch, DATA_LCD,== ASCIIp код подаем на входы данных ЖКД
    Strobe 0x04,== строб+управл. код подаем на управл. входы ЖКД
    ENDM,== 04h - бит D2(~C/D=1) признак записи байта данных
;=====
LCDinit MACRO,== начальная настройка ЖКД (запись управляющих байтов)
    IRP tt, <#0c7h,#0feh,#0f3h,#0f9h>,== встроенный макрос повторения команд
        LCDctrl tt,== будут 4 раза повторены эти команды (tt=#0xc7,#0xfe,...)
    ENDM,== конец встроенного макроса
    ENDM,== конец макроопределения LCDinit
;== 0c7h - 8-ми битный режим загрузки байтов в ЖКД
;== 0feh - очистка дисплея
;== 0f3h - включить дисплей
;== 0f9h- автоинкремент позиции курсора
;=====
CSEG at 0,== начало сегмента программы

```

Два макроса для вывода байта с символическим именем “ch” достаточно прокомментированы.

#####

ВНИМАНИЕ: В макроопределении “LCDinit” **четыре** указанных управляющих байта **ДОЛЖНЫ** быть вами **ПЕРЕСЧИТАНЫ**, в соответствии с таблицей 3, которая была приведена ранее.

#####

Например, для расчета управляющего байта автоинкремента позиции курсора находим внизу таблицы строчку “ID автоинкрементирования позиции курсора”. Далее находим третью строку в таблице, в которой встречается бит ID: 0 0 0 0 0 1 ID S - где S бит смещения изображения. В нашей задаче – бит S=0, поэтому легко вычисляем значение этого управляющего байта – **6h** (или просто 6). Это рассчитанное значение, должно быть записано в макросе, вместо неправильного значения **0f9h**.

Таким же образом найдите 3 оставшихся управляющих байта (команды) ЖКД, принимая во внимание, что:

- разрядность данных ЖКД – 8 бит
- число строк в дисплее – 2
- матрица точек – 5x8
- мерцание курсора во-первых отключено
- а во-вторых его использование запрещено

Рассчитанные значения запишите вместо неправильных. И последний штрих – введите последние на данном этапе строчки кода.

```
main_prog: ;== точка входа в основную программу
    mov sp,#dno_steka;== записываем адрес начала стека в регистр sp
    // Delay 10,229;== задержка на 5 миллисекунд
    // nop;== только для точки останова
    Delay 30,229;== задержка на 15мсек после включения
    call InitALL
    LCDchar #'K';== вывод на дисплей ASCII кода буквы 'K'
    88888888;== начало бесконечного цикла
    jmp 88888888;== конец бесконечного цикла
;====endmain====endmain====endmain====endmain====
;== ДАЛЕЕ РАЗМЕЩАЮТСЯ ПОДПРОГРАММЫ ==

InitALL:
    mov DPP,#DPP_page_addr;== адрес страницы, на кот. находятся BV
    LCDinit
    ret
;=====
END
```

Две ненужных теперь команды закомментируйте или лучше всего удалите. Первый аргумент макроса Delay вы уже рассчитали. Макрос LCDinit полезно поместить в подпрограмму инициализации всего и вся “InitALL”, в которую по мере “обрастания” будем добавлять и другие компоненты. В настоящий момент текст программы должен иметь следующий вид.

```

#include <ADUC812.h>;== файл с определениями адресов регистров МК
#include "C:\EMUL\Work\Key_module.a51";== всё для работы с
клавиатурой
;=====
Tochka_Vhoda EQU 4000h;== адрес входа в основную программу
DATA_LCD EQU 0FEh;== адрес регистра данных RG2 ЖКД (рис.3)
CNTR_LCD EQU 0F9h;== адрес регистра управл. RG3 ЖКД (рис.3)
;=====
DSEG;== начало сегмента данных
DL1:      DS 1;== счетчик циклов для макроса Delay
DL2:      DS 1;== еще один
ctrl:     DS 1;== код на управляющих входах ЖКД
dno_steka: DS 1;== нужен ТОЛЬКО адрес - dno_steka
;=====
Delay MACRO Dmax1,Dmax2;== Dmax1=1(0.0005сек) при Dmax2=229
    LOCAL L1,L2;== объявление локальных меток
    mov DL1,#Dmax1;== константа для внешнего цикла
L1:  mov DL2,#Dmax2;== константа для внутреннего цикла
L2:  djnz DL2,L2;==(L2:djnz DL2,L2 - вложенный цикл)
     djnz DL1,L1;==(L1:..djnz DL1,L1 - внешний цикл)
    ENDM;== конец макроопределения
;=====
StrobE MACRO Data1_Cntr0;== бит RS он же D/~C
    mov ctrl, #Data1_Cntr0;== определяемся: данные или управление
    orL ctrl, #0x01;== передний фронт строб-импульса (бит D0=1 "E")
    Write ctrl, CNTR_LCD;== управл. код подаем на управл. входы ЖКД
    anL ctrl, #0xFE;== задний фронт строб-импульса (бит D0=0 "E")
    Write ctrl, CNTR_LCD;== управл. код подаем на управл. входы ЖКД
    Delay 10,229;== задержка на 5 мсек (по справочнику)
    ENDM
;=====
LCDctrl MACRO ch;== макрос вывода управл. байта в контроллер ЖКД
    Write ch, DATA_LCD;== упр. байт код подаем на входы данных
ЖКД
    StrobE 0x00;== строб+управл. код подаем на управл. входы ЖКД
    ENDM;== 00h - бит D2(~C/D=0) признак записи упр. байта
;=====
LCDchar  MACRO ch;== макрос вывода расшир. ASCII кода на ЖКД
    Write ch, DATA_LCD;== ASCIIp код подаем на входы данных ЖКД
    StrobE 0x04;== строб+управл. код подаем на управл. входы ЖКД
    ENDM;== 04h - бит D2(~C/D=1) признак записи байта данных
;=====

```

LCDinit MACRO;== начальная настройка ЖКД (запись управляющих байтов)

IRP tt, <#0c7h,#0feh,#0f3h,#0f9h>;== встроенный макрос повторения команд

LCDctrl tt;== будут 4 раза повторены эти команды (tt=#0xc7,#0xfe,...)

ENDM;== конец встроенного макроса

ENDM;== конец макроопределения LCDinit

;== 0c7h - 8-ми битный режим загрузки байтов в ЖКД

;== 0feh - очистка дисплея

;== 0f3h - включить дисплей

;== 0f9h- автоинкремент позиции курсора

=====

CSEG at 0;== начало сегмента программы

jmp Tochka_Vhoda;== переход к началу программы

ORG Tochka_Vhoda;== с этого адреса располагается код программы

=====main=====main=====main=====main=====main=====main

main_prog: ;== точка входа в основную программу

mov sp,#dno_steka;== записываем адрес начала стека в регистр sp

Delay 30,229;== задержка на 15мсек после включения

call InitALL

LCDchar #'K';== вывод на дисплей ASCII кода буквы 'K'

_88888888;== начало бесконечного цикла

jmp _88888888;== конец бесконечного цикла

=====endmain=====endmain=====endmain=====endmain=====

;== ДАЛЕЕ РАЗМЕЩАЮТСЯ ПОДПРОГРАММЫ ==

InitALL;== п/п инициализации всего и вся

mov DPP,#DPP_page_addr;== адрес страницы, на кот. находятся ВУ

LCDinit

ret

=====

END


#####

ВНИМАНИЕ: В вашем варианте 8 операндов, выделенных жирным шрифтом, необходимо пересчитать в соответствии с заданием (в том числе вместо 'K' должен отображаться другой символ) .

#####

Оттранслируйте программу  и убедитесь, что ошибок нет.

III) Пробный запуск программы.

 **3**Создадим загрузочный (или рабочий) файл программы, в котором каждый байт программы записывается в виде двух ASCII

шестнадцатеричных цифр и имеет расширение “HEX”. Ниже показан **фрагмент Intel’овского HEX файла**. Файл состоит из однотипных строк, например:


```
:0F40000075847F90003F74BDF075840080FE32A0
```

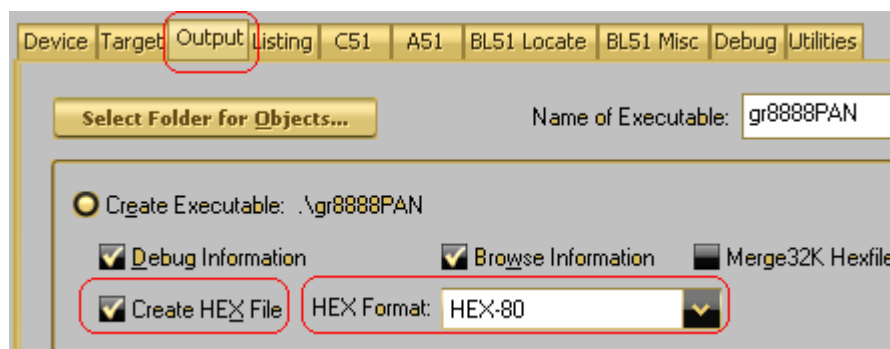
```
.....
```

```
:00000001FF
```



Каждый байт машинного кода записывается двумя ASCII символами, соответствующим двум 16-ным(HEX) цифрам, двух тетрад каждого байта. Каждая строчка начинается с двоеточия “:”. Рассмотрим первую строчку. Следующие после двоеточия 2 символа, например 0F(15) обозначают, количество информационных байтов в строке. Следующие 4 символа, 4000 являются двухбайтовым адресом, с которого будут располагаться 15 байтов программы текущей строки в памяти (ОЗУ) целевой платы. После адреса следуют два служебных символа – 00(строка данных) или 01(завершающая строка). Последние 2 символа в строке A0 – байт дополнения контрольной суммы всех предыдущих байтов строки до 256-ти. Оставшиеся байты – информационные. В первой строке примера их 15(0F): 75 84 7F 90 00 3F 74 BD F0 75 84 00 80 FE 32.

Специальная строка: 00000001FF завершает информационные строки. Последнюю нестандартную строку вида 02XXXX060000SS со стартовым адресом XXXX программа-загрузчик дописывает к HEX-файлу самостоятельно, т.к. она не генерируется транслятором. В этой строке 02 код команды перехода LJMP XXXX. XXXX – 16-ный адрес (Tochka_Vhoda), с которого будет стартовать целевая программа. 060000 – служебные символы. SS - дополнение контрольной суммы всех предыдущих байтов строки до 256-ти. В нашем примере XXXX=4000h.


3 Вызовите окно мастера шаблонов кнопкой  (или п. меню “Project | Options for Target...”) и отметьте на странице “Output” формат файла “HEX-80”.

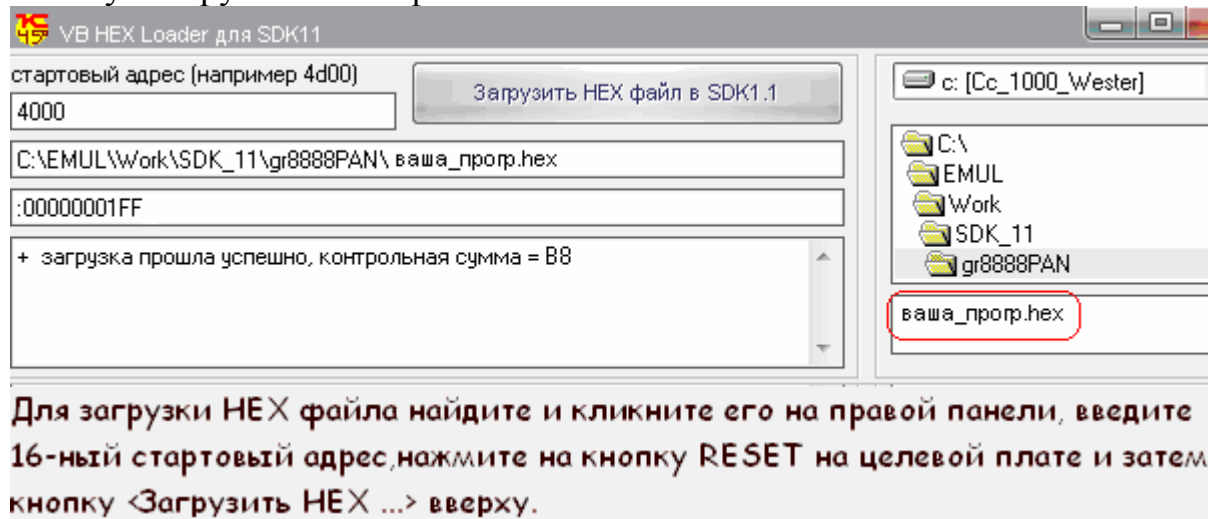


После этого выйдите из диалога, нажав на кнопку “OK”.

Теперь, создайте загрузочный HEX-файл программы, для чего нажмите на кнопку “Build Target”  или  “Rebuild All ...” на панели инструментов или в п. меню “Project | Build Target”. Создать загрузочный файл можно и с помощью “горячей” клавиши <F7>. Если трансляция и

компоновка прошли успешно, то появится уже знакомое сообщение “0 Error(s)”.

Произведем первый пробный запуск программы. **ВНИМАНИЕ:** перед каждой загрузкой нужно нажать на кнопку “**RESET**” в левом нижнем углу рабочего стенда. Загрузим полученный HEX-файл в УМК с помощью, разработанного нами (в лабор. работе №10) инструментального загрузчика . Копия загрузчика лежит в папке “C:\EMUL\Work\SDK_11\W_hex202.exe”. Запустите загрузчик и в появившемся окне отыщите свой файл с расширением “*.hex”, кликните по нему, задайте свой вариант стартового адреса **4000** и нажмите на кнопку “Загрузить HEX-файл...”.



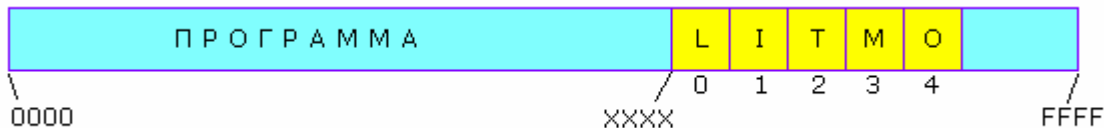
Дождитесь появления сообщения “... загрузка прошла успешно...”. На экране ЖКД в левом верхнем углу должен высветиться приведенный в задании символ, что и требовалось на данном этапе. **В противном случае необходимо пересчитать указанные 8 операндов программы.** Закрывать окно загрузчика “VB HEX Loader ...” не нужно, т.к. он еще понадобится.

Результат покажите преподавателю

IV). Разработка шаблона программы с меню выбора действий

Выводить на дисплей длинные последовательности с помощью “LCDchar” неудобно, поэтому создадим макроопределение вывода строки произвольной длины “LCDstrn” и заодно макрос для перемещения курсора в любую позицию дисплея. Признаком окончания строки является нулевой байт (код ASCII = 0).

Приведенный ниже фрагмент поместите в указанное место программы. Операнды 40h и 80h вычислены по таблицам 1 и 2. Следующий рисунок, на котором изображена нулевая страница памяти, иллюстрирует действие команды “MOVC A,@A+DPTR”.



(DPTR) = XXXX - адрес начала строки
 (аккумулятор) = 3 - смещение или номер символа в строке
 (DPTR) + (аккумулятор) = XXXX + 3 - адрес буквы 'M'
 MOVC A, @A + DPTR - извлекает байт с адресом XXXX + 3 и помещает его в аккумулятор

```

;== 0f9h- автоинкремент позиции курсора
;=====
LCDstrn MACRO addr,;== вывод строки символов на ЖК дисплей
    LOCAL povtor,next,vyhod,;== addr - адрес строки
    mov cnt,#0,;== начало строки ( cnt - смещение символа в строке)
povtor:
    mov a,cnt,;== текущий номер выводимого символа
    mov DPTR,#addr,;== в цикле т.к. LCDchar переопределяет DPTR
    movc a,@a + DPTR,;== @a + DPTR - адрес символа в строке
    cjne a,#0,next,;== если текущий символ=0 - то конец строки
    jmp vyhod,;== и закончить вывод
next:
    LCDchar acc,;== если нет - вывести на ЖКД очередной символ
    inc cnt,;== перейти к следующему символу
    jmp povtor,;== и повторить вывод
vyhod:
    ENDM
;=====
LCDxy MACRO x,y,;== макрос позиционирования курсора ЖКД
    LOCAL up,;== метка up - верхняя строка
    mov a,#y,;== Y-координата строки 0-верхняя, 1-нижняя
    jz up,;== если Y=0, значит строка верхняя и ее адрес=0
    mov a,#40h,;== если Y=1, строка нижняя и ее адрес = 40h
up: add a,#x,;== прибавляем позицию в строке к ее адресу
    add a,#80h,;== старш. бит упр. байта адреса позиции=1
    Write acc, DATA_LCD,;== переводим курсор
    Strobe 0x00,;== в указанную поз-ю
    ENDM
;=====
CSEG at 0,;== начало сегмента программы

```

В макросе “LCDstrn” появилась переменная “cnt” хранящая смещение символа в строке, которую нужно добавить в раздел переменных.

```

ctrl:      DS 1,;== код на управляющих входах ЖКД
cnt:       DS 1,;== вспомогательная переменная - счетчик
dno_steka: DS 1,;== нужен ТОЛЬКО адрес - dno_steka

```

Теперь две подпрограммы: вывода меню и очистки экрана. Внизу приведены строки меню. Для иллюстрации, во второй строке выводится слово “АЦП”, но так как позиции русских букв в расширенной таблице не совпадают со стандартными кодировками, приходится вводить текст кодами из таблицы 32.1. Например: код буквы ‘П’ = 0xA8. Вставьте

указанные фрагменты в программу и русифицируйте часть меню в соответствии со своим заданием, например ADCpusk → АЦПпуск. Обратите также внимание, что строки должны завершаться нулем, иначе "LCDstrn" начнет "вываливать" на экран "мусор" и/или соседние строки, пока не наткнется на случайный 0.

```

LCDinit;== настройка ЖКД
;=====
menu;== подпрограмма вывода меню действий
LCDxy 0,0;== xy=00 - верхняя строка, слева
LCDstrn menu1;== выводим на ЖКД первую строку меню
LCDxy 0,1;== xy=01 - нижняя строка, слева
LCDstrn menu2;== выводим на ЖКД вторую строку меню
ret
;=====
LCDclear;== очистка экрана ЖКД
LCDctrl 1
ret
;=====
menu1: DB "0-clear *-menu",0
menu2: DB "#-ADCpusk D-",0e1h,"A",0a8h,0;==0e1h,"A",0a8h - ЦАП
adc_pusk: DB "# - PUSK ADC",0
dac_code: DB "D - vyberite code",0
;=====
END

```

Последнее дополнение - в основной программе выводим меню. Ненужную строку можно закомментировать или удалить.

```

call InitALL;== начальная настройка
// LCDchar #'K';== вывод на дисплей ASCII кода буквы 'K'
call menu;== текст меню действий - на экран дисплея
_88888888;== начало бесконечного цикла
jmp _88888888;== конец бесконечного цикла

```

На этом этапе программа должна иметь следующий вид.

```

#include <ADUC812.h>;== файл с определениями адресов регистров МК
#include "C:\EMUL\Work\Key_module.a51";== всё для работы с клав-рой
;=====
Tochka_Vhoda EQU 4000h;== адрес входа в основную программу
DATA_LCD EQU 0FEh;== адрес регистра данных RG2 ЖКД (рис.3)
CNTR_LCD EQU 0F9h;== адрес регистра управл. RG3 ЖКД (рис.3)
;=====
DSEG;== начало сегмента данных
DL1: DS 1;== счетчик циклов для макроса Delay
DL2: DS 1;== еще один
ctrl: DS 1;== код на управляющих входах ЖКД
cnt: DS 1;== вспомогательная переменная - счетчик
dno_steka: DS 1;== нужен ТОЛЬКО адрес - dno_steka
;=====
Delay MACRO Dmax1,Dmax2;== Dmax1=1(0.0005сек) при Dmax2=229
LOCAL L1,L2;== объявление локальных меток

```

```

mov DL1,#Dmax1;== константа для внешнего цикла
L1: mov DL2,#Dmax2;== константа для внутреннего цикла
L2: djnz DL2,L2;==(L2:djnz DL2,L2 - вложенный цикл)
    djnz DL1,L1;==(L1:..djnz DL1,L1 - внешний цикл)
ENDM;== конец макроопределения
;=====
;
StrobE MACRO Data1_Cntr0;== бит RS он же D/~C
    mov ctrl, #Data1_Cntr0;== определяемся: данные или управление
    orL ctrl, #0x01;== передний фронт строб-импульса (бит D0=1 "E")
    Write ctrl, CNTR_LCD;== управл. код подаем на управл. входы ЖКД
    anL ctrl, #0xFE;== задний фронт строб-импульса (бит D0=0 "E")
    Write ctrl, CNTR_LCD;== управл. код подаем на управл. входы ЖКД
    Delay 10,229;== задержка на 5 мсек (по справочнику)
ENDM
;=====
;
LCDctrl MACRO ch;== макрос вывода управл. байта в контроллер ЖКД
    Write ch, DATA_LCD;== упр. байт код подаем на входы данных
ЖКД
    StrobE 0x00;== строб+управл. код подаем на управл. входы ЖКД
    ENDM;== 00h - бит D2(~C/D=0) признак записи упр. байта
;=====
;
LCDchar    MACRO ch;== макрос вывода расшир. ASCII кода на ЖКД
    Write ch, DATA_LCD;== ASCIIp код подаем на входы данных ЖКД
    StrobE 0x04;== строб+управл. код подаем на управл. входы ЖКД
    ENDM;== 04h - бит D2(~C/D=1) признак записи байта данных
;=====
;
LCDinit MACRO;== начальная настройка ЖКД (запись управляющих
байтов)
    IRP tt, <#0c7h,#0feh,#0f3h,#0f9h>;== встроенный макрос повторения
команд
        LCDctrl tt;== будут 4 раза повторены эти команды (tt=c7,fe,f3,f9)
        ENDM;== конец встроенного макроса
    ENDM;== конец макроопределения LCDinit
;=====
;
LCDstrn MACRO addr;== вывод строки символов на ЖК дисплей
    LOCAL povtor,next,vyhod;== addr - адрес строки
    mov cnt,#0;== начало строки ( cnt - смещение символа в строке)
povtor:
    mov a,cnt;== текущий номер выводимого символа
    mov DPTR,#addr;== в цикле т.к. LCDchar переопределяет DPTR
    movc a,@a + DPTR;== @a + DPTR - адрес символа в строке
    cjne a,#0,next;== если текущий символ=0 - то конец строки
    jmp vyhod;== и закончить вывод
next:

```

```

LCDchar acc;== если нет - вывести на ЖКД очередной символ
inc cnt;== перейти к следующему символу
jmp povtor;== и повторить вывод
vyhod:
    ENDM
;=====
LCDxy MACRO x,y;== макрос позиционирования курсора ЖКД
    LOCAL up;== метка up - верхняя строка
    mov a,#y;== Y-координата строки 0-верхняя, 1-нижняя
    jz up;== если Y=0, значит строка верхняя и ее адрес=0
    mov a,#40h;== если Y=1, строка нижняя и ее адрес = 40h
up:    add a,#x;== прибавляем позицию в строке к ее адресу
    add a,#80h;== старш. бит упр. байта адреса позиции=1
    Write acc, DATA_LCD;== переводим курсор
    Strobe 0x00;== в указанную поз-ю
    ENDM
;=====
    CSEG at 0;== начало сегмента программы
    jmp Tochka_Vhoda;== переход к началу программы
    ORG Tochka_Vhoda;== с этого адреса располагается код программы
;====main====main====main====main====main====main
main_prog: ;== точка входа в основную программу
    mov sp,#dno_steka;== записываем адрес начала стека в регистр sp
    Delay 30,229;== задержка на 15мсек после включения ЖКД
    call InitALL;== начальная настройка
    call menu;== текст меню действий – на экран дисплея
_88888888;== начало бесконечного цикла
    jmp _88888888;== конец бесконечного цикла
;====endmain====endmain====endmain====endmain====
;== ДАЛЕЕ РАЗМЕЩАЮТСЯ ПОДПРОГРАММЫ ==
InitALL;== п/п инициализации всего и вся
    mov DPP,#DPP_page_addr;== адрес страницы, на кот. находятся ВУ
    LCDinit;== настройка ЖКД
    ret
;=====
menu;== подпрограмма вывода меню действий
    LCDxy 0,0;== xy=00 - верхняя строка, слева
    LCDstrn menu1;== выводим на ЖКД первую строку меню
    LCDxy 0,1;== xy=01 - нижняя строка, слева
    LCDstrn menu2;== выводим на ЖКД вторую строку меню
    ret
;=====
LCDclear;== очистка экрана ЖКД
    LCDctrl 1

```

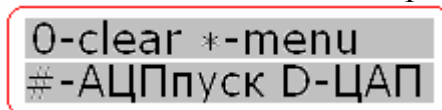
```

ret
;=====
menu1:    DB "0-clear *-menu",0
menu2:    DB "#-ADCpusk D-",0e1h,"A",0a8h,0;==0e1h,"A",0a8h - ЦАП
adc_pusk: DB "# - PUSK ADC",0
dac_code: DB "D - vyberite code",0
;=====
END

```


ВНИМАНИЕ: Одну из записей в меню, в соответствии с заданием необходимо русифицировать, например **ADCpusk** в **АЦПпуск**.
 #####

Оттранслируйте и загрузите программу. В следующем разделе 4-ре нижних клавиши будут задействованы для выбора задач.




Результат покажите преподавателю

V). Разработка модуля простейшего цифрового вольтметра с использованием ЦАП и АЦП.

В основной программе предусмотрим 4-ре задачи (действия):

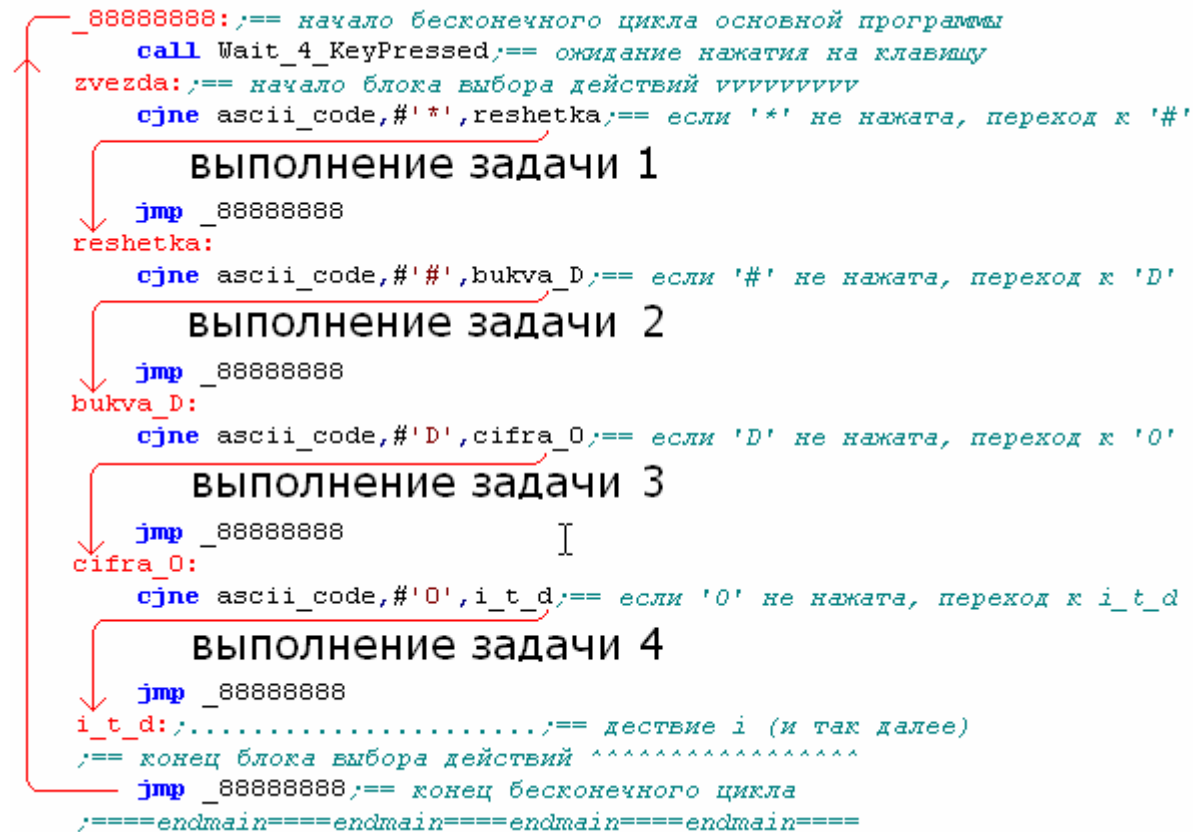
- вывод меню по нажатию на клавишу '*'
- очистка экрана – клавиша '0'
- изменение напряжения с помощью ЦАП - клавиша 'D'
- измерение напряжения с использованием АЦП – клавиша '#'

Для этой цели используем 4-ре указанных клавиши и библиотеку макроопределений и подпрограмм, разработанную в лаб. работе №31.

 **4** В соответствии с поставленной задачей модернизируем основную программу (см. рисунок-диаграмму внизу). Операторы осуществляющие выбор действий помещены внутрь бесконечного цикла: “_88888888: jmp _88888888” Первая команда “Wait_4_KeyPressed” вызывает подпрограмму ожидания нажатия на клавишу (находится в файле “Key_module.a51”). При отпускании клавиши ее ASCII код записывается в переменную “ascii_code”.

Следом начинается блок множественного выбора, построенный по известному алгоритму. Код нажатой клавиши сравнивается (командой **cjne...**) с заданным кодом, в нашем случае с '*'. Если коды совпадают, выполняется “задача 1” после чего команда **jmp _88888888** возвращает нас к новому циклу ожидания нажатия на клавишу. Если коды не совпадают, команда **cjne.....** переходит к метке “reshetka” и далее процесс сравнения повторяется. Если не нажата ни одна из четырех

функциональных клавиш, то завершающая команда **jmp _88888888** вернет программу в начало цикла ожидания.



4 Если все понятно, переходим к наполнению задач соответствующими командами.

```

  call InitALL;== начальная настройка
// call menu;== текст меню действий - на экран дисплея
_88888888:;== начало бесконечного цикла основной программы
  call Wait_4_KeyPressed;== ожидание нажатия на клавишу
zvezda:;== начало блока выбора действий vvvvvvvvvv
  cjne ascii_code,#'*',reshetka;== если '*' не нажата, переход к '#'
  call LCDclear;== если нажата - выводим на LCD меню действий
  call menu;== действие 1 (вывод меню)
  jmp _88888888
reshetka:
  cjne ascii_code,#' #',bukva_D;== если '#' не нажата, переход к 'D'
  call LCDclear;== иначе выводим .....
  LCDstrn adc_pusk;== действие 2 (пуск АЦП)
  jmp _88888888
bukva_D:
  cjne ascii_code,#'D',cifra_0;== если 'D' не нажата, переход к '0'
  call LCDclear;== иначе выводим текст - "D - выберите код"
  LCDstrn dac_code;== действие 3 (выбор кода ЦАП)
  jmp _88888888
cifra_0:
  cjne ascii_code,#'0',i_t_d;== если '0' не нажата, переход к i_t_d
  call LCDclear;== иначе очищаем дисплей (действие 4)
  jmp _88888888
i_t_d:;.....;== действие i (и так далее)
;== конец блока выбора действий ^^^^^^^^^^^^^^^^^
  jmp _88888888;== конец бесконечного цикла
;====endmain====endmain====endmain====endmain====

```

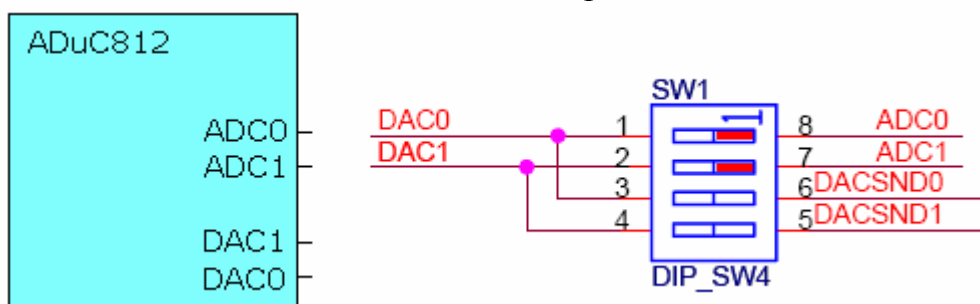
На этом этапе в каждый пункт меню добавлена очистка экрана LCDclear и в первые три – введены наименования действий.

Добавьте новые операторы, оттранслируйте и загрузите программу. Поочередно нажимая на функциональные клавиши, убедитесь, что все 4 пункта меню выполняются, а на остальные клавиши программа не реагирует. Если есть желание “стереть” экран при нажатии незадействованных клавиш, можно записать вызов подпрограммы очистки перед последней командой **jmp _88888888**.

V-1). Встроенные ЦАП и АЦП

5 В микроконтроллере ADuC812 используется 8-ми канальный 12-ти разрядный АЦП последовательных приближений(входы ADC0..ADC7) и два 12-ти битных ЦАП.

ВНИМАНИЕ: С помощью переключателя в левом верхнем углу лицевой панели стенда НУЖНО соединить выводы ADC0, DAC0 и ADC1, DAC1, что даст возможность исследовать работу этих устройств без подключения внешних источников напряжения.



V-1-1). Регистры АЦП

АЦП имеет три управляющих регистра - ADCCON1.. ADCCON3 и два регистра данных ADCDATAL и ADCDATAH. Младшие 8 бит АЦП записывает в ADCDATAL, а 4-ре старших в младшую тетраду ADCDATAH

Управляющий регистр АЦП - ADCCON1

MD1	MD0	CK1	CK0	AQ1	AQ0	T2C	EXC
0	0	Powered Down – АЦП обесточено (за исключ. Vref (Uo))					
0	1	Normal Mode					
1	0	Powered Down – АЦП обесточено					
1	1	Standby - АЦП бездействует					
		0	0	Fmclk / 1		Делитель	

	0	1	Fmclk / 2		основной тактовой частоты (Master Clock)	
	1	0	Fmclk / 4			
	1	1	Fmclk / 8			
Число тактовых циклов clk для выборки входного напряжения	1		0	0		
	2		0	1		
	4		1	0		
	8		1	1		
Запуск АЦП по переполнению Таймера2 (Timer2 OverFlow)					1	
Запуск АЦП внешним инверсным сигналом на входе CONVST						1

В этом регистре в обязательном порядке необходимо записать '1' в бит **MD0** (что значит – нормальный режим работы). Значения остальных битов оставим “по умолчанию”, т.е. равными нулю. **ADCCON1 = 40h**. Подробное описание действия битов можно найти в справочнике.

Управляющий регистр АЦП - ADCCON2

ADCI	DMA	CCONV	SCONV	CS3	CS2	CS1	CS0
------	-----	-------	-------	-----	-----	-----	-----

Бит **SCONV** – Single CONVersion (однократное преобразование). Запись в SCONV единицы начинает преобразование. Бит автоматически сбрасывается в конце преобразования. **ADCI** – флаг окончания преобразования (автоматически устанавливается в '1'). Сбрасывается также автоматически при переходе к обработчику прерывания. Если прерывания не используются, сбрасывать его нужно принудительно. Если бит CS3=0, то биты CS2..CS0 – определяют номер входного канала, по умолчанию – нулевой (вход ADC0).

Управляющий регистр АЦП – ADCCON3

BUSY							
------	--	--	--	--	--	--	--

BUSY – бит занятости АЦП. Равен '1', если преобразование не закончилось. В конце автоматически обнуляется. Используется для обнаружения окончания преобразования. Остальные биты этого регистра не используются. **У этого регистра нет битовой адресации**, поэтому BUSY необходимо ДОПОЛНИТЕЛЬНО выделять маской.


V-1-2). Регистры ЦАП

Оба ЦАП'а имеет один управляющий регистр DACCON и по два регистра данных DACxL и DACxH (x=0,1). Младшие 8 бит данных нужно записывать в DACxL, а 4-ре старших в младшую тетраду DACxH.

MODE	RNG1	RNG0	CLR1	CLR0	SYNC	~PD1	~PD0
------	------	------	------	------	------	------	------

бит	Мнемоника	Описание
7	MODE	Бит устанавливает режим работы обоих ЦАП. Если = 1, то 8-ми битный (запись 8-ми битов в DACxL SFR). Если = 0, то 12-битный.
6	RNG1	Бит выбора диапазона ЦАП1. Если = 1, то диапазон ЦАП1 0 .. Vdd. Если = 0, то диапазон ЦАП1 0 .. Vref.
5	RNG0	Бит выбора диапазона ЦАП0. Если = 1, то диапазон ЦАП0 0 .. Vdd. Если = 0, то диапазон ЦАП0 0 .. Vref.
4	CLR1	Бит очистки ЦАП1. Если = 1, то выход ЦАП1 соответствует коду. Если = 0, то выход ЦАП1 = 0В.
3	CLR0	Бит очистки ЦАП0. Если = 1, то выход ЦАП0 соответствует коду. Если = 0, то выход ЦАП0 = 0В.
2	SYNC	Бит синхронизации ЦАП0/1. Если = 1, то выходы ЦАПов изменяются сразу, как только данные попадают в регистры DACxL SFRs. Пользователь может одновременно обновить выходы обоих ЦАПов путем предварительной записи данных в DACxL/H при SYNC = 0. Выходы обоих ЦАПов одновременно обновятся теперь при установке SYNC = 1.
1	PD1	Бит выключения ЦАП1. Если = 1, то ЦАП1 включен. Если = 0, то ЦАП1 выключен.
0	PD0	Бит выключения ЦАП0. Если = 1, то ЦАП0 включен. Если = 0, то ЦАП0 выключен.

V-2). Программирование встроенных ЦАП и АЦП.

5  Команды первоначальной настройки ЦАП и АЦП помещены в п/п "DAC_ADC_init". Для упрощения отображения кодов ЦАП и АЦП на линейке из 8-ми светодиодов запрограммируем ЦАП на 8-ми битный режим, а в АЦП используем только 8 старших бит.

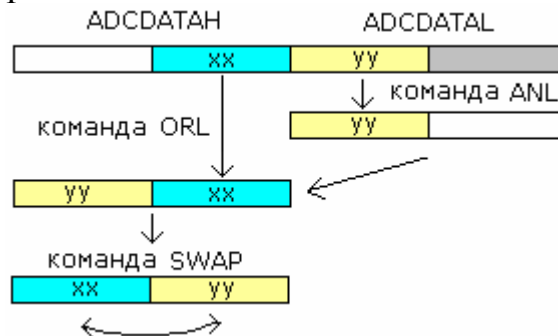
Первая команда настраивает ЦАП на 8-ми битный режим. Остальные биты в соответствии с вышеприведенной таблицей. Первый пуск по справочнику д.б. холостым и после него требуется задержка на 1сек (20 циклов по 0.05 сек). Предпоследняя команда инициализирует АЦП. Введите процедуру в программу.

```

LCDclear:;== очистка экрана ЖКД
    LCDctrl 1
    ret
;=====
DAC_ADC_init:;== начальная настройка ЦАП и АЦП
    mov DACCON,#0xFF;== настройка ЦАП (8/12бит выбран 8-битный режим)
    mov DACDH,#0;== загрузка старшего байта в ЦАП
    mov DACDL,#0;== пуск ЦАП при загрузке мл. байта (холостой запуск!)
    mov cnt,#20;== задержка для первого пуска ЦАП (справочник)
nx: Delay 100,229;== 0.05сек * 20 = 1сек
    djnz cnt,nx;== уменьшаем cnt=20 до 0
    mov ADCCON1,#40h;== настройка АЦП
    ret
;=====

```

Следующая подпрограмма считывает сформированный АЦП 12-ти битный код. Выделяет 8 старших бит и выводит их на светодиоды. Конечно, нагляднее выводить значения на ЖКД, но тогда потребуются дополнительное преобразование BIN кода в ASCII. На рисунке проиллюстрирован алгоритм объединения младшей тетрады старшего байта и старшей тетрады младшего байта.



После выполнения команды 'swar' в аккумуляторе находится выходной код АЦП, и его можно было бы сразу отправить на индикацию "Write ass, LED_address". Однако, из-за особенностей реализации SDK1.1 выходной код АЦП (для режима сквозного тестирования канала ЦАП-АЦП) необходимо калибровать. Следующие 4-ре команды выполняют простейшую калибровку. Этот фрагмент также добавьте в программу.

```

    mov ADCCON1,#40h;== настройка АЦП
    ret
;=====
readADC:;== чтение и отображение кода АЦП на линейке LED
    mov a,ADCDATAH;== записать в аккумулятор мл. байт из АЦП
    anl a,#0xF0;== выделить ст. тетраду мл. байта маской F0
    orl a,ADCDATAH;== объединить по ИЛИ обе тетрады в аккумуляторе
    swap a;== переставить тетрады местами (вернуть их на место)
    mov cnt,a;== результат во временную переменную cnt
    mov b,#11;== k-находится экспериментально (калибровка)
    div ab;== разделить (a) на (b)
    add a,cnt;== поправочный коэффициент (1+1/k)* cnt
    Write ass,LED_address;== высветить код АЦП на светодиодах
    ret
;=====

```

Входное напряжение для АЦП получается на выходе ЦАП. От произвольного начального кода ЦАП (в программе DACfst=250) при

каждом нажатии на клавишу 'D' вычитается некоторое значение (в программе 10). И начальное значение и декремент могут быть любыми. Запишите вспомогательную п/п.

```

;=====
smena_koda_ADC:== вспомогательная подпрограмма
    mov a,daccode,== для смены кода на входе ЦАП
    subb a,#10,== любое значение
    mov daccode,a
    Write daccode,LED_address,== отобразить новый код на LED
    mov DACOL,daccode,== код записывается в мл. байт ЦАП
    ret
;=====
menu1: DB "0-clear *-menu",0

```

Далее объявляются одна константа и одна переменная:

```

CNTR_LCD EQU 0F9h ;== адрес регистра управл. RG3 ЖКД (рис.3)
DACfst EQU 250,== некоторое начальное значение ЦАП
;=====
DSEG,== начало сегмента данных
daccode: DS 1,== код на входах ЦАП
DL1: DS 1,== счетчик циклов для макроса Delay

```

Модернизация подпрограммы InitALL.

```

InitALL:== п/п инициализации всего и вся
    mov DPP,#DPP_page_addr,== адрес страницы, на кот. находятся ВУ
    LCDinit,== настройка ЖКД
    mov daccode,#DACfst,== некое нач. значение ЦАП (255..0)
    call DAC_ADC_init,== настройка ЦАП и АЦП
    ret

```

И наконец добавим команды запуска АЦП и ЦАП

```

reshetka:
    cjne ascii_code,'#',bukva_D,== если '#' не нажата, переход к 'D'
    call LCDclear
    LCDstrn adc_pusk,== иначе выводим строку "# - PUSK ADC"
    setb SCONV,== пуск АЦП
w1: jnb ADSC, w1,== ждем окончание цикла преобр. (ADSC бит готовности)
    clr ADSC,== преобр. завершено - обнуляем признак готовности
    call readADC,== читаем и отображаем выходной код АЦП
    jmp _88888888
bukva_D:
    cjne ascii_code,'D',cifra_0,== если 'D' не нажата, переход к '0'
    call LCDclear
    LCDstrn dac_code,== иначе выводим текст - "D - выберите код ЦАП"
    call smena_koda_ADC
    jmp _88888888
cifra_0:
    cjne ascii_code,'0',i_t_d,== если '0' не нажата, переход к i_t_d
    call LCDclear,== иначе очищаем дисплей
    Write #0,LED_address,==гашение светодиодов
    mov daccode,#DACfst,== загрузить в ЦАП некий начальный код
    jmp _88888888

```

Окончательный текст программы.

```
#include <ADUC812.h>;== файл с определениями адресов регистров МК
#include "C:\EMUL\Work\Key_module.a51";== всё для работы с
клавиатурой
;=====
Tochka_Vhoda EQU 4000h;== адрес входа в основную программу
DATA_LCD EQU 0FEh ;== адрес регистра данных RG2 ЖКД (рис.3)
CNTR_LCD EQU 0F9h ;== адрес регистра управл. RG3 ЖКД (рис.3)
DACfst EQU 250;== некоторое начальное значение ЦАП
;=====
DSEG;== начало сегмента данных
daccode: DS 1;== код на входах ЦАП
DL1: DS 1;== счетчик циклов для макроса Delay
DL2: DS 1;== еще один
ctrl: DS 1;== код на управляющих входах ЖКД
cnt: DS 1;== вспомогательная переменная - счетчик
dno_steka: DS 1;== нужен ТОЛЬКО адрес - dno_steka
;=====
Delay MACRO Dmax1,Dmax2;== Dmax1=1(0.0005сек) при Dmax2=229
LOCAL L1,L2;== объявление локальных меток
mov DL1,#Dmax1;== константа для внешнего цикла
L1: mov DL2,#Dmax2;== константа для внутреннего цикла
L2: djnz DL2,L2;==(L2:djnz DL2,L2 - вложенный цикл)
djbz DL1,L1;==(L1:djbz DL1,L1 - внешний цикл)
ENDM;== конец макроопределения
;=====
Strobe MACRO Data1_Cntr0;== бит RS он же D/~C
mov ctrl, #Data1_Cntr0;== определяемся: данные или управление
orL ctrl, #0x01;== передний фронт строб-импульса (бит D0=1 "E")
Write ctrl, CNTR_LCD;== управл. код подаем на управл. входы ЖКД
anL ctrl, #0xFE;== задний фронт строб-импульса (бит D0=0 "E")
Write ctrl, CNTR_LCD;== управл. код подаем на управл. входы ЖКД
Delay 10,229;== задержка на 5 мсек (по справочнику)
ENDM
;=====
LCDctrl MACRO ch;== макрос вывода управл. байта в контроллер ЖКД
Write ch, DATA_LCD;== упр. байт код подаем на входы данных
ЖКД
Strobe 0x00;== строб+управл. код подаем на управл. входы ЖКД
ENDM;== 00h - бит D2(~C/D=0) признак записи упр. байта
;=====
LCDchar MACRO ch;== макрос вывода расшир. ASCII кода на ЖКД
Write ch, DATA_LCD;== ASCII код подаем на входы данных ЖКД
```

```

StrobE 0x04;== строб+управл. код подаем на управл. входы ЖКД
ENDM;== 04h - бит D2(~C/D=1) признак записи байта данных
;=====
LCDinit MACRO;== начальная настройка ЖКД (запись управляющих
байтов)
    IRP tt, <#0c7h,#0feh,#0f3h,#0f9h>;== встроенный макрос повторения
команд
        LCDctrl tt;== будут 4 раза повторены эти команды (tt=c7,fe,f3,f9)
    ENDM;== конец встроенного макроса
    ENDM;== конец макроопределения LCDinit
;=====
LCDstrn MACRO addr;== вывод строки символов на ЖК дисплей
    LOCAL povtor,next,vyhod;== addr - адрес строки
    mov cnt,#0;== начало строки ( cnt - смещение символа в строке)
povtor:
    mov a,cnt;== текущий номер выводимого символа
    mov DPTR,#addr;== в цикле т.к. LCDchar переопределяет DPTR
    movc a,@a + DPTR;== @a + DPTR - адрес символа в строке
    cjne a,#0,next;== если текущий символ=0 - то конец строки
    jmp vyhod;== и закончить вывод
next:
    LCDchar acc;== если нет - вывести на ЖКД очередной символ
    inc cnt;== перейти к следующему символу
    jmp povtor;== и повторить вывод
vyhod:
    ENDM
;=====
LCDxy MACRO x,y;== макрос позиционирования курсора ЖКД
    LOCAL up;== метка up - верхняя строка
    mov a,#y;== Y-координата строки 0-верхняя, 1-нижняя
    jz up;== если Y=0, значит строка верхняя и ее адрес=0
    mov a,#40h;== если Y=1, строка нижняя и ее адрес = 40h
up:    add a,#x;== прибавляем позицию в строке к ее адресу
    add a,#80h;== старш. бит упр. байта адреса позиции=1
    Write acc, DATA_LCD;== переводим курсор
    StrobE 0x00;== в указанную поз-ю
    ENDM
;=====
    CSEG at 0;== начало сегмента программы
    jmp Tochka_Vhoda;== переход к началу программы
    ORG Tochka_Vhoda;== с этого адреса располагается код программы
;=====main=====main=====main=====main=====main
main_prog: ;== точка входа в основную программу
    mov sp,#dno_steka;== записываем адрес начала стека в регистр sp

```

```

Delay 30,229;== задержка на 15мсек после включения ЖКД
call InitALL;== начальная настройка
call menu;== текст меню действий - на экран дисплея
_88888888;== начало бесконечного цикла основной программы
call Wait_4_KeyPressed;== ожидание нажатия на клавишу
zvezda;== начало блока выбора действий vvvvvvvvvv
cjne ascii_code,#'*',reshetka;== если '*' не нажата, переход к '#'
call LCDclear;==
call menu;== если нажата - выводим на LCD меню действий
jmp _88888888
reshetka:
cjne ascii_code,#'#',bukva_D;== если '#' не нажата, переход к 'D'
call LCDclear
LCDstrn adc_pusk;== иначе выводим строку "# - PUSK ADC"
setb SCONV;== пуск АЦП
w1: jnb ADCl, w1;== ждем окончание цикла преобр. (ADCl бит
готовности)
clr ADCl;== преобр. завершено - обнуляем признак готовности
call readADC;== читаем и отображаем выходной код АЦП
jmp _88888888
bukva_D:
cjne ascii_code,#'D',cifra_0;== если 'D' не нажата, переход к '0'
call LCDclear
LCDstrn dac_code;== иначе выводим текст - "D - выберите код ЦАП"
call smena_koda_ADC
jmp _88888888
cifra_0:
cjne ascii_code,#'0',i_t_d;== если '0' не нажата, переход к i_t_d
call LCDclear;== иначе очищаем дисплей
Write #0,LED_address;==гашение светодиодов
mov daccode,#DACfst;== загрузить в ЦАП некий начальный код
jmp _88888888
i_t_d;.....;== действие i (и так далее)
;== конец блока выбора действий ^^^^^^^^^^^^^^^^^^
jmp _88888888;== конец бесконечного цикла
;====endmain====endmain====endmain====endmain====
;== ДАЛЕЕ РАЗМЕЩАЮТСЯ ПОДПРОГРАММЫ ==
InitALL;== п/п инициализации всего и вся
mov DPP,#DPP_page_addr;== адрес страницы, на кот. находятся ВУ
LCDinit;== настройка ЖКД
mov daccode,#DACfst;== некое нач. значение ЦАП (255..0)
call DAC_ADC_init;== настройка ЦАП и АЦП
ret
;=====

```

```

menu:;== подпрограмма вывода меню действий
    LCDxy 0,0;== xy=00 - верхняя строка, слева
    LCDstrn menu1;== выводим на ЖКД первую строку меню
    LCDxy 0,1;== xy=01 - нижняя строка, слева
    LCDstrn menu2;== выводим на ЖКД вторую строку меню
    ret
;=====
LCDclear:;== очистка экрана ЖКД
    LCDctrl 1
    ret
;=====
DAC_ADC_init:;== начальная настройка ЦАП и АЦП
    mov DACCON,#0xFF;== инициализация ЦАП (8/12бит - выбран 8-
битный режим)
    mov DAC0H, #0;== загрузка старшего байта в ЦАП
    mov DAC0L, #0;== пуск ЦАП при загрузке мл. байта (холостой
запуск!)
    mov cnt,#20;== задержка для первого пуска ЦАП (справочник)
nx: Delay 100,229;== 0.05сек * 20 = 1сек
    djnz cnt,nx;== уменьшаем cnt=20 до 0
    mov ADCCON1,#40h;== настройка АЦП
    ret
;=====
readADC:;== чтение и отображение кода АЦП на линейке LED
    mov a,ADCDATL;== записать в аккумулятор мл. байт из АЦП
    anL a,#0xF0;== выделить ст. тетраду мл. байта маской F0
    orL a,ADCDATH;== объединить по ИЛИ обе тетрады в
аккумуляторе
    swap a;== переставить тетрады местами (вернуть их на место)
    mov cnt,a;== результат во временную переменную cnt
    mov b,#11;== k-находится экспериментально (калибровка)
    div ab;== разделить (a) на (b)
    add a,cnt;== поправочный коэффициент (1+1/k)* cnt
    Write acc,LED_address;== высветить код АЦП на светодиодах
    ret
;=====
smena_koda_ADC:;== вспомогательная подпрограмма
    mov a,daccode;== для смены кода на входе ЦАП
    subb a,#10;== любое значение
    mov daccode,a
    Write daccode,LED_address;== отобразить новый код на LED
    mov DAC0L,daccode;== код записывается в мл. байт ЦАП
    ret
;=====

```





```

menu1:    DB "0-clear *-menu",0
menu2:    DB "#-ADCpusk D-",0e1h,"A",0a8h,0;==0e1h,"A",0a8h - ЦАП
adc_pusk: DB "# - PUSK ADC",0
dac_code: DB "D - vyberite code",0
;=====
END

```

V-3) Окончательная проверка работы программы

Скомпилируйте  и загрузите  программу. Убедитесь, что программа работает по назначению. Клавишей “D” выберите любой код на входе ЦАП. Запишите его. Клавишей “#” несколько раз запустите АЦП, записывая выходной код. Вычислите среднюю разность между входным кодом ЦАП и выходным кодом АЦП. Измените значение “k” и снова проведите ряд измерений. Коды должны отображаться на светодиодах. На рисунке высвечивается код 8Ch.

ст. разряд сверху  мл. разряд внизу

Результат покажите преподавателю

Для удобства чтения выходного кода самостоятельно напишите подпрограмму преобразования одного байта кода АЦП, в два байта ASCII кода - “byte2ASCII” по приведенному алгоритму. В строку с именем “HEX” запишите ASCII коды 16-ных (hex) цифр.

```

HEX: DB "0123456789abcdef"

```

END

В следующей таблице - 11 строчек с описаниями команд. Каждой строчке соответствует одна команда подпрограммы. Список команд приведен в приложении 1.

Byte2ASCII:	
1	В регистр DPTR записать символический адрес таблицы #HEX
2	В аккумулятор переслать переменную “cnt” – код АЦП
3	Выделить маской #0F младшую тетраду кода (в ней младшая цифра)
4	Переслать в аккумулятор байт с адресом равным (a)+(DPTR). Теперь в аккумуляторе ASCII код младшей цифры кода
5	Поместить его из аккумулятора в регистр R2.
6	Снова в аккумулятор переслать переменную “cnt” – код АЦП
7	Поменять местами тетрады
8	Выделить маской #0F младшую тетраду кода (в ней

	теперь старшая цифра)
9	Переслать в аккумулятор байт с адресом равным (a)+(DPTR). Теперь в аккумуляторе ASCII код старшей цифры кода
10	Поместить его из аккумулятора в регистр R3.
11	Команда возврата из подпрограммы

#####

ВНИМАНИЕ: Полученную из таблицы подпрограмму введите в текст основной программы.

#####

Также добавьте подпрограмму вывода 2-х цифр на ЖКД.

```

=====
_2char2LCD:
    mov cnt, a;== readADC возвращает код в аккумуляторе, запишем его в cnt
    call byte2ASCII;== вызов п/п преобразования
    LCDxy 7,1;== курсор на нижнюю строку в 7-ю позицию
    LCDchar r3;==вывод ст. цифры кода
    LCDchar r2;==вывод мл. цифры кода
    ret
=====
HEX: DB "0123456789abcdef"
=====
END

```

Вместо регистров R2,R3 можно использовать две дополнительные переменные, например tH и tL для старшей и младшей тетрады. Ну и напоследок, добавьте вызовы этой подпрограммы в блоках для клавиш “reshetka” и “cifra_D”:

```

call readADC;== читаем
call _2char2LCD;== и отображаем выходной код АЦП
jmp _88888888
bukva_D:
    cjne ascii_code,#'D',cifra_0;== если 'D' не нажата, переход к '0'
    call LCDclear
    LCDstrn dac_code;== иначе выводим текст - "D - выберите код ЦАП"
    call smena_koda_ADC;== меняем входной код ЦАП
    mov a,DACOL;== записываем его в аккумулятор
    call _2char2LCD;== и отображаем на дисплее
    jmp _88888888
cifra_0:

```

После загрузки программы в нижней строке будет отображаться либо код АЦП (клавиша ‘#’), либо код ЦАП (клавиша ‘D’).

D - vyberite cod e6	# - PUSK ADC e7
------------------------	--------------------

Результат покажите преподавателю

ПРИЛОЖЕНИЕ №1. Некоторые команды и директивы ассемблера MCS-51

DST, SRC – операнд приемник, источник

operand – операнд

bit – однобитовый операнд

xrl DST, SRC - “поразрядное исключающее ИЛИ”, байт SRC может трактоваться как маска

anl DST, SRC - “поразрядное И”, байт SRC может трактоваться как маска

orl DST, SRC - “поразрядное ИЛИ”, байт SRC может трактоваться как маска

clr bit - обнулить бит

setb bit - установить бит

cjne mem, #operand, address - сравнить содержимое ячейки памяти (mem) с операндом и, если они не равны перейти по адресу.

movx @dptr, a - переслать содержимое аккумулятора в ВУ/ЗУ с адресом, который находится в двухбайтовом регистре DPTR.

movx a, @dptr - в обратном направлении

mov DST, SRC – переслать (скопировать) операнд источник в приемник

movc a, @a + DPTR – байт из ячейки внешней памяти программы с адресом равным сумме адреса в регистре DPTR и содержимого аккумулятора пересылается в аккумулятор

jmp address – безусловный переход по адресу (в зависимости от модели памяти трансформируется компилятором в LJMP, AJMP..... или SJMP).

jnb bit, address – условный переход по указанному адресу, если бит равен нулю (jump if not bit)

jz address – условный переход, если содержимое аккумулятора равно нулю

call address – вызов подпрограммы по адресу address и в стек помещается адрес возврата.

ret – возврат из подпрограммы (из стека извлекается адрес возврата)

reti – возврат из подпрограммы обработчика прерывания (из стека извлекается адрес возврата)

push operand – поместить операнд в стек

pop operand – извлечь операнд из стека

inc operand – увеличить операнд на единицу

jnc address – условный переход по адресу, если во флаге переноса “0” (т.е. нет переноса)

rr a (rl a) – циклический (круговой) сдвиг содержимого аккумулятора вправо (влево)

rrc a (rlc a) – циклический сдвиг аккумулятора вправо (влево) через флаг переноса

swap a – поменять местами тетрады аккумулятора

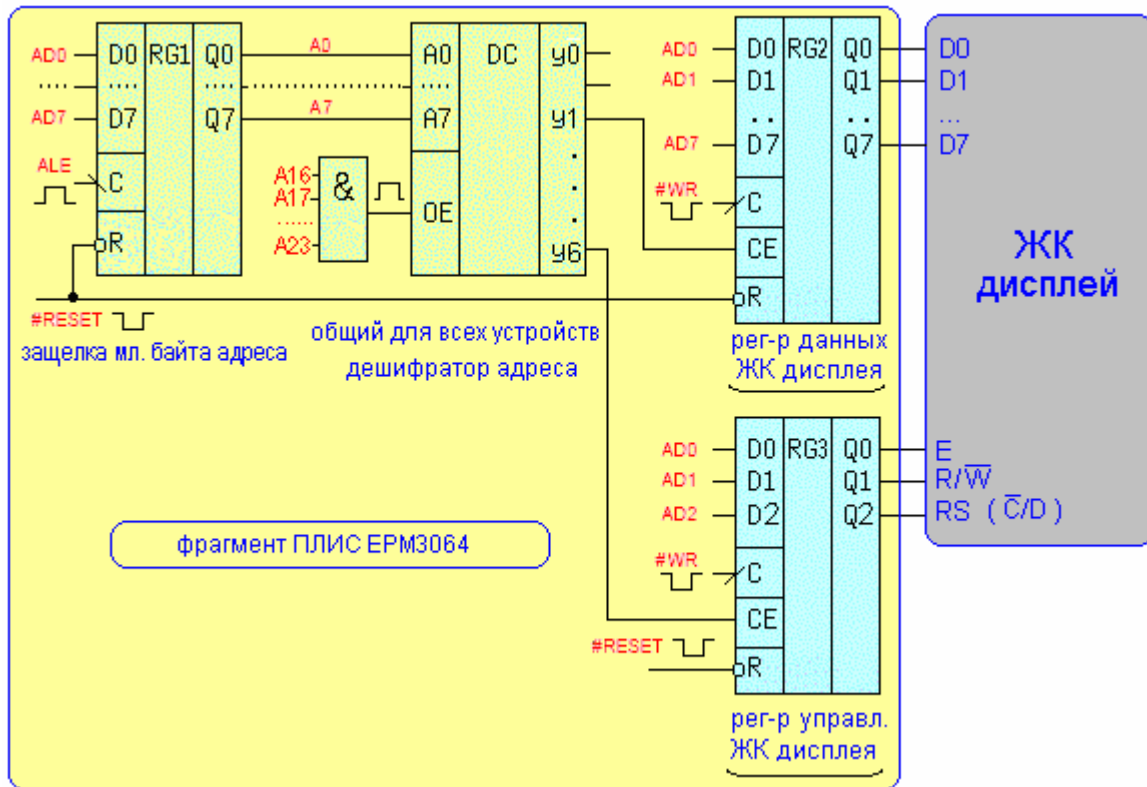
div a,b – байт в аккумуляторе делится на байт в регистре ‘b’ (после деления - частное в аккумуляторе, остаток в ‘b’)
subb a, operand – вычитание байтов

ВАРИАНТЫ технического задания

Yx=Y17 Yz=Y54 Tochka_Vhoda 3ef0 Буква ‘Ю’ ADCpush --> АЦПпуск Курсор не виден	Yx=Y49 Yz=Y86 Tochka_Vhoda 5bb0 Буква ‘Й’ clear --> очистка Курсор виден и мигает	Yx=Y81 Yz=Y46 Tochka_Vhoda 5bb0 Буква ‘Б’ clear --> очистка Курсор виден и не мигает
Yx=Y41 Yz=Y70 Tochka_Vhoda 4e40 Буква ‘Ч’ menu --> меню Курсор виден и мигает	Yx=Y97 Yz=Y38 Tochka_Vhoda 44d0 Буква ‘Я’ menu --> МЕНЮ Курсор виден и не мигает	Yx=Y105 Yz=Y126 Tochka_Vhoda 3d80 Буква ‘Ж’ clear --> ОЧИСТКА Курсор не виден
Yx=Y33 Yz=Y78 Tochka_Vhoda 5a80 Буква ‘Э’ clear --> стереть Курсор виден и не мигает	Yx=Y65 Yz=Y22 Tochka_Vhoda 29f0 Буква ‘Ю’ ADCpush --> АЦППУСК Курсор не виден	Yx=Y73 Yz=Y62 Tochka_Vhoda 68c0 Буква ‘П’ ADC --> цап Курсор виден и мигает

ВОПРОСЫ ДЛЯ ЗАЩИТЫ И ЭКЗАМЕНА

- 1) Понимать назначение отдельных составляющих МК системы: ШАД, ША, ШУ, защелки, RAM, стробы (ALE, #WR, #RD и #PSEN).
- 2) Структурная схема МК ADuC812.
- 3) Работа схемы по приведенному рисунку, расчет 24-х битных адресов RG2 и RG3. Выполнение команды “MOVX @DPTR,A” для приведенного рисунка (взаимодействие с ША, ШД и ШУ). Подробно этот вопрос рассмотрен в лекции “8.5.1 МК система с тремя шинами” и в разделе “Расчет адреса регистра RG2”.



4) Уметь комментировать и понимать работу этих подпрограмм (применительно к рисунку п.3)

```

;=====
Strobe MACRO Data1_Cntr0,== бит RS он же D/~C
    mov ctrl, #Data1_Cntr0,== определяем: данные или управление
    orL ctrl, #0x01,== передний фронт строб-импульса (бит D0=1 "E")
    Write ctrl, CNTR_LCD,== управл. код подаем на управл. входы ЖКД
    andL ctrl, #0xFE,== задний фронт строб-импульса (бит D0=0 "E")
    Write ctrl, CNTR_LCD,== управл. код подаем на управл. входы ЖКД
    Delay 10,229,== задержка на 5 мсек (по справочнику)
ENDM

;=====
LCDctrl MACRO ch,== макрос вывода управл. байта в контроллер ЖКД
    Write ch, DATA_LCD,== упр. байт код подаем на входы данных ЖКД
    Strobe 0x00,== строб+управл. код подаем на управл. входы ЖКД
ENDM,== 00h - бит D2(~C/D=0) признак записи упр. байта

;=====
LCDchar MACRO ch,== макрос вывода расшир. ASCII кода на ЖКД
    Write ch, DATA_LCD,== ASCIIp код подаем на входы данных ЖКД
    Strobe 0x04,== строб+управл. код подаем на управл. входы ЖКД
ENDM,== 04h - бит D2(~C/D=1) признак записи байта данных

;=====
LCDinit MACRO,== начальная настройка ЖКД (запись управляющих байтов)
    IRP tt, <#0c7h,#0feh,#0f3h,#0f9h>,== макрос повторения команд
    LCDctrl tt,== будут 4 раза повторены эти команды (tt=c7,fe,f3,f9)
    ENDM,== конец встроенного макроса
ENDM,== конец макроопределения LCDinit
;=====

```

5) Уметь комментировать и понимать работу основной программы

```

main_prog: ;== точка входа в основную программу
mov sp,#dno_steka;== записываем адрес начала стека в регистр sp
Delay 30,229;== задержка на 15мсек после включения ЖКД
call InitALL;== начальная настройка
call menu;== текст меню действий - на экран дисплея
_88888888:;== начало бесконечного цикла основной программы
call Wait_4_KeyPressed;== ожидание нажатия на клавишу
zvezda:;== начало блока выбора действий vvvvvvvvvv
cjne ascii_code,'#*',reshetka;== если '*' не нажата, переход к '#'
call LCDclear;==
call menu;== если нажата - выводим на LCD меню действий
jmp _88888888
reshetka:
cjne ascii_code,'#',bukva_D;== если '#' не нажата, переход к 'D'
call LCDclear
LCDstrn adc_pusk;== иначе выводим строку "# - ПУСК ADC"
setb SCONV;== пуск АЦП
w1: jnb ADCI, w1;== ждем окончание цикла преобр. (ADCI бит готовности)
clr ADCI;== преобр. завершено - обнуляем признак готовности
call readADC;== читаем и отображаем выходной код АЦП
jmp _88888888
bukva_D:
cjne ascii_code,'#D',cifra_0;== если 'D' не нажата, переход к '0'
call LCDclear
LCDstrn dac_code;== иначе выводим текст - "D - выберите код ЦАП"
call smena_koda_ADC
jmp _88888888
cifra_0:
cjne ascii_code,'#0',i_t_d;== если '0' не нажата, переход к i_t_d
call LCDclear;== иначе очищаем дисплей
Write #0,LED_address;==гашение светодиодов
mov daccode,#DACfst;== загрузить в ЦАП некий начальный код
jmp _88888888
i_t_d:;.....;== действие i (и так далее)

```

6). Назначение битов управляющих байтов АЦП: ADCI, SCONV, BUSY, MD1, MD0, CS2...CS0.

7) Показать, где и как в программе используются данные из этой таблицы (приведен ее небольшой фрагмент)

Таблица 3										Команда/функции
RS~C/D	R/~W	D7	D6	D5	D4	D3	D2	D1	D0	
0	0	0	0	0	0	0	0	0	1	
0	0	0	0	0	0	0	0	1	x	
0	0	0	0	0	0	0	1	ID	S	

8) Почему команда LCDchar #'S' выведет на экран ЖКД букву 'S', а команда LCDchar #'Ы' – вместо 'Ы', что-то совсем другое (и кстати, что)?

9) Как работает приведенная подпрограмма?

```

readADC:;== чтение и отображение кода АЦП на линейке LED
    mov a,ADCDATAL;== записать в аккумулятор мл. байт из АЦП
    anl a,#0xF0;== выделить ст. тетраду мл. байта маской F0
    orl a,ADCDATAH;== объединить по ИЛИ обе тетрады в аккумуляторе
    swap a;== переставить тетрады местами (вернуть их на место)
    .....
    Write acc,LED_address;== высветить код АЦП на светодиодах
    ret

```

10) Что будет в регистрах r2 и r3 после выполнения следующего фрагмента

```

.....
mov DPTR, #HEX
mov a, cnt
anl a,#0fh
movc a,@a+DPTR
mov r2,a
mov a, cnt
swap a
anl a,#0fh
movc a,@a+DPTR
mov r3,a
ret
HEX: DB "0123456789abcdef"

```

ВНИМАНИЕ: Во время экзамена фрагменты программы будут предъявляться **БЕЗ КОММЕНТАРИЕВ.**

ЛИТЕРАТУРА

1. Китаев Ю.В. Основы цифровой техники: [учебное пособие], М-во образования и науки РФ;СПбГУ ИТМО, [Каф. электроники]. СПб.: СПбГУ ИТМО, 2007 .— 87 с.
2. Китаев Ю.В. Основы программирования микроконтроллеров ATMEGA 128 и 68HC908: [учебное пособие], М-во образования и науки РФ; СПбГУ ИТМО, [Каф. электроники] .— СПб.: СПбГУ ИТМО, 2007 .— 107 с.
3. Китаев Ю.В. Лабораторные и практические работы "Электроника и МП техника", [учебное пособие], М-во образования и науки РФ; СПбГУ ИТМО, [Каф. электроники]. СПб.: СПбГУ ИТМО, 2008 .— 92 с.: ил .— (Приоритетные национальные проекты. Образование).
4. Китаев Ю.В. Лабораторные работы "Программирование МК на ассемблере ASM-51", [учебное пособие], М-во образования и науки РФ; СПбГУ ИТМО, [Каф. электроники].— СПб.: СПбГУ ИТМО, 2010 .— 92 с.
5. http://faculty.ifmo.ru/electron/cons/raspisanie_current.htm



В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена Программа развития государственного образовательного учреждения высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» на 2009–2018 годы.

КАФЕДРА ЭЛЕКТРОНИКИ

Заведующий кафедрой: д.т.н., проф. Г.Н. Лукьянов.

Кафедра Электроники (первоначальное название “Радиотехники”) была основана в 1945 году. Первым руководителем кафедры был С.И. Зилитинкевич известный в стране и за рубежом ученый в области физической электроники и радиотехники, активный работник высшей школы, заслуженный деятель науки и техники РСФСР, доктор технических наук, профессор ЛИТМО с 1938 г., инициатор создания в ЛИТМО инженерно-физического и радиотехнического факультетов (1946г.). С.И. Зилитинкевич заведовал кафедрой с 1945 до 1978 года. Под его научным руководством аспирантами и соискателями выполнено более 50 кандидатских диссертаций, многие его ученики стали докторами наук.

В дальнейшем, с 1978 г. по 1985 г. кафедру возглавил к.т.н., доцент Е.К. Алахов, один из учеников С.И. Зилитинкевича.

С 1985 г. по 2006 г. руководителем кафедры стал д.т.н., профессор В.В. Тогатов, известный специалист в области силовой электроники и приборов для измерения параметров полупроводниковых структур.

Начиная с 2006 г. кафедрой заведует д.т.н., профессор Г.Н. Лукьянов, под руководством и при участии которого кардинально обновилось лабораторное оборудование в рамках инновационной программы развития.

Основные направления кафедры связаны с разработкой приборов для лазерной и медицинской техники, приборов для измерения параметров полупроводниковых структур, а также встраиваемых цифровых и микропроцессорных устройств.

Под руководством В.В. Тогатова было разработано и изготовлено большое число приборов различного назначения:

- Измеритель параметров ультрабыстрых диодов;
- Универсальное устройство для исследования переходных процессов в силовых полупроводниковых структурах;
- Измеритель времени жизни заряда в слаболегированных областях диодных, тиристорных и транзисторных структур;
- Универсальный разрядный модуль для накачки твердотельных лазеров;
- и ряд других.

На кафедре написаны и размещены на сайте ЦДО следующие материалы для дистанционного обучения (автор Ю.В. Китаев):

- Конспект лекций по дисциплине “Электроника и микропроцессорная техника”;
- свыше 600 вопросов к обучающим и аттестующим тестам;
- 18 дистанционных лабораторных и практических работ

На кафедре имеются следующие компьютеризированные учебные лаборатории:

- АРМС – полупроводниковые приборы;
- Устройства на полупроводниковых приборах;
- Цифровая техника;
- Микропроцессорная техника
- Моделирование электронных устройств.

Юрий Васильевич Китаев

Программирование МК на
ассемблере ASM-51и AVR Pascal

Учебное пособие

В авторской редакции

Дизайн Китаев Ю.В.

Верстка Китаев Ю.В.

Редакционно-издательский отдел Санкт-Петербургского государственного
университета информационных технологий, механики и оптики

Зав. РИО Н.Ф. Гусарова

Лицензия ИД 3 00408 от 05.11.99

Подписано к печати _____

Заказ № _____

Тираж 100 экз.

Отпечатано на ризографе

Редакционно-издательский отдел
Санкт-Петербургского государственного
университета информационных технологий,
механики и оптики
197101, Санкт-Петербург, Кронверкский пр., 49

