

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Ю.В. Китаев

ПРОГРАММИРОВАНИЕ МК НА

АССЕМБЛЕРЕ ASM-51

Учебное пособие



Санкт-Петербург

2010

Китаев Ю.В. “Программирование МК на ассемблере ASM-51”. Учебное пособие: СПб: СПбГУ ИТМО, 2010. ____ с.

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

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 06 октября 2009г., 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, 2010

© Ю.В. Китаев, 2010

ОГЛАВЛЕНИЕ

ЛАБОРАТОРНАЯ РАБОТА № 30 “РАЗРАБОТКА ОБРАБОТЧИКОВ ВНЕШНИХ ПЕРЕРЫВАНИЙ И ТАЙМЕРА”	5
ВВЕДЕНИЕ.....	5
ТЕХНИЧЕСКОЕ ЗАДАНИЕ.....	9
РАСЧЕТ АДРЕСА РЕГИСТРА УПРАВЛЕНИЯ СВЕТОДИОДАМИ	9
Расчет адреса регистра RG2.	10
РАЗРАБОТКА ПРОГРАММЫ	11
I) Создание шаблона программы на ассемблере.....	11
II). Создание начальной программы на ассемблере ASM51.....	14
III) Пояснения к программе.....	15
IV) Пробный запуск программы.....	16
ИСПОЛЬЗОВАНИЕ ТАЙМЕРА ДЛЯ ПЕРИОДИЧЕСКОГО УПРАВЛЕНИЯ ПРОЦЕССОМ	20
V). Адреса обработчиков прерываний.....	20
VI) Создание обработчика прерывания от переполнения таймера2 и расчет его параметров.....	22
VII) Создание обработчиков прерываний от внешних событий.....	29
ПРОВЕРКА РАБОТЫ ПРОГРАММЫ В СИМУЛЯТОРЕ-ОТЛАДЧИКЕ	34
ПРИЛОЖЕНИЕ 1. РЕГИСТРЫ УПРАВЛЕНИЯ И СОСТОЯНИЯ	37
ПРИЛОЖЕНИЕ 2. БИТЫ УПРАВЛЕНИЯ И СОСТОЯНИЯ.....	38
ПРИЛОЖЕНИЕ 3. ТАБЛИЦА ВЕКТОРОВ ПЕРЕРЫВАНИЙ	39
ПРИЛОЖЕНИЕ 4. РЕГИСТРЫ УПРАВЛЕНИЯ И СОСТОЯНИЯ ТАЙМЕРА2	39
ПРИЛОЖЕНИЕ 5. НЕКОТОРЫЕ КОМАНДЫ И ДИРЕКТИВЫ АССЕМБЛЕРА MCS-51	40
ПРИЛОЖЕНИЕ 6. ШИНА VS ПОРТ	41
ВАРИАНТЫ ТЕХНИЧЕСКОГО ЗАДАНИЯ	42
ВОПРОСЫ ДЛЯ ЗАЩИТЫ И ЭКЗАМЕНА.....	43
ЛАБОРАТОРНАЯ РАБОТА № 31 “ПРОГРАММИРОВАНИЕ КЛАВИАТУРЫ”	47
ТЕХНИЧЕСКОЕ ЗАДАНИЕ.....	47
РАСЧЕТ АДРЕСА РЕГИСТРА УПРАВЛЕНИЯ КЛАВИАТУРОЙ.....	47
Расчет адреса регистра RG2.	48
РАЗРАБОТКА ПРОГРАММЫ	49
I) Создание шаблона программы на ассемблере.....	49
II). Разработка программы фиксирующей момент нажатия на одну клавишу.....	52
III) Пояснения к программе.....	57
IV) Пробный запуск программы.....	58

V). Разработка программы фиксирующей момент нажатия на любую клавишу.	60
V-1) Модуль сканирования	62
V-2) Подпрограмма формирования скан-кода клавиши.....	64
V-3) Модуль сдвига “бегущего нуля” и корректировка основного модуля “main_prog”	64
V-4) Подпрограмма преобразования скан-кода клавиши в ее порядковый номер.....	68
V-5) Подпрограмма преобразования порядкового номера клавиши в ASCII код.	74
VI) Модификация программы в соответствии с заданием.....	75
VII) Использование прерываний от клавиатуры.....	76
ПРИЛОЖЕНИЕ №1. Некоторые команды и директивы ассемблера MCS-51	76
ПРИЛОЖЕНИЕ №2. Принципиальная схема подключения клавиатуры..	78
Мультиплексор приведенный на рисунке состоит из 8-ми параллельно включенных мультиплексоров “4 -> 1”, по одному мультиплексору на один выходной двоичный разряд.	79
ВАРИАНТЫ технического задания.	80
ВОПРОСЫ ДЛЯ ЗАЩИТЫ И ЭКЗАМЕНА	81
ВАРИАНТЫ технического задания.	85
ВАРИАНТЫ технического задания.	85
ВОПРОСЫ ДЛЯ ЗАЩИТЫ И ЭКЗАМЕНА	86
ЛИТЕРАТУРА.....	89

ЛАБОРАТОРНАЯ РАБОТА № 30 “РАЗРАБОТКА ОБРАБОТЧИКОВ ВНЕШНИХ ПРЕРЫВАНИЙ И ТАЙМЕРА”

ВВЕДЕНИЕ

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.

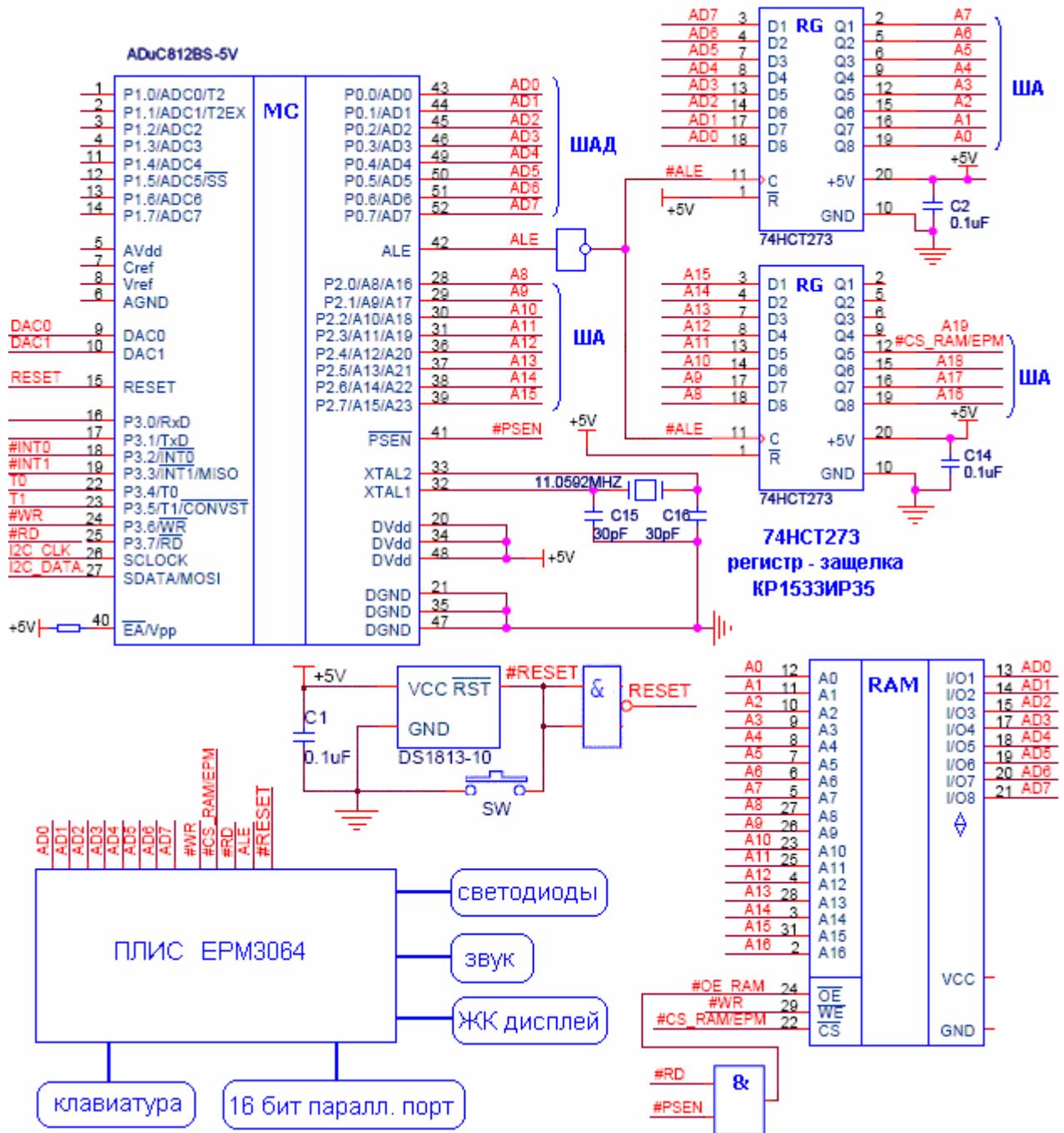
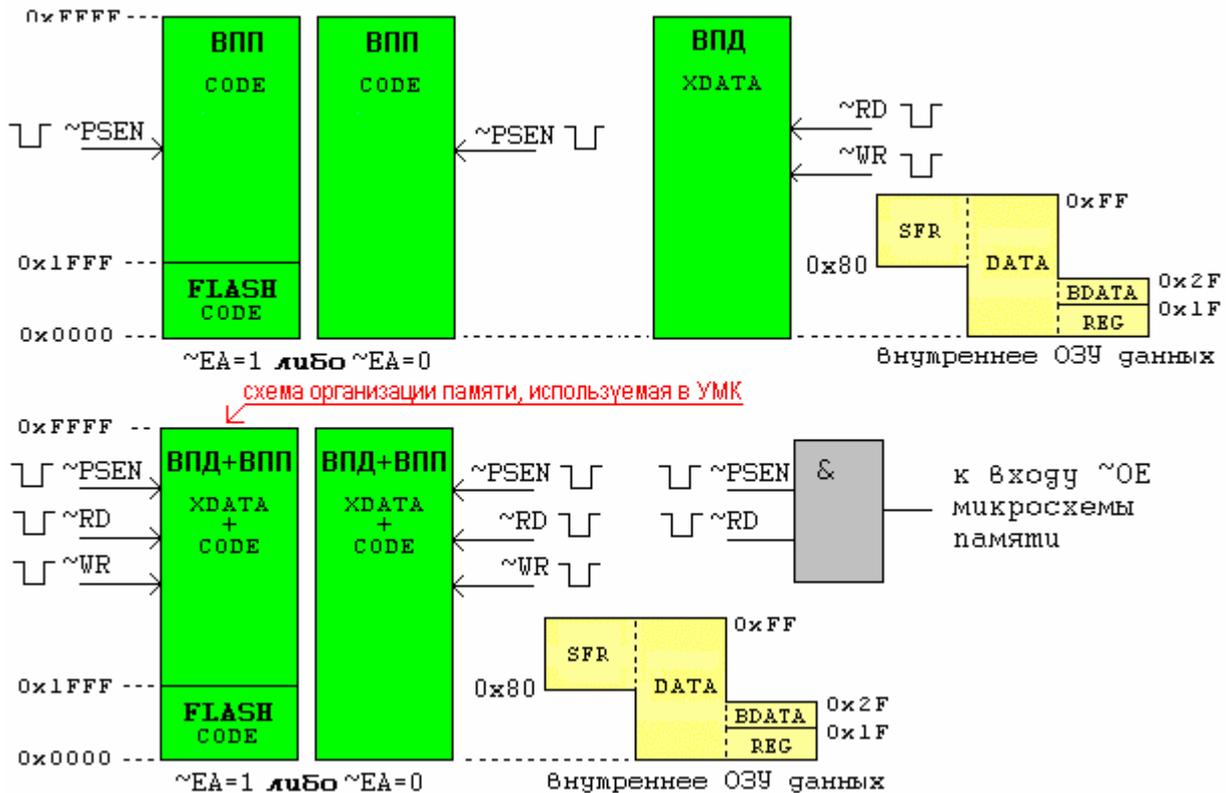


Рис.1 Принципиальная схема УМК

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). 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 строба. Однако, у микросхем памяти для этих целей только два входа: \sim WE (Write Enable) для строба \sim WR и \sim OE (Output Enable) для

строба $\sim RD$. Стандартным решением является объединение двух стробов 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).



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 $\sim EA$. Если на входе $\sim EA=1$, то МК может выбирать команды, как из внутренней памяти (флэш), так и из внешней памяти

программ (ОЗУ или ПЗУ). Если $\sim EA = 0$, внутренняя флэш-память МК недоступна.

Отладку и пробные пуски программ удобно вести с использованием совмещения ВПД + ВПП при $\sim EA = 1$. 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.30.2-1 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.30.1).



Рис.30.2-1

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Необходимо написать программу, которая управляет процессами, с заданной периодичностью. Основной процесс моделируется включением и выключением линейки светодиодов по заданному алгоритму. Управление производится циклически от внутреннего таймера МК и асинхронно с помощью двух внешних сигналов. Для решения задачи необходимо задействовать три источника прерывания.

РАСЧЕТ АДРЕСА РЕГИСТРА УПРАВЛЕНИЯ СВЕТОДИОДАМИ

В простых схемах, когда достаточно свободных выходов портов, светодиоды можно подключать через ограничивающие ток резисторы, прямо к этим выходам. 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.30.3 приведен фрагмент схемы, запрограммированной в ПЛИС EPM3064 и управляющей светодиодами.

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

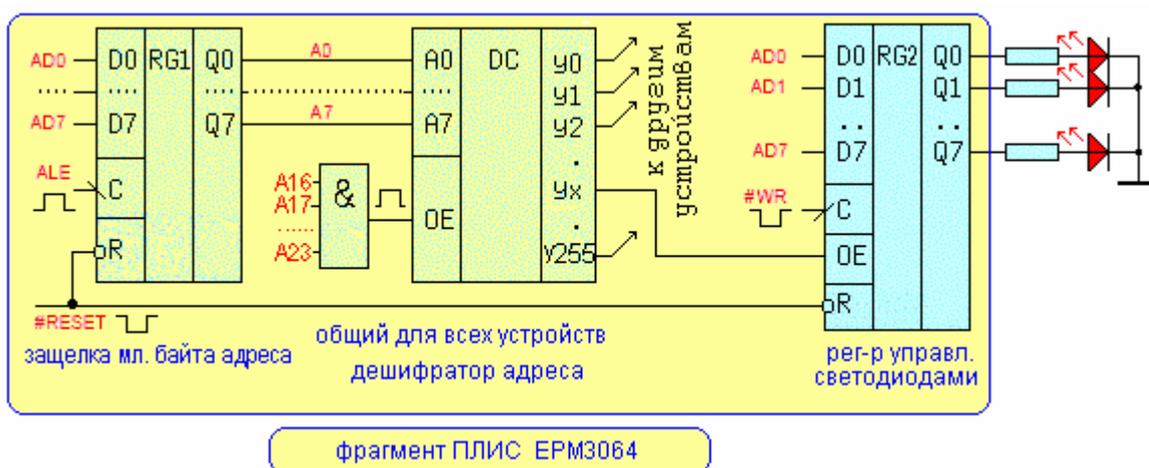
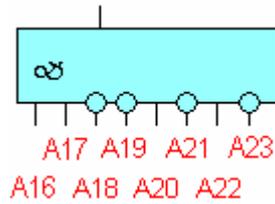


Рис.30.3

1 **Вариант задания:** 1) Задействован 62-й выход дешифратора, т.е. $Y_x = Y_{62}$, 2) Четыре входа элемента "И" A22, A20, A17 и A16 – прямые, остальные – инверсные (как показано на рисунке).



Расчет адреса регистра RG2.

Во время выполнения команды ассемблера “MOVX @DPTR, A” младший байт адреса ШАД (AD7, AD6,...,AD0) записывается в регистре RG1 стробом адреса ALE (A7, A6,...,A0). Далее, дешифратор адреса формирует один из 256-х управляющих сигналов (y0..y255), каждый из которых поступает на вход разрешения того или иного устройства (OE, CS и т.д.). На рис.30.3 сигнал Yx разрешает выходы по входу OE (Output Enable) регистра RG2. Далее за стробом ALE следует строб записи данных #WR, который защелкивает в RG2 байт данных (D7..D0), поступающий по ШАД. Естественно, что во время этих операций сигнал #RESET д.б. = 1, т.е. иметь пассивный уровень. Теперь в соответствии с кодом (Q7..Q0), высокий уровень Qi = 1 включит светодиод, а низкий погасит его.

Рассчитаем адрес для обращения к регистру управления светодиодами 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’).



Принимая во внимание, что на выходе ЛЭ “И” единица будет при единичных значениях на прямых входах и нулевых на инверсных входах получим, что A23,A22,A21,A20,A19,A18,A17,A16 = 01010011(BIN) = 53(HEX). Выход Y62 дешифратора будет активирован, когда на его адресных входах будет соответствующий двоичный код (62₁₀=00111110₂=3E₁₆). Тогда для нашей конкретной схемы (см. рис.30.3) и варианта задания запишем адрес регистра управления светодиодами:



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.

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

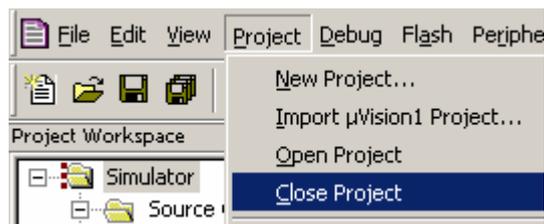
#####

ВНИМАНИЕ: Вам необходимо рассчитать свои значения (DPP) и (DPTR) в соответствии с полученным заданием.

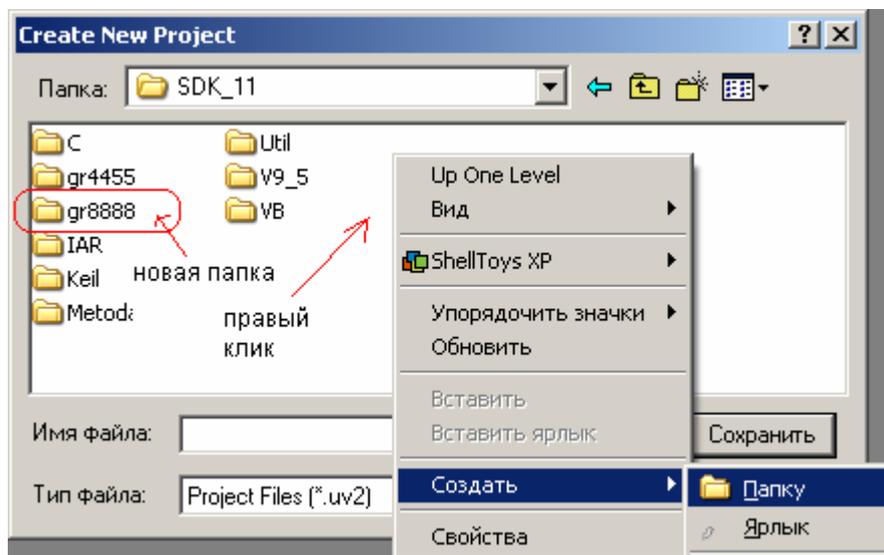
#####

РАЗРАБОТКА ПРОГРАММЫ

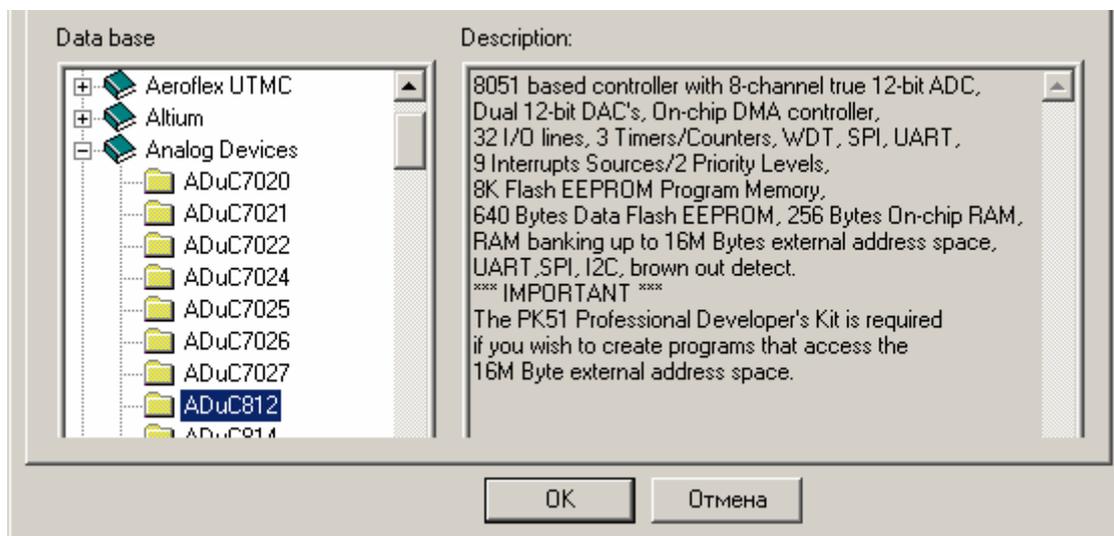
1) **Создание шаблона программы на ассемблере.** Запустите интегрированную среду разработки (IDE) для МК семейства MCS-51 “Keil uVision”  или  в зависимости от версии. Обычно при запуске открывается предыдущий проект, поэтому закройте его из основного меню “Project | Close Project”.



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 prMyTimer 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.30.4.

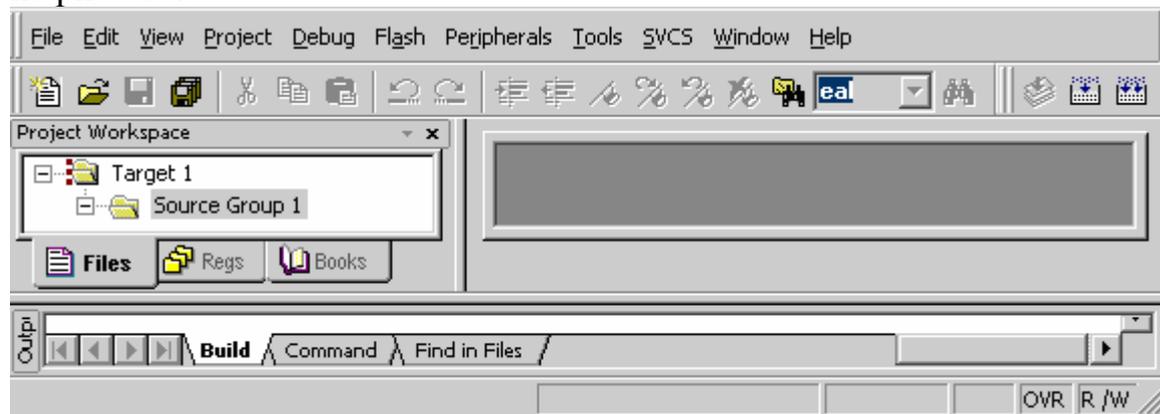
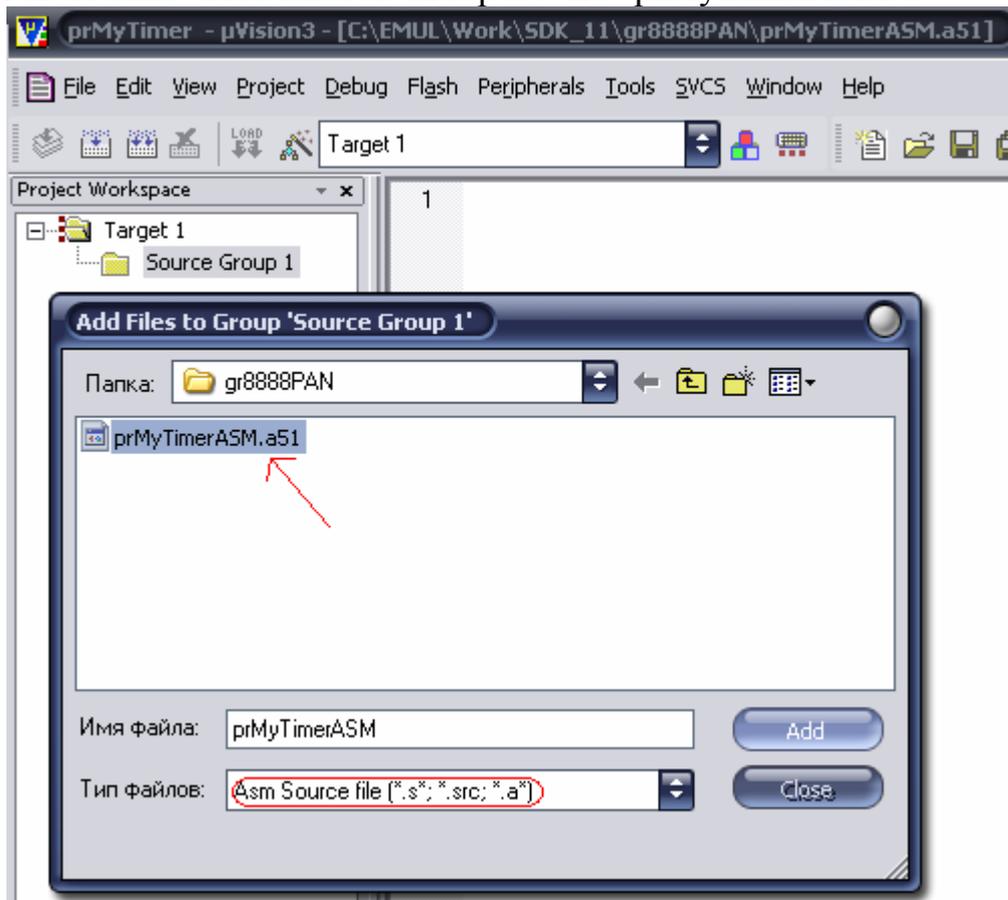


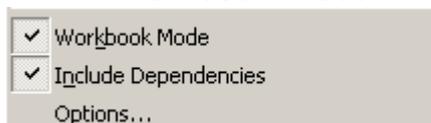
Рис.30.4

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 “prMyTimerASM.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 “prMyTimerASM”.



Теперь нажимаем на кнопки “Add” и “Close”. Отметим, что в левой панели в группе файлов сразу же появится новый файл “prMyTimerASM.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. Для этого из п. основного меню “File | New” создадим текстовый файл с именем “Text...” по умолчанию и с помощью п. меню “File | Save As” в появившемся диалоговом окне записываем новое имя файла “aduc812.h_my”. В дальнейшем этот файл будет включен в проект автоматически - строкой #include “aduc812.h_my”.

II). Создание начальной программы на ассемблере ASM51. Такая программа (аналог “Хелло Ворлд” в Си) для микроконтроллеров обычно “мигает” светодиодом.

Добавим в исходный текст операторы, которые выведут на линейку светодиодов заданный байт данных.



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_my" ;== файл с привязкой внутр. ресурсов МК к внутр. адресам
Wr_byte EQU R1 ;== байт, записываемый в ВУ или ОЗУ хранится в регистре R1
Tochka_Vhoda EQU 4000h ;== адрес входа в основную программу
t2OF EQU R0 ;== переменная t2OF (счетчик переполнений) в регистре R0
;== t2OF (timer2 OverFlow)
;=====
Write MACRO Wr_byte, reg_addr, page_num ;== макроопр-е с тремя формал. парам-и
    mov DPP, #page_num ;== номер страницы памяти
    mov DPTR, #reg_addr ;== адрес регистра ВУ или ЯП внешн. ОЗУ внутри страницы
    mov a, Wr_byte ;== байт данных из регистра R1 пересылаем в аккумулятор
    movx @DPTR, a ;== и выводим его в ВУ/ОЗУ по адресу в DPTR
    mov DPP, #0 ;== вернуться к нулевой странице памяти
ENDM ;== конец макроса
;=====
CSEG at 0 ;== начало сегмента кодов программы (диапазон 0000h..FFFFh)
;== т.к. 0000..1FFF - Flash ПЗУ, то доступны адреса начиная с 2000h,
;== верхняя граница ПРОГРАММЫ=64K (емкость внешнего ОЗУ м.б. >64K)
jmp Tochka_Vhoda ;== для отладчика (Debugger)
;== резидентный загрузчик САМ переходит по этому адресу
;=====
ORG Tochka_Vhoda ;== с этого адреса располагается код программы
;=====
main_p: ;== точка входа в основную программу
    mov Wr_byte, #0bdh ;== запись высвечиваемого кода в регистр R1
    Write Wr_byte, 3eh, 53h ;== вызов макроса с рассчитанными параметрами
;== и включение светодиодов
_8: jmp _8 ;== бесконечный цикл
;=====
END ;== конец программы
```


ВНИМАНИЕ: В вашем варианте адрес “Tochka_Vhoda” и высвечиваемый код имеют другие значения, в соответствии с заданием.
 #####

III) Пояснения к программе.  **2** Вначале расположены объявления двух переменных, которые расположены в двух рабочих регистрах МК и начальный адрес основной программы. Ниже приведено макроопределение записи/вывода байта во внешнюю ячейку памяти или ВУ во всем диапазоне адресов (000000 ... FFFFFFFF = 16МБ). Команда “movx @DPTR, a” в макросе является ключевой и пересылает байт из аккумулятора во внешнее устройство или внешнюю ячейку памяти. Подробно этот процесс был описан в лекции [“МК система с шинным интерфейсом \(или с тремя шинами\)”](#).

Директива CSEG(Code Segment) задает адрес (начало), с которого будут располагаться коды программы (по умолчанию с нулевого адреса). Используется также CSEG at XXXX – если необходимо уточнить начальный адрес XXXX сегмента. Команда “jmp Tochka_Vhoda” осуществляет переход к адресу, с которого располагается в ОЗУ, разрабатываемая программа и должна располагаться по адресу 0000h во внутренней флэш-памяти МК. При включении или рестарте МК начинает свою работу именно с этого адреса 0000h.

Следующая директива ORG “Tochka_Vhoda” переопределяет начальный адрес следующего за ней кода программы. В примере, в качестве варианта - с адреса 4000h. Метка main_p (имя произвольное) задает адрес основной программы. Команда “mov Wr_byte,#0bdh” пересылает комбинацию, высвечиваемую на диодах в регистр R1. Далее следует вызов макроопределения “Write 0bdh,3eh,53h” с тремя рассчитанными параметрами.

Завершает программу бесконечный цикл “_8: jmp _8” и директива конца программы “END”. Без этого цикла микроконтроллер, закончив выполнение программы, будет бесконтрольно выбирать неинициализированные байты из свободной области памяти и выполнять их. Это приведет либо к “зависанию”, либо к перезагрузке системы, если МК снова доберется до нулевого адреса. В худшем случае, если последовательность таких байтов образует вредоносный код, то УМК может выйти из строя. Для решения этой проблемы можно, либо оформить программу в виде бесконечного цикла, либо записать в конце программы бесконечный цикл (как в примере)

 **2** Этот фрагмент транслируем в машинный код и убедимся, что на этом коротком отрезке мы не наделали синтаксических ошибок. Для этого нажмем на кнопку  на верхней панели инструментов “uVision2”.

Синтаксических ошибок нет, но есть неопределенный (UNDEFINED) символ, о чем можно прочитать внизу в окне вывода.

```

x Build target 'Target 1'
  assembling prMyTimerASM.a51...
prMyTimerASM.a51(26) : error A45: UNDEFINED SYMBOL (PASS-2)

```

Это предупреждение относится к отсутствию определения регистра-указателя страницы памяти DPP.

#####

Поэтому в файл “aduc812.h_my” необходимо записать первую строчку:

```
DPP data XXh;== адрес регистра DPP
```

#####

Вместо XX должен быть 16-ный адрес, который можно найти из справочной таблицы в ПРИЛОЖЕНИИ-1 в конце этого пособия. Найдите ячейку регистра DPP. Справа внизу ячейки начальное значение в регистре при включении или рестарте МК. Слева – адрес, который нужно записать вместо XX в файле “aduc812.h_my”. Если снова оттранслировать программу, то в окне вывода прочитаем запись об отсутствии ошибок и предупреждений.

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

IV) 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

```

.....
:0F40000075847F90003F74BDF075840080FE32A0
.....
:00000001FF

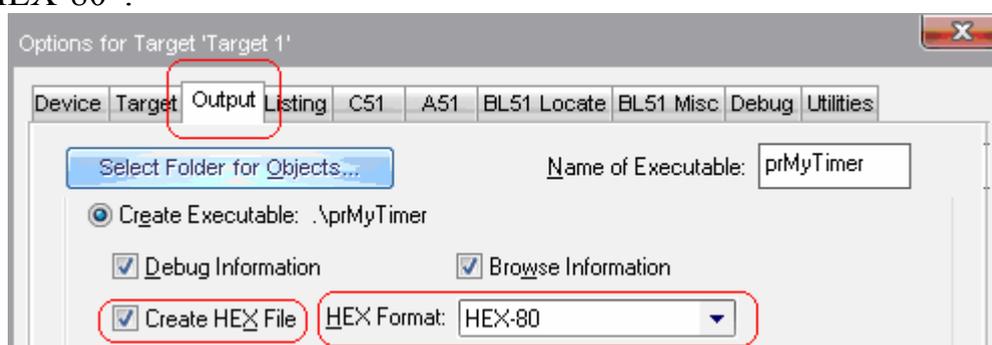
```

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(заверш. строка). Последние 2 символа в строке A0 – байт дополнения контрольной суммы всех предыдущих байтов строки до 256-

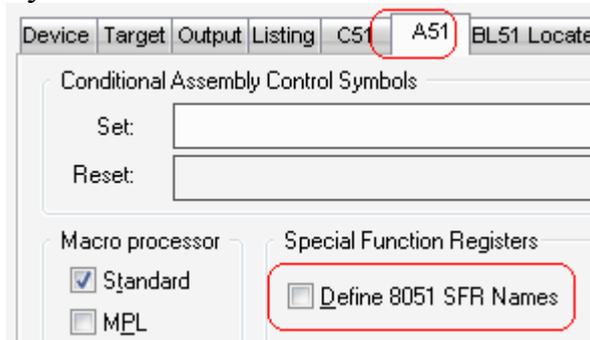
ти. Оставшиеся байты – информационные. В первой строке примера их 15(0F): 75 84 7F 90 00 3F 74 BD F0 75 84 00 80 FE 32.

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 (Точка_Входа), 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”.



Заодно снимите “птичку” с “Define 8051 SFR Names”

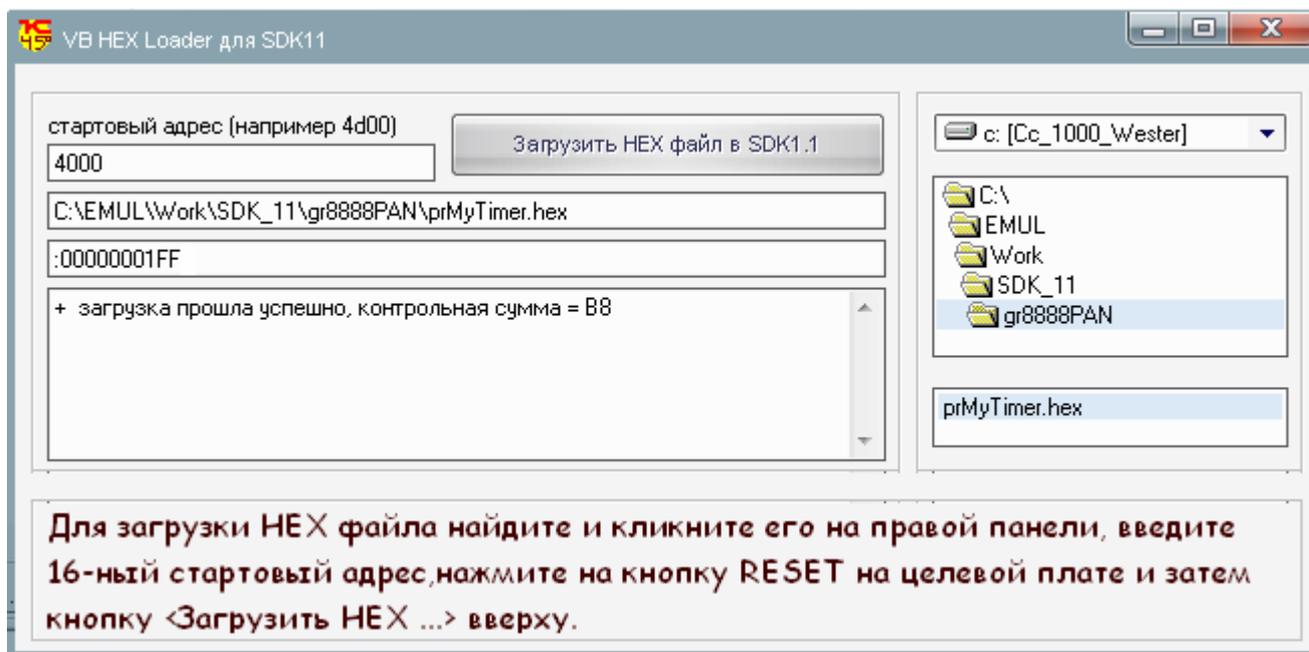


После этого выйдите из диалога, нажав на кнопку “OK”.

IV-I) 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 в рабочей папке 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...” и переведите взгляд на целевую плату УМК. Если необходимые расчеты верны, то линейка светодиодов будет иметь вид:

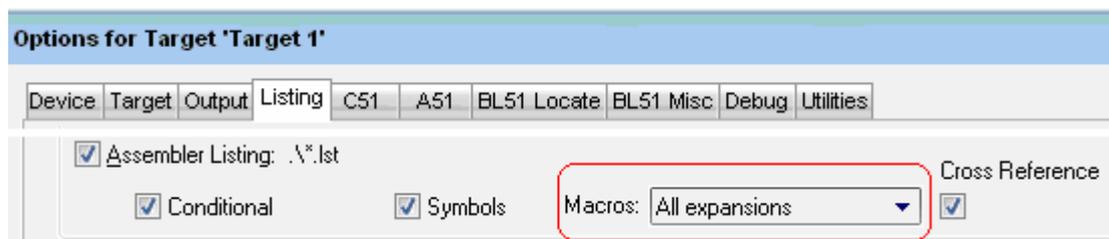


что и требовалось на данном этапе. В противном случае необходимо пересчитать адреса и/или правильно рассчитать байт данных, проверить адрес DPP и заданный адрес точки входа.

Закрывать окно загрузчика “VB HEX Loader ...” не нужно, т.к. он еще понадобится.

Результат покажите преподавателю

ПРИМЕЧАНИЕ: 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 .

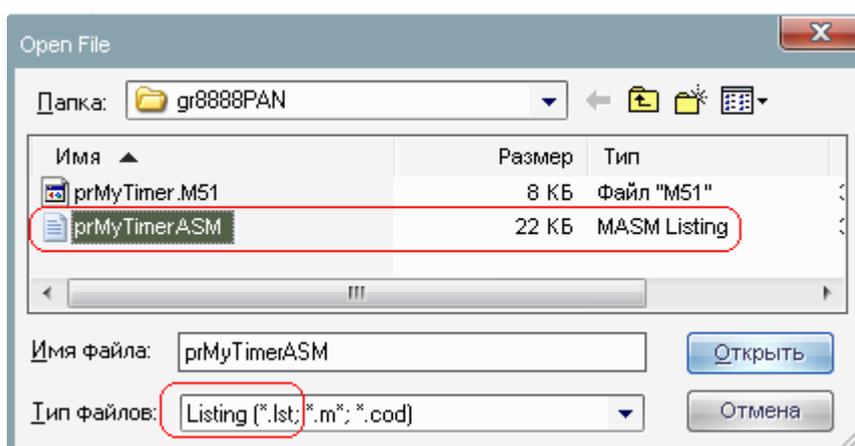
Пример синтаксической ошибки (вместо DPPR записали DPPP).

```
07 movx @DPTR, a,== и выводим его в ВУ/ЗУ по адресу в DPTR
08 mov DPPP, #0,== вернуться к нулевой странице памяти
09 ENDM,== ↑конец макроса
```

После трансляции курсор, к сожалению, не укажет на строку с ошибкой в исходном тексте макроса, а только в месте его вызова (как на рисунке). Если макросов много и они вложены друг в друга, поиск ошибки будет сильно затруднен.

```
07 movx @DPTR, a,== и выводим его в ВУ/ЗУ по адресу в DPTR
08 mov DPPP, #0,== вернуться к нулевой странице памяти
09 ENDM,== конец макроса
10 ;=====
11 CSEG,== начало сегмента кодов программы (диапазон 0000h..F
12 ;== т.к. 0000..1FFF - Flash ПЗУ, то доступны адреса на
13 ;== верхний предел обусловлен емкостью 64КБ внешнего О
14 ;=====
15 ORG 4000h,== с этого адреса будем располагать код основной
16 ;== резидентный загрузчик SAM переходит по этому адресу
17 main_p: ;== точка входа в основную программу
18 Write Wr_byte, 3eh, 53h,== вызов макроса с тремя рассчитанными
```

К нашему “большому счастью” есть выход. Откройте файл листинга из п. меню “File | Open...”.



Найдите строку с ошибкой. Поиск по служебной строке “ERROR”.

```

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

```

Вернитесь к исходному тексту, сделайте исправление(я) и запустите программу, не забыв перед ее загрузкой нажать на кнопку “RESET”.

ИСПОЛЬЗОВАНИЕ ТАЙМЕРА ДЛЯ ПЕРИОДИЧЕСКОГО УПРАВЛЕНИЯ ПРОЦЕССОМ

V). Адреса обработчиков прерываний.

 **4** Наш учебный процесс управления светодиодами на данном этапе будет контролироваться периодически. Одним из способов управления процессами через заданные интервалы времени является использование счетчиков-таймеров, встроенных во все современные МК. Причем, временные интервалы могут формироваться программными, аппаратными и программно-аппаратными средствами, с помощью подпрограмм-обработчиков аппаратных прерываний. Напишем код обработчика, переход к которому происходит в момент возникновения прерывания от переполнения счетчика-таймера. Для суммирующего счетчика-таймера переполнением является процесс смены кода с FFFFh на 0000h. В этот момент на выходе переноса счетчика-таймера формируется импульс, который может использоваться для прерывания основной программы и незамедлительного обслуживания внешнего устройства или процесса. В нашей программе процесс моделируется отображением информации на линейке светодиодов, а вывод информации производится в моменты выполнения подпрограммы-обработчика прерывания в заданные моменты времени.

В каждом типе МК существует несколько источников прерываний и соответствующих им ячеек ПЗУ, например во внутренней флэш-памяти. Адреса 0000h, 0003h, 000bh, ..., 0043h внутренней флэш-памяти МК семейства MCS-51 зарезервированы за соответствующими типами прерываний (см. ПРИЛОЖЕНИЕ 3). При возникновении прерывания МК автоматически начнет выполнять код, начиная с одного из указанных адресов. Задача программиста записать туда свои обработчики.

Но вот беда, как видно из приведенной последовательности адресов, для обработчика отводится всего-то 8 байтов (по адресу 0000 всего-то 3 байта). Поэтому сами подпрограммы располагают в других (свободных) областях памяти, а по указанным адресам записывают 3-х байтовые команды переходов LJMP XXXX к ним.

Но и это еще не выход из положения. Пользователь, во время разработки программы и ее пробных запусков в ОЗУ(SRAM), не имеет возможность подставить по адресам векторов прерываний во флэш - памяти команды перехода к своим обработчикам прерываний. Это связано с тем, что во-первых во время отладки программы в ОЗУ, во флэш памяти находится резидентный загрузчик и при каждом обновлении таблицы векторов придется его стирать и потом записывать снова, что повысит время разработки во много раз. А во-вторых, транслятор-компоновщик при последовательной разработке программы, с каждым изменением объема кода будет присваивать различные физические адреса началу обработчиков.

Эта проблемы решается следующим образом: по адресам 0000h, 0003h ... 0043h флэш-памяти, куда МК обращается при возникновении прерываний (эти адреса стандартны для архитектуры МК семейства MCS-51), заранее записаны команды переходов LJMP 20XX к адресам 2000h, 2003h, ..., 2043h, которые располагаются во внешнем ОЗУ (XRAM = eXternalRAM). Адреса могут быть и другими. Этот участок памяти называется пользовательской таблицей векторов. И вот по этим адресам 2000h, 2003h...2043h разработчик в программе подставляет команды переходов уже не к физическим адресам обработчиков своих прерываний, а к их символическим именам. Адреса этих обработчиков формируются во время компоновки программы автоматически и знать их в общем то и необязательно.

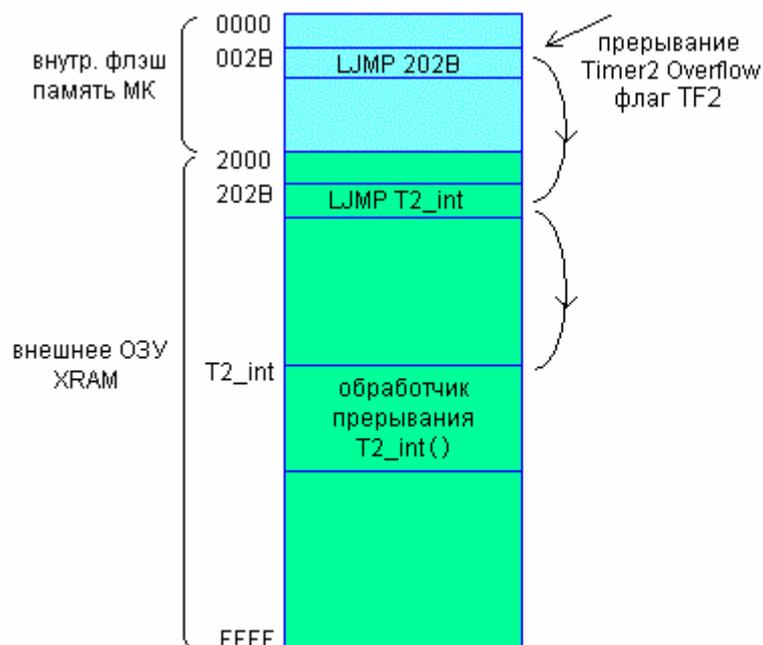


Table XXVI. Interrupt Vector Addresses

Source	Vector Address
IE0	0003H
TF0	000BH
IE1	0013H
TF1	001BH
RI + TI	0023H
TF2 + EXF2	002BH
ADCI	0033H
I2CI + ISPI	003BH
PSMI	0043H

Для нашей задачи требуется обработчик прерывания, возникающего при переполнении суммирующего счетчика таймера (т.е. при смене кода с FFFFh на 0000h). Вектор этого обработчика (точнее команда перехода) по справочнику располагается по адресу 002bh. Следовательно, по адресу 202bh необходимо поместить команду перехода LJMP T2_int, где T2_int

символический адрес (метка) подпрограммы обработчика этого прерывания. Имя T2_int - произвольное.

VI) Создание обработчика прерывания от переполнения таймера2 и расчет его параметров.

4👉 Сначала добавим в программу стандартную заготовку обработчика “T2_int” и подпрограмму “Init_Timer2” инициализации (начальной установки) таймера2.

БУДЬТЕ ВНИМАТЕЛЬНЫ: Добавлять нужно только ВЫДЕЛЕННЫЕ фрагменты и только в соответствующие места программы!
 #####

```

;=====
main p: ;== точка входа в основную программу
  call Init_Timer2
  mov Wr_byte,#0bdh ;== текущий байт в регистр R1
  Write Wr_byte,3eh,53h ;== вызов макроса с рассчитанными параметрами
  _8: jmp _8 ;== бесконечный цикл

Init_Timer2:
  ret
T2_int:
  reti
;=====
  END ;== конец программы
  
```

5👉 На этом этапе, в соответствии с техническим заданием рассчитаем параметры таймера Timer2 МК. Из трех таймеров 0, 1 и 2 выбран таймер2, т.к. только он имеет режим 16-ти битной автоматической перезагрузки “autoreload” см. рис.30.5

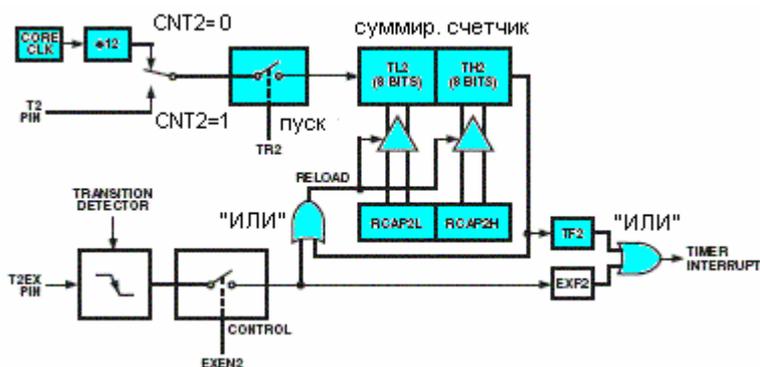


Рис.30.5

В режиме автоматической перезагрузки суммирующий счетчик TL2TH2 начинает счет, с некоторого начального значения: (TL2-мл.байт и TH2-ст.байт). В момент переполнения, т.е. при смене кода с FFFF на 0000, импульс переноса с выхода TH2 устанавливает флаг переполнения TF2 и далее через логический элемент “ИЛИ” инициирует прерывание МК (если

прерывания разрешены). Одновременно через другой ЛЭ “ИЛИ” импульс переноса перезагружает (RELOAD) счетчик начальным значением, которое хранится в паре однобайтных регистров RCAP2H и RCAP2L и далее счет продолжается с этого нового значения, а не с 0000(HEX), затем снова возникает переполнение и т.д. Поэтому код, загруженный в RCAP2H и RCAP2L, определяет период повторения прерываний. Необходимо учесть, что флаг TF2 аппаратно устанавливается, но аппаратно не сбрасывается, поэтому в обработчике прерывания необходимо программно обнулить TF2., иначе сразу же после выхода из обработчика возникнет следующее прерывание и т.д. и программа поведет себя непредсказуемо.

Выбор режима работы 2-го таймера производится с помощью управляющего регистра (регистра специальных функций - SFR) T2CON. Назначение битов этого регистра приведено в таблице:

Регистр специальных функций T2CON (регистр управления и состояния)

D7	D6	D5	D4	D3	D2	D1	D0
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	CNT2	CAP2

TF2 – флаг переполнения устанавливается аппаратно в ‘1’, если произошло переполнение таймера2, сбрасываться должен программистом.
TR2 - запуск/остановка таймера2

Назначение битов управляющего регистра T2CON

Bit	Description
TF2	Timer 2 Overflow Flag. Set by hardware on a Timer 2 overflow. TF2 will not be set when either RCLK = 1 or TCLK = 1. Cleared by user software.
EXF2	Timer 2 External Flag. Set by hardware when either a capture or reload is caused by a negative transition on T2EX and EXEN2 = 1. Cleared by user software.
RCLK	Receive Clock Enable Bit. Set by user to enable the serial port to use Timer 2 overflow pulses for its receive clock in serial port Modes 1 and 3. Cleared by user to enable Timer 1 overflow to be used for the receive clock.
TCLK	Transmit Clock Enable Bit. Set by user to enable the serial port to use Timer 2 overflow pulses for its transmit clock in serial port Modes 1 and 3. Cleared by user to enable Timer 1 overflow to be used for the transmit clock.
EXEN2	Timer 2 External Enable Flag. Set by user to enable a capture or reload to occur as a result of a negative transition on T2EX if Timer 2 is not being used to clock the serial port. Cleared by user for Timer 2 to ignore events at T2EX.
TR2	Timer 2 Start/Stop Control Bit. Set by user to start Timer 2. Cleared by user to stop Timer 2.
CNT2	Timer 2 Timer or Counter Function Select Bit. Set by the user to select counter function (input from external T2 pin). Cleared by the user to select timer function (input from on-chip core clock).
CAP2	Timer 2 Capture/Reload Select Bit. Set by user to enable captures on negative transitions at T2EX if EXEN2 = 1. Cleared by user to enable autoreloads with Timer 2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the timer is forced to autoreload on Timer 2 overflow.

При инициализации МК биты T2CON обнуляются (см. ПРИЛОЖЕНИЕ 1).

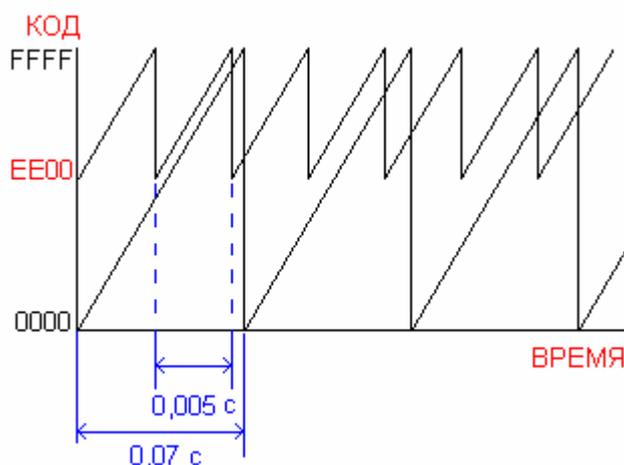
Помимо управляющего регистра T2CON у второго таймера есть еще два двухбайтовых регистра данных. TH2 и TL2 – старший и младший байт текущего значения таймера2. RCAP2H и RCAP2L – старший и младший байт регистра для перезагрузки таймера или для защелкивания текущего значения таймера (последняя конфигурация на рис. таймера не показана).

В соответствии с заданием необходимо сформировать интервал времени длительностью 1 сек. Для этого сконфигурируем таймер2 для работы в режиме таймера, а не счетчика внешних импульсов. В режиме таймера (CNT2=0) суммирующий счетчик таймера подсчитывает число импульсов от внутреннего генератора (CORE CLOCK). В режиме счетчика (CNT2=1) суммирующий счетчик таймера подсчитывает число импульсов, поступающих на внешний вход Т2 МК.

Для нашей задачи нужен режим не счетчика, а таймера (поэтому CNT2=0) с автоматической перезагрузкой (CAP2=0) при инициализации .

Подсчитаем минимальное значение частоты на выходе счетчика/таймера2. Частота CoreClock =11.059.200 Гц . Тогда $11.059.200 \text{ Гц} / 12 / 65536 = 14,0625 \text{ Гц}$ (макс. модуль счета 16-ти битного счетчика $2^{16}=65536$). Причем на 12 частота делится принудительно (см. рис.30.5). Длительность периода повторения равна $1 / 14.0625 \text{ Гц} = 0.07 \text{ сек}$. А по заданию должна быть 1 секунда. Поэтому, для получения нужного интервала в 1с, частоту $921600 \text{ Гц} = 11059200 / 12$ поделим, например на 4608(DEC) = 1200(HEX). В результате получим круглое число 200Гц (Ttimer2=5мсек=0,005сек). Т.е. за 1сек. Таймер2 200 раз переполнится и соответственно 200 раз инициирует прерывание (см. рис.30.5). Нам только остается в обработчике прерываний подсчитать 200 переполнений (т.е. отсчитать программно интервал равный 1сек)

Максимальный модуль счета счетчика/таймера2 равен 65536(DEC)=10000(HEX). Поэтому для уменьшения его модуля счета до 1200(hex) необходимо в момент переполнения загружать в СУММИРУЮЩИЙ счетчик число 10000(HEX) – 1200(HEX) = **EE00(HEX)**.



5  #####

ВНИМАНИЕ: В вашем варианте значения Ttimer2 и T могут быть другими, поэтому начальное значение **EE00**, записываемое в счетчик и число прерываний (переполнений счетчика) **200** необходимо пересчитать по следующей методике.

1). Кол-во переполнений (timer2_Overflow) $t2OF=T/Ttimer2$ – в примере = $1\text{сек}/0.005\text{сек}=200$

2). Модуль счета (Коэфф. деления) = $921600 \text{Гц} / (1 / T_{\text{timer}2}) \text{Гц}$ - в примере = $921600 / 200 = 4608$

3). Начальное значение, загружаемое в счетчик 65536 - 4608 = 60928(DEC) = **EE00(HEX)**

#####

Рассчитанное начальное значение счетчика-таймера2 EE00h должно быть загружено в регистр RCAP2 (RCAP2H – старший байт и RCAP2L - младший байт). Для того, чтобы счетчик и во время первого прохода начал счет с EE00, в него в подпрограмме “Init_Timer2” также нужно записать EE00, т.к. при включении МК регистры TH2 и TL2 счетчика обнуляются.

КСТАТИ: Рассчитывать EE00 не обязательно. Часть операций может проделать транслятор:

```
mov RCAP2H,#HIGH((not 4608)+1)
mov RCAP2L,#LOW((not 4608)+1)
```

65536-X=(not X)+1 – дополнительный до 2-х код “X” (двухбайтовый).

Разрешение или запрещение различных прерываний производится записью значений отдельных битов в управляющий регистр “IE”.

D7	D6	D5	D4	D3	D2	D1	D0
EA	EADC	ET2	ES	ET1	EX1	ET0	EX0

EA = 0 – Запрещает **все** прерывания (‘1’ – не запрещает)

EADC – разрешает “1” или запрещает “0” прерывания при завершении цикла преобразования от АЦП

ETx – разрешает “1” или запрещает “0” прерывания от таймера_x (x=0,1,2)

ES – разрешает “1” или запрещает “0” прерывания от последовательного порта UART

EXx– разрешает “1” или запрещает “0” прерывания внешние прерывания на входах (x=0,1)

Если бит EA=0, то никакие прерывания невозможны, даже если записать в биты D6...D0 единицы. Поэтому в программе мы должны записать ET2=1 и EA=1. Бит EA удобен, если нужно запретить одновременно все прерывания, не меняя значений в битах D6...D0.

Добавьте в программу выделенные фрагменты.

#####

ВНИМАНИЕ: Вместо операндов 200 и EE00 в программе должны быть ваши значения

#####

```
t2OF EQU R0,== переменная t2OF (счетчик переполнений) в регистре R0
      ;== t2OF (timer2 Overflow)
Reg1 EQU 1,== альтернативное имя регистра R1, находящегося в 1-й яч. памяти
```

```

=====
ORG 202bh,== с этого адреса переход к подпрограмме обработчику прерывания
jmp T2_int,== по переполнению от "таймера 2" - T2_int
=====

```

```

ORG Tochka_Vhoda,== с этого адреса располагается код программы

```

```

Init_Timer2:

```

```

mov t2OF,#200d,== 200*0,005сек=1сек
mov RCAP2H,#0eeh,== EE00 - дает интервал 0.005 сек.
mov RCAP2L,#0
mov TH2,#0eeh,== только для первого цикла счетчика
mov TL2,#0,
setb ET2,== разрешить прерывания от событий таймера2
setb TR2,== загрузить счетчик-таймер2
setb EA,== снять запрет со ВСЕХ прерываний
ret,== возврат из подпрограммы

```

```

T2_int:

```

```

clr TF2,== обязат. сброс флага переполнения TF2
djnz t2OF,vyhod,== декремент t2OF и если t2OF<>0 переход по адресу vyhod
mov t2OF,#200d,== секунда прошла - в счетчик прерываний снова 200
xrl Reg1,#0ffh,== инверсия кода для светодиодов (имитация процесса)
Write Wr_byte,3eh,53h,== вывод кода на светодиоды
vyhod:
reti,== возврат из подпрограммы обработчика прерывания

```

На данном этапе вид программы будет следующим (**жирным шрифтом** выделены новые фрагменты программы):

```

#include "aduc812.h_my" ;== файл с привязкой внутр. ресурсов МК к
внутр. адресам

```

```

Wr_byte EQU R1;== байт, записываемый в ВУ или ОЗУ хранится в
регистре R1

```

```

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

```

```

t2OF EQU R0;== переменная t2OF (счетчик переполнений) в регистре R0
;== t2OF (timer2 OverFlow)

```

```

Reg1 EQU 1;== альтернативное имя регистра R1, находящегося в 1-й
яч. памяти

```

```

;=====
Write MACRO Wr_byte, reg_addr, page_num;== макроопр-е с тремя
формал. парам-и

```

```

    mov DPP,#page_num;== номер страницы памяти

```

```

    mov DPTR,#reg_addr;== адрес регистра ВУ или ЯП внешн. ОЗУ

```

```

    внутри страницы

```

```

    mov a,Wr_byte;== байт данных из регистра R1 пересылаем в

```

```

    аккумулятор

```

```

    movx @DPTR,a;== и выводим его в ВУ/ОЗУ по адресу в DPTR

```

```

    mov DPP,#0;== вернуться к нулевой странице памяти

```

```

    ENDM;== конец макроса

```

```

=====
;
    CSEG at 0;== начало сегмента кодов программы (диапазон
0000h..FFFFh)
        ;== т.к. 0000..1FFF - Flash ПЗУ, то доступны адреса начиная с
2000h,
        ;== верхняя граница ПРОГРАММЫ=64К (емкость внешнего
ОЗУ м.б. >64К)
        jmp Tochka_Vhoda;== для отладчика (Debugger)
        ;== резидентный загрузчик САМ переходит по этому адресу
;=====
ORG 202bh;== с этого адреса переход к подпрограмме
обработчику прерывания
        jmp T2_int;== по переполнению от "таймера 2" - T2_int
;=====
    ORG Tochka_Vhoda;== с этого адреса располагается код программы
main_p: ;== точка входа в основную программу
        call Init_Timer2
        mov Wr_byte,#0bdh;== текущий байт в регистр R1
        Write Wr_byte,3eh,53h;== вызов макроса с рассчитанными
параметрами
_8:    jmp _8;== бесконечный цикл
;=====
Init_Timer2:
    mov t2OF,#200d;== 200 интервалов по 0.005сек = 1сек
    mov RCAP2H,#0eeh;== EE00 - дает интервал 0.005 сек.
    mov RCAP2L,#0
    mov TH2,#0eeh;== только для первого цикла счетчика
    mov TL2,#0;
    setb ET2;== разрешить прерывания от событий таймера2
    setb TR2;== запустить счетчик-таймер2
    setb EA;== снять запрет со ВСЕХ прерываний
    ret;== возврат из подпрограммы
;=====
T2_int:
    clr TF2;== обязательный сброс флага переполнения TF2
    djnz t2OF,vuhod;== 200 интервалов (по 0.005сек = 1сек) прошло ?
    mov t2OF,#200d;== секунда прошла – в счетчик прерываний
снова 200
    xrl Reg1,#0ffh;== инверсия кода, выводимого на светодиоды
    Write Wr_byte,3eh,53h
vuhod:
    reti;== возврат из подпрограммы обработчика прерывания
;=====
    END;== конец программы

```

Проведем сборку нашей программы кнопкой  или <F7> и убедимся, что появилось несколько сообщений типа “UNDEFINED SYMBOL” (при условии, что нет синтаксических ошибок).

```

x Build target 'Target 1'
  assembling prMyTimerASM.a51...
  prMyTimerASM.a51(26) : error A45: UNDEFINED SYMBOL (PASS-2)

```

Это означает, что несколько новых регистров и битов МК не определены в программе. Открываем файл “aduc812.h_my” и вставляем строки определения адресов. Адреса регистров берем из ПРИЛОЖЕНИЯ 1, а битов и флагов из ПРИЛОЖЕНИЯ 2.

ПРИМЕР:

DPP **data** 84h;== адрес **регистра** DPP (эта запись уже должна быть)

TF2 **bit** 0xxh;== адрес **флага(бита)** переполнения TF2 (xx – найдете сами)

..... и т.д.

#####

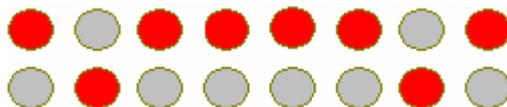
ВНИМАНИЕ: Имена остальных регистров и битов перечислены в подпрограмме Init_Timer2. Их адреса также занесите в файл “aduc812.h_my”. Будьте внимательны: регистры должны иметь квалификатор DATA, а биты BIT.

#####

После того, как определены **ВСЕ** “UNDEFINED SYMBOL’s” наконец-то появится долгожданное сообщение:

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

В очередной раз нажмите на “RESET”. Загрузите программу в УМК и убедитесь, что код на светодиодах инвертируется с заданной частотой. А это значит, что вы научились управлять процессом с помощью обработчика прерывания от таймера.



ВНИМАНИЕ: Если хоть в **ОДНОМ** критическом регистре или бите вы неправильно задали адрес, то этой “картинки” вы не увидите. **Также убедитесь, что ВСЕ переключатели в правой верхней части рабочей платы находятся в ПРАВОМ положении.**

Результат покажите преподавателю

VII) Создание обработчиков прерываний от внешних событий. В соответствии с техническим заданием, необходимо предусмотреть два внешних источника управления процессом (переключением светодиодов). В семействе МК MCS-51 имеются два инверсных входа \sim Int0 и \sim Int1 для подачи на них запросов прерываний от внешних устройств. Каждый из этих входов может быть настроен либо на низкий уровень сигнала, либо на отрицательный фронт (срез). В схеме внешние сигналы имитируются двумя переключателями, расположенными на панели управления.

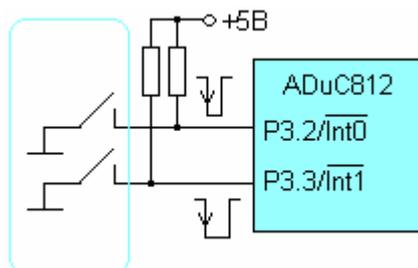


Рис.30.5.1

В программе, прерывания будут возникать при поступлении отрицательного фронта сигналов на входы \sim Int0,1. Для этого, во-первых, нужно разрешить эти прерывания битами **EX0** и **EX1** регистра IE (см. ПРИЛОЖЕНИЕ 2), а во-вторых, задать режим работы этих двух входов с помощью двух битов **IT0** и **IT1** в регистре управления TCON - Timer Control (см. ПРИЛОЖЕНИЕ 2). *Если ITx = 1, то прерывание возникнет от среза сигнала.*

ВНИМАНИЕ: Адреса этих 4-х битов из приложения-2 также запишите в файл “aduc812.h_my”
 #####

D7	D6	D5	D4	D3	D2	D1	D0
EA	EADC	ET2	ES	ET1	EX1	ET0	EX0

регистр управления прерываниями IE

Bit	Name	Description
7	EA	Written by user to enable “1” or disable “0” all interrupt sources.
6	EADC	Written by user to enable “1” or disable “0” ADC interrupt.
5	ET2	Written by user to enable “1” or disable “0” Timer 2 interrupt.
4	ES	Written by user to enable “1” or disable “0” UART serial port interrupt.
3	ET1	Written by user to enable “1” or disable “0” Timer 1 interrupt.
2	EX1	Written by user to enable “1” or disable “0” External Interrupt 1.
1	ET0	Written by user to enable “1” or disable “0” Timer 0 interrupt.
0	EX0	Written by user to enable “1” or disable “0” External Interrupt 0.

Ниже на рисунке показаны переходы к обработчику внешнего прерывания **Ext1_int** при появлении отрицательного перепада (или уровня) на входе \sim Int1 (имя Ext1_int – произвольное).

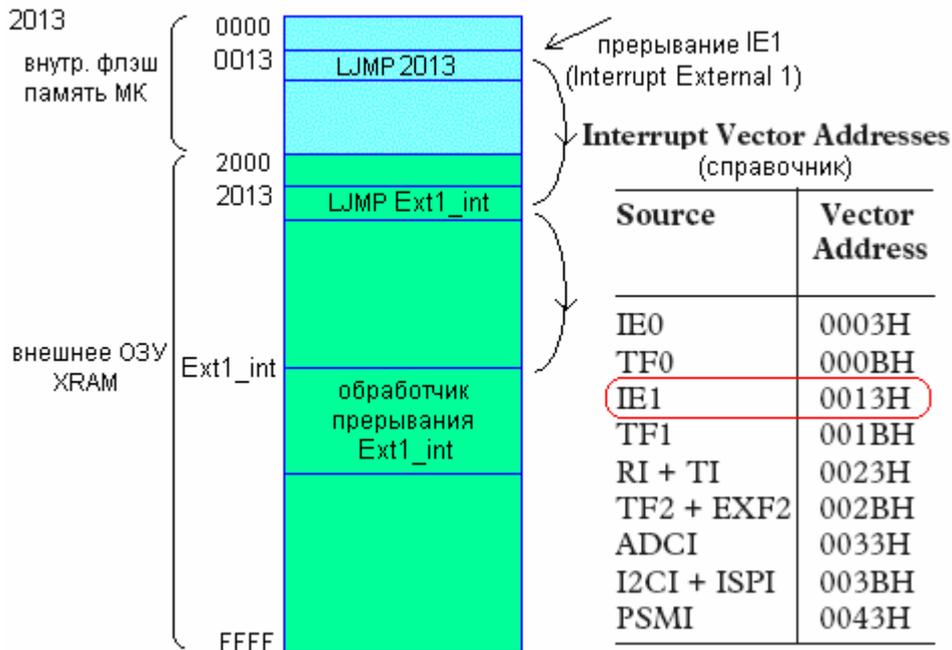


Рис.30.6

#####

ВНИМАНИЕ: Производится ОЧЕРЕДНАЯ корректировка программы

В соответствии со справочной таблицей (Рис.30.6) вместо многоточий необходимо записать адреса в указанных местах программы.

Например, для внешнего прерывания **IE1** адрес в директиве **ORG** будет равен $0013h + 2000h = 2013h$. В команде **jmp...** вместо физического адреса перехода указываем его символическое имя (метку): **Ext1_int**. Аналогично заполните еще две строчки для прерывания **IE0** (метка **Ext0_int**).

```

ORG 202bh
jmp T2 int
ORG .....
jmp .....
ORG .....
jmp .....

```

Добавьте две подпрограммы-обработчика и впишите требуемые **ДВЕ** команды.

ПОДСКАЗКА: 1) Сменить код можно одной командой ассемблера, записав в переменную **Wr_byte** другое значение, например **#55h** (найдите в программе аналогичную команду), 2) Если запускается счетчик установкой бита **TR2=1** (командой **setb TR2**), то останавливается чем? Аналогичная команда также есть в программе и в ПРИЛОЖЕНИИ-5, найдите ее.

```

=====
Ext0_int: ;обработчик прерывания Ext0_int
          ..... ;== смена высвечиваемого кода
          reti
;
Ext1_int: ;обработчик прерывания Ext1_int
          ..... ;== останавливаем счетчик
          reti
;
          END ;== конец программы
=====

```

Следующие 4 команды должны: а) разрешить 2 внешних прерывания (биты **EX0=EX1=1**) б) по отрицательному фронту (биты **IT0=IT01=1**) на входах \sim Int0,1. Например команда **setb EX0** разрешит прерывания от внешнего сигнала на входе \sim Int0. Аналогично запишите остальные три команды.

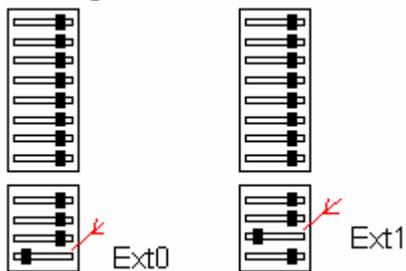
```

setb TR2 ;== запустить счетчик-таймер2
..... ;== разрешить внешние прерывания по входу P3.2(Int0)
..... ;== по отрицательному фронту сигнала
..... ;== разрешить внешние прерывания по входу P3.3(Int1)
..... ;== по отрицательному фронту сигнала
setb EA ;== снять запрет со ВСЕХ прерываний
ret ;== возврат из подпрограммы
;
T2_int:
=====

```

Запишите все 4-ре фрагмента в указанные места. Убедитесь, что объявления 4-х новых битов добавлены в файл "aduc812.h_my" и проведите окончательную сборку программы  или <F7>.

Загрузите программу в УМК. На этом этапе программа ведет себя, как и прежде, код на линейке светодиодов сменяется с заданным периодом повторения.



Теперь иницируйте внешнее прерывание на входе \sim Int0 МК, для чего переключите контакт микропереключателя из правого положения в левое, как показано на рисунке. При этом на контакте INT0 МК возникнет отрицательный перепад (1 -> 0) напряжения. МК зафиксирует появление перепада и перейдет к выполнению обработчика прерывания Ext0_int, в котором мы моделируем изменение кодовой последовательности. Код на линейке светодиодов изменится. Затем вторым контактом иницируйте внешнее прерывание на входе \sim Int1 МК. Таймер остановится и смена кодов на линейке светодиодов прекратится.

результат покажите преподавателю

На этом этапе текст программы должен иметь следующий вид (последние дополнения выделены **жирным шрифтом**):

👉6

```
#include "aduc812.h_my" ;== файл с привязкой внутр. ресурсов МК к
внутр. адресам
Wr_byte EQU R1;== байт, записываемый в ВУ или ОЗУ хранится в
регистре R1
Tochka_Vhoda EQU 4000h;== адрес входа в основную программу
t2OF EQU R0;== переменная t2OF (счетчик переполнений) в регистре R0
;== t2OF (timer2 OverFlow)
Reg1 EQU 1;== альтернативное имя регистра R1, находящегося в 1-й яч.
памяти
;=====
Write MACRO Wr_byte, reg_addr, page_num;== макроопр-е с тремя
формал. парам-и
    mov DPP,#page_num;== номер страницы памяти
    mov DPTR,#reg_addr;== адрес регистра ВУ или ЯП внешн. ОЗУ
внутри страницы
    mov a,Wr_byte;== байт данных из регистра R1 пересылаем в
аккумулятор
    movx @DPTR,a;== и выводим его в ВУ/ОЗУ по адресу в DPTR
    mov DPP,#0;== вернуться к нулевой странице памяти
    ENDM;== конец макроса
;=====
    CSEG at 0;== начало сегмента кодов программы (диапазон
0000h..FFFFh)
;== т.к. 0000..1FFF - Flash ПЗУ, то доступны адреса начиная с
2000h,
;== верхняя граница ПРОГРАММЫ=64К (емкость внешнего
ОЗУ м.б. >64К)
    jmp Tochka_Vhoda;== только для отладчика (Debugger) или готовой
программы без
;== резидентного загрузчика (резидентный загрузчик САМ
переходит по этому адресу)
;=====
    ORG 202bh;== с этого адреса переход к подпрограмме обработчику
прерывания
    jmp T2_int;== по переполнению от "таймера 2" - T2_int
.....;== промежуточный адрес Ext0_int (специфика SDK
1.1)
.....;== переход к обработчику Ext0_int
.....;== промежуточный адрес Ext1_int (специфика SDK
1.1)
.....;== переход к обработчику Ext1_int
```

```

=====
;
    ORG Tochka_Vhoda;== с этого адреса располагается код программы
main_p: ;== точка входа в основную программу
    call Init_Timer2
    mov Wr_byte,#0bdh;== запись высвечиваемого кода в регистр R1
    Write Wr_byte,3eh,53h;== вызов макроса с рассчитанными
параметрами
                                ;== и включение светодиодов
_8:   jmp _8;== бесконечный цикл
;
=====
Init_Timer2:
    mov t2OF,#200d;== 200*0,005сек=1сек
    mov RCAP2H,#0eeh;== EE00 - дает интервал 0.005 сек.
    mov RCAP2L,#0
    mov TH2,#0eeh;== только для первого цикла счетчика
    mov TL2,#0;
    setb ET2;== разрешить прерывания от событий таймера2
    setb TR2;== запустить счетчик-таймер2
    .....;== разрешить внешние прерывания по входу P3.2(Int0)
    .....;== по отрицательному фронту сигнала
    .....;== разрешить внешние прерывания по входу P3.3(Int1)
    .....;== по отрицательному фронту сигнала
    setb EA;== снять запрет со ВСЕХ прерываний
    ret;== возврат из подпрограммы
;
=====
T2_int:
    clr TF2;== обязат. сброс флага переполнения TF2
    djnz t2OF,vuhod;== декремент t2OF и если t2OF<>0 переход по
адресу vuhod
    mov t2OF,#200d;== секунда прошла - в счетчик прерываний снова
200
    xrl Reg1,#0ffh;== инверсия кода для светодиодов (имитация
процесса)
    Write Wr_byte,3eh,53h ;== вывод кода на светодиоды
vuhod:
    reti;== возврат из подпрограммы обработчика прерывания
;
=====
Ext0_int: ;обработчик прерывания IE0(Ext0_int)
    .....;== смена высвечиваемого кода
    reti
;
=====
Ext1_int: ;обработчик прерывания IE1(Ext1_int)
    .....;== останавливаем счетчик
    reti

```

```
=====
;
END;== конец программы
```

Теперь можно открыть “маленький секрет”. Запись объявлений регистров и битов МК в файл "aduc812.h_my" производилась для знакомства со справочными данными. В среде ПО “Keil uVision” (как и в IDE для других МК) файлы с объявлениями для каждого типа МК уже имеются. Например, для ADuC812 существует файл "aduc812.h", который подключается строчкой #include <aduc812.h>. Подправьте первую строку программы и убедитесь в этом, оттранслировав в последний раз проект.

Возможно, придется вручную удалить из проекта файл "aduc812.h_my".

ПРОВЕРКА РАБОТЫ ПРОГРАММЫ В СИМУЛЯТОРЕ-ОТЛАДЧИКЕ

Сначала приведем программу к окончательному виду, годному для “прошивки” во встроенное ПЗУ микроконтроллера, вместо резидентного загрузчика, который будет стерт. Для чего адрес “Точка_Vhoda” измените на 47h (как раз после таблицы векторов прерываний) и скорректируйте таблицу переходов (удалены промежуточные переходы по адресам 202bh, 2003h и 2013h). Необходимые изменения “взяты” в рамку.



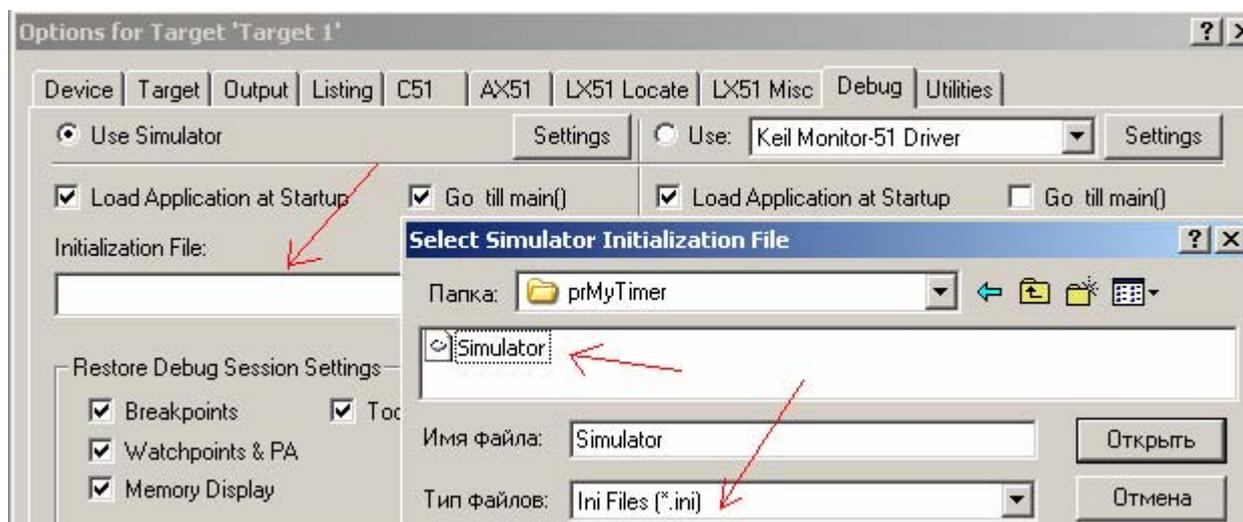
```
Wr_byte EQU R1,== байт, записываемый в ВУ или ОЗУ хранится в регистре R1
Tочка_Vhoda EQU 47h,== адрес входа в основную программу
t20F EQU R0,== переменная t20F (счетчик переполнений) в регистре R0

;
;=====
CSEG at 0,== начало сегмента кодов программы (диапазон 0000h..FFFFh)
;== т.к. 0000..1FFF - Flash ПЗУ, то доступны адреса начиная с 2000h,
;== верхняя граница ПРОГРАММЫ=64К (емкость внешнего ОЗУ м.б. >64К)
jmp Tочка_Vhoda,== только для отладчика (Debugger) или готовой программы без
;== резидентного загрузчика (резидентный загрузчик САМ переходит по этому адресу)
ORG 2bh,== с этого адреса переход к подпрограмме обработчику прерывания
jmp T2_int,== по переполнению от "таймера 2" - T2_int
ORG 3h,== адрес Ext0_int
jmp Ext0_int,== переход к обработчику Ext0_int
ORG 13h,== адрес Ext1_int
jmp Ext1_int,== переход к обработчику Ext1_int
;=====
ORG Tочка_Vhoda,== с этого адреса располагается код программы
main_p: ;== точка входа в основную программу
;=====
```

Оттранслируйте программу.

Для комфортной работы в отладчике-симуляторе создайте файл с любым именем, например “Simulator”, но с обязательным расширением “.ini”. В этом файле “Simulator.ini” запишите строчку:

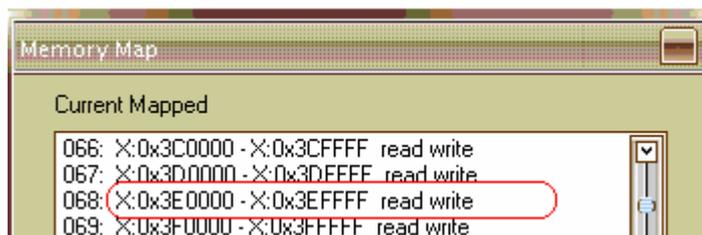
MAP X:0x000000, X:0xFFFF READ WRITE и сохраните этот файл в своей рабочей папке. Затем кликните  и на странице “Debug” поместите файл в текстовое окно Initialization File”



Теперь в симуляторе не будет остановок с предупреждениями:

*** error 65: access violation at X:0x080007 : no 'write' permission

Перейдите в симулятор <Cntr+F5> или кнопкой  и в п. меню Debugger | Memory Map убедитесь, что разрешение на чтение-запись во внешнюю память получено.



Двойным кликом или клавишей <F9> установите точку останова (breakpoint) в указанном месте подпрограммы (номер строки – 60 может быть и другим).

```

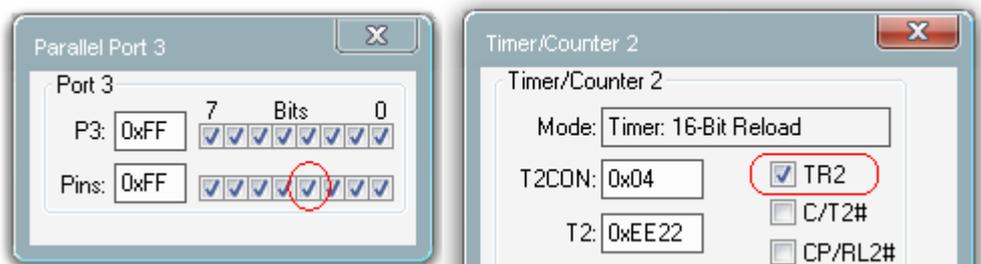
55 T2_int:
56     clr TF2,== обязат. сброс флага переполнения TF2
57     djnz t2OF,vyhod,== декремент t2OF и если t2OF<>0 переход по адресу vyhod
58     mov t2OF,#200d,== секунда прошла - в счетчик прерываний снова 200
59     xrl Reg1,#0ffh,== инверсия кода для светодиодов (имитация процесса)
60     Write Wr_byte,3eh,53h,== вывод кода на светодиоды
61     vyhod:
62     reti,== возврат из подпрограммы обработчика прерывания
63     ;=====

```

Несколько раз запустите программу кнопкой  отладчика и убедитесь в правильности своих расчетов. В строчке “sec” окна “Project Window” приращение равно точно одной секунде.

states	921623	→	states	1843223	→	states	2764823
sec	1.00002496		sec	2.00002496		sec	3.00002496
psw	0x00		psw	0x00		psw	0x00

Теперь проверим функционирование системы внешних прерываний по входам P3.2 и P3.3 микроконтроллера (Рис. 5.1). Не выходя из симулятора, откройте отладочные окна “Parallel Port 3” и “Timer/Counter 2” из п. меню “Peripherals | I/O-Ports” и “Peripherals | Timer | Timer2”.



Удалите предыдущую точку останова  и установите новую (номер строки м.б. другим).

```

68 Ext1_int: ;обработчик прерывания IE1(Ext1_int)
69     clr TR2,== останавливаем счетчик
70     reti

```

Кнопкой  верните программу в исходное состояние и снова ее запустите . Отметьте смену кода суммирующего счетчика в поле “T2:” окна “Timer/Counter 2” (бит TR2=1). Снимите “птичку” с бита 3 порта 3. На этом входе будет сымитирован отрицательный перепад, который вызовет внешнее прерывание и программа остановится в указанной точке.

```

68 Ext1_int: ;обработчик прерывания IE1(Ext1_int)
69     clr TR2,== останавливаем счетчик
70     reti

```

Снова установите бит 3 порта 3 и продолжите работу программы . Отметьте остановку счетчика в окне “Timer/Counter 2” и обнуление бита TR2. “Кликните” несколько раз по полю “TR2” и убедитесь, что именно этот бит управляет запуском/остановкой счетчика.

ВЫВОД: внешнее прерывание Ext1_int функционирует в полном соответствии с техническим заданием.

Работу прерывания Ext0_int проверьте самостоятельно примерно по такой же методике.

ПРИЛОЖЕНИЕ 1. РЕГИСТРЫ УПРАВЛЕНИЯ И СОСТОЯНИЯ

8-ми битные регистры управления и состояния – РУС (или регистры специальных функций –SFR)
МК ADuC812 (семейство MCS-51)

SPICON ¹ F8H 00H	DAC0L F9H 00H	DAC0H FAH 00H	DAC1L FBH 00H	DAC1H FCH 00H	DACCON FDH 04H		
B ¹ F0H 00H	ADCOFSL ² F1H 00H	ADCOFSH ² F2H 20H	ADCGAINL ² F3H 00H	ADCGAINH ² F4H 00H	ADCCON3 F5H 00H	SPIDAT F7H 00H	
I2CCON ¹ E8H 00H						ADCCON1 EFH 20H	
ACC ¹ E0H 00H							
ADCCON2 ¹ D8H 00H	ADCDATAL D9H 00H	ADCDATAH DAH 00H				PSMCON DFH DEH	
PSW ¹ D0H 00H		DMAL D2H 00H	DMAH D3H 00H	DMAP D4H 00H			
T2CON ¹ C8H 00H		RCAP2L CAH 00H	RCAP2H CBH 00H	TL2 CCH 00H	TH2 CDH 00H		
WDCON ¹ C0H 00H				ETIM3 C4H C9H		EDARL C6H 00H	
IP ¹ B8H 00H	ECON B9H 00H	ETIM1 BAH 52H	ETIM2 BBH 04H	EDATA1 BCH 00H	EDATA2 BDH 00H	EDATA3 BEH 00H	EDATA4 BFH 00H
P3 ¹ B0H FFH							
IE ¹ A8H 00H	IE2 A9H 00H						
P2 ¹ A0H FFH							
SCON ¹ 98H 00H	SBUF 99H 00H	I2CDAT 9AH 00H	I2CADD 9BH 55H				
P1 ^{1,3} 90H FFH							
TCON ¹ 88H 00H	TMOD 89H 00H	TL0 8AH 00H	TL1 8BH 00H	TH0 8CH 00H	TH1 8DH 00H		
P0 ¹ 80H FFH	SP 81H 07H	DPL 82H 00H	DPH 83H 00H	DPP 84H 00H		PCON 87H 00H	

ПРИЛОЖЕНИЕ 2. БИТЫ УПРАВЛЕНИЯ И СОСТОЯНИЯ
биты управления и состояния и соответствующие им регистры (справа)
МК ADuC812 (семейство MCS-51)

ISPI FFH 0	WCOL FEH 0	SPE FDH 0	SPIM FCH 0	CPOL FBH 0	CPHA FAH 0	SPR1 F9H 0	SPR0 F8H 0	BITS	SPICON ¹ F8H 00H
F7H 0	F6H 0	F5H 0	F4H 0	F3H 0	F2H 0	F1H 0	F0H 0	BITS	B ¹ F0H 00H
MDO EFH 0	MDE EEH 0	MCO EDH 0	MDI ECH 0	I2CM EBH 0	I2CRS EAH 0	I2CTX E9H 0	I2CI E8H 0	BITS	I2CCON ¹ E8H 00H
E7H 0	E6H 0	E5H 0	E4H 0	E3H 0	E2H 0	E1H 0	E0H 0	BITS	ACC ¹ E0H 00H
ADC1 DFH 0	DMA DEH 0	CCONV DDH 0	SCONV DCH 0	CS3 DBH 0	CS2 DAH 0	CS1 D9H 0	CS0 D8H 0	BITS	ADCCON ¹ D8H 00H
CY D7H 0	AC D6H 0	F0 D5H 0	RS1 D4H 0	RS0 D3H 0	OV D2H 0	FI D1H 0	P D0H 0	BITS	PSW ¹ D0H 00H
TF2 CFH 0	EXF2 CEH 0	RCLK CDH 0	TCLK CCH 0	EXEN2 CBH 0	TR2 CAH 0	CNT2 C9H 0	CAP2 C8H 0	BITS	T2CON ¹ C8H 00H
PRE2 C7H 0	PRE1 C6H 0	PRE0 C5H 0		WDR1 C3H 0	WDR2 C2H 0	WDS C1H 0	WDE C0H 0	BITS	WDCON ¹ C0H 00H
PSI BFH 0	PADC BEH 0	PT2 BDH 0	PS BCH 0	PT1 BBH 0	PX1 BAH 0	PT0 B9H 0	PX0 B8H 0	BITS	IP ¹ B8H 00H
RD B7H 1	WR B6H 1	T1 B5H 1	To B4H 1	INT1 B3H 1	INT0 B2H 1	TxD B1H 1	RxD B0H 1	BITS	P3 ¹ B0H FFH
EA AFH 0	EADC AEH 0	ET2 ADH 0	ES ACH 0	ET1 ABH 0	EX1 AAH 0	ET0 A9H 0	EX0 A8H 0	BITS	IE ¹ A8H 00H
A7H 1	A6H 1	A5H 1	A4H 1	A3H 1	A2H 1	A1H 1	A0H 1	BITS	P2 ¹ A0H FFH
SM0 9FH 0	SM1 9EH 0	SM2 9DH 0	REN 9CH 0	TB8 9BH 0	RB8 9AH 0	TI 99H 0	RI 98H 0	BITS	SCON ¹ 98H 00H
97H 1	96H 1	95H 1	94H 1	93H 1	92H 1	T2EX 91H 1	T2 90H 1	BITS	P1 ^{1,3} 90H FFH
TF1 8FH 0	TR1 8EH 0	TF0 8DH 0	TR0 8CH 0	IE1 8BH 0	IT1 8AH 0	IE0 89H 0	IT0 88H 0	BITS	TCON ¹ 88H 00H
87H 1	86H 1	85H 1	84H 1	83H 1	82H 1	81H 1	80H 1	BITS	P0 ¹ 80H FFH

ПРИЛОЖЕНИЕ 3. ТАБЛИЦА ВЕКТОРОВ ПЕРЕРЫВАНИЙ

Table XXVI. Interrupt Vector Addresses

Source	Vector Address
IE0	0003H
TF0	000BH
IE1	0013H
TF1	001BH
RI + TI	0023H
TF2 + EXF2	002BH
ADCI	0033H
I2CI + ISPI	003BH
PSMI	0043H

Table XXV. Priority within an Interrupt Level

Source	Priority	Description
PSMI	1 (Highest)	Power Supply Monitor Interrupt
IE0	2	External Interrupt 0
ADCI	3	ADC Interrupt
TF0	4	Timer/Counter 0 Interrupt
IE1	5	External Interrupt 1
TF1	6	Timer/Counter 1 Interrupt
I2CI + ISPI	7	I ² C/SPI Interrupt
RI + TI	8	Serial Interrupt
TF2 + EXF2	9 (Lowest)	Timer/Counter 2 Interrupt

ПРИЛОЖЕНИЕ 4. РЕГИСТРЫ УПРАВЛЕНИЯ И СОСТОЯНИЯ ТАЙМЕРА2

Некоторые регистры управления и состояния – РУС (Special Function Registers – SFR)

Регистр разрешения прерываний **IE** (Interrupt Enable Register)

D7	D6	D5	D4	D3	D2	D1	D0
EA	EADC	ET2	ES	ET1	EX1	ET0	EX0

EA = 0 – Запрещает все прерывания ('1' – не запрещает)

EADC – разрешает "1" или запрещает "0" прерывания при завершении цикла преобразования от АЦП

ETx – разрешает “1” или запрещает “0” прерывания от таймера_x (x=0,1,2)

ES – разрешает “1” или запрещает “0” прерывания от последовательного порта UART

EXx– разрешает “1” или запрещает “0” прерывания внешние прерывания на входах (x=0,1)

Регистр управления таймером-счетчиком2 T2CON

D7	D6	D5	D4	D3	D2	D1	D0
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	CNT2	CAP2

TF2 – флаг переполнения, устанавливается в ‘1’, если произошло переполнение таймера2

TR2 - запуск таймера2

Регистр перезагрузки/захвата младшего байта таймера2 RCAP2L

D7	D6	D5	D4	D3	D2	D1	D0

Регистр перезагрузки/захвата/ старшего байта таймера2 RCAP2H

D7	D6	D5	D4	D3	D2	D1	D0

ПРИЛОЖЕНИЕ 5. НЕКОТОРЫЕ КОМАНДЫ И ДИРЕКТИВЫ АССЕМБЛЕРА MCS-51

DST, SRC – операнд приемник, источник (байты)

operand – операнд (байт)

bit – однобитовый операнд

xrl DST, SRC - “поразрядное исключяющее ИЛИ”

anl DST, SRC - “поразрядное И”

clr bit - обнулить бит

setb bit - установить бит

cjne a, operand, address - сравнить содержимое аккумулятора с операндом и, если они не равны перейти по адресу.

djnz operand, адрес – уменьшить операнд на 1 и перейти по адресу, если результат $\neq 0$

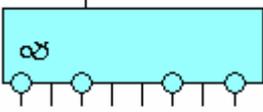
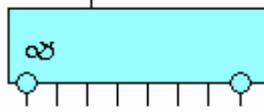
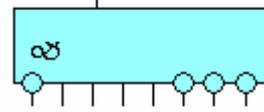
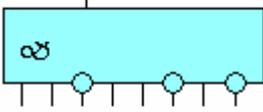
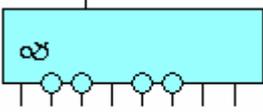
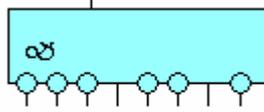
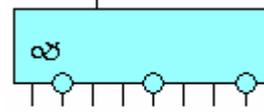
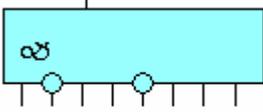
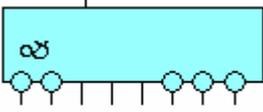
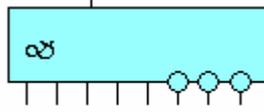
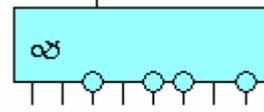
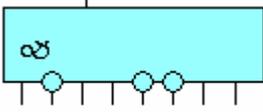
movx @dptr, a - переслать содержимое аккумулятора в ВУ/ЗУ с адресом, который находится в двухбайтовом регистре DPTR.

movx a, @dptr - в обратном направлении

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

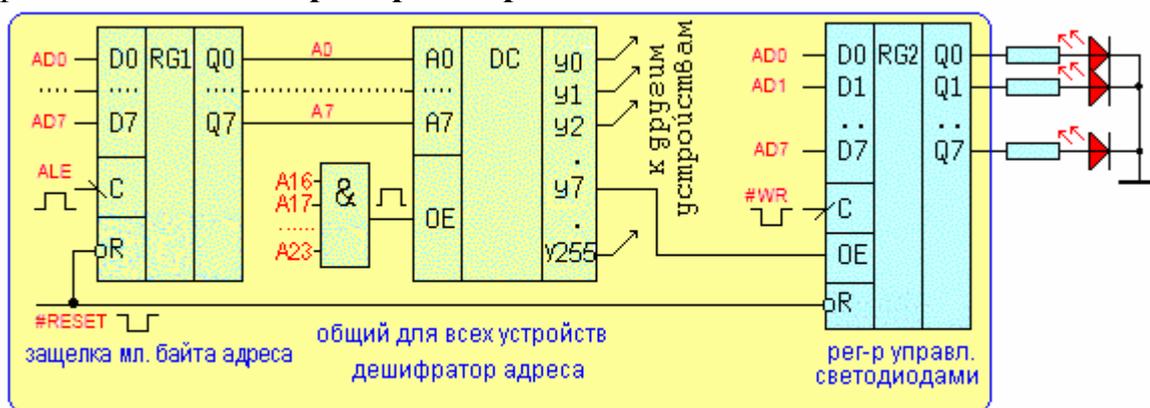
call address – вызов подпрограммы по адресу address и в стек помещается адрес возврата.

ВАРИАНТЫ ТЕХНИЧЕСКОГО ЗАДАНИЯ

 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y15 Tochka_Vhoda 3000h Ttimer2=50msec T=3сек</p>  <p>СТ. МЛ.</p>	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y151 Tochka_Vhoda 5100h Ttimer2=2.5msek T=0.5сек</p>  <p>СТ. МЛ.</p>	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y55 Tochka_Vhoda 2500h Ttimer2=25msec T=1сек</p>  <p>СТ. МЛ.</p>	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y143 Tochka_Vhoda 7000h Ttimer2=10msec T=2сек</p>  <p>СТ. МЛ.</p>
 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y199 Tochka_Vhoda 3650h Ttimer2=50msec T=4сек</p>  <p>СТ. МЛ.</p>	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y23 Tochka_Vhoda 2A00h Ttimer2=20msec T=5сек</p>  <p>СТ. МЛ.</p>	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y71 Tochka_Vhoda 4E00h Ttimer2=8msec T=1сек</p>  <p>СТ. МЛ.</p>	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y31 Tochka_Vhoda 5D00h Ttimer2=4msec T=1сек</p>  <p>СТ. МЛ.</p>
 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y47 Tochka_Vhoda 6AB0h Ttimer2=25msec T=0.5сек</p>  <p>СТ. МЛ.</p>	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y135 Tochka_Vhoda 44E0h Ttimer2=40msec T=2сек</p>  <p>СТ. МЛ.</p>	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y39 Tochka_Vhoda 5E30h Ttimer2=50msec T=0.5сек</p>  <p>СТ. МЛ.</p>	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=79 Tochka_Vhoda 2A00h Ttimer2=20msec T=2сек</p>  <p>СТ. МЛ.</p>

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 битного адреса RG2. Выполнение команды “MOVX @DPTR,A” и “MOVX A,@DPTR” 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”.



фрагмент ПЛИС EPM3064

- 4) Уметь комментировать и понимать работу основной программы и подпрограмм (применительно к рисунку п.3).

```

#include "aduc812.h_my" ;== файл с привязкой внутр. ресурсов МК к адресам
Wr_byte EQU R1;== байт, записываемый в ВУ или ОЗУ хранится в регистре R1
Tochka_Vhoda EQU 47h;== адрес входа в основную программу
t2OF EQU R0;== переменная t2OF (счетчик переполнений) в регистре R0
;== t2OF (timer2 Overflow)
Reg1 EQU 1;== альтерн. имя регистра R1, находящегося в 1-й яч. памяти
;=====
Write MACRO Wr_byte, reg_addr, page_num;== макроопр-е с тремя парам-и
    mov DPP,#page_num;== номер страницы памяти
    mov DPTR,#reg_addr;== адрес регистра ВУ или ЯП ОЗУ внутри страницы
    mov a,Wr_byte;== байт данных из регистра R1 пересылаем в аккумулятор
    movx @DPTR,a;== и выводим его в ВУ/ОЗУ по адресу в DPTR
    mov DPP,#0;== вернуться к нулевой странице памяти
    ENDM;== конец макроса
;=====
CSEG at 0;== начало сегмента кодов программы (диапазон 0000h..FFFFh)
;== т.к. 0000..1FFF - Flash ПЗУ, то доступны адреса начиная с 2000h,
;== верхняя граница ПРОГРАММЫ=64К (емкость внешнего ОЗУ м.б. >64К)
jmp Tochka_Vhoda;
ORG 2bh;== адреса перехода к подпрограмме обработчику прерывания
jmp T2_int;== по переполнению от "таймера 2" - T2_int
ORG 3h;== адрес Ext0_int
jmp Ext0_int;== переход к обработчику Ext0_int
ORG 13h;== адрес Ext1_int
jmp Ext1_int;== переход к обработчику Ext1_int
;=====
ORG Tochka_Vhoda;== с этого адреса располагается код программы
main_p: ;== точка входа в основную программу
    call Init_Timer2
    mov Wr_byte,#0bdh;== запись высвечиваемого кода в регистр R1
    Write Wr_byte,3fh,7fh;== вызов макроса с рассчитанными параметрами
;== и включение светодиодов
_8:
    jmp _8;== бесконечный цикл
;=====
;=====
Init_Timer2:
    mov t2OF,#200d;== 200*0,005сек=1сек
    mov RCAP2H,#HIGH((not 4608)+1);== EE00 - дает интервал 0.005 сек.
    mov RCAP2L,#LOW((not 4608)+1)
    mov TH2,#0eeh;== только для первого цикла счетчика
    mov TL2,#0;
    setb ET2;== разрешить прерывания от событий таймера2
    setb TR2;== запустить счетчик-таймер2
    setb EX0;== разрешить внешние прерывания по входу P3.2(Int0)
    setb ITO;== по отрицательному фронту сигнала
    setb EX1;== разрешить внешние прерывания по входу P3.3(Int1)
    setb IT1;== по отрицательному фронту сигнала
    setb EA;== снять запрет со ВСЕХ прерываний
    ret;== возврат из подпрограммы

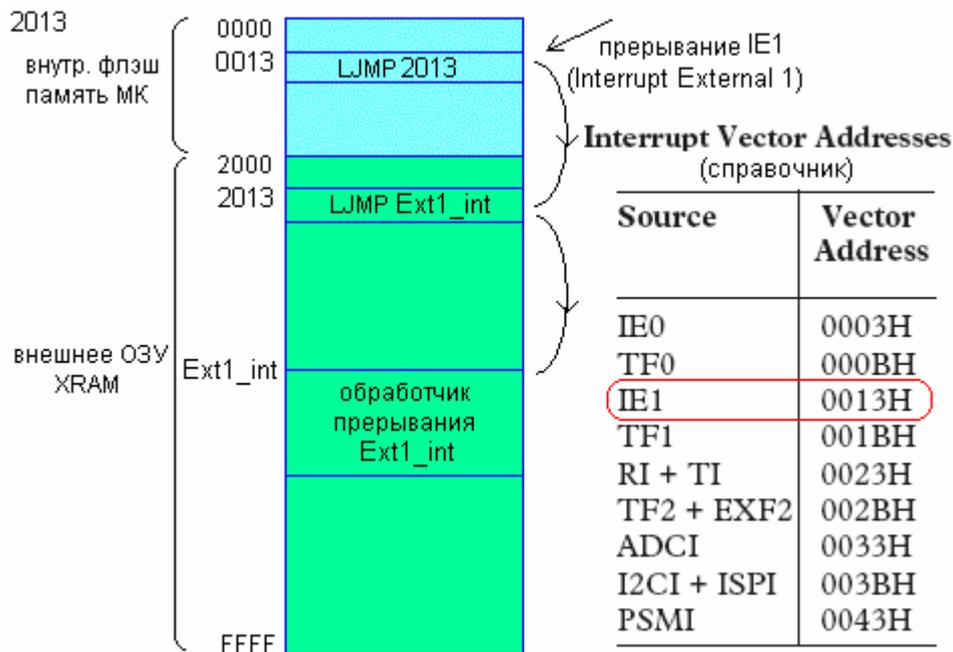
```

```

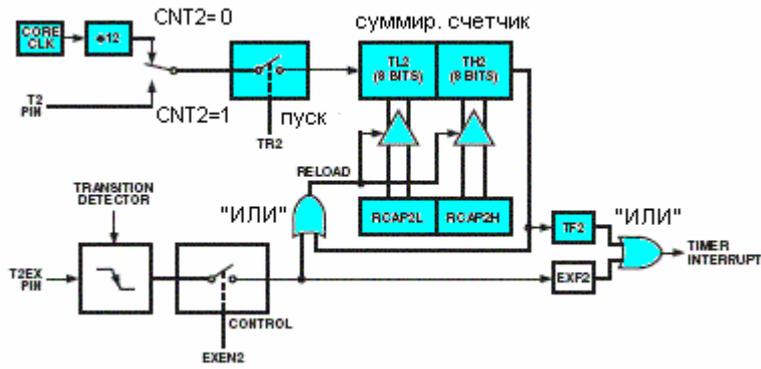
;=====
T2_int:
  clr TF2,;== обязат. сброс флага переполнения TF2
  djnz t2OF,vuhod,;== декремент t2OF и если t2OF<>0 переход по адресу vuhod
  mov t2OF,#200d,;== секунда прошла - в счетчик прерываний снова 200
  xrl Reg1,#0ffh,;== инверсия кода для светодиодов (имитация процесса)
  Write Wr_byte,3fh,7fh ;== вывод кода на светодиоды
vuhod:
  reti,;== возврат из подпрограммы обработчика прерывания
;=====
Ext0_int: ;обработчик прерывания IE0(Ext0_int)
  mov Wr_byte,#0aah,;== смена высвечиваемого кода
  reti
;=====
Ext1_int: ;обработчик прерывания IE1(Ext1_int)
  clr TR2,;== останавливаем счетчик
  reti
;=====
  END,;== конец программы

```

- 5). Назначение битов управляющих байтов регистра **IE**: EA, ET2, EX1, EX0 и регистра **T2CON**: TR2 и TF2.
 8) Пояснить систему прерываний по этому рисунку.



- 6) Назначение регистров **RCAP2L** и **RCAP2H** и работа таймера 2 (по рисунку).



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

ЛАБОРАТОРНАЯ РАБОТА № 31 “ПРОГРАММИРОВАНИЕ КЛАВИАТУРЫ”

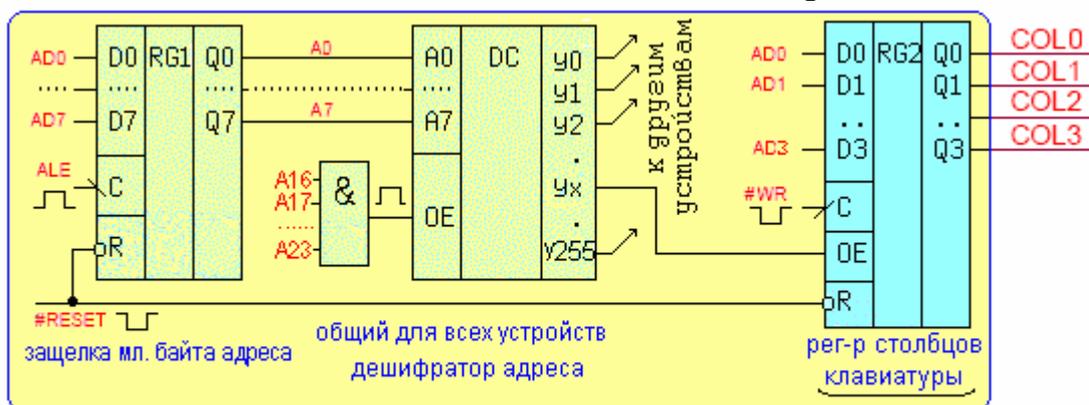
ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Необходимо написать программу, которая управляет работой клавиатуры и отображает ASCII код нажатой клавиши на линейке светодиодов.

РАСЧЕТ АДРЕСА РЕГИСТРА УПРАВЛЕНИЯ КЛАВИАТУРОЙ

В простых схемах, когда достаточно свободных выходов портов, светодиоды можно подключать через ограничивающие ток резисторы, прямо к этим выходам. 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.30.3 приведен фрагмент схемы, запрограммированной в ПЛИС EPM3064. В регистр 2 периодически посылается код, в котором 3 бита равны “1” и один бит равен “0”. Причем в каждом периоде положение нуля смещается. Такая смена кодов называется “бегущим нулем”.

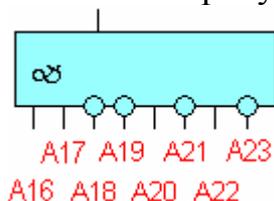
ВНИМАНИЕ: 1) *Схема скорректирована* для большего числа вариантов заданий. Часть входов элемента “И” – *инверсные*.



фрагмент ПЛИС EPM3064

Рис.30.3

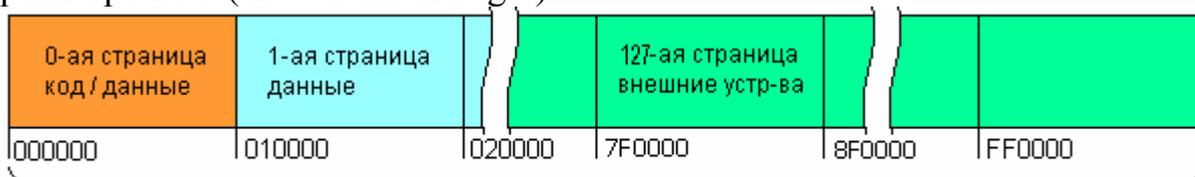
1 **Вариант задания** 1) Задействован 62-й выход дешифратора, т.е. $Y_x = Y_{62}$, 2) Четыре входа элемента “И” A_{22}, A_{20}, A_{17} и A_{16} – прямые, остальные – инверсные (как показано на рисунке).



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 (OE, CS и т.д.). На рис.30.3 сигнал Y_x разрешает выходы по входу OE (Output Enable) регистра RG2. Далее за стробом ALE следует строб записи данных #WR, который защелкивает в RG2 четыре мл. бита $D_3..D_0$ (“бегущий 0”), поступающие по ШАД.

Рассчитаем адрес для обращения к регистру столбцов клавиатуры 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 $A_{23}, A_{22}, A_{21}, A_{20}, A_{19}, A_{18}, A_{17}, A_{16} = 01010011$ (BIN) = 53 (HEX). Выход Y_{62} 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.30.3) и варианта задания запишем адрес регистра столбцов клавиатуры:



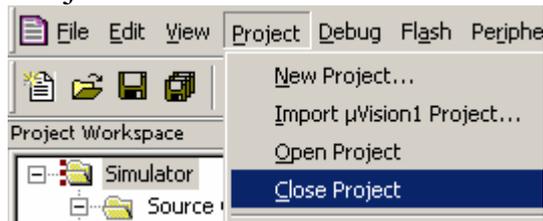
Как обычно, неиспользуемые биты ША могут быть любыми, например нулевыми и адрес RG2 будет в этом случае равен 0101 0011 0000 0000 0011 1110=53003E(HEX) или (DPP)=53h, (DPTR)=003E=3Eh.

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

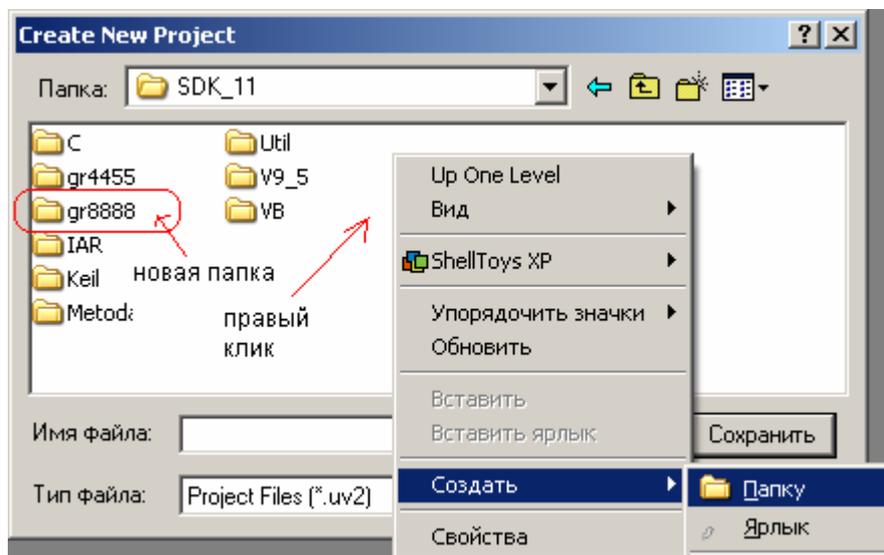
ВНИМАНИЕ: Вам необходимо вместо чисел 53h и 3Eh рассчитать свои значения (DPP) и (DPTR) в соответствии с полученным заданием.
 #####

РАЗРАБОТКА ПРОГРАММЫ

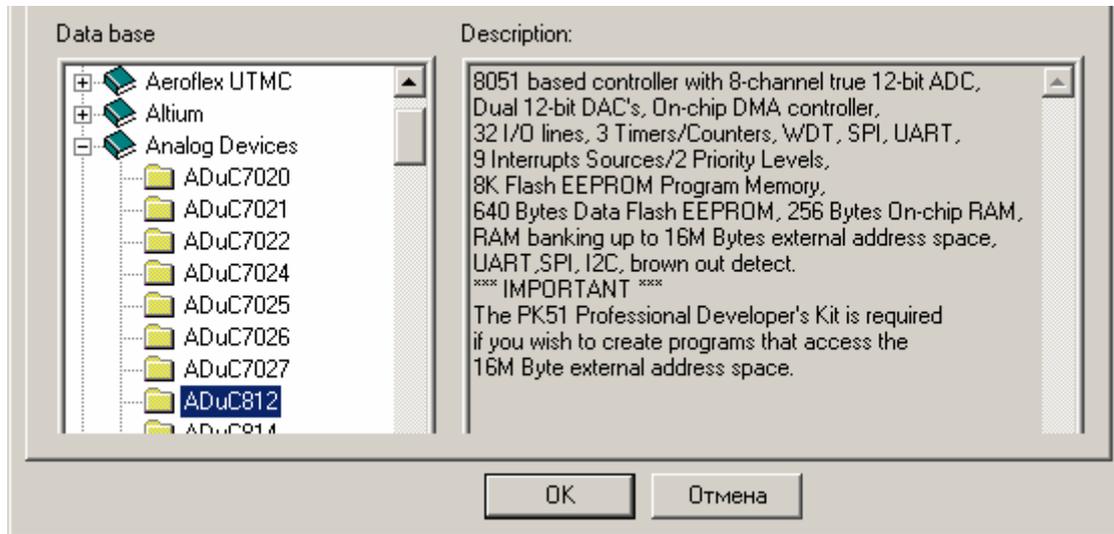
I) Создание шаблона программы на ассемблере. Запустите интегрированную среду разработки (IDE) для МК семейства MCS-51 “Keil uVision”  или  в зависимости от версии. Обычно при запуске открывается предыдущий проект, поэтому закройте его из основного меню “Project | Close Project”.



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 prMyKey затем кликните по кнопке “Сохранить”. Откроется диалоговое окно “Select Device for Target ...”.



Выбираем МК ADuC812 и ждем 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.30.4.

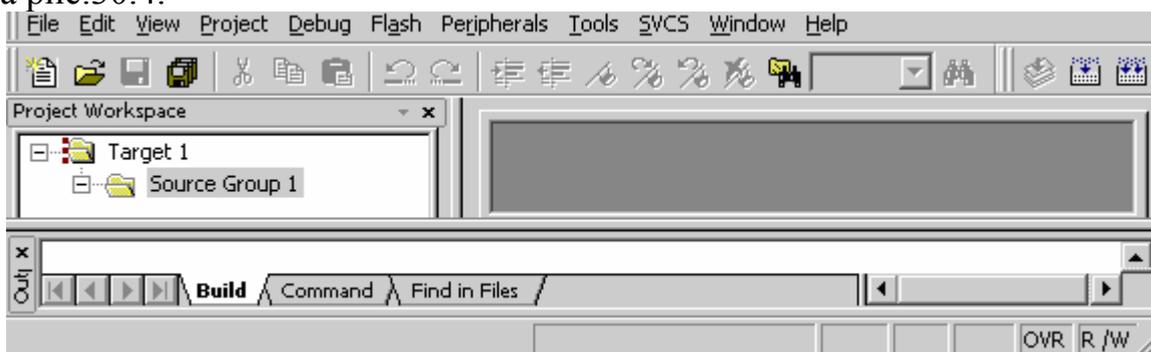
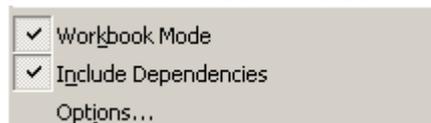


Рис.30.4

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 “prKey_LED.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 “prKey_LED.a51”.

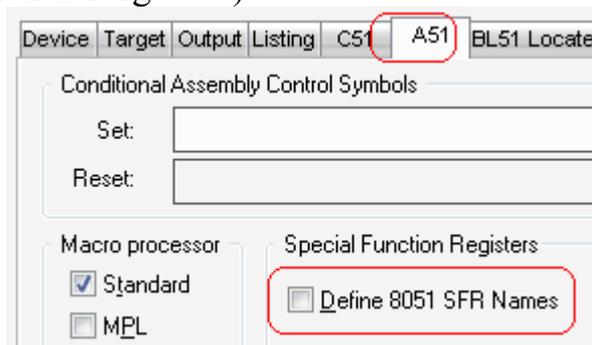
Нажимаем на кнопки “Add” и “Close”. Отметим, что в левой панели в группе файлов сразу же появится новый файл “ prKey_LED.a51”, пока пустой.

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”

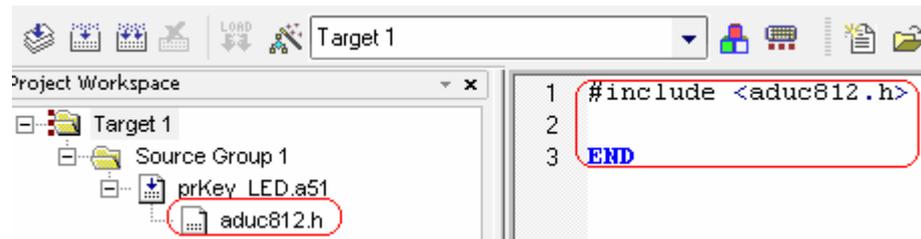


В поздних версиях IDE Keil uVision этот режим устанавливается из п. “Edit | Configuration | Editor”.

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 “prKey_LED.a51” записываем строку: #include <aduc812.h> и директиву END, которая 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.



II). Разработка программы фиксирующей момент нажатия на одну клавишу.

Напишем программу, которая будет реагировать на нажатие ОДНОЙ из 16-ти клавиш, например на **клавишу “1”** и выводить ASCII код этой клавиши на светодиоды. На рисунке 5 приведена упрощенная типовая схема подключения клавиатуры к шине данных - ШД (или порту). Буферные элементы на схеме не показаны.

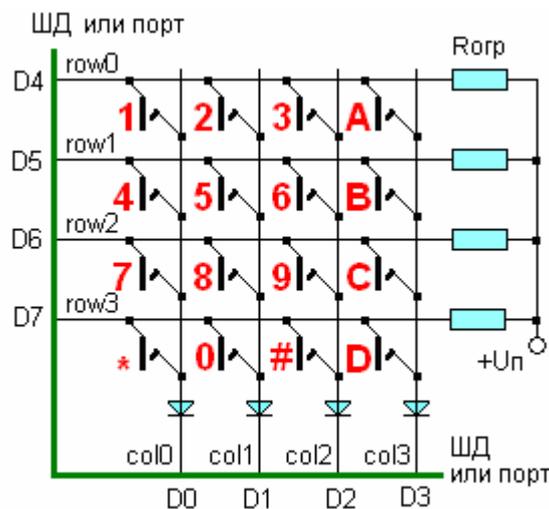


Рис. 5

На линии ШД D0..D3 (называемые также ЛИНИЯМИ СДВИГА или SHIFT LINES) поочередно подается “0” (т.н. “бегущий ноль”). Одновременно считывается код с линий ШД D4..D7 (которые еще называются ЛИНИЯМИ ВОЗВРАТА или RETURN LINES). Если не нажата НИ ОДНА из клавиш, т.е. не замкнут ни один контакт, с линий D4..D7 будет считываться высокий уровень близкий к +Uп, т.е. логическая “1” (D7..D4=1111). При нажатии хотя бы на одну из клавиш, например на кл.”1”, нулевой уровень попадет через замкнутые контакты кл.”1” и будет считан в МК (код D7..D4=1110). В общем случае, если код **D7..D4 не равен 1111**, то это является ПРИЗНАКОМ замыкания хотя бы одного из контактов.

Таким образом, для определения нажатия на клавишу ‘1’ необходимо записать в регистр клавиатуры код D7D6D5D4D3D2D1D0 = **11111110** (имеют значения только 4 младших бита). Если контакты кл.’1’ замкнуты, то ‘0’ из столбца ‘col0’ попадет в строку ‘row0’ и с помощью операнда маски D7D6D5**D4**D3D2D1D0 = **00010000** может быть считан

(имеют значения только 4 старших бита). Это связано с тем, что ячейки памяти и регистры микроконтроллера 8-ми битные и если линий сдвига и линий возврата только 4-ре, то операнд в программе все равно должен быть 8-ми битным. Просто неиспользуемые биты не учитываются или маскируются. Далее в программе это будет видно.

ВНИМАНИЕ: В вашем варианте клавиша другая и эти 2 операнда имеют также другие значения.
 #####

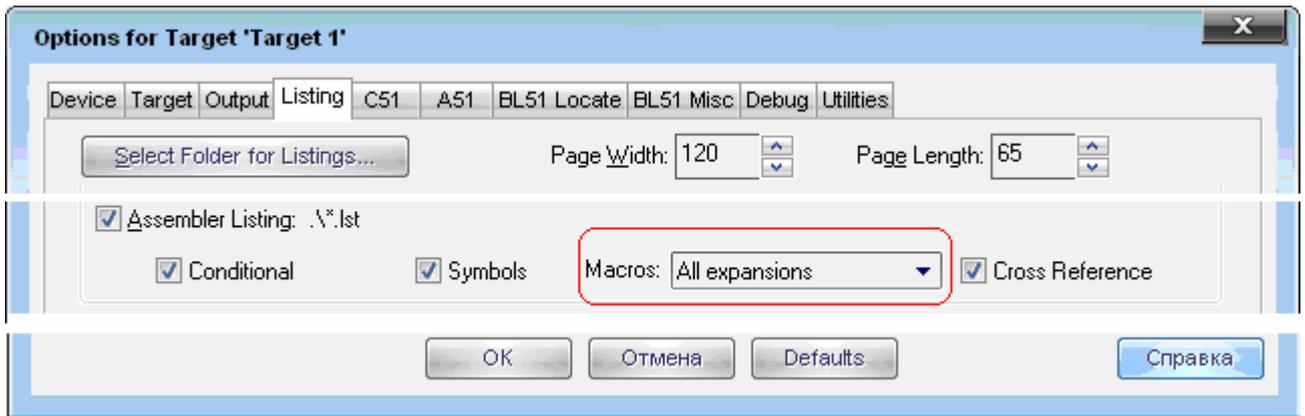
Для начала в программу запишем необходимые вспомогательные фрагменты, известные по лабораторной работе №30.

```
#include <aduc812.h>
Rd_ROW EQU R2;== читаемый код строки хранится в регистре R2
Wr_COL EQU R3;== записываемый код столбца хранится в регистре R3
Mask EQU R4;== маска для выделения линии строки хранится в регистре R4
DPP_page_num EQU 53h;== код номера страницы внешних устройств
DPTR_KEYreg_num EQU 3eh;== номер (адрес) регистра клавиатуры
DPTR_LEDreg_num EQU 3fh;== номер (адрес) регистра светодиодов (лаб. раб. №30)
Tochka_Vhoda EQU 4000h;== адрес входа в основную программу
;=====
Write MACRO Wr_byte, reg_addr;== макроопр-е с двумя формал. парам-и
    mov DPTR,#reg_addr;== адрес регистра ВУ или ЯП внешн. ОЗУ внутри страницы
    mov a,Wr_byte;== байт данных Wr_byte пересылаем в аккумулятор
    movx @DPTR,a;== и выводим его в ВУ/ОЗУ по адресу в DPTR
    ENDM;== конец макроса
;=====
Read MACRO Rd_byte, reg_addr;== макроопр-е с двумя формал. парам-и
    mov DPTR,#reg_addr;== адрес регистра ВУ или ЯП внешн. ОЗУ внутри страницы
    movx a,@DPTR;== вводим байт из ВУ/ОЗУ по адресу в DPTR
    mov Rd_byte,a;== байт данных из аккумулятора пересылаем в переменную Rd_byte
    ENDM;== конец макроса
;=====
    CSEG at 0
    jmp Tochka_Vhoda
;=====
    ORG Tochka_Vhoda;== с этого адреса располагается код программы
main_prog: ;== точка входа в основную программу
_8:
    jmp _8;== бесконечный цикл
;=====
END
```

Оттранслируйте программу  и убедитесь, что ошибок нет.

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

ПРИМЕЧАНИЕ: 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 положение ошибки внутри тела макроса транслятор не указывает. Если листинг не обновляется, попробуйте запустить трансляцию программы кнопкой , а не . **Пример синтаксической ошибки** (вместо DPP записали DPPP).

```

07 movx @DPTR, a,== и выводим его в ВУ/ЗУ по адресу в DPTR
08 mov DPPP, #0,== вернуться к нулевой странице памяти
09 ENDM,== ↑конец макроса

```

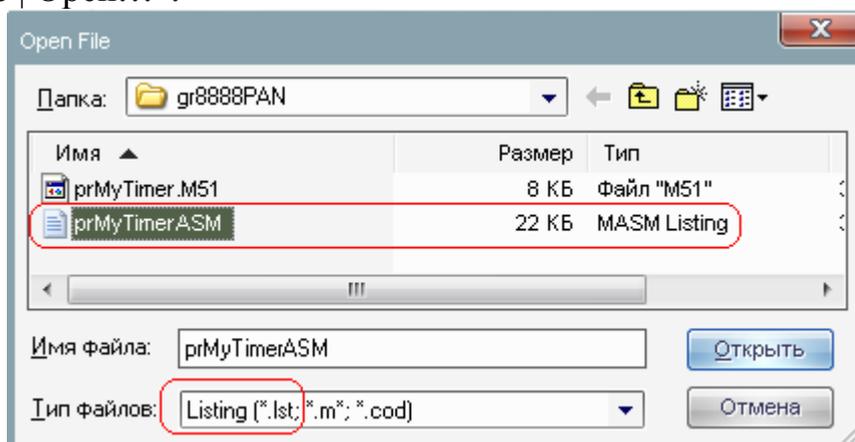
После трансляции курсор, к сожалению, не укажет на строку с ошибкой в исходном тексте макроса, а только в месте его вызова (как на рисунке ниже). Если макросов много и они вложены друг в друга, поиск ошибки будет сильно затруднен.

```

07 movx @DPTR, a,== и выводим его в ВУ/ЗУ по адресу в DPTR
08 mov DPPP, #0,== вернуться к нулевой странице памяти
09 ENDM,== конец макроса
10 ;=====
11 CSEG,== начало сегмента кодов программы (диапазон 0000h..F
12 ;== т.к. 0000..1FFF - Flash ПЗУ, то доступны адреса на
13 ;== верхний предел обусловлен емкостью 64KB внешнего О
14 ;=====
15 ORG 4000h,== с этого адреса будем располагать код основной
16 ;== резидентный загрузчик САМ переходит по этому адресу
17 main_p: ;== точка входа в основную программу
18 Write Wr_byte, 3eh, 53h,== вызов макроса с тремя рассчитанными

```

К нашему “большому счастью” есть выход. Откройте файл листинга из п. меню “File | Open...”.



Найдите строку с ошибкой. Поиск по служебной строке “ERROR”.

```

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 запустите программу, не забыв перед ее загрузкой нажать на кнопку “RESET”.

Теперь добавим команды, которые позволят реагировать на нажатие **ТОЛЬКО ОДНОЙ** клавиши, а именно – клавиши ‘1’. Часть новых команд разместим внутри “бесконечного” цикла _8: jmp _8.

```

main_prog: ;== точка входа в основную программу
mov DPP,#DPP_page_num
mov Mask,#00010000b;== маска = 0001 0000 - выделить ROW0
_8:
mov Wr_COL,#11111110b;== 1111 1110 - подать COLO=0 (бит D0)
Write Wr_COL,DPTR_KEYreg_num;== вывести '0' в нулевой столбец
Read Rd_ROW,DPTR_KEYreg_num;== считать код строк
mov a,Rd_ROW;== переслать егѣ в аккумулятор
anl a,Mask;== и выделить маской нулевую строку ROW0 (бит D4)
cjne a,#0,_8;== если D4<>0, то кл.'1' не нажата и повторить цикл ожидания
Write #31h,DPTR_LEDreg_num;== если нажата вывести ASCII код цифры '1' = 31h.
jmp _8;== бесконечный цикл
;=====
END
; ROW0--<---|кл.1|
; битD4      |
;           ^
;           |
;           |
;           |
;           COLO
;           битD0

```

Таблица шестнадцатеричных ASCII кодов приведена ниже, например код буквы ‘A’=41h.

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1.	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2.		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4.	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5.	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6.	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7.	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

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).

#####

ВНИМАНИЕ: В вашем варианте 3 первых операнда (53h, 3eh и 4000h), выделенных жирным шрифтом, необходимо пересчитать или заменить в соответствии с заданием.

#####

```
#include <aduc812.h>
Rd_ROW EQU R2;== читаемый код строки хранится в регистре R2
Wr_COL EQU R3;== записываемый код столбца хранится в регистре R3
Mask EQU R4;== маска для выделения линии строки хранится в
регистре R4
DPP_page_num EQU 53h;== код номера страницы внешних устройств
DPTR_KEYreg_num EQU 3eh;== номер (адрес) регистра клавиатуры
DPTR_LEDreg_num EQU 3fh;== номер (адрес) регистра светодиодов
(лаб. раб. №30)
Tochka_Vhoda EQU 4000h;== адрес входа в основную программу
;=====
Write MACRO Wr_byte, reg_addr;== макроопр-е с двумя формал. парам-и
mov DPTR,#reg_addr;== адрес регистра ВУ или ЯП внешн. ОЗУ
внутри страницы
mov a,Wr_byte;== байт данных Wr_byte пересылаем в аккумулятор
movx @DPTR,a;== и выводим его в ВУ/ОЗУ по адресу в DPTR
ENDM;== конец макроса
;=====
Read MACRO Rd_byte, reg_addr;== макроопр-е с двумя формал. парам-и
mov DPTR,#reg_addr;== адрес регистра ВУ или ЯП внешн. ОЗУ
внутри страницы
movx a,@DPTR;== вводим байт из ВУ/ОЗУ по адресу в DPTR
mov Rd_byte,a;== байт данных из аккумулятора пересылаем в
переменную Rd_byte
ENDM;== конец макроса
;=====
CSEG at 0
jmp Tochka_Vhoda
;=====
ORG Tochka_Vhoda;== с этого адреса располагается код программы
main_prog: ;== точка входа в основную программу
mov DPP,#DPP_page_num
mov Mask,#00010000b;== маска = 0001 0000 - выделить ROW0
```

```

_8: ;== метка “бесконечного цикла”
    mov Wr_COL,#1111110b;== 1111 1110 - подать COL0=0 (бит D0)
    Write Wr_COL,DPTR_KEYreg_num;== вывести '0' в нулевой столбец
    Read Rd_ROW,DPTR_KEYreg_num;== считать код строк
    mov a,Rd_ROW;== переслать его в аккумулятор
    anl a,Mask;== и выделить маской нулевую строку ROW0 (бит D4)
    cjne a,#0,_8;== если D4<>0, то кл.'1' не нажата и повторить цикл
ожидания
    Write #31h,DPTR_LEDreg_num;== если нажата, вывести ASCII код
цифры '1' = 31h.
    jmp _8;== бесконечный цикл
;=====
END

```

III) Пояснения к программе. Вначале приведены объявления нескольких переменных, которые расположены в рабочих регистрах R2..R4 микроконтроллера и четыре адреса. Далее приведены два макроопределения записи/вывода и чтения/ввода байта в/из внешней ячейки памяти или ВУ в диапазоне текущей страницы (0000...FFFF) адресного пространства (000000 ... FFFFFFFF = 16МБ). Команды “movx @DPTR, a” и “movx a, @DPTR” в макросах являются ключевыми и пересылают байт из аккумулятора во внешнее устройство (внешнюю ячейку памяти) или наоборот. Подробно этот процесс был описан в лекции “[МК система с шинным интерфейсом \(или с тремя шинами\)](#)”.

Директива CSEG(Code Segment) задает адрес (начало), с которого будут располагаться коды программы (по умолчанию с нулевого адреса). Используется также CSEG at XXXX – если необходимо уточнить начальный адрес XXXX сегмента. Команда “jmp Toчка_Vhoda” осуществляет переход к адресу, с которого располагается в ОЗУ, разрабатываемая программа и должна располагаться по адресу 0000h во внутренней флэш-памяти МК. При включении или рестарте МК начинает свою работу именно с этого адреса 0000h.

Следующая директива ORG “Toчка_Vhoda” переопределяет начальный адрес следующего за ней кода программы. В примере, в качестве варианта - с адреса 4000h. Метка main_p (имя произвольное) задает адрес основной программы.

Завершает программу бесконечный цикл “_8:jmp _8” и директива конца программы “END”. В отличие от программы в лабораторной работе №30, в цикле размещены команды. Первые две (mov Wr_COL,#1111110b и Write Wr_COL,DPTR_KEYreg_num) периодически выводят логич.0 в линию ‘col0’ (рис.30.5). Следующие две (Read Rd_ROW,DPTR_KEYreg_num и mov a,Rd_ROW) считывают код клавиатуры и помещают его в аккумулятор.

ПРИМЕЧАНИЕ: На принципиальной схеме адрес регистра столбцов клавиатуры и адрес входов мультиплексора строк клавиатуры совпадают, поэтому в макросах Write и Read используется один адрес DPTR_KEYreg_num, который вы уже вычислили.

Команда “anl a,Mask – (ANd Logic)” выделяет бит D4 (строка row0) обнуляя остальные биты, а команда “cjne a,#0,_8” во-первых сравнивает результат в аккумуляторе с нулем и во-вторых осуществляет переход к метке “_8”, если результат НЕ РАВЕН НУЛЮ (jne), т.е. если клавиша ‘1’ НЕ НАЖАТА. Если клавиша ‘1’ нажата бит D4=0, содержимое аккумулятора равно нулю, перехода нет и выполняется следующая команда “Write #31h,DPTR_LEDreg_num”, которая выводит на линейку светодиодов ASCII код цифры ‘1’. Последняя команда “jmp _8” закидывает перечисленные действия.

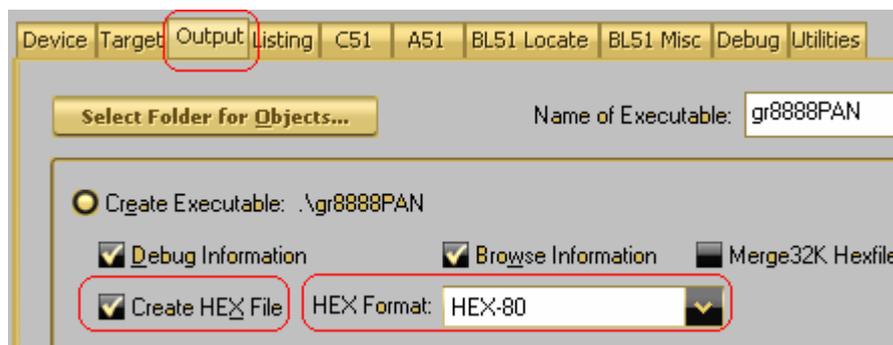
Без этого цикла микроконтроллер, закончив выполнение программы, будет бесконтрольно выбирать неинициализированные байты из свободной области памяти и выполнять их. Это приведет либо к “зависанию”, либо к перезагрузке системы, если МК снова доберется до нулевого адреса. В худшем случае, если последовательность таких байтов образует вредоносный код, то УМК может выйти из строя. Для решения этой проблемы можно, либо оформить программу в виде бесконечного цикла (как в примере), либо записать в конце программы бесконечный цикл.

Этот фрагмент транслируем в машинный код и убедимся, что на этом коротком отрезке мы не наделали синтаксических ошибок. Для этого нажмем на кнопку  на верхней панели инструментов “uVision2”.

IV) Пробный запуск программы.

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

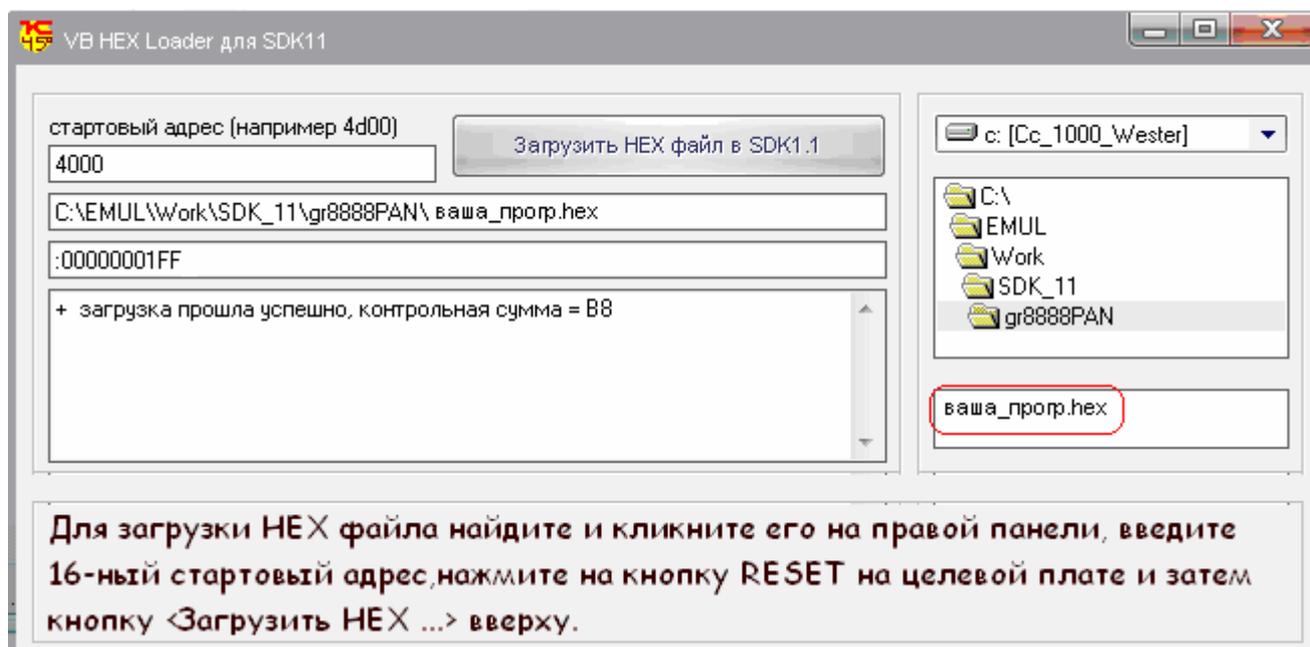
Вызовите окно мастера шаблонов кнопкой  (или п. меню “Project | Options for Target...”) и отметьте на странице “Output” формат файла “HEX-80”.



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...”. Нажмите несколько раз на любые клавиши, кроме ‘1’ – светодиоды д.б. **выключены**. Если теперь нажать на Кл.’1’ на линейке светодиодов должен высветиться ASCII код единицы:



что и требовалось на данном этапе. **В противном случае необходимо пересчитать указанные 6 операндов программы.** Закрывать окно загрузчика “VB HEX Loader ...” не нужно, т.к. он еще понадобится.

Результат покажите преподавателю

#####

ВНИМАНИЕ: Теперь пересчитайте 3 последних операнда (#00010000b, #1111110b и 31h), выделенных жирным шрифтом, для клавиши, приведенной в задании.

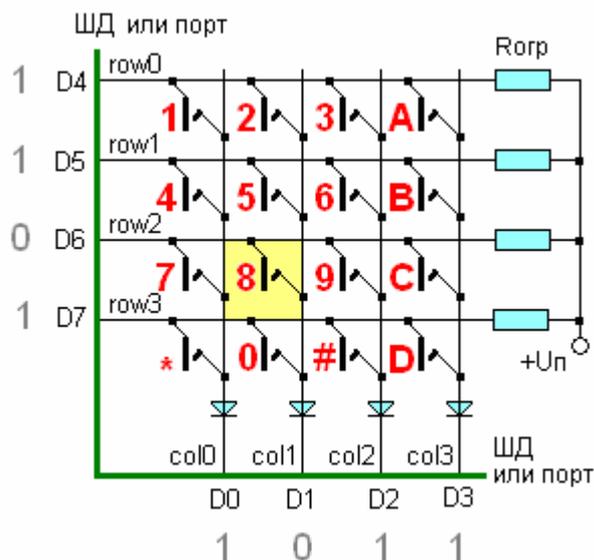
#####

Оттранслируйте программу и убедитесь, что “ваша” клавиша функционирует.

Результат снова покажите преподавателю

V). Разработка программы фиксирующей момент нажатия на любую клавишу.

Типовой алгоритм заключается в следующем. Клавиши (кнопки, контакты) располагаются на пересечении строк (row) и столбцов (column) условной двумерной матрицы. Линии столбцов и строк подключены к контактам, как показано на рисунке и не соединены друг с другом, если НЕ НАЖАТА ни одна из клавиш. При замыкании контактов соответствующая строка и столбец соединяются.



Подавая ПООЧЕРЕДНО на одну из групп линий, например, на столбцы тестовый сигнал, можно его зафиксировать на той или иной строке, если соответствующая пара контактов замкнута. Такой процесс периодической подачи пробного сигнала и его считывания называется сканированием.

Обычно к одной группе линий (в схеме это строки) через ограничительные резисторы подается высокий уровень сигнала (обычно +Up) – логическая “1”. На каждую линию другой группы (в схеме это столбцы) периодически подается низкий уровень – логический “0”. Периодическое смещение нуля от разряда к разряду в определенных кругах называется “бегущим нулем”.

Пока не замкнута НИ ОДНА из пар контактов, на линиях строк будет высокий потенциал – “1”. В нашей схеме ПРИЗНАКОМ “не нажатия” на клавишу будут ВСЕ единицы на линиях row0...row3 = 1111. При замыкании пары контактов “бегущий 0” рано или поздно (микросекунды) появится на соответствующей строке, что и явится признаком нажатия на клавишу.

Например, при активировании клавиши “8”, ноль по линии col1 (D3D2D1D0=1101) попадет на линию row2 (D7D6D5D4=1011). Комбинация D7D6D5D4 D3D2D1D0=10111101 – называется **скан-кодом** и является уникальной для каждой клавиши. Диоды в схеме предназначены для предотвращения попадания нулей на выходы элементов, подключенных к другим столбцам, если случайно нажаты несколько клавиш.

С учетом изложенного, на данном этапе в программе необходимо решить две задачи: 1) организовать сканирование клавиатуры и 2) сформировать скан-код нажатой клавиши.

Вернемся к программе. Удалите часть ненужного теперь кода. Программа должна иметь следующий вид. Естественно, три константы **53**, **3e** и **4000** у вас могут иметь другие значения.

```
#include <aduc812.h>
DPP_page_num EQU 53h ;== код номера страницы внешних устройств
DPTR_KEYreg_num EQU 3eh ;== номер (адрес) регистра клавиатуры
DPTR_LEDreg_num EQU 3fh ;== номер (адрес) регистра светодиодов (лаб. раб. №30)
Tochka_Vhoda EQU 4000h ;== адрес входа в основную программу
;=====
Write MACRO Wr_byte, reg_addr ;== макроопр-е с двумя формал. парам-и
    mov DPTR, #reg_addr ;== адрес регистра ВУ или ЯП внешн. ОЗУ внутри страницы
    mov a, Wr_byte ;== байт данных Wr_byte пересылаем в аккумулятор
    movx @DPTR, a ;== и выводим его в ВУ/ОЗУ по адресу в DPTR
    ENDM ;== конец макроса
;=====
Read MACRO Rd_byte, reg_addr ;== макроопр-е с двумя формал. парам-и
    mov DPTR, #reg_addr ;== адрес регистра ВУ или ЯП внешн. ОЗУ внутри страницы
    movx a, @DPTR ;== вводим байт из ВУ/ОЗУ по адресу в DPTR
    mov Rd_byte, a ;== байт данных из аккумулятора пересылаем в переменную Rd_byte
    ENDM ;== конец макроса
;=====
    CSEG at 0
    jmp Tochka_Vhoda
;=====

    ORG Tochka_Vhoda ;== с этого адреса располагается код программы
main_prog: ;== точка входа в основную программу
    mov DPP, #DPP page num ;== загрузить адрес страницы, на кот. находятся ВУ
_8:
    jmp _8 ;== бесконечный цикл
;=====
END ;
```

V-1) Модуль сканирования

Теперь добавьте указанные фрагменты, которые реализуют описанный алгоритм сканирования.

```

Точка Входа    EQU 4000h
cur_COL      EQU R5;== позиция "бегущего 0" (текущий столбец) в регистре R5
с_С         EQU 5;== синоним (alias) этого регистра по адресу 5
cur_ROW     EQU R6;== код строк в регистре R6
с_Р         EQU 6;== синоним (alias) этого регистра по адресу 6
fst_COL     EQU 1111110b;== нач знач "бегущего нуля" бит D0
end_COL     EQU 1111011b;== кон знач "бегущего нуля" бит D3
;=====
DSEG at 8;== начало сегмента данных (после банка регистров R0..R7)
scan_code: DS 1;== скан-код клавиши в ячейке с именем scan_code
key_num:    DS 1;== номер клавиши в ячейке с именем key_num
ascii_code: DS 1;== ASCII код в ячейке с именем ascii_code
dno_steka:  DS 1;== dno_steka просто адрес дна стека (ПОСЛЕ ВСЕХ переменных)
;=====
BSEG;== сегмент однобитовых переменных
key_pressed: DBIT 1;== бит-признак нажатия на клавишу
;=====
Write MACRO Wr_byte, reg_addr;== макроопр-е с двумя формал. парам-и

```

Из комментариев и так понятно для чего вводятся указанные переменные и константы. Кроме, наверное, синонимов. Ассемблер для микроконтроллеров семейства MCS-51 не так богат на число команд и способы адресации, как ассемблер семейства x86(x64). Например, если код строк матрицы клавиатуры хранится в регистре R6 (ячейка памяти с адресом 6), то логическая команда “ANL – И” ДОЛЖНА обращаться к этому операнду, как к содержимому ячейки памяти с адресом 6 (с_Р). А вот команда “CJNE – сравнить и перейти, если рез-т не равен нулю” ДОЛЖНА обращаться к этому же операнду, как к регистру (cur_ROW), что наглядно видно из следующего примера, который встретится далее в программе.

```

    anl c_R,#11110000b;== выделить 4 строки (биты D7..D4)
    cjne cur_ROW,#11110000b,LED_Display;//== высветить код
клавиши

```

ПРИМЕЧАНИЕ: По возможности нужно стремиться хранить операнды в регистрах, так как такие команды не только короче, но и быстрее выполняются. Например, команда **ANL A, R5** (регистравая адресация) – занимает 1 байт памяти, а **ANL A, 5** (прямая адресация) – 2 байта, так как в первом случае трехбитный код регистра упакован в байте КОП (кода операции 00101101), а во втором случае к байту КОП добавляется байт адреса этого регистра (00100101 00000101).

Кстати, неиссякаемым источником ошибок является невнимательное отношение к использованию символа непосредственной

адресации “#” перед операндом. Например, в команде **MOV A,5** байт из ячейки памяти с адресом 5 пересылается в аккумулятор (прямая адресация), а в команде **MOV A,#5** в аккумулятор заносится число 5 (непосредственная адресация).

Вернемся к программе и запишем подпрограмму, в которой производится одноразовое сканирование клавиатуры в соответствии с изложенным алгоритмом.

```

_8:
    jmp _8;== бесконечный цикл
;=====
Scan_Key_Once:
    mov cur_COL,#fst_COL;== начало однократного цикла сканирования
pv: Write cur_COL,DPTR_KEYreg_num;== записать "0" в текущий столбец
    Read cur_ROW,DPTR_KEYreg_num;== прочитать код строк
;== if cur_ROW<>1111b then (клавиша нажата)
        setb key_pressed;== установить бит "клавиша-нажата"
        anl c_R,#11110000b;== выделить 4 строки (биты D7..D4)
        cjne cur_ROW,#11110000b,Scan_2_ASCII;== высветить код клавиши
;== else (клавиша НЕ нажата)
;==     if cur_COL<>end_COL then (не все столбцы)
            cjne cur_COL,#end_COL,nxt_COL;== переход к след. столбцу
;==     else (все столбцы просканированы - выйти из подпрограммы)
            clr key_pressed;== сбросить бит "клавиша-нажата"
;==     end if
;== end if
ext:
    ret;== выйти из подпрограммы
;=====
END

```

Команда `mov cur_COL,#fst_COL` записывает “бегущий 0” в младший разряд переменной `cur_COL` (`fst_COL` это `first column`, т.е. крайний левый столбец, `cur_COL` это `current column` – текущий столбец). Следующая команда (вызов макроса `Write`) выводит его в регистр клавиатуры (см. рис.30.3), выходы которого подключены к 4-м столбцам матрицы крайний левый столбец `col0` (бит `D0`). Вызов макроса `Read` считывает текущий код строк матрицы в регистр `R6` (`cur_ROW`). Дальнейшая часть (6 команд) структурирована для большего понимания с помощью комментариев “if-else...”.

Итак, после чтения кода строк имеется 2 варианта: либо 1) в 4-х старших битах присутствует 0, значит нажата одна из клавиш, либо 2) в 4-х старших битах нет нуля, т.е. ни одна из пар контактов не замкнута.

В первом случае сначала устанавливаем признак “клавиша нажата”. Далее с помощью маски `11110000` и команды “И” выделяем 4 ст. бита и затем, если они не равны “1111” (клавиша нажата), по команде сравнения и условного перехода (`Compare and Jump if Not Equal` – `CJNE`) переходим к модулю (метка `Scan_2_ASCII`) преобразования скан-кода нажатой клавиши в `ASCII` код.

Во втором случае (клавиша не нажата) может быть два варианта: 2-1) если “0” подан не на все столбцы (текущее сканирование не завершено) с помощью команды условного перехода “сjne cur_COL,#end_COL,nxt_COL” переходим к модулю (метка nxt_COL – следующий столбец), в котором осуществляется сдвиг “бегущего нуля” в следующую позицию и 2-2) если номер текущего столбца (в регистре cur_COL) совпадает с номером последнего (#end_COL) команда сjne не выполняется, следующая за ней команда “clr key_pressed” сбрасывает признак “клавиша нажата” и происходит выход из подпрограммы по команде “ret”.

Если сейчас попытаться оттранслировать программу, возникнут две ошибки, по причине отсутствия двух указанных переходов: Scan_2_ASCII и nxt_COL. Поэтому двинемся дальше.

V-2) Подпрограмма формирования скан-кода клавиши.

Новую подпрограмму “make_scan_code” запишите в указанное место программы. Логическими командами “И” выделяем 4 бита столбцов и 4 бита строк и объединяем их командой “ИЛИ” в одном байте (ячейка памяти scan_code). Можно было бы хранить scan_code в одном из оставшихся регистров Ri, но так как их число значительно меньше числа ячеек внутренней памяти данных (ВПД), и они понадобятся в следующих лаб. работах остановимся на ячейке памяти

```

;== end if
ext:
    ret ;== выйти из подпрограммы
;=====
make_scan_code:
    mov a,cur_COL ;== записать в аккумулятор код столбцов
    and a,#0fh ;== маской 00001111 выделить мл. тетраду столбцов
    mov scan_code,a ;== и записать ее в скан-код
    mov a,cur_ROW ;== записать в аккумулятор код строк
    and a,#0f0h ;== маской 11110000 выделить ст. тетраду строк
    orl scan_code,a ;== и объединить мл. и ст. тетрады в скан-код
    ret
;=====
END

```

V-3) Модуль сдвига “бегущего нуля” и корректировка основного модуля “main_prog”.

Операнд команд сдвига располагается только в аккумуляторе, поэтому и используется двойная перегрузка из регистра cur_COL в аккумулятор и обратно. После сдвига переходим к метке pv (povtor) для вывода нуля в следующую колонку. Команда RL производит циклический

сдвиг влево (rotate left) “бегущего нуля”, т.е. от младшего разряда к старшему. Заодно добавлена программная заглушка для модуля “scan_2_ASCII”. Сам модуль на этом этапе не нужен, но транслятор будет “ругаться” при его отсутствии.

```

    orl scan_code,a;== и объединить мл. и ст. тетрады в скан-код
    ret
;=====
nxt_COL:;== сдвиг "бегущего нуля"
    mov a,cur_COL
    rl a;== сдвинуть "бегущий 0" (сдвиг только в аккумуляторе)
    mov cur_COL,a
    jmp pv;== перейти к выводу "0" в следующий столбец
;=====
scan_2_ASCII:
    jmp ext
;=====
END

```

Добавьте в программу приведенный фрагмент.

Теперь допишем в основную программу команды недостающие начальной инициализации и команды, высвечивающие на светодиодах “скан-код” нажатой клавиши.

```

main_prog:;== точка входа в основную программу
    mov DPP,#DPP_page_num;== загрузить адрес страницы, на кот. находятся ВУ
    mov sp,#dno_steka;== загрузить в регистр sp адрес дна стека
    clr key_pressed;== обнулить бит "клавиша нажата"
;=====
_8:
wp: call Scan_Key_Once
    jnb key_pressed, wp;== ждем нажатия клавиши (wp- wait for pressed)
    call make_scan_code
wu: call Scan_Key_Once
    jnb key_pressed, wu;== ждем отпущения клавиши (wu- wait for unpressed)
    Write scan_code,DPTR_LEDreg_num;== высветить скан-код нажатой клавиши
    jmp _8;== бесконечный цикл
;=====
Scan_Key_Once:

```

Первый циклический вызов подпрограммы Scan_Key_Once проверяет, нажата ли клавиша, и если не нажата, то бит “key_pressed” равен нулю. Команда условного перехода “Jump if Not Bit” проверяет этот бит и если он не равен “1”, и повторяет цикл ввода с метки “wp”. При нажатии на клавишу признак “key_pressed” устанавливается в “1”, условный переход не выполняется. Производится вызов подпрограммы, формирующей скан-код клавиши “make_scan_code”.

Начинается следующий цикл ожидания, но теперь уже момента отпущения клавиши. Циклический вызов Scan_Key_Once будет продолжаться до тех пор, пока клавиша не будет отпущена. В этот момент бит “key_pressed” сбрасывается в “0” и команда условного перехода “Jump if Bit” выполняться не будет. В последней строчке макрос Write выводит скан-код клавиши на светодиоды.

Окончательно на данном этапе программа должна иметь следующий вид:

```

#include <aduc812.h>
DPP_page_num EQU 53h;== код номера страницы внешних
устройств
DPTR_KEYreg_num EQU 3eh;== номер (адрес) регистра
клавиатуры
DPTR_LEDreg_num EQU 3fh;== номер (адрес) регистра
светодиодов (лаб. раб. №30)
Tochka_Vhoda EQU 4000h;== адрес входа в основную программу
cur_COL EQU R5;== позиция "беущего 0" (текущий
столбец) в регистре R5
с_С EQU 5;== синоним (alias) этого регистра по адресу 5
cur_ROW EQU R6;== код строк в регистре R6
с_Р EQU 6;== синоним (alias) этого регистра по адресу 6
fst_COL EQU 1111110b;== нач знач "бегущего нуля" бит D0
end_COL EQU 11110111b;== кон знач "бегущего нуля" бит D3
;=====
DSEG at 8;== начало сегмента данных (после банка регистров
R0..R7)
scan_code: DS 1;== скан-код клавиши в ячейке с именем scan_code
key_num: DS 1;== номер клавиши в ячейке с именем key_num
ascii_code: DS 1;== ASCII код в ячейке с именем ascii_code
dno_steka: DS 1;== dno_steka просто адрес дна стека (ПОСЛЕ
ВСЕХ переменных)
;=====
BSEG;== сегмент однобитовых переменных
key_pressed: DBIT 1;== бит-признак нажатия на клавишу
;=====
Write MACRO Wr_byte, reg_addr;== макроопр-е с двумя формал.
парам-и
mov DPTR,#reg_addr;== адрес регистра ВУ или ЯП внешн.
ОЗУ внутри страницы
mov a,Wr_byte;== байт данных Wr_byte пересылаем в
аккумулятор
movx @DPTR,a;== и выводим его в ВУ/ОЗУ по адресу в DPTR
ENDM;== конец макроса
;=====
Read MACRO Rd_byte, reg_addr;== макроопр-е с двумя формал.
парам-и
mov DPTR,#reg_addr;== адрес регистра ВУ или ЯП внешн.
ОЗУ внутри страницы
movx a,@DPTR;== вводим байт из ВУ/ОЗУ по адресу в DPTR

```

```

        mov Rd_byte,a;== байт данных из аккумулятора пересылаем в
переменную Rd_byte
        ENDM;== конец макроса
;=====
        CSEG at 0
        jmp Tochka_Vhoda;== резидентный загрузчик САМ переходит
по этому адресу
        ORG Tochka_Vhoda;== с этого адреса располагается код
программы
        main_prog: ;== точка входа в основную программу
        mov DPP,#DPP_page_num;== загрузить адрес страницы, на
кот. находятся ВУ
        mov sp,#dno_steka;== загрузить в регистр sp адрес дна стека
        clr key_pressed;== обнулить бит "клавиша нажата"
        _8:
        wr:  call Scan_Key_Once
            jnb key_pressed, wr;== ждем нажатия клавиши (wr- wait for
pressed)
            call make_scan_code
        wu:  call Scan_Key_Once
            jb key_pressed, wu;== ждем отпущения клавиши (wu- wait for
unpressed)
            Write scan_code,DPTR_LEDreg_num;== высветить скан-код
нажатой клавиши
            jmp _8;== бесконечный цикл
;=====
        Scan_Key_Once:
            mov cur_COL,#fst_COL;== начало однократного цикла
сканирования
            rv:  Write cur_COL,DPTR_KEYreg_num;== записать "0" в текущий
столбец
                Read cur_ROW,DPTR_KEYreg_num;== прочитать код строк
;== if cur_ROW<>1111b then (клавиша нажата)
                    setb key_pressed;== установить бит "клавиша-нажата"
                    anl c_R,#11110000b;== выделить 4 строки (биты D7..D4)
                    cjne cur_ROW,#11110000b,scan_2_ASCII;//== высветить
код клавиши
;== else (клавиша НЕ нажата)
;==   if cur_COL<>end_COL then (не все столбцы)
                        cjne cur_COL,#end_COL,nxt_COL;== переход к
след. столбцу
;==   else (все столбцы просканированы - выйти из подпрограамы)
                        clr key_pressed;== сбросить бит "клавиша-нажата"
;==   end if

```

```

;== end if
ext:
    ret;== выйти из подпрограммы
;=====
make_scan_code:
    mov a,cur_COL;== записать в аккумулятор код столбцов
    andl a,#0fh;== маской 00001111 выделить мл. тетраду столбцов
    mov scan_code,a;== и записать ее в скан-код
    mov a,cur_ROW;== записать в аккумулятор код строк
    andl a,#0f0h;== маской 11110000 выделить ст. тетраду строк
    orl scan_code,a;== и объединить мл. и ст. тетрады в скан-код
    ret
;=====
nxt_COL::;== сдвиг "бегущего нуля"
    mov a,cur_COL
    rll a;== сдвинуть "бегущий 0" (сдвиг только в аккумуляторе)
    mov cur_COL,a

    jmp rv;== перейти к выводу "0" в следующий столбец
;=====
scan_2_ASCII:
    jmp ext
;=====
END

```

Снова оттранслируйте  и загрузите  программу. Далее нажимая на клавиши, убедитесь, что высвечиваемый скан-код соответствует этим клавишам. Например, как уже говорилось, при нажатии на клавишу “8” высветится скан-код D7D6D5D4D3D2D1D0=10111101. Скан-код “1” - D7D6D5D4D3D2D1D0 = 11101110, и т.д.

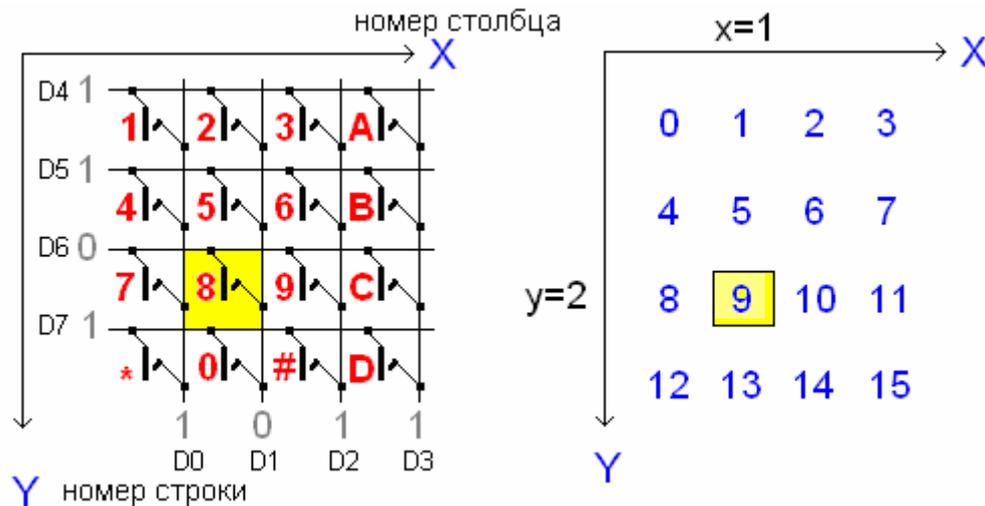


Результат покажите преподавателю

V-4) Подпрограмма преобразования скан-кода клавиши в ее порядковый номер.

Как мы только что убедились, каждой клавише соответствует единственный скан-код. Используя его уже можно “повесить” на каждую клавишу определенное действие. Однако оперировать скан-кодами неудобно по одной простой причине: десятичный эквивалент скан-кода никак не соответствует ни порядковому номеру клавиши, ни ее ASCII коду (если на клавишах приведены буквенно-цифровые обозначения).

Поэтому следующим этапом явится добавление в программу подпрограммы преобразования скан-кода клавиши в ее порядковый номер. Пронумеровать клавиши можно по-разному, например, слева направо и сверху вниз. Справа на рисунке - обозначения клавиш, слева – их порядковые номера.



Из школьной арифметики известно, что номер элемента матрицы, клавиши (`key_num`) определяется по формуле: $key_num = n * y + x$, где x – номер столбца, y – номер строки, n – число столбцов в строке (в нашем случае $n=4$). Нумерация строк и столбцов естественно начинается с нуля.

Из приведенного примера с нажатой клавишей “8” видно, что позиции строки и столбца определяют два нуля. Координата столбца $x=1$, а координата строки $y=2$. Тогда по формуле порядковый номер клавиши с маркировкой “8” равен $key_num = 4 * 2 + 1 = 9$.

Один из простейших алгоритмов нахождения координат строк и столбцов заключается в сдвиге нуля в сторону младшего разряда, до тех пор, когда “0” попадет во флаг переноса и подсчета числа сдвигов ДО этого момента.

Для вычисления номера клавиши при каждом сдвиге скан-кода столбцов к текущему значению номера прибавляется 1, а при сдвиге скан-кода строк к текущему номеру прибавляется $n = 4$. Так как 4 бита скан-кода строк занимают 4 старших разряда, перед сдвигом их помещают в младшие 4 разряда командой ассемблера “swap”.

В соответствии с рассмотренным алгоритмом, внесите в программу следующие изменения:

```

end_COL      EQU 11110111b,== кон знач "бегущего нуля" бит D3
SCAN        EQU 1,== константа для условной компиляции
KEYN        EQU 2,== константа для условной компиляции
ASCII       EQU 3,== константа для условной компиляции
;
=====
DSEG at 8,== начало сегмента данных (после банка регистров R0..R7)
.....

```

```

wu: call Scan_Key_Once
    jb key_pressed, wu,== ждем отпущения клавиши (wu- wait for unpressed)
DSPL EQU ASCII,== DSPL = KEYN либо SCAN либо ASCII
IF (DSPL = SCAN),== начало блока условной компиляции
    Write scan_code,DPTR_LEDreg_num,== либо вывести скан-код нажатой клавиши
ELSEIF (DSPL = KEYN)
    Write key_num,DPTR_LEDreg_num,== либо высветить номер нажатой клавиши
ELSEIF (DSPL = ASCII),== либо высветить ее ASCII код
ENDIF,== конец блока условной компиляции
    jmp _8,== бесконечный цикл
;=====
Scan_Key_Once:

```

Для удобства отладки или для получения нескольких вариантов окончательного вида готовой программы используется условная трансляция с помощью ключевых слов IF... ELSEIF....ENDIF и вспомогательной пользовательской переменной DSPL, которая в программе может принимать три значения SCAN, KEYN и ASCII. Соответственно будет транслироваться только один из трех условных блоков, и на светодиоды будет выводиться либо скан-код, либо порядковый номер, либо ASCII код клавиши.

```

.....
scan_2_ASCII:

```

```

    call scan_code_2_key_num
    jmp ext
;=====

```

```

scan_code_2_key_num:

```

```

;== подпрограмма преобразования скан-кода клавиши в ее порядковый номер
mov key_num,#0,== нач. значение = 0
mov a,scan_code,== в мл. тетраде скан-код столбцов
nc: rrc a,== сдвиг вправо кода столбцов в сторону младшего разряда
    jnc r,== выйти из подсчета числа столбцов, если "0" попал во флаг переноса
    inc key_num,== иначе увеличить порядковый номер на 1
    jmp nc,== повторить цикл сдвига
r: mov a,scan_code,== в ст. тетраде скан-код строк
    swap a,== меняем местами тетрады (скан-код строк теперь в младшей)
nr: rrc a,== сдвиг вправо в сторону младшего разряда
    jnc ex,== выйти из подсчета числа строк, если "0" попал во флаг переноса
    inc key_num,== иначе увеличить порядковый номер на 4
    inc key_num
    inc key_num
    inc key_num
//REPT 4 ;== или так с помощью встроенного макроса
// inc key_num
//ENDM
    jmp nr,== повторить цикл сдвига
ex: ret,== возврат из подпрограммы
;=====

```

```

END

```

Подпрограмма “scan_code_2_key_num”, как следует из ее названия, преобразует скан-код клавиши в ее порядковый номер: “1” – 0, “2” – 1, ...”#” – 14 и “D” – 15.

ДЛЯ ЛЮБОЗНАТЕЛЬНЫХ: Как уже отмечалось, ассемблер asm-51 имеет меньший набор команд и способов адресации, чем ассемблер

x86(x64). Например, нет команды “ADD регистр, #число”, а есть только “ADD A, #число”.

Поэтому вместо одной несуществующей команды “ADD key_num, #4” пришлось бы городить следующее (5 команд):

push асс;== опять синоним аккумулятора (аккумулятор нужно сохранить, т.к. он используется

;== для сдвига скан-кода)

mov a, key_num

add a,#4

mov key_num,a

pop асс;== восстанавливаем содержимое аккумулятора

Выбор – 4 команды “inc” или встроенный макрос уже дело вкуса. В настоящий момент программа должна иметь следующий вид:

```
#include <aduc812.h>
```

```
DPP_page_num EQU 53h;== код номера страницы внешних устройств
```

```
DPTR_KEYreg_num EQU 3eh;== номер (адрес) регистра клавиатуры
```

```
DPTR_LEDreg_num EQU 3fh;== номер (адрес) регистра светодиодов  
(лаб. раб. №30)
```

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

```
cur_COL EQU R5;== позиция "беущего 0" (текущий столбец) в  
регистре R5
```

```
c_C EQU 5;== синоним (alias) этого регистра по адресу 5
```

```
cur_ROW EQU R6;== код строк в регистре R6
```

```
c_R EQU 6;== синоним (alias) этого регистра по адресу 6
```

```
fst_COL EQU 1111110b;== нач знач "бегущего нуля" бит D0
```

```
end_COL EQU 1111011b;== кон знач "бегущего нуля" бит D3
```

```
SCAN EQU 1;== константа для условной компиляции
```

```
KEYN EQU 2;== константа для условной компиляции
```

```
ASCII EQU 3;== константа для условной компиляции
```

```
=====
```

```
DSEG at 8;== начало сегмента данных (после банка регистров  
R0..R7)
```

```
scan_code: DS 1;== скан-код клавиши в ячейке с именем scan_code
```

```
key_num: DS 1;== номер клавиши в ячейке с именем key_num
```

```
ascii_code: DS 1;== ASCII код в ячейке с именем ascii_code
```

```
dno_steka: DS 1;== dno_steka просто адрес дна стека (ПОСЛЕ ВСЕХ  
переменных)
```

```
=====
```

```
BSEG;== сегмент однобитовых переменных
```

```
key_pressed: DBIT 1;== бит-признак нажатия на клавишу
```

```
=====
```

```
Write MACRO Wr_byte, reg_addr;== макроопр-е с двумя формал. парам-и
```

```

mov DPTR,#reg_addr;== адрес регистра ВУ или ЯП внешн. ОЗУ
внутри страницы
mov a,Wr_byte;== байт данных Wr_byte пересылаем в аккумулятор
movx @DPTR,a;== и выводим его в ВУ/ОЗУ по адресу в DPTR
ENDM;== конец макроса
;=====
; Read MACRO Rd_byte, reg_addr;== макроопр-е с двумя формал. парам-и
mov DPTR,#reg_addr;== адрес регистра ВУ или ЯП внешн. ОЗУ
внутри страницы
movx a,@DPTR;== вводим байт из ВУ/ОЗУ по адресу в DPTR
mov Rd_byte,a;== байт данных из аккумулятора пересылаем в
переменную Rd_byte
ENDM;== конец макроса
;=====
; CSEG at 0
jmp Tochka_Vhoda;== резидентный загрузчик САМ переходит по
этому адресу
ORG Tochka_Vhoda;== с этого адреса располагается код программы
main_prog: ;== точка входа в основную программу
mov DPP,#DPP_page_num;== загрузить адрес страницы, на кот.
находятся ВУ
mov sp,#dno_steka;== загрузить в регистр sp адрес дна стека
clr key_pressed;== обнулить бит "клавиша нажата"
_8:
wr: call Scan_Key_Once
jnb key_pressed, wr;== ждем нажатия клавиши (wr- wait for pressed)
call make_scan_code
wu: call Scan_Key_Once
jnb key_pressed, wu;== ждем отпускания клавиши (wu- wait for
unpressed)
DSPL EQU KEYN;== DSPL = KEYN либо SCAN либо ASCII
IF (DSPL = SCAN);== начало блока условной компиляции
Write scan_code,DPTR_LEDreg_num;== либо вывести скан-код
нажатой клавиши
ELSEIF (DSPL = KEYN)
Write key_num,DPTR_LEDreg_num;== либо высветить номер
нажатой клавиши
ELSEIF (DSPL = ASCII);== либо высветить ее ASCII код
ENDIF;== конец блока условной компиляции
jmp _8;== бесконечный цикл
;=====
Scan_Key_Once:
mov cur_COL,#fst_COL;== начало однократного цикла
сканирования

```

```

pv:  Write cur_COL,DPTR_KEYreg_num;== записать "0" в текущий
      столбец
      Read cur_ROW,DPTR_KEYreg_num;== прочитать код строк
;== if cur_ROW<>1111b then (клавиша нажата)
      setb key_pressed;== установить бит "клавиша-нажата"
      anl c_R,#11110000b;== выделить 4 строки (биты D7..D4)
      cjne cur_ROW,#11110000b,scan_2_ASCII;== высветить код
      клавиши
;== else (клавиша НЕ нажата)
;==   if cur_COL<>end_COL then (не все столбцы)
      cjne cur_COL,#end_COL,nxt_COL;== переход к след.
      столбцу
;==   else (все столбцы просканированы - выйти из подпрограамы)
      clr key_pressed;== сбросить бит "клавиша-нажата"
;==   end if
;== end if
ext:
      ret;== выйти из подпрограамы
;=====
make_scan_code:
      mov a,cur_COL;== записать в аккумулятор код столбцов
      anl a,#0fh;== маской 00001111 выделить мл. тетраду столбцов
      mov scan_code,a;== и записать ее в скан-код
      mov a,cur_ROW;== записать в аккумулятор код строк
      anl a,#0f0h;== маской 11110000 выделить ст. тетраду строк
      orl scan_code,a;== и объединить мл. и ст. тетрады в скан-код
      ret
;=====
nxt_COL;== сдвиг "бегущего нуля"
      mov a,cur_COL
      rl a;== сдвинуть "бегущий 0" (сдвиг только в аккумуляторе)
      mov cur_COL,a

      jmp pv;== перейти к выводу "0" в следующий столбец
;=====
scan_2_ASCII:
      call scan_code_2_key_num
      jmp ext
;=====
scan_code_2_key_num:
;== подпрограмма преобразования скан-кода клавиши в ее порядковый
номер
      mov key_num,#0;== нач. значение = 0
      mov a,scan_code;== в мл. тетраде скан-код столбцов

```

```

pc:   rrc a;== сдвиг вправо кода столбцов в сторону младшего разряда
      jnc r;== выйти из подсчета числа столбцов, если "0" попал во флаг
переноса
      inc key_num;== иначе увеличить порядковый номер на 1
      jmp pc;== повторить цикл сдвига
r:    mov a,scan_code;== в ст. тетраде скан-код строк
      swp a;== меняем местами тетрады (скан-код строк теперь в
младшей)
nr:   rrc a;== сдвиг вправо в сторону младшего разряда
      jnc ex;== выйти из подсчета числа строк, если "0" попал во флаг
переноса
      inc key_num;== иначе увеличить порядковый номер на 4

      inc key_num
      inc key_num
      inc key_num
//REPT 4 ;== или так с помощью встроенного макроса
//   inc key_num
//ENDM
      jmp nr;== повторить цикл сдвига
ex:   ret;== возврат из подпрограммы
;=====
END

```

Снова оттранслируйте  и загрузите  программу. Далее нажимая на клавиши, убедитесь, что высвечиваемый порядковый номер от 0 до 15-ти (00000000...00001111) соответствует этим клавишам.

V-5) Подпрограмма преобразования порядкового номера клавиши в ASCII код.

Порядковый номер клавиши используется в нашей программе для обращения к нужному ASCII коду. ASCII коды располагаются в виде массива и имеют те же порядковые номера, что и клавиши. Наша задача записать в программе таблицу этих кодов, по какому-то адресу, например по адресу "key_ASCII". Далее, например, для извлечения из таблицы ASCII кода клавиши "8", к адресу таблицы нужно прибавить порядковый номер клавиши "8" и извлечь ее ASCII код в аккумулятор командой "movc a, @a + DPTR". Эта команда, как это видно из ее мнемоники и производит сложение и извлекает код. Затем в программе этот код высвечивается.

Добавьте в указанные места программы, следующие фрагменты.

```

ELSEIF (DSPL = ASCII)
    Write ascii_code,DPTR_LEDreg_num,== либо - ASCII код нажатой клавиши
ENDIF,== конец блока условной компиляции
    jmp _8,== бесконечный цикл
;=====
Scan_Key_Once:
.....

    jmp nr,== повторить цикл сдвига
ex: ret,== возврат из подпрограммы
;=====
key_num_2_ascii_code:
;== подпрограмма преобразования порядкового номера клавиши в ASCII код
    mov DPTR, #key_ASCII,== в DPTR записать адрес таблицы ASCII кодов
    mov a, key_num,== в аккумулятор записать номер клавиши в таблице
    movc a, @a + DPTR,== считать в аккумуляторе ASCII код клавиши
    mov ascii_code, a,== и поместить его в ячейку памяти ascii_code
    ret,== возврат из подпрограммы
key_ASCII:,== массив ASCII кодов клавиш
    DB '1','2','3','A';
    DB '4','5','6','B';
    DB '7','8','9','C';
    DB '*','0','#','D';
;=====
END
.....
scan_2_ASCII:
    call scan_code_2_key_num
    call key_num_2_ascii_code
    jmp ext

```

Присвойте вспомогательной переменной DSPL значение ASCII (строка программы “DSPL EQU ASCII”) и снова запустите программу. На светодиодах должны высвечиваться ASCII коды нажимаемых клавиш.

Результат покажите преподавателю

VI) Модификация программы в соответствии с заданием

Предположим, что в разрабатываемом вами устройстве используются не все 16 клавиш, а например, только 4, как показано на рисунке. Причем высвечиваться должны только коды этих клавиш. При нажатии на остальные никаких действий не производится.

1	2	3	A
4	5	6	B
7	8	9	C
*	0	#	D

Из нескольких вариантов решения остановимся на том, в котором сканирование ограничивается указанными клавишами. Причем алгоритм

сканирования остается прежним и требуется только **пересчитать некоторые константы**.

Произведите необходимые изменения.

Результат покажите преподавателю

VII) Использование прерываний от клавиатуры

В настоящей лабораторной работе сканирование клавиатуры производится непрерывно (...jnb key_pressed, wp;.... jб key_pressed, wu....). Возможен другой способ, когда сигналы линий, на которые заводится “бегущий 0” объединяются по “ИЛИ”, с помощью логического элемента “И” или “И-НЕ”. Выход этого элемента подключается к одному из входов для внешних прерываний (например int0 или int1). При нажатии на клавишу, “бегущий 0” попадает на вход intx и вызывает прерывание. В обработчике прерывания производится однократное сканирование и определение, какая клавиша нажата по методике, которая рассмотрена.

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”

anl DST, SRC - “поразрядное И”

orl DST, SRC - “поразрядное ИЛИ”

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

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

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

mov DST, SRC – скопировать операнд источник в приемник

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

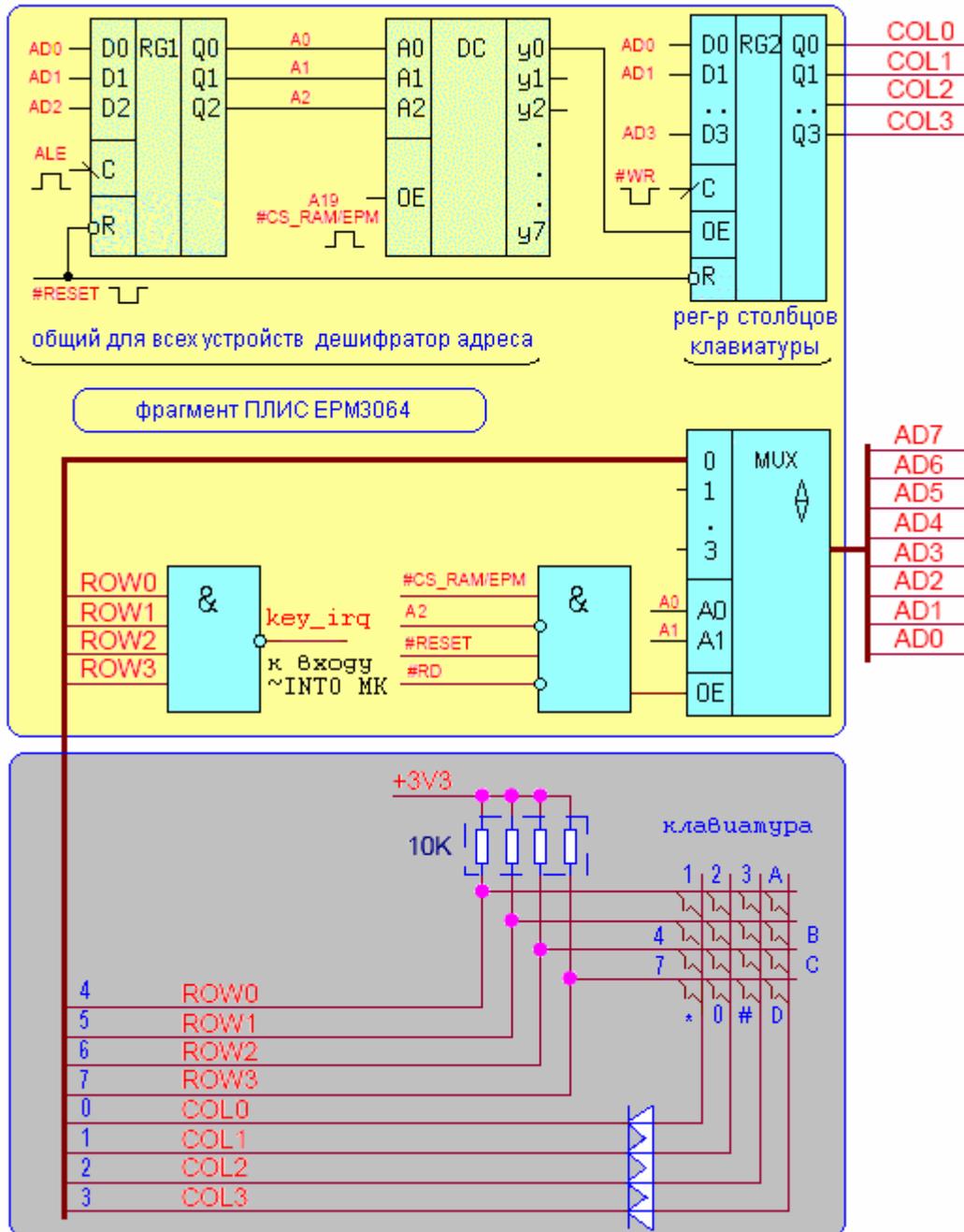
rr a (rl a) – циклический (круговой) сдвиг содержимого аккумулятора вправо (влево)

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

swap a – поменять местами тетрады аккумулятора

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

ПРИЛОЖЕНИЕ №2. Принципиальная схема подключения клавиатуры



На рисунке приведен фрагмент схемы, запрограммированной в ПЛИС EPM3064 для сканирования клавиатуры в виде матрицы 4 x 4 и считывающей код нажатой клавиши

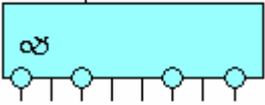
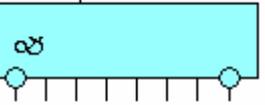
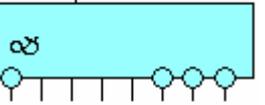
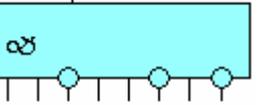
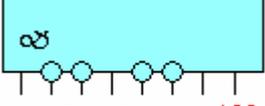
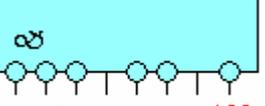
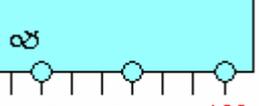
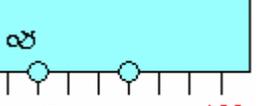
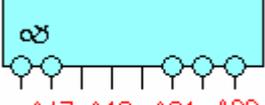
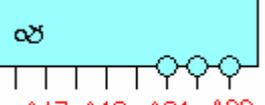
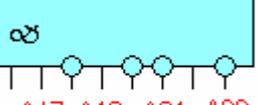
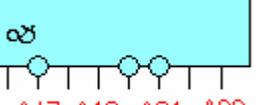
Определение факта нажатия на клавишу производится периодической записью на выходы COL_i регистра RG2 кода, в котором поочередно сменяются (сдвигаются) комбинации 0111, 1011, 1101, 1110 снова 0111 и т.д. Периодический сдвиг кода может производиться и в другом направлении. Такая смена кода часто называется “бегущим нулем”, а сам процесс – сканированием клавиатуры.

Если не нажата ни одна из клавиш, то на всех выходах ROW_i будет высокий уровень 3.3В (лог.1). При нажатии на одну из клавиш, “бегущий 0” максимум через 3 сдвига кода на столбцах матрицы COL_i попадет на тот ряд ROW_i, в котором нажата клавиша. Появление нуля хотя бы в одном из старших битов (4..7) и будет свидетельствовать о нажатии на клавишу. Определение момента замыкания контактов клавиши может осуществляться, либо периодическим чтением кода с выходов мультиплексора, либо однократно по сигналу запроса на прерывание “key_irq”. Логический элемент “И-НЕ” выполняет функцию “ИЛИ-НЕ” для сигналов ROW_i. Сигнал “key_irq” с его выхода можно направить на вход \sim INT0 микроконтроллера.

Резисторы служат для исключения короткого замыкания (КЗ) источника питания 3.3В при нажатии на клавишу. Диоды нужны для предотвращения короткого замыкания COL_j=1 и COL_i=0, при случайном нажатии на две и более клавиш в одном ряду.

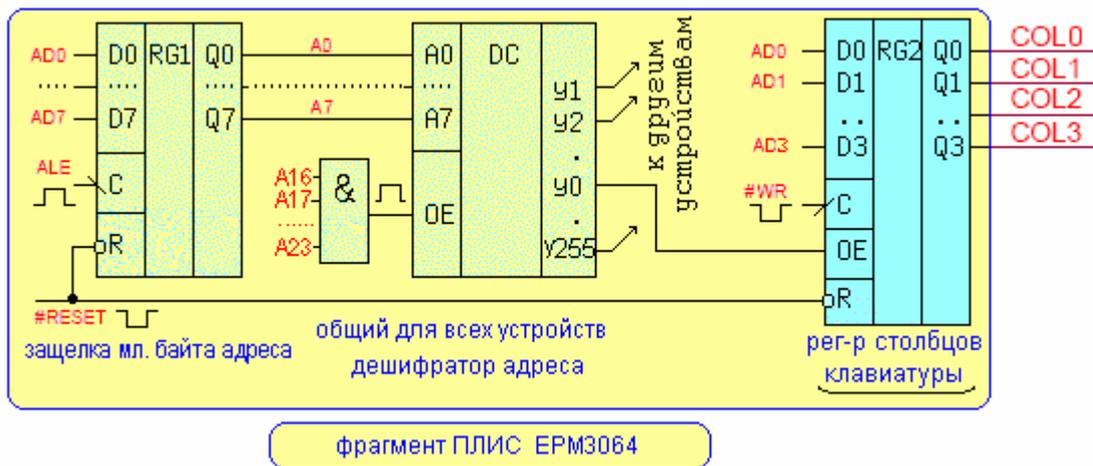
Мультиплексор приведенный на рисунке состоит из 8-ми параллельно включенных мультиплексоров “4 -> 1”, по одному мультиплексору на один выходной двоичный разряд.

ВАРИАНТЫ технического задания.

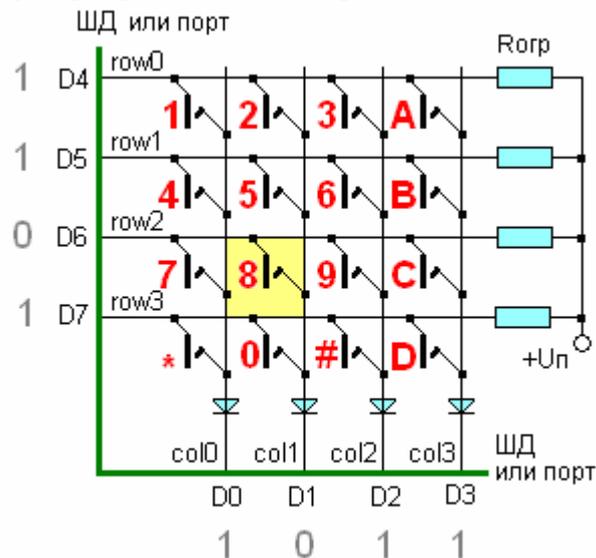
 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y88 Tochka_Vhoda 3e00 Ваша кл-ша "2"</p> <table border="1" data-bbox="231 660 435 860"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y32 Tochka_Vhoda 5b00 Ваша кл-ша "3"</p> <table border="1" data-bbox="545 660 750 860"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y40 Tochka_Vhoda 25c0 Ваша кл-ша "A"</p> <table border="1" data-bbox="858 660 1062 860"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y56 Tochka_Vhoda 7a00 Ваша кл-ша "4"</p> <table border="1" data-bbox="1171 660 1375 860"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y80 Tochka_Vhoda 3d50 Ваша кл-ша "7"</p> <table border="1" data-bbox="231 1243 435 1442"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y120 Tochka_Vhoda 2a00 Ваша кл-ша "*"</p> <table border="1" data-bbox="545 1243 750 1442"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y72 Tochka_Vhoda 4ee0 Ваша кл-ша "5"</p> <table border="1" data-bbox="858 1243 1062 1442"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y136 Tochka_Vhoda 5d00 Ваша кл-ша "0"</p> <table border="1" data-bbox="1171 1243 1375 1442"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y16 Tochka_Vhoda 6AB0 Ваша кл-ша "8"</p> <table border="1" data-bbox="231 1825 435 2024"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y24 Tochka_Vhoda 44E0 Ваша кл-ша "#"</p> <table border="1" data-bbox="545 1825 750 2024"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=Y8 Tochka_Vhoda 5E30 Ваша кл-ша "9"</p> <table border="1" data-bbox="858 1825 1062 2024"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Y_x=144 Tochka_Vhoda 2A00 Ваша кл-ша "B"</p> <table border="1" data-bbox="1171 1825 1375 2024"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																

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 битного адреса RG2. Выполнение команд “MOVX @DPTR,A” и “MOVX A,@DPTR” 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) Пояснить по рисунку принцип сканирования клавиатуры



- 1) Уметь комментировать и понимать работу основной программы. Для чего нужны директивы условной компиляции.

```

;=====
CSEG at 0
jmp Tochka_Vhoda,== резидентный загрузчик SAM переходит по этому адресу
ORG Tochka_Vhoda,== с этого адреса располагается код программы
main_prog: ;== точка входа в основную программу
mov DPP,#DPP_page_num,== загрузить адрес страницы, на кот. находятся BV
mov sp,#dno_steka,== загрузить в регистр sp адрес дна стека
clr key_pressed,== обнулить бит "клавиша нажата"
_8:
wp: call Scan_Key_Once
jnb key_pressed, wp,== ждем нажатия клавиши (wp- wait for pressed)
// delay 30,== дребезга нет (задержка не нужна)
call make_scan_code
wu: call Scan_Key_Once
jb key_pressed, wu,== ждем отпущения клавиши (wu- wait for unpressed)
// delay 30,== дребезга нет (задержка не нужна)
DSPL EQU ASCII,== DSPL = KEYN либо SCAN либо ASCII
IF (DSPL = SCAN),== начало блока условной компиляции
Write scan_code,DPTR_LEDreg_num,== либо вывести скан-код нажатой клавиши
ELSEIF (DSPL = KEYN)
Write key_num,DPTR_LEDreg_num,== либо высветить номер нажатой клавиши
ELSEIF (DSPL = ASCII)
Write ascii_code,DPTR_LEDreg_num,== либо - ASCII код нажатой клавиши
ENDIF,== конец блока условной компиляции
jmp _8,== бесконечный цикл
;=====

```

2) Уметь комментировать и понимать работу этих подпрограмм:

```

Scan_Key_Once:
mov cur_COL,#fst_COL,== начало однократного цикла сканирования
pv: Write cur_COL,DPTR_KEYreg_num,== записать "0" в текущий столбец
Read cur_ROW,DPTR_KEYreg_num,== прочитать код строк
;== if cur_ROW<>1111b then (клавиша нажата)
setb key_pressed,== установить бит "клавиша-нажата"
anl c_R,#11110000b,== выделить 4 строки (биты D7..D4)
cjne cur_ROW,#11110000b,scan_2_ASCII,== высветить код клавиши
;== else (клавиша НЕ нажата)
;== if cur_COL<>end_COL then (не все столбцы)
cjne cur_COL,#end_COL,nxt_COL,== переход к след. столбцу
;== else (все столбцы просканированы - выйти из подпрограммы)
clr key_pressed,== сбросить бит "клавиша-нажата"
;== end if
;== end if
ext:
ret,== выйти из подпрограммы
;=====
nxt_COL:;== сдвиг "бегущего нуля"
mov a,cur_COL
rl a,== сдвинуть "бегущий 0" (сдвиг только в аккумуляторе)
mov cur_COL,a
jmp pv,== перейти к выводу "0" в следующий столбец
;=====
scan_2_ASCII:
call scan_code_2_key_num
call key_num_2_ascii_code
jmp ext

```

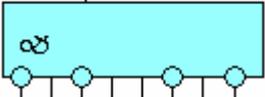
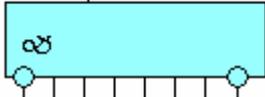
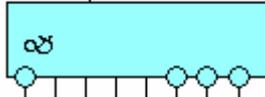
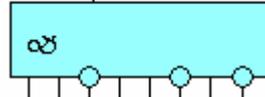
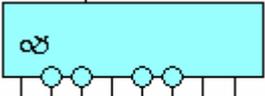
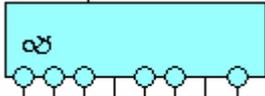
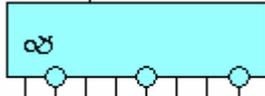
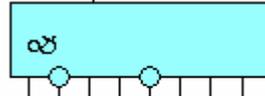
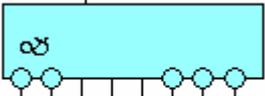
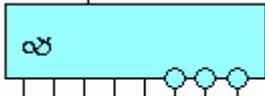
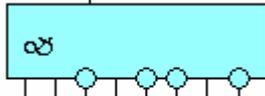
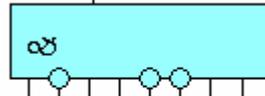
```

;=====
scan_code_2_key_num:
;== подпрограмма преобразования скан-кода клавиши в ее порядковый номер
mov key_num,#0;== нач. значение = 0
mov a,scan_code;== в мл. тетраде скан-код столбцов
nc: rrc a;== сдвиг вправо кода столбцов в сторону младшего разряда
jnc r;== выйти из подсчета числа столбцов, если "0" попал во флаг переноса
inc key_num;== иначе увеличить порядковый номер на 1
jmp nc;== повторить цикл сдвига
r: mov a,scan_code;== в ст. тетраде скан-код строк
swap a;== меняем местами тетрады (скан-код строк теперь в младшей)
nr: rrc a;== сдвиг вправо в сторону младшего разряда
jnc ex;== выйти из подсчета числа строк, если "0" попал во флаг переноса
inc key_num;== иначе увеличить порядковый номер на 4
inc key_num
inc key_num
inc key_num
//REPT 4 ;== или так с помощью встроенного макроса
// inc key_num
//ENDM
jmp nr;== повторить цикл сдвига
ex: ret;== возврат из подпрограммы
;=====
key_num_2_ascii_code:
;== подпрограмма преобразования порядкового номера клавиши в ASCII код
mov DPTR, #key_ASCII;== в DPTR записать адрес таблицы ASCII кодов
mov a, key_num;== в аккумулятор записать номер клавиши в таблице
movc a, @a + DPTR;== считать в аккумуля. ASCII код клавиши
mov ascii_code, a;== и поместить его в ячейку памяти ascii_code
ret;== возврат из подпрограммы
key_ASCII:;== массив ASCII кодов клавиш
DB '1','2','3','A';
DB '4','5','6','B';
DB '7','8','9','C';
DB '*','0','#','D';
;=====

```

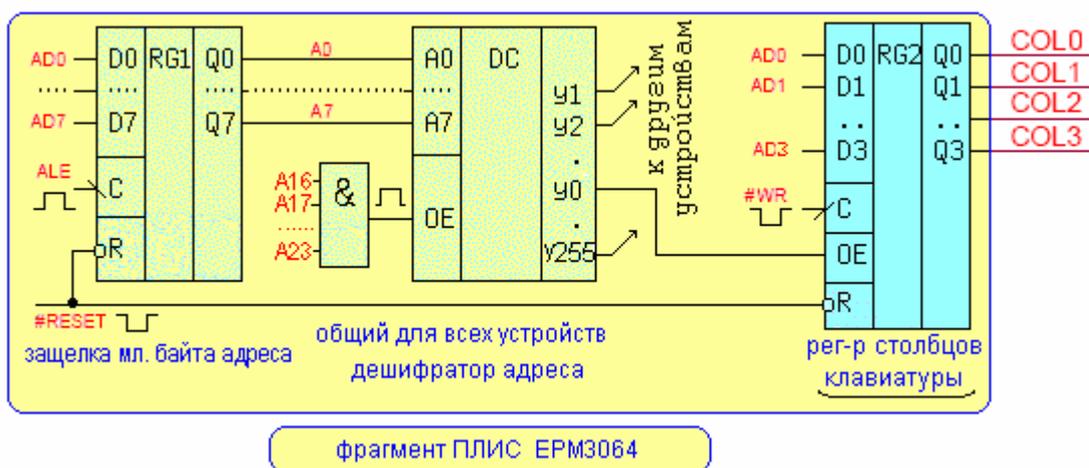
ВНИМАНИЕ: Во время экзамена фрагменты программы будут предъявляться **БЕЗ КОММЕНТАРИЕВ.**

ВАРИАНТЫ технического задания.

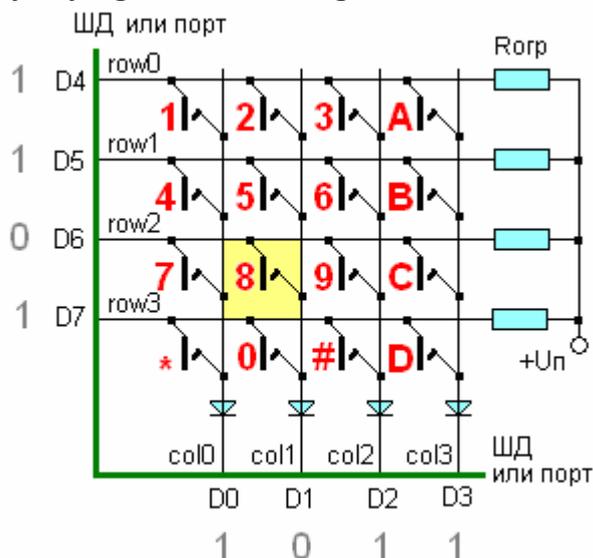
 <p>A17 A19 A21 A23 A16 A18 A20 A22 Yx=Y88 Tochka_Vhoda 3e00 Ваша кл-ша "2"</p> <table border="1" data-bbox="231 633 435 837"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Yx=Y32 Tochka_Vhoda 5b00 Ваша кл-ша "3"</p> <table border="1" data-bbox="537 633 742 837"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Yx=Y40 Tochka_Vhoda 25c0 Ваша кл-ша "A"</p> <table border="1" data-bbox="844 633 1048 837"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Yx=Y56 Tochka_Vhoda 7a00 Ваша кл-ша "4"</p> <table border="1" data-bbox="1150 633 1355 837"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
 <p>A17 A19 A21 A23 A16 A18 A20 A22 Yx=Y80 Tochka_Vhoda 3d50 Ваша кл-ша "7"</p> <table border="1" data-bbox="231 1211 435 1415"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Yx=Y120 Tochka_Vhoda 2a00 Ваша кл-ша "*"</p> <table border="1" data-bbox="537 1211 742 1415"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Yx=Y72 Tochka_Vhoda 4ee0 Ваша кл-ша "5"</p> <table border="1" data-bbox="844 1211 1048 1415"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Yx=Y136 Tochka_Vhoda 5d00 Ваша кл-ша "0"</p> <table border="1" data-bbox="1150 1211 1355 1415"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
 <p>A17 A19 A21 A23 A16 A18 A20 A22 Yx=Y16 Tochka_Vhoda 6AB0 Ваша кл-ша "8"</p> <table border="1" data-bbox="231 1789 435 1991"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Yx=Y24 Tochka_Vhoda 44E0 Ваша кл-ша "#"</p> <table border="1" data-bbox="537 1789 742 1991"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Yx=Y8 Tochka_Vhoda 5E30 Ваша кл-ша "9"</p> <table border="1" data-bbox="844 1789 1048 1991"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D	 <p>A17 A19 A21 A23 A16 A18 A20 A22 Yx=144 Tochka_Vhoda 2A00 Ваша кл-ша "B"</p> <table border="1" data-bbox="1150 1789 1355 1991"> <tr><td>1</td><td>2</td><td>3</td><td>A</td></tr> <tr><td>4</td><td>5</td><td>6</td><td>B</td></tr> <tr><td>7</td><td>8</td><td>9</td><td>C</td></tr> <tr><td>*</td><td>0</td><td>#</td><td>D</td></tr> </table>	1	2	3	A	4	5	6	B	7	8	9	C	*	0	#	D
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																
1	2	3	A																																																																
4	5	6	B																																																																
7	8	9	C																																																																
*	0	#	D																																																																

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 битного адреса RG2. Выполнение команд “MOVX @DPTR,A” и “MOVX A,@DPTR” 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) Пояснить по рисунку принцип сканирования клавиатуры



- 3) Уметь комментировать и понимать работу основной программы. Для чего нужны директивы условной компиляции.

```

;=====
CSEG at 0
jmp Tochka_Vhoda,== резидентный загрузчик SAM переходит по этому адресу
ORG Tochka_Vhoda,== с этого адреса располагается код программы
main_prog: ;== точка входа в основную программу
mov DPP,#DPP_page_num,== загрузить адрес страницы, на кот. находятся BV
mov sp,#dno_steka,== загрузить в регистр sp адрес дна стека
clr key_pressed,== обнулить бит "клавиша нажата"
_8:
wp: call Scan_Key_Once
jnb key_pressed, wp,== ждем нажатия клавиши (wp- wait for pressed)
// delay 30,== дребезга нет (задержка не нужна)
call make_scan_code
wu: call Scan_Key_Once
jb key_pressed, wu,== ждем отпущения клавиши (wu- wait for unpressed)
// delay 30,== дребезга нет (задержка не нужна)
DSPL EQU ASCII,== DSPL = KEYN либо SCAN либо ASCII
IF (DSPL = SCAN),== начало блока условной компиляции
Write scan_code,DPTR_LEDreg_num,== либо вывести скан-код нажатой клавиши
ELSEIF (DSPL = KEYN)
Write key_num,DPTR_LEDreg_num,== либо высветить номер нажатой клавиши
ELSEIF (DSPL = ASCII)
Write ascii_code,DPTR_LEDreg_num,== либо - ASCII код нажатой клавиши
ENDIF,== конец блока условной компиляции
jmp _8,== бесконечный цикл
;=====

```

4) Уметь комментировать и понимать работу этих подпрограмм:

```

Scan_Key_Once:
mov cur_COL,#fst_COL,== начало однократного цикла сканирования
pv: Write cur_COL,DPTR_KEYreg_num,== записать "0" в текущий столбец
Read cur_ROW,DPTR_KEYreg_num,== прочитать код строк
;== if cur_ROW<>1111b then (клавиша нажата)
setb key_pressed,== установить бит "клавиша-нажата"
anl c_R,#11110000b,== выделить 4 строки (биты D7..D4)
cjne cur_ROW,#11110000b,scan_2_ASCII,== высветить код клавиши
;== else (клавиша НЕ нажата)
;== if cur_COL<>end_COL then (не все столбцы)
cjne cur_COL,#end_COL,nxt_COL,== переход к след. столбцу
;== else (все столбцы просканированы - выйти из подпрограммы)
clr key_pressed,== сбросить бит "клавиша-нажата"
;== end if
;== end if
ext:
ret,== выйти из подпрограммы
;=====
nxt_COL:;== сдвиг "бегущего нуля"
mov a,cur_COL
rl a,== сдвинуть "бегущий 0" (сдвиг только в аккумуляторе)
mov cur_COL,a
jmp pv,== перейти к выводу "0" в следующий столбец
;=====
scan_2_ASCII:
call scan_code_2_key_num
call key_num_2_ascii_code
jmp ext

```

```

;=====
scan_code_2_key_num:
;== подпрограмма преобразования скан-кода клавиши в ее порядковый номер
mov key_num,#0;== нач. значение = 0
mov a,scan_code;== в мл. тетраде скан-код столбцов
nc: rrc a;== сдвиг вправо кода столбцов в сторону младшего разряда
jnc r;== выйти из подсчета числа столбцов, если "0" попал во флаг переноса
inc key_num;== иначе увеличить порядковый номер на 1
jmp nc;== повторить цикл сдвига
r: mov a,scan_code;== в ст. тетраде скан-код строк
swap a;== меняем местами тетрады (скан-код строк теперь в младшей)
nr: rrc a;== сдвиг вправо в сторону младшего разряда
jnc ex;== выйти из подсчета числа строк, если "0" попал во флаг переноса
inc key_num;== иначе увеличить порядковый номер на 4
inc key_num
inc key_num
inc key_num
//REPT 4 ;== или так с помощью встроенного макроса
// inc key_num
//ENDM
jmp nr;== повторить цикл сдвига
ex: ret;== возврат из подпрограммы
;=====
key_num_2_ascii_code:
;== подпрограмма преобразования порядкового номера клавиши в ASCII код
mov DPTR, #key_ASCII;== в DPTR записать адрес таблицы ASCII кодов
mov a, key_num;== в аккумулятор записать номер клавиши в таблице
movc a, @a + DPTR;== считать в аккумуля. ASCII код клавиши
mov ascii_code, a;== и поместить его в ячейку памяти ascii_code
ret;== возврат из подпрограммы
key_ASCII:;== массив ASCII кодов клавиши
DB '1','2','3','A';
DB '4','5','6','B';
DB '7','8','9','C';
DB '*','0','#','D';
;=====

```

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 Рос. Федерации; Федер. агентство по образованию ; СПбГУ ИТМО, [Каф. электроники]. СПб.: СПбГУ ИТМО, 2007 .— 87 с.: ил .— (Приоритетные национальные проекты. Образование).
2. Китаев Ю.В. Основы программирования микроконтроллеров ATMEGA 128 и 68HC908: [учебное пособие], М-во образования и науки Рос. Федерации; Федер. агентство по образованию ; СПбГУ ИТМО, [Каф. электроники] .— СПб.: СПбГУ ИТМО, 2007 .— 107 с.: ил .— (Приоритетные национальные проекты. Образование)
3. Китаев Ю.В. Лабораторные и практические работы "Электроника и МП техника", [учебное пособие], М-во образования и науки Рос. Федерации; Федер. агентство по образованию ; СПбГУ ИТМО, [Каф. электроники]. СПб.: СПбГУ ИТМО, 2008 .— 92 с.: ил .— (Приоритетные национальные проекты. Образование).
4. Интерактивный курс “Цифровая и микропроцессорная техника” - 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-51

Учебное пособие

В авторской редакции

Дизайн

Верстка

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 ____ экз.

Отпечатано на ризографе

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

