МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

Колледж Пятигорского института (филиал) СКФУ в г.Пятигорке

МДК.02.01 Микропроцессорные системы
МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

Специальности СПО

09.02.01 Компьютерные системы и комплексы

Квалификация: техник по компьютерным системам

Пятигорск 2022 г.

Методические указания для практических занятий по дисциплине МДК.02.01 Микропроцессорные системы составлены в соответствии с требованиями ФГОС СПО. Предназначены для студентов, обучающихся по специальности 09.02.01 Компьютерные системы и комплексы

Введение

Целью данных методических указаний является создание единого учебно-методического комплекса по изучению дисциплины «Микропроцессорные системы».

Данное методическое указание предназначено для студентов третьего курса СПО специальности 09.02.01 «Компьютерные системы и комплексы» очной формы обучения и содержат материалы для практических занятий.

В результате освоения учебной дисциплины обучающийся должен **уметь**

-составлять программы на языке ассемблера для микропроцессорных систем;

-производить тестирование и отладку микропроцессорных систем (далее - МПС);

-выбирать микроконтроллер/микропроцессор для конкретной системы управления;

-осуществлять установку и конфигурирование персональных компьютеров и подключение периферийных устройств;

-подготавливать компьютерную систему к работе;

-проводить инсталляцию и настройку компьютерных систем;

-выявлять причины неисправностей и сбоев, принимать меры по их устранению;

В результате освоения учебной дисциплины обучающийся должен **знать:**

-базовую функциональную схему МПС;

-программное обеспечение микропроцессорных систем;

структуру типовой системы управления (контроллер) и организацию микроконтроллерных систем;

-методы тестирования и способы отладки МПС;

-информационное взаимодействие различных устройств через информационно-телекоммуникационную сеть "Интернет" (далее - сеть Интернет);

-состояние производства и использование МПС;

-способы конфигурирования и установки персональных компьютеров, программную поддержку их работы;

-классификацию, общие принципы построения и физические основы работы периферийных устройств;

-способы подключения стандартных и нестандартных программных утилит;

-причины неисправностей и возможных сбоев.

ОГЛАВЛЕНИЕ

ПРАКТИЧЕСКАЯ РАБОТА №1. ОЗНАКОМЛЕНИЕ С РАБОТОЙ УЧЕБНОГО МИКРОПРОЦЕССОРНОГО КОМПЛЕКСА....6

ПРАКТИЧЕСКАЯ РАБОТА №2. ИЗУЧЕНИЕ РАБОТЫ УМК В ПОШАГОВОМ РЕЖИМЕ....14

ПРАКТИЧЕСКАЯ РАБОТА №3. ИЗУЧЕНИЕ РЕГИСТРОВ МИКРОПРОЦЕССОРА....18

ПРАКТИЧЕСКАЯ РАБОТА №4. ИЗУЧЕНИЕ МЕТОДОВ АДРЕСАЦИИ ПАМЯТИ....23

ПРАКТИЧЕСКАЯ РАБОТА №5. ИЗУЧЕНИЕ АРИФМЕТИЧЕСКИХ КОМАНД....29

ПРАКТИЧЕСКАЯ РАБОТА №6. ИЗУЧЕНИЕ КОМАНД ВВОДА-ВЫВОДА....36

ПРАКТИЧЕСКАЯ РАБОТА №7. ИЗУЧЕНИЕ КОМАНД МАНИПУЛЯЦИИ СТЕКОМ. ВЫЗОВ ПОДПРОГРАММЫ И ВОЗВРАТ ИЗ НЕЁ....39

ПРАКТИЧЕСКАЯ РАБОТА №8. ИЗУЧЕНИЕ КОМАНД БЕЗУСЛОВНОГО И УСЛОВНОГО ПЕРЕХОДОВ. 44

ПРАКТИЧЕСКАЯ РАБОТА №9. ОРГАНИЗАЦИЯ ПРЕРЫВАНИЙ В МИКРОПРОЦЕССОРНОЙ СИСТЕМЕ52

ПРАКТИЧЕСКАЯ РАБОТА № 10. ОРГАНИЗАЦИЯ ПРЯМОГО ДОСТУПА К ПАМЯТИ....56

ПРАКТИЧЕСКАЯ РАБОТА №11. ИЗУЧЕНИЕ РАБОТЫ ОЗУ (ТЕСТИРОВАНИЕ И ОТЛАДКА)....58

ПРАКТИЧЕСКАЯ РАБОТА №12. ОСОБЕННОСТИ АРХИТЕКТУРЫ ОДНОКРИСТАЛЬНЫХ МИКРОКОНТРОЛЛЕРОВ....63

ПРАКТИЧЕСКАЯ РАБОТА №13. СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА AVR....70

ПРАКТИЧЕСКАЯ РАБОТА №14. ИЗУЧЕНИЕ СПОСОБОВ АДРЕСАЦИИ ОПЕРАНДОВ В AVR-МИКРОКОНТРОЛЛЕРАХ....77

ПРАКТИЧЕСКАЯ РАБОТА №15. ЭЛЕМЕНТАРНЫЕ ПРИЕМЫ ПРОГРАММИРОВАНИЯ (ВЕТВЛЕНИЯ И ЦИКЛЫ)....81

ПРАКТИЧЕСКАЯ РАБОТА №16. ЭЛЕМЕНТАРНЫЕ ПРИЕМЫ ПРОГРАММИРОВАНИЯ (ПОДПРОГРАММЫ). 84

ПРАКТИЧЕСКАЯ РАБОТА №17. ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА AVR-МИКРОКОНТРОЛЛЕРА....88

ПРАКТИЧЕСКАЯ РАБОТА №18. ИЗУЧЕНИЕ СИСТЕМЫ ПРЕРЫВАНИЙ AVR-МИКРОКОНТРОЛЛЕРА.92

ПРАКТИЧЕСКАЯ РАБОТА №19. ПРОГРАММА УПРАВЛЕНИЯ ТАЙМЕР-СЧЕТЧИКОМ В РЕЖИМЕ ШИРОТНО-ИМПУЛЬСНОЙ МОДУЛЯЦИИ....97

ПРАКТИЧЕСКАЯ РАБОТА №20. ПРОГРАММИРОВАНИЕ АНАЛОГОВОГО КОМПАРАТОРА МИКРОКОНТРОЛЛЕРОВ....107

ПРАКТИЧЕСКАЯ РАБОТА №21. СИСТЕМЫ УПРАВЛЕНИЯ И КОНТРОЛЯ НА ОДНОКРИСТАЛЬНЫХ МИКРОКОНТРОЛЛЕРАХ ФИРМЫ MICROCHIP.....112

ПРАКТИЧЕСКАЯ РАБОТА №22. КОММУНИКАЦИОННЫЕ МИКРОКОНТРОЛЛЕРЫ (ЦИФРОВЫЕ СИГНАЛЬНЫЕ ПРОЦЕССОРЫ (ЦСП) ФИРМЫ *MICROCHIP*)....120

ПРАКТИЧЕСКАЯ РАБОТА №23. КОММУНИКАЦИОННЫЕ МИКРОКОНТРОЛЛЕРЫ (РЕГИСТРЫ МИКРОПРОЦЕССОРА)....124

ПРАКТИЧЕСКАЯ РАБОТА №24. КОММУНИКАЦИОННЫЕ МИКРОКОНТРОЛЛЕРЫ (КОМАНДЫ ПЕРЕСЫЛКИ ДАННЫХ И АРИФМЕТИЧЕСКИЕ КОМАНДЫ)....128

ПРАКТИЧЕСКАЯ РАБОТА №25. КОММУНИКАЦИОННЫЕ МИКРОКОНТРОЛЛЕРЫ (ОРГАНИЗАЦИЯ ЦИКЛОВ)....131

ПРАКТИЧЕСКАЯ РАБОТА №26. ПРОГРАММНОЕ УПРАВЛЕНИЕ ПОРТАМИ ВВОДА/ВЫВОДА МИКРОКОНТРОЛЛЕРА PIC16F84A....133

ПРАКТИЧЕСКАЯ РАБОТА №27. МИКРОКОНТРОЛЛЕРЫ СЕМЕЙСТВА MK51....141

ПРАКТИЧЕСКАЯ РАБОТА №28. СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРА СЕМЕЙСТВА MK51.....147

ПРАКТИЧЕСКАЯ РАБОТА №29. ПРИЕМЫ РАБОТЫ С ВИДЕОПАМЯТЬЮ....151

ПРАКТИЧЕСКАЯ РАБОТА №30. ВЗАИМОДЕЙСТВИЕ С ВНЕШНИМИ УСТРОЙСТВАМИ....154

ПРАКТИЧЕСКАЯ РАБОТА №31. ПРОГРАММИРОВАНИЕ ПЕРИФЕРИЙНЫХ УСТРОЙСТВ, ДОСТУПНЫХ ЧЕРЕЗ ШИНУ I2C....157

ПРАКТИЧЕСКАЯ РАБОТА №32. ИЗУЧЕНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ТАЙМЕР-СЧЕТЧИКА МИКРОКОНТРОЛЛЕРА MCS-51.....160

ПРАКТИЧЕСКАЯ РАБОТА №33. ИССЛЕДОВАНИЕ СИСТЕМЫ ПРЕРЫВАНИЙ МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА MCS-51.....165

ПРАКТИЧЕСКАЯ РАБОТА №34. РАЗРАБОТКА ПРОГРАММЫ ИЗМЕРЕНИЯ НАПРЯЖЕНИЯ.....167

ПРАКТИЧЕСКАЯ РАБОТА №35. ИССЛЕДОВАНИЕ РАБОТЫ ТАЙМЕРОВ/СЧЕТЧИКОВ СОБЫТИЙ МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА MCS-51.....171

ПРАКТИЧЕСКАЯ РАБОТА №36. УСТАНОВКА ПЕРИФЕРИЙНЫХ УСТРОЙСТВ.....173

ПРАКТИЧЕСКАЯ РАБОТА №37. ГЕНЕРАТОР ИМПУЛЬСНОГО СИГНАЛА.....175

ПРАКТИЧЕСКАЯ РАБОТА №38. ПРОГРАММНЫЕ МОДЕЛИ АППАРАТНЫХ СРЕДСТВ МИКРОПРОЦЕССОРНЫХ СИСТЕМ.....178

ПРАКТИЧЕСКАЯ РАБОТА №39. АППАРАТУРА И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ТИПОВОЙ МИКРОПРОЦЕССОРНОЙ СИСТЕМЫ.....182

ПРАКТИЧЕСКАЯ РАБОТА №40. СБРОС И ПРЕРЫВАНИЯ МИКРОКОНТРОЛЛЕРОВ В ПРОЦЕССЕ ВЫПОЛНЕНИЯ ПРОГРАММЫ УПРАВЛЕНИЯ.....190

ПРАКТИЧЕСКАЯ РАБОТА №41. ТЕСТИРОВАНИЕ И ОТЛАДКА МИКРОПРОЦЕССОРНЫХ СИСТЕМ..195

ПРАКТИЧЕСКАЯ РАБОТА №42. ПОЛУЧЕНИЕ НАВЫКОВ РАБОТЫ С УПРАВЛЯЮЩЕЙ ПРОГРАММОЙ ДЛЯ ОТЛАДКИ МИКРОПРОЦЕССОРНОЙ СИСТЕМЫ.....206

ПРАКТИЧЕСКАЯ РАБОТА №43. ТЕСТИРОВАНИЕ И ОТЛАДКА МИКРОПРОЦЕССОРНЫХ СИСТЕМ ПРИ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.....211

ПРАКТИЧЕСКАЯ РАБОТА №1. ОЗНАКОМЛЕНИЕ С РАБОТОЙ УЧЕБНОГО МИКРОПРОЦЕССОРНОГО КОМПЛЕКСА.

Цель работы: Изучение микропроцессорного комплекта КР580ВМ80А. Основные команды. Правила ввода данных. Управление вводом/выводом. Индикация работы УМК.

Теоретическая часть

Учебный микропроцессорный комплект (УМК) создан на базе микропроцессора КР580ВМ80А и предназначен для подготовки специалистов в области микропроцессорной техники и обучению основам программирования.

Микропроцессор КР580ВМ80А предназначен для выполнения определенного набора команд (представленного в приложении 1) и реализован на одной БИС.

Все команды микропроцессора можно разделить на 8 основных групп:

команды пересылки данных и загрузки регистров;

команды работы с памятью;

арифметические, логические команды и команды сравнения и сдвига;

команды передачи управления;

команды манипуляции стеком;

команды вызова подпрограмм и возврата из них;

команды ввода/вывода;

команды обработки прерываний.

Для оперативного хранения данных в микропроцессоре имеется семь 8-разрядных регистров общего назначения:

A - регистр-накопитель, аккумулятор;

B - регистр общего назначения;

C - регистр общего назначения;

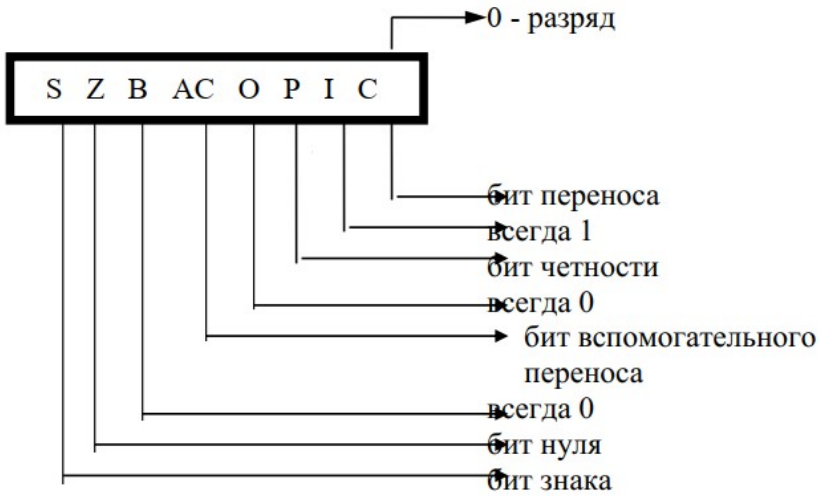
D - регистр общего назначения;

E - регистр общего назначения;

H - регистр косвенного адреса;

L - регистр косвенного адреса.

Для хранения битов условий имеется 8-разрядный регистр признаков PSW. Распределение и назначение битов условий регистров PSW следующее:



Бит переноса устанавливается в 1, если возникает переполнение старшего разряда в байте, в противном случае устанавливается в 0.

Бит четности устанавливается в 1, если количество единиц в байте четное, в противном случае устанавливается в 0.

Бит вспомогательного переноса устанавливается в 1, если возникает перенос из младшей тетрады байта в старшую, в противном случае - в 0.

Бит нуля устанавливается в 1, если результат выполнения операции равен 0, в противном случае в 0.

Бит знака устанавливается в 1, если результат операции отрицательный (ст. разряд байта равен 1), в противном случае, устанавливается в 0.

16-разрядный регистр-указатель стека SP служит для определения нижней границы области стека.

16-разрядный регистр-счетчик команд PC служит для хранения адреса выполненной команды.

Двоичная и шестнадцатиричная система счисления.

В цифровой технике для представления данных используют двоичную систему счисления, т.к. для этих систем используемый цифровой сигнал может иметь только значения - 0 и 1. Т.о., одним разрядом двоичного числа можно представить два числа. Используя два разряда можно представить 4 числа ит. д... В микропроцессоре используются 8-разрядные двоичные слова, называемые байтами. Каждый байт может представить число от 0 до 255, в зависимости от комбинации в нем 0 и 1.

Например.

00000000 - 0
00000001 - 1
00000010 - 2
00000011 – 3
…
00001111 – 15
…
11111111 - 255

Для удобства оперирования двоичными данными используют их шестнадцатиричное представление. Для этого байт делят пополам (на тетрады) и каждую тетраду кодируют следующим образом:

0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - A
0011 - 3	1011 - B
0100 - 4	1100 - C
0101 - 5	1101 - D
0110 - 6	1110 - E
0111 - 7	1111 - F

Каждая тетрада представляет 1 из 16 чисел. При этом вес старшей тетрады в байте равен 16. Например, число 10 в шестнадцатиричном коде представляет число 16.

Например:

двоичное	шестнадцатиричное	десятичное
00010000	10	16
00100100	24	36
11111111	FF	255

В дальнейшем используется шестнадцатиричная система счисления для представления данных.

Содержание работы

1. Ознакомление с органами управления, ввода, индикации УМК.
2. Включение и запуск УМК.
3. Команды системного монитора УМК.
4. Просмотр и модификация содержимого ячейки памяти.

5. Просмотр и модификация содержимого регистра микропроцессора (МП).
6. Старт программы.
7. Заполнение массива памяти константой.
8. Подсчет контрольной суммы.
9. Перемещение массива памяти.

Выполнение работы

1. Ознакомление с органами управления, ввода, индикации.

На лицевой панели УМК размещены кнопки включения/выключения, сброса и прерывания, кнопки управления пошаговым режимом работы УМК, функциональная клавиатура, клавиатура ввода данных, шести сегментный дисплей, светодиоды индикации состояния шины данных, шины адреса, управляющие сигналы МП.

Кнопка включения/выключения УМК СЕТЬ расположена в левой нижней части лицевой панели. На самой кнопке выгравирован символ " ~ ". Эта кнопка служит для включения (нажатое состояние кнопки) и выключения (отжатое состояние кнопки) УМК. Над кнопкой размещены три светодиода, над которыми выгравировано: +5В,-5В,+12В. При перегрузках срабатывает защита блока питания и загорается соответствующий светодиод. В случае необходимо выключить УМК и вызвать специалиста по обслуживанию. Кнопка СБРОС расположена в правой средней части лицевой панели УМК. На кнопке выгравировано СБ. Эта кнопка служит для инициализации системной программы МОНИТОР, и лицевой позиции шестигсегментного дисплея появляется символ " - ". УМК готов к приему команд.

Кнопка ПРЕРЫВАНИЕ располагается под кнопкой СБРОС. На ней выгравировано ПР. При нажатии на эту кнопку вырабатывается сигнал ЗАПРОС НА ПРЕРЫВАНИЕ 7-ого уровня и, если прерывания разрешены (выполнена команда EI), будет закончено выполнение текущей команды, и управление будет передано на адрес 38Н (что соответствует седьмому уровню прерываний). Начиная с этого адреса, располагается программа обработки прерывания. Если кнопка ПР будет нажата во время работы программы МОНИТОР (на любой стадии), на дисплей будет выведен символ " ? ". В противном случае на дисплей будет выведен адрес точки прерывания и управление будет передано программе МОНИТОР.

Управление пошаговым режимам работы УМК производится с помощью кнопок, на которых выгравированы надписи: РБ/ЩГ,

КМ/ЦК и ШГ. С помощью этих кнопок может быть установлен один из двух режимов работы УМК по шагам. Первый режим - покомандный. Для установки этого режима необходимо нажать кнопку РБ/ШГ (она останется в нажатом состоянии).

Для выполнения команд необходимо нажать кнопку ШГ. Каждое нажатие кнопки ШГ вызовет выполнение текущей команды. При этом на светодиодах индикации состояние шин данных, адреса и управляющих сигналов, расположенных в центре панели УМК, будут высвечиваться в двоичном коде (каждый светодиод отражает соответствующий разряд), соответственно, адрес и код выполняемой команды, а также управляющие сигналы МП.

Второй режим работы по шагам - работа по командным циклам. Для установки этого режима необходимо нажать кнопки: РБ/ШГ и КМ/ЦК. В этом случае можно проследить ход выполнения команды. При каждом нажатии на кнопку ШГ будет выполнен следующий машинный цикл. При этом на светодиодах индикации будет отражаться информация, соответствующая каждому машинному циклу.

Для работы в автоматическом режиме обе клавиши РБ/ШГ и КМ/ЦК должны быть в отжатом состоянии. Клавиатура УМК располагается в правой нижней части лицевой панели и разделена на две части. В левой части расположены функциональные клавиши. За каждой закреплена определенная функция системной программы МОНИТОР. На них выгравированы соответствующие идентификаторы функции:

П - просмотр и модификация содержимого ячейки памяти;

РГ - просмотр и модификация содержимого регистров МП;

СТ - старт;

КС - подсчет контрольной суммы;

ЗК - заполнение массива памяти константой;

ПМ - перемещение массива памяти;

[1]—[1] - разделитель;

ВП - выполнить.

Первая часть часть клавиш предназначена для ввода параметров в шестнадцатиричной форме. В дальнейшем подразумевается, что все вводимые и выводимые данные имеют шестнадцатиричный формат. На них выгравированы символы: 0,1,2,3,4,5,6,7,8,9,A,B, C,D,E,F. Кроме этого на клавишах 4, 5, 6, 7, 8, 9 под цифрами выгравированы идентификаторы регистров микропроцессора.

PH - ст. байт счетчика команд;

PL-мл. байт счетчика команд;

SH-ст. байт указателя стека;

SL-мл. байт указателя стека;

H-регистр H;

L-регистр L.

Для идентификации остальных регистров МП используются клавиши:

A-регистр A;

B-регистр B;

C-регистр C;

D-регистр D;

E-регистр E;

F-регистр признаков.

Шестисегментный дисплей расположен в центре лицевой панели УМК над клавиатурой и предназначен для отображения данных в шестнадцатеричной форме (как вводимых, так и результата). При этом левые 4 сегмента используются для отображения данных типа байт. Светодиоды индикации состояния шин адреса, данных и управляющих сигналов МП расположены в центре лицевой панели УМК. Шина адреса состоит из 16 светодиодов, а шина данных - из 8 светодиодов (для отображения в двоичном коде). Над ними выгравированы надписи, соответственно, АДРЕС и ДАННЫЕ. Под каждым светодиодом этих шин выгравированы номера разрядов, которым они соответствуют. Под ними размещены светодиоды управляющих сигналов МП с соответствующими надписями - СОСТОЯНИЕ и названия сигналов.

Включение и запуск УМК.

Для включения УМК необходимо:

а) отжать (выключить) кнопку СЕТЬ, если она была включена;

б) подключить сетевой шнур к сети переменного тока напряжением

220 В и частотой 50 Гц;

в) нажать кнопку СЕТЬ, она должна остаться в нажатом состоянии.

При этом не должны загораться светодиоды защиты +5В,- 5В,+12В

Повторное включение УМК должно производиться не менее, чем через 20 сек после выключения. В противном случае сработает защита блока питания и загорятся соответствующие светодиоды защиты. В этом случае необходимо выключить УМК и дождаться, когда светодиоды защиты погаснут, и только после этого можно повторно включать УМК.

Для запуска УМК необходимо:

отжать кнопки РБ/ШГ и КМ/ЦК, тем самым перевести УМК в

автоматический режим работы;

нажать на кнопку СБ. При этом в левой части дисплея появится символ " - ", что означает, что УМК готов к работе и находится в режиме приёма команд.

Команды системного монитора УМК.

Для ввода команд в УМК необходимо:

а) на функциональной клавиатуре нажать клавишу, соответствующую выбранной команде. При этом экран дисплея погаснет;

б) ввести параметры команд (если их несколько, то между ними необходимо нажать клавишу РАЗДЕЛИТЕЛЬ). По мере ввода данных, они будут отображаться в соответствующей части дисплея. При вводе параметров необязательно вводить лидирующие нули, вместо 01 можно ввести 1

или вместо 0023 - 23. Длина параметра не ограничена, однако при вводе параметров, отображающих адреса существенными являются только 4 правые позиции, при вводе байтовых значений - только 2;

в) нажать клавишу ВП. Результат выполнения команды появится на дисплее. Если при вводе команды будет допущена ошибка, на экран дисплея будет выведен символ "?", и команда будет снята. Оператор должен повторить ввод.

Команда ПРОСМОТР И МОДИФИКАЦИЯ СОДЕРЖИМОГО ЯЧЕЙКИ ПАМЯТИ.

Эта команда используется для отображения или записи в память данных. Для выполнения этой команды необходимо:

а) нажать клавишу П;

б) ввести адрес ячейки памяти, например, 800Н;

в) нажать клавишу ВП. На дисплее в правой части появится содержимое заданной ячейки

800 XX

г) введите новое значение - 0;

800 0

д) нажмите клавишу РАЗДЕЛИТЕЛЬ, осуществить переход к

следующей ячейке памяти

801 XX

е) введите новое значение - 1

1

ж) действуя аналогично пунктам д), е) введите далее следующие данные: 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F;

з) нажмите клавишу ВП.

На этом выполнение данной команды ЗАПИСЬ В ПАМЯТЬ будет закончено. УМК перейдёт в режим ввода следующей команды.

Для проверки правильности выполнения этой команды необходимо выполнить эту же команду без модификации содержимого памяти, т.е. в режиме чтения:

а) нажать клавишу П;

б) ввести адрес ячейки памяти - 800Н;

в) нажать клавишу ВП. На дисплее должно появиться:

800 00;

г) нажмите клавишу РАЗДЕЛИТЕЛЬ

01;

д) и т.д. до появления на дисплее

80F 0F;

Таким образом, вы можете просмотреть содержимое ячеек 800Н

80FH. Оно должно совпадать с тем, что вы ввели раньше;

е) нажмите клавишу ВП.

Задание:

Начиная с адреса 81ОН введите последовательно в ячейки памяти и проверьте данные:

OFH, OEH, ODH, OCH, OBH, OAH, 9, 8, 7, 6, 5, 4, 3, 2, 1. Примечание: здесь и далее символ Н при обозначении данных означает шестнадцатиричный формат. При вводе данных символ Н вводить не надо.

По адресу 820Н введите ААН

“ 821Н “ 55

“ 822Н “ АА

“ 823Н “ 55

“ 824Н “ АА

“ 825Н “ 55

Заполните массив памяти 900Н - 90FH значением 0.

Запишите в ячейки:

920Н - 80Н 92АН - 80Н 940Н - 02Н

923Н - 40Н 92FH - 08Н 945Н - 01Н

927Н - 20Н 932Н - 04Н 94ЕН - 00Н

Команда ПРОСМОТР И МОДИФИКАЦИЯ СОДЕРЖИМОГО РЕГИСТРОВ.

Эта команда используется как для просмотра, так и модификации регистров микропроцессора.

Изменим содержимое регистров микропроцессора A и B:

а) нажмите клавишу РГ;

б) введите идентификатор регистра A - A (на клавиатуре ввода данных). На дисплее появится содержимое регистра A;

A - XX;

введите новое значение:

A - 0A;

г) нажмите клавишу РАЗДЕЛИТЕЛЬ;

д) введите идентификатор регистра B - B:

B - XX;

е) введите новое значение - 0B:

B - 0B;

ж) нажмите клавишу ВП.

Для проверки правильности выполнения команды выполните следующие действия:

а) нажмите клавишу РГ;

б) введите идентификатор регистра A - A, должно появиться на дисплее:

A - 0A;

в) нажмите клавишу РАЗДЕЛИТЕЛЬ;

г) введите идентификатор регистра B - B, должно появиться на дисплее:

B - 0B;

д) нажмите клавишу ВП.

Если содержимое регистров A и B будет отлично от OAH и OBH, значит при выполнении команды модификации регистров вы неправильно ввели новые значения. Выполните команду повторно.

Задание:

Установите значение регистров:

A - 00H PH - 07H
B - 01H PL - 08H
C - 02H SH - 0BH
D - 03H SL - 0AH
E - 04H F - FFH H - 05H L - 06H

Проверить правильность выполнения команды.

Команда СТАРТ ПРОГРАММЫ.

Эта команда используется для запуска и отладки программ пользователя. Для выполнения этой команды необходимо предварительно записать в память машинные коды программы, например, коды последовательности пустых команд NOP - 00H.

а) используя команду МОНИТОРА ПРОСМОТР И МОДИФИКАЦИЯ СОДЕРЖИМОГО ЯЧЕЙКИ ПАМЯТИ, запишите в ячейки памяти с адресами 800H - 80EH коды команды NOP - 00H;

б) в ячейку с адресом 80FN запишите FFH, код команды PST 7, выполняющей функцию программного прерывания для прекращения выполнения программы. Теперь выполним команду СТАРТ ПРОГРАММЫ. Для этого необходимо:

нажать клавишу СТ;

ввести стартовый адрес программы - 800H;

установить альтернативные точки останова;

нажать клавишу РАЗДЕЛИТЕЛЬ, на дисплее появятся символ

;

ввести адрес первой точки останова - 803H;

нажать клавишу РАЗДЕЛИТЕЛЬ, на дисплее появится символ

55.

;

ввести адрес альтернативной точки останова - 805H;

нажать клавишу ВП, осуществится передача управления на адрес 800Н. На дисплее появится адрес первой точки останова 803.

Продолжим выполнение программы, начиная с адреса первой точки останова, адреса 803Н, до адреса 805Н. Для этого:

нажмите клавишу СТ;

нажмите клавишу РАЗДЕЛИТЕЛЬ, это означает, что выполнение программы необходимо продолжить с текущего адреса;

введите адрес точки останова 805Н;

нажмите клавишу ВП. Выполнение программы будет продолжено.

На дисплее появится 805 - адрес точки останова. Теперь продолжим выполнение программы без указания точки останова. Для запуска программы с начала, адреса 800Н, без установки точек останова :

нажмите клавишу СТ;

введите стартовый адрес - 800Н;

нажмите клавишу ВП.

На дисплее появится адрес 810. Программа выполнена.

Задание:

Запишите в ячейки памяти (830Н - 83ЕН) - 00Н.

Запишите в ячейки памяти 83FH - FFH.

Выполните команду СТАРТ ПРОГРАММЫ с установкой двух точек останова 835Н и 83 АН.

Продолжите выполнение программы до адреса 83СН.

Продолжите выполнение программы без указания точек останова.

Запустите программу с адреса 830программу с адреса 830Н без указания точек останова.

Команда ЗАПОЛНЕНИЕ МАССИВА ПАМЯТИ
КОНСТАНТОЙ.

Эта команда используется для записи в массив памяти константы. Для выполнения этой команды:

а) нажмите клавишу ЗК;

б) введите начальный адрес массива - 800Н;

в) нажмите клавишу РАЗДЕЛИТЕЛЬ, экран дисплея погаснет;

г) введите конечный адрес массива 80FH;

д) нажмите клавишу РАЗДЕЛИТЕЛЬ, последний введенный
адрес останется на дисплее;

е) введите константу ААН;

ж) нажмите клавишу ВП.

Для проверки правильности выполнения команды заполнения содержимого ячейки памяти (см. п. 1.3.1).

Задание:

Заполните (и проверьте) массив памяти с адресом 840Н - 84FH данными - FFH.

Заполните (и проверьте) массив памяти с адресом 93 АН - 952Н данными 05Н.

Заполните (и проверьте) массив памяти с адресом 960Н - 97FH данными С7Н.

Команда ПОДСЧЁТ КОНТРОЛЬНОЙ СУММЫ.

Перед выполнением этой команды заполните массив памяти 840^84FH данными 0АН (контрольная сумма будет равна 96Н).

Для выполнения команды ПОДСЧЕТ КОНТРОЛЬНОЙ СУММЫ:

а) нажмите клавишу КС;

б) введите начальный адрес массива 840Н;

в) нажмите клавишу РАЗДЕЛИТЕЛЬ;

г) введите конечный адрес массива 84FH;

д) нажмите клавишу ВП. На дисплее появится значение контрольной суммы массива - 96.

Задание:

Заполните массив памяти (800Н - 845Н) данными 0IH и подсчитайте контрольную сумму.

Заполните массив памяти (852Н - 8FFH) данными С7Н и подсчитайте контрольную сумму.

Заполните массив памяти (800H - 9FFH) данными FFH и подсчитайте контрольную сумму.

Команда ПЕРЕМЕЩЕНИЕ МАССИВА ПАМЯТИ.

Эта команда используется для пересылки данных из одной области памяти в другую. Например, для перемещения кодов программы.

Перед выполнением этой команды предварительно заполните массив памяти 800H - 83FH данными 55H и подсчитайте контрольную сумму этого массива. Должно получиться 40H.

Для выполнения команды ПРЕМЕЩЕНИ Я МАССИВА ПАМЯТИ:

а) нажмите клавишу ПМ;

б) введите начальный адрес перемещаемого массива - 800H;

в) нажмите клавишу РАЗДЕЛИТЕЛЬ;

г) введите конечный адрес перемещаемого массива - 83FH;

д) нажмите клавишу РАЗДЕЛИТЕЛЬ;

е) введите начальный адрес массива, куда осуществляется перемещение - 900H;

ж) нажмите клавишу ВП.

Для проверки правильности выполнения команды перемещения подсчитайте контрольную сумму нового массива (900H - 93FH). Она должна совпадать с контрольной суммой перемещаемого массива (800H - 83F^ = 40H.

Задание:

Заполните массив памяти (850H - 8FFH) данными 03H. Подсчитайте контрольную сумму. Переместите этот массив в область с начальным адресом 950H. Проверьте правильность выполнения команды перемещения.

Переместите массив памяти (0 - 0FFH) в область с начальным адресом 800H. Проверьте правильность выполнения команды.

Переместите массив памяти (800H - 8FFH) в область с начальным адресом 880H. Проверьте правильность выполнения команды перемещения.

Примечание. Так как в задании 3 области памяти перекрываются, для сохранения данных используйте промежуточный буфер.

Контрольные вопросы

1. Как осуществляется включение и запуск УМК?
2. Какие команды МП вы знаете?
3. Как осуществляется просмотр и модификация содержимого ячеек памяти?
4. Как осуществляется просмотр и модификация содержимого регистров МП?
5. Как осуществляется старт программы?
6. Как заполняется массив памяти константой?
7. Как производится подсчет контрольной суммы?
8. Как производится перемещение массива памяти?

ПРАКТИЧЕСКАЯ РАБОТА №2. ИЗУЧЕНИЕ РАБОТЫ УМК В ПОШАГОВОМ РЕЖИМЕ

Цель работы: Освоение режимов работы УМК по машинным циклам и по командам (установка режимов и применение при выполнении конкретных последовательностей команд).

Теоретическая часть

Существуют два режима работы УМК по шагам:

- режим работы по машинным циклам;
- режим работы по командам.

Режим работы УМК по машинным циклам используется для изучения процесса выполнения команд в микропроцессоре.

Режим работы УМК по командам используется для изучения команд и отладки программ.

Выполнение работы

1. Режим работы УМК по машинным циклам.

Установка режима работы УМК по машинным циклам.

Для этого выполните следующие действия:

включите и приведите в рабочее состояние УМК (см. Лаб. Раб. №1);

нажмите клавишу РБ/ШГ (нажатое состояние);

нажмите клавишу КМ/ЦК (нажатое состояние).

Выполнение последовательности команд.

Рассмотрим пример выполнения следующих команд:

Адрес	Команда	Машинный код	Комментарий
800	NOP	00	; пустая команда
801	MOV A, B	78	; пересылка B A
802	ADD C	81	; A = A + C
803	RRC	0F	; циклический сдвиг на 1 вправо
804	ANI 0FH	E6 0F	; A = AND 0FH
806	PUSH H	E5	; запись в стек HL
807	LXI H, 900H	21 00 09	; загрузка HL=900H (адрес M)
80A	MOV M, A	77	; запись M=A по адресу HL = 900H
80B	MVI A, 0FFH	3E FF	; загрузка A = 0FFH
80D	SUB M	96	; A=A-M по адресу HL
80E	POP H	E1	; HL стек
80F	JMP 800H	C3 00 08	; переход на адрес 800H

Запишите в память, начиная с адреса 800H, коды указанной последовательности команд (используйте команду монитора ПРОСМОТР И МОДИФИКАЦИЯ СОДЕРЖИМОГО ЯЧЕЙКИ ПАМЯТИ).

Запустите программу с адреса 800H (используйте команду СТАРТ ПРОГРАММЫ).

СТ800ВП

На шине адреса появится адрес первой выполняемой команды 800, на шине данных - код выполняемой команды NOP - 00 (все светодиоды шины данных погашены). После нажатия клавиши ШГ микропроцессор перейдёт к выполнению следующей команды, т.к. первая команда NOP состоит из одного машинного цикла и для её выполнения достаточно выполнить один шаг. Таким образом, нажимая клавишу ШГ, можно проследить ход выполнения всей введённой последовательности команд. Результаты выполнения этого задания сведены в таблицу:

Команда	Шина адреса	Шина данных Код Данные	№ шага	Порядковый № машинного цикла	Примечание
NOP	800	00	0	1	чтение кода команды
MOV A,B	801	78	1	1	выполнение пред. команды чтение кода команды
ADD C	802	81	2	1	выполнение пред. команды чтение кода команды
RRC	803	0F	3	1	выполнение пред. команды чтение кода команды

ANI 0FH	804	E6	4	1	выполнение пред. команды чтение кода команды
	805	0F	5	2	чтение 2 байта команды
PUSH H	806	E5	6	1	выполнение пред. команды чтение кода команды
	BC9	12	7	2	запись H (12) по УС
	BC8	34	8	3	запись L (34) по УС-1
LXI H, 900H	807	21	9	1	выполнение пред. команды чтение кода команды
	808	00	10	2	чтение 2 байта команды
	809	09	11	3	чтение 3 байта команды
MOV M,A	80A	77	12	1	выполнение пред. команды чтение кода команды
	900	03	13	2	запись в ячейку памяти по адресу 900H содержимого регистра A (03)
MVI A, 0FFH	80B	3E	14	1	выполнение пред. команды чтение кода команды
	80C	FF	15	2	чтение 2 байта команды
SUB M	80D	96	16	1	выполнение пред. команды чтение кода команды
	900	03	17	2	чтение ячейки памяти по адресу 900H
POP H	80E	E1	18	1	выполнение пред. команды чтение кода команды
	BC8	34	19	2	запись в H из стека по УС = BC8H
	BC9	12	20	3	запись в L из стека по УС=УС+1=BC9H

JMP 800H	80F	C3	21	1	вып. пред. команды чтение кода команды
	810	00	22	2	чт. мл. байта адр. пер.
	811	08	23	3	чт. ст. байта адр. перехода

Последняя выполняемая команда передаёт управление на начало введённой последовательности команд. Чтобы закончить выполнение данного задания, нажмите клавишу СБРОС, УМК перейдёт в режим приёма команд.

Задание на практическую работу

Для следующей последовательности команд запишите машинные коды.

Адрес	Команда	Машинный код	Комментарий
800	LXI B, 940H		; загрузка BC = 940H
803	LXI D, 960H		; загрузка DE = 960H
806	MVI L, 10H		; загрузка L = 10H
808	LDAX B		; загрузка A=M по адресу BC
809	STAX D		; запись M=A по адресу DE
80A	INX B		; BC = BC+1
80B	INX D		; DE = DE+1
80C	DCR L		; L = L-1
80D	JMP 800H		; переход по адресу 800H

Выполните действия для этих команд.

Примечание: закончить выполнение последовательности команд командой JMP 800H (на шине адреса должен появиться опять адрес 800H).

2. Режим работы УМК по командам.

Установка режима работы УМК по командам.

Для этого отожмите клавишу КМ/ЦК.

Выполнение последовательности команд

Рассмотрим работу УМК в покомандном режиме на примере последовательности команд:

Адрес	Команда	Машинный код	Комментарий
800	LXI H, 900H	21 00 09	; загрузка HL=900H (адрес M)
803	MVI M, 0AAH	36 AA	; запись M=0AA по адресу HL
805	MVI A, 55H	3E 55	; загрузка A = 55H
807	ADD M	86	; A=A+M по адресу HL
808	STA 901H	32 01 09	; запись M=A по адресу 901H
80B	LXI D, 902H	11 02 09	; загрузка DE=902H
80E	LDAX D	1A	; загрузка A=M по адресу DE

80F	ORA M	B6	;A = A OR M по адресу HL
810	LXI B, 903H	01 03 09	; загрузка BC = 903H
813	STAX B	02	; запись M=A по адресу BC
814	JMP 800H	C3 00 08	; переход на адрес 800H

Результаты выполнения этого задания сведены в таблицу:

Команда	Шина адреса	Шина данных	№ шага	Примечание
LXI H, 900H	800	21	0	; чтение кода команды
MVI M, 0AAH	803	36	1	; чтение кода команды ; выполнение пред. команды
MVI A, 55H	805	3E	2	; чтение кода команды ; выполнение пред. команды
ADD M	807	86	3	; чтение кода команды ; выполнение пред. команды
STA 901H	808	32	5	; чтение кода команды ; выполнение пред. команды
LXI D, 902H	80B	11	6	; чтение кода команды ; выполнение пред. команды
LDAX D	80E	1A	7	; чтение кода команды ; выполнение пред. команды
ORA M	80F	B6	8	; чтение кода команды ; выполнение пред. команды
LXI B, 903H	811	01	9	; чтение кода команды ; выполнение пред. команды
STAX B	813	02	10	; чтение кода команды ; выполнение пред. команды
JMP 800H	814	C3	11	; чтение кода команды ; выполнение пред. команды

После выполнения последней команды (JMP 800H) управление будет передано вновь на начало программы. Для завершения выполнения задания нажмите кнопку СБРОС.

Задание на практическую работу

Запишите машинные коды для следующей последовательности команд:

Адрес	Команда	Машинный код	Комментарий
800	LXI H, 900H		; загрузка HL=900H (адрес)
803	LXI B, 920H		; загрузка BC = 920H (адрес)
806	LXI D, 940H		; загрузка DE = 940H (адрес)
809	LDAX B		; загрузка A=M по адресу BC
80A	SUB M		; A=A2-M по адресу H

80B	STA 950H		; запись M=A по адресу 950H
80E	LDAX D		; загрузка A=M по адресу DE
80F	ANA M		; A = A AND M по адресу HL
810	RAL		; сдвиг влево на 1 регистр A
811	STA 951H		; запись M=A по адресу 951H
814	LDA 950H		; загрузка A=M по адресу 950H
817	MOV L, A		; пересылка L^ A
818	LDA 951H		; загрузка A=M по адресу 951H
81B	MOV H, A		; пересылка H^ A
81C	SHLD 952H		; запись HL по адресу 952H-953H
81F	JMP 800H		; переход на адрес 800H

Выполните действия, для этой последовательности команд. Результаты сведите в таблицу.

Контрольные вопросы

1. Какие режимы работы предусмотрены для данного микропроцессорного комплекта?
2. Какие действия необходимо предпринять для установки режима работы по машинным циклам?
3. Для чего используется режим работы по командам?
4. Какой из режимов позволяет более детально изучить работу команд микропроцессора?

ПРАКТИЧЕСКАЯ РАБОТА №3. ИЗУЧЕНИЕ РЕГИСТРОВ МИКРОПРОЦЕССОРА.

Цель работы: Изучение видов регистров микропроцессора, освоение их использования при программировании (загрузка регистров различных типов, пересылка данных из одного регистра в другой).

Теоретическая часть

В микропроцессоре КР 580ВМ80А для программирования доступны следующие регистры:

- 16-разрядный счётчик команд (PC), который содержит адрес выполняемой команды;
- 16-разрядный регистр - указатель стека (SP), который определяет адрес специализированной области ОЗУ - стека;
- 8-разрядный регистр-накопитель (A), который используется для хранения и накопления результата в арифметических, логических операциях, а также операциях ввода-вывода и сдвига. Кроме того, он может быть использован в качестве регистра общего назначения для хранения данных;
- шесть 8-разрядных регистров общего назначения B, C, D, E, H, L;
- 8-разрядный регистр признаков PSW (F), который содержит биты условий:

C - перенос,
AC - вспомогательный перенос ,
S - знак,
Z - нуль,
P - чётность.

Эти биты устанавливаются в зависимости от результата операции при выполнении арифметических, логических команд, команд сдвига и сравнения. Распределение битов условий в байте признаков следующее:

7							0
S	Z	0	AC	0	P	1	C

Регистры общего назначения могут использоваться для манипуляции 16-разрядными данными. Для этого регистры объединяются в пары следующим образом: B - C, D - E, H - L. В каждой паре первый регистр используется для хранения старшего байта (например, B), а второй - для хранения младшего байта (например, C). В пару объединяются также регистры PSW и A.

Содержание работы

1. Команды загрузки регистров общего назначения.
2. Команды загрузки регистров 16-разрядными (двухбайтовыми) данными.
3. Команды загрузки регистра указателя стека.
4. Команды пересылки.
5. Команды загрузки счётчика команд.
6. Просмотр регистра признаков.

Выполнение работы

1. Команды загрузки регистров общего пользования.

Общий вид команды:

MVI R, B

где R - идентификатор одного из регистров A, B, C, D, E, H, L;

B - непосредственный операнд, байтовое число.

Запишите в память, начиная с адреса 800H, следующую последовательность команд:

Адрес	Команда	Машинный код	Комментарий
800	MVI A, 0	3E 00	; загрузка регистра A = 00H
802	MVI B, 1	06 01	; загрузка регистра B = 01H
804	MVI C, 2	0E 02	; загрузка регистра C = 02H
806	MVI D, 3	16 03	; загрузка регистра D = 03H
808	MVI E, 4	1E 04	; загрузка регистра E = 04H
80A	MVI H, 5	26 05	; загрузка регистра H = 05H
80C	MVI L, 6	2E 06	; загрузка регистра L = 06H

Выполните эту последовательность, используя команду:

СТ800 80ЕВП (на дисплее появится адрес останова 80E).

Используя команду ПРОСМОТР И МОДИФИКАЦИЯ РЕГИСТРОВ, посмотрите содержимое регистров общего назначения A, B, C, D, E, H, L.

Значения регистров должны быть следующими:

A = 00 B = 01 C = 02 D = 03 E = 04 H = 05 L = 06

2. Команды манипуляции 16-разрядными данными.

Общий вид команды:

LXI R, <BHBL>

где R - идентификатор регистра B, D, H;

BH - старший байт 16-разрядного операнда;
BL - младший байт 16-разрядного операнда.
Запишите в память, начиная с адреса 800H, коды следующей последовательности команд:

Адрес	Команда	Машинный код	Комментарий
800	LXI B, 3132H	01 32 31	; загрузка пары регистров BC = 3132H
803	LXI D, 3334H	11 34 33	; загрузка пары регистров DE = 3334H
806	LXI H, 3536H	21 36 35	; загрузка пары регистров HL = 3536H

Примечание: в памяти располагается сначала младший байт операнда, затем - старший.
Выполните эту последовательность команд:
CT800 809ВП
Посмотрите содержимое регистров. Значения регистров должны быть следующими:
B = 31H C = 32H D = 33H E = 34H H = 35H L = 36H
Команды загрузки регистра указателя стека
Команда непосредственной загрузки регистра указателя стека имеет вид:
LXI SP, BHBL
где BHBL - значение операнда;
BH - старший байт;
BL - младший байт.
Запишите в память по адресу 800H коды команды:

Адрес	Команда	Машинный код	Комментарий
800	LXI SP, 0B10H	31 00 0B	; загрузка указателя стека SP=0B10H

Выполните эту команду:
CT800 803ВП
Посмотрите содержимое регистра указателя стека. Значение старшего байта должно быть SH = 0BH, значение младшего байта SL = 10H.
Команда косвенной загрузки регистра указателя стека имеет вид: SPHL
По этой команде в указатель стека загружается содержимое регистровой пары HL. Поэтому, чтобы в указатель стека загрузить, например, число 0B30H, его предварительно надо загрузить в регистровую пару HL.
Запишите в память по адресу 800H коды следующих команд:

Адрес	Команда	Машинный код	Комментарий
800	LXI H, 0B30H	21 30 0B	; загрузка HL = 0B30H
803	SPHL	F9	; загрузка SP = HL

Выполните эти команды:
CT800 804ВП
Посмотрите содержимое регистра указателя стека. Значение старшего байта должно быть SH = 0BH, значение младшего байта - SL = 30H.
Команды пересылки.
Общий вид команды:
MOV R1, R2
где R1 - идентификатор регистра получателя: A, B, C, D, E, H, L;
R2 - идентификатор регистра источника: A, B, C, D, E, H, L.
Запишите в память, начиная с адреса 800H, коды следующей последовательности команд:

Адрес	Команда	Машинный код	Комментарий
800	MVI A, FFH	3E FF	; загрузка регистра A = FF
802	MOV B, A	47	; пересылка B A
803	MOV C, B	48	; пересылка C B
804	MOV D, C	51	; пересылка D C
805	MOV E, D	5A	; пересылка E ^-D
806	MOV H, E	63	; пересылка H E
807	MOV L, H	6C	; пересылка L H

Выполните эту последовательность команд:
СТ800 808ВП
Просмотрите содержимое регистров. Их значения должны быть равны FFH.
Команда загрузки счётчика команд PCHL
По этой команде в счётчик команд записывается содержимое пары регистров HL. Таким образом, для того, чтобы загрузить в счётчик команд адрес 0900H, необходимо сначала это число загрузить в регистровую пару HL и затем выполнить команду PCHL.
Запишите в память по адресу 800H коды следующих команд:

Адрес	Команда	Машинный код	Комментарий
800	LXI H, 0900H	21 00 09	; загрузка HL = 0900H
803	PCHL	E9	; загрузка счётчика команд PC = HL. Переход на адрес 0900H

Задание на самостоятельную работу

Напишите и выполните программу загрузки регистров по следующим правилам:

1 вариант	2 вариант	3 вариант
B : = F0H	B : = 23H	B:= BBH
C : = 33H	C : = 45H	C: = CCH
D : = EEH	D : = 10H	D:= DDH
E : = AAH	E : = 62H	E : = EEH
H : = 00H	H : = A5H	H : = 12H
L : =19H	L : =97H	L : =34H
A : =FFH	A : =0EH	A:=AAH

Проверьте правильность выполнения программы.
Напишите и выполните программу загрузки регистровых пар:

1 вариант	2 вариант	3 вариант

BC := FFFFH DE := 0123H HL := 55AAH	BC := 789AH DE := F0E9H HL := 56DCH	BC := BBCCH DE := DDEEH HL := 1234H

Проверьте правильность работы программы.

Напишите и выполните программу загрузки регистра указателя стека:

а) командой непосредственной загрузки:

SP := 0820H

SP := 0840H SP := 0AFFH

б) используя команду косвенной загрузки:

SP := 0825H

SP := 089FH SP := 0A5AH

Для проверки результатов выполняйте программу покомандно с остановкой после каждой команды и смотрите значение регистра SP.

Напишите и выполните программу пересылки, предварительно загрузив регистры указанными значениями:

1 вариант	2 вариант	3 вариант
BA (значением 00) CL (22H) HB (EEH)	DH (55H) EC (81H) LD (FFH)	DH (BCH) EL (03H) AB (7AH)

Проверьте правильность выполнения этих программ.

Напишите и выполните программу перехода на адрес:

1 вариант	2 вариант	3 вариант
0860H 0810H 085FH	0923H 091F 0953	0A00H 0A21H 0A3FH

с адреса 800H.

Контрольные вопросы

1. Какие типы регистров используются в микропроцессоре?
2. Биты каких условий содержит регистр признаков?
3. Как размещается в памяти 16-разрядный операнд?
4. Какие команды используются для загрузки указателя стека и счётчика команд?
5. Какова структура команды пересылки?

ПРАКТИЧЕСКАЯ РАБОТА №4. ИЗУЧЕНИЕ МЕТОДОВ АДРЕСАЦИИ ПАМЯТИ.

Цель работы: изучение непосредственного и косвенного методов адресации к памяти. Освоение команд чтения и записи в память при использовании различных способов адресации.

Теоретический обзор

Память представляет собой последовательность ячеек размером в один байт. Каждая ячейка имеет свой адрес в диапазоне от 0 до 65535. Для удобства обычно используется шестнадцатеричное значение адреса, тогда диапазон адресации составляет 0000H - FFFFH.

В микропроцессорной системе применяется два метода адресации к памяти : непосредственный и косвенный.

При **непосредственной адресации** адрес ячейки памяти указывается в самой команде во втором и третьем байтах команды. В общем виде команда выглядит следующим образом:

КОП AHAL

где КОП - код операции (чтение или запись),

AHAL - адрес ячейки памяти,

AH - старший байт адреса,

AL - младший байт адреса.

В памяти такая команда будет размещена в следующем порядке:

КОП

AL

AH,

то есть после байта кода операции располагается сначала младший байт адреса, а затем - старший.

Косвенная адресация предполагает, что адрес ячейки памяти будет располагаться в регистровых парах HL, BC, DE. Для каждой конкретной команды работы с памятью закреплена своя регистровая пара. Таким образом, прежде чем

выполнить такую команду, необходимо сначала задать адрес в соответствующей регистровой паре. Например,

```
LXI H, 800H
MOV M, A     ; запись в память регистра A по адресу HL
или
LXI D, 900H
STAX D       ; запись в память регистра A по адресу DE
```

Выполнение работы

1. Команды непосредственной записи в память

Существуют две команды непосредственной записи в память:

STA AHAL	запись в память по непосредственному адресу AHAL содержимого регистра A.
SHLD AHAL	запись в память содержимого регистровой пары HL. Причём по адресу AHAL будет записано содержимое регистра L, а по адресу AHAL+1 будет записано содержимое регистра H.

Запишите в память, начиная с адреса 800H, коды следующих команд:

Адрес	Команда	Машинный код	Комментарий
800	STA 880H	32 80 08	; запись в память содержимого регистра A по адресу 880H
803	SHLD 890H	22 90 08	; запись в память содержимого регистра L
			по адресу 890H, содержимого регистра H
			по адресу 891H.

Посредством команды монитора ПРОСМОТР И МОДИФИКАЦИЯ РЕГИСТРОВ установите следующие значения регистров:

A = C7H, H = 55H, L = AAH.

Выполните введённые команды:

CT800 806ВП

Посмотрите содержимое ячеек памяти с адресами 880H, 890H, 891H. Их значения должны быть:

880H = C7H 890H = AAH 891H = 55H

Задание на практическую работу

Напишите и выполните программу записи данных в память из регистра А в соответствии с нижеприведённой таблицей. Для этого используйте команду загрузки регистра А и команду записи в память регистра А по непосредственному адресу.

Адрес	900	905	90C	912	916	91F
Данные	00	01	02	04	08	10

Проверьте правильность работы программы.

Напишите и выполните программу записи данных из регистровой пары HL в соответствии с нижеприведённой таблицей. Для этого используйте команду загрузки регистровой пары HL и команду записи в память регистровой пары HL по непосредственному адресу.

Адрес	920	921	925	936	92C	92D
Данные	31	32	4F	50	FD	01

Проверьте правильность работы программы.

Напишите и выполните программу записи данных в память в соответствии с таблицей:

Адрес	940	941	943	947	94A	94B	94F	951
Данные	00	01	03	07	0A	0B	0F	51

Проверьте правильность работы программы.

Аналогично командам непосредственной записи существуют две команды непосредственного чтения из памяти:

LDA AHAL	чтение памяти по непосредственному адресу AHAL в регистр А
LHLD AHAL	чтение памяти по непосредственному адресу AHAL в регистровую пару HL. При этом в регистр L будет записано содержимое ячейки с адресом AHAL, в регистр H - содержимое ячейки с адресом AHAL+1.

Запишите в память по адресу 800H коды следующих команд:

Адрес	Команда	Машинный код	Комментарий
800	LDA 880H	3A 80 08	; чтение в регистр А содержимого ячейки с адресом 880H
803	LHLD 890H	2A 90 08	; чтение в регистр L содержимого ячейки с адресом 890H, в регистр H содержимого ячейки с адресом 891H

Выполните эти команды: CT800 806ВП

Посмотрите содержимое регистров. Оно должно быть следующим: A = C7H

H = 55H L = AAH.

Задание на практическую работу

Напишите и выполните программу загрузки регистров B, C, D, E, H, L из памяти в соответствии с нижеприведённой таблицей. Используйте команды чтения памяти в регистр А по непосредственному адресу и команды пересылки.

Адрес	900	905	90C	912	916	91F
Регистр	B	C	D	E	H	L
Результат	00	01	02	04	08	10

Проверьте правильность работы программы.

Напишите и выполните программу загрузки регистров B, C, D, E, H, L из памяти в соответствии с нижеприведённой таблицей. Используйте команды чтения памяти в регистровую пару HL по непосредственному адресу и команды пересылки.

Адрес	920	921	925	926	92C	92D
Регистр	B	C	D	E	H	L
Результат	31	32	4F	50	1F	01

Проверьте правильность работы программы

Напишите и выполните программу перезаписи данных из одних ячеек памяти в другие в соответствии с таблицей:

Адрес исх. ячейки	940	941	943	947	94A	94B	94F	951
Адрес запис. ячейки	960	961	963	967	96A	94B	94F	971
Результат	00	01	03	07	0A	0B	0F	51

Проверьте правильность работы программы

Команды чтения/записи памяти при адресации через регистровую пару HL

Общий вид команды:

MOV M, R	запись в память содержимого регистра
MOV R, M	загрузка регистра из памяти

где R - регистр общего пользования: A, B, C, D, E, H, L.

Запишите в память, начиная с адреса 800H, коды следующей программы:

Адрес	Команда	Машинный код	Комментарий
800	MVI A, 0AAH	3E AA	; загрузка регистров
802	MVI B, 0BBH	06 BB	
804	MVI C, 0CCH	0E CC	
806	MVI D, 0DDH	16 DD	
808	MVI E, 0EEH	1E EE	
80A	LXI H, 900H	21 00 09	; загрузка HL = 900H (адрес M)
80D	MOV M, A	77	; запись M=A, по адресу HL
80E	LXI H, 901H	21 01 09	; и т. д.

811	MOV M, C	71	
812	LXI H, 902H	21 02 09	
815	MOV M, B	70	
816	LXI H, 903H	21 03 09	
819	MOV M, E	73	
81A	LXI H, 904H	21 04 09	
81D	MOV M, D	72	
81E	LXI H, 905H	21 05 09	
821	MOV M, H	74	
822	LXI H, 906H	21 06 09	
825	MOV M, L	75	

Выполните эту последовательность команд: СТ800 826ВП

Проверьте правильность выполнения программы. Значения ячеек памяти должны быть следующими: 900H = AAH 901H = CCH 902H = BBH 903H = EEH 904H = DDH 905H = 09H 906H = 06H.

Запишите в память коды следующей программы:

Адрес	Команда	Машинный код	Комментарий
800	LXI H, 900H	21 00 09	; загрузка HL = 900H (адрес M)
803	MOV E, M	5E	; чтение E = M по адресу HL
804	LXI H, 901H	21 01 09	; и т. д.
807	MOV D, M	56	
808	LXI H, 902H	21 02 09	
80B	MOV C, M	4E	
80C	LXI H, 903H	21 03 09	
80F	MOV B, M	46	
810	LXI H, 904H	21 04 09	
813	MOV A, M	7E	
814	LXI H, 905H	21 05 09	
817	MOV H, M	66	
818	LXI H, 906H	21 06 09	
81B	MOV L, M	6E	

Выполните эту последовательность команд: СТ800 81СВП

Проверьте содержимое регистров. Оно должно быть следующим: регистр A = DDH, регистр B = EEH регистр C = BBH, регистр D = CCH, регистр E = AAH, регистр H = 09H, регистр L = 06H.

Задание на практическую работу

Напишите и выполните программу записи в память содержимого регистров в соответствии с таблицей:

Регистр	Адрес ячейки памяти	Содержимое
A	920	FF
B	921	EF
C	922	DF
D	923	CF
E	924	BF
H	925	09
L	926	26

Напишите и выполните программу чтения содержимого памяти и заполните таблицу:

Регистр	Адрес ячейки памяти	Содержимое	Должно быть
B	920		FF
C	921		EF
D	922		DF
E	923		CF
A	924		BF
H	925		09
L	926		26

Команды чтения/записи при адресации через регистровые пары BC, DE

STAX B	запись содержимого регистра A в память, адрес в регистровой паре BC
STAX D	запись содержимого регистра A в память, адрес в регистровой паре DE
LDAX B	чтение содержимого памяти в регистр A, адрес в регистровой паре BC
LDAX D	чтение содержимого памяти в регистр A, адрес в регистровой паре DE

Запишите в память, начиная с адреса 800H, коды программы:

Адрес	Команда	Машинный код	Комментарий
800	LXI B, 900H	01 00 09	; загрузка BC = 900H
803	MVI A, 0FH	3E 0F	; загрузка A = 0FH
805	STAX B	02	; запись M=A по адресу BC
806	LXI D, 910H	11 10 09	; загрузка DE = 910H

809	MVI A, 0F0H	3E F0	; загрузка A = F0H
80B	STAX D	12	; запись M=A по адресу DE

Выполните эту последовательность команд: СТ800 80СВП

Посмотрите содержимое ячеек памяти 900H и 910H. Оно должно быть следующим: 900H = 0FH, 910H = F0H.

Запишите в память, начиная с адреса 800H, коды программы:

Адрес	Команда	Машинный код	Комментарий
800	LXI D, 900H	11 00 09	; загрузка DE = 900H
803	LDAX D	1A	; чтение A=M по адресу DE
804	MOV L, A	6F	; пересылка L^ A
805	LXI B, 910H	01 10 09	; загрузка BC = 910H
808	LDAX B	0A	; чтение A=M по адресу BC
809	MOV H, A	67	; пересылка H^ A

Выполните эту последовательность команд: СТ800 80АВП.

Посмотрите содержимое регистров H, L. Оно должно быть следующим: H = F0H, L = 0FH.

Задание на практическую работу

Напишите и выполните программу записи данных в две области памяти, используя для адресации регистровую пару BC и регистровую пару DE в соответствии с таблицей:

Адреса 1 области памяти	910	912	915	919	921	927
Адреса 2 области памяти	930	932	935	939	941	947
Данные	10	12	15	19	21	27

Проверьте правильность работы программы.

Напишите и выполните программу перезаписи данных из одной области памяти (адресуйте через BC) в другую область памяти (DE) в соответствии с таблицей:

Адреса исходной области памяти	930	932	935	939	941	947
Адреса записываемой области памяти	960	961	962	963	964	965
Результат	10	12	15	19	21	27

Проверьте правильность работы программы.

Контрольные вопросы

1. Какие методы адресации существуют в данной микропроцессорной системе?
2. Как размещается в памяти команда, использующая непосредственную адресацию?

3. Перечислите команды непосредственной записи в память и чтения из памяти.
4. Действие каких команд осуществляется через регистровые пары BC, DE?
5. Какие регистры используются в команде пересылки?

ПРАКТИЧЕСКАЯ РАБОТА №5. ИЗУЧЕНИЕ АРИФМЕТИЧЕСКИХ КОМАНД.

Цель работы: Изучение команд микропроцессора, выполняющих арифметические действия с операндами (сложение, вычитание, инкремент, декремент). Применение этих команд для вычисления арифметических выражений.

Ход работы

В микропроцессорной системе КР580ВМ80А предусмотрены следующие команды двоичной арифметики:

сложение 8-разрядной чисел;
сложение 16-разрядной чисел;
вычитание 8-разрядных чисел;
инкремент;
декремент.

Все арифметические операции с 8-разрядными операндами предполагают, что один из операндов размещается в регистре-аккумуляторе, а другой либо в регистре, либо в памяти (при этом адрес ячейки задаётся в регистровой паре HL), либо является непосредственным числом, заданным в самой команде. Вычитание производится всегда из регистра-аккумулятора. Результат арифметической операции записывается в аккумулятор. Кроме того, по результату арифметических операций сложения и вычитания устанавливаются биты признаков: C - переноса, Z- нуля, S - знака, P - чётности, AC - вспомогательного переноса.

Команды сложения 16-разрядных чисел, так называемые команды двойного сложения, предусматривают, что один из операндов находится в регистровой паре HL, а второй - либо в DE, либо в BC. Результат записывается в HL. Кроме того, по результату операции устанавливается, либо сбрасывается бит переноса - C.

Команды инкремента увеличивают содержимое регистров, ячейки памяти по адресу в HL, и регистровых пар на единицу. Команда инкремента регистра и памяти изменяет биты признаков: Z, S, P, AC. Инкремент регистровой пары не затрагивает биты признаков.

Команды декремента уменьшают содержимое регистров, ячейки памяти по адресу в HL, и регистровых пар на единицу. Биты признаков изменяются аналогично команде инкремента.

1. Команды сложения 8-разрядных чисел

D R AD	сложение с регистром: A, B, C, D, E, H, L;
D M AD	сложение с ячейкой памяти (адрес в HL);
I B AD	сложение с непосредственным числом, B - байт;
C R AD	сложение с регистром A, B, C, D, E, H, L плюс бит переноса C ;
C M AD	сложение с ячейкой памяти (адрес в HL) плюс бит переноса C ;
I B AC	сложение с непосредственным числом, B - байт, плюс бит переноса C;

Запишите в память, начиная с адреса 800H, коды программы, реализующей выражение:
A = A + B + M + 1

Адрес	Команда	Машинный код	Комментарий
800	ADD B	80	; A = A+B

801	LXI H, 900H	21 00 09	; загрузка HL = 900H (адрес M)
804	ADD M	86	; A = A + M
805	ADI 1	C6 01	; A = A + 1

Выполните программу, предварительно задавая исходные значения в соответствии с таблицей, и проверьте полученные результаты (результат операции в регистре A, биты условий в регистре F):

СТ800 807ВП

A (исх.)	00	00	00	F0	FF	5 5
B	00	02	10	0E	00	A A
M	00	03	45	00	00	F F
A (рез)	01	06	56	FF	00	F F
F	02	06	06	86	57	6 8

Запишите в память, начиная с адреса 800H, коды программы сложения 16-разрядных чисел, используя команды 8-разрядного сложения: HL = DE + BC

Адрес	Команда	Машинный код	Комментарий
800	MOV A, C	79	
801	ADD E	83	; сложение младших байтов, установка бита переноса в случае переполнения
802	MOV L, A	6F	; занесение младшего байта в L
803	MOV L, B	78	
804	ADC D	8A	; сложение старших байт с учетом переноса
805	MOV H, A	67	; занесение старшего байта в H

Выполните программу, предварительно задавая исходные значения в соответствии с таблицей. Проверьте полученные результаты. СТ800 806ВП

BC	01	00	C5	02	00	F0	37	81	09F	8	FFF F
DE	FE	00	F1	03	FF	0F	D9	72	121	8	001 0
HL	FF	00	B6	06	FF	FF	10	F4	1C0	0	000 0
F	46		06		86		82		3	0	7 5

Задание на практическую работу

Напишите и выполните программу, реализующую вычисление выражения C = D + E и заполните таблицу:

D	10	FF	C7	19	AA	5E
E	80	01	8	49	55	F0
C						
F						

Напишите и выполните программу сложения двух ячеек памяти по правилу M1 = M2 + M3 и заполните таблицу.

Адрес M1 = 900H

Адрес M2 = 901H

Адрес M3 = 902H

M2	0	FE	D5	22	61	19
M3	F0	02	C2	BB	95	33
M1						
F						

Напишите и выполните программу сложения HL = BC + E + 4000H и заполните таблицу. Исходные значения возьмите произвольно.

2. Команды вычитания 8 - разрядных чисел

SUB R	вычитание регистра: A, B, C, D, E, H, L;
SUB M	вычитание ячейки памяти (адрес HL);
SUB B	вычитание непосредственного числа, B - байт;
SBB R	вычитание регистра: A, B, C, D, E, H, L минус бит переноса C;
SBB M	вычитание ячейки памяти (адрес HL) минус бит переноса C;
SBI B	вычитание непосредственного числа минус бит переноса C.

Запишите в память, начиная с адреса 800H, коды программы, реализующей вычисление выражения A = A - B - M - 1:

Адрес	Команда	Машинный код	Комментарий
800	SUB B	90	; A = A - B
801	LXI H, 900H	21 00 09	; загрузка HL = 900H (адрес M)
804	SUB M	96	; A = A - M
805	SBI 1	DE 01	; A = A - 1

Выполните программу, предварительно задав исходные значения в соответствии с таблицей. Проверьте полученные результаты. СТ800 807ВП

A(исх.)	FF	00	01	25	00	05
B	01	FF	01	20	00	06
M	01	00	00	04	00	FF
A(рез)	FC	00	FF	00	FF	FF
F	96	56	87	56	87	87

Запишите в память, начиная с адреса 800H, коды программы вычитания 16-разрядных чисел: HL = DE - BC

Адрес	Команда	Машинный код	Комментарий
800	MOV A, E	7B	
801	SUB C	91	; вычитание младших байтов E -C
802	MOV L, A	6F	; если E < C, перенос = 1
803	MOV A, D	7A	
804	SBB B	98	; вычитание ст. байтов с учётом переноса: D - B - бит переноса
805	MOV H, A	67	

Выполните программу, предварительно задавая исходные значения в соответствии с таблицей. Проверьте полученные результаты. СТ800 806ВП

DE	0000	FFFF	0FF0	8000

BC	FFFF	0FF0	0001	7FFF
HL	0001	F00F	0FEF	0001
F	47	96	16	46

Задание на практическую работу

1. Напишите и выполните программу, реализующую вычисление выражения C = D - E - 10H и заполните таблицу. Исходные значения возьмите произвольно.

D	
E	
C	
F	

2. Напишите и выполните программу вычитания двух ячеек памяти M1 = M2 - M3 и заполните таблицу. Исходные значения возьмите произвольно.

Адрес M1 = 900H
Адрес M2 = 901H
Адрес M3 = 902H

M2	
M3	
M1	
F	

3. Напишите и выполните программу вычитания HL = BC - E - 0FFFH и заполните таблицу. Исходные значения возьмите произвольно.

B	
E	
H	
F	

5.3. Команды двойного сложения.

DAD H	сложение HL = HL + HL
DAD B	сложение HL = HL + BC
DAD D	сложение HL = HL + DE

Запишите в память, начиная с адреса 800H, коды программы, реализующей сложение: HL = BC + DE

Адрес	Команда	Машинный код	Комментарий
800	MOV H, B	60	; пересылка H B
801	MOV L, C	69	; пересылка L C
802	DAD D	19	; HL = HL + DE

Выполните программу, предварительно задавая исходные значения в соответствии с таблицей. Проверьте полученные результаты. СТ800 803ВП

C	B	000	0	FFF	7	00	80	AA	55	B9	EC	FF	FF		
E	D	FFF	7	000	8	00	80	55	AA	47	13	00	80		
L	H	FFF	7	FFF	F	00	00	FF	FF	FF	FF	FF	7F		

F	6 4	6 4	47	46	47	47

Запишите в память, начиная с адреса 800H, коды программы вычитания адреса элемента массива по его порядковому номеру и чтения элемента массива в регистр A (в регистре C находится номер элемента (0 - 256), в HL - базовый адрес массива).

Адрес	Команда	Машинный код	Комментарий
800	LXI H,900H	21 00 09	; загрузка в HL базового адреса массива
803	MVI B, 0	06 00	; загрузка B = 0 (старший байт смещения)
805	DAD B	09	; HL = HL+ BC
806	MOV A, M	7E	; чтение элемента массива в регистр A

Выполните эту программу, предварительно задавая номер элемента массива в регистре C. Заполните таблицу.

C	C	1	5	0F	2	9
H (адрес элемента массива)						
A						

Сравните полученные результаты (регистр A) с содержимым соответствующих ячеек памяти, используя команду монитора ПРОСМОТР И МОДИФИКАЦИЯ СОДЕРЖИМОГО ЯЧЕЙКИ ПАМЯТИ.

Задание на практическую работу

1. Напишите и выполните программу заполнения массива по заданному индексу элемента массива в соответствии с таблицей.

Базовый адрес массива	900					
Номер элемента	00	03		1	8	1F
Адрес элемента						
Содержимое элемента массива	00	01	02	03	04	05

Проверьте правильность работы программы.

Напишите и выполните программу перезаписи содержимого массива 1, заданного в первом пункте, в массив 2 в соответствии с таблицей:

Базовый адрес массива 2	10					
Номер элемента массива 2	2	4				
Адрес элемента массива 2						
Содержимое элемента массива 2						

Команды инкремента

INR R	увеличение на 1 содержимого регистра: A, B, C, D, E, H, L;
INR M	увеличение на 1 содержимого ячейки памяти, адрес M в HL;

Запишите в память по адресу 800H код команды:

Адрес	Команда	Машинный код	Комментарий

800	INR E	1C	; E = E+1

Выполните данную команду для следующих исходных значений регистра E и проверьте полученные результаты: СТ800 801ВП

E (исх.)	00	0F	F0	FF
E (рез.)	01	10	F1	00
F	02	12	82	56

Запишите в память, начиная с адреса 800H, коды команд:

Адрес	Команда	Машинныйкод	Комментарий
800	LXI H,900H	21 00 09	; загрузка HL = 900H (адрес M)
803	INR M	34	; M = M+ 1

Выполните данную последовательность команд для следующих исходных значений содержимого ячейки памяти. Проверьте полученные результаты. СТ800 804ВП

M (исх.)	00	0F	F0	FF
M (рез.)	01	10	F1	00
F	02	12	82	56

Запишите в память по адресу 800H код команды:

Адрес	Команда	Машинный код	Комментарий
800	INX D	13	; DE = DE + 1

Выполните команду для следующих исходных значений пары регистров DE. Проверьте полученные результаты.

E (исх.)	0000	0F0F	00FF	FFFF
E (рез.)	0001	0F10	0100	0000
F	56	056	56	56

Так как команда инкремента пары регистров не затрагивает биты признаков, значение регистра признаков остаётся равным значению регистра признаков в последнем задании.

Задание на практическую работу

Напишите и выполните программу заполнения массива памяти (900H - 904H), соответственно, данными (00 - 04), используя команды инкремента пары регистров и регистра.

Команды декремента

DCR R	уменьшение на 1 содержимого регистра A, B, C, D, E, H, L;
DCR M	уменьшение на 1 содержимого ячейки памяти, адрес M в HL;
DCX R	уменьшение на 1 содержимого пары регистров BC, DE, HL, SP. В команде указывается старшего регистра, например, DCX B

Запишите в память по адресу 800H код команды:

Адрес	Команда	Машинный код	Комментарий
800	DCR C	0D	; C = C-1

Выполните эту команду для следующих значений регистра С. Проверьте полученные результаты. СТ800 801ВП

С (исх.)	00	01	10	11
С (рез)	FF	00	0F	10
F	86	56	06	12

Запишите в память, начиная с адреса 800Н, коды команд:

Адрес	Команда	Машинный код	Комментарий
800	LXI H, 900H	21 00 09	; загрузка HL = 900H (адрес М)
803	DCR M	35	; M = M-1

Выполните эти команды для следующих исходных значений содержимого ячейки памяти. Проверьте полученные результаты. СТ800 804ВП

М (исх.)	00	01	10	11
М (рез.)	FF	00	0F	10
F	86	56	06	12

Запишите в память по адресу 800Н код команды:

Адрес	Команда	Машинный код	Комментарий
800	DCX H	2B	; HL = HL - 1

Выполните команду для следующих исходных значений пары регистров HL. Проверьте полученные результаты. СТ800 801ВП

HL (исх.)	0000	1000	FFFF	0001
HL (рез)	FFFF	0FFF	FFFE	0000
F	12	12	12	12

Команда декремента пары регистров не затрагивает биты признаков.

Задание на практическую работу

1. Напишите и выполните программу заполнения массива памяти (90FH- 90AH) данными, соответственно, (0FH - 0AH), используя команды декремента пары регистров и регистра.

Контрольные вопросы

1. Где хранится результат арифметической операции?
2. Как размещаются операнды и результат при выполнении команды двойного сложения?
3. Как изменяется содержимое PSW при выполнении операции декремента содержимого ячейки, регистра, регистровой пары?
4. Перечислите команды сложения и вычитания 8-разрядных чисел.
5. Какие команды используются для изменения на единицу значения операнда?

ПРАКТИЧЕСКАЯ РАБОТА №6. ИЗУЧЕНИЕ КОМАНД ВВОДА-ВЫВОДА.

Цель работы: освоить принципы взаимодействия микропроцессора с внешними устройствами, основанные на командах ввода и вывода данных. В качестве примера изучить работу программы-драйвера, осуществляющей приём данных с клавиатуры и вывод их на дисплей.

Теоретические основы

Для организации взаимодействия микропроцессора с внешними устройствами предусмотрены две команды: команда ввода и команда вывода 8разрядных данных.

По **команде ввода** осуществляется считывание данных из внешнего порта, адрес которого задан во втором байте команды, в регистр-аккумулятор микропроцессора. Например:

IN PORT1,

где PORT1 - адрес порта ввода.

По **команде вывода** осуществляется вывод данных из регистра- аккумулятора микропроцессора во внешний порт, адрес которого указывается во втором байте команды. Например:

OUT PORT2,

где PORT2 - адрес порта вывода.

Используя эти команды, пользователь имеет возможность создавать программы-драйверы, обеспечивающие взаимодействие микропроцессора с периферийной техникой.

Выполнение работы

1. Программа-драйвер.

Исходные данные приведены в следующей таблице:

PORT A	порт номера индикатора содержит: 20H - 0 индикатор (самый правый) 10H- 1 индикатор 08H- 2 индикатор 04H- 3 индикатор 02H- 4 индикатор 01H- 5 индикатор Адрес: PORTA = F8H
PORT B	порт вывода данных - выводится код отображаемого символа. Адрес: PORTB = F9H
PORT C	порт состояния клавиатуры. Адрес: PORTC = FAH

Для отображения символов на дисплее необходимо сначала вывести в PORTA номер соответствующего индикатора, а затем в PORTB вывести код символа. Для того, чтобы отображаемые символы постоянно присутствовали на индикаторах дисплея, его надо постоянно регенерировать посредством вывода соответствующих данных в PORTA и PORTB. Для этой цели используется буфер регенерации, в котором хранятся коды отображаемых символов. Для того, чтобы погасить индикатор, в PORTA надо вывести номер этого индикатора, а в PORTB - 0. При нажатии на клавишу в порту состояния устанавливаются соответствующие биты, говорящие о том, что есть данные для ввода.

Запишите в память коды программы ввода данных с клавиатуры и отображения их на экране дисплея:

Адрес	Команда	Машинный код	Комментарий
800	LXI H, 8AEH	21 AE 08	;обнулить буфер регенерации
803	MVI C, 6	0E 06	
805	MVI M, 0	36	
807	INX H	23	
808	DCR C	0D	
809	JNZ 805	C2 05 08	

80C	LXI H, 08AEH	21 AE 08	;загр. HL (адр. буфера регенерации дисплея)
80F	MVI B, 20H	06 20	;загрузка B (номер мл. индикатора)
811	MOV A, B	78	
812	OUT PORTA	D3 F8	; вывод номера индикатора
814	MOV A, M	7E	; чтение A = M (код выводимого символа)
815	OUT PORTB	D3 F9	; вывод символа на дисплей
817	IN PORTÑ	DB F9	; поймать момент
819	ANI 74H	E6 74	; нажатия клавиши
81B	CPI 74H	FE 74	
81D	MVI A, 0	3E 00	; сбросить выведенный
81F	OUT PORT B	D3 F9	; символ
821	JNZ 82EH	C2 2E 08	; переход, если клавиша нажата
824	INX H	23	;HL=HL+1 (адрес след. выводимого символа)
825	MOV A, B	78	; вычисление номера след. индикатора
826	RRC	0F	
827	MOV B, A	47	
828	JNC 811H	D2 11 08	; цикл регенерации экрана и проверки ввода ;символа, след. индикатор
82B	JMP 80CH	C3 0C 08	; перейти к след. индикатору
82E	CALL 865H	CD 65 08	; вызов подпрограммы задержки (борьба с дребезгом)
831	IN PORTC	DB FA	; чтение введённых данных
833	MOV C, A	4F	
834	IN PORTC	DB FA	; поймать момент
836	ANI 74H	E6 74	; отпускания клавиши
838	CPI 74H	FE 74	
83A	JNZ 834H	C2 34 08	
83D	CALL 865H	CD 65 08	; вызов подпрограммы задержки
840	CALL 86FH	CD 6F 08	; вызов п/п преобразования кода клавиши в ;код ASCII. Результат возвращается в A
843	LXI H, 89EH	21 9E 08	;преобразование кода ASCII принятого ;символа в код вывода этого символа
846	CPI A	FE 41	; HL - адрес таблицы кодов символов
848	JC 84DH	DA 4D 08	; A - код ASCII символа - индекс в таблице ;кодов символов
84B	SUI 7	D6 07	
84D	ANI 0FH	E6 0F	
84F	MOV E, A	5F	
850	MVI D, 0	16 00	
852	DAD D	19	
853	MOV E, M	5E	; E - код выводимого символа
854	LXI H, 8AEH	21 AE 08	; записать в буфер вывода
857	MVI D, 6	16 06 08	; сдвиг влево буфера вывода
859	MOV A, M	7E	
85A	MOV M, E	73	
85B	MOV E, A	5F	
85C	INX H	23	
85D	DCR D	15	
85E	JNZ 859H	C2 59 08	

861	JMP 80CH	C3 0C 08	; продолжать
864	NOP	00	
865	LXI D, 1000H	11 00 10	; подпрограмма задержки на 10 мс
868	DCX D	1B	; DE = DE - 1
869	MOV A, E	7B	
86A	ORA D	B2	
86B	JNZ 868H	C2 68 08	
86E	RET	C9	
86F	MOV A, C	79	; преобразование
870	ANI 10H	E6 10	;кода принятого
872	JNZ 877H	C2 77 08	;символа с информационной
875	MVI C, 0	0E 00	;клавиатуры в код ASCII,
877	MOV A, C9EH	79	;а коды функциональных
878	ANI 64H	E6 64	;клавиш, соответственно,
87A	RAR	1F	;в коды 0 -7
87B	RAR	1F	
87C	RAR	1F	
87D	MOV C, A	4F	
87E	MOV A, B	78	
87F	ANI 3	E6 03	
881	JNZ 897H	C2 97 08	
884	MOV A, B	78	
885	RAR	1F	
886	RAR	1F	
887	RAR	1F	
888	CPI 4	FE 04	
88A	JNZ 88EH	C2 8E 08	
88D	DCR A	3D	
88E	ADD C	81	
88F	ORI 0	F6 30	
891	CPI 3AH	FE 3A	
893	RC	D8	
894	ADI 7	C6 07	
896	RET	C9	
897	DCR A	3D	
898	MOV B, A	47	
899	MOV A, C	79	
89A	RRC	0F	
89B	ADD B	80	
89C	RET	C9	
89D	NOP	00	
89E	DB 3FH, 6H, 5BH, 4FH	; таблица кодов вывода символов	
8A2	DB 66H, 6DH, 7DH, 7H		
8A6	DB 7FH, 6FH, 77H, 7CH		
8AA	DB 39H, 5EH, 79H, 71H		

8AE	; буфер регенерации	

Подсчитайте контрольную сумму кодов введённой программы (800H - 80DH). Она должна быть равна 44H.

Запустите программу: CT800ВП

При нажатии на клавишу клавиатуры данных (правая часть кнопок), на самом правом индикаторе дисплея будет отображаться введённый символ. Ранее введённый символ переместится на следующий индикатор (влево).

Для выхода из программы нажмите кнопку ПРЕРЫВАНИЕ.

Контрольные вопросы

1. Для чего служит регистр-аккумулятор процессора?
2. С какими внешними устройствами может взаимодействовать микропроцессор?
3. Какие порты и с какой целью используются в программе?
4. Для чего предназначен буфер регенерации?

ПРАКТИЧЕСКАЯ РАБОТА №7. ИЗУЧЕНИЕ КОМАНД МАНИПУЛЯЦИИ СТЕКОМ. ВЫЗОВ ПОДПРОГРАММЫ И ВОЗВРАТ ИЗ НЕЁ.

Цель работы: освоение основных команд работы со стеком (запись в стек, восстановление из стека, обмен со стеком). Изучение использования стека при операциях с подпрограммами (вызов подпрограммы, возврат из подпрограммы).

Теоретические основы

Стек - это специальная область ОЗУ, используемая для сохранения и восстановления данных, а также адресов возврата при вызове подпрограммы. Нижняя граница области стека определяется 16-разрядным регистром- указателем стека УС. В микропроцессорной системе предусмотрены 3 типа операций со стеком:

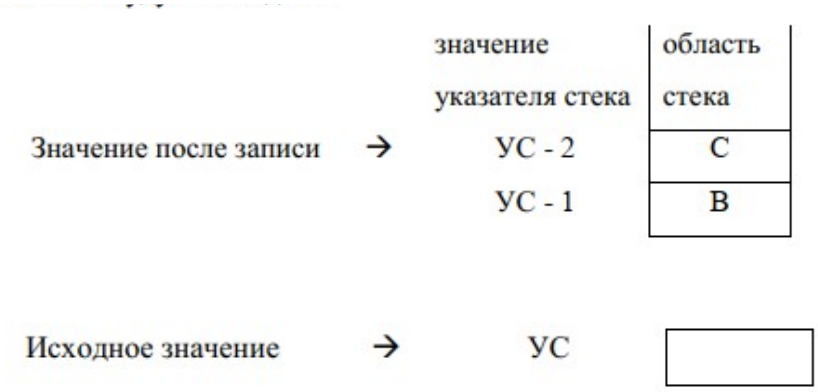
- запись в стек;
- восстановление из стека;
- обмен со стеком.

Запись в стек осуществляется при сохранении содержимого пар регистров (BC, DE, HL, PSWA), а также при вызове подпрограммы (при этом в стек помещается адрес возврата).

Запись в стек производится следующим образом:

1) из указателя стека вычитается 1;
2) по адресу, содержащемуся в указателе стека, записывается старший байт (старший регистр или старший байт адреса);
3) из указателя стека вычитается 1;
4) записывается младший байт.

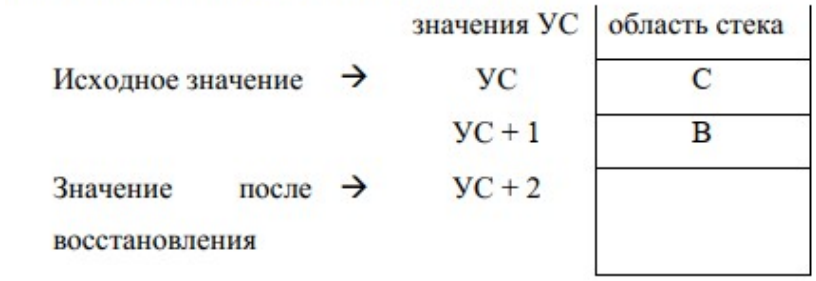
Например, при сохранении содержимого пар регистров BC указатель стека и сам стек будут выглядеть:



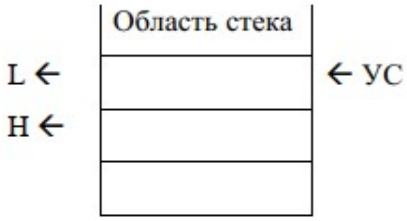
Восстановление из стека содержимого пар регистров и возврат из подпрограммы осуществляются в обратном порядке:

1) по адресу, содержащемуся в указателе стека, считывается младший байт;
2) к указателю стека прибавляется 1;
3) считывается содержимое старшего байта;
4) к указателю стека прибавляется 1.

При восстановлении из стека содержимого пары регистров BC значения указателя стека и область стека будут иметь вид:



При **обмене со стеком** осуществляется обмен содержимого верхнего элемента стека и пары регистров HL. Значение указателя стека при этом не изменяется.



Восстановление содержимого пар регистров из стека должно производиться в обратном порядке записи. Так, например, если в стек была произведена запись в последовательности BC - DE - HL, то восстановление должно производиться в обратном порядке: HL - DE - BC . Таким образом, стек работает по принципу “первым пришёл, последним ушёл”.

При **вызове подпрограммы** в стек записывается адрес команды, следующей за командой вызова подпрограммы (аналогично операции записи в стек). Затем, при возврате из подпрограммы, из стека в счётчик команд PC записывается адрес возврата. Необходимо следить за тем, чтобы непосредственно перед выполнением команды возврата из подпрограммы верхним элементом стека являлось значение адреса возврата, то есть, чтобы между командами вызова подпрограммы и возврата из неё не было загрузки стека без восстановления.

Порядок выполнения работы

1. Команды записи в стек, восстановления и обмена со стеком.

В таблице приведены команды работы со стеком - записи, восстановления, обмена.

PUSH B	запись в стек регистровой пары BC
PUSH D	DE
PUSH H	HL
PUSH PSW	F, A;
POP B	восстановление регистр. пары BC
POP D	DE
POP H	HL
POP PSW	F, A;
XTHL	обмен содержимого верхнего элемента стека и HL

Запишите в память, начиная с адреса 800H, коды программы сохранения содержимого регистров BC, DE, HL, F, A:

Адрес	Команда	Машинный код	Комментарий
800	LXI SP, 920H	31 20 0B	; загрузка SP = 920H (область стека)
803	PUSH B	C5	; запись BC в стеке
804	PUSH D	D5	; запись DE в стеке
805	PUSH H	E5	; запись HL в стеке
806	PUSH PSW	F5	; запись F, A

Выполните программу и заполните таблицу: CT800 807ВП

BC	DE	HL	PSW A	УС

Измените содержимое регистров микропроцессора B, C, D, E, H, L, A, F.
Запишите программу восстановления регистров:

Адрес	Команда	Машинный код	Комментарий
800	POP PSW	F1	; восстановление F и A
801	POP H	E1	; восстановление регистр. пары HL
802	POP D	D1	; DE
803	POP B	C1	; BC

Выполните программу и заполните таблицу: CT800 804ВП

BC	DE	HL	F, A	УС

***Примечание*:** значения регистров B, C, D, E, H, L, A, F должны совпадать с соответствующими значениями таблицы из пункта 10.1.2. Значение указателя стека УС = 0920H.
Запишите в память программу пересылки данных из одного массива в другой, используя для адресации регистровую пару HL и команду обмена стеком:
Выполните программу. Подсчитайте контрольную сумму (КС) исходного массива (0 - 7FH) и КС второго массива (900H - 97FH), они должны совпадать. Заполните таблицу: CT800 817ВП.

HL адрес 1 массива	DE адрес 2 массива	C длина	КС 1 массива	КС 2 массива
0000	0900	80		

Задание на практическую работу

1. Напишите и выполните программу обмена регистровыми парами, используя команды записи в стек и восстановления из стека, следующим образом:

BC → DE

DE → HL

HL →BC

Проверьте результаты.

2. Напишите и выполните программу побайтного сложения двух массивов, без учёта переноса, и записи результатов в первый массив, используя для адресации обоих массивов регистровую пару HL.

Адреса массивов и значение указателя стека следующие:

1 массив	0 - FFH
2 массив	100H - 1FFH
УС	B20H

2. Команды вызова подпрограммы и возврата из неё

Команды работы с подпрограммами приведены в следующей таблице:

CALL ADR	вызов подпрограммы по адресу ADR, указанному во втором и третьем байтах команды, причём при записи кодов команды вызова в памяти сначала записывается младший байт адреса, а затем старший, по аналогии с записью кодов команд загрузки регистровых пар;
RET	возврат из подпрограммы.

Запишите в память программу, использующую в своём теле вызовы подпрограммы, осуществляющей сравнение HL и DE.

Адрес	Команда	Машинный код	Комментарий
800	LXI H, 900H	21 00 09	; загрузка HL (начальный адрес массива)
803	LXI D, 97FH	11 7F 09	; загрузка DE (конечный адрес массива)
806	LXI SP, 0B00H	31 00 0B	; загр.УС = B00H (нижняя граница стека)
809	MVI B, 0	06 00	; загрузка B = 0 (исходное значение КС)
80B	MOV A, B	78	; подсчёт КС массива памяти
80C	ADD M	86	
80D	MOV B, A	47	
80E	INX H	23	; HL = HL+1 (адрес следующей ячейки)
80F	CALL 816H	CD 16 08	; вызов подпрограммы

812	JNC 80BH	2 0B 08	
815	NOP		; пустая команда
816	MOV A, E	7B	; подпрограмма сравнения:
817	SUB L	95	; если HL > DE то бит переноса = 1,
818	MOV A, D	7A	; иначе бит переноса = 0
819	SBB H	9C	
81A	RET	C9	; возврат из п/п

Выполните программу поэтапно, устанавливая точки останова (ТО) в соответствии с приведённой ниже таблицей. Запишите в таблицу значения указателя стека в точках останова.

СТ800 ТОВП

	ТО1 = 80FH	ТО2 = 816H	ТО3 = 81AH	ТО4 = 812H	ТО5 = 815H
УС					

В точке останова ТО3 посмотрите содержимое ячеек с адресами УС, УС+1. В них должны находиться младший и старший байты адреса:

M (УС)	12
M (УС+1)	08

Запишите в память программу, содержащую вызов подпрограммы с передачей параметров через стек:

Адрес	Команда	Машинный код	Комментарий
800	LXI SP, 0B00H	31 00 0B	; загрузка УС = B00H
803	LXI H, 5678H	21 78 56	; загрузка HL = 5678H
803	LXI B, 0123H	01 23 01	; загрузка BC = 0123H
809	LXI D, 9ABCH	11 BC 9A	; загрузка DE = 9ABCH
80C	PUSH H	E5	; запись значений регистров в стек
80E	PUSH D	D5	
80F	CALL 813H	CD 13 08	; вызов подпрограммы (п/п)
812	NOP		
813	POP H	E1	; загрузка HL (адрес возврата из п/п)
814	POP D	D1	; восстановление DE
815	POP B	C1	; восстановление BC
816	XTHL	E3	; обмен стека и HL (в стеке адрес ;возврата, в HL - восстановл. значение)
817	RET	C9	; возврат из п/п

Выполните программу и проверьте значения регистров в соответствии с таблицей:

СТ800 812ВП

BC	DE	HL	УС

0123	9ABC	5678	0A00

Задание на практическую работу

1. Напишите и выполните программу, содержащую вызов подпрограммы, выполняющей сложение четырёх двухбайтовых величин. Исходные величины передаются в подпрограмму через стек, возврат результата в регистровой паре HL. Исходные значения двухбайтовых величин возьмите произвольно.

2. Напишите и выполните программу в соответствии со следующим алгоритмом:

1) заполнить массив (900H - 9FFH) константой EEH;
2) подсчитать контрольную сумму этого массива по модулю 256 (без учёта переноса);
3) заполнить массив ^00H - AFFH) константой 11H;
4) подсчитать контрольную сумму.

Подсчёт контрольной суммы оформите как подпрограмму, с передачей в качестве входных параметров адреса начала и конца массива, а в качестве выходного параметра - значение контрольной суммы массива.

Контрольные вопросы

1. Что из себя представляет и для чего используется указатель стека ?
2. Как указатель стека УС обозначается в программах, работающих со стеком ?
3. Пусть начальное значение УС - 1000H. Чему будет равен указатель стека после записи в стек трёх 16-разрядных величин ? Обоснуйте ответ.
4. Пусть в стеке содержится адрес команды, следующей за командой вызова подпрограммы. Значение УС при этом равно 1000H. Чему будет равен указатель стека после возврата из подпрограммы ?
5. Перечислите основные команды работы со стеком.
6. Опишите механизм использования стека при работе с подпрограммами.
7. Для чего в программе 10.2.3. после команды вызова подпрограммы стоит команда, не выполняющая никаких действий?

ПРАКТИЧЕСКАЯ РАБОТА №8. ИЗУЧЕНИЕ КОМАНД БЕЗУСЛОВНОГО И УСЛОВНОГО ПЕРЕХОДОВ.

Цель работы: изучение команд безусловного перехода и условных переходов по признакам Z (ноль), C (перенос), P (чётность), S (знак). Освоение использования этих команд для решения различных практических задач (реализация бесконечного цикла, работа с массивами, с содержимым ячеек памяти).

Теоретические основы.

В системе команд микропроцессора КР580ВМ80А предусмотрены команды изменения последовательности выполнения команд для организации циклов, обработки условий, передачи управления и т.д. . Существуют два типа команд перехода: безусловный и условный.

При выполнении команд безусловного перехода осуществляется передача управления по адресу, заданному во втором и третьем байтах команды, либо по адресу, заданному в регистровой паре.

Команды условного перехода выполняются в том случае, если установлен или сброшен соответствующий бит признака, в противном случае команда игнорируется и выполняется следующая за ней команда.

Существуют команды условного перехода для следующих битов признаков:

- бита нуля;
- бита переноса;
- бита знака;
- бита чётности.

Для каждого бита признака предусмотрены две команды перехода: переход по установленному биту признака (равен единице) и переход по сброшенному биту признака (равен нулю).

Соответствие выполнения команды и признаков приведены в таблице 1.

Таблица 1.

Признак	Ноль		Перенос		Чётность		Знак	
Команда	Z		C		P		S	
	1	0	1	0	1	0	1	0
JZ	да	-	-	-	-	-	-	-
JNZ	-	да	-	-	-	-	-	-
JC	-	-	да	-	-	-	-	-
JNC	-	-	-	да	-	-	-	-
JPE	-	-	-	-	да	-	-	-
JPO	-	-	-	-	-	да	-	-
JM	-	-	-	-	-	-	да	-
JP	-	-	-	-	-	-	-	да

Выполнение работы

1. Команды безусловного перехода

JMP ADR	безусловный переход по адресу, указанному во 2 и 3 байтах команды
PCHL	безусловный переход по адресу, заданному в HL

Запишите в память, начиная с адреса 800H, коды программы, реализирующей бесконечный цикл:

Адрес	Команда	Машинный код	Комментарий
800	NOP	00	; пустая команда
801	NOP	00	; пустая команда
802	NOP	00	; пустая команда
803	JMP 800	C3 00 08	; безусловный переход на начало программы

Выполните программу: СТ800ВП

Данная программа будет выполнять бесконечный цикл.

Нажмите кнопку ПРЕРЫВАНИЕ.

На дисплее отобразится адрес точки прерывания (в пределах от 800H до 803H).

Запишите в память, начиная с адреса 800H, коды программы, осуществляющей переход по адресу, записанному в массиве (адрес является элементом массива). Адрес выбирается в соответствии с заданным индексом массива. Массив адресов размещается в памяти, начиная с адреса 810H и содержит 5 элементов. Индекс задаётся в регистре C.

Адрес	Команда	Машинный код	Комментарий
800	LXI D, 810H	11 10 08	; загрузка DE (нач. адрес таблицы - база)
803	MOV L, C	69	; пересылка L C (индекс массива)
804	MOV H, 0	26 00 00	; загрузка HL (индекс элемента массива)
806	DAD H	29	; HL = HL*2 (смещение в таблице)
807	DAD D	19	; HL = база массива + смещение
808	MOV A, M	7E	; пересылка A M (мл. байт адреса перехода)
809	INX H	23	
80A	MOV H, M	66	; пересылка A M (ст. байт адреса перехода)
80B	MOV L, A	6F	; пересылка L A (мл. байт адреса перехода)
80C	PCHL	E9	; переход, загрузка PC = HL
810	DW 900H	00 09	; массив адресов перехода
812	DW 902H	02 09	
814	DW 904H	04 09	
816	DW 906H	06 09	
818	DW 908H	08 09	

Выполните программу поэтапно, предварительно задавая исходные значения и адрес точки останова (второй адрес в команде монитора СТАРТ ПРОГРАММЫ) в соответствии с таблицей, и заполните свободные места в ней. СТ800 ТОВП

C	00	01	02	03	04
T01 = 807H смещение (HL)					
T02 = 808H адрес элемента массива (HL)					
T03 = 80Ш адрес перехода (HL)					
T04	900	902	904	906	908

Задание на практическую работу

1. Напишите и выполните программу, реализирующую бесконечный цикл суммирования содержимого массива памяти с адресами (0-3FFH).

2. Напишите и выполните программу, осуществляющую переход по адресу, который вычисляется по формуле:

АП = БА + И*16,

где АП - адрес перехода;

БА - базовый адрес, БА = 920H;

И - индекс, задаётся в регистре E.

Устанавливая точки останова в выполняемой программе, заполните таблицу:

И=E	0	2	5	7	10

СМ =И*16					
АП					

2. Команды перехода по признаку Z - НОЛЬ.

JZ ADR	переход, если Z = 1;
JNZ ADR	переход, если Z = 0.

Запишите в память, начиная с адреса 800H, программу заполнения 10H ячеек памяти нулями:

Адрес	Команда	Машинный код	Комментарий
800	MVI C, 10H	0E 10	; загрузка C=10H (длина массива)
802	LXI H, 900H	21 00 09	; загрузка HL=900H (начальный адрес массива)
805	MVI M, 0	36	; загрузка M=0
807	INX H	23	; HL=HL+1 (следующий адрес)
808	DCR C	0D	; C=C-1 (длина массива)
809	JNZ 805H	C2 05 08	; переход, если<>0

Выполните программу: СТ800 80СВП
Проверьте результаты выполнения программы в соответствии с таблицей:

C	H	M(900H) - M(90F)
00	910	00

Запишите в память, начиная с адреса 800H, программу замены данных в массиве памяти (900H - 90FH), отличных от 00H, на 00H.

Адрес	Команда	Машинный код	Комментарий
800	MVI C, 1OH	0E 10	; загрузка C=10H (длина массива)
802	LXI H, 900H	21 00 09	; загрузка HL=900H (начальный адрес массива)
805	MOV A, M	7E	; чтение A ← M
806	CPI 0	FE 00	; проверка условия A=0
808	JZ 80DH	CA 0D 08	; переход, если A=0
80B	MVI M, 0	36 00	; если A <> 0, загрузка M=0
80C	INX H	23	; HL = HL+1 (следующий адрес)
Адрес	Команда	Машинный код	Комментарий
80E	DCR C	0D	; C=C-1 (длина массива -1)
80F	JNZ 805H	C2 05 08	; продолжать, если длина массива <>0

Выполните программу: СТ800 812ВП.
Проверьте результаты выполнения программы в соответствии с таблицей:

C	H	M(900H) - M(90FH)

00	910	00

Задание на практическую работу

1. Напишите и выполните программу заполнения массива памяти (900H-9FFH) следующим образом: в чётные ячейки (900H, 902H, 904H и т.д.) записать 55H, в нечётные (901H, 903H, 905H и т.д.) записать AAH.

2. Напишите и выполните программу подсчёта контрольной суммы массива памяти (900H-9FFH) по модулю 256, т.е. без учёта переполнения байта. Команды перехода по признаку C – ПЕРЕНОС

JC ADR	переход, если C=1
JNC ADR	переход, если C=0

3. Запишите в память, начиная с адреса 800H, программу подсчёта нулей в байте. Исходное значение задано в регистре C.

Адрес	Команда	Машин. код	Комментарий
800	MOV A, C	79	
801	MVI B, 8	06 08	; загрузка B=8 (количество разрядов в байте)
803	MVI E, 0	1E 00	; загр. E=0 (исходное значение кол.-ва нулей)
805	RAR	1F	; значение мл.бита в бит переноса и сдвиг вправо
806	JC 80AH	DA 0A 08	; если перенос =1, обойти инкрем. Счётчика нулей
809	INR E	1C	; E=E+1 (инкремент счётчика нулей)
Адрес	Команда	Машин. код	Комментарий
80A	DCR B	05	; B=B-1 (следующий разряд)
80B	JNZ 805H	C2 05 08	; переход на проверку следующего разряда байта

4. Выполните программу, предварительно задавая исходные значения в соответствии с таблицей. Проверьте результаты. СТ800 80ЕВП

C	00	DB	FF	03	9A	55
E	00	06	08	02	04	04

5. Запишите в память, начиная с адреса 800H, программу, осуществляющую сравнение двух 16-разрядных величин на больше или равно по следующему правилу: HL >= DE ?

Если HL<DE C=1, иначе C=0.

Адрес	Команда	Машинный код	Комментарий
800	MVI C, 0	0E 00	; загрузка C = 0, для случая HL >=DE
802	MOV A, L	7D	; сравнение младших байтов
803	SUB E	93	; если L<E, перенос установки в 1
804	MOV A, H	7C	

805	SBB D	9A	; сравнение старших байтов с учётом переноса
806	JNC 80BH	D2 0B 08	; если HL >=DE, перенос = 0, C=0
809	MVI C, 1	0E 01	; если HL < DE, перенос =1, C=1

6. Выполните программу, предварительно задавая исходные значения в соответствии с таблицей. Проверьте результаты. СТ800 80ВВП

H	0000	FFFF	8002	5A7F	3A55	0000
DE	0001	FFFF	7FFF	6080	C201	FFFF
C	01	00	00	01	01	01

7. Напишите и выполните программу подсчёта количества единиц и нулей в байтах массива памяти, начальный и конечный адреса которого задаются, соответственно, в парах регистров HL и DE.

HL (нач. адр.)	0000	0800	0000
DE (кон. Адр.)	03FF	0AFF	FFFF
кол. единиц			
кол. нулей			

Команды перехода по признаку P - ЧЁТНОСТЬ.

JPE ADR	переход, если P=1;
JPO ADR	переход, если P=0.

8. Запишите в память, начиная с адреса 800H, программу дополнения байта до чётности в старшем разряде. Исходное число находится в регистре C.

Адрес	Команда	Машинный код	Комментарий
800	MOV A, C	79	; пересылка A 0 (исходный байт)
801	ANI 7FH	E67F	; обнуление старшего разряда
803	ORA A	B7	; A OR A - установка бита чётности
804	JPO 809H	E2 09 08	; переход, если исходный байт чётный
807	ORI 80H	F6 80H	; дополнить до чётности
809	MOV C, A	4F	; результат

9. Выполните программу, задавая исходные значения в соответствии с таблицей. Проверьте результат.

C (исходное значение)	00	FF	B6	80	CD	75
C(результат)	00	FF	36	00	4D	F5

10. Напишите и выполните программу дополнения байта до нечётности и заполните таблицу:

C (исходное	00	FF	B6	80	CD	75

значение)						
C(результат)						

11. Напишите и выполните программу подсчёта в массиве памяти количества чётных и нечётных байт. Начальный и конечный адреса задаются парами регистров HL и DE соответственно. Заполните таблицу:

HL	0000	0800	0C00
DE	03FF	0AFF	FFFF
Кол-во чётных			
Кол-во нечётных			

Команды перехода по признаку S – ЗНАК

JM ADR	переход, если S = 1 (число <0)
JP ADR	переход, если S = 0 (число >=0)

Запишите в память, начиная с адреса 800H, программу выделения из массива (100H - 10FH) цифровых символов с кодами 30H...39H и записи их в другой массив (начальный адрес 900H).

Адрес	Команда	Машинный код	Комментарий
800	LXI H, 100H	21 00 01	; загрузка HL - нач. адрес исх. массива
803	MVI B, 0FH	06 0F	; загрузка B = 0FH (длина массива)
Адрес	Команда	Машинный код	Комментарий
805	LXI D, 900H	11 00 09	; загрузка DE (нач. адрес выходного массива)
808	MOV A, M	7E	; чтение A = M (эл-т массива). Проверить
			; условие 30H <= A <= 39H
809	SBI 30H	DE 30	; A = A-30H, если A<30H, S=1
80B	JM 805H	FA 05 08	; переход к следующему элементу
80D	SBI 0AH	DE 0A	; A>=30H, A = A - 0A, если A>=0, S=0
80F	JP 805H	F2 05 08	; переход к след. элементу
812	MOV A, M	7E	;30H <= A <= 39H, перепис. в массив вывода
813	STAX	12	
814	INX D	13	; адрес след. элемента массива вывода
815	INX H	23	; адрес след. элемента исходного массива
816	DCR B	05	; длина массива - 1 = 0 ?
817	JNZ 808H	C2 08 08	; если нет, продолжать.

Выполните программу. Проверьте правильность работы программы, просмотрев выходной массив (900H - 90FH). Значения элементов массива должны соответствовать условию:

30H <= элемент массива <= 30H

Задание на практическую работу

Напишите и выполните программу, формирующую из одного массива два в соответствии со следующими требованиями:

	Нач. адрес	Конеч. Адрес	Элемент массива
Исх. Массив	0	7FH	
1 массив	900H		00H <= эл.м.<=7FH
2 массив	980H		80H <= эл.м.<=FFH

и подсчитайте, соответственно, количество элементов в каждом массиве. Проверьте правильность работы программы, просмотрев соответствующее количество ячеек обоих массивов.

Контрольные вопросы

1. Перечислите признаки, используемые в командах условных переходов.
2. Что означает выражение “сброшенный бит признака” ?
3. Чему будет равен счётчик команд после выполнения операции JMP 920H ?
4. Каков будет результат работы программы из пункта 9.2.1., если команду JNZ заменить на JZ?
5. Какого знака должен быть регистр A, чтобы осуществился переход по команде JP 930H ?

ПРАКТИЧЕСКАЯ РАБОТА №9. ОРГАНИЗАЦИЯ ПРЕРЫВАНИЙ В МИКРОПРОЦЕССОРНОЙ СИСТЕМЕ

Цель работы: Изучение организации многоуровневой системы прерываний с применением программируемого контроллера КР580ВН59.

Теоретическая часть

Программируемый контроллер прерываний КР580ВН59 представляет собой законченное устройство, которое позволяет реализовывать 8-уровневую векторную систему прерываний с возможностью маскирования и динамического изменения дисциплины обслуживания. Для перехода к подпрограммам обслуживания прерываний контроллер формирует и подает на ШД процессора код команды CALL. Каскадированием БИС КР580ВН59 число обслуживаемых уровней прерывания может быть увеличено до 64. Контроллер может использоваться как для организации обмена информацией в режиме прерывания, так и для организации программно-управляемого обмена. В первом случае БИС КР580ВН59 на приоритетной основе формирует запрос на прерывание для МП и адрес подпрограммы обслуживания, во втором - процессор считыванием слова состояния контроллера определяет устройство с наивысшим приоритетом, готовое к обмену.

Структура программируемого контроллера прерываний КР580ВН59 показана на рис.1.

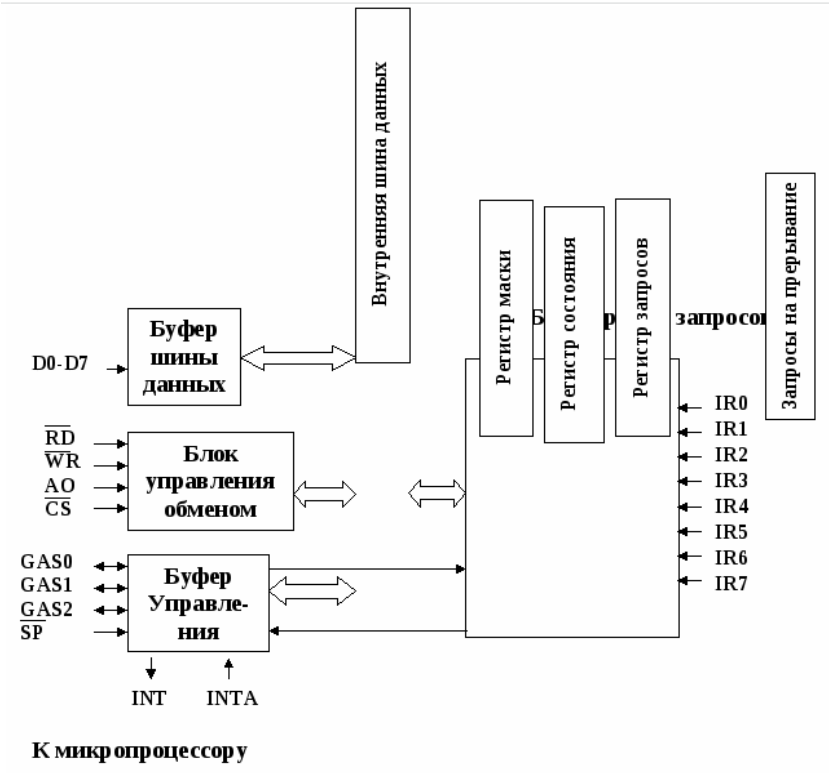


Рис.1 БИС КР580ВВ55

Микросхема КР580ВН59 размещена в пластиковом корпусе с 28 выводами, требует напряжение питания 5В и потребляет мощность 1Вт.

Контроллер позволяет реализовывать приоритетный режим и режим циклического приоритета обслуживания прерываний. При реализации простого приоритетного режима всем восьми входам запросов на прерывание присваиваются фиксированные приоритеты, причем наивысший приоритет присваивается входу IR0, низший IR7. В режиме циклического после окончания обслуживания любого устройства приоритеты входов контроллера циклически изменяются таким образом, что устройству, обслуженному последним, присваивается низший приоритет. Кроме того, при работе в режиме циклического приоритета низший приоритет может быть присвоен программным способом любому входу запроса.

Запросы на прерывания от внешних устройств подаются на входы IR0-IR7 контроллера и запоминаются в регистре запросов. Регистр состояния, каждый разряд которого соответствует одному из входов IR0-IR7, содержит все запросы прерывания, обслуживаемые в текущий момент. Регистр маски содержит единицы в разрядах, соответствующих маскируемым в настоящий момент входам запросов. Запросы на прерывания по любому входу могут быть поданы в потенциальной или импульсной форме. Однако каждый последующий запрос на прерывание воспринимается контроллером только после выполнения подпрограммы обслуживания текущего запроса по данному входу и сброса соответствующего разряда регистра состояния, что осуществляется программным способом (специальной командой). Установка в единицу того или иного разряда регистра маски блокирует передачу запроса на прерывание. Однако, если через некоторое время после подачи запроса на прерывание маска будет снята программой, запрос будет обслужен.

Программирование контроллера осуществляется двумя типами управляющих слов: управляющими словами инициализации (ICW) и операционными управляющими словами (OCW). Ввод команд в контроллер прерываний осуществляется микропроцессором командой OUT (вывод). Однако в системе может быть организовано обращение к контроллеру и как к ячейке памяти. Управляющие слова инициализации подаются перед началом работы контроллера. Они задают стартовые адреса подпрограмм обслуживания прерываний, расстояния между соседними стартовыми адресами и указывают, если необходимо, на наличие других контроллеров в системе. Операционные управляющие слова служат для оперативного изменения режимов обслуживания прерываний и могут подаваться в любое время в процессе работы контроллера.

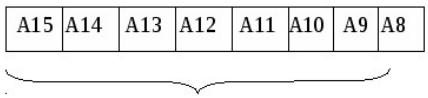
Процедура инициализации заключается в последовательном вводе в контроллер управляющих слов инициализации (ICW1 и ICW2). Если система содержит несколько контроллеров прерываний, для каждого из них вслед за ICW1 и ICW2 вводится третье управляющее слово ICW3, которое определяет соподчиненность контроллеров. Перед проведением инициализации прием прерываний микропроцессором должен быть запрещен с помощью команды DI (запретить прерывание).

Управление режимами работы контроллеров, изменение характера обслуживания, управление маскированием прерываний осуществляется операционными управляющими словами (OCW1, OCW2, OCW3).

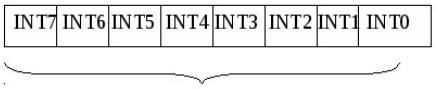
Таблица 1 Сигналы управления контроллера прерываний КР580ВН59

Обозначение	Наименование	Функциональное назначение и условия возникновения
Сигналы управления записью / считыванием		
RD	Запись (входной)	Запись в БИС управляющих слов
WR	Чтение (входной)	Считывание из БИС содержимого внутренних регистров
CS	Выбор микросхемы (входной)	Формируется при определенной комбинации сигналов на ША (1-15). Осуществляет подключение ШД (0-7) БИС к шине данных системы. При отсутствии сигнала выводы ШД (0-7) БИС находятся в состоянии с высоким выходным сопротивлением
AO	Адрес (входной)	Адресация регистров
Сигналы управления прерыванием		
IR0-IR7	Запросы на прерывание (входные)	Запросы на прерывание от внешних устройств
INT	Запрос на прерывание (выходной)	Запрос на прерывание, выдаваемый контроллером на МП
INTA	Разрешение прерывания	После поступления этого сигнала от МП контроллер осуществляет ввод в МП команды CALL
Сигналы управления каскадированием		
SP	Ведомый (входной)	Высокий уровень, если контроллер является ведущим, низкий уровень – если ведомым
GAS0-GAS2	Каскадирование	Сигналы GAS0-GAS2 являются выходными, если контроллер является ведущим (SP=1), и входными, если – подчиненным (SP=0). Значения сигналов GAS0-GAS2ведущего контроллера, подаваемые на входы GAS0-GAS2 подчиненных, указывают подчиненный контроллер, который формирует и выдает на МП адрес своей подпрограммы обслуживания

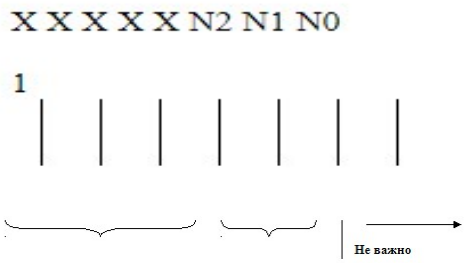
Управляющие слова инициализации КР580ВН59
ICW1
A0 D7 D6 D5 D4 D3 D2 D1 D0
A7 A6 A5 1 0 A2 A1 0
Число контроллеров в системе
1 – один
0 – более одного
Разность между стартовыми адресами
1 – 4 байта
0 – 8 байтов
Разряды A7 – A5 младшего байта адреса подпрограмм обработки прерываний
0 – более одного
ICW2
A0 D7 D6 D5 D4 D3 D2 D1 D0



Старший байт адреса подпрограмм обработки прерываний
0 – более одного
ICW3 (для ведущего)
A0 D7 D6 D5 D4 D3 D2 D1 D0



Подключение к входу INT починенного контроллера
1 – подключен
0 – не подключен
ICW3 (для ведомого)
A0 D7 D6 D5 D4 D3 D2 D1 D0



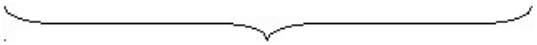
Номер входа ведущего контроллера, к которому подключен данный подчиненный.
Операционные управляющие слова КР580ВН59.
OCW1

A0 D7 D6 D5 D4 D3 D2 D1 D0

Маскирование входа

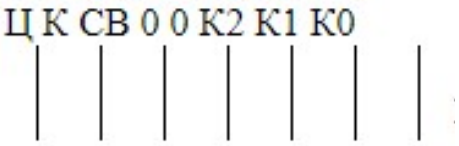
1 – маска установлена

0 – маска сброшена



OCW2

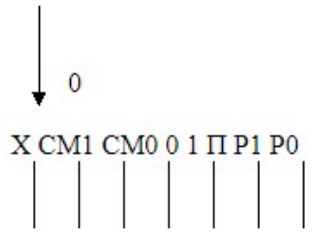
A0 D7 D6 D5 D4 D3 D2 D1 D0



1 – разряды К0 используются
0 – не используются
1 – сброс разряда с высшим приоритетом
0 – не действует
1 – режим циклического приоритета
0 – простой приоритетный режим
OCW3

A0 D7 D6 D5 D4 D3 D2 D1 D0

Безразлично



Считывание регистров
0 1 0 1
0 0 1 1

Режим маскирования
0 1 0 1
0 0 1 1
Сброс Установка
1 – разрешает в следующем цикле считать номер запроса с наивысшим приоритетом

Для задания простого приоритетного режима не требуется никаких дополнительных команд, кроме команд инициализации. После приема микропроцессором запроса на прерывание INT и выдачи сигнала разрешения прерываний INTA триггер разрешения прерываний микропроцессора сбрасывается, запрещая тем самым прием прерываний. Поэтому для организации многоуровневой системы прерываний необходимо выполнить программную установку этого триггера командой EI (разрешить прерывание) МП.

Об окончании выполнения подпрограммы обслуживания каждого запроса на прерывание МП должен сообщать контроллеру с помощью OCW2. При этом осуществляется сброс либо указанного в команде разряда регистра состояния контроллера, либо разряда, соответствующего обслуженному прерыванию. Для реализации многоуровневой системы прерываний каждая из подпрограмм обслуживания должна иметь определенную структуру.

ПРАКТИЧЕСКАЯ РАБОТА № 10. ОРГАНИЗАЦИЯ ПРЯМОГО ДОСТУПА К ПАМЯТИ.

Цель работы: изучение режимов работы и получение навыков программирования контроллера прямого доступа к памяти, закрепление полученных теоретических знаний по теме «Программируемый контроллер прямого доступа к памяти».

Теоретические сведения

Обслуживание внешних устройств по прерываниям может замедлить работу микро-ЭВМ, если главное назначение прерываний заключается в передаче большого количества данных. Для решения этой проблемы используется прямой доступ к памяти, суть которого в том, что в обмене данным и не принимает участия микропроцессор.

В практической работе исследуется работа микросхемы контроллера прямого доступа к памяти (КПДП) КР580ВТ57, управляющего работой четырех независимых каналов прямого доступа к памяти с учетом приоритетов внешних устройств. Подобно контроллеру прерываний эти приоритеты могут быть циклически и заменяемы.

Перед работой КПДП его необходимо запрограммировать (инициализировать). Инициализация контроллера прямого доступа к памяти осуществляется загрузкой исходных данных (начальные адреса областей ОЗУ, размер блоков данных и коды режимов) в адресуемые регистры и слова управления в регистр режима КПДП. Содержимое любого регистра и счетчика, кроме регистра режима, управляющего передачей данных из МП в ВУ и обратно, можно считать. После чего, можно осуществить передачу данных по выбранному каналу в нужном направлении.

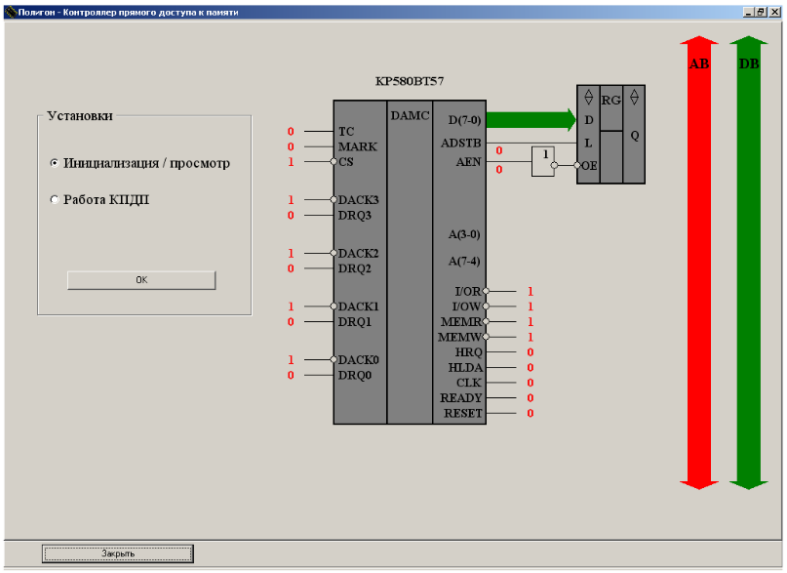
Практическая часть

Стартовое окно режима «Полигон» содержит условное изображение микросхемы и окно «Установки». Перед работой контроллер необходимо запрограммировать, выбрав опцию «Инициализация/просмотр».

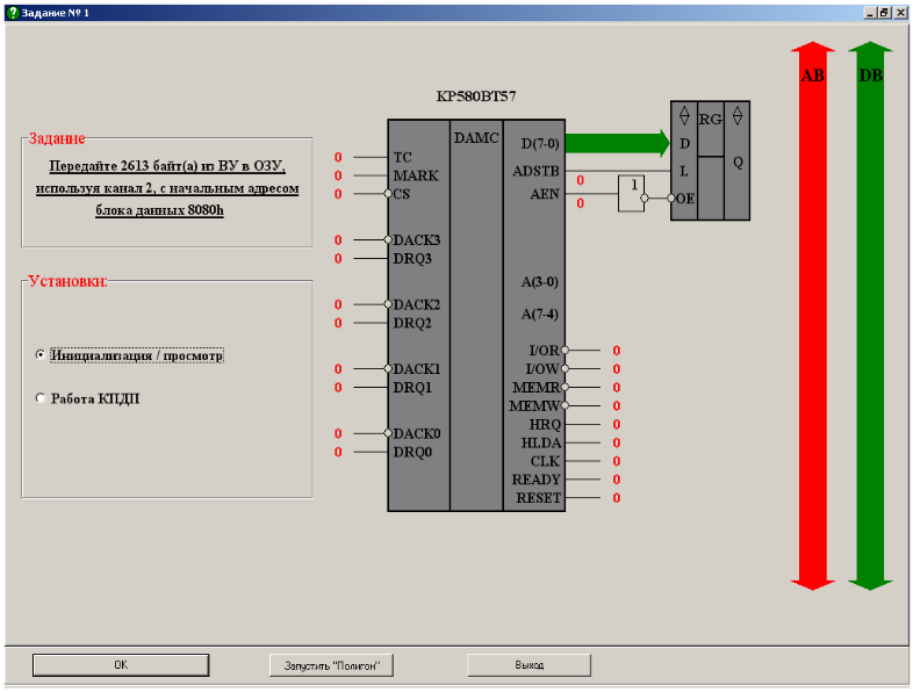
Инициализация осуществляется загрузкой исходных данных (начальные адреса областей ОЗУ, размер блоков данных и код ы режимов) в адресуемые регистры и регистр режима контроллера. Содержимое любого регистра и счетчика, кроме регистра режима, можно считать.

После программирования контроллера, выбрав опцию «Работа КПДП», можно осуществить передачу данных по выбранному каналу в нужном направлении.

После изучения работы микросхемы контроллера прямого доступа к памяти в «Полигоне» следует приступить к выполнению работы, предварительно зарегистрировавшись.



Для выполнения практической работы по исследованию контроллера прямого доступа к памяти необходимо выполнить ряд заданий. Примерный вариант задания представлен на следующем рисунке:



Задания в практической работе можно разделить на два типа:

1) задания, в которых необходимо настроить контроллер на указанный режим работы;

2) задания, в которых необходимо произвести передачу данных с указанными параметрами.

Для выполнения задания первого типа необходимо запрограммировать контроллер.

В заданиях второго типа после программирования необходимо передать данные.

Все задания выполняются в несколько этапов. Если на некотором этапе совершаются ошибочные действия, текущее задание считается невыполненным и выводится следующее задание.

Контрольные вопросы

1. В чем заключается функциональное назначение КПДП?
2. Каким образом осуществляется программирование контроллера КР580ВТ57?
3. Какую роль выполняют выводы микросхемы контроллера КР580ВТ57?
4. Какие преимущества дает прямой доступ к памяти перед работой по прерываниям?
5. Для чего нужен режим расширенной записи?

ПРАКТИЧЕСКАЯ РАБОТА №11. ИЗУЧЕНИЕ РАБОТЫ ОЗУ (ТЕСТИРОВАНИЕ И ОТЛАДКА).

Цель работы: изучить работу УОУ «Электроника-580»; перевести тест-программу проверки ОЗУ с машинного языка на Ассемблер, используя таблицу кодировки команд.

Теоретические сведения

Описание учебно-отладочного устройства «Электроника - 580»

Центральным элементом УОУ является однокристальный микропроцессор КР580ИК80А, обеспечивающий обработку информации и управление всеми остальными узлами УОУ.

Взаимодействие всех модулей УОУ осуществляется по трехшинной схеме, состоящей из 16-разрядной шины адреса, 8-разрядной двунаправленной шины данных и десяти линий шины управления.

Непосредственно к шине данных подключен только регистр слова состояния РСС, фиксирующий в начале каждого машинного цикла информацию о микропроцессоре.

В произвольный момент времени с МП взаимодействует только один из модулей, выбор которого осуществляется встроенным двухступенчатым дешифратором адреса, собранным на микросхемах К155ИД3 и К155ИД4. Он обрабатывает только 6 старших разрядов шины адреса.

Режим работы модулей (запись или чтение) ПЗУ, ОЗУ и ППИ определяется управляющими сигналами, поступающими с выхода ФУС. Эти сигналы формируются на основе информации о состоянии МП, которая фиксируется в РСС, а также с непосредственным использованием некоторых выходов МП.

Кроме сигналов записи и чтения ФУС формирует следующие сигналы:

строб слова состояния;

команда сброса МП RESET;

запрос прерывания INT.

Для осуществления диалога пользователя с УОУ предусмотрены клавиатура и цифровой дисплей. Клавиатура содержит 25 клавиш. С помощью клавиши RST формируется сигнал сброса МП, опрос остальных производится с помощью ППИ типа КР480ИК55. Верхний и правый ряды клавиш содержат 9 командных клавиш, остальные 16 клавиш служат для ввода в УОУ шестнадцатиричных цифр.

Цифровой дисплей служит для вывода информации в шестнадцатиричном коде о состоянии различных узлов УОУ и выполнен на восьми светодиодных семисегменных индикаторах. Дисплей управляется индикаторным узлом прямого доступа к памяти ПДП, который осуществляет выборку информации для дисплея из 3 ячеек ОЗУ и передачу ее на индикацию.

Значительную часть лицевой панели занимает таблица кодировки команд УОУ, позволяющая поставить в соответствие мнемоническому изображению команд ее двухразрядный шестнадцатиричный код (Н-код). Кроме того, выделено поле «Операции аккумулятора», в котором расшифровано значение некоторых команд.

На индикаторы АДРЕС, РЕГИСТР и ДАННЫЕ информация выводится с стилизованном виде. На лицевую панель выведена также индикация состояния флажков нуля и переноса МП при помощи светодиодов Z и C соответственно.

Клавиатура.

Клавиатура состоит из 9 командных клавиш и 16 клавиш данных. Назначение командных клавиш указано в таблице 1. Клавиши данных могут вводить в УОУ как цифровую, так и буквенную информацию. В последнем случае клавиши данных используются для задания имен регистров и регистровых пар МП:

клавиши A, B, C, D, E, 8/H, 9/L и F – для обозначения аккумулятора A, регистров B – L и регистра признака F соответственно;

клавиша I/P для обозначения указателя стека SP;

клавиша 2/T для обозначения содержимого вершины стека.

Таблица 1

Обозна-чение	Название	Назначение клавиши
RST	Сброс	Служит для формирования сигнала сброса УОУ.
ADDR	Адрес	Служит для перевода УОУ в режим задания адреса ячейки памяти.
MEM	Память	Служит для перевода УОУ в режим записи данных в ячейку памяти.
NEXT	Следующий	Служит для увеличения на единицу адреса индицируемой ячейки памяти или регистра МП.
CLR	Восстанов-ление	Служит для восстановления начального значения адреса или данных, если после их ввода не нажимались другие командные клавиши. При этом, если индицируется содержимое ячейки памяти и ранее была нажата клавиша MEM, то восстанавливается предыдущее значение данных. Если индицируется содержимое ячейки памяти, но клавиша MEM не была нажата, то клавиша CLR выведет на дисплей содержимое счетчика команд пользователя. При вводе адреса с клавиатуры клавиша CLR выводит на дисплей содержимое счетчика команд пользователя и не запрещает ввод нового адреса (можно не нажимать повторно клавишу ADDR). Нажатие на клавишу MEM восстановит предыдущий адрес ячейки памяти (но не выведет его на дисплей). Если высвечивается содержимое регистра МП, клавиша CLR вызовет отображение предыдущего содержимого регистра и его символа. При этом разрешено изменение содержимого регистра.
REG	Регистр	Предназначена для чтения и записи содержимого восьмиразрядного регистра МП. При нажатии на клавишу REG на дисплее появится текущий адрес команд пользователя (АДРЕС), символ регистра (РЕГИСТР) и содержимое выбранного регистра МП (ДАННЫЕ). Текущее значение регистра индицируется при выполнении шагов программы. Возврат к индикации содержимого произойдет, если нажать клавишу MEM. При этом символ регистра на дисплее будет стерт. Наименование регистра хранится в ячейке памяти PGNAM после использования клавиш NEXT все время, пока индицируется содержимое регистра.
STEP	Шаг	Предназначена для выполнения программы в пошаговом режиме, когда УОУ находится в состоянии «Отладка» и происходит останов после выполнения каждой команды. При нажатии STEP в ячейку памяти SFLAG записывается единица, что означает, что при следующем вызове дисплея будут разблокированы дисплей и клавиатура. Если перед клавишей STEP нажать клавишу ADDR и четыре цифровые клавиши, то в счетчик команд пользователя введется новый адрес и программа пользователя будет выполняться с этого адреса.
RUN	Прогон	Предназначена для выполнения программы в непрерывном режиме. При этом, для того, чтобы после выполнения программы произошло прерывание и обращение к монитору, необходимо в качестве команды останова программы использовать команду RST4, а не команду HLT.
BRK	Контрольная точка	Служит для задания адресов контрольных точек в программе с целью автоматического останова программы по заданному адресу после выполнения какой-либо команды, что позволяет проверить промежуточные результаты.

Командные клавиши инициируют выполнение отдельных модулей программы монитора, что приводит к выполнению соответствующей команды монитора.

Командная клавиша RST служит для установки УОУ в исходное состояние. При нажатии на клавишу RST на индикаторах АДРЕС появится значение 8200, на индикаторах РЕГИСТР – пробел, а на индикаторах ДАННЫЕ – случайная информация.

Командная клавиша ADDR вызывает счетчик команд пользователя и отображает его содержимое на четырех левых индикаторах АДРЕС, т.е. этой клавишей определяется адрес ячейки памяти.

Если за клавишей ADDR последовательно нажимаются четыре цифровые клавиши, то индицируемый адрес заменяется новым. При этом на двух правых индикаторах ДАННЫЕ появится содержимое адресуемой ячейки памяти. нажатие клавиши NEXT выведет на индикатор информацию об адресе и значении следующей ячейки памяти.

Командная клавиша MEM инициирует адрес ячейки памяти, содержимое которой можно изменить при помощи цифровых клавиш. Нажатие этой клавиши фиксируется появлением точки шестого слева индикатора дисплея, что указывает на то, что ввод в память разрешен. Если эта точка не светится, то данные вводиться не будут.

При последовательном нажатии шести клавиш

ADDR KKKK MEM

где К – символ цифровой клавиши, на индикаторах АДРЕС высветится адрес ячейки памяти КККК, а на индикаторах ДАННЫЕ - содержимое этой ячейки. При этом загорится соответствующая точка, что позволяет осуществить ввод новых данных в заданную ячейку памяти нажатием одной или двух шестнадцатиричных клавиш.

Для перехода к адресу следующей по порядку ячейки памяти нужно нажать клавишу NEXT, при этом нет необходимости нажимать клавишу MEM еще раз. Повторные нажатия клавиши MEM будут уменьшать на единицу адрес ячейки памяти.

При последовательном нажатии трех клавиш

ADDR P MEM

в разрядах дисплея РЕГИСТР отобразится имя регистровой пары, а в разрядах АДРЕС – ее содержимое. В разрядах данные отобразится содержимое ячейки памяти, адресуемое регистровой парой. Эти данные могут быть заменены другими, путем ввода с цифровых клавиш.

Адрес, указываемый регистровой парой, можно увеличить или уменьшить на единицу клавишами NEXT или MEM, при этом символ регистровой пары на дисплее замещается пробелами.

Командная клавиша NEXT служит для перехода к следующему адресу. Этим адресом моет быть адрес ячейки памяти, адрес контрольной точки, а также символ восьмиразрядного регистра в такой последовательности: A, B, C, D, E, F, H, L, A, B, C, … .

Командная клавиша CLR при нажатии восстанавливает значения адреса или данных. При этом, если индицируется содержимое ячейки памяти и ранее была нажата клавиша MEM, то восстанавливается предыдущее значение данных.

Если индицируется содержимое ячейки памяти, но клавиша MEM не была нажата, клавиша CLR выведет на дисплей содержимое счетчика команд пользователя. Изменения в адресуемой им ячейке памяти возможны после нажатия клавиши MEM.

При вводе адреса с клавиатуры клавиша CLR выводит на дисплей содержимое счетчик а команд пользователя и не запрещает ввод нового адреса. Нажатие на клавишу MEM восстановит предыдущий адрес ячейки памяти (но не выведет его на дисплей).

Если высвечивается содержимое регистра МП, клавиша CLR вызовет отображение предыдущего содер содержимого регистра и его символа.

Командная клавиша REG предназначена для чтения и записи содержимого восьмиразрядного регистра МП. При нажатии на клавиш REG на дисплее появится текущий адрес команд пользователя (АДРЕС), символ регистра (РЕГИСТР) и содержимое выбранного регистра МП (ДАННЫЕ).

Для ввода новых данных в регистр необходимо последовательно нажать клавиши:

REG R KK,

где R – символ восьмиразрядного регистра, а КК – символическое обозначение двух последовательно нажимаемых клавиш данных.

Командная клавиша STEP предназначена для выполнения программы в пошаговом режиме, когда УОУ находится в состоянии «Отладка» и происходит останов после выполнения каждой команды.

При нажатии клавиши STEP в ячейку памяти SFLAG записывается единица, что означает, что при следующем вызове дисплея будут разблокированы дисплей и клавиатура. содержимое пользователя во всех регистрах МП восстанавливается, система прерываний разблокируется и управление передается программе пользователя с адреса, записанного в ячейке памяти. При этом после выполнения очередной команды вызывается программа монитора.

Если перед клавишей STEP нажать клавишу ADDR и четыре цифровые клавиши, то в счетчик команд пользователя введется новый адрес и программа пользователя будет выполняться с этого адреса.

Командная клавиша RUN предназначена для выполнения программы в непрерывном режиме, когда УОУ находится в состоянии «Прогон». При этом, для того, чтобы после выполнения программы произошло прерывание и обращение к монитору, который включает индикатор, необходимо в качестве команды останова програ программы использовать команду RST4, а не команду HLT.

Для запуска программы в непрерывном режиме необходимо последовательно нажать клавиши

ADDR KKKK RUN,

где КККК – начальный адрес программы.

Если УОУ установлено в состояние «Отладка», то указанная последовательность нажатия клавиш приведет к выполнению программы с остановом по заранее заданным контрольным точкам (адресам).

Командная клавиша BRK служит для задания адресов контрольных точек в программе с целью автоматического останова программы по заданному адресу после выполнения какой-либо команды.

Ввод контрольной точки производится в режиме «Отладка» последовательным нажатием клавиш:

ADDR KKKK BRK NN,

где КККК – адрес контрольной точки, NN – число проходов контрольной точки (наибольшее число проходов контрольной точки равно 256).

Помимо указанного способа контроля, когда останов производится по адресу заданной команды, система контрольных точек позволяет остановить программу после выполнения заданного числа команд. Для этого необходимо в ячейку памяти с адресом 83E6 записать число команд КК, которое надо выполнить, для чего необходимо нажать следующие клавиши:

ADDR 83E6 BRK KK,

где КК – число команд.

Ячейка памяти 83E6 рассматривается как счетчик команд и имеет символическое обозначение PCADDR. Так как его содержимое меняется при выполнении каждой команды, останов произойдет после выполнения установленного в счетчике числа проходов.

Составление и запись программ.

При написании программ для УОУ «Электроника-580» нужно вручную переводить язык Ассемблер на машинный язык. Затем машинный код с клавиатуры вводится в заданную область памяти УОУ и производится отладка программы.

Для перевода программы, записанной с использованием мнемоники языка Ассемблер, на машинный язык следует использовать таблицу 2. Эта таблица, составленная в матричном виде, содержит в качестве элементов матрицы мнемонические обозначения команд УОУ. При использовании таблицы для определения кода команды необходимо найти в нем мнемоническое изображение команды УОУ. Это изображение находится на пересечении столбца и строки, которые определяют соответственно старший и младший разряды машинного кода.

Например, изображение команды LDA находится на пересечении столбца 3 и строки A; следовательно, машинный код команды LDA имеет вид 3A, а для команды загрузки аккумулятора MVI машинный код равен 3E.

Практическая часть

Тест-программа проверки функционирования ОЗУ.

Программа проверки функционирования ОЗУ, записанная на машинном языке, представлена в табл. 1. Тест состоит из трёх подпрограмм: тест «число», тест» адрес, тест «бегущие значения».

Данный тест проверяет объём встроенного ОЗУ, его работоспособность, а также возможность выполнения программы пользователя в автоматическом режиме.

аблица 1

Тест проверки функционирования ОЗУ

Адрес	0 1 2 3 4 5 6 7 8 9 A B C D E F
800	31 E0 83 21 E0 80 AF 37 17 77 BE CA 12 80 CD CE
801	02 76 2F 77 BE C2 0E 80 2F 07 D2 09 80 23 7C FE
802	88 C2 06 80 CD 9D 80 21 00 88 2B 74 2B 75 7C FE
803	80 C2 2A 80 7D FE EO C2 2A 80 7D BE C2 OB 80 23
804	7C BE OB 80 23 7C FE 88 C2 3A 80 CD A8 80 00 00
805	00 00 06 FF OE 00 21 EO 80 71 23 7C FE 88 C2 59
806	80 21 EO 80 54 5D 78 12 79 BE CA 99 80 7C FE 88
807	C2 8B 80 21 EO 80 79 12 13 7A FE 88 C2 66 80 AF
808	00 CA B3 80 06 00 OE FF C3 56 80 BA C2 OB 80 7D
809	BB C2 OB 80 7E B8 C2 OB 80 23 C3 68 80 21 FE 83
80A	36 39 2B 36 66 C3 BF 80 21 FF 83 36 5E 2B 36 77
80B	C3 BF 80 21 FF 83 36 76 2B 36 00 CD BF 80 76 2B
80C	36 6E 2B 36 4F 2B 36 07 2B 36 39 2B 36 79 2B 36
80D	07 06 FF OE FB E3 E3 OD C2 B5 80 05 C2 D3 80 C9

Примечания:

1. Загрузка программы производится последовательным нажатием клавиш RST ADDR 8000 MEM 31 NEXT 83…80 NEXT C9.

2. Выполнение программы в непрерывном режиме инициируется последовательным нажатием клавиш RST ADDR 8000 RUN.

3.В ходе работы программы на индикаторы выводятся фразы:

-по окончании теста «число»: ТЕСТ ЗУЧС

-по окончании теста «адрес»: ТЕСТ ЗУАД

-по окончании теста «бегущие огни» : ТЕСТ ЗУН.

Индикация первых двух фраз осуществляется в течение 1-2 с. В ходе работы тестов на индикаторах может находиться произвольная информация. Если какой-либо тест выполняется неправильно, то в первых четырёх разрядах высвечивается адрес неисправной ячейки памяти, а в двух последних разрядах – содержимое этой ячейки.

Контрольные вопросы

1. С какой целью осуществляется мультиплексирование ЩД?
2. Перечислите все программно-недоступные регистры МП с указанием их назначения.
3. Какую информацию содержит первый байт команды? Второй? Третий?
4. Объясните работу стека в МП-системе.
5. Какую функцию выполняет схема десятичной коррекции в АЛУ?
6. Объясните назначение всех управляющих выводов в МП.
7. С какой целью используется ПДП?
8. Каким образом осуществляется адресация РОН?
9. Объясните работу всех командных клавиш УОУ.
10. Какую информацию и с какой целью фиксирует регистр признаков АЛУ?
11. Объясните характер выполнения всех возможных операций в АЛУ.
12. В каком случае изменяется содержимое программного счётчика.

13. Какая наибольшая ёмкость памяти может быть адресована 16 адресными линиями?
14. Как изменится содержимое FF регистровой пары HL после инкрементирования?
15. Определите слово состояния программы и слово состояния процессора.

ПРАКТИЧЕСКАЯ РАБОТА №12. ОСОБЕННОСТИ АРХИТЕКТУРЫ ОДНОКРИСТАЛЬНЫХ МИКРОКОНТРОЛЛЕРОВ

Цель работы: изучение назначения и особенностей архитектуры однокристальных микроконтроллеров; ознакомление с архитектурой и программной моделью AVR-микроконтроллеров.

Основные теоретические сведения

Микроконтроллеры составляют наиболее широкий класс микропроцессоров, используемых в приборах, устройствах и системах различного назначения. Микроконтроллер - это специализированный микропроцессор, предназначенный для построения устройств управления техническими объектами и технологическими процессами. Конструктивно микроконтроллер представляет собой большую интегральную схему (БИС), на кристалле которой размещены все составные части типовой вычислительной системы: микропроцессор, память, а также периферийные устройства для реализации дополнительных функций. Так как все элементы микроконтроллера размещены на одном кристалле, их также называют *однокристальными* (однокорпусными) *микро-ЭВМ* или *однокристальными микроконтроллерами.* Цель применения микроконтроллеров - сокращение числа компонентов, уменьшение размеров и снижение стоимости приборов (систем).

Как правило, микроконтроллеры имеют RISC-архитектуру (RISC - Reduced Instruction Set Computer - вычислитель с сокращённым набором команд), незначительную ёмкость памяти, физическое и логическое разделение памяти программ и памяти данных, ориентированную на задачи управления систему команд. Таким образом, микроконтроллеры предназначены для решения задач управления, контроля, регулирования и первичной обработки информации и менее эффективны при реализации сложных алгоритмов обработки данных.

В состав типовой микроконтроллерной системы управления входит микроконтроллер и аппаратура его сопряжения с объектом управления (рис. 1). Микроконтроллер производит периодический опрос сигналов состояния объекта и в соответствии с заложенным алгоритмом генерирует последовательности сигналов управления. *Сигналы состояния* характеризуют текущие параметры объекта управления. Они формируются путём преобразования выходных сигналов датчиков (Д) с помощью аналого-цифровых преобразователей (АЦП) или формирователей сигналов состояния (ФСС); последние чаще всего выполняют функции гальванической развязки и формирования уровней. *Сигналы управления,* выработанные микроконтроллером, подвергаются преобразованию с помощью цифро-аналоговых преобразователей (ЦАП) или формирователей сигналов управления (ФСУ), в качестве которых применяются усилители мощности, оптроны, ключи и др. Выходные сигналы ЦАП и ФСУ представляют собой управляющие воздействия, которые поступают на исполнительные устройства (ИУ). В системе может быть также предусмотрены панель управления, устройство индикации и интерфейс для обмена информацией с внешними устройствами. В зависимости от назначения и характеристик конкретной системы некоторые из указанных элементов могут отсутствовать.

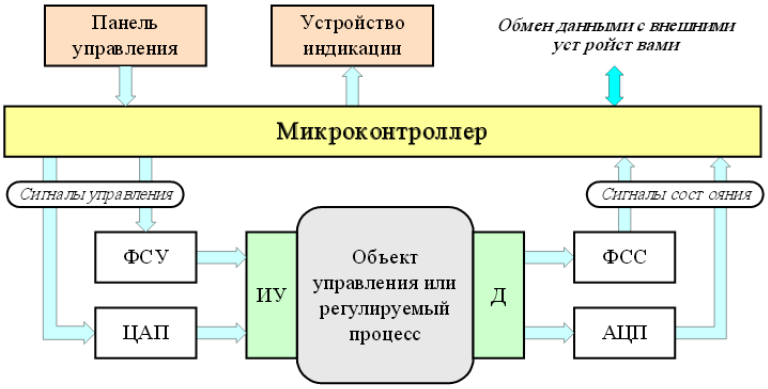


Рис. 1. *Типовая структура системы управления на основе микроконтроллера.*

ФСУ – формирователи сигналов управления; ИУ – исполнительные устройства; Д – датчики; ФСС – формирователи сигналов состояния

Разрядность выпускаемых микроконтроллеров варьируется от 4 до 64 бит. Наибольшее распространение получили 8-разрядные микроконтроллеры как пригодные для использования в различных приложениях и имеющие низкую стоимость. Характерными представителями таких устройств являются микроконтроллеры семейства **AVR** фирмы **Atmel.**

AVR-микроконтроллеры - это 8-разрядные RISC-микроконтроллеры, отличительными особенностями которых являются наличие FLASH-памяти программ, широкий спектр периферийных устройств, высокая вычислительная производительность, а также доступность средств разработки программного обеспечения.

В состав семейства **AVR** в настоящее время входят более 50 различных устройств, которые подразделяются на несколько групп.

Универсальные **AVR**-микроконтроллеры входят в группы **Tiny AVR** и **Mega AVR. Tiny AVR (ATtinyXXX)** - дешёвые устройства с небольшим количеством выводов. **Mega AVR (ATmegaXXX)** - мощные **AVR**-микроконтроллеры, имеющие наибольшие объёмы памяти и количество выводов, а также максимально полный набор периферийных устройств.

Специализированные **AVR**-микроконтроллеры представлены группами **LCD AVR (ATmega169, ATmega329)** - микроконтроллеры для работы с жидкокристаллическими индикаторами; **USB AVR (AT43USBX\X, AT76C711)** микроконтроллеры с интерфейсом **USB; DVD AVR (AT78CXXX)** - контроллеры CD/DVD-приводов; **RF AVR (AT86RFXXX)** - микроконтроллеры для построения систем беспроводной связи; **Secure AVR (AT90SCXXXAT, AT97SCXXXX)** - микроконтроллеры для смарт-карт; **FPGA AVR (AT94K\XAL)** - AVR-микроконтроллеры, выполненные на одном кристалле с ПЛИС.

Кроме того, ранее выпускалось группа **Classic AVR (AT90SXXXX)** - устройства, занимающие промежуточное положение между микроконтроллерами групп **Mega** и **Tiny** (заменены совместимыми усовершенствованными аналогами группы **Mega)**.

Архитектура AVR-микроконтроллеров. AVR-микроконтроллеры содержат на кристалле следующие аппаратные средства: 8-разрядное процессорное ядро, память программ, оперативную память данных, энергонезависимую память данных, регистры ввода-вывода, схему прерываний, схему программирования, а также периферийные устройства (рис. 2)

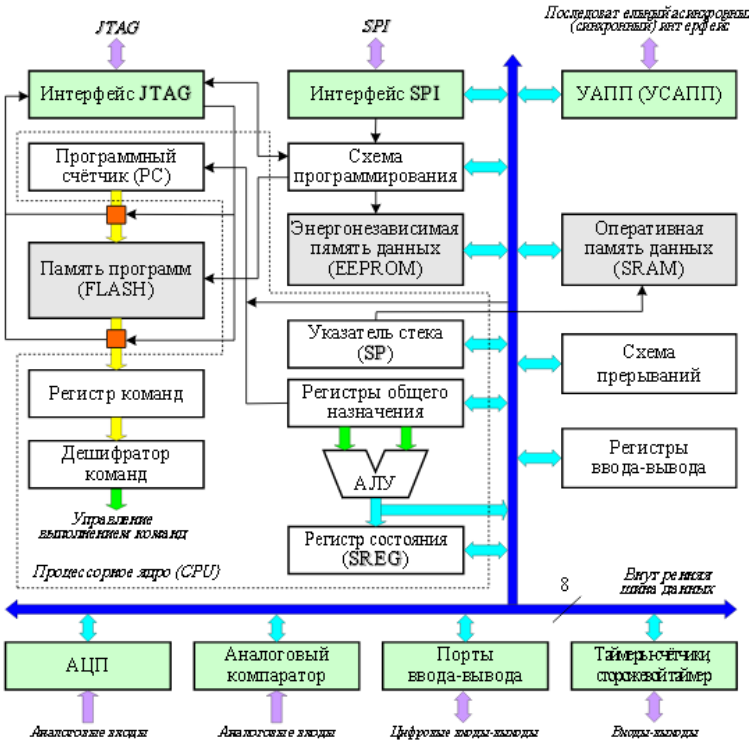


Рис. 2. *Архитектура микроконтроллеров семейства AVR*

Процессорное ядро (Central Processing Unit - CPU) **AVR**-микроконтроллеров содержит арифметико-логическое устройство (АЛУ), регистры общего назначения (РОН), программный счётчик, указатель стека, регистр состояния, регистр команд, дешифратор команд, схему управления выполнением команд.

В *АЛУ* выполняются все вычислительные операции. Операции производятся только над содержимым РОН. На выборку содержимого регистров, выполнение операции и запись результата обратно в РОН затрачивается один машинный такт (один период тактовой частоты).

Регистры общего назначения представляют собой 8-разрядные ячейки памяти с быстрым доступом, непосредственно доступные АЛУ. В **AVR**-микроконтроллерах имеется 32 РОН.

Программный счётчик (Program Counter - **PC)** содержит адрес следующей выполняемой команды.

Указатель стека (Stack Pointer - **SP)** служит для хранения адреса вершины стека (см. лабораторную работу № 5).

Регистр состояния (Status Register - **SREG)** содержит слово состояния процессора.

Регистр команд, дешифратор команд и схема управления выполнением команд обеспечивают выборку из памяти программ команды, адрес которой содержится в программном счётчике, её декодирование, определение способа доступа к указанным в команде аргументам и собственно выполнение команды. Для ускорения выполнения команд используется механизм конвейеризации, который заключается в том, что во время исполнения текущей команды программный код следующей выбирается из памяти и декодируется.

Память **AVR**-микроконтроллеров организована по схеме гарвардского типа - адресные пространства памяти программ и памяти данных разделены.

Память программ представляет собой перепрограммируемое ПЗУ типа FLASH и выполнена в виде последовательности 16-разрядных ячеек, так как большинство команд **AVR**-микроконтроллера являются 16-разрядными словами. FLASH-память не обладает возможностью перезаписи отдельных ячеек, поэтому всегда выполняется полная очистка всей памяти программ. При этом гарантируется не менее 10 000 циклов перезаписи. Память программ имеет размер от 1 до 128K слов.

Оперативная память данных представляет собой статическое ОЗУ (SRAM - Static Random-Access Memory) и организована как последовательность 8-разрядных ячеек. Оперативная память данных может быть внутренней (до 16K байт) и внешней (до 64K байт).

Энергонезависимая (nonvolatile) *память данных* организована как последовательность 8-разрядных ячеек и представляет собой перепрограммируемое ПЗУ с электрическим стиранием (РПЗУ-ЭС, или EEPROM - Electrically Erasable Programmable Read-only Memory). Энергонезависимая память данных имеет размер до 64К байт.

Регистры ввода-вывода предназначены для управления процессорным ядром и периферийными устройствами **AVR**-микроконтроллера.

Схема прерываний обеспечивает возможность асинхронного прерывания процесса выполнения программы при определённых условиях.

К *периферийным устройствам* **AVR**-микроконтроллера относятся порты ввода-вывода, таймеры, счётчики, сторожевой таймер, аналоговый компаратор, аналого-цифровой преобразователь, универсальный асинхронный (синхронно-асинхронный) приёмопередатчик - УАПП (УСАПП), последовательный периферийный интерфейс **SPI,** интерфейс **JTAG** и др. Набором периферийных устройств определяются функциональные возможности микроконтроллера.

Обмен информацией между устройствами **AVR**-микроконтроллера осуществляется посредством внутренней 8-разрядной шины данных.

Программная модель AVR-микроконтроллеров.

Программная модель микропроцессора представляет собой совокупность программно доступных ресурсов. В программную модель микроконтроллеров семейства **AVR** входят РОН, регистры ввода-вывода, память программ, оперативная память данных и энергонезависимая память данных (рис. 3).

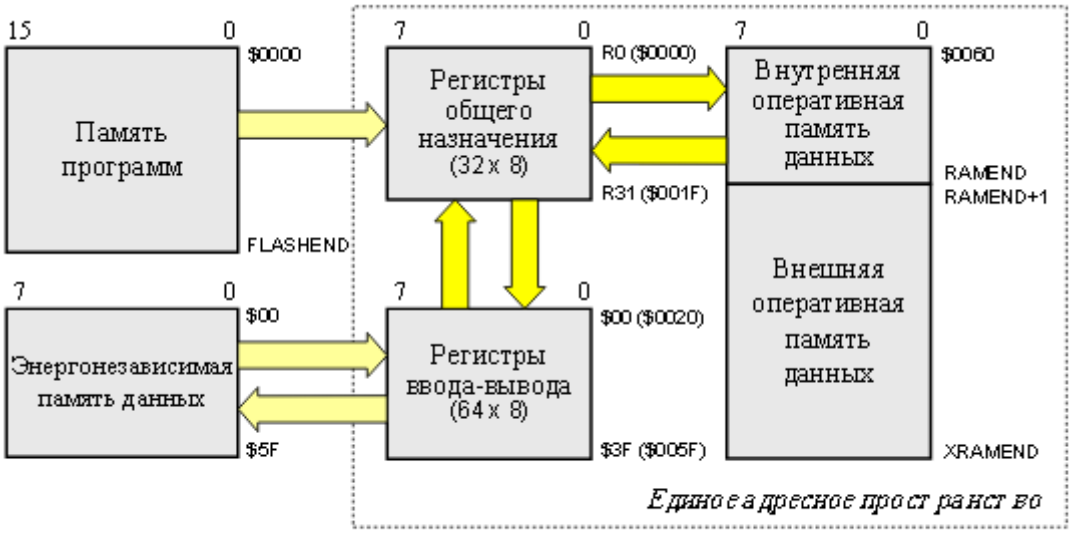


Рис. 3. *Программная модель AVR-микроконтроллеров*

РОН **(R0...R31)** могут использоваться в программе для хранения данных, адресов, констант и другой информации. Шесть старших регистров объединены попарно и составляют три 16-разрядных регистра **X [R27:R26], Y [R29:R28]** и **Z [R31:R30]** (рис. 4).

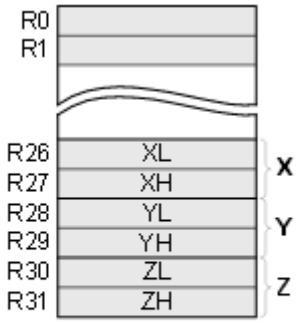


Рис. 4. *Регистры общего назначения*

РОН, регистры ввода-вывода и оперативная память данных образуют единое адресное пространство. Адресное пространство - это множество доступных ячеек памяти, различимых по адресам; адресом называется число, однозначно идентифицирующее ячейку памяти (регистр). Адреса ячеек памяти традиционно записываются в шестнадцатеричной системе счисления, на что указывает знак $ в обозначении адреса. Адреса в едином адресном пространстве **AVR**-микроконтроллеров распределяются следующим образом. Младшие 32 адреса ($0000...$001F) соответствуют РОН. Следующие 64 адреса ($0020...$005F) занимают регистры ввода-вывода. Внутренняя оперативная память данных у **AVR**-микроконтроллеров начинается с адреса $0060.

В память программ, кроме собственно программы, могут быть записаны постоянные данные, которые не изменяются в процессе работы микропроцессорной системы (константы, таблицы линеаризации датчиков и т. п.). Выполнение программы при включении питания или после сброса микроконтроллера начинается с команды, находящейся по адресу $0000 (т. е. в первой ячейке) памяти программ.

Энергонезависимая память данных предназначена для хранения информации, которая может изменяться непосредственно в процессе работы микропроцессорной системы (калибровочные коэффициенты, конфигурационные параметры и т. п.). Энергонезависимая память данных имеет отдельное адресное пространство и может быть считана и записана программным путём.

Система команд AVR-микроконтроллеров. Система команд (instruction set) микропроцессора представляет собой совокупность выполняемых микропроцессором операций и правила их кодирования в программе. Система команд **AVR**-микроконтроллеров включает команды (инструкции) арифметических и логических операций, команды ветвления, управляющие последовательностью выполнения программы, команды передачи данных и команды операций с битами. Всего в систему команд входит более 130 инструкций. Младшие модели микроконтроллеров не поддерживают некоторых из них.

Система команд **AVR**-микроконтроллеров приведена в Приложении 2.

Программирование микроконтроллеров. Процесс разработки прикладного ПО устройств на основе однокристальных микроконтроллеров включает следующие этапы (рис. 5):

- разработки алгоритма и структуры программы;
- написания исходного текста программы;
- получения выполняемой программы;
- тестирования и отладки программы;
- получения загрузочной программы.

Рис. 5. *Последовательность разработки ПО для микроконтроллеров*

На этапе разработки алгоритма и структуры программы выбирается метод решения задачи и разрабатывается алгоритм его реализации. Алгоритм - это набор правил или описание последовательности операций для решения определённой задачи или достижения некоторой цели. Графическим изображением алгоритма является *схема алгоритма* (flowchart), выполняемая в соответствии с ГОСТ 19.701-90 «Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения».

На этапе написания исходного текста программы разработанный алгоритм записывается в виде программы на исходном языке (ассемблере или языке высокого уровня).

Языком ассемблера называется язык программирования, в котором каждой команде процессора или совокупности команд процессора соответствует сокращённая символическая запись (мнемоника). Использование символического обозначения команд, а также адресов регистров и ячеек памяти, переменных, констант и других элементов программы существенно облегчает процесс составления программ по сравнению с программированием на уровне машинных кодов. Символические обозначения элементов обычно отражают их содержательный смысл. Язык ассемблера обеспечивает возможность гибкого доступа ко всем ресурсам программируемого микропроцессора (микроконтроллера) и позволяет создавать программы, наиболее эффективные как по быстродействию, так и по объёму занимаемой программной памяти. В этой связи программирование на языке ассемблера предполагает знание архитектуры и свойств микропроцессора, т. е. всего того, что входит в понятие «программная модель». Языки ассемблера являются машинно-ориентированными и, следовательно, отличаются для разных типов микропроцессоров. В ряде ассемблеров допускается оформление повторяющейся последовательности команд как одной макрокоманды (макроса), такие ассемблеры называют *макроассемблерами.*

Языки высокого уровня (С, Паскаль, Бейсик и др.), как и ассемблер, обеспечивают доступ ко всем ресурсам микроконтроллера, но вместе с тем дают возможность создавать хорошо структурированные программы, снимают с программиста заботу о распределении памяти и содержат большой набор библиотечных функций для выполнения стандартных операций.

На этапе получения выполняемой программы исходный текст программы с помощью специальных средств (трансляторов, компиляторов, компоновщиков и др.) преобразуется в исполняемый код. *Транслятором* (translator) называют программу, служащую для перевода (трансляции) программ на языке ассемблера в машинный код, «понимаемый» процессором.

Компилятор (compiler) представляет собой программу, преобразующую в эквивалентный машинный код текст программы на языке высокого уровня. Результатом работы транслятора или компилятора может быть как выполняемый загрузочный модуль, так и объектный модуль (программа, команды, переменные и константы которой не «привязаны» к конкретным адресам ячеек памяти). Для построения выполняемой программы из объектных модулей применяется *компоновщик* (редактор связей, linker). В процессе получения выполняемой программы из исходного текста программы устраняются синтаксические ошибки, состоящие в нарушении правил синтаксиса используемого языка программирования.

На этапе тестирования и отладки программы производится поиск, локализация и устранение в ней логических ошибок. *Тестирование* служит для обнаружения в программе ошибок и выполняется с использованием некоторого набора тестовых данных. Тестовые данные должны обеспечивать проверку всех ветвей алгоритма. При тестировании программы могут подвергаться проверке также некоторые показатели системы, связанные с программой (например, объём кодов и данных). После тестирования программа должна быть подвергнута *отладке* (debug), задачей которой является локализация ошибки, т. е. нахождение места в программе, вызывающего ошибку.

Тестирование и отладка программы могут привести (и, как правило, приводят) к необходимости возврата к ранним этапам процесса разработки программы для устранения ошибок в постановке задачи, разработке алгоритма, написании исходного текста и т. д. Таким образом, процесс разработки программы, как и весь процесс проектирования, является итерационным.

На этапе получения загрузочной программы производится «освобождение» программы от лишних фрагментов, использовавшихся для тестирования и отладки. Эти фрагменты увеличивают объём программы и не нужны при нормальном функционировании микропроцессорной системы. Далее полученная загрузочная программа заносится в память микроконтроллера.

По завершении процесса разработки производится *документирование,* т. е. составление комплекта документов, необходимых для эксплуатации и сопровождения программы. Сопровождение программы (program maintenance) - это процесс внесения изменений, исправления оставшихся ошибок и проведения консультаций по программе, находящейся в эксплуатации. Виды программных документов регламентированы ГОСТ 19.101-77 «Единая система программной документации. Виды программ и программных документов».

Разработка ПО для встраиваемых микропроцессоров производится на персональном компьютере с использованием специальных программных и аппаратных средств. Такой способ создания ПО носит название *кросс-разработки.* Совокупность аппаратных и программных средств, применяемых для разработки и отладки ПО, объединяют общим наименованием *средства поддержки разработки.*

В настоящем практикуме процесс разработки ПО изучается на примере языка ассемблера **AVR**-микроконтроллеров. Создание исходного текста программы, трансляция и отладка выполняются в интегрированной среде разработки (Integrated Development Environment - IDE) **AVR Studio.**

ПРАКТИЧЕСКАЯ РАБОТА №13. СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА AVR.

Цель работы: изучить систему команд микроконтроллеров семейства AVR.

Основные теоретические сведения

Система команд (instruction set) микропроцессора представляет собой совокупность выполняемых микропроцессором операций и правила их кодирования в программе. Система команд AVR-микроконтроллеров включает команды (инструкции) арифметических и логических операций, команды ветвления, управляющие последовательностью выполнения программы, команды передачи данных и команды операций с битами. Всего в систему команд входит более 130 инструкций.

Младшие модели микроконтроллеров не поддерживают некоторых из них. Система команд AVR-микроконтроллеров приведена ниже.

Система команд микроконтроллеров семейства AVR

Мнемоника	Операнды	Описание	Операция	Флаги	Число тактов
1	2	3	4	5	6
Арифметические и логические команды					
ADD	**Rd, Rr**	Сложить без учёта флага переноса	**Rd ← Rd + Rr**	**Z, C, N, V, H, S**	1
ADC	**Rd, Rr**	Сложить с учётом флага переноса	**Rd ← Rd + Rr + C**	**Z, C, N, V, H, S**	1
ADIW	**Rdl, K6**	Сложить слово и константу	**Rdh:Rdl ← Rdh:Rdl – K6**	**Z, C, N, V, S**	2
SUB	**Rd, Rr**	Вычесть без учёта флага переноса	**Rd ← Rd – Rr**	**Z, C, N, V, H, S**	1
SUBI	**Rd, K8**	Вычесть константу	**Rd ← Rd – K8**	**Z, C, N, V, H, S**	1
SBC	**Rd, Rr**	Вычесть с учётом флага переноса	**Rd ← Rd – Rr – C**	**Z, C, N, V, H, S**	1
SBCI	**Rd, K8**	Вычесть константу с учётом флага переноса	**Rd ← Rd – K8 – C**	**Z, C, N, V, H, S**	1
SBIW	**Rdl, K6**	Вычесть константу из слова	**Rdh:Rdl ← Rdh:Rdl – K6**	**Z, C, N, V, S**	2
AND	**Rd, Rr**	Выполнить логическое И	**Rd ← Rd · Rr**	**Z, N, V, S**	1
ANDI	**Rd, K8**	Выполнить логическое И с константой	**Rd ← Rd · K8**	**Z, N, V, S**	1
OR	**Rd, Rr**	Выполнить логическое ИЛИ	**Rd ← Rd v Rr**	**Z, N, V, S**	1
ORI	**Rd, K8**	Выполнить логическое ИЛИ с константой	**Rd ← Rd v K8**	**Z, N, V, S**	1
EOR	**Rd, Rr**	Выполнить логическое исключающее ИЛИ	**Rd ← Rd ⊕ Rr**	**Z, N, V, S**	1
COM	**Rd**	Вычислить дополнение до одного (поразрядная инверсия)	**Rd ← $FF – Rd**	**Z, C, N, V, S**	1
NEG	**Rd**	Вычислить дополнение до двух (изменить знак)	**Rd ← $00 – Rd**	**Z, C, N, V, H, S**	1
SBR	**Rd, K8**	Установить разряд (разряды) в РОН	**Rd ← Rd v K8**	**Z, C, N, V, S**	1
CBR	**Rd, K8**	Сбросить разряды в РОН	**Rd ← Rd · ($FF – K8)**	**Z, C, N, V, S**	1
INC	**Rd**	Инкрементировать содержимое РОН	**Rd ← Rd + 1**	**Z, N, V, S**	1
DEC	**Rd**	Декрементировать содержимое РОН	**Rd ← Rd – 1**	**Z, N, V, S**	1
TST	**Rd**	Проверить на равенство нулю или отрицательное значение	**Rd ← Rd · Rd**	**Z, C, N, V, S**	1
CLR	**Rd**	Очистить все разряды РОН	**Rd ← Rd ⊕ Rd**	**Z, C, N, V, S**	1
SER	**Rd**	Установить все разряды РОН	**Rd ← $FF**	Нет	1
CP	**Rd, Rr**	Сравнить	**Rd – Rr**	**Z, C, N, V, H, S**	1
CPC	**Rd, Rr**	Сравнить с учётом флага переноса	**Rd – Rr – C**	**Z, C, N, V, H, S**	1

1	2	3	4	5	6
CPI	**Rd, K8**	Сравнить с константой	**Rd – K8**	**Z, C, N, V, H, S**	1
MUL	**Rd, Rr**	Перемножить содержимое двух РОН (без знака)	**R1:R0 ← Rd · Rr**	**Z, C**	2
MULS	**Rd, Rr**	Перемножить содержимое двух РОН (со знаком)	**R1:R0 ← Rd · Rr**	**Z, C**	2
MULSU	**Rd, Rr**	Перемножить содержимое двух РОН (**Rd** – со знаком; **Rr** – без знака)	**R1:R0 ← Rd · Rr**	**Z, C**	2
FMUL	**Rd, Rr**	Перемножить содержимое двух РОН (без знака) со сдвигом влево на 1 разряд	**R1:R0 ← Rd · Rr** << 1	**Z, C**	2
FMULS	**Rd, Rr**	Перемножить содержимое двух РОН (со знаком) со сдвигом влево на 1 разряд	**R1:R0 ← Rd · Rr** << 1	**Z, C**	2
FMULSU	**Rd, Rr**	Перемножить содержимое двух РОН (**Rd** – со знаком; **Rr** – без знака) со сдвигом влево на 1 разряд	**R1:R0 ← Rd · Rr** << 1	**Z, C**	2
Команды ветвления					
RJMP	**k**	Относительный переход	**PC ← PC + k +1**	Нет	2
IJMP	Нет	Косвенный переход	**PC ← Z**	Нет	2
EIJMP	Нет	Расширенный косвенный переход	**PC**(15:0) ← **Z**, **PC**(21:16) ← **EIND**	Нет	2
JMP	**k**	Косвенный переход	**PC ← k**	Нет	3
RCALL	**k**	Относительный вызов подпрограммы	**STACK ← PC + 1**, **PC ← PC + k + 1**	Нет	3
ICALL	Нет	Косвенный вызов подпрограммы	**STACK ← PC + 1**, **PC ← Z**	Нет	3
EICALL	Нет	Расширенный косвенный вызов подпрограммы	**STACK ← PC + 1**, **PC**(15:0) ← **Z**, **PC**(21:16) ← **EIND**	Нет	3
CALL	**k**	Вызов подпрограммы	**STACK ← PC + 1**, **PC ← k**	Нет	4/5
RET	Нет	Возврат из подпрограммы	**PC ← STACK**	Нет	4
RETI	Нет	Возврат из подпрограммы обработки прерывания	**PC ← STACK**	**I**	4
CPSE	**Rd, Rr**	Сравнить и пропустить, если равно	если **Rd = Rr**, **PC ← PC + 2** (или 3)	Нет	1/2/3
SBRC	**Rr, b**	Пропустить, если бит в РОН сброшен	если **Rr(b)** = 0, **PC ← PC + 2** (или 3)	Нет	1/2/3
SBRS	**Rr, b**	Пропустить, если бит в РОН установлен	если **Rr(b)**=1, **PC ← PC + 2** (или 3)	Нет	1/2/3
SBIC	**I/O, b**	Пропустить, если бит в регистре ввода-вывода сброшен	если **I/O(b)** = 0, **PC ← PC + 2** (или 3)	Нет	1/2/3
SBIS	**I/O, b**	Пропустить, если бит в регистре ввода-вывода установлен	если **I/O(b)** = 1, **PC ← PC + 2** (или 3)	Нет	1/2/3
BRBC	**s, k**	Перейти, если флаг в регистре **SREG** сброшен	если **SREG(s)** = 0, **PC ← PC + k + 1**	Нет	1/2

1	2	3	4	5	6
BRBS	**s, k**	Перейти, если флаг в регистре **SREG** установлен	если **SREG(s)** = 1, **PC** ← **PC** + **k** + 1	Нет	1/2
BREQ	**k**	Перейти, если равно	если **Z** = 1, **PC** ← **PC** + **k** + 1	Нет	1/2
BRNE	**k**	Перейти, если не равно	если **Z** = 0, **PC** ← **PC** + **k** + 1	Нет	1/2
BRCS	**k**	Перейти, если флаг переноса установлен	если **C** = 1, **PC** ← **PC** + **k** + 1	Нет	1/2
BRCC	**k**	Перейти, если флаг переноса сброшен	если **C** = 0, **PC** ← **PC** + **k** + 1	Нет	1/2
BRSH	**k**	Перейти, если равно или больше	если **C** = 0, **PC** ← **PC** + **k** + 1	Нет	1/2
BRLO	**k**	Перейти, если меньше	если **C** = 1, **PC** ← **PC** + **k** + 1	Нет	1/2
BRMI	**k**	Перейти, если минус	если **N** = 1, **PC** ← **PC** + **k** + 1	Нет	1/2
BRPL	**k**	Перейти, если плюс	если **N** = 0, **PC** ← **PC** + **k** + 1	Нет	1/2
BRGE	**k**	Перейти, если больше или равно (со знаком)	если **S** = 0, **PC** ← **PC** + **k** + 1	Нет	1/2
BRLT	**k**	Перейти, если меньше (со знаком)	если **S** = 1, **PC** ← **PC** + **k** + 1	Нет	1/2
BRHS	**k**	Перейти, если флаг полупереноса установлен	если **H** = 1, **PC** ← **PC** + **k** + 1	Нет	1/2
BRHC	**k**	Перейти, если флаг полупереноса сброшен	если **H** = 0, **PC** ← **PC** + **k** + 1	Нет	1/2
BRTS	**k**	Перейти, если флаг **T** установлен	если **T** = 1, **PC** ← **PC** + **k** + 1	Нет	1/2
BRTC	**k**	Перейти, если флаг **T** сброшен	если **T** = 0, **PC** ← **PC** + **k** + 1	Нет	1/2
BRVS	**k**	Перейти, если флаг переполнения установлен	если **V** = 1, **PC** ← **PC** + **k** + 1	Нет	1/2
BRVC	**k**	Перейти, если флаг переполнения сброшен	если **V** = 0, **PC** ← **PC** + **k** + 1	Нет	1/2
BRIE	**k**	Перейти, если прерывания разрешены	если **I** = 1, **PC** ← **PC** + **k** + 1	Нет	1/2
BRID	**k**	Перейти, если прерывания запрещены	если **I** = 0, **PC** ← **PC** + **k** + 1	Нет	1/2
Команды передачи данных					
MOV	**Rd, Rr**	Копирование РОН	**Rd** ← **Rr**	Нет	1
MOVW	**Rd, Rr**	Копирование пары РОН	**Rd**+1:**Rd** ← **Rr**+1:**Rr**	Нет	1
LDI	**Rd, K8**	Загрузка константы в РОН	**Rd** ← **K8**	Нет	1
LDS	**Rd, k**	Прямая загрузка из ОЗУ в РОН	**Rd** ← (**k**)	Нет	2
LD	**Rd, X**	Косвенная загрузка из ОЗУ в РОН	**Rd** ← (**X**)	Нет	2

1	2	3	4	5	6
LD	**Rd, Y**	Косвенная загрузка из ОЗУ в РОН	**Rd ← (Y)**	Нет	2
LD	**Rd, Y+**	Косвенная загрузка из ОЗУ с постинкрементом	**Rd ← (Y), Y ← Y + 1**	Нет	2
LD	**Rd, –Y**	Косвенная загрузка из ОЗУ с предекрементом	**Y ← Y – 1, Rd ← (Y)**	Нет	2
LDD	**Rd, Y+q**	Косвенная загрузка из ОЗУ со смещением	**Rd ← (Y + q)**	Нет	2
LD	**Rd, Z**	Косвенная загрузка из ОЗУ в РОН	**Rd ← (Z)**	Нет	2
LD	**Rd, Z+**	Косвенная загрузка из ОЗУ с постинкрементом	**Rd ← (Z), Z ← Z + 1**	Нет	2
LD	**Rd, –Z**	Косвенная загрузка из ОЗУ с предекрементом	**Z ← Z – 1, Rd ← (Z)**	Нет	2
LDD	**Rd, Z+q**	Косвенная загрузка из ОЗУ со смещением	**Rd ← (Z + q)**	Нет	2
STS	**k, Rr**	Прямое сохранение в ОЗУ	**(k) ← Rr**	Нет	2
ST	**X, Rr**	Косвенное сохранение в ОЗУ	**(X) ← Rr**	Нет	2
ST	**X+, Rr**	Косвенное сохранение в ОЗУ с постинкрементом	**(X) ← Rr, X ← X + 1**	Нет	2
ST	**–X, Rr**	Косвенное сохранение в ОЗУ с предекрементом	**X ← X – 1, (X) ← Rr**	Нет	2
ST	**Y, Rr**	Косвенное сохранение в ОЗУ	**(Y) ← Rr**	Нет	2
ST	**Y+, Rr**	Косвенное сохранение в ОЗУ с постинкрементом	**(Y) ← Rr, Y ← Y + 1**	Нет	2
ST	**–Y, Rr**	Косвенное сохранение в ОЗУ с предекрементом	**Y ← Y – 1, (Y) ← Rr**	Нет	2
STD	**Y+q, Rr**	Косвенное сохранение в ОЗУ со смещением	**(Y + q) ← Rr**	Нет	2
ST	**Z, Rr**	Косвенное сохранение в ОЗУ	**(Z) ← Rr**	Нет	2
ST	**Z+, Rr**	Косвенное сохранение в ОЗУ с постинкрементом	**(Z) ← Rr, Z ← Z + 1**	Нет	2
ST	**–Z, Rr**	Косвенное сохранение в ОЗУ с предекрементом	**Z ← Z – 1, (Z) ← Rr**	Нет	2
ST	**Z+q, Rr**	Косвенное сохранение в ОЗУ со смещением	**(Z + q) ← Rr**	Нет	2
LPM	Нет	Загрузка байта из памяти программ	**R0 ← (Z)**	Нет	3
LPM	**Rd, Z**	Загрузка байта из памяти программ	**Rd ← (Z)**	Нет	3
LPM	**Rd, Z+**	Загрузка байта из памяти программ с постинкрементом	**Rd ← (Z), Z ← Z + 1**	Нет	3
ELPM	Нет	Расширенная загрузка байта из памяти программ	**R0 ← (RAMPZ:Z)**	Нет	3
ELPM	**Rd, Z**	Расширенная загрузка байта из памяти программ	**Rd ← (RAMPZ:Z)**	Нет	3
ELPM	**Rd, Z+**	Расширенная загрузка байта из памяти программ с постинкрементом	**Rd ← (RAMPZ:Z), Z ← Z + 1**	Нет	3

1	2	3	4	5	6
SPM	Нет	Запись в память программ	(Z) ← R1:R0	Нет	–
IN	Rd, I/O	Чтение регистра ввода-вывода	Rd ← I/O	Нет	1
OUT	I/O, Rr	Запись в регистр ввода-вывода	I/O ← Rr	Нет	1
PUSH	Rr	Занесение содержимого РОН в стек	STACK ← Rr	Нет	2
POP	Rd	Извлечение из стека в РОН	Rd ← STACK	Нет	2
Команды работы с битами					
LSL	Rd	Логический сдвиг влево	Rd(n + 1) ← Rd(n), Rd(0) ← 0, C ← Rd(7)	Z, C, N, V, H, S	1
LSR	Rd	Логический сдвиг вправо	Rd(n) ← Rd(n + 1), Rd(7) ← 0, C ← Rd(0)	Z, C, N, V, H, S	1
ROL	Rd	Циклический сдвиг влево через флаг переноса	Rd(0) ← C, Rd(n + 1) ← ← Rd(n), C ← Rd(7)	Z, C, N, V, H, S	1
ROR	Rd	Циклический сдвиг вправо через флаг переноса	Rd(7) ← C, Rd(n) ← ← Rd(n + 1), C ← Rd(0)	Z, C, N, V, S	1
ASR	Rd	Арифметический сдвиг вправо	Rd(n) ← Rd(n + 1), n = 0...6	Z, C, N, V, S	1
SWAP	Rd	Поменять нибблы местами (перестановка тетрад)	Rd(3...0) ← Rd(7...4), Rd(7...4) ← Rd(3...0)	Нет	1
BSET	s	Установить флаг в регистре состояния SREG	SREG(s) ← 1	SREG(s)	1
BCLR	s	Сбросить флаг в регистре состояния SREG	SREG(s) ← 0	SREG(s)	1
SBI	I/O, b	Установить разряд в регистре ввода-вывода	I/O(b) ← 1	Нет	2
CBI	I/O, b	Сбросить разряд в регистре ввода-вывода	I/O(b) ← 0	Нет	2
BST	Rr, b	Сохранить разряд РОН во флаге T	T ← Rr(b)	T	1
BLD	Rd, b	Загрузить разряд из флага T в РОН	Rd(b) ← T	Нет	1
SEC	Нет	Установить флаг переноса	C ← 1	C	1
CLC	Нет	Сбросить флаг переноса	C ← 0	C	1
SEN	Нет	Установить флаг отрицательного числа	N ← 1	N	1
CLN	Нет	Сбросить флаг отрицательного числа	N ← 0	N	1
SEZ	Нет	Установить флаг нуля	Z ← 1	Z	1
CLZ	Нет	Сбросить флага нуля	Z ← 0	Z	1
SEI	Нет	Установить флаг разрешения прерываний	I ← 1	I	1
CLI	Нет	Сбросить флаг разрешения прерываний	I ← 0	I	1
SES	Нет	Установить флаг числа со знаком	S ← 1	S	1
CLS	Нет	Сбросить флаг числа со знаком	S ← 0	S	1

1	2	3	4	5	6
SEV	Нет	Установить флаг переполнения	**V** ← 1	**V**	1
CLV	Нет	Сбросить флаг переполнения	**V** ← 0	**V**	1
SET	Нет	Установить флаг **T**	**T** ← 1	**T**	1
CLT	Нет	Сбросить флаг **T**	**T** ← 0	**T**	1
SEH	Нет	Установить флаг внутреннего переноса	**H** ← 1	**H**	1
CLH	Нет	Сбросить флаг внутреннего переноса	**H** ← 0	**H**	1
NOP	Нет	Нет операции	Нет	Нет	1
BREAK	Нет	Сброс	См. описание конкретного микроконтроллера	Нет	1
SLEEP	Нет	Уменьшить энергопотребление	См. описание конкретного микроконтроллера	Нет	1
WDR	Нет	Сбросить сторожевой таймер	См. описание конкретного микроконтроллера	Нет	1

Условные обозначения:

Rd – результирующий (destination) и исходный РОН;
Rr – исходный (source) РОН;
I/O – регистр ввода-вывода;
b – разряд в регистре общего назначения или регистре ввода-вывода;
s – разряд (флаг) в регистре состояния **SREG**;
Rdl – регистры **R24**, **R26**, **R28**, **R30**;
X, **Y**, **Z** – регистры-указатели для косвенной адресации (**X** = **R27**:**R26**, **Y** = **R29**:**R28**, **Z** = **R31**:**R30**);
RAMPX, **RAMPY**, **RAMPZ** – регистры, связанные с регистрами-указателями **X**, **Y** и **Z** и обеспечивающие косвенную адресацию по всему объёму памяти данных (при размере памяти данных более 64К байт) и доступ к размещённым в памяти программ константам в микроконтроллерах с размером памяти программ более 64К байт;
PC – программный счётчик (Program Counter);
STACK – стек для хранения адресов возврата и содержимого регистров;
SP – указатель стека (Stack Pointer);
EIND – регистр, связанный с программным счётчиком и обеспечивающий косвенную адресацию по всему объёму памяти программ для микроконтроллеров с размером памяти программ более 64К байт;
K6 – константа (6 разрядов), может быть константное выражение;
K8 – константа (8 разрядов), может быть константное выражение;
k – адресная константа для программного счётчика (размер зависит от команды);
q – смещение при косвенной адресации (6 разрядов).

Разряды регистра состояния **SREG**:

C – флаг переноса;
Z – флаг нулевого значения;
N – флаг отрицательного значения;
V – флаг переполнения;
S = **N** ⊕ **V** – флаг для проверок со знаком при переполнении;
H – флаг полупереноса (переноса между третьим и четвёртым разрядами байта);
T – флаг для команд пересылки;
I – флаг глобального разрешения (запрещения) прерываний.

ПРАКТИЧЕСКАЯ РАБОТА №14. ИЗУЧЕНИЕ СПОСОБОВ АДРЕСАЦИИ ОПЕРАНДОВ В AVR-МИКРОКОНТРОЛЛЕРАХ.

Цель работы: изучение способов адресации операндов в AVR-микроконтроллерах.

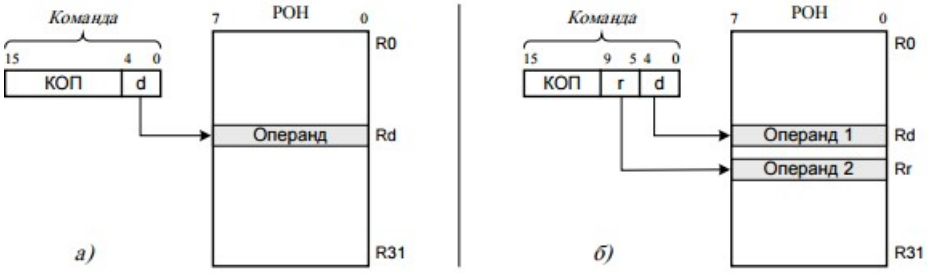
Основные сведения

В зависимости от количества используемых операндов возможны три типа команд AVR-микроконтроллера: безадресные, одноадресные и двухадресные. В первом типе команд присутствует только код операции (КОП), определяющий выполняемую командой функцию. В командах второго и третьего типов помимо кода операции содержится адресная часть, устанавливающая способ доступа соответственно к одному или двум участвующим в команде операндам (аргументам команды). Способ формирования адреса операнда, указание на который содержится в команде, называется адресацией (addressing). С помощью того или иного способа адресации вычисляется физический адрес, который поступает на шину адреса процессора для выбора ячейки памяти или регистра, используемых в команде.

В соответствии с типом адресуемой памяти способы адресации в AVR- микроконтроллерах можно разделить на способы адресации РОН и регистров ввода-вывода, способы адресации оперативной памяти данных (ОЗУ) и способы адресации памяти программ.

Возможность использования различных способов адресации позволяет сократить размер и время выполнения программ. Для адресации РОН и регистров ввода-вывода предусмотрен всего один режим – прямая регистровая адресация.

При прямой регистровой адресации РОН операндом является содержимое регистра общего назначения, указанного в команде. Команды с прямой регистровой адресацией могут адресовать один (Rd) или два (Rr и Rd) РОН. Во втором случае результат выполнения команды сохраняется в регистре Rd. Прямая регистровая адресация РОН применяется во всех арифметических и логических командах, а также в некоторых командах работы с битами, т. к. эти команды выполняются в АЛУ только над содержимым РОН. Команды, вторым операндом которых является константа, могут использовать в качестве первого операнда только регистры из старшей половины РОН (R16…R31).

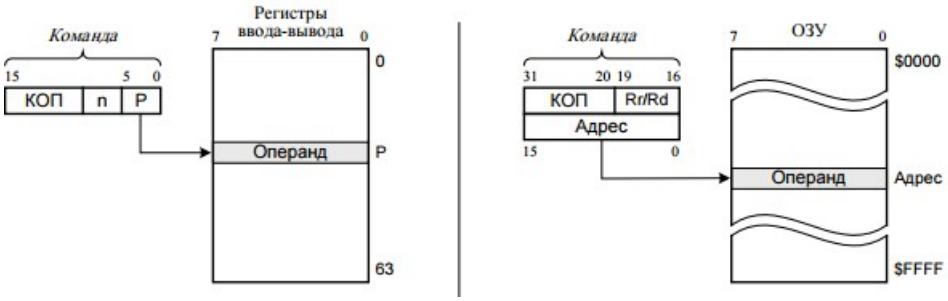


При прямой регистровой адресации регистра ввода-вывода операнд содержится в регистре ввода-вывода, указанном в команде. Адрес регистра ввода- вывода хранится в шести разрядах слова команды, где n определяет адрес регистра-источника или регистра-приёмника в РОН). Прямая регистровая адресация регистров ввода-вывода используется в командах чтения IN и записи OUT регистра ввода-вывода, а также в ряде других команд работы с регистрами ввода-вывода. Примеры использования прямой регистровой адресации:

```
; прямая регистровая адресация одного РОН
clr R1
; очистка всех разрядов регистра R1
; прямая регистровая адресация двух РОН
add R11, R12
; сложение содержимого регистров R11 и R12
```

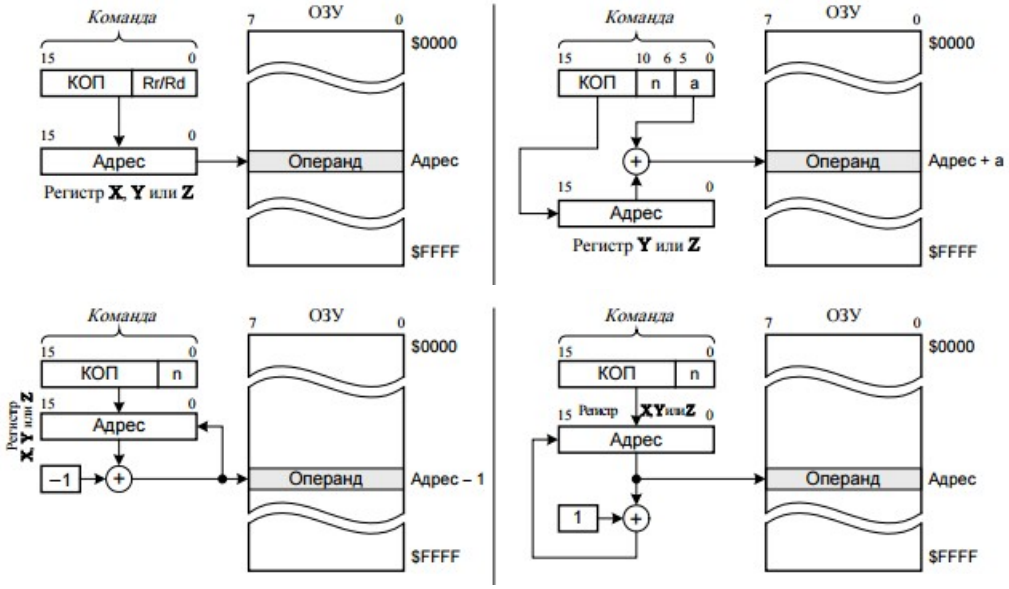
Для адресации оперативной памяти данных используются пять способов адресации: непосредственная, косвенная, косвенная со смещением, косвенная с предекрементом и косвенная с постинкрементом.

1. При непосредственной адресации оперативной памяти данных операндом является содержимое ячейки ОЗУ, адрес которой указан в команде. Адрес операнда содержится в 16 младших разрядах 32-разрядной команды, где Rr/Rd определяет адрес регистра-источника или регистра-приёмника в РОН). Непосредственная адресация используется в команде LDS (Load Direct from Data Space) загрузки из ОЗУ и в команде STS (Store Direct to Data Space) загрузки в ОЗУ. Зарезервировать байты в ОЗУ позволяет директива .byte. Для того чтобы на выделенную область памяти можно было ссылаться, директиве .byte должна предшествовать метка. Директива .byte имеет один параметр – количество выделяемых байт и может использоваться только в сегменте данных, определяемом с помощью директивы .dseg (data segment). Задать тре- буемое размещение выделяемой области памяти позволяет директива .org (origin – смещение). Начало программного сегмента указывается с помощью директивы .cseg (code segment). Директивы .dseg и .cseg не имеют параметров. Выделенные в оперативной памяти данных байты не инициализируются.



Пример использования непосредственной адресации:

```
; непосредственная адресация
.dseg ; сегмент данных (оперативная память данных)
.org $0065 ; по адресу $0065
cnt1: .byte 1 ; резервирование 1 байта для cnt1
.cseg ; программный сегмент (память программ) ;...
lds R10, cnt1 ; загрузка cnt1 в R10 2.
```



При косвенной адресации оперативной памяти данных операндом является содержимое ячейки ОЗУ, адрес которой находится в регистре X, Y или Z. Косвенная адресация используется в команде LD (Load Indirect) косвенной загрузки из ОЗУ и команде ST (Store Indirect) косвенной загрузки в ОЗУ. Например:

```
 ; косвенная адресация
.dseg ; сегмент данных (оперативная память данных)
cnt1: .byte 1 ; резервирование 1 байта для cnt1
.cseg ; программный сегмент (память программ)
;...
ldi R30, low(cnt1) ; загрузка в R30 младшего байта адреса
```

cnt1 ldi R31, high(cnt1) ; загрузка в R31 старшего байта адреса
cnt1 ld R1, Z ; загрузка cnt1 в R1, т. е. R1 < (R31:R30)

В приведённом примере использованы функции low и high, возвращающие соответственно младший и старший байты выражения (в данном случае – адреса ячейки памяти с символическим именем cnt1).

3. При косвенной адресации оперативной памяти данных со смещением адрес операнда в оперативной памяти данных вычисляется путём прибавления к содержимому регистра Y или Z смещения, указанного в команде. Смещение содержится в шести разрядах слова команды.

Косвенная адресация со смещением используется в команде LDD (Load Indirect with Displacement) косвенной загрузки из ОЗУ со смещением и в команде STD (Store Indirect with Displacement) косвенной загрузки в ОЗУ со смещением. Например:

```
; косвенная адресация со смещением
.dseg ; сегмент данных (оперативная память данных)
arr: .byte 5 ; резервирование 5 байт для массива arr
.cseg ; программный сегмент (память программ)
ldi R30,
low(arr) ; загрузка в R30 младшего байта адреса
arr ldi R31, high(arr) ; загрузка в R31 старшего байта адреса arr
;...
ldd R10, Z + 0 ; загрузка первого элемента массива arr в R10
ldd R11, Z + 1 ; загрузка второго элемента массива arr в R11
```

4. При косвенной адресации оперативной памяти данных с предекрементом (лат. decrementum – уменьшение, убыль) перед выполнением команды содержимое указанного в команде регистра X, Y или Z декрементируется (уменьшается на единицу); декрементированное содержимое регистра X, Y или Z является адресом операнда в оперативной памяти данных. Косвенная адресация с предекрементом используется в команде LD косвенной загрузки из ОЗУ и команде ST косвенной загрузки в ОЗУ. Например:

```
; косвенная адресация с предекрементом
ld R10, -Z ; Z <- Z – 1, R10 <- (Z)
```

5. При косвенной адресации оперативной памяти данных с постинкрементом (лат. incrementum – увеличение, рост) адресом операнда в оперативной памяти данных является содержимое регистра X, Y или Z, указанного в команде; после выполнения команды содержимое регистра X, Y или Z инкрементируется, т. е. увеличивается на единицу.

Косвенная адресация с постинкрементом используется в команде LD косвенной загрузки из ОЗУ и команде ST косвенной загрузки в ОЗУ. Например:

```
; косвенная адресация с постинкрементом
ld R10, Z+ ; R10 <- (Z), Z <- Z + 1
```

Для адресации памяти программ используется непосредственная адресация, косвенная адресация, относительная адресация, адресация константы и адресация константы с постинкрементом.

При непосредственной адресации памяти программ выполнение программы продолжается с адреса, указанного в команде. Непосредственная адресация памяти программ используется в командах JMP и CALL. При косвенной адресации памяти программ выполнение программы про- должается с адреса, содержащегося в регистре Z, т. е. в программный счётчик загружается содержимое регистра Z.

Косвенная адресация памяти программ используется в командах IJMP и ICALL. При относительной адресации памяти программ выполнение программы продолжается с адреса (PC + k + 1), где PC – содержимое программного счёт- чика; k – указанный в команде относительный адрес, который может принимать значения от –2048 до 2047.

Относительная адресация памяти программ используется в командах RJMP и RCALL. При адресации константы адрес байта константы содержится в регистре Z (рис. 20): старшие 15 разрядов занимает адрес ячейки памяти программ (от 0 до 32К); состояние младшего разряда (LSB) определяет выбор младшего (LSB = 0) или старшего (LSB = 1) байта адресуемой ячейки памяти.

Адресация константы в памяти программ используется в командах LPM (Load Program Memory), которая загружает адресованный регистром Z байт в указанный в команде регистр. Если

регистр-приёмник не указан (команда используется без операндов), байт загружается в регистр R0. Команда ELPM (Extended Load Program Memory) служит для загрузки константы из памяти программ объёмом более 64К слов. При этом для расширения регистра-указателя Z используется регистр RAMPZ, связанный с регистром Z. Задать данные в память программ позволяет директива .db (define bytes). Для того, чтобы на заданные ячейки памяти можно было ссылаться, директиве должна предшествовать метка.

Параметры директивы – последовательность выражений, разделённых запятыми; каждое выражение должно быть числом в диапазоне –128…255 или в результате вычисления давать результат в этом же диапазоне, в противном случае число усекается до байта. Директива .db раз- мещается в программном сегменте и может использоваться совместно с директивой .org. Задавать положение данных в памяти программ следует таким об- разом, чтобы была исключена возможность непреднамеренного перехода к выполнению их как команд микроконтроллера. Пример использования адресации константы в памяти программ:

ldi R31, high

Практическая часть

Составить программу сложения двух целых 8-разрядных чисел:

1) с использованием прямой регистровой адресации РОН. Результат сложения в этом и последующих пунктах задания сохранить в РОН. Значения операндов взять из задания лабораторной работы № 1 (числа A и B);

2) с использованием непосредственной адресации оперативной памяти данных. Для этого зарезервировать в ОЗУ байты под слагаемые с помощью ди- ректив .byte. Занести слагаемые в зарезервированные ячейки ОЗУ командой STS с непосредственной адресацией. Сложить операнды, предварительно загрузив их в РОН командой LDS с непосредственной адресацией;

3) с использованием косвенной адресации оперативной памяти данных. Адреса слагаемых в ОЗУ загрузить в регистры X и Y с помощью команды LDI и функций low и high. Занести слагаемые в зарезервированные ячейки ОЗУ командой ST с косвенной адресацией. Сложить операнды, предварительно загрузив их в РОН командой LD с косвенной адресацией;

4) с использованием косвенной адресации оперативной памяти данных со смещением. Для этого зарезервировать в ОЗУ байты под слагаемые в одной директиве .byte. Адрес начала блока данных загрузить в регистр Z с помощью команды LDI и функций low и high. Занести слагаемые в зарезервированные ячейки ОЗУ командой STD с косвенной адресацией со смещением. Сложить операнды, предварительно загрузив их в РОН командой LDD с косвенной адре- сацией со смещением;

5) с использованием косвенной адресации оперативной памяти данных с предекрементом. Для этого зарезервировать в ОЗУ байты под слагаемые в одной директиве .byte. В регистр Z загрузить адрес ячейки ОЗУ, следующей за блоком зарезервированных байтов. Занести слагаемые в зарезервированные ячейки ОЗУ командой ST с косвенной адресацией с предекрементом. Сложить операнды, предварительно загрузив их в РОН командой LD с косвенной адресацией ячеек ОЗУ с предекрементом;

6) с использованием косвенной адресации оперативной памяти данных с постинкрементом. Для этого зарезервировать в ОЗУ байты под слагаемые в одной директиве .byte. В регистр Z загрузить адрес начала блока зарезервиро- ванных байтов. Занести слагаемые в зарезервированные ячейки ОЗУ командой ST с косвенной адресацией с постинкрементом. Сложить операнды, предвари- тельно загрузив их в РОН командой LD с косвенной адресацией с постинкре- ментом;

7) с использованием адресации константы в памяти программ. Для этого задать слагаемые в памяти программ в одной директиве .db. Сложить операнды, предварительно загрузив их в РОН с помощью команды LPM. Для пересылки слагаемых между РОН использовать команду MOV;

8) с использованием адресации константы в памяти программ с постинкрементом. Для загрузки слагаемых в РОН использовать команду LPM с постинкрементом. В диалоговом окне AVR Simulator Options в разделе Device Selection установить тактовую частоту моделирования работы микроконтроллера, равную 8,0 МГц (поле Frequency).

Выполнить трансляцию и отладку созданных программ. По данным, выводимым после трансляции на закладке Build окна Output, проанализировать использование памяти программ (Program

memory usage) под код программы (Code) и константы (Constants), оценить объём неиспользованной (Unused) и общей занятой (Total) памяти. Занести эти сведения в отчёт.

При отладке программ использовать средства наблюдения за содержимым регистров и ячеек памяти. По полю Cycle Counter объекта Processor закладки I/O окна Workspace определить число тактов выполнения программы, по полю Stop Watch – время выполнения программы (до выполнения команды, организующей бесконечный цикл). Зафиксировать эти сведения в отчёте.

По результатам выполнения программ сделать выводы.

9) Оформить отчет. Отчёт должен содержать: титульный лист с указанием номера и названия работы, номера группы и фамилий выполнивших работу; цель работы; листинги трансляции программ и сведения, указанные в задании; схемы образования адреса для использованных способов адресации.

Контрольные вопросы

1. Адресация РОН и регистров ввода-вывода AVR-микроконтроллеров.
2. Способы адресации памяти данных AVR-микроконтроллеров.
3. Способы адресации памяти программ AVR-микроконтроллеров.
4. Особенности выполнения арифметических и логических операций в AVR-микроконтроллерах.
5. Назначение и использование регистров X, Y и Z.

ПРАКТИЧЕСКАЯ РАБОТА №15. ЭЛЕМЕНТАРНЫЕ ПРИЕМЫ ПРОГРАММИРОВАНИЯ (ВЕТВЛЕНИЯ И ЦИКЛЫ).

Цель работы: изучение принципов реализации типовых алгоритмических структур на примере ветвлений и циклических программ.

Основные сведения

Любая процедура управления или обработки данных представляет собой совокупность некоторых алгоритмических структур, с помощью которых выполняются требуемые операции. Наиболее распространёнными алгоритмическими структурами являются ветвления (branching) и циклы (loop). Ветвления используются для выполнения различных частей программы (разделения ветвей алгоритма) в зависимости от некоторых условий. В циклах одна и та же операция выполняется над содержимым нескольких последовательно расположенных в памяти ячеек или элементов данных. Использование циклических программ целесообразно при обработке массивов, таблиц и подобных по структуре данных. Числом повторений цикла управляют счётчики, а обрабатываемый при данном проходе цикла элемент определяется с помощью индекса или указателя.

В циклической программе можно выделить четыре основных блока.

1. Блок инициализации (от лат. initium – начало), в котором производится присвоение начальных значений переменным, счётчикам, индексам и указателям. Указатели представляют собой адреса данных в памяти.
2. Блок обработки, в котором выполняются требуемые вычисления, т. е. одинаковые повторяющиеся действия над различными последовательно расположенными в памяти данными.
3. Блок управления циклом, в котором изменяются значения счётчиков и индексов (указателей) перед выполнением следующей повторяющейся операции, а также производится проверка условия выхода из цикла.
4. Заключительный блок, в котором производится сохранение полученных результатов.

Блоки 2 и 3 составляют тело цикла (loop body). Для повышения быстродействия и сокращения размера циклических программ следует разгружать тело цикла от операций, которые могут быть выполнены за его пределами.

Для организации циклических программ, а также ветвлений в программах используются команды безусловных и условных переходов. Кроме того, для построения циклов могут применяться специальные команды циклов, выполняющие несколько действий одновременно (в системе команд AVR- микроконтроллеров отсутствуют).

Команды безусловных переходов JMP, RJMP, IJMP и EIJMP передают управление по указанному в команде адресу памяти программ.

Команда JMP (Jump – переход) позволяет передавать управление внутри всего объёма памяти программ.

Команда RJMP (Relative Jump – относительный переход) обеспечивает переход в пределах ±2К слов (±4К байт) относительно текущего содержимого программного счётчика.

По команде IJMP (Indirect Jump – косвенный переход) выполняется косвенный переход по адресу, указанному регистром Z; максимальное смещение составляет 64К слов (128К байт).

Команда EIJMP (Extended Indirect Jump – расширенный косвенный переход) обеспечивает косвенный переход по всему объёму памяти программ; для расширения программного счётчика используется регистр EIND.

При выполнении команд безусловных переходов в программный счётчик загружается адрес ячейки памяти программ, на которую передаётся управление.

Команды условных переходов передают управление по указанному адресу памяти программ в случае выполнения некоторых условий.

Команды BRxx (Branch if … – перейти, если …) выполняют переход на расстояние –64…+63 слова относительно текущего содержимого программного счётчика по результатам проверки разрядов регистра состояния SREG (кодов или флагов условий).

Регистр состояния SREG находится в адресном пространстве регистров ввода-вывода. Коды условий (C, Z, N, V, S, H) формируются в регистре состояния при выполнении арифметических, логических команд и команд работы с битами и представляют собой признаки результата операции.

Разряд C (carry – перенос) устанавливается, если при выполнении команды был перенос из старшего разряда результата.

Разряд Z (zero – нуль) устанавливается, если результат выполнения команды равен нулю.

Разряд N (negative – отрицательный результат) устанавливается, если старший значащий разряд результата равен 1 (правильно показывает знак результата, если не было переполнения разрядной сетки числа со знаком).

Разряд V (overflow – переполнение) устанавливается, если при выполнении команды произошло переполнение разрядной сетки числа со знаком.

Разряд $S = N \oplus V$ (sign – знак) правильно показывает знак результата при переполнении разрядной сетки числа со знаком.

Разряд H (half carry – полуперенос) устанавливается, если при выполнении команды был перенос из третьего разряда результата.

Для организации ветвлений при сравнении операндов команды BRxx ис- пользуются совместно с командами CP (Compare) сравнения содержимого двух РОН, CPC (Compare with Carry) сравнения с учётом признака переноса и CPI (Compare with Immediate) сравнения с константой. Команды ветвления BRxx отличаются для операндов без знака и со знаком. Числа без знака представляются прямым кодом, числа со знаком – дополнительным кодом.

Команды условных переходов, используемые для ветвлений при сравнении операндов, сведены в следующей таблице.

Условие	Логическое выражение	Команда		Операнды
		сравнения	перехода	
Rd > Rr	Z·(N ⊕ V) = 0	CP Rr, Rd	BRLT	со знаком
	C + Z = 0	CP Rr, Rd	BRLO	без знака
Rd ≥ Rr	(N ⊕ V) = 0	CP Rd, Rr	BRGE	со знаком
	C = 0	CP Rd, Rr	BRSH/BRCC	без знака
Rd = Rr	Z = 1	CP Rd, Rr	BREQ	со знаком, без знака
Rd ≠ Rr	Z = 0	CP Rd, Rr	BRNE	со знаком, без знака
Rd ≤ Rr	Z + (N ⊕ V) = 1	CP Rr, Rd	BRGE	со знаком
	C + Z = 1	CP Rr, Rd	BRSH	без знака
Rd < Rr	(N ⊕ V) = 1	CP Rd, Rr	BRLT	со знаком
	C = 1	CP Rd, Rr	BRLO/BRCS	без знака

К командам условных переходов также относится команда CPSE (Compare and Skip if Equal – сравнить и пропустить, если равно), которая сравнивает содержимое двух РОН и пропускает следующую за ней команду, если содержимое одинаково.

Команды SBRS, SBRC, SBIS, SBIC (Skip if Bit in Register [I/O Register] is Set [Cleared] – пропустить, если разряд в регистре общего назначения [ввода- вывода] установлен [сброшен]) пропускают следующую команду в случае выполнения соответствующего условия. При обработке массивов в циклических программах эффективно использование косвенной адресации памяти данных с предекрементом и постинкрементом, а также косвенной адресации памяти дан- ных со смещением. На рисунке ниже приведён фрагмент программы, в которой число 100 заносится в ячейки массива из пяти байт. Для проверки условия выхода из цикла и передачи управления используется команда BRNE. Предел повторений цикла равен 5, шаг равен –1, параметр цикла (счётчик) содержится в регистре R16.

```
; ...
array:    .byte 5       ; 5 байт для массива array
; ...
   ldi    R16, 5        ; предел повторений цикла
   ldi    R17, 100      ; число, заносимое в массив array
   ldi    R18, 1

   ldi    R30, low(array)    ; младший байт адреса массива array
   ldi    R31, high(array)   ; старший байт адреса массива array

loop:                   ; тело цикла
   st     Z, R17        ; занесение числа 100 в массив array
   add    R30, R18      ; адрес следующего байта массива array
   sub    R16, R18      ; счётчик числа проходов, шаг равен -1
   brne   loop          ; повторить, если счётчик не равен нулю

; ...
```

Практическая часть.

1. Дополнить фрагмент программы необходимыми директивами. Изменить число, заносимое в ячейки массива, в соответствии с заданным вариантом. Выполнить программу в пошаговом режиме с помощью симулятора-отладчика.

2. Произвести изменения в программе: заменить команды ADD (сложение) и SUB (вычитание) на INC (инкремент) и DEC (декремент) соответственно.

№ вар.	Число	Массив I	Массив II	№ вар.	Число	Массив I	Массив II
1	99	13; 78; 1; 24; 18	81; 10; 201; 33; 8	**16**	84	65; 2; 43; 10; 125	84; 95; 5; 116; 48
2	98	5; 61; 75; 17; 27	42; 137; 72; 9; 53	**17**	83	14; 23; 83; 30; 66	47; 50; 36; 21; 74
3	97	33; 44; 29; 81; 20	7; 100; 38; 49; 99	**18**	82	34; 18; 136; 27; 5	94; 52; 47; 85; 21
4	96	24; 31; 6; 55; 71	30; 127; 23; 8; 17	**19**	81	23; 75; 30; 15; 41	110; 4; 39; 40; 33
5	95	68; 41; 25; 13; 57	48; 4; 15; 36; 121	**20**	80	71; 52; 19; 24; 88	37; 44; 26; 60; 18
6	94	45; 55; 2; 109; 33	9; 57; 15; 22; 207	**21**	79	49; 117, 29; 6; 21	51; 14; 57; 23; 48
7	93	23; 13; 67; 39; 48	47; 180; 3; 10; 55	**22**	78	83; 16; 54; 27; 30	94; 35; 76; 55; 81
8	92	34; 92; 8; 20; 71	36; 76; 23; 99; 40	**23**	77	37; 65; 29; 86; 24	81; 23; 70; 64; 32
9	91	28; 0; 139; 36; 17	128; 35; 5; 68; 72	**24**	76	51; 36; 48; 25; 80	78; 94; 8; 24; 128
10	90	61; 40; 22; 27; 66	59; 31; 129; 18; 63	**25**	75	13; 41; 27; 82; 77	53; 67; 15; 56; 30
11	89	7; 56; 29; 16; 104	87; 23; 90; 44; 62	**26**	74	94; 2; 17; 38; 45	17; 0; 49; 69; 32
12	88	49; 24; 49; 84; 15	75; 3; 12; 64; 227	**27**	73	6; 60; 73; 18; 44	100; 22; 37; 9; 56
13	87	51; 33; 19; 48; 80	145; 26; 1; 13; 88	**28**	72	48; 14; 23; 50; 65	62; 58; 46; 59; 33
14	86	67; 30; 25; 52; 38	35; 62; 8; 59; 46	**29**	71	31; 52; 17; 24; 78	3; 88; 53; 162; 72
15	85	120; 36; 7; 10; 45;	53; 47; 35; 62; 81	**30**	70	66; 70; 42; 13; 29	42; 15; 76; 38; 86

3. В последнем варианте программы исключить команду INC за счёт использования косвенной адресации памяти данных с постинкрементом.

4. Изменить порядок подсчёта числа проходов цикла, задав изменение параметра цикла (счётчика) не с предела, а с нуля.

5. Составить программу пересылки массива из памяти программ в память данных. Массив в памяти программ задать в директиве .db; значения элементов массива взять из таблицы (Массив I) в соответствии с заданным вариантом.

6. Составить программу суммирования элементов массива. Значения элементов массива задать аналогично п. 5.

7. Составить программу поэлементного сложения двух массивов. Значения элементов массивов взять из таблицы (Массив I и Массив II) в соответствии с заданным вариантом.

8. Составить программу слияния двух массивов, где в результате попарного сравнения двух элементов из разных массивов образуется новый массив из наибольших элементов. Значения элементов массивов взять аналогично п. 7.

9. Оформить отчет. Отчёт должен содержать: титульный лист с указанием номера и названия работы, номера группы и фамилий выполнивших работу; цель работы; листинги трансляции программ в соответствии с заданием.

Контрольные вопросы

1. Наиболее распространённые алгоритмические структуры.
2. Назначение и общая структура циклических программ.
3. Команды, используемые для организации ветвлений и циклов.
4. Назначение и выполнение команд безусловных переходов.
5. Назначение и выполнение команд условных переходов.
6. Использование кодов (флагов) условий в командах условных переходов.

ПРАКТИЧЕСКАЯ РАБОТА №16. ЭЛЕМЕНТАРНЫЕ ПРИЕМЫ ПРОГРАММИРОВАНИЯ (ПОДПРОГРАММЫ).

Цель работы: изучение принципов организации подпрограмм и передачи параметров; освоение команд, обеспечивающих взаимодействие вызывающей программы с подпрограммой.

Основные сведения

В большой и сложной программе можно выделить последовательности команд, выполняющие некоторые законченные функции (процедуры). Если такие последовательности команд оформить в виде отдельных модулей – подпрограмм (routine, subroutine), то в программе эти команды могут быть заменены вызовами соответствующих подпрограмм. Такое модульное (структурное) программирование даёт следующие преимущества:

− ускоряется и упрощается отладка всей программы;

− подпрограммы, реализующие универсальные функции, могут использоваться при разработке других программ;

− в одной программе могут быть объединены модули, полученные после трансляции программ, написанных на разных языках программирования.

Для взаимодействия вызывающей программы с подпрограммой необходимо выполнение следующих условий:

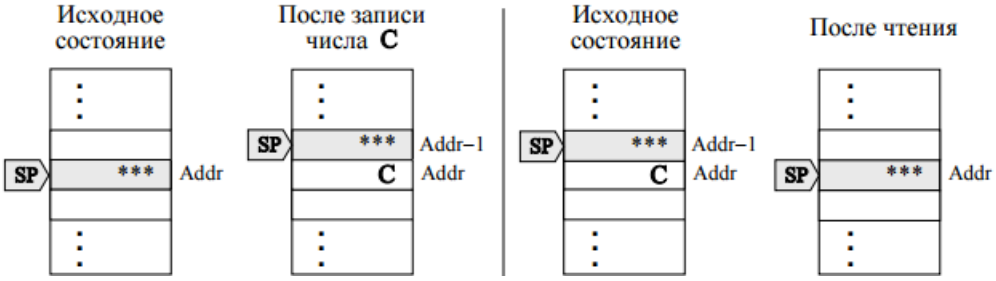
− вызывающей программе должно быть известно положение подпрограммы в общей структуре программы;

− должны быть определены способы вызова подпрограммы и возврата из неё;

− должен быть выбран способ обмена данными между вызывающей программой и подпрограммой.

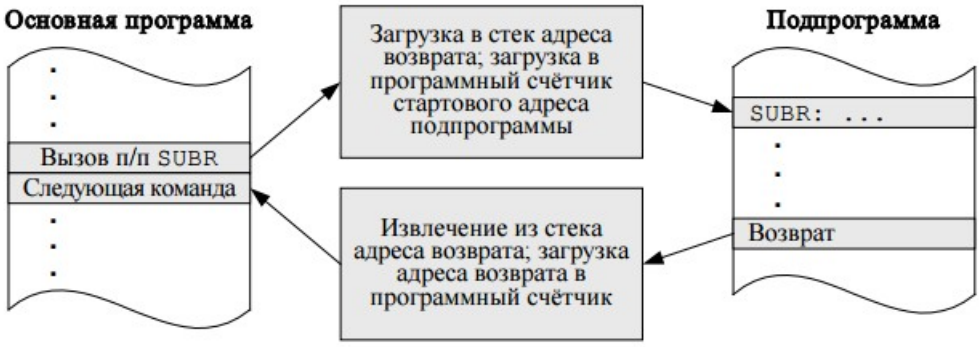
Положение подпрограммы в общей структуре программы определяется по её имени. Имя подпрограммы на ассемблере, задаваемое символической меткой, является стартовым адресом исполняемой части подпрограммы. Вызов подпрограммы подразумевает передачу в неё управления ходом выполнения команд. Передача управления осуществляется путём загрузки в программный счётчик (PC) стартового адреса подпрограммы. При этом необходимо обеспечить сохранение адреса возврата из подпрограммы, т. е. адреса команды, следующей за командой вызова подпрограммы. Для хранения адресов возврата из подпрограмм используется стек (stack). Стек – специальным образом организованная последовательность ячеек памяти с дисциплиной обслуживания «последним пришёл – первым вышел» (LIFO: Last-In – First-Out). При занесении в стек новых данных предыдущие данные сохраняются, но становятся временно недоступными («опускаются»). Выгружаются из стека («поднимаются», «выталкиваются») данные в обратном порядке.

Стек может быть реализован как аппаратно, так и программно. Аппаратный стек (Hardware Stack) выполняется непосредственно в составе процессора в виде набора специальных регистров и, как правило, имеет небольшую глубину. Программный стек (Software Stack) организуется в оперативной памяти; глубина определяется размером и степенью использования памяти. Адрес очередной свободной ячейки стека содержится в специальном регистре – указателе стека SP (Stack Pointer). При записи в стек число помещается в ячейку с адресом, содержащимся в указателе стека, после чего содержимое указателя стека уменьшается на единицу. При чтении из стека производится выборка содержимого ячейки по адресу, на единицу большему содержимого указателя стека. Таким образом, при записи стека и чтении из него содержимое указателя стека изменяется.



Механизм использования стека для хранения адресов возврата из подпрограмм состоит в следующем. Когда в программе встречается команда вызова подпрограммы, в ячейку памяти, определяемую указателем стека, записывается адрес возврата; значение указателя стека декрементируется; в программный счётчик загружается стартовый адрес подпрограммы. Далее выполняются команды подпрограммы. Когда в подпрограмме встречается команда возврата в вызывающую программу, значение указателя стека инкрементируется; сохра- нённый в стеке адрес

возврата загружается в программный счётчик. Использование стека для работы с подпрограммами позволяет одной подпрограмме вызывать другую, т. е. обеспечивает возможность вложения подпрограмм. Глубина вложения подпрограмм ограничивается размером стека.



В большинстве AVR-микроконтроллеров стек размещается в оперативной памяти. Указатель стека представляет собой пару 8-разрядных регистров SPH (старший байт указателя стека) и SPL (младший байт указателя стека), находящихся в адресном пространстве регистров ввода-вывода. В микроконтроллерах, размер оперативной памяти которых не превышает 256 байт, для хранения указателя стека используется только один регистр SPL (SP). Микроконтроллеры, не имеющие оперативной памяти, содержат трёхуровневый аппаратный стек. Организуемый в оперативной памяти стек «растёт» от старших адресов к младшим. Учитывая, что начальное значение указателя стека после сброса микроконтроллера равно нулю, в инициализирующей (начальной) части программы необходимо произвести его установку, если предполагается использование хотя бы одной подпрограммы. При организации стека во внутренней оперативной памяти это может быть сделано, например, следующим образом:

```
ldi    R16, low(RAMEND)         ; младшая часть адреса RAMEND
out    SPL, R16                 ; инициализация SPL
ldi    R16, high(RAMEND)        ; старшая часть адреса RAMEND
out    SPH, R16                 ; инициализация SPH
```

где команда OUT осуществляет запись содержимого регистра общего назначения в регистр ввода-вывода; RAMEND – символическое имя адреса последней ячейки внутренней оперативной памяти. Для того, чтобы использовать в программе символические имена адресов периферийных устройств AVR-микроконтроллера, необходимо при помощи директивы .include подключить файл определения адресов периферийных устройств (inc-файл).

Например, для микроконтроллера ATmega8535 следует подключить файл m8535def.inc: .include "m8535def.inc" К подключаемому inc-файлу должен быть указан путь в поле Include Path окна AVR Assembler, вызов которого осуществляется командой AVR Assembler Setup меню Project. При использовании директивы .include нет необходимости включать в программу директиву .device, т. к. она содержится в соответствующем inc-файле. Избежать появления в листинге трансляции текста inc-файла позволяет директива .nolist отключения листинга. Директива .nolist используется совместно с директивой .list включения листинга (по умолчанию режим генерации листнга включён). Исключение inc-файла из листинга трансляции может быть произведено следующим образом:

```
.nolist                      ; отключить генерацию листинга
.include "m8535def.inc"      ; подключить inc-файл
.list                        ; включить генерацию листинга
```

Вызов подпрограммы на языке ассемблера AVR-микроконтроллеров осуществляется командами RCALL, ICALL, CALL и EICALL.

Команда RCALL (Relative Call – относительный вызов) обеспечивает вызов подпрограммы, смещение начального адреса которой относительно текущего значения программного счётчика лежит в пределах ±2К слов (±4К байт), с помощью относительной адресации памяти программ.

Команда ICALL (Indirect Call – косвенный вызов) производит косвенный вызов подпрограммы по адресу памяти программ, содержащемуся в регистре-указателе Z; максимальное смещение начального адреса подпрограммы составляет 64К слов (128К байт).

Команда CALL (Call – вызов) обеспечивает вызов подпрограммы из памяти программ размером до 4М слов (8М байт) с использованием непосредственной адресации.

Команда EICALL (Extended Indirect Call – расширенный косвенный вызов) позволяет выполнять косвенный вызов подпрограмм из памяти программ размером до 4M слов (8M байт).

При выполнении команд RCALL, ICALL, CALL и EICALL текущее значение программного счётчика заносится в стек (2 байта в микроконтроллерах с 16-разрядным про- граммным счётчиком или 3 байта в микроконтроллерах с 22-разрядным про- граммным счётчиком). Содержимое указателя стека (SPH:SPL) уменьшается соответственно на 2 или 3.

Примеры использования команд вызова подпрограмм:

```
   rcall    subr1       ; относительный вызов подпрограммы subr1

; косвенный вызов подпрограммы subr2
   ldi    R30, low(subr2)
   ldi    R31, high(subr2)
   icall               ; команда icall не имеет операндов
```

Для возврата из подпрограмм используется команда RET. При выполнении команды RET адрес возврата загружается из стека в программный счётчик. При этом содержимое указателя стека увеличивается на 2 или 3 в зависимости от разрядности программного счётчика (см. выше). Стек может также использоваться для сохранения содержимого РОН на время выполнения подпрограмм.

Для сохранения в стеке и извлечения из стека содержимого РОН служат команды PUSH и POP. Команда PUSH заносит содержимое регистра в стек по адресу, хранящемуся в указателе стека, при этом значение указателя стека уменьшается на единицу (SPH:SPL = SPH:SPL – 1). Команда POP выполняет обратные действия: значение указателя стека увеличивается на единицу (SPH:SPL = SPH:SPL + 1); содержимое ячейки памяти по адресу, хранящемуся в указателе стека, загружается в регистр.

Программы с использованием подпрограмм обычно начинаются с команды относительного перехода к основной программе, в которой прежде всего производится инициализация стека. Тексты подпрограмм обычно размещают перед текстом основной программы.

```
.nolist                      ; отключить генерацию листинга
.include "m8535def.inc"      ; подключить inc-файл
.list                        ; включить генерацию листинга

   rjmp  RESET               ; переход к основной программе

PODPR:                       ; подпрограмма PODPR
   ; ...
   ret                       ; возврат в основную программу

RESET:                       ; основная программа
   ldi   R16, low(RAMEND)
   out   SPL, R16            ; инициализация SPL
   ldi   R16, high(RAMEND)
   out   SPH, R16            ; инициализация SPH

   ; ...
   rcall    PODPR            ; вызов подпрограммы PODPR
   ; ...
```

При работе с подпрограммами важно обеспечить передачу параметров из вызывающей программы в подпрограмму и возврат результатов выполнения подпрограммы обратно в вызывающую программу. В ассемблере AVR-микроконтроллеров способы обмена данными между вызывающей программой и подпрограммой не формализованы. Для передачи параметров могут использоваться регистры общего назначения, ячейки оперативной памяти и стек. Передача параметров через регистры общего назначения пригодна только для небольшого числа параметров, т. к. число РОН ограничено и занятые под параметры регистры уже нельзя использовать в подпрограмме для участия в других вычислениях и хранения других данных. Однако это самый простой и прозрачный способ передачи параметров, обеспечивающий самый быстрый доступ к передаваемым параметрам.

Использование оперативной памяти для передачи параметров требует жёсткой регламентации правил обращения к ним. Например, можно организовать в памяти массив (таблицу) для непосредственного хранения значений передаваемых параметров или их адресов; адрес начала массива занести в РОН. Имея начальный адрес массива, вызывающая программа и подпрограмма смогут получить доступ к требуемым параметрам. При передаче параметров через стек перед вызовом подпрограммы передаваемые параметры заносятся в стек. Следует помнить, что после выполнения команды вызова подпрограммы в стек будет добавлен адрес возврата в вызывающую программу. Задавшись значением указателя стека в качестве базового адреса и используя косвенную адресацию

памяти данных со смещением, в подпрограмме можно получить доступ к содержащимся в стеке параметрам. Например, если в основной программе перед вызовом подпрограммы занести в стек значение некоторого параметра:

```
ldi R16, $33 ; R16 <- $33
push R16 ; сохранение содержимого регистра R16 в стеке
```

то в подпрограмме можно получить к нему доступ:

```
in R30, SPL ; младший байт указателя стека
in R31, SPH ; старший байт указателя стека
ldd R20, Z+3 ; загрузка числа $33 из стека в регистр R20 (команда IN служит для считывания содержимого регистра ввода-вывода в РОН).
```

Аналогично можно использовать стек для хранения адреса массива передаваемых параметров, расположенного в оперативной памяти данных.

Практическая часть

1. Составить программу, использующую для вызова подпрограммы команду RCALL. Для передачи параметров в подпрограмму использовать регистры общего назначения. Выполнить программу в пошаговом режиме, отслеживая изменение содержимого программного счётчика, указателя стека, а также занесение в стек адреса возврата из подпрограммы.
2. Выполнить задание п. 1, используя стек для передачи параметров в подпрограмму. Проследить в симуляторе занесение передаваемых параметров в стек.
3. Выполнить задание п. 2, применив для вызова подпрограммы команду ICALL.
4. Оформить отчет. Отчёт должен содержать: титульный лист с указанием номера и названия работы, номера группы и фамилий выполнивших работу; цель работы; листинги трансляции программ в соответствии с заданием.

Контрольные вопросы

1. Условия взаимодействия вызывающей программы и подпрограммы.
2. Принципы организации и назначение стека.
3. Механизм вызова подпрограмм.
4. Команды работы с подпрограммами на ассемблере микроконтроллеров семейства AVR.
5. Способы обмена данными между вызывающей программой и подпрограммой.

ПРАКТИЧЕСКАЯ РАБОТА №17. ПЕРИФЕРИЙНЫЕ УСТРОЙСТВА AVR-МИКРОКОНТРОЛЛЕРА

Цель работы: ознакомление с периферийными устройствами AVR-микроконтроллера; изучение назначения и возможностей применения портов ввода- вывода при реализации типичных процедур управления и обмена данными.

Основные сведения.

В состав микроконтроллера помимо процессорного ядра и блоков памяти входят периферийные устройства, обеспечивающие различные дополнительные функции. К внутренней периферии относятся аналого-цифровые и цифро-аналоговые преобразователи, порты, таймеры, контроллеры интерфейсов и другие устройства. Кроме внутренних периферийных устройств, дополнительные функции могут обеспечиваться подключением к микроконтроллеру внешних периферийных устройств, выполненных в виде самостоятельных микросхем. Внешняя периферия представлена различными запоминающими устройствами, внешними аналого-цифровыми и цифро-аналоговыми преобразователями, средствами сопряжения с каналом связи, устройствами отладки и рядом других.

Одними из наиболее важных внутренних периферийных устройств микроконтроллера являются порты ввода-вывода (I/O port), представляющие собой средства информационного обмена с внешними устройствами. В зависимости от способа обмена данными порты ввода-вывода бывают

последовательными и параллельными. Последовательный порт имеет одноразрядный формат и передаёт (принимает) информацию по принципу «один бит за другим». Параллельный порт имеет формат в несколько разрядов (обычно 8, 16 или 32), которые передаются или принимаются одновременно.

По виду синхронизации (под синхронизацией в данном случае понимается временно́е согласование работы устройств передачи и приёма информации) порты делятся на синхронные и асинхронные. Синхронными называют порты, передача и приём информации с помощью которых осуществляется с жёсткой временно́й синхронизацией. Асинхронными называют порты, которые передают (принимают) данные с различной скоростью.

С точки зрения возможностей использования различают специализированные и универсальные порты. Специализированные порты предназначены для реализации определённых интерфейсов обмена данными (RS232/422/485, USB, SCI, SPI, I 2 C, CAN и др.). Интерфейсы типа RS232/422/485 обеспечиваются универсальными асинхронными приёмопередатчиками – УАПП (UART – Universal Asinchronous Receiver-Transmitter). Синхронные интерфейсы реализуются универсальными синхронно-асинхронными приёмопередатчиками – УАПП (UART – Universal Sinchonous-Asinchronous Receiver-Transmitter). Универсальные порты позволяют программно управлять основными параметрами процесса обмена информацией (временными характеристиками, форматом данных и др.).

Порты ввода-вывода могут использоваться для обмена информацией в различных режимах: программно-управляемого обмена, обмена по прерываниям и обмена в режиме прямого доступа к памяти (ПДП). При программно- управляемом обмене (program-driven I/O) все операции ввода-вывода выполня- ются в соответствии с заложенной программой с проверкой готовности внешнего устройства к обмену. Обмен по прерываниям (interrupt-driven I/O) осуществляется с использованием механизма прерываний. Обмен в режиме ПДП (Direct Memory Access – DMA) осуществляется с помощью аппаратных средств независимо от процессора, который в данном случае только инициирует процесс ввода-вывода и управляет соответствующими ресурсами. Ввод и вывод данных с помощью портов, а также управление портами осуществляется посредством чтения и записи специальных регистров или ячеек памяти. Доступ к другим периферийным устройствам производится аналогично. Периферийные устройства могут иметь собственное адресное пространство или для них выделяется часть адресов пространства памяти. В последнем случае говорят о вводе-выводе, отображённом на память (memory-mapped I/O).

Порты ввода-вывода AVR-микроконтроллеров. В AVR-микроконтроллерах в зависимости от типа имеется от двух до шести универсальных двунаправленных портов ввода-вывода (порты A…F), содержащих семь или восемь сигнальных линий, которые соединены с соответствующими выводами БИС микроконтроллера. Например, в микроконтроллере ATmega8535 имеется четыре 8-разрядных порта A, B, C и D. Сигнальные линии порта A обозначаются PA0…PA7, порта B – PB0…PB7 и т. д. С каждым портом связаны три регистра ввода-вывода: регистр DDRx направления передачи данных, регистр PORTx данных порта и регистр PINx выводов порта, где x – имя порта (A…F). Содержимое разрядов регистра DDRx определяет направление передачи данных по каждой сигнальной линии порта (0 – порт используется для ввода данных, 1 – для вывода данных). Регистр PORTx позволяет задавать состояние сигнальных линий порта при выводе информации; регистр PINx – считывать состояние сигнальных линий порта при вводе информации.

С помощью указанных регистров реализуется требуемый протокол обмена данными (протокол – совокупность правил передачи кодированной информации). Основными процедурами при реализации некоторого протокола являются операции инициализации порта, вывода данных в порт, ввода данных из порта, а также формирования временных задержек. Инициализация порта состоит в назначении направления обмена данными по сигнальным линиям порта. Например, если сигнальные линии PA0, PA2, PA4, PA5 порта A служат для вывода данных, а сигнальные линии PA1, PA3, PA6, PA7 – для ввода данных, инициализация порта может быть произведена следующим образом: ldi R16, 0b00110101 out DDRA, R16 Вывод данных в порт производится путём записи значений в регистр PORTx, например: ldi R16, 0b10011101 out PORTA, R16 ; вывод содержимого регистра R16 в порт A Для изменения состояния только одного разряда порта ввода-вывода удоб- но использовать команды установки SBI (Set Bit in I/O Register) и сброса CBI (Clear Bit in I/O Register) бита в регистре ввода-вывода. Ввод данных из порта производится путём чтения содержимого регистра PINx выводов порта,

например: in R16, PINA; считывание состояния порта A в регистр R16 Формирование временных задержек необходимо для обеспечения требуемых временных соотношений при обмене данными.
Временные задержки могут быть созданы включением в программу последовательностей пустых команд NOP, методом программных циклов и с помощью таймера. Первый способ служит для формирования коротких задержек (от долей до единиц микросекунд), второй и третий – более длительных задержек (до сотен миллисекунд). При реализации временной задержки методом программных циклов в некоторый РОН загружается число, которое затем на каждом проходе цикла уменьшается на единицу. Так продолжается до тех пор, пока содержимое регистра не станет равным нулю, что является условием выхода из цикла. Время задержки при этом определяется числом, загруженным в РОН, и временем выполнения команд, образующих цикл. При необходимости формирования задержек бóльшей длительности организуют вложенные циклы. Точная подстройка задержки достигается подбором значений чисел, заносимых в РОН, и добавлением команд NOP. Для удобства использования цикл формирования задержки целесообразно оформлять в виде подпрограммы.
Пример подпрограммы, реализующей задержку на 10 мс при тактовой частоте 4 МГц приведён на рисунке ниже. Для данного примера с учётом затрат времени на вызов подпрограммы (3 такта) и возврат из неё (4 такта) общая длительность задержки составит: 503·0,25 + 2·0,25 + ((3·0,25·239 + 2·0,25) + 3·0,25) + ((3·0,25·255 + 2·0,25) + + 3·0,25)·50 + ((3·0,25·255 + 2·0,25) + 2·0,25 + 4·0,25 = 10000 мкс = 10 мс.

```
DELAY_10_ms:      ; задержка на 10 мс при тактовой частоте 4 МГц
   ldi  R20, 240 ; 1 такт
   ldi  R21, 52  ; 1 такт

DLY:
   dec  R20      ; 1 такт
   brne DLY      ; 2 такта, если условие ИСТИННО; 1 такт, если ЛОЖНО
   dec  R21      ; 1 такт
   brne DLY      ; 2 такта, если условие ИСТИННО; 1 такт, если ЛОЖНО

   ret           ; 4 такта
```

Наиболее простой интерфейс обмена данными организуется для опроса двоичных датчиков и управления релейными элементами. Опрос двоичных датчиков представляет собой процедуру определения состояния элементов с выходным сигналом типа «да/нет», например, кнопок, сигнализаторов, контактов реле, концевых выключателей и т. п. Такой опрос производится путём чтения содержимого регистра PINx с определённой периодичностью или по сигналу прерывания при изменении состояния датчика. Если используемый источник двоичного сигнала имеет механические или электромеханические контакты, то при их замыкании возникает сложный переходной процесс, называемый «дребезгом контактов». При наличии дребезга сигнал с контакта может быть прочитан как случайная последовательность нулей и единиц. Устранить это явление можно схемотехническими средствами (например, с помощью буферного триггера), однако чаще это производится программным путём. Наибольшее распространение получили два программных способа ожидания установившегося значения: подсчёт заданного числа совпадающих значений сигнала и временна́я задержка. Первый способ состоит в многократном считывании сигнала с контакта. Подсчёт опросов, при которых установлено замыкание контакта, ведётся программным счётчиком. Если после серии удачных опросов встречается неудачный, опрос начинается с начала. Контакт считается устойчиво замкнутым, если последовало N удачных опросов. Число N подбирается экспериментально для каждого типа датчиков и лежит в пределах от 5 до 50. Устранение дребезга путём введения временной задержки заключается в том, что программа, обнаружив замыкание контакта, запрещает опрос состояния этого контакта на время, заведомо большее длительности переходного процесса. Длительность временно́й задержки подбирается экспериментально для каждого типа датчиков (типичные значения 1…10 мс) и реализуется подпрограммой формирования задержки. Управление релейными элементами, т. е. элементами с двумя устойчивыми состояниями (например, электромагнитными клапанами, электромеханическими приводами и т. п.), производится путём записи значений в регистр PORTx. Более сложные процедуры управления могут быть реализованы с помощью встроенных широтно-импульсных модуляторов и внешних цифро-аналоговых преобразователей. Большинство сигнальных линий портов ввода-вывода AVR-микро- контроллера имеют альтернативное назначение (входы АЦП, входы для сигналов внешних прерываний и др.), а также реализуют функции специализированных

портов ввода-вывода, например, универсального асинхронного приёмопе- редатчика, последовательного периферийного интерфейса SPI и др. УАПП служит для организации последовательных интерфейсов типа RS232/422/485. Интерфейс SPI используется для занесения программы в память программ AVR-микроконтроллера непосредственно в микропроцессорной системе (программирования в системе – In-System-Programming, ISP), а также может применяться для обмена данными с внешними устройствами. Режим ПДП при обмене данными в AVR-микроконтроллерах не предусмотрен.

Практическая часть.

1. Составить программу опроса двух кнопок, служащих соответственно для увеличения и уменьшения значения некоторого числа (диапазон изменения 0…255). Число поместить в РОН. Для ввода в микроконтроллер сигналов от кнопок использовать внешние прерывания INT0 и INT1 (сигнал прерывания INT0 поступает на вывод PD2 порта D, сигнал прерывания INT1 – на вывод PD3). Задать режим запуска внешних прерываний по положительному фронту. Дребезг контактов устранить программно с помощью временно́й задержки. На- копленное в РОН число выводить в порт B параллельным 8-разрядным кодом.
2. Проверить работу созданной программы с помощью отладочной платы STK500. Убедиться, что микроконтроллер ATmega8535 установлен в разъём SCKT3100A3. Произвести на плате следующие подключения: разъём ISP6PIN соединить с разъёмом SPROG3, разъём SWITCHES – с разъёмом PORTD с помощью плоского 10-жильного кабеля; разъём LEDS – с разъёмом PORTB с помощью плоского 10-жильного кабеля (красный провод должен быть подклю- чён к первому контакту). Перемычку OSCSEL установить в положение 2-3; установить перемычки VTARGET, AREF, RESET, XTAL1. В разъём CRYSTAL установить кварцевый резонатор (8,0 МГц). С помощью интерфейсного кабеля соединить разъём RS232 CTRL платы STK500 с COM-портом компьютера. Включить питание отладочной платы. Средства работы с платой STK500 в среде AVR Studio находятся в одно- имённом диалоговом окне, вызов которого осуществляется командой STK500/AVRISP/JTAG ICE → STK500/AVRISP/JTAG ICE меню Tools. Настройка параметров работы платы STK500 производится на закладке Board диалогового окна STK500. В поле VTarget установить напряжение питания микроконтроллера, равное 5 В. Нажать кнопку Write Voltages. Занесение программы в память микроконтроллера производится следующим образом. На закладке Program диалогового окна STK500 в поле Device выбрать из списка тип используемого микроконтроллера (ATmega8535). В разделе Programming mode выбрать способ программирования – программирова- ние в системе (ISP); убедиться, что установлены флажки Erase Device Before Programming (стереть кристалл перед программированием) и Verify Device After Programming (проверить кристалл после программирования). В разделе Flash указать в качестве источника загружаемой программы текущую програм- му симулятора-отладчика (Use Current Simulator/Emulator FLASH Memory). Можно также непосредственно указать путь к файлу-прошивке памяти про- грамм в поле Input HEX File. Для загрузки программы в память программ на- жать кнопку Program раздела Flash. Проверка содержимого памяти программ может быть произведена с помощью кнопки Verify, чтение содержимого памяти программ – с помощью команды Read. По окончанию программирования микроконтроллер начинает функционировать по загруженной программе.
3. Оформить отчет.Отчёт должен содержать: титульный лист с указанием номера и названия работы, номера группы и фамилий выполнивших работу; цель работы; листинг трансляции созданной программы.

Контрольные вопросы.

1. Типы и назначение периферийных устройств.
2. Виды и особенности портов ввода-вывода.
3. Режимы обмена информацией; их преимущества и недостатки.
4. Реализация портов ввода-вывода в AVR-микроконтроллерах.
5. Способы формирования временных задержек.
6. Опрос двоичных датчиков и управление релейными элементами.
7. Способы подавления дребезга контактов

ПРАКТИЧЕСКАЯ РАБОТА №18. ИЗУЧЕНИЕ СИСТЕМЫ ПРЕРЫВАНИЙ AVR-МИКРОКОНТРОЛЛЕРА.

Цель работы: изучение изучение системы прерываний на примере прерывания по переполнению встроенного таймера-счётчика AVR-микроконтроллера.

Основные сведения

При работе реальной микропроцессорной системы в ней или вне её могут произойти события, требующие немедленной реакции. Такая реакция обеспечивается процедурой прерывания (interrupt), которая состоит в том, что выполнение текущей программы приостанавливается, запоминается состояние на момент прерывания, выполняется другая программа, после чего восстанавливается сохранённое до прерывания состояние процессора и продолжается выполнение прерванной программы. Сигнал, вызвавший прерывание текущей программы, называется запросом на прерывание (interrupt request – IRQ); источник этого сигнала – источником прерывания; последовательность действий, выполняемая по запросу на прерывание, – обслуживанием прерывания, а выполняемая по прерыванию программа – подпрограммой обработки прерывания (interrupt handler, interrupt routine).

Различают два типа источников прерывания – аппаратные и программные. Источниками аппаратных прерываний служат внешние и внутренние периферийные устройства. Запросом на прерывание от внешнего источника является активный сигнал на соответствующем выводе процессора; источник прерывания определяется по выводу, на котором появляется такой сигнал. К источникам программного прерывания относятся специальные команды прерываний (trap) – управляемые программные прерывания и особые условия (exception – исключение) – неуправляемые программные прерывания, являющиеся реакцией процессора на исключительную ситуацию, возникшую при выполнении некоторой команды (переполнение, деление на нуль и т. п.). Запросом на прерывание от программного источника является непосредственно команда прерывания или установка бита (битов), фиксирующих возникновение особого условия. Общее количество источников аппаратных и программных прерываний может быть различным – от единиц до нескольких десятков.

Процедура обслуживания прерываний по запросам от нескольких источников в различных процессорах выполняется по-разному. Тем не менее, основные принципы реализации механизма прерываний являются общими. Управление процедурой прерываний осуществляется специальными устройствами в составе аппаратного обеспечения процессора (контроллерами, схемами управления и т. п.). Основными средствами управления прерываниями являются:

− векторы прерываний;
− приоритеты прерываний;
− операция маскирования прерываний;
− флаги прерываний.

В микроконтроллерах указанные средства управления прерываниями реализуются следующим образом. Для управления прерываниями от N источников в адресном пространстве памяти программ выделяется специальная область из N ячеек памяти (или N блоков, состоящих из нескольких ячеек). В каждой из этих ячеек размещаются команды перехода к соответствующей подпрограмме обработки прерывания или (в случае блока из нескольких ячеек) непосредственно команды, которые необходимо выполнить по запросу на прерывание. Эти ячейки памяти (блоки) называются векторами прерываний (или просто векторами), адрес ячейки (пер- вой ячейки каждого блока) – адресом вектора прерывания. Таким образом, каждому источнику прерывания ставится в соответствие свой адрес вектора прерывания. Совокупность N векторов образует таблицу векторов прерываний, которая обычно располагается начиная с нулевого адреса памяти программ.

Приоритеты прерываний (interrupt priority) определяют очерёдность обслуживания запросов на прерывания. Введение приоритетов необходимо, если возможно одновременное (в течение одного периода тактовой частоты) поступление запросов на прерывание от различных источников или поступление нового запроса на прерывание во время обслуживания прерывания по ранее

поступившему запросу. Виды и структура приоритетов прерываний определяются архитектурой процессора. Наиболее простым способом задания приоритетов является последовательное присвоение значений приоритетов в таблице векторов прерываний от высшего к низшему. Высший приоритет всегда имеет аппаратный сброс; далее располагаются векторы прерываний от других источников.

Для того, чтобы запретить обслуживание неиспользуемых прерываний, служит операция маскирования. В зависимости от возможности маскирования источники прерывания делятся на маскируемые (maskable), прерывания от которых могут разрешаться или запрещаться, и немаскируемые (nonmaskable), прерывания от которых не могут запрещаться. Маскирование может быть общим и индивидуальным. При общем (глобальном) маскировании все прерыва- ния, кроме немаскируемых, запрещены независимо от их индивидуального маскирования. Индивидуальное маскирование позволяет запрещать (разрешать) прерывание от каждого источника отдельно. Флаги прерываний представляют собой разряды специальных регистров, устанавливающиеся при поступлении запроса на прерывание от некоторого источника.

Процедура обслуживания прерывания может быть упрощённо представлена состоящей из следующих этапов:

− приёма запросов на прерывание;

− арбитража прерываний;

− выполнения подпрограммы обслуживания прерывания.

При приёме запроса на прерывание от немаскируемого источника сразу осуществляется переход к следующему этапу его обслуживания – арбитражу. Запрос на прерывание от маскируемого источника обрабатывается по более сложному алгоритму. При поступлении запроса устанавливается соответствующий флаг прерывания. Далее проверяется наличие общего маскирования прерываний. Если режим общего маскирования установлен, то запросы на пре- рывания от всех маскируемых источников игнорируются и продолжается выполнение текущей программы. Если режим общего маскирования не задан, то запрещение или разрешение данного прерывания определяется наличием (отсутствием) индивидуального маскирования. Если данное прерывание замаскировано, то запросы на прерывание от данного источника запрещены и продолжается выполнение текущей программы. В противном случае прерывания от данного источника разрешены и для него начинается следующий этап обслуживания – арбитраж.

Арбитраж прерываний служит для определения прерывания с наивысшим приоритетом из очереди поступивших запросов на прерывание. После арбитража начинается выполнение выбранного запроса на прерывание. Выполнение прерывания состоит в переходе к подпрограмме обслуживания прерывания, её выполнении и возврату к выполнению текущей программы. Перед выполнением прерывания производится общее маскирование, т. е. запре- щение всех прерываний, кроме немаскируемых, а также очищается флаг обслуживаемого прерывания. Собственно выполнение прерывания начинается с обращения к вектору прерывания обслуживаемого источника. Обслуживаемое прерывание может быть прервано по запросам от источников, имеющих более высокий приоритет. Прерывания, для обслуживания которых прерывается выполнение подпрограммы обработки другого прерывания, называются вложенными. Процедура их обслуживания аналогична обслуживанию обычных прерываний; отличие состоит лишь в том, что прерывается выполнение не основной программы, а подпрограммы обработки прерывания от источника с более низким приоритетом.

В микропроцессорных системах механизм прерываний используется для обмена информацией с различными устройствами ввода-вывода. Такой способ обмена данными называется обменом по прерываниям. Типичными примерами запросов на прерывание являются запросы по готовности результата аналого- цифрового преобразования, готовности устройства к приёму (передаче) информации, переполнению некоторого регистра и т. п.

Использование механизма прерываний позволяет значительно повысить производительность системы при работе с медленно действующими устройствами, обслуживание которых в таком случае занимает процессорное время только при их готовности к обмену.

В AVR-микроконтроллерах механизм прерываний реализуется следующим образом. Управление прерываниями осуществляется с помощью схемы прерываний. Область векторов прерываний размещается в начале памяти программ; каждый вектор состоит из одной ячейки. При

необходимости область векторов прерываний может быть перемещена в другое место памяти программ. Прерывания с младшими адресами имеют бóльший уровень приоритета. Источниками всех прерываний являются аппаратные средства (внешние или внутренние); источники программных прерываний отсутствуют.

Все источники прерываний являются маскируемыми. Общее маскирование осуществляется очисткой бита I глобального разрешения прерываний в регистре состояния SREG. Количество векторов прерываний в AVR-микроконтроллерах составляет от 3 до 35 в зависимости от типа. Например, в микроконтроллере ATmega8535 имеется 21 вектор прерывания: 3 от внешних источников и 18 от внутренней периферии. Работа с внешними прерываниями осуществляется с помощью регистра управления GICR (General Interrupt Control Register) и регистра флагов GIFR (General Interrupt Flag Register), расположенных в адресном пространстве регистров ввода-вывода. Установка разряда 7 (INT1) регистра управления GICR разрешает внешнее прерывание INT1, установка разряда 6 (INT0) – внешнее прерывание INT0, установка разряда 5 (INT2) – внешнее прерывание INT2. Разряд 7 (INTF1) регистра флагов GIFR устанавливается при поступлении за- проса на прерывание INT1, разряд 6 (INTF0) – запроса на прерывание INT0; разряд 5 (INTF2) – запроса на прерывание INT2. Очистка установленных фла- гов прерываний производится записью единиц в соответствующие разряды регистра GIFR. Режим запуска внешних прерываний INT0 и INT1 задают разряды 0…3 (ISC00, ISC01, ISC10, ISC11) регистра управления MCUCR. Запись в разряды ISC00, ISC01 соответственно значений 0, 0 задаёт режим запуска внешнего прерывания INT0 по низкому уровню; 0, 1 – по отрицательному фронту; 1, 1 – по положительному фронту; значения 1, 0 не используются. Аналогично с по- мощью разрядов ISC10, ISC11 задаётся режим запуска внешнего прерывания INT1. Режим запуска внешнего прерывания INT2 задаётся разрядом 6 (ISC2) регистра управления и состояния MCUCSR: 0 – по отрицательному фронту; – по положительному фронту.

Для управления прерываниями от внутренних периферийных устройств в адресном пространстве регистров ввода-вывода также предусмотрены специальные регистры. Например, управление прерываниями по запросам от встроенных таймеров-счётчиков осуществляется с помощью регистра масок TIMSK (Timer/Counter Interrupt Mask Register) и регистра флагов TIFR (Timer/Counter Interrupt Flag Register). Кроме того, с каждым аппаратным устройством AVR- микроконтроллера ассоциированы управляющие регистры, расположенные в адресном пространстве регистров ввода-вывода.

Например, управление встроенным 8-разрядным таймером-счётчиком T/C0 (Timer/Counter0) осуществляется с помощью регистра TCCR0 (Timer/Counter0 Control Register) и регистра и TCNT0 (Timer/Counter0). Разряды 0…2 (CS00, CS01, CS02) регистра TCCR0 задают режим работы таймера-счётчика T/C0: при записи в разряды CS00, CS01, CS02 соответственно значений 0, 0, 0 таймер-счётчик остановлен; 1, 0, 0 – содержимое регистра TCNT0 инкрементируется на каждом такте тактового генератора; 0, 1, 0 – на каждом 8-м такте; 1, 1, 0 – на каждом 64-м такте; 0, 0, 1 – на каждом 256-м такте; 1, 0, 1 – на каждом 1024-м такте; значения 0, 1, 1 и 1, 1, 1 устанавливают режим подсчёта числа импульсов внешнего источника по отрицательному и положительному фронту соответственно. Таймер-счётчик T/C0 генерирует запрос на прерывание при переполнении регистра TCNT0. В регистре масок TIMSK прерыванию при переполнению таймера-счётчика T/C0 соответствует разряд 1 (TOIE0); в регистре флагов TIFR – разряд 1 (TOV0). Установка разряда TOIE0 разрешает прерывание по переполнению регистра TCNT0; флаг TOIF0 устанавливается при поступлении запроса на прерывание по переполнению регистра TCNT0.

Пример программы с использованием прерываний приведён на рисунке ниже. Программы с использованием прерываний начинаются с определения области векторов прерываний. Адреса векторов прерываний указываются символическими именами и помощью директив .org. По адресам векторов прерываний размещают команды относительного перехода к подпрограммам обработки прерываний, которые обычно располагают непосредственно после области векторов прерываний. Подпрограммы обработки прерываний завершаются командами reti возврата в основную программу. Команда reti выполняет те же действия, что и команда ret, а также восстанавливает бит I общего (глобального) разрешения прерываний в регистре состояния SREG.

В основной программе производится инициализация стека и прерываний. Инициализация прерываний осуществляется путём установки определённых разрядов в соответствующих регистрах

ввода-вывода; при этом в командах используются символические обозначения как самих регистров, так и отдельных их разрядов. После инициализации прерываний производится общее разрешение прерываний путём установки бита I в регистре состояния SREG. Для этого предусмотрена специальная команда sei (Set Global Interrupt Flag). Процедура обслуживания прерываний в AVR-микроконтроллерах выполняется согласно приведённому выше алгоритму. Для организации вложенных прерываний необходимо в подпрограмме обработки прерывания восстанавливать бит I общего разрешения прерываний в регистре состояния SREG.

Практическая часть

1. Дополнить программу, приведённую на рисунке, необходимыми директивами и командами. В подпрограмму обработки прерывания по таймеру-счётчику T/C0 поместить команду загрузки числа в РОН. Выполнить программу в пошаговом режиме. Проследить изменение содержимого стека при обработке прерывания, а также установку и сброс бита I общего разрешения прерываний и флага TOV0 прерывания по таймеру-счётчику T/C0. Для контроля содержимого регистров таймера-счётчика T/C0 раскрыть пункт TIMER_COUNTER_0 объекта I/O ATMEGA8535 закладки I/O окна Workspace.

```
; область векторов прерываний
.org $0000
   rjmp RESET              ; переход к основной программе
.org INT0addr
   rjmp EXT_INT0           ; внешнее прерывание INT0
.org OVF0addr
   rjmp TMR0 INT           ; прерывание по таймеру T/C0

; подпрограмма обработки внешнего прерывания INT0
EXT_INT0:
   ; ...
   reti                    ; возврат

; подпрограмма обработки прерывания по таймеру T/C0
TMR0_INT:
   ; ...
   reti                    ; возврат

RESET:                     ; основная программа

   ; инициализация стека
; ...

   ; инициализация внешнего прерывания INT0
   ldi  R16, (1<<ISC01)|(1<<ISC00)
   out  MCUCR, R16         ; по положительному фронту

   ldi  R16, (1<<INTF1)|(1<<INTF0)
   out  GIFR, R16          ; очистка флагов внешних прерываний

   ldi  R16, 1<<INT0
   out  GICR, R16          ; разрешение внешнего прерывания INT0

   ; инициализация прерывания по таймеру T/C0
   ldi  R16, 1<<CS00
   out  TCCR0, R16         ; деления частоты нет

   ldi  R16, 1<<TOIE0
   out  TIMSK, R16         ; разрешение прерывания по таймеру T/C0

   sei                     ; общее разрешение прерываний

forever:
   nop                     ; пустая команда (no operation)
   rjmp forever            ; бесконечный цикл

; ...
```

2. Исследовать процедуру обработки вложенных прерываний, внеся соответствующие изменения в программу. В подпрограмму обработки прерывания по внешнему прерыванию INT0 поместить команду очистки РОН, использующегося в подпрограмме обработки прерывания по таймеру-счётчику T/C0. В симуляторе после перехода в подпрограмму обработки прерывания по таймеру-счётчику T/C0 смоделировать поступление сигнала внешнего прерывания INT0. Для этого в симуляторе установить флаг INTF0 в регистре GIFR группы EXTERNAL_INTERRUPT объекта I/O ATMEGA8535 закладки I/O окна Workspace.

3. Проследить изменение содержимого стека при обработке вложенных прерываний.

4. Оформить отчет. Отчёт должен содержать: титульный лист с указанием номера и названия работы, номера группы и фамилий выполнивших работу; цель работы; листинги трансляции программ в соответствии с заданием.

Контрольные вопросы

1. Назначение прерываний.
2. Типы прерываний.
3. Средства управления прерываниями.
4. Операция маскирования прерываний.
5. Этапы процедуры прерывания.
6. Реализация прерываний в AVR-микроконтроллерах.

ПРАКТИЧЕСКАЯ РАБОТА №19. ПРОГРАММА УПРАВЛЕНИЯ ТАЙМЕР-СЧЕТЧИКОМ В РЕЖИМЕ ШИРОТНО-ИМПУЛЬСНОЙ МОДУЛЯЦИИ

Цель работы: Изучение проектирования таймера/счетчика в контроллере AVR на примере программы генерации синусоидального сигнала в режиме широтно-импульсной модуляции.

Краткие теоретические сведения

Таймер/счетчик 0

Таймер-счетчик 0 - модуль многофункционального одноканального 8разрядного таймера-счетчика. Функциональная схема таймера-счетчика представлена на рис. 1.

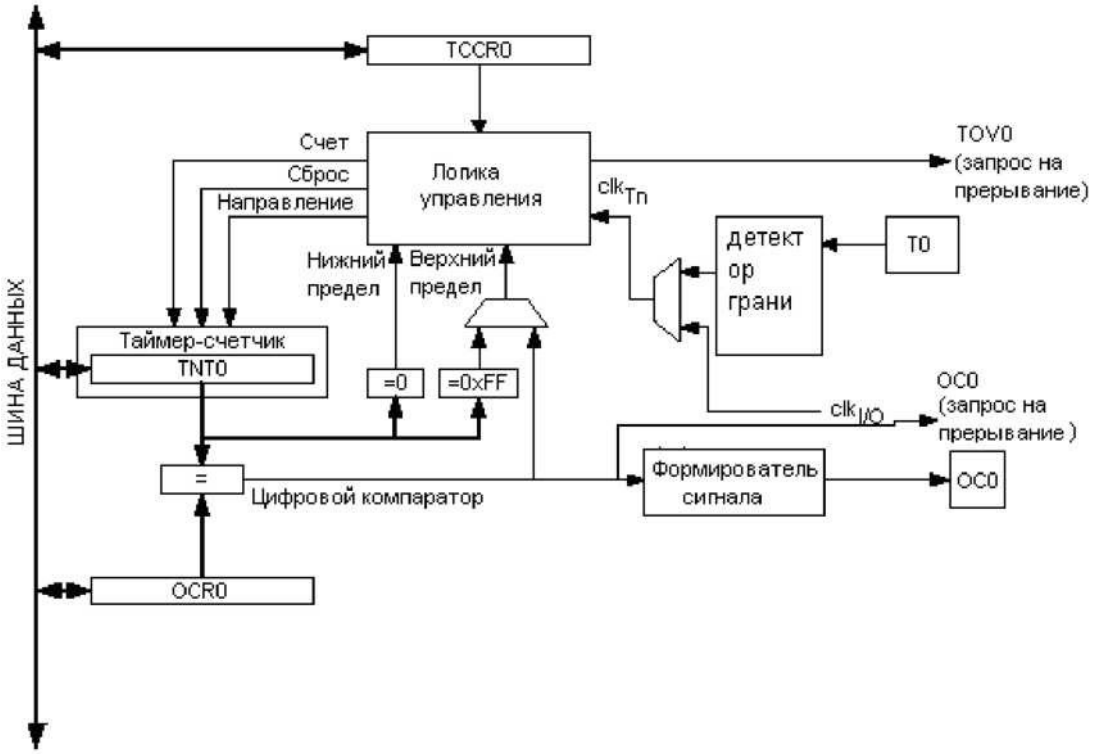


Рис. 1. Функциональная схема 8-разр. таймера-счетчика 0

Описание _регистров 8-разрядного таймера-счетчика 0

Регистр управления таймером-счетчиком 0 - TCCR0 В таблице 1 представлена структура регистра, режимы управления и начальные значения разрядов регистра TCCR0.

Табл. 1. Регистр TCCR0

Разряд	7	6	5	4	3	2	1	0
	F0C0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Чт./зап.	Чт.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.
Исх.зн.	0	0	0	0	0	0	0	0

Разряд 7 - FOCO: Принудительная установка результата сравнения

Строб FOCO не генерирует каких-либо прерываний, а также не вызывает сброс таймера в режиме CTC, где регистр OCRO задает верхний предел счета. Бит FOCO всегда считывается как 0.

Разряд 6, 3 - WGM01:0: Режим работы таймера-счетчика 0

Данные биты представлены в таблице 11, определяют: алгоритм счета счетчика, источник, который задает верхний предел счета и тип генерируемых прямоугольных импульсов.

Таблица 2. Описание бит, задающих режим работы таймера-счетчика 0

Номер реж.	WGM01	WGM00	Режим работы тайм/сч. 0	Верхний предел счета	Условие обновления регистра OCR0	Условие уст. флага TOV0
0	0	0	Нормальный	0xFF	Сразу после записи в регистр	Дост–ние макс–го зн. (0xFF)
1	0	1	ШИМ с фаз. кор–й	0xFF	Достижение верхнего предела счета	Дост–ние мин. зн. (0x00)
2	1	0	Сброс при совп–нии	OCR0	Сразу после записи в регистр	Дост–ние макс. зн. (0xFF)
3	1	1	Быстрый ШИМ	0xFF	Достижение верхнего предела счета	Дост–ние макс. зн. (0xFF)

Разряд 5:4 - COMOl, COMOO: Режим формирования выходного сигнала

Данные биты представлены в таблице 3, определяют алгоритм изменения сигнала на выводе OCO.

Таблица 3. Режимы формирования выходного сигнала в режимах работы таймера О без ШИМ

COM01	COM00	Описание
0	0	Функция обычного порта ввода-вывода. OCO
0	1	Переключение (инвертирование) OCO при каждом
1	0	Сброс OCO при каждом совпадении
1	1	Установка OCO при каждом совпадении

В таблице 4 приведено назначение бит COMOl, COMOO для режима работы таймера-счетчика О с быстрой ШИМ (WGMO1:O).

Таблица 4. Режимы формирования выходного сигнала в режиме таймера 0 с быстрым ШИМ(1)

COM01	COM00	Описание
0	0	Функция обычного порта ввода-вывода. OCO
0	1	Зарезервировано
1	0	Сброс OCO при совпадении, установка по
1	1	Установка OCO при совпадении, сброс по

В таблице 5 приведено действие бит COMOl, COMOO для режима ШИМ с фазовой коррекцией, заданного с помощью бит WGM01, WGM00.

Таблица 5. Режимы выходного сигнала в режиме ШИМ с фазовой коррекцией (1)

COM01	COM00	Описание
0	0	Функция обычного порта ввода-вывода. OCO
0	1	Зарезервировано
1	0	Сброс OCO при совпадении во время прямого счета.
1	1	Установка OCO при совпадении во время прямого

Разряд 2:0 - CS02:0: Настройка частоты синхронизации таймера

С помощью трех настроечных бит имеется возможность выбрать различные тактовые частоты, кратные исходной частоте синхронизации (см. табл. 6).

Таблица 6. Выбор частоты синхронизации таймера 0

CS02	CS01	CS00	Описание
0	0	0	Нет синхронизации. Таймер-счетчик О
0	0	1	clkTOS/1 (без предделения)
0	1	0	clkTOS/8 (с предделением)
0	1	1	clkTOS/32 (с предделением)
1	0	0	clkTOS/64 (с предделением)
1	0	1	clkTOS/128 (с предделением)
1	1	0	clkTOS/256 (с предделением)
1	1	1	clkTOS/1O24 (с предделением)

Регистр таймера-счетчика, разряды которого представлены в таблице 7, характеризуется двунаправленностью доступа к 8-разрядному счетчику таймера 0. Запись в регистр TCNT0 блокирует отработку возникающего совпадения на следующем после записи такте синхронизации таймера. Изменение содержимого счетчика (TCNT0) во время счета связано с риском потери результата сравнения между TCNT0 и регистром OCRO.

Таблица 7. Регистр таймера-счетчика - TCNT0

Разряд	7	6	5	4	3	2	1	0
TCNT0	TCNT0[7:0]							
Чт./зап.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.
Исх. знач.	0	0	0	0	0	0	0	0

Регистр порога сравнения, разряды которого представлены в таблице 8, содержит 8-разр. значение, которое непрерывно сравнивается цифровым компаратором со значением 8-разр. счетчика (TCNT0). Факт совпадения значений может использоваться для генерации прерывания по выполнению условия сравнения или для генерации прямоугольных импульсов на выводе OCO.

Таблица 8. Регистр порога сравнения - OCR0

Разряд	7	6	5	4	3	2	1	0
OCR0	OCR0[7:0]							
Чт./зап.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.
Исх. зн.	0	0	0	0	0	0	0	0

Таймер/счетчик 1

16-ти разрядный таймер-счетчик 1 предназначен для точного задания временных интервалов, генерации прямоугольных импульсов и измерения временных характеристик импульсных сигналов.

На рисунках и в тексте описания индекс “n” заменяет номер таймера- счетчика , а “x” заменяет наименование канала сравнения (A, или B). Однако при программировании необходимо использовать фактические номера и наименования.

В таблице 9 представлены разряды регистра A управления таймером- счетчиком 1 - TCCR1A.

В таблице 10 приведены режимы управления сигналов OCnA/OCnB.

В таблице 11 представлены разряды регистра B управления таймером- счетчиком 1 - TCCR1B.

Функциональная схема 16–разр. таймера–счетчика показана на рисунке.

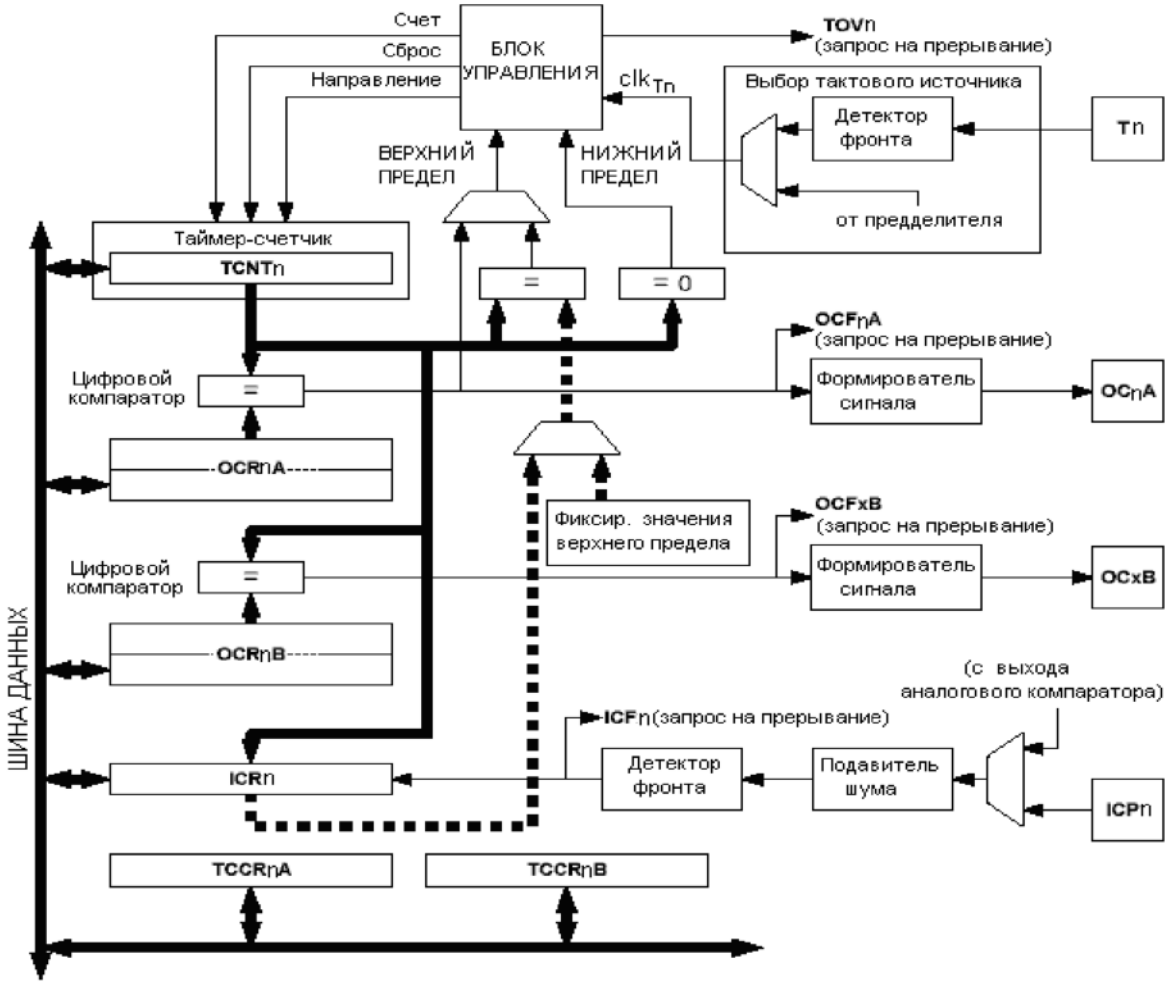


Таблица 9. Регистр A управления таймером–счетчиком 1 – TCCR1A

Bit	7	6	5	4	3	2	1	0
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
Чт./зап.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.
Исх. зн.	0	0	0	0	0	0	0	0

Биты 7:6 – COM1A1:0 Режим формирования выходного сигнала канала A
Биты 5:4 – COM1B1:0 Режим формирования выходного сигнала канала B

Таблица 10. Управление режимами сигналов OCnA/OCnB

COMnA1/	COMnA0/ COMnB0	Описание
0	0	Нормальная работа порта, сигналы OCnA/OCnB
0	1	Переключение (инвертирование) OCnA/OCnB
1	0	Сброс OCnA/OCnB при совпадении (установка
1	1	Установка OCnA/OCnB при совпадении

Биты 3:2 - FOC1A:FOC1B: Режим формирования силы выходного сигнала.
Разряд 1:0 - WGMn1:0: Режим работы таймера-счетчика

Таблица 11. Регистр B управления таймером-счетчиком 1 - TCCR1B

Разряд	7	6	5	4	3	2	1	0
	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

Чт./зап.	Чт./Зп.	Чт./Зп.	Чт.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.
Исх. зн.	0	0	0	0	0	0	0	0

Разряд 7 - ICNC1: Подавитель шума на входе захвата (задержка сигнала с входа захвата на 4 такта)

Разряд 6 - ICES 1: Выбор детектируемого фронта на входе захвата

Разряд 5 - Зарезервированный бит

Разряд 4:3 - WGM1 3:2: Режим работы таймера-счетчика

Разряд 2:0 - CS12:0: Выбор тактового источника

Данные три бита, представленные в таблице 21, позволяют выбрать тактовый источник для таймера-счетчика.

Таблица 12. Описание бит выбора тактового источника

S12	S11	S10	Описание
			Нет синхронизации. Таймер-счетчик остановлен.
			clkI/O/1 (без предделения)
			clkI/O /8 (с предделением)
			clkI/O/64 (с предделением)
			clkI/O/256 (с предделением)
			clkI/O/1024 (с предделением)
			Внешний тактовый источник с выв. T1. Синхронизация по падающему
			Внешний тактовый источник с выв. T1. Синхронизация по нарастающему

Если для тактирования таймера выбран внешний вывод T1, то данная функция за ним сохраняется, даже при его настройке на вывод. Данная функция позволяет программно управлять счетом.

В таблице 13 представлены разряды таймер-счетчиков 1 - TCNT1H и TCNT1L.

Таблица 13. Таймер-счетчик 1 - TCNT1H и TCNT1L

Разряд	7	6	5	4	3	2	1	0	
	TCNT1[15:8]								TCNT1H
	TCNT1	[7:0]							TCNT1L
Чт./зап.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	
Исх. зн.	0	0	0	0	0	0	0	0	

Две ячейки в области ввода-вывода (TCNT1H и TCNT1L, вместе TCNT1) дают полный доступ, как на чтение, так и на запись к 16-разрядному счетчику. В целях гарантирования одновременности чтения и записи старшего и младшего байтов этих регистров, доступ организован с использованием 8-разрядного временного регистра старшего байта (TEMP). Временный регистр является общим для всех 16-разрядных регистров таймера.

Изменение содержимого счетчика TCNT1 во время его работы (счета) связано с риском возникновения совпадения между TCNT1 и одним из регистров OCR1x. Запись в регистр TCNT1 блокирует отработку совпадения, которое возникнет на следующем такте, для всех блоков сравнения.

В таблицах 14 и 15 представлены разряды регистров сравнения 1A - OCR1AH, OCR1AL и 1B - OCR1BH и OCR1BL

Таблица 14. Регистр сравнения 1 A - OCR1AH и OCR1AL

Разряд	7	6	5	4	3	2	1	0	
	OCR1A [15:8]								OCR1AH
	OCR1A [7:0]								OCR1AL
Чт./зап.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	
Исх. зн.	0	0	0	0	0	0	0	0	

Таблица 15. Регистр сравнения 1B - OCR1BH и OCR1BL

Разряд	7	6	5	4	3	2	1	0	
	OCR1B [15:8]								OCR1BH
	OCR1B [7:0]								OCR1BL
Чт./зап.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	
Исх. зн.	0	0	0	0	0	0	0	0	

В регистрах сравнения хранится 16-разр. значение, которое непрерывно сравнивается со значением счетчика (TCNTn). Возникающее совпадение может использоваться для генерации прерывания по результату сравнения и генерации прямоугольных импульсов на выводе OCnx.

Регистры сравнения являются 16-разрядными, поэтому, одновременность записи младшего и старшего байтов достигнута за счет использования 8- разр. временного регистра старшего байта (TEMP). Временный регистр является общим для всех 16-разрядных регистров таймера.

Регистры захвата, представленные в таблице 16, обновляются содержимым соответствующего счетчика (TCNTn) при каждом определении условия захвата на входе ICPn (или альтернативно на выходе аналогового компаратора для таймера-счетчика 1).

Таблица 16. Регистр захвата 1 - ICR1H и ICR1L

Разряд	7	6	5	4	3	2	1	0	
	ICR1H [15:8]								ICR1H
	ICR1L [7:0]								ICR1L
Чт./зап.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	Зп./Чт.	
Исх. зн.	0	0	0	0	0	0	0	0	

Регистры захвата альтернативно могут использоваться для задания верхнего предела счета.

Регистры захвата также являются 16-разрядными, поэтому, одновременность записи младшего и старшего байтов достигнута за счет использования 8-разр. временного регистра старшего байта (TEMP). Временный регистр является общим для всех 16-разрядных регистров таймера.

В таблице 17 представлены разряды регистра маски прерываний таймера- счетчика - TIMSK

Таблица 17. Регистр маски прерываний таймера-счетчика - TIMSK

Разряд	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Чт//зап.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	
Исх. зн.	0	0	0	0	0	0	0	0	

Прим.: Данный регистр биты управления прерываниями для нескольких таймер-счетчиков, но в данном разделе детализированы только биты таймера 1. Описание остальных бит необходимо искать при описании соответствующих таймеров.

Разряд 5 - TICIE1: Разрешение прерывания по захвату состояния таймера-счетчика 1

Если в данный бит записана лог. 1, а также установлен флаг I в регистре статуса (активно общее разрешение прерываний), то разрешается прерывание по захвату состояния таймера-счетчика 1. Если устанавливается флаг в регистре TIFR, программа переходит на соответствующий вектор прерывания .

Разряд 4 - OCIE1A: Разрешение прерывания по результату сравнения канала А таймера-счетчика 1

Если в данный бит записана лог. 1 и установлен флаг I в регистре статуса, то разрешается работа прерывания по результату сравнения канала А. Если устанавливается флаг OCF1A в регистре TIFR, то программа переходит на соответствующий вектор прерываний.

Разряд 3 - OCIE1B: Разрешение прерывания по результату сравнения канала В таймера-счетчика 1

Действие аналогично предыдущему, но в отношении канала сравнения В.

Разряд 2 - TOIE1: Разрешение прерывания при переполнении таймера- счетчика 1

Если в данный бит записана лог. 1 и установлен флаг I в регистре статуса, то разрешается прерывание по переполнению таймера-счетчика 1. После этого, установка флага TOV1 в регистре TIFR приведет к переходу на соответствующий вектор прерывания.

В таблице 18 представлен регистр флагов прерываний таймеров- счетчиков TIFR.

Таблица 18. Регистр флагов прерываний таймеров-счетчиков - TIFR

Разряд	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOF0	TIFR
Чт./зап.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	Чт./Зп.	
Исх. зн.	0	0	0	0	0	0	0	0	

Прим.: Биты данного регистра относятся к нескольким таймерам, но в данном параграфе рассматриваются биты только одного таймера. Описание остальных бит необходимо смотреть в соответствующих разделах.

Разряд 5 - ICF1: Флаг захвата состояния таймера-счетчика 1

Флаг устанавливается, если на входе ICP1 определяется условие захвата. Если регистр захвата ICR1 выбран с помощью бит WGMn3:0 в качестве источника верхнего предела счета, флаг ICF1 устанавливается по достижении верхнего предела счета.

ICF1 автоматически сбрасывается при переходе на вектор прерывания по захвату состояния таймера-счетчика. Альтернативно флаг ICF1 можно сбрасывать путем записи в него лог. 1.

Разряд 4 - OCF1A: Флаг результата сравнения канала А таймера-счетчика 1.

Данный флаг устанавливается следующим тактом после совпадения значения TCNT1 с регистром А порога сравнения (OCR1A).

Обратите внимание, что строб принудительной установки результата сравнения (FOC1A) не устанавливает флаг OCF1A. Флаг OCF1A автоматически сбрасывается при переходе на соответствующий вектор прерывания.

Альтернативно, флаг OCF1A сбрасывается путем записи в него лог. 1.

Разряд 3 - OCF1B: Флаг результата сравнения канала В таймера-счетчика 1.

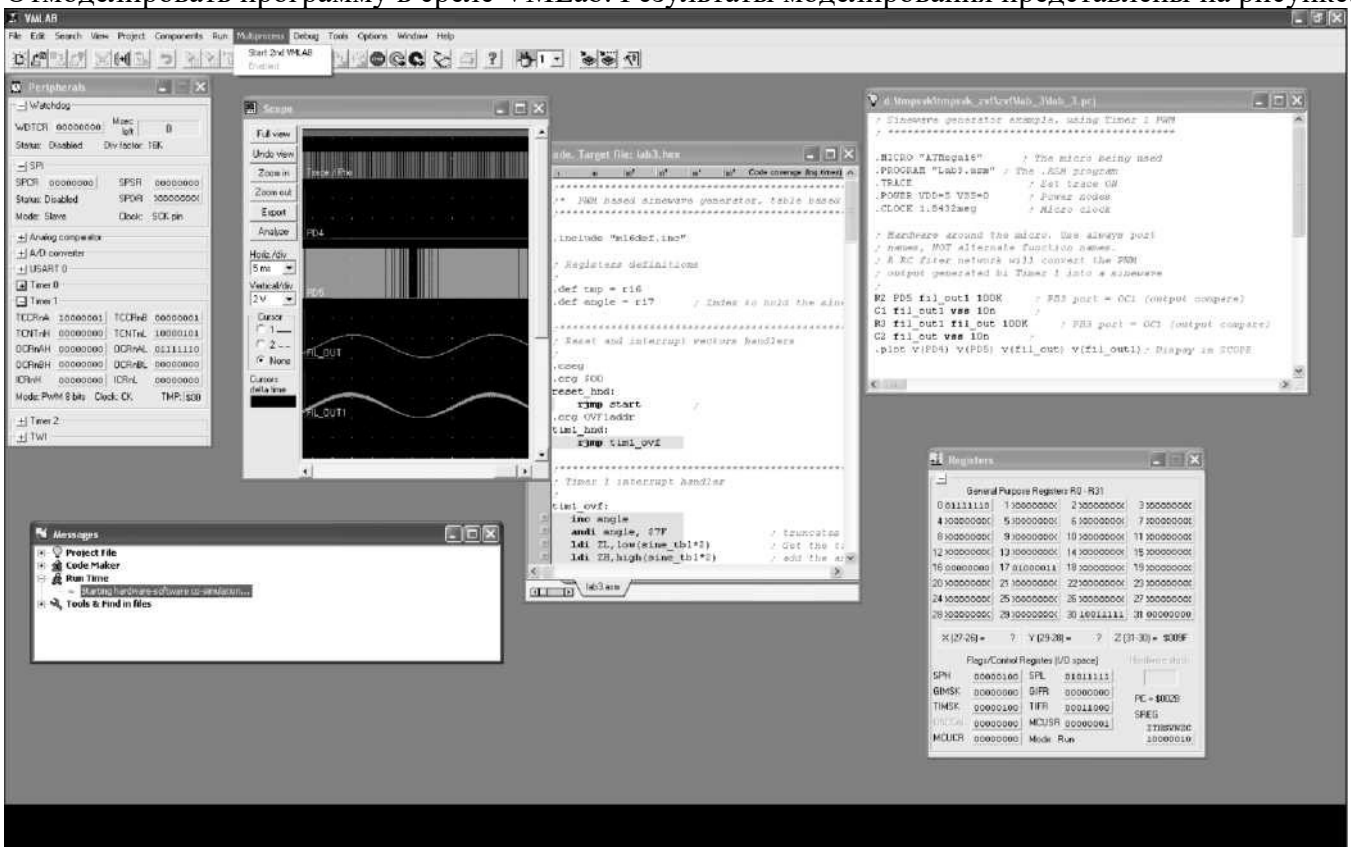
Данный флаг действует аналогично предыдущему, но в отношении канала сравнения B.

Разряд 2 - TOV1: Флаг переполнения таймера-счетчика 1

Установка данного флага зависит от значений бит WGMn3:0. В нормальном режиме и режиме CTC флаг TOV1 устанавливается при переполнении таймера-счетчика. См. табл. 61 для изучения поведения флага TOV1 при задании других значений WGMn3:0. Флаг TOV1 автоматически сбрасывается при переходе на вектор прерывания по переполнению таймера-счетчика 1. Альтернативно флаг TOV1 сбрасывается путем записи в него лог. 1.

Задание к практической работе

Изучить файл проекта Lab_3.prj и файл программы Lab3.asm на языке Ассемблер. Отмоделировать программу в среле VMLab. Результаты моделирования представлены на рисунке.



Файл Lab_3.prj

; Sinewave generator example, using Timer 1 PWM .MICRO "ATmega16" ; The micro being used .PROGRAM "Lab3.asm" ; The .ASM program .TRACE ; Set trace ON

.POWER VDD=5 VSS=0 ; Power nodes

.CLOCK 1.8432meg ; Micro clock

R2 PD5 fil_out1 100K ; PB3 port = OC1 (output compare)

C1 fil_out1 vss 10n ;

R3 fil_out1 fil_out 100K ; PB3 port = OC1 (output compare)

C2 fil_out vss 10n ;

.plot v(PD4) v(PD5) v(fil_out) v(fil_out1); Dispay in SCOPE

Файл Lab3.asm

;

;* PWM based sinewave generator, table based

include "m16def.inc"

Registers definitions def tmp = r16

def angle = r17 ; Index to hold the sine phase angle (0 to 127)

Reset and interrupt vectors handlers

cseg org $00 reset_hnd: rjmp start ;

```
.org OVF1addr tim1_hnd:
rjmp tim1_ovf
;
; Timer 1 interrupt handler
;
tim1_ovf: inc angle
andi angle, $7F ; truncates to 7 bits (0 - 127)
ldi ZL,low(sine_tbl*2) ; Get the table address and ldi ZH,high(sine_tbl*2) ; add the angle phase. Result
add ZL,angle ; will be in R0 after calling 'lpm'
clr tmp; lpm uses 16 bits index (Z)
adc ZH, tmp ; Z index is the pointer to the
lpm ; sine_tbl value for the angle phase clr tmp
out OCR1AH, tmp ; Reload Timer 1 compare value.
out OCR1AL,R0 ; Necessary to reload both registers
reti
;
; Reset handler. Initalizes port and Timer 1, and stay in a endless loop
;
start:
sbi DDRD, PD4 ; Set pin PD5 as output (is OC1 pin) sbi DDRD, PD5 ldi tmp, high(RAMEND) ;
инициализация памяти стека out SPH, tmp ldi tmp, low(RAMEND)
out SPL, tmp ; завершение инициализации памяти стека ldi tmp,(1<<TOIE1)
out TIMSK,tmp ; Enable Timer1_ovf interrupt
ldi tmp,(1<<PWM10)+(1<<COM1A1) ; Set Timer 1 in PWM mode out TCCR1A,tmp ; 8 bit PWM
not reverse (Fck/510)
ldi tmp,(1<<CS10)
out TCCR1B,tmp ; prescaler = 1
clr angle ; Start with phase = 0
sei ; Enable interrupts
main:
nop
nop
rjmp main ; Infinite loop: Timer1 interrupt will handle all
************************** SINE TABLE
; Samples table : one period sampled on 128 samples and ; quantized on 7 bit
;
sine_tbl:
.db 0,0 .db 1,1 .db 2,3 .db 4,6
db
db 10,12
db 14,16
db 18,21
db 23,25
db 28,31
db 33,36
db 39,42
db 45,48
db 51,54
db 57,60
db 64,67
db 70,73
```

```
db 76,79
db 82,85
db 88,91
db 94,96
db 99,102
db 104,106
db 109,111
db 113,115
db 117,118
db 120,121
db 123,124
db 125,126
db 126,127
db 127,127
db 127,127
db 127,127
db 126,126
db 125,124
db 123,121
db 120,118
db 117,115
db 113,111
db 109,106
db 104,102
db 99,96
db 94,91
db 88,85
db 82,79
db 76,73
db 70,67
db 64,60
db 57,54
db 51,48
db 45,42
db 39,36
db 33,31
db 28,25
db 23,21
db 18,16
db 14,12
db 10,9
db 7,6
db 4,3
db 2,1
db 1,0
```

```
db 0,0
db 0,0
```

Порядок выполнения работы

Нужно выполнить все необходимые операции с проектом Lab.prj и программой Lab.asm в среде VMLab.

Содержание отчета

Отчет по проделанной работе должен содержать:
структурную схему алгоритма программы на практическую работу;
схему электрическую принципиальную разработанного устройства;
текст программы на языке AVR Ассемблере (распечатка файла *asm);
распечатка файла проекта (*.prj);
результаты моделирования (окно VMLab с запущенной программой и результатами вычислений).

Контрольные вопросы

1. Регистр состояния микроконтроллера ATmega16.
2. Стек и его инициализация.
3. Память микроконтроллера ATmega16.
4. Процессорное ядро микроконтроллеров AVR.
5. Непосредственная адресация данных в микроконтроллерах AVR.
6. Прямая адресация данных в микроконтроллерах AVR.
7. Косвенная адресация данных в микроконтроллерах AVR.
8. Относительная адресация данных в микроконтроллерах AVR.

ПРАКТИЧЕСКАЯ РАБОТА №20. ПРОГРАММИРОВАНИЕ АНАЛОГОВОГО КОМПАРАТОРА МИКРОКОНТРОЛЛЕРОВ

Цель работы: Изучение и освоение методики программирования аналогового компаратора микроконтроллеров.

Краткие теоретические сведения

Аналоговый компаратор микроконтроллера ATmega16 имеет два входа — AIN0 и AIN1. В состав аналогового компаратора кроме базового компаратора входит регистр управления-состояния ACSR (№ $08) и элементы, управляющие работой схемы. Результатом работы компаратора является запрос прерывания ANA COMP, который формируется, когда разность значений напряжения на входах компаратора меняет знак.

Схема управления СУ при определенном изменении сигнала ACO устанавливает в единичное состояние разряд ACI регистра ACSR и при единичном состоянии разряда ACIE регистра ACSR в блок прерываний поступает запрос прерывания ANA COMP. Разряд ACI сбрасывается в нулевое состояние аппаратно при переходе к выполнению прерывающей программы или программно путем записи единицы в разряд ACI.

Выбор вида изменения сигнала ACO на входе схемы управления СУ, при котором формируется запрос прерывания, определяется комбинацией состояний разрядов ACISO и ACIS1 регистра ACSR в соответствии с табл. 1.

Таблица 1. Выбор вида сигнала компаратора

ACIS1	ACIS0	Изменение сигнала ACO
0	0	любое
0	1	—
1	0	1^0
1	1	0^1

В микроконтроллере ATmega16 сигнал ACO с выхода базового компаратора при единичном состоянии разряда ACIC принимается в таймер-счетчик в качестве сигнала, управляющего захватом.

При установке в единичное состояние разряда ACD регистра ACSR отключается питание базового компаратора и уменьшается ток потребления микроконтроллера.

В микроконтроллере ATmega16 имеется возможность подключать к входу «+» базового компаратора вместо входа AIN0 выход внутреннего источника эталонного напряжения VR (1,22 ± 0,05 В). Подключение источника VR выполняется при единичном состоянии разряда AINBG регистра ACSR, кроме того, имеется возможность подключать к входу «-» базового компаратора входы аналого-цифрового преобразователя ADC0...ADC7. Подключение выполняется при нулевом состоянии разряда ADEN регистра ADCSR (№ $06) и единичном состоянии разряда ACME регистра SFIOR (№ $30). Выбор подключаемого входа определяется комбинацией состояний разрядов MUX2, MUX1 и MUX0 регистра ADMUX (№ $07).

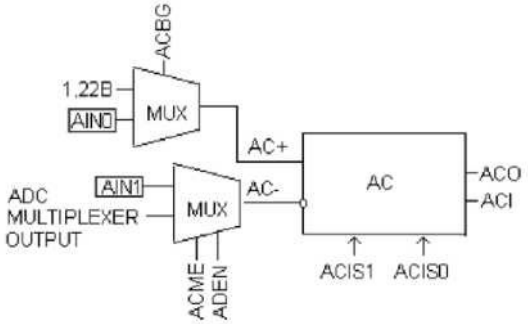
В табл. 2 указаны выводы микроконтроллера, используемые в качестве входов AIN0 и AIN1, у микроконтроллеров разных типов.

Таблица 2. Входы AIN0 и AIN1

Вход	t11112	t15 t28	1200 2313	4433	8515 8535	m163	m103*
AIN0	PB0	PB0	PB0	PD6	PB2	PB2	PE2
AIN1	PB1	PB1	PB1	PD7	PB3	PB3	PE3

* — AC+, AC-

Схема компаратора микроконтроллера *ATmega16* приведена на следующем рисунке:



В работе компаратора используются регистры:
Регистр управления аналоговым компаратором ACSR
Регистр специальных функций ввода/вывода
Регистр состояния аналого-цифрового преобразователя
Регистр мультиплексора аналого-цифрового преобразователя
Регистр состояния микроконтроллера SREG.

Биты	7	6	5	4	3	2	1	0
ACSR $08 ($28)	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
SFIOR $30 ($50)					ACME			
ADCSR $06 ($26)	ADEN							
ADMUX $07 ($27)						MUX2	MUX1	MUX0
SREG $3F ($5F)	I							

Регистр ACSR предназначен для управления аналоговым компаратором.

Бит 7 - ACD - отключает аналоговый компаратор.

Бит 6 - ACBG - выбор эталона аналогового компаратора.

Бит 5 - ACO - выход аналогового компаратора.

Бит 4 - ACI - флаг прерывания по аналоговому компаратору.

Бит 3 - ACIE - разрешение прерывания по аналоговому компаратору.

Бит 2 - ACIC - разрешение входа захвата аналогового компаратора.

Биты 1,0 - ACIS1, ACIS0 - выбор режима прерывания по аналоговому компаратору. Варианты установок показаны в таблице 3.

Таблица 3. Установки битов ACIS1/ACIS0

A	A	Режим прерывания
0	0	Прерывание по переключению выхода компаратора
1	0	Зарезервировано
1	0	Прерывание по падающему фронту на выходе компаратора
1	1	Прерывание по нарастающему фронту на выходе компаратора

При изменении состояния битов ACIS1/ACIS0 прерывание по аналоговому компаратору должно быть запрещено очисткой бита разрешения прерывания в регистре ACSR. В противном случае, при изменении состояния битов может произойти прерывание.

На отрицательный вход аналогового компаратора можно скоммутировать любой из входов порта PORTA (PA7 .. PA0). Для выбора входа используется мультиплексор аналогоцифрового преобразователя.

Бит ACME в регистре специальных функций ввода вывода SFIOR предназначен для подключения мультиплексора к аналоговому компаратору.

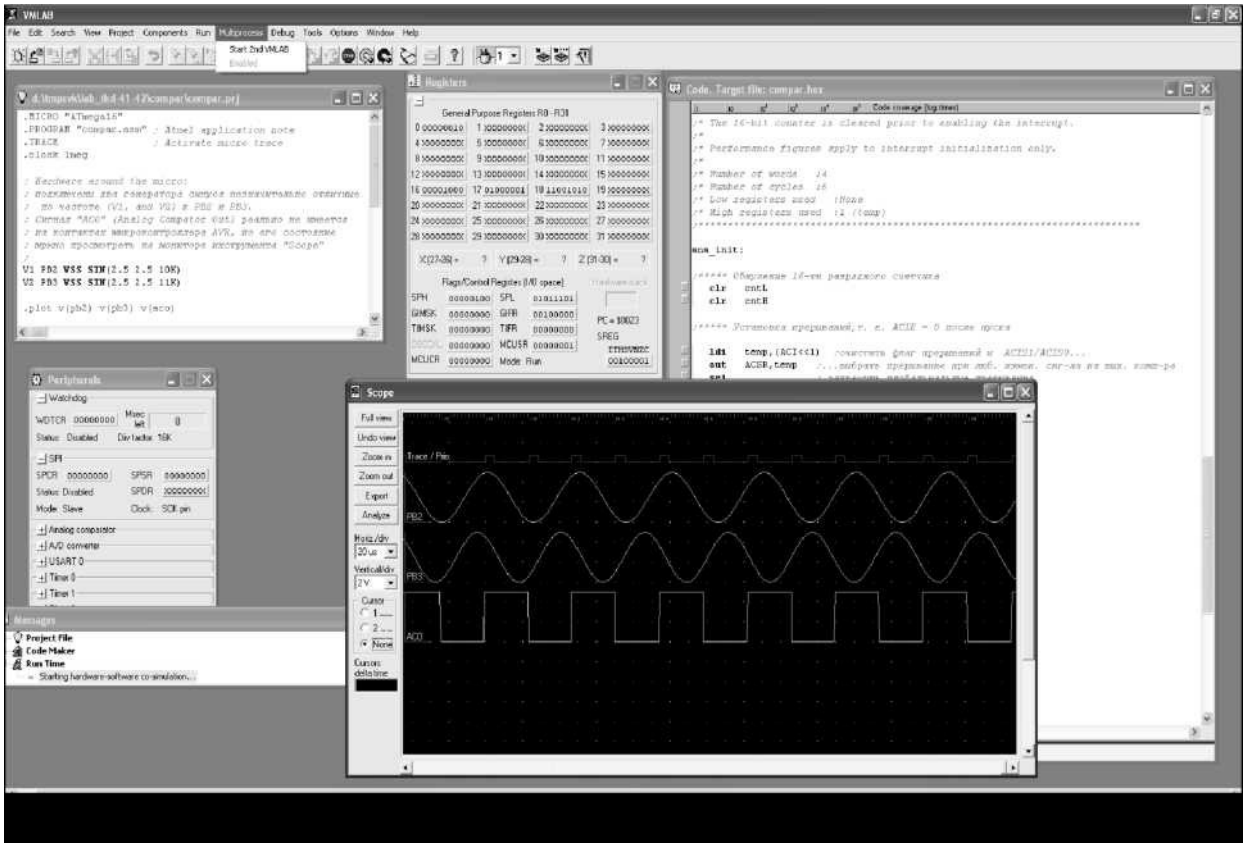
Битами MUX2..0 в регистре мультиплексора ADMUX выбирается контакт на входе (табл. 4).

Таблица 4. Логика подключения отрицательного входа компаратора.

ACME	ADEN	MUX2..0	Отрицательный вход аналогового компаратора
0	X	XXX	AIN1
1 1		XXX	AIN1
1	0	000	PAO
1	0	001	PA1
1	0	010	PA2
1	0	011	PA3
1	0	100	PA4
1	0	101	PA5
1	0	110 PA6	
1	0	111 PA7	

Задание к практической работе

Изучить программу Lb.asm и файл проекта Lb.prj. Отладить программу в среде VMLab, подключив необходимую периферию к микроконтроллеру AVR - источники синусоидального напряжения. Программа сравнивает аналоговые напряжения на контактах PB2 и PB3, на которые подаются сигналы от генераторов синусоидальных сигналов с разной частотой. Визуально просматривается сигнал с выхода аналогового компаратора. На рисунке представлены результаты моделирования программы управления аналоговым преобразователем по прерываниям.



.MICRO "ATmega16"
.PROGRAM "Lab7.asm" ; Atmel application note .TRACE ; Activate micro trace
.clock 1meg
; подключены два генератора синуса незначительно отличные ; по частоте (V1, and V2) к PB2 и PB3.
; Сигнал «AC0» (Analog Compator Out) реально не имеется
; на контактах микроконтроллера AVR, но его состояние ; можно просмотреть на мониторе инструмента «Scope» V1 PB2 VSS SIN(2.5 2.5 10K)
V2 PB3 VSS SIN(2.5 2.5 11K)
.plot v(pb2) v(pb3) v(aco)
Программа использует аналоговый компаратор микроконтроллера
- ожидание по положительному перепаду выхода компаратора
- ожидание по положительному перепаду флага прерывния
- Enable interrupt on comparator output toggle. The interrupt routine
- инкремент 16-ти разрядного счетчика ter counter each time it is executed include "C:\vmlab\include\m16def.inc"
def temp =r16 ;temporary storage register def cntL =r17 ;регистр счета младшего байта def cntH =r18 ;регистр счета старшего байта ;* Инициализация векторов прерываний rjmp RESET ;В начало программы org ACIaddr
rjmp ANA_COMP ;В подпрограмму обработки прерывания от АС ******** Подпрограмма обработки прерывания от АС**********

Эта подпрограмма увеличивает содержимое 16-ти разрядного счетчика
всякий раз когда возникает прерывание по аналоговому компаратору
def ac_tmp =r0 ;временный регистр хранения для SREG ANA_COMP: in ac_tmp,SREG запоминание SREG subi cntL,low(-1)
sbci cntH,high(-1) ;инкремент 16-ти разрядного счетчика cnt(H,L)=cnt(H,L)- (-1)
out SREG,ac_tmp ;восстановление SREG reti ; возврат из прерываний
********** Основная программа ********** RESET:
****** Инициализация стека ldi temp,low(RAMEND) out SPL,temp ldi temp,high(RAMEND) out SPH,temp ********** «ожидание edge1»
;* Эта часть осуществляет ожидание (задержку) пока выход компаратора ;* (ACO-bit в ACSR) не станет равным 1.
;* extremely short pulses can be missed, since the program runs three clock
cycles between each time the comparator is checked. Another disadvantage
is that the program has to wait for the output to be come negative first,
in case the output is positive when polling starts.
Number of words :4
Number of cycles :4 per loop. Response time: 3 - 5 clock cycles
Low registers used :None
High registers used :None wait_edge1:
sbic ACSR,ACO ;if output is high rjmp wait_edge1 ; wait we1_1:
sbis ACSR,ACO ;if output is low rjmp we1_1 ; wait
********** «ожидание edge2» **********
This piece of code waits until the output of the comparator (the ACO-bit
in ACSR) goes high. This is a more secure solution, since the interrupt
flag is polled. This allows the user to insert code within the wait loop
because hardware «remembers» pulses of shorter duration than the polling
interval. Another positive feature is that there is no need to wait for
a preceeding negative edge.
Number of words :5
Number of cycles :Inital setup :2
Flag clearing: 1
Loop :4
Response time:3 - 5
Low registers used :None
High registers used :None wait_edge2:
.***** Инициализация режима прерывания, при пуске (reset) ACIE = 0 sbi ACSR,ACIS0
sbi ACSR,ACIS1 .установка режима прерывания с 0 в 1 .***** Ожидание sbi ACSR,ACI .запись «1» в ACI we2_1:
sbis ACSR,ACI ;if ACI is low rjmp we2_1
********** «ana init» **********
This code segment enables Analog Comparator Interrupt on output toggle.
The program then enters an infinite loop.
The 16-bit counter is cleared prior to enabling the interrupt.
Performance figures apply to interrupt initialization only.
Number of words :4
Number of cycles :5
Low registers used :None
;* High registers used : 1 (temp)
;**********
;
*
ana_init:
.***** Обнуление 16-ти разрядного счетчика clr cntL clr cntH

.***** установка прерываний, т. к. ACIE = 0 после пуска ldi temp,(ACI<<1) .очистить флаг прерываний и ACIS1/ACIS0... out ACSR,temp ....выбрать прерывание при люб. измен. сиг-ла на вых. комп-ра

sei . разрешить глобальные прерывания

sbi ACSR,ACIE .разрешение прерывания от AC forever:rjmp forever

Порядок выполнения лабораторной работы

Нужно выполнить все необходимые операции с проектом Lab_7.prj и программой Lab7.asm в среде VMLab.

Содержание отчета

Отчет по проделанной работе должен содержать:

структурную схему алгоритма программы на практическую работу.

схему электрическую принципиальную разработанного устройства.

текст программы на языке ассемблера (распечатка файла *asm);

распечатка файла проекта (*.prj);

результаты моделирования (окно VMLab с запущенной программой и результатами вычислений).

Контрольные вопросы

1. Входные сигналы аналогового компаратора микроконтроллера ATmega16.
2. Управление компаратором микроконтроллера ATmega16.
3. Задание режима прерывания микроконтроллера ATmega16.
4. Источники входных сигналов аналогового компаратора в микроконтроллере ATmega16.
5. Разрешение входа захвата таймером AC в микроконтроллере ATmega16.
6. Бит разрешения прерывания в AC микроконтроллера AVR ATmega16.
7. Флаг прерывания в AC микроконтроллера ATmega16.
8. Бит выхода аналогового компаратора микроконтроллера ATmega16.
9. Бит выбора эталона аналогового компаратора микроконтроллера ATmega16.
10. Бит отключения аналогового компаратора микроконтроллера ATmega16

ПРАКТИЧЕСКАЯ РАБОТА №21. СИСТЕМЫ УПРАВЛЕНИЯ И КОНТРОЛЯ НА ОДНОКРИСТАЛЬНЫХ МИКРОКОНТРОЛЛЕРАХ ФИРМЫ MICROCHIP.

Цель работы: ознакомиться с системами управления и контроля на однокристальных микроконтроллерах фирмы *Microchip*.

Теоретические сведения

Разработанная фирмой *Microchip* 16-битная платформа реализована в двух семействах 16-битных микроконтроллеров и в двух семействах цифровых сигнальных контроллеров. Цифровые сигнальные процессоры (ЦСП) - в английском написании *DSP (Digital Signal Processor)* - известны с 1982 года, когда компания *Texas Instruments* выпустила на рынок свой *TMS32010.* Изначально ЦСП предназначались для военных применений, таких как радиолокация, шифрование и др. Среди решаемых ими задач были реализация быстрого преобразования Фурье (БПФ), фильтрация, координатные преобразования и т. п. Все эти функции требовали предельного быстродействия, поскольку должны были выполняться в реальном масштабе времени. В связи с этим отличительной чертой архитектуры ЦСП являются наличие независимых умножителя и АЛУ, встроенной памяти для команд и отдельно для данных, а также нескольких шин, позволяющих одновременно производить выборку команды, выборку данных и запись результата. С учетом того, что большинство операций выполняются в ЦСП за один машинный такт, эти устройства позволяют достигать производительности, на порядок превышающей производительность универсальных микроконтроллеров (МК).

Главным достоинством ЦСП является универсальность (в смысле возможности решения широкого круга задач одним МК) и относительная простота программирования (вскоре вслед за возникновением первых кристаллов появился не только ассемблер, но и языки высокого уровня - С и Ада). Благодаря стремительному развитию электронной технологии стало возможным не только реализовать на одном кристалле быстрый вычислитель, но и существенно расширить периферию, дополнить систему команд типичными микроконтроллерными инструкциями, реализовать эффективную систему прерываний. Сегодня некоторые ЦСП производительностью 100 *MIPS* (миллионов инструкций в секунду) стоят всего порядка 5 долларов. Таким образом, доведенная до совершенства возможность эффективного решения наукоемких задач и реально возникшая потребность рынка в таких задачах удачно разрешились появлением цифровых сигнальных процессоров для микроконтроллерных приложений.

Большинство производителей современных 8-разрядных МК используют *гарвардскую архитектуру,* основной особенностью которой является использование раздельных адресных пространств для хранения команд и данных. Однако гарвардская архитектура является недостаточно гибкой для реализации некоторых программных процедур.

Семейства 16-битных микроконтроллеров и цифровых сигнальных контроллеров объединяет ряд общих характеристик:

совместимость по назначению выводов различных 16-битных устройств;

возможность использования для всех устройств одних и тех же инструментальных средств разработки программного обеспечения;

аппаратно-программная совместимость всех одноименных периферийных модулей микроконтроллеров;

общая базовая система команд процессора, используемая во всех семействах.

Выбор той или иной модели микроконтроллера или сигнального контроллера зависит от требований к разрабатываемому приложению. Для большинства недорогих устройств средней производительности подходят микроконтроллеры *PIC24F,* максимальная производительность которых составляет 16 *MIPS*. Для устройств, требующих высокой производительности, можно использовать микроконтроллеры *PIC24H* с максимальным быстродействием 40 *MIPS.* Микроконтроллеры семейств *PIC24F* и *PIC24H* работают с одним и тем же набором инструкций процессора, включают одни и те же периферийные модули, имеют одну и ту же цоколевку, и для работы с ними используются одни и те же инструментальные средства для разработки программного обеспечения.

Если требуются дополнительные возможности по обработке сигналов, то вместо микроконтроллеров семейств *PIC24F/H* можно применить цифровые сигнальные контроллеры семейства *dsPIC30F,* которые могут помимо всего прочего работать при напряжении питания 5 В, или высокопроизводительные (40 *MIPS)* контроллеры *dsPIC33F*, которые имеют большой объем памяти и используют низковольтное (3.3 В) питание. В качестве инструментального средства разработки программного обеспечения 16-битных микроконтроллеров и цифровых сигнальных контроллеров используется свободно распространяемая интегрированная среда разработки (ИСР) *MPLAB IDE* фирмы *Microchip,* которая позволяет разрабатывать и отлаживать 8-, 16- и 32-битные приложения. Среда *MPLAB IDE* позволяет выполнять тестирование и отладку программ с использованием мощного программного симулятора *MPLAB SIM.* Кроме того, для разработки программного обеспечения для 16-битных систем в среде *MPLAB IDE* можно использовать следующие инструментальные средства:

ассемблер *ASM30* — полнофункциональный макроассемблер, в котором можно создавать пользовательские макросы и использовать условное ассемблирование. Многочисленные директивы языка делают макроассемблер очень мощным средством разработки программ;

компилятор программ, написанных на языке Си, который называется *MPLAB C* для *PIC24.* Этот компилятор используется для компиляции и оптимизации программ, написанных для 16битных микроконтроллеров *PIC24F/H* и цифровых сигнальных контроллеров *dsPIC30/33.* Он совместим со стандартом *ANSI C* и включает полную библиотеку стандартных функций *ANSI C*, в числе которых функции манипулирования строками, функции работы с динамической памятью, функции преобразования даты/времени и математические функции. В компиляторе *MPLAB C* для *PIC24* имеется мощный оптимизатор, позволяющий почти в 1,5 раза уменьшить размер программного кода по сравнению с компиляторами других фирм-производителей;

визуальный генератор кода инициализации *MPLAB VDI,* позволяющий значительно упростить процесс создания инициализационного кода программы. С помощью *VDI* можно в графическом виде сконфигурировать устройство и по завершении вставить сгенерированный программный код инициализации в программу на языке Си или ассемблере;

библиотеку периферийных модулей, включающую более чем 270 функций для работы с различными периферийными модулями;

библиотеку математических функций, совместимую со стандартом *IEEE-754,* которая включает ряд функций для выполнения операций над обычными вещественными числами и вещественными числами с двойной точностью.

Функции этой библиотеки могут использоваться как в программах на языке Си, так и на ассемблере.

Кроме инструментальных средств разработки и отладки программного обеспечения фирмы-производителя на рынке присутствуют и программные средства, выпускаемые многими известными фирмами *(Hi-Tech, CCS* и т.д.). Из аппаратных средств разработки наиболее известна и популярна отладочная плата *«Explorer 16 Development Board»* фирмы *Microchip,* хотя другие фирмы также приступили к выпуску отладочных плат на базе 16битных микроконтроллеров.

Архитектура dsPIC/PIC24

Устройства семейства *dsPIC/PIC24* имеют высокопроизводительный модуль центрального процессорного устройства с 16-битовой (данные) гарвардской двухшинной архитектурой с раздельной памятью программ и данных для конвейерной обработки.

.

Таблица 1

Сводные характеристики микроконтроллеров Microchip с 16-битной архитектурой

Семейство	DSP ядро	Макс. Производительность, MIPS	Прямой доступ к памяти	Макс. объем Flash, кбайт	Макс. объем ОЗУ, кбайт	EEPROM	Макс. кол-во выводов	Диапазон напряжений питания, В	JTAG
PIC24F	-	16	-	128	8	-	100	2,0 - 3,6	+
dsPIC30	+	30	-	144	8	+	80	2,5 - 5,5	-
PIC24H	-	40	+	256	16	-	100	3,0 - 3,3	+
dsPIC33	+	40	+	256	30	-	100	3,0 - 3,3	+

ЦПУ имеет 24-битовое командное слово. Счетчик команд имеет 23 бита и адресуется к области памяти пользовательской программы 4Мх24 бита. ЦПУ имеет шестнадцать 16-битовых рабочих регистров для данных и адресов, шестнадцатый регистр служит как указатель стека для прерываний и вызовов. Также центральный процессор включает в себя 16-битовое арифметико-логическое устройство, блок умножения 17х17 бит, блок деления 32 бита/16 бит, 40-битовый накопитель данных и многорегистровое циклическое сдвиговое устройство.

Особенности архитектуры устройств семейства *dsPIC/PIC24* заключаются в разделении областей памяти и шин для данных и программ (рис. 1).

Эта архитектура также дает возможность прямого доступа к программной памяти из пространства данных во время выполнения кодовой команды. Область памяти адресного пространства программы составляет 4М инструкций. Большинство периферийного оборудования поддерживает прямой доступ к памяти.

Контроллер прерываний dsPIC/PIC24 сводит многочисленные периферийные сигналы запроса на прерывание к единственному сигналу запроса на прерывание к ЦПУ. Он имеет следующие особенности: 7 программируемых уровней приоритетов, таблица векторов прерывания до 118 векторов, уникальный вектор для каждого прерывания, до 67 доступных источников прерывания, до 5

внешних прерываний, альтернативная таблица векторов прерывания для поддержки отладчика.

Все выводы устройства разделены на периферийные и параллельные порты ввода/вывода. Для повышения помехоустойчивости на всех портах ввода имеется триггер Шмидта. Особенности цифрового ввода/вывода устройств *dsPIC/PIC24:* до 85 программируемых цифровых входов/выходов, выходные контакты могут управляться напряжением от 3 В до 3,6 В. Все цифровые входы толерантны к напряжению 5 В (напряжение на выводе цифрового входа может быть от 0,3 В до 5,6 В), нагрузка по току (сток тока) 4 мА на всех контактах вход/выход.

Управление системой имеет следующие особенности: внешний кварцевый резонатор, встроенная система фазовой автоподстройки частоты (ФАПС), таймер запуска осциллятора, сторожевой таймер со своим собственным осциллятором, сброс в исходное положение от многих источников.

Управление режимом электропитания: режим ожидания, "спящий" режим и нерабочий режим с быстрым запуском.

Модуль таймера представляет собой 16-битовый таймер, который может служить как счетчик времени для часов реального времени или как автономный не синхронизированный счетчик интервалов. Таймеры 2/3, 4/5, 6/7 и 8/9 являются 32-битовыми и могут конфигурироваться как четыре независимых таймера с выбираемыми рабочими режимами.

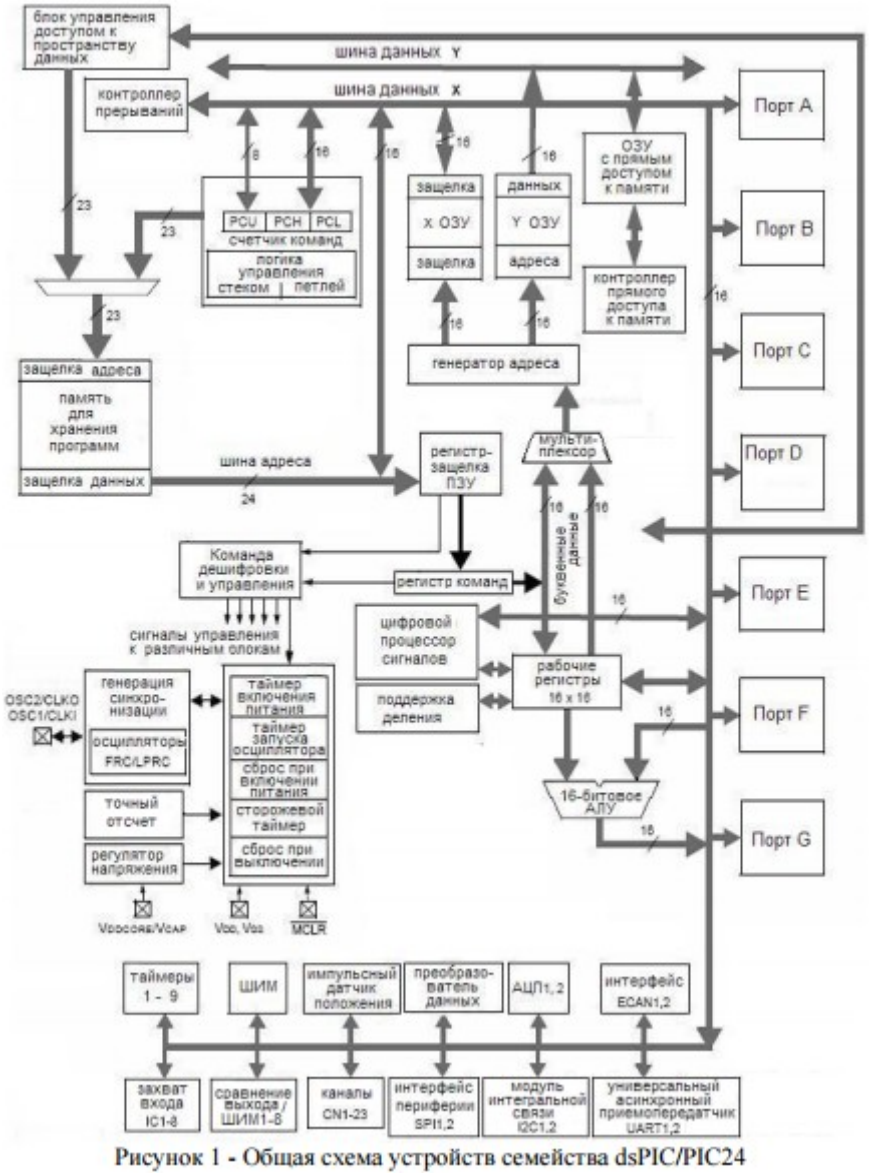


Рисунок 1 - Общая схема устройств семейства dsPIC/PIC24

Организация памяти программ и памяти данных.

В микроконтроллерах семейства *PIC24* реализована гарвардская архитектура, в которой память

программ и память данных разделены, что позволяет осуществлять прямой доступ к памяти программ из памяти данных во время выполнения программного кода. Организация памяти программ для ЦСП семейства *dsPIC* со 128 Кбайт флэш-памяти показана на рисунке 2.

Карта памяти программ всех контроллеров *dsPIC/PIC24* линейная и несегментированная. Все инструкции имеют фиксированную длину 23бита; счетчик инструкций - 23-битный, младший бит всегда равен 0 для обеспечения выравнивания данных при выборке инструкции. Таким образом, эффективное количество адресуемых инструкций равно ~ 4 млн.

Пользовательским программам доступна область памяти в диапазоне адресов 0x000000...0x7FFFFF, за исключением области конфигурации устройства, доступ к которой осуществляется посредством инструкций *tblrd/tblwt*. Память программ организована в виде блоков, адресуемых пословно. Хотя адрес памяти является 24битным, более удобно рассматривать любой адрес в виде младшего и старшего слова, при этом старший байт старшего слова не используется и равен 0. Младшее слово всегда располагается по четному адресу, а старшее - по нечетному. Следует сказать, что адреса памяти программ всегда выравниваются по границе слова и при выполнении программного кода всегда инкрементируются и декрементируются на 2.

Область памяти программ между адресами 0x000000 и 0x000200 зарезервирована для векторов прерываний. По адресам 0x000000 и 0x000002 размещается команда перехода к фактическому началу программы, при этом по первому адресу располагается код операции инструкции *goto,* а по второму — собственно адрес точки входа в программу. Также в памяти программ размещаются две таблицы векторов прерываний. Одна из них располагается в диапазоне адресов *0x000004...0x0000FF,* а вторая (альтернативная) — в диапазоне адресов *0x000100...0x0001FF*.

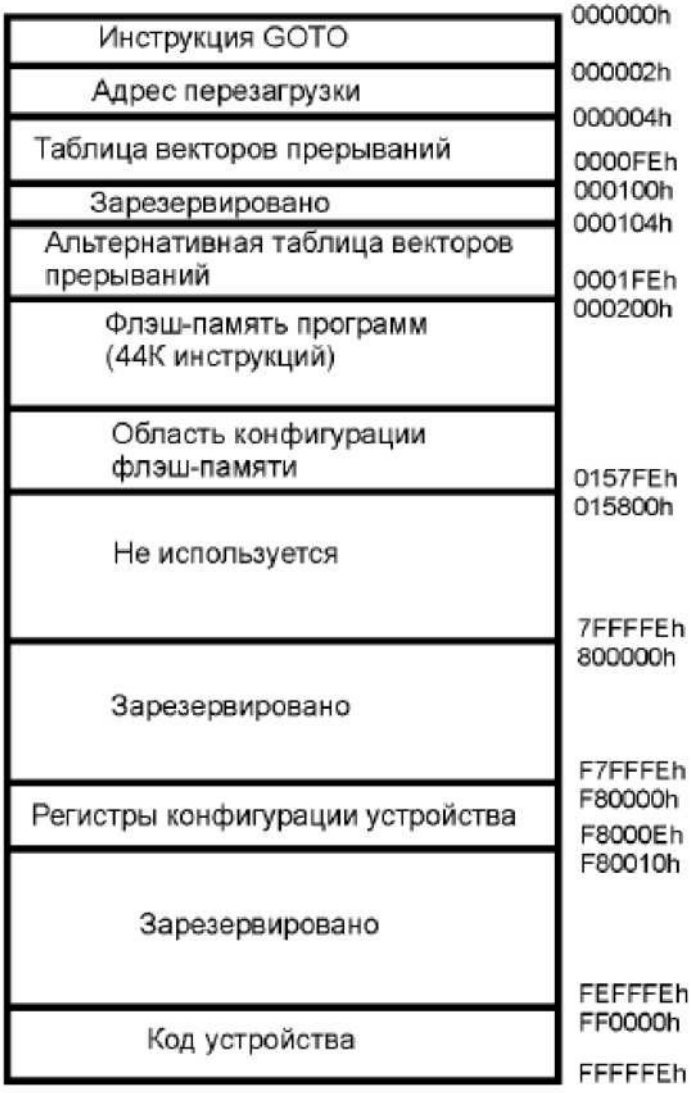


Рисунок 2 - Организация памяти программ ЦСП семейства *dsPIC33*

Физически программная память во всех контроллерах 16-битного семейства реализована в виде перепрограммируемой Flas-памяти. Все контроллеры поддерживают внутрисхемное программирование и программирование в ходе выполнения программы.

В ходе выполнения программы существует три способа доступа к программной памяти:

а) Выборка инструкции в соответствии со значением командного счетчика - собственно само выполнение программы.

б) Использование инструкций табличного чтения-записи, позволяющее получить доступ как к слову (16 бит), так и к байту программной памяти. Табличная запись осуществляется через

буфер, не отображаемый в ОЗУ. Для контроллеров *dsPIC30* минимальный объем записываемых данных равен 12 байтам (4 программных слова, 1/8 сектора Flas^-памяти), для контроллеров *dsPIC33* и *PIC24F/H* - 192 байта (64 программных слова — один сектор Flas-памяти).

в) Чтение программной памяти с помощью механизма *PSV (Program Space Visibility*) - отображения сектора памяти программ в область ОЗУ. Механизм *PSV* позволяет отобразить любую часть программной памяти объемом 32 кбайт в верхнюю, не реализованную физически область ОЗУ.

PSV предоставляет уникальную возможность обращаться к младшим 16 битам программного слова как к динамическим данным - любые инструкции, осуществляющие чтение из ОЗУ, могут использовать программную память в качестве источника. Механизм *PSV* применяется, если алгоритм содержит большие массивы констант: текстовые строки, коэффициенты цифровых фильтров и т. п.

Существенное отличие *dsPIC30* от остальных 16-битных семейств *(dsPIC33,PIC24F/H)*заключается в технологии изготовления интегрированной Flas-памяти. Как следствие - различное количество циклов перепрограммирования. Для *dsPIC30* оно составляет 100 тыс., для *dsPIC33* и *PIC24F/H*— всего 1 тыс. Может показаться, что это серьезный недостаток новых контроллеров, однако, как показывает практика, 1 тыс. циклов перезаписи достаточно для большинства задач, в том числе и для реализации калибруемых устройств.

Различная технология изготовления кристалла определяет и различие в спецификации программирования - микроконтроллеры *dsPIC33* и *PIC24F/H* не требуют подачи высокого напряжения (13 В) на кристалл во время программирования. Кроме того, значительно увеличилась скорость операций с памятью - для контроллеров с объемом *Flash* 64 кбайт полный цикл стирание/запись занимает менее 1 секунды.

Адресное пространство памяти данных для ЦСП семейства *dsPIC* представляет собой непрерывную область 16-битных адресов с линейной адресацией (рисунок 3). Адрес для доступа к данным формируется двумя модулями генерации адресов, один из которых используется в операциях записи, а другой — в операциях чтения данных.

Шина адреса данных микроконтроллеров *dsPIC30/33* и *PIC24F/H* позволяет адресовать до 64 кбайт памяти ОЗУ, которая физически выполнена в качестве статической памяти с возможностью байтового доступа. Почти все инструкции, работающие с ОЗУ, имеют спецификатор, указывающий, будет ли осуществляться доступ к слову (16 бит) или к байту.

В начале ОЗУ находится область регистров специального назначения *(SFR)* объемом 2 кбайт. Внутри семейства все регистры *SFR* расположены статически по одним адресам.

С адреса *0x800* начинается сектор ОЗУ общего назначения, максимальный объем которого составляет 30 кбайт; в разных контроллерах только часть этого сектора может быть реализована физически. Верхнюю половину ОЗУ занимает область, в которую отображается часть программной памяти при использовании механизма *PSV*.

В семействах с .DSP-ядром *(dsPIC30/33)* реализовано два адресных генератора, что позволяет инструкциям DSP-ядра делать две выборки из ОЗУ за один командный такт. Это объясняет разделение области ОЗУ общего назначения на два сектора *X и Y*. Однако такое разделение не является сегментированием или ограничением - для всех инструкций *CPU-ядра* сектор ОЗУ общего назначения линеен.

Первые 8 кбайт ОЗУ (включая область *SFR)* называются пространством ближней памяти *(Near Data Space)*. Прямая адресация возможна только к этому сегменту. Остальная часть ОЗУ может быть адресована косвенно.

16-битная архитектура *Microchip* предусматривает использование программного стека, указателем на который является один из рабочих *(work)* регистров. Стек растет с увеличением

указателя, при этом возможен аппаратный контроль переполнения и опустошения стека. В случае переполнения или опустошения стека, ядром процессора генерируется аппаратное исключение - специальный вид прерывания. Размер стека определяется программно.

При вызове подпрограммы в стек заносится адрес возврата и часть регистра статуса. Указатель на стек может быть изменен программно, что позволяет реализовывать гибкие системы реального времени.

В 16-битном ядре также присутствует *LINK-регистр,* который позволяет выделять в стеке фрейм для локальных переменных функции. Процедуры выделения и сброса фрейма выполняются программно за один командный такт.

Все действующие адреса памяти данных *(Effective Addresses, EA)* имеют размер 16 бит, что позволяет адресовать 64 Кбайт памяти. Нижняя половина адресов памяти данных, для которых старший бит действующего адреса равен 0, используется для адресации данных, а старшая половина адресов, для которых старший бит адреса равен 1, для отображения памяти программ на память данных *(Program Space Visibility Area, PSVA).* Данные в памяти выравниваются по границе слова, при этом младшие байты имеют четные, а старшие — нечетные адреса.

Микроконтроллеры семейства *dsPIC30* имеют интегрированную память *EEPROM,* которая отображена в соответствующем секторе программной памяти. Семейства *dsPIC33, PIC24F/H* не имеют *EEPROM,* что связано с изменением технологии изготовления кристаллов. Поэтому при необходимости наличия в системе энергонезависимой памяти малого объема с большим количеством циклов перепрограммирования рекомендуется использовать внешние микросхемы памяти *EEPROM* с последовательным интерфейсом.

Режимы адресации и система команд.

16-битная архитектура *Microchip* имеет в своем составе блок из 16 рабочих регистров *W0...W15.* Все регистры ортогональны с точки зрения системы команд, то есть могут быть использованы в качестве операнда инструкции. Часть регистров имеет служебные функции: *W15* - это указатель стека, *W14* - указатель стекового фрейма. Некоторые инструкции могут использоваться в качестве операндов или регистров сохранения результата только в определенный W- регистр или регистровую пару.

16-битные микроконтроллеры *Microchip* имеют расширенный набор инструкций, большинство из которых поддерживает операции типа «чтение-модификация-запись», что позволяет работать с данными напрямую в ОЗУ, не используя рабочие регистры. Большинство инструкций являются трехоперандными и могут работать как с байтами, так и с 16-битными словами.

Семейства с ASP-ядром *(dsPIC30/33)* имеют 83 инструкции (включая инструкции *DSP-ядра),* семейства без *DSP-ядра (PIC24F/H)* - 76.

Все инструкции выполняются за один командный такт за исключением:

- инструкций изменения программного потока - 2 или 3 такта;
- инструкций табличного чтения/записи - 2 такта;
- инструкции *MOV.D* - 2 такта;
- инструкции *DO* - 2 такта.

Все инструкции можно условно разделить на несколько групп:

- *MOVE* - инструкции перемещения данных;
- *MATH* - инструкции выполнения математических операций;
- *LOGIC* - инструкции выполнения поразрядных логических операций;
- *SHIFT/ROTATE* - инструкции сдвига с переносом и без переноса;
- *BIT* - инструкции работы с битами;
- *STACK* - инструкции работы со стеком;
- *PROGRAMM FLOW* - инструкции изменения программного потока (переходы, вызовы, условные переходы);
- *CONTROL* - инструкции управления ядром (инициализация аппаратных циклов, программный сброс, перевод контроллера в энергосберегающий режим);

- *DSP* - инструкции DSP-ядра, присутствуют только в системе команд контроллеров *dsPIC30/33.*

Архитектура поддерживает большое количество методов адресации:

1. Непосредственная адресация, позволяющая задавать значение

операнда в команде. Таким образом, в команде содержится не адрес операнда, а непосредственно сам операнд. При непосредственной адресации не требуется обращения к памяти для выборки операнда и ячейки памяти для его хранения. Это способствует уменьшению времени выполнения программы и занимаемого ею объёма памяти. Непосредственная адресация удобна для хранения различного рода констант. Операнд должен быть расположен в области *Near Space*:

; Сложение значения 0x900 с W0.

add #0x900, W0, W1 ; Результат сохраняется в W1

Прямая адресация, когда часть команды является адресом

операнда (обрабатываемого слова) в памяти. Если известен адрес

операнда располагающегося в памяти:

; Помещение b в регистр W1. mov b, W1

Регистровая адресация, когда операнд находится в одном из

регистров процессора. Операнды могут располагаются в любых регистрах общего назначения или сегментных регистрах.

; Поразрядное логическое ИЛИ регистров W0 и W2, ior W0, W2, W5 ; сохранение результата в W5

Регистровая косвенная адресация, когда адрес операнда извлекается из регистра процессора:

add W4, [W5], [W6]

; сложение W4 со значением, адрес которого хранится в ; W5, сохранение результата по указателю W6

Косвенная адресация с пре/пост инкрементом и декрементом, при этом учитываются правила адресной арифметики в зависимости от типа операнда (байт или слово):

mov [++W0], [W1--], W6 ; Увеличение указателя W0 на 2, перемещение значения ; по указателю W0 в ячейку, на которую указывает W1,

; уменьшение указателя W1 на 2,

; сохранение результата в W6

косвенная адресация со смещением: mov [W4+W5], [W6++], W2 ; Получение указателя на ячейку операнда,

; путем сложения W4 и W5, перемещение значения ; по полученному указателю по адресу W6,

; увеличение значения W6 на 2,

; сохранение результата в W2

Поддержка различных методов адресации позволяет получать очень компактный код при использовании компиляторов с языков высокого уровня. Следует также отметить возможность вызова и перехода по значению в регистре (аналог вызова функции по указателю в Си) и инструкцию запрещения прерываний на определенное количество командных тактов (позволяет выполнять набор инструкций как одну атомарную).

ПРАКТИЧЕСКАЯ РАБОТА №22. КОММУНИКАЦИОННЫЕ МИКРОКОНТРОЛЛЕРЫ (ЦИФРОВЫЕ СИГНАЛЬНЫЕ ПРОЦЕССОРЫ (ЦСП) ФИРМЫ *MICROCHIP*).

Цель работы: изучить основы программирования цифровых сигнальных процессоров (ЦСП) фирмы *Microchip.* Научиться создавать новый проект, настраивать параметры проекта, компилировать проект.

Теоретическая часть

Для программирования ЦСП фирма *Microchip* предоставляет разработчику программную среду *MPLAB IDE.* Данная программная среда позволяет выполнить весь спектр операций при создании работоспособного кода для ЦСП. Разработчик имеет возможность создать программный код как на языке низкого уровня - ассемблер, так и на языке высокого уровня - СИ. Выбор языка программирования выполняется на этапе создания проекта. Также, в особых случаях, существует возможность использования двух языков программирования. После написания программы, ее компиляции (создания машинного кода по написанному разработчиком ассемблерному или СИ коду), имеется возможность пошаговой отладки программного кода. Процесс отладки представляет собой важный этап в разработке программного кода, при котором выявляются ошибки невидимые компилятором (ошибки, не являющиеся синтаксическими), например взаимодействие отдельных блоков кода программы, взаимные противоречия и т.п. Уменьшить возможные логические ошибки помогает детальная схема алгоритма программного кода. Последним этапом является программирование ЦСП откомпилированным машинным кодом. Для программирования ЦСП в работе используется внутрисхемный отладчик *ICD2.* Он может работать как программатор, так и внутрисхемный отладчик. В режиме работы *ICD2* как внутрисхемного отладчика пользователь может отлаживать (прогонять) программу непосредственно в готовом устройстве. Существуют ошибки программного кода, которые возможно выявить только при использовании внутрисхемного отладчика.

Практическая часть

1. Запустите *MPLAB IDE.*
2. Выберите меню <Project\Project Wizard.. >.
3. На первом шаге необходимо выбрать процессор *dsPIC33FJ256GP710.*
4. На втором шаге выберите язык программирования - *Microchip ASM30 Toolsuite.*
5. На третьем шаге необходимо определить каталог нового проекта и название. Следует отметить, что путь должен быть только из латинских букв и не слишком длинным.
6. На четвертом шаге в проект включаются уже готовые модули. Пока пропустите данный этап.

После выполнения вышеприведенных шагов экран будет выглядеть, как показано на рисунке 4. В данном случае в окне с названием проекта нет ни одного рабочего файла.

Подключение библиотек и упаковочных файлов процессора

1. Нажмите правой клавишей по полю с надписью <*Linker Scripts*>.
2. *Выберите* <AddFiles.. >.
3. *Необходимо подключить файл* <p33FJ256GP710.gld>, *расположенный в каталоге* <C:\ Program Files\Microchip \MPLAB ASM30 Suite \Support\dsPIC33F\gld>
4. Нажмите правой клавишей по полю с надписью <*Header Files*>.
5. *Выберите* <AddFiles.. >.

fi| laM - MPLAB IDE v7.60 |T|[п]^	
File Edit View Project Debugger Programmer Tools Configure Window Help	
D [£ H & Ъ a S H i f Release v i Q I O # И *Ш*	
dlabl.mcw _	
0 D labi.mcp 1 Source Files 1 Header Files CD Object Files 1 Library Files 1 Linker Scripts CD Other Files	
□ Files Symbols	

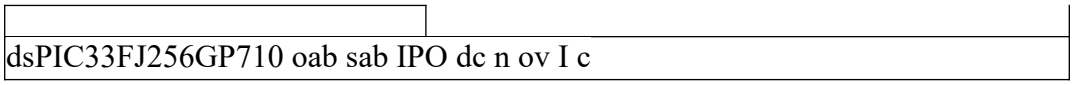


Рисунок 4 - Вид программы после создания нового проекта

6. Необходимо подключить файл <*p33FJ256GP710.inc*>, *расположенный в каталоге* <C:\ Program Files\Microchip\MPLAB ASM30 Suite\Support\dsPIC33F \inc>

После подключения необходимых библиотек окно программы будет выглядеть подобно рисунку 5.

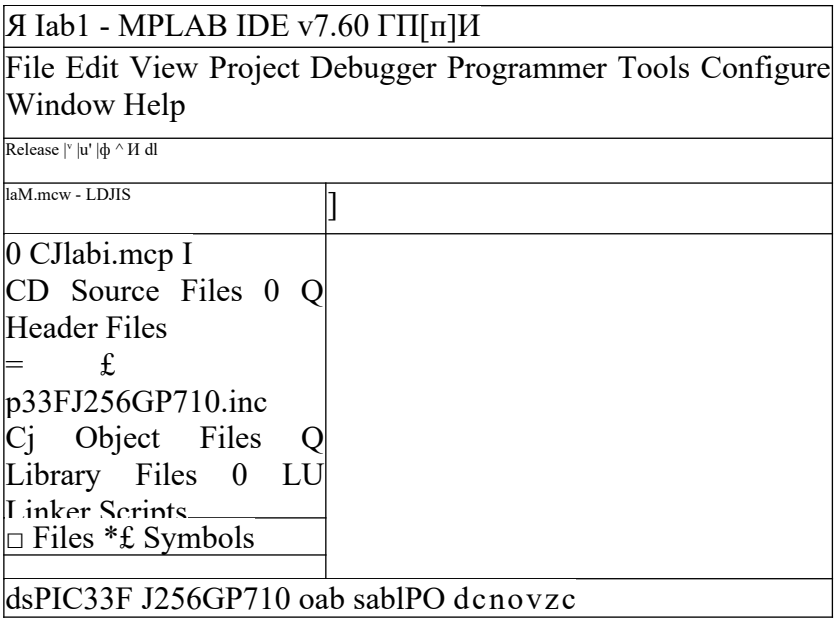


Рисунок 5 - Вид программы после подключения библиотек и упаковочных файлов процессора

Создание файла с исполняемым кодом

1. Выберите меню <*File\New*>
2. Запишите его с именем <*main.s*> в каталог с программой, т.е. *D:\brig1\lab1\lab1*
3. Нажмите правой клавишей по полю с надписью <*Source Files*>.
4. *Выберите* <AddFiles.. >.
5. Необходимо подключить файл <*main.s*>.
6. Введите в файл <*main.s*> следующий код:

```
.include "p33FJ256GP710.inc"

    reset:    .global     reset
              .text
              nop



              mov         #0, W1
main:         mov         #100, W2

              inc         W1, W1
              inc         W2, W2
              add         W1, W2, W3
```

```
.end                goto        main
```

В первой строке подключается заголовочный файл процессора *<p33fj256gp710.inc>*. Во второй, метке *<reset>* назначается глобальный статус, по этой метке код привязывается к стартовому адресу. Далее директива *<.text>,* которая определяет следующий за ним текст как исполняемый код. Далее идет ассемблерный код. Заканчивается ассемблерный код директивой *<.end>*.

При правильном выполнении операций вид программы будет соответствовать рисунку *6.*

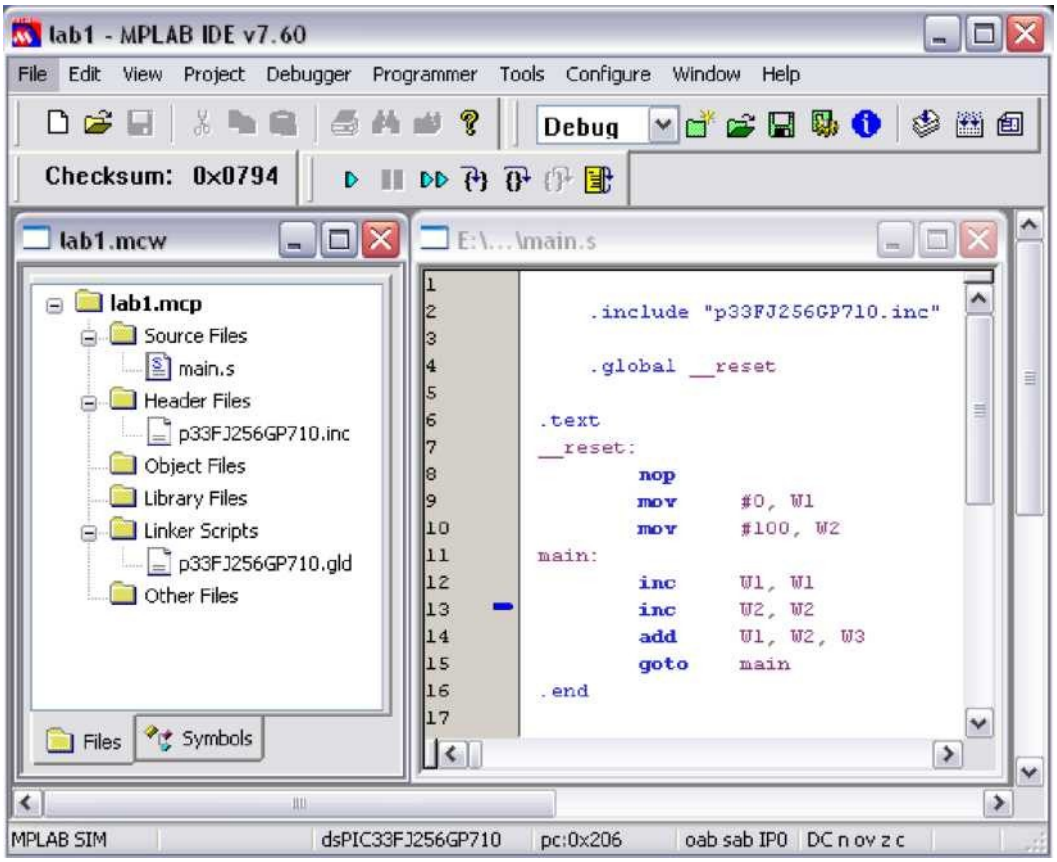


Рисунок 6 - Создание файла с исполняемым кодом

Задание для самостоятельного выполнения

1. Создайте новый проект. Процессор - *dsPIC33FJ256GP710.*
2. Подключите необходимые библиотеки и перепишите программу, приведенную выше.
3. Необходимо подключить отладчик (симулятор), встроенный в среду *MPLAB IDE.* Выберите в меню <*Debugger\Select Tool\MPLAB SIM*>.
4. Скомпилируйте проект, для этого выберите в меню <*Project\Build All*>. Если код написан правильно и ошибок при компиляции нет, то можно открыть окно памяти программ, в котором написанная программа представиться в машинных кодах по жестко прописанным ячейкам памяти. Нажмите в меню <*View\Program Memory*>.
5. Откройте окно *Watch,* которое предназначено для отображения состояния необходимых переменных в режиме отладки программы. Для этого выберите в меню <*View\ Watch*>. Добавьте в окно *Watch* регистры, которые мы хотим наблюдать в ходе выполнения программы. Для этого в левом окошке найдите *WREG1* и нажмите кнопку <*Add SFR*>. Далее добавьте регистры *WREG2, WREG3.* Результат представлен на рисунке 7.

Watch

Add SFR	ACCA	v Add Symbol SP [v|
Address		Symbol... Value Decimal
□ □□2		WREG1 Ж 0x0007 7\
□ 004 WREG2 0x00 6B 107 0006 WREG3 0x0072 114		
| Watch	Watch 2 Watch 3 Watch 4	

Рисунок 7 - Окно Watch

6. Далее приступите к отладке кода. Перейдите в окно <*Program Memory*>, причем окна должны так располагаться, чтобы вы могли наблюдать за изменением состояния регистров в окне <*Watch*>. При каждом нажатии на клавишу <*F7* > ассемблерный код будет выполняться по

одной команде. Исследуйте изменение регистров согласно пошаговому выполнению программы.

7. Для того чтобы программа выполнялась не в пошаговом режиме (для отладки), а в обычном необходимо нажать клавишу <*F9*>. Так же, если необходимо прервать программу нажмите <*F5*>, а если установить начальный (стартовый) адрес, следует нажать клавишу <*F6*>.

Контрольные вопросы

1. На каком языке программирования можно написать код для процессоров, рассматриваемых в работе?
2. Назначение *MPLAB SIM.*
3. Назначение *MPLAD IDE.*
4. Объясните процесс компиляции проекта.
5. Назначение окна *Watch.*
6. Назначение окна *Program Mem* ory.
7. С помощью какой инструкции осуществляется инкремент содержимого регистра?
8. С помощью какой инструкции осуществляется декремент содержимого регистра?

ПРАКТИЧЕСКАЯ РАБОТА №23. КОММУНИКАЦИОННЫЕ МИКРОКОНТРОЛЛЕРЫ (РЕГИСТРЫ МИКРОПРОЦЕССОРА).

Цель работы: изучить основные регистры микропроцессора. Научиться работать с битовыми и логическими инструкциями.

Теоретическая часть.

Работа микропроцессора, а также его основных блоков (АЦП, таймеры и т.д.) полностью зависят от настроек, которые хранятся в регистрах. Рассмотрим основные регистры микропроцессора.

Рабочие регистры

В процессоре имеется 16 рабочих регистров *(W0 - W15),* которые могут использоваться как регистры, содержащие данные или команды, а также как регистры смещения *(offset registers).* Рабочие регистры 16-ти разрядные.

Регистры управления стеком

Регистры *W14/Frame Pointer, W 15/Stack Pointer, SPLIM* управляют работой программного стека. Если стек не используется, то рабочий регистр *W14* можно использовать для работы.

Аккумуляторы процессора

Аккумулятор *A (ACCA)* и аккумулятор *B (ACCB)* имеют 40 разрядов. В эти регистры записывается результат выполнения арифметических и сдвигающих операций. Каждый аккумулятор располагается в 3 регистрах памяти:

- *AccxU* (39-42 биты);
- *AccxH* (31-16 биты);
- *AccxL* (15-0 биты).

Программный счетчик

Программный счетчик (program counter) имеет 23 разряда. Инструкции могут адресовать 4M*24 битной пользовательской программной памяти программным счетчиком.

Табличные регистры

Регистр *TBLPAG, PSVPAG* используется при операциях записи данных в память программ.

Регистры управления циклом

Регистры *RCOUNT, DCOUNT, DOSTART, DOEND* предназначены для организации циклов в выполняемом коде.

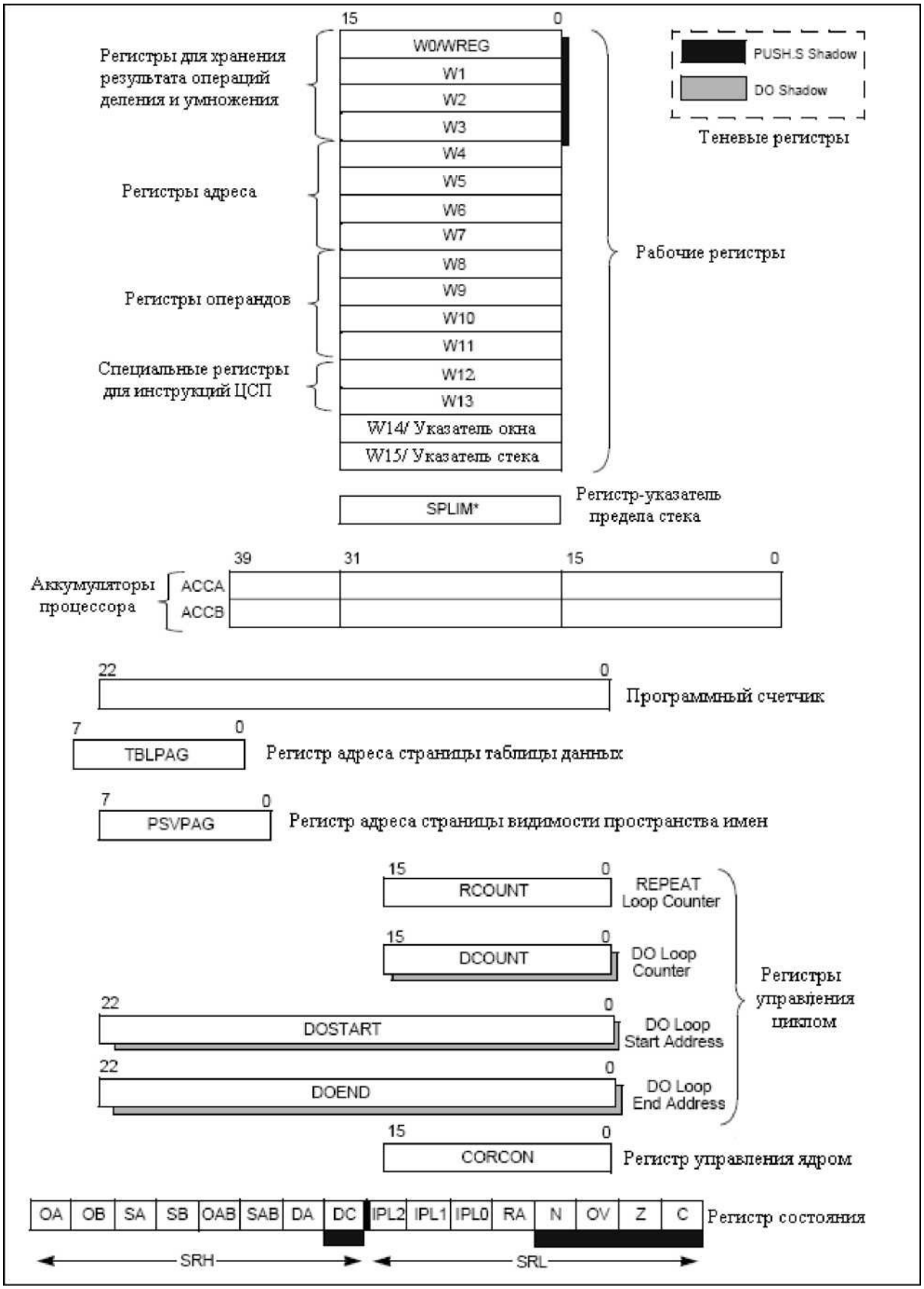


Рисунок 8 - Регистры микропроцессора

Регистр управления ядром процессора

Регистр *CORCON* - настраивает работу процессора (выбор между аккумуляторами,

настройка уровня выполнения прерываний).

Регистр состояния

В регистре состояния *(status register)* располагаются битовые поля, характеризующие результат выполнения последней операции:

- бит Z - установлен, если результат последней операции нулевой;
- бит *C* - установлен, если в последней операции был заем бита;
- бит *N* - установлен, если результат операции инверсный;
- бит *OV* - установлен, если в последней операции было переполнение.

Рассмотрим основные битовые и логические операции.

Битовые операции

К основным битовым операциям относятся:

- установка бита - *BSET;*
- сброс бита - *BCLR;*
- инверсия бита - *BTG.*

Логические операции

К основным битовым операциям относятся:

- логическая операция <и> - *AND;*
- логическая операция <или> - *IOR;*
- логическая операция <исключающее или> - *XOR;*
- логическая инверсия -*COM;*
- сброс всех битов регистра - *CLR;*
- установка всех битов регистра - *SETM.*

Синтаксис битовых и логических операций подробно рассмотрен в таблицах 4 и 6 приложения 1.

Пример программы, реализующий логическую функцию: *d = (a and 0xF00F) or (b and c).*

```
.include "p33FJ256GP710.inc"
; - Объявление переменных
.equ                b, 0x802
.equ                c, 0x804
.equ                d, 0x806
.text
.global ,                   reset
        reset:

;                  -переменных
mov                 #0x1111, W1
mov                 W1, a
mov                 #0x2222, W1
mov                 W1, b
mov                 #0x3333, W1
mov                 W1, c
; - Реализация логической функции
mov                 a, W1
mov                 #0xF00F, W2
and                 W1, W2, W3
mov                 b, W1
mov                 c, W2
and                 W1, W2, W4
ior                 W3, W4, W1
mov                 W1, d
```

```
nop
.end
```

Для объявления констант *a, b, c* и *d* служит директива *.equ.* В данном примере для хранения этих констант выделяется область памяти данных, начиная с ячейки *0800h.* Операция *nop* - пустая операция.

Практическая часть

1. Создайте новый проект. Процессор - *dsPIC33FJ256GP710.*
2. Подключите необходимые библиотеки и перепишите программу, приведенную выше.
3. Необходимо подключить отладчик (симулятор), встроенный в среду *MPLAB IDE.* Выберете в меню *<Debugger\Select*

4. Создайте схему алгоритма и напишите программу, выполняющую битовые операции.

№ Варианта	Задание					
	Установить		Сбросить		Инвертировать	
	Регистр	Бит	Регистр	Бит	Регистр	Бит
1	W1	15	W2	0	W3	3
2	W2	14	W3	2	W4	1
3	W3	13	W4	4	W5	2
4	W4	12	W5	6	W6	5
5	W5	11	W6	8	W7	4
6	W6	10	W7	10	W8	7
7	W7	9	W8	12	W9	6
8	W8	8	W9	14	W0	9
9	W9	7	W0	15	W1	8
10	W0	6	W1	13	W2	10
11	W1	5	W2	11	W3	12
12	W2	4	W3	9	W4	11

5. Откройте окно *Watch* и внесите в него все регистры, которые используются в коде. В пошаговом режиме отладьте код, контролируя изменение регистров в окне *Watch.* После отладки программы, покажите код и результаты работы программы преподавателю.
6. Создайте схему алгоритма и напишите программу, выполняющую заданную логическую функцию. Переменные *a, b, c* имеют разрядность 16 бит.
7. Выполнить пункт 5 для второй программы.

№ Варианта	Логическая функция
1	d = (a and 0x000F) or (b and 0x00F0) or (c and 0x0F00)
2	d = [not(a and 0x0F00)] or (b and 0x00F0) or (c and 0x000F)

3	d = (a or 0x000F) and (b or 0x00F0) and (c or 0x0F00)
4	d = [not(a)] or (b and 0x00F0) or (c and 0x0F00)
5	d = (a and 0x00F0) or [not(b) and 0xF000] or (c and 0x0F00)
6	d = [not(a) and 0x000F] or [not(b) and 0x00F0] or [not(c) and 0x0F00]
7	d = not[not(a and b) or (c and 0x5A5A)]
8	d = (a and 0xAAAA) or (b and 0x5555) or c
9	d = [not(a and c)] or (b and 0x3333)
10	d = (a and 0x8888) or (b and 0x6666) or (c and 0x1111)
11	d = not(a) or (b and 0xF0F0) or (c and 0x0F0F)
12	d = (a and b) or (c and 0xAAAA)

Контрольные вопросы

1. Назначение рабочих регистров.
2. Как обозначаются рабочие регистры?
3. Назовите регистры, которые управляют работой АЦП, таймером, портом ввода вывода А?
4. Назовите основные битовые инструкции.
5. Назовите основные логические инструкции.
6. Поясните операции:

```
mov   #0xA, W1;
and   W1, W2, W3;
ior   W4, W5, W6;
xor   W7, W8, W9;
bset  W1, #7;
bclr  W2, #15;
btg   W3, #1.
```

ПРАКТИЧЕСКАЯ РАБОТА №24. КОММУНИКАЦИОННЫЕ МИКРОКОНТРОЛЛЕРЫ (КОМАНДЫ ПЕРЕСЫЛКИ ДАННЫХ И АРИФМЕТИЧЕСКИЕ КОМАНДЫ).

Цель работы: изучить и приобрести практические навыки с командами пересылки данных и арифметическими инструкциями.

Теоретическая часть.

К основным арифметическим операциям относятся.

Сложение - *ADD*;

- Вычитание *SUB;* Умножение - *MUL;*
- Деление - *DIV;*
- Инкремент - *INC, INC2;*
- Декремент - *DEC, DEC2.*

Синтаксис написания арифметических инструкций приведен в таблице 3 приложения 1. Рассмотрим основные инструкции подробно.

Инструкция сложения

Синтаксис:

{label:} ADD{.B} Wb, Ws, Wd [Ws], [Ws++], [Ws--], [+ + Ws], [--Ws], [Wd++] [Wd--] [+ + Wd] [-- Wd]

[W0... W15]

[W0... W15]

[W0... W15]

Операция: Wb +Ws ^ Wd

Изменяемые биты регистра состояния: *DC, N, OV, Z, C.*

Пример:

ADD.B W5, W6, W7;

Складывает *W5* и *W6,* результат записывается в *W7,* символ *«B»* задает 8-ми битовый режим сложения.

ADD W5, W6, W7;

Складывает *W5* и *W6*, результат записывается в *W7* (16-ти битный режим).

Инструкции умножения

Синтаксис:

{label:} MUL.SS Wb, Ws, Wnd

[Ws]

[Ws++]

[Ws--]

[+ + Ws]

[--Ws]

Операнды: *Wb* [6] *[W0 ... W15]*

Ws 6 [W0 ... W15]

Wnd 6 [W0, W2, W4 ... W12]

Операция: знаковое (Wb) * знаковое *(Ws) ^ Wnd: Wnd + 1* Изменяемые биты регистра состояния: нет

Пояснение: При умножении *Wb* и *Ws* результатом является 32-х битное слово. Результат умножения записывается в *Wnd* (младшее слово) и *Wnd+1* (старшее слово).

Пример:

MUL.SS W0, W1, W12 Умножение *W0* и *W1,* результат записывается в *W12:W13.*

MUL.SS W2, [--W4], W0

- Содержимое регистра *W4* уменьшается на единицу.
- Умножается число из регистра *W2* с числом расположенным в ячейке памяти, адрес которой содержится в регистре *W4.*
- Результат записывается в *W0* (младшее слово) и *W1* (старшее слово).

Инструкция вычитания

Синтаксис:

{label:} SUB Wb, Ws, Wnd

Операция: Wnd=Wb-Ws.

Инструкция деления

Синтаксис:

{label:} DIV Wm, Wn

Операция: *Wm/Wn ^ W0:W1*, то есть *Wm* делится на Wn, результат записывается в регистровую пару *W0:W1.* Причем в *W0* помещается целая часть, а в *W1* - остаток от деления.

Следует заметить, что семейство *PIC24* не имеет аппаратного делителя как такового. Однако АЛУ семейства имеет аппаратную поддержку деления (инструкция *DIV).* Использование инструкции *DIV* в сочетании с инструкцией аппаратного цикла *REPEAT* позволяет производить

итерационную операцию деления за 18 командных тактов.

Пример программы, реализующий арифметическую функцию:

e = (a + b) * c / d.

```
.include "p33FJ256GP710.inc"
.equ                    a,      0x800
.equ                    b,      0x802
.equ                    c,      0x804
.equ                    d,      0x806
.equ                    e,      0x808
.text
.global _               _reset
        reset:
; - Инициализация      переменных
mov                     #-3     , W0
mov                     W0,     a
mov                     #21     , W0
mov                     W0,     b
mov                     #10000, W0
```

```
mov                     W0, c
mov                     #-37, W0
mov                     W0, d
; - Реализация          арифметической функции
mov                     a, W0
mov
add
mov
mul.ss
mov                     d, W2
repeat                  #17
div.sd                  W0, W2
mov
nop
.end
```

Задание для самостоятельного выполнения

1. Создайте новый проект. Процессор - *dsPIC33FJ256GP710.*
2. Подключите необходимые библиотеки и перепишите программу, приведенную выше. Разберитесь в ее работе.

3. Необходимо подключить отладчик (симулятор), встроенный в среду *MPLAB IDE.* Выберите в меню <*Debugger\Select Tool\MPLAB SIM*>.

4. Создайте схему алгоритма и скорректируйте код программы, выполняющий заданную арифметическую функцию. Переменные

a, b, c, d - знаковые 16-ти разрядные числа.

5. Откройте окно *Watch* и внесите в него все регистры, которые используются в коде. В пошаговом режиме отладьте код, контролируя изменение регистров в окне *Watch.* После отладки программы, покажите код и результаты работы программы преподавателю.

№ Варианта	Арифметическая функция
1	e = (a + 1) * (b - 100) / (c + 1000) * (d - 1000)
2	e = (a + b) * (c - b) / (c + d)
3	e = (a - b) * (c / d) + 12345
4	e = (100 - a) / (100 - b) * (c + d) - 4321
5	e = (a + b - c) * d / (b - a) + 9876
6	e = (a * b + c) * (b - 100) / d
7	e = a * (b - c) / (c - d) + 12346
8	e = a* b + b / c - c * d - 13579
9	e = (a * b + 100) / (c + d - 100)
10	e = (a + 1000) * c + (b - 1000) / d
11	e = (a - d) * (b - c) / d + 2468
12	e = a *d * (b + 1000) / c - 1357

Контрольные вопросы

1. Назовите основные арифметические инструкции.
2. Назовите способы адресации, используемые в командах вашей программы.
3. Назовите количество тактов, за которые выполняются арифметические операции.
4. В каких регистрах хранится результат операции деления?
5. Назовите варианты операций умножения.
6. Чем команда *ADD* отличается от команды *ADD.B?*
7. Почему в примере перед командой *DIV.SD* стоит команда *repeate #17?*

ПРАКТИЧЕСКАЯ РАБОТА №25. КОММУНИКАЦИОННЫЕ МИКРОКОНТРОЛЛЕРЫ (ОРГАНИЗАЦИЯ ЦИКЛОВ).

Цель работы: изучить и приобрести практические навыки с командами организации циклов и инструкциями сравнения.

Теоретическая часть

Организация цикла

К основным командам организации циклов относятся *DO* и *REPEAT*. Рассмотрим

основные отличия данных инструкций.

```
mov         #0, W0
mov         #mas, W1
do #15,     end
inc         W0, W0
mov         W0, [W1++]
end:        nop
```

В данном примере используется инструкция *DO*. Первыми двумя командами инициализируются рабочие регистры: *W0* обнуляется, а в регистр *W1* заносится начальный адрес массива *mas*. Следующая команда определяет начало цикла *DO*, количество итераций равное 16 (нумерация идет с нуля, поэтому в программе указывается *#15)*. Цикл заканчивается меткой *end*. Между началом и концом цикла может находиться необходимое количество команд. Следующая команда инкрементирует регистр *W0,* а далее значение *W0* записывается в массив *mas*. Указатель массива *W1* инкрементируется в процессе выполнения операции *mov*.

mov #mas1, W1 mov #mas2, W2 repeat #15 mov [W1++], [W2++] nop

В примере используется инструкция *REPEAT*. Основное отличие данной инструкции от *DO* что она выполняет в цикле только следующую за ней инструкцию необходимое количество раз.

Также компилятор поддерживает другое написание данных инструкций. В них вместо заданного числа *(#15)* можно использовать рабочий регистр. Но перед использованием данной конструкции

команды необходимо предварительно задать значение этого рабочего регистра.

Инструкции сравнения

Полный список инструкций сравнения представлен в таблице 7 приложения 1. Рассмотрим некоторые из них.

mov #100, W0 mov #-100, W1 cpsgt W0, W1 goto _if_ nop nop _if_: nop nop

Инструкции переходов

В примере используется инструкция *CPSGT*. Если *W0* больше *W1,* то следующая инструкция за командой *CPSGT* пропускается. Таким образом, в данном примере после выполнения инструкции сравнения, пропускается инструкция безусловного перехода и выполняется операция *nop.*

mov #100, W0 mov #10, W1 cpslt W0, W1 goto _if_ nop nop _if_: nop nop

В примере используется инструкция *CPSLT*. Если *W0* меньше *W1,* то следующая инструкция за командой *CPSLT* пропускается.

Таким образом, в данном примере после выполнения инструкции сравнения, выполняется инструкция безусловного перехода на метку *_if*, далее выполняется операция *nop.*

mov #100, W0 mov #100, W1 cpsne W0, W1 goto _if_ nop nop _if_: nop nop

В примере используется инструкция *CPSNE*. Если *W0* не равно *W1,* то следующая инструкция за командой *CPSNE* пропускается. Таким образом, в данном примере после выполнения инструкции сравнения, выполняется инструкция безусловного перехода на метку *_if_,* далее выполняется операция *nop.*

Задание для самостоятельного выполнения

1. Создайте новый проект. Процессор - *dsPIC33FJ256GP710.*
2. Подключите необходимые библиотеки.
3. Подключите отладчик (симулятор), встроенный в среду *MPLAB* IDE.
4. Выполните задание. Создайте схему алгоритма программы

№ Варианта	Задание
1	Разработать программу, которая находит сумму всех нечётных элементов массива. В массиве 7 элементов.
2	Разработать программу, которая находит сумму элементов массива с чётными индексами. В массиве 9 элементов.
3	Разработать программу, которая складывает чётные и вычитает все нечётные элементы массива. В массиве 8 элементов.
4	Разработать программу, которая находит сумму всех элементов, которые делятся на три без остатка. В массиве 12 элементов.
5	Разработать программу, которая находит сумму элементов массива с нечётными индексами. В массиве 11 элементов.
6	Разработать программу, для вычисления суммы тех элементов массива, значения которых не меньше двенадцати. В массиве 7 элементов.
7	Разработать программу, которая находит сумму всех чётных элементов массива. В массиве 10 элементов.
8	Разработать программу для вычисления суммы тех элементов массива, значения которых не больше 47. В массиве 10 элементов.
9	Разработать программу, которая складывает элементы массива с чётными индексами и вычитает все элементы с нечётными индексами. В массиве 12 элементов.
10	Разработать программу, которая складывает нечётные и вычитает все чётные элементы массива. В массиве 7 элементов.
11	Разработать программу для вычисления суммы элементов массива, значения которых больше 10 и меньше 78. В массиве 9 элементов.
12	Разработать программу, которая суммирует каждый третий элемент массива. В массиве 16 элементов.

5. Откройте окно Watch и внесите в него все регистры, которые используются в коде. В пошаговом режиме отладьте код, контролируя изменение регистров в окне Watch. После отладки программы, покажите код и результаты работы программы преподавателю.
6. Подготовьте отчет в соответствие с требованиями на стр. 21.

Контрольные вопросы

1. Назовите основные инструкции сравнения.
2. Назовите способы адресации, используемые в командах вашей программы.
3. Назовите количество тактов, за которые выполняются инструкции сравнения.
4. Можно ли использовать цикл *REPEAT,* если в теле цикла несколько команд?
5. Можно ли использовать цикл *DO,* если в теле цикла одна команда?
6. Можно ли использовать циклы *DO* и *REPEAT,* если заранее неизвестно количество итераций цикла?

ПРАКТИЧЕСКАЯ РАБОТА №26. ПРОГРАММНОЕ УПРАВЛЕНИЕ ПОРТАМИ ВВОДА/ВЫВОДА МИКРОКОНТРОЛЛЕРА PIC16F84A

Цель работы: Освоить базовые принципы управления портами ввода-вывода PIC-контроллеров, в среде MPLAB IDE научиться пользоваться асинхронными внешними

воздействиями (стимулами) и использовать их для отладки программ работы с портами ввода-вывода.

Теоретическая часть

Микропроцессорное устройство управления светодидом и генерации звука

Устройство состоит из двух кнопок S1 и S2, светодиода D1, генератора тактовых импульсов частотой 4 МГц с кварцевым резонатором XTAL1, динамика LS1, усилителя звуковых колебаний на базе транзистора VT1, токоограничивающих резисторов и частотозадающих конденсаторов. Управление работой устройства осуществляется с помощью микроконтроллера PIC16F84A. Заметим, что каждый вывод МК семейства PIC может непосредственно управлять подключенным к нему светодиодом без дополнительных усилителей.

Когда включается питание, МК по умолчанию устанавливает все разряды портов A и B на ввод и начинает выполнять управляющую программу с адреса 00h.

Управление миганием светодиода

Ниже приведен базовый код программы мигания светодиода, написанной на языке ассемблера MPASM.

Базовый код программы управления светодиодом

```
; === Секция заголовка ===
list p=PIC16F84A
include <P16F84A.INC>
  __config _CP_OFF& _WDT_OFF& _PWRTE_ON& _XT_OSC
```

cnt1 cnt2	equ equ	0x0C 0x0D 0x0E	; счетчик циклов 1 ; счетчик циклов 2 ; регистр светодиода
ledset equ			

```
; === Рабочая секция ===
 org      0x00
 goto     start
 org
 start
 clrf
 clrf
 bsf
 movlw                                          0x08
 movwf
 movlw
 movwf
 bcf
 bcf
 clrf
 PORTA
 PORTB
 STATUS,RP0                                     ; очистить порт A
 b'11111101'                                    ; очистить порт B
 TRISA                                          ; выбрать банк 1
 b'00000101'
 TRISB
 ; установить линию RA1 порта A на выход
 ; установить линии RP7:PR3 и RP1 порта B на выход
 OPTION_REG,7                                   ; включить подтягивающие резисторы
                                                ; выбрать банк 0
```

```
                                                    ; очистить регистр светодиода
STATUS,RP0
ledset
;  ---  Бесконечный  цикл  включения/выключения
светодиода ---
loop
movlw
xorwf                                               ; включить/выключить светодиод
movf
movwf
b'00000010'
ledset
ledset,W
PORTA
; --- Программная задержка ---
;movlw
movlw
movwf                                               0xFF
cycle1                                              0x03
;movlw                                              cnt1
movlw
movwf
cycle2
0xFF
0x03
cnt2
decfsz                                  cnt2
goto                                    cycle2
decfsz                                  cnt1
goto                                    cycle1
; --- Конец программной задержки ---
goto
                                        loop        ; бесконечный цикл
end
```

где строка __config _CP_OFF&_WDT_OFF&_PWRTE_ON&_XT_OSC – директива ассемблера, под руководством которой в выходной файл заносится информация о слове состояния процессора. Она означает:

• _CP_OFF - бит защиты кода после программирования не устанавливать;
• _WDT_OFF - сторожевой таймер отключен;
• _PWRTE_ON - таймер задержки сброса при подаче питания включен;
• _XT_OSC - тип резонатора: керамический или кварцевый (кроме XT также могут быть значения LP, HS или RC, в соответствии с резонатором в схеме).

Как работает эта программа? Указав ассемблеру тип процессора и соответствующий ему файл включений P16F84A.INC, мы описываем, в каких ячейках ОЗУ (регистрах общего назначения) будут храниться значения переменных программы, а затем настраиваем порты ввода-вывода.

Для этого сначала обнуляются значения в выходных защелках портов. В данной программе мы могли бы этого и не делать, но правильный стиль программирования PIC-микроконтроллеров требует, чтобы значения в выходных защелках были явно определены перед тем, как некоторые линии будут настроены на вывод. Затем, установив в 1 бит RP0 регистра STATUS, получаем доступ к регистровому банку 1 и, обращаясь к регистрам состояния портов

TRISA и TRISB, настраиваем часть линий на ввод, а часть на вывод согласно схеме микропроцессорного устройства (рис.1). После чего устанавливаем бит RP0 регистра STATUS обратно в 0, тем самым возвращаясь к банку памяти 0, и обнуляем переменную ledset.

Далее (метка loop) при помощи операции XOR ′Исключающее ИЛИ′ инвертируем бит <1> переменной ledset, сохраняем полученное значение обратно в ledset (команда xorwf ledset) и, скопировав результат в аккумулятор (команда movf ledset,w), выводим значение ledset в выходную защелку порта A. В результате на линии RA1 микроконтроллера появится уровень напряжения, соответствующий текущему значению бита <1> переменной ledset.

Затем инициализируем переменную внешнего цикла cnt1. Внутрь этого цикла вложен еще один цикл с переменной cnt2. Назначение этих циклов - сформировать программную задержку, в течение которой на выводе RA1 порта A удерживается установленное значение ledset. Когда задержка исчерпана, в порт выводится инверсное значение, и цикл повторяется.

Когда вы наберете текст программы, сохраните его под произвольным именем в файле с расширением .asm, создайте в MPLAB новый проект и добавьте в него этот файл. Затем скомпилируйте программу (F10), выполните в команду меню **Debugger/Animate** и наблюдайте, как изменяется состояние линии RA1 порта A во время выполнения программы.

Если вы введете во внутренний цикл три-четыре холостых команды NOP, то задержка заметно увеличится, соответственно уменьшится частота мигания светодиода.

Попробуйте также увеличить частоту мигания, уменьшая начальные значения переменных cnt1 и cnt2. Но учтите, что если установить частоту мигания больше 20...25 Гц, то из-за инерционности зрения будет казаться, что светодиод горит непрерывно, хотя программа работает правильно.

Управление генерацией звука

Ниже приведен базовый код программы генерации звука, написанной на языке ассемблера MPASM. Поскольку для генерации импульсов звуковой частоты нужна относительно небольшая временная задержка, программа оказалась еще проще, чем для мигающего светодиода. Понадобилось сделать лишь небольшие изменения созданной ранее программы управления светодиодом.

Базовый код программы генерации звука

```
        ; === Секция заголовка ===
        list p=PIC16F84A
        include <P16F84A.INC>
        __config _CP_OFF&_WDT_OFF&_PWRTE_ON&_XT_OSC
 cnt        equ     0x0D      ;  счетчик  циклов
 ledset     equ     0x0E      ; регистр звука
; === Рабочая секция ===
 org    0x00
 goto   start
 org
 start
 clrf
 clrf
 bsf
 movlw
 movwf                                0x08
 movlw
 movwf
 bcf
 bcf
 clrf
 PORTA                                ;  очистить  порт  A
 PORTB                                ;  очистить  порт  B
```

```
STATUS,RP0
b'11111101'
TRISA
b'00000101'                          ; выбрать банк 1
TRISB
OPTION_REG,7
STATUS,RP0
ledset
; установить линию RA1 порта A на выход
; установить линии RP7:PR3 и RP1 порта B на выход
; включить подтягивающие резисторы
; выбрать банк 0
; очистить регистр звука
; --- Бесконечный цикл генерации звуковой частоты ---
loop
movlw
xorwf
movf
movwf
b'00010000'
ledset
ledset,W
PORTB
; инвертировать бит RB4 порта звука
; --- Программная задержка ---
;movlw
movlw                                0xA4
movwf                                0x0A                ; отладочное значение величины задержки
cycle                                cnt
decfsz
cnt
goto                                 cycle
; --- Конец программной задержки ---
goto
end                                  loop                ; бесконечный цикл
```

В этой программе мы убрали одну переменную цикла, и оставили только один цикл программной задержки для формирования импульсов (меандра). Кроме того, изменено значение константы, при помощи которой инвертируется бит в ledset, таким образом, чтобы инвертировался бит <4>, а не бит <1>. Значение ledset теперь выводится на линию RB4 порта B. Начальное значение переменной цикла cnt подобрано так, чтобы при тактовой частоте 4МГц генерировались импульсы с частотой 1000 Гц. Изменяя значение этой переменной, вы можете изменять частоту генерации звука.

Обработка нажатия кнопки

Теперь несколько усложним задачу. Построим программу так, чтобы в дежурном режиме мигал светодиод, а при нажатии на кнопку S2, подключенную к выводу RB2 порта B, программа генерировала звуковой сигнал. Для этого во время генерации импульсов низкой частоты, управляющей миганиями светодиода, будем периодически опрашивать состояние кнопки S2, и, если она нажата, вызывать подпрограмму генерации звука. Таким образом, мы получаем комбинацию двух ранее написанных программ, из которых одна является главной, а другая вызывается как подпрограмма.

Базовый код программы обработки нажатия кнопки

```
        ; === Секция заголовка ===
        list p=PIC16F84A
        include <P16F84A.INC>
        __config _CP_OFF&_WDT_OFF&_PWRTE_ON&_XT_OSC
 cnt1       equ     0x0C        ;       счетчик         циклов      1
 cnt2       equ     0x0D        ;       счетчик         циклов      2
 cnt3       equ     0x0E        ;       счетчик         циклов      3
 ledset     equ     0x0F        ; значение для вывода через порт
; === Рабочая секция ===
 org        0x00
 goto       start
org                                                                 0x08
start
 clrf                           PORTA
 bsf                            STATUS,RP0
 movlw                          b'11111011'
 movwf                          TRISB           ;       очистить        порт     A
 bcf                            STATUS,RP0      ; выбрать банк 1
 clrf                           PORTB
 ;movlw                         b'00000100'
 ;movwf                         PORTB
 ; установить линию RB2 порта B на
 выход
 ;       выбрать         банк        0
 ; очистить порт B
 ; установить бит RB2 порта B

 bsf                            STATUS,RP0
 movlw                          b'11111101'
 movwf                          TRISA           ; выбрать банк 1
 movlw                          b'00000101'
 ; установить линию RA1 порта A на
 выход

                                TRISB           ; установить линии RB2,RB0
 movwf                          OPTION_REG,7    порта B на вход
 bcf                            STATUS,RP0      ; включить подтягивающие
 bcf                            ledset          резисторы
 clrf                                           ;       выбрать         банк        0
                                                ; очистить порт светодиода
 ;       ---         Бесконечный         цикл
 включения/выключения светодиода ---
 loop                           b'00000010'     ;       включить/выключить
                                ledset          светодиод
                                ledset,W
```

```
                                PORTA
 movlw
 xorwf
 movf
 movwf
 ; --- Программная задержка ---
 ;movlw                         0xFF
 Movlw                          0x02
 movwf                          cnt1
 cycle1
 ;movlw                         0xF0
 Movlw                          0x03
 movwf                          cnt2
 cycle2
 btfss                          PORTB,2
 call                           beep        ; проверяем нажатие кнопки S2
 decfsz                         cnt2        ; если нажата, вызываем процедуру CALL
 goto                           cycle2      ; иначе продолжаем выполнение цикла
 decfsz                         cnt1
 goto                           cycle1
 ; --- Конец программной задержки ---
 goto                           loop        ; бесконечный цикл
 ; --- Подпрограмма beep ---
                                0xA0
 beep                           0x02        ; инициализируем переменную цикла
                                cnt3
 ;movlw
 movlw
 movwf
 loop_b
 movlw
 xorwf                          b'00010000'
 movf                           ledset
 movwf                          ledset,W    ; инвертируем уровень на выходе RB4
 decfsz                         PORTB
 goto                           cnt3
 return                         loop_b
 end
```

Теперь, если нажать кнопку S2, светодиод будет мигать очень редко, и из динамика раздастся звук с частотой примерно 1000 Гц. После отпускания кнопки светодиод начнет мигать с обычной частотой, а звук прекратится.

К сожалению, при отладке программы в MPLAB SIM у нас нет возможности нажать кнопку S2, поэтому нажатие кнопки приходится имитировать.

Приведенная выше программа настроена таким образом, будто кнопка S2 всегда нажата. Убедитесь в этом, создав в MPLAB проект, откомпилировав и выполнив команду **Debugger/Animate**.

Чтобы перенастроить программу таким образом, будто кнопка S2 всегда отпущена, раскомментируйте строки 25 и 26 программы (в листинге программы они выделены жирным шрифтом) и снова выполните команду **Debugger/Animate**.

Разумеется, данная программа написана не оптимально с точки зрения конечного результата, т.к. цикл управления светодиодом продолжает выполняться, несмотря на нажатие

кнопки, звук прерывается легкими щелчками и светодиод хоть и редко, но переключается. Эти проблемы легко устранить, но тогда программа не будет настолько простой и наглядной.

Внешние асинхронные воздействия (асинхронные стимулы)

Функции внешнего воздействия (стимулы) MPLAB предназначены для симуляции внешних событий и принудительного управления содержимым регистров, с их помощью можно имитировать появление на выводах МК сигналов высокого или низкого уровня и заносить значения непосредственно в регистры.

Существует четыре режима внешних событий:

• асинхронное воздействие - при помощи интерактивного диалога, управляющего внешними уровнями на входах контроллера;

• файл внешних воздействий на выводы - текстовый файл, описывающий внешние сигналы на входах контроллера;

• файл внешних воздействий на регистры - текстовый файл, содержащий набор 8-битных значений для записи непосредственно в регистры;

• частотное воздействие - программируемый источник регулярных внешних импульсов.

Асинхронное внешнее воздействие осуществляется при помощи специального диалогового окна с настраиваемыми кнопками. Откройте диалоговое окно *Stimulus*, выполнив команду меню **Debugger/Stimulus/New Workbook** и перейдите на вкладку *Asynch.* Пользователю предоставлено 13 кнопок, параметры которых можно настроить индивидуально.

Щелкните кнопкой мыши, например, на выделенном поле *Pin/SFR* первой строки и в появившемся списке доступных для внешнего воздействия выводов используемого МК выберите значение RB2.

Затем щелкните мышью на поле *Action* и в появившемся списке выберите, например, значение *Toggle* (Инвертировать). Вместе с *Toggle* доступны следующие формы внешнего асинхронного воздействия:

• *Set High* - устанавливает на указанном входе постоянный высокий уровень;

• *Set Low* - устанавливает на указанном входе постоянный низкий уровень;

• *Toggle* - изменяет уровень входного сигнала на противоположный и держит его;

• *Pulse High* - кратковременный импульс высокого уровня;

• *Pulse Low* - кратковременный импульс низкого уровня.

В поле *Comments/Message* первой строки добавьте поясняющий комментарий *Стимул входа RB2 порта B*. В результате этих действий будет создан первый асинхронный стимул.

Аналогичным образом можно создать и другие асинхронные стимулы для данного и других входов МК. Отметим, что одному и тому же входу можно поставить в соответствие несколько стимулов и смоделировать самые различные внешние воздействия на него.

Закончив создание стимулов, скомпилируйте и запустите программу на исполнение в режиме *Debugger/Animate*. Нажимая мышью кнопки *Fire* созданных асинхронных стимулов, в соответствующих окнах отладки можно наблюдать изменение уровней на входах портов и реакцию программы на эти изменения.

Чтобы сохранить рабочую книгу стимулов, выполните команду меню **Debugger/Stimulus/Save Workbook** и сохраните ее под именем вашего проекта.

Чтобы закрыть рабочую книгу стимулов, выполните команду меню **Debugger/Stimulus/Close Workbook**.

Чтобы воспользоваться стимулами вторично, выполните команду меню **Debugger/Stimulus/Open Workbook** и откройте файл сохраненной рабочей книги стимулов.

ЗАДАНИЯ ДЛЯ ВЫПОЛНЕНИЯ

1. Создайте в MPLAB проекты *Управление миганием светодиода*, *Управление генерацией звука*, *Обработка нажатия кнопки*, скомпилируйте их и отладьте с помощью инструмента MPLAB SIM.

2. В проекте *Обработка нажатия кнопки* создайте асинхронные стимулы, имитирующие нажатия кнопок S1 и S2, и в режиме **Debugger/Animate** убедитесь в их работоспособности.

3. Разработайте алгоритм и измените программу *Обработка нажатия кнопки* таким образом, чтобы при нажатии кнопки S2 генерировался звук, а мигания светодиода прекращались. И наоборот, при отпущенной кнопке S2 должны выполняться только мигания светодиода.

4. Еще раз измените программу *Обработка нажатия кнопки* таким образом, чтобы при нажатии кнопки S2 раздавался короткий сигнал *beep*, после чего программа возвращалась к нормальным миганиям светодиода независимо от состояния этой кнопки. При отпускании и повторном нажатии кнопки S2 все должно повторяться. Кратковременное нажатие кнопки S1 должно отключать генерацию сигнала *beep*, а повторное нажатие этой кнопки –снова включать ее.

ПРАКТИЧЕСКАЯ РАБОТА №27. МИКРОКОНТРОЛЛЕРЫ СЕМЕЙСТВА МК51.

Цель работы: изучить характеристики 8-разрядных микроконтроллеров семейства МК51.

Теоретические сведения

1. Характеристика отечественных 8-разрядных микроконтроллеров

Семейство однокристальных 8-разрядных микроконтроллеров (МК) серий 1816. 1830 включает ряд моделей, модификации которых варьируются в зависимости от объема и характера вычислительных ресурсов (памяти программ и данных, тактовой частоты).

Их недостатком является небольшой температурный диапазон эксплуатации (-10 - +70 С) , в то время как зарубежные фирмы Intel, Siemens производят модификации контроллеров, рассчитанные на применение в диапазонах: 0 - +70 С, -40 - +85 С, - 40 - +110 С и по особому закону -40 - +125С [5].

Семейство отечественных микроконтроллеров МК51 включает 6 модификаций серий КР1816, КР1830, КР1835, отличающихся по реализации резидентной памяти программ и мощности потреблении. В таблице 1 приводятся основные модели из указанного семейства микроконтроллеров.

Таблица 1 - Семейство МК51

Модель	Аналог	Объем резидентной памяти программ, Кбайт	Объем резидентной памяти данных, байт	Тактовая частота, МГц	Ток потребления, мА
КР1816ВЕ31	8031АН		128	12	150
КР18I6ВЕ51	8051АН	4К	128	12	150
КМ1816ВЕ751	875IАН	4К	128	12	220
КР1830ВЕ31	80С3IВН		128	12	18
КР1830ВЕ51	80С51ВН	4К	128	12	18
КР1835ВЕ51	80С51ВU	4К	128	12	18

Из таблицы 1 видно, что наиболее экономичными являются большие интегральные схемы (БИС) серии КР1830 при одинаковых значениях основных технических характеристик.

КМ1816ВЕ751, в отличии от КР1816ВЕ51, содержит внутреннее ППЗУ емкостью 4 Кбайт с ультрафиолетовым стиранием.

Следует отметить, что у всех моделей МК-51 за счет использования внешней памяти, емкость памяти программ и данных может быть расширена до 64 Кбайт.

Приведем некоторые особенности моделей КМ1830ВЕ751 и КМ1830ВЕ753 (аналог 8753Н фирмы AMD). Последняя отличается наличием перепрограммируемого запоминающего устройства (ППЗУ) с ультрафиолетовым стиранием на 8 Кбайт.

Микросхемы имеют особенности:

- специальный режим эксплуатации;
- дополнительные средства защиты памяти программ, расположенной на кристалле два бита защиты памяти и шифровальную таблицу;
- алгоритм программирования укороченными импульсами.

Общие сведения об однокристальных микроконтроллерах семейства МК51

Восьмиразрядные высокопроизводительные однокристальные микроконтроллеры семейства МК51 выполнены по высококачественной n-МОП технологии (серия 1816) и КМОП технологии (серия 1830).

Использование микроконтроллера семейства МК51 по сравнению с МК48 обеспечивает увеличение объема памяти команд и памяти данных. Новые возможности ввода-вывода и периферийных устройств расширяют диапазон применения и снижают общие затраты системы. В зависимости от условий использования быстродействие системы увеличивается минимум в два с половиной раза и максимум в десять раз.

МК КМ1816ВЕ751 содержит ППЗУ емкостью 4096 байт со стиранием ультрафиолетовым излучением и удобен на этапе разработки системы при отладке программ, а также при производстве небольшими партиями или при создании систем, требующих в процессе эксплуатации периодической подстройки. За счет использования внешних микросхем памяти общий объем памяти программ может быть расширен до 64 Кбайт.

МК КР1816ВЕ31 и КР1830ВЕ31 не содержат встроенной памяти программ, и могут использовать до 64 Кбайт внешней постоянной или перепрограммируемой памяти программ и эффективно использоваться в системах, требующих существенно большего по объему (чем 4 Кбайт на кристалле) ПЗУ памяти программ.

Каждый МК рассматриваемого семейства содержит встроенное ОЗУ памяти данных емкостью 128 байт с возможностью расширения общего объема оперативной памяти данных до 64 Кбайт за счет использования внешних микросхем ЗУПВ.

Общий объем памяти МК семейства МК51 может достигать 128 Кбайт: 64 Кбайт – память программ и 64 Кбайт – память данных.

При разработке на базе МК более сложных систем могут быть использованы стандартные ИС с байтовой организацией, например, серии КР580.

МК содержат все узлы, необходимые для автономной работы:

- центральный восьмиразрядный процессор;
- память программ объемом 4 Кбайт (только КМ1816ВЕ751, КР1816ВЕ51 и КР1830ВЕ51);
- память данных объемом 128 байт;
- четыре восьмиразрядных программируемых канала ввода-вывода (порты Р0, Р1, Р2, Р3);
- два 16-битовых многорежимных таймера/счетчика;
- система прерываний с пятью векторами и двумя уровнями;
- последовательный интерфейс;
- тактовый генератор.

Система прерываний, блок последовательного интерфейса и таймеры объединены в один блок.

Память программ предназначена для хранения программ и имеет отдельное от памяти данных адресное пространство объемом до 64 Кбайт, причем для микросхем КР1816ВЕ51, КМ1816ВЕ751 и для КР1830ВЕ51 часть памяти программ с адресами 0000Н -0FFFН расположена на кристалле МК. Память программ, расположенная на кристалле, состоит из 12-разрядного дешифратора и ПЗУ емкостью 4К*8 бит для микросхем КР1816ВЕ51, КР1830ВЕ51 или ППЗУ с ультрафиолетовым стиранием емкостью 4К*8 бит для КМ1816ВЕ751. Запись программ в ПЗУ происходит во время изготовления кристаллов.

Если на вывод МК EA/NPP подано напряжение питания Vcc то обращение к внешней памяти программ происходит автоматически при выработке счетчиком команд адреса, превышающего 0FFFH. Если адрес находится в пределах 0000H - 0FFFH, обращение происходит к памяти программ, расположенной на кристалле (внутренней памяти программ).

Если на вывод EA/NPP подан «0», внутренняя память программ отключается и, начиная с адреса 0000H, все обращения выполняются к внешней памяти программ.

Если МК не имеет внутренней памяти программ, ее вывод EA/NPP должен быть подключен к общей шине.

Чтение из внешней памяти программ стробируется сигналом PSEN. При работе с внутренней памятью программ сигнал PSEN не формируется. МК не имеют инструкций и аппаратных средств для программной записи в память программ.

При обращениях к внешней памяти программ всегда формируется 16-разрядный адрес, младший байт которого выдается через порт P0, а старший - через порт P2. При этом байт адреса, выдаваемый через порт P0, должен быть зафиксирован во внешнем регистре по спаду сигнала ALE, т. к. в дальнейшем линии порта P0 используются в качестве шины данных, по которой байт из внешней памяти программ вводится в МК.

2. Функциональная схема включения МК МК51 с внешним ППЗУ программ

Порт P0 работает как мультиплексированная шина адрес/данные: выдает младший байт счетчика команд, а затем переходит в высокоимпедансное состояние и ожидает прихода байта из ППЗУ программ. Когда младший байт адреса находится на выходах порта P0, сигнал ALE защелкивает его в адресном регистре RG. Старший байт адреса находится на выходах порта P2 в течение всего времени обращения к ППЗУ. Сигнал PME разрешает выборку байта из ППЗУ, после чего выбранный байт поступает на порт P0 МК51 и вводится в МК.

Память данных предназначена для приема, хранения и выдачи информации, используемой в процессе выполнения программы. Память данных, расположенная на кристалле МК, состоит из регистра адреса ОЗУ, дешифратора, ОЗУ и указателя стека.

Регистр адреса ОЗУ предназначен для приема и хранения адреса выбираемой с помощью дешифратора ячейки памяти, которая может содержать как бит, так и байт информации.

ОЗУ представляет собой 128 восьмиразрядных регистров, предназначенных для приема, хранения и выдачи различной информации.

Указатель стека представляет собой восьмиразрядный регистр, предназначенный для приема и хранения адреса ячейки стека, к которой было последнее обращение. При выполнении команд LCALL, ACALL содержимое указателя стека увеличивается на 2. При выполнении команд RET, RETI содержимое указателя стека уменьшается на 2. При выполнении команды PUSH direct содержимое указателя стека увеличивается на 1. При выполнении команды POP direct содержимое указателя стека уменьшается на 1. После сброса в указателе стека устанавливается адрес 07H, что соответствует началу стека с адресом 08H.

В МК предусмотрена возможность расширения памяти данных путем подключения внешних устройств емкостью до 64 Кбайт. При этом обращение к внешней памяти данных возможно только с помощью команд MOVX.

Команды MOVX ©Ri,A и MOVX A,@Ri формируют восьмиразрядный адрес, выдаваемый через порт P0. Команды MOVX @DPTR,A и MOVX @A,DPTR формируют 16-разрядный адрес, младший байт которого выдается через порт P0, а старший – через порт P2.

Байт адреса, выдаваемый через порт P0, должен быть зафиксирован во внешнем регистре по спаду сигнала ALE, т. к. в дальнейшем линии порта P0 используются как шина данных, через которую байт данных принимается из памяти при чтении или выдается в память данных при записи. При этом чтение стробируется сигналом RD, а запись – сигналом WR. При работе с внутренней памятью данных сигналы RD и WR не формируются.

Порты P0, P1, P2, P3 являются двунаправленными портами ввода-вывода и предназначены для обеспечения обмена информацией МК с внешними устройствами, образуя 32 линии ввода-вывода. Каждый из портов содержит фиксатор-защелку, который представляет

собой восьмиразрядный регистр, имеющий байтовую и битовую адресацию для установки (сброса) разрядов с помощью программного обеспечения.

Физические адреса P0, P1, P2, P3 составляют для:

P0 – 80H, при битовой адресации 80H -87H;

P1 – 90H, при битовой адресации 90H -97H;

P2 – A0H, при битовой адресации A0H – A7H;

P3 – B0H, при битовой адресации B0H – B7H.

Помимо работы в качестве обычных портов ввода/вывода линии портов P0 – P3 могут выполнять ряд дополнительных функций, описанных ниже.

1.3 Структура микроконтроллера КР1816ВЕ51 (МК-51)

Условное графическое обозначение микроконтроллера МК-51 показано на рисунке 1.

Назначение выводов:

Uss - потенциал общего провода ("земли");

Ucc - основное напряжение литания +5 В;

X1, X2 - выводы для подключения кварцевого резонатора;

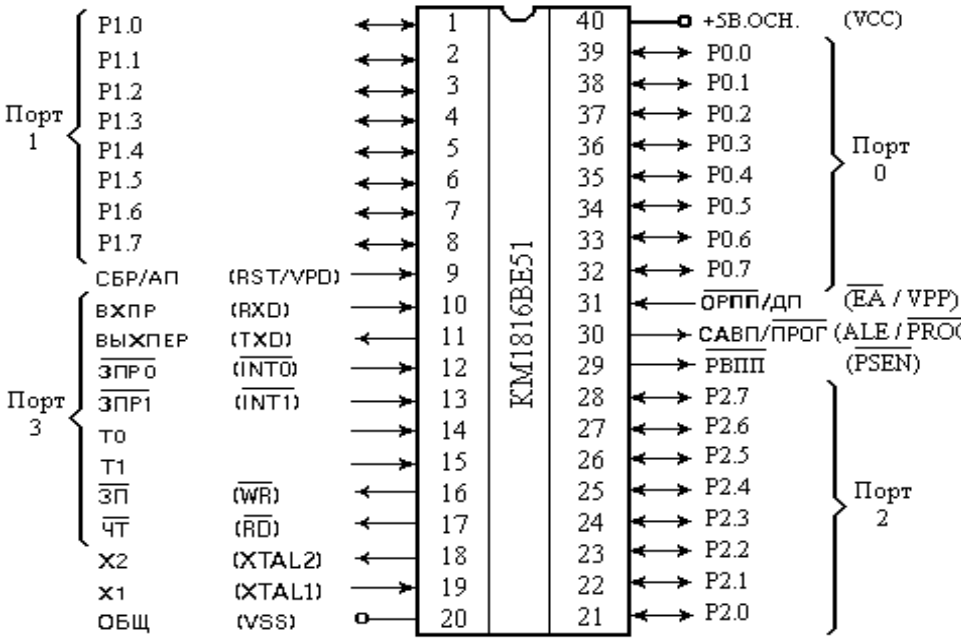


Рисунок 1 – Условное графическое обозначение микроконтроллера МК-51

RST/VPD - вход общего сброса/подключения резервного источника питания;

PSEN - разрешение внешней памяти программ; выдается только при обращении к внешнему ПЗУ;

ALE - строб адреса внешней памяти;

EA - отключение внутренней программной памяти; уровень 0 на этом входе заставляет микроконтроллер выполнять программу только внешней ПЗУ; игнорируя внутреннюю (если последняя имеется);

P1 - восьмибитный квазидвунаправленный порт ввода/вывода: каждый разряд порта может быть запрограммирован как на ввод, так и на вывод информации, независимо от состояния других разрядов;

P2 - восьмибитный квазидвунаправленный порт, аналогичный P1; кроме того, выводы этого порта используются для выдачи адресной информации (старшего байта) при обращении к внешней памяти программ или данных (если используется 16-битовая адресация последней). Выводы порта используются так же для подключения и программирования расширителя (К580ВР43);

P3 - восьмибитный квазидвунаправленный порт, аналогичный. P1; кроме того, выводы этого порта могут выполнять ряд альтернативных функций, которые используются при работе

таймеров, порта последовательного ввода-вывода, контроллера прерываний, и внешней памяти программ и данных;

Альтернативные функции порта P3:

RxD – вход приемника последовательного порта в режиме УАПП. Ввод/вывод данных в режиме сдвигающего регистра;

TxD – выход передатчика последовательного порта в режиме УАПП. Выход синхронизации в режиме сдвигающего регистра;

INT0 (инв), INT1 (инв) – входы запросов на прерывание (срабатывает по низкому уровню или срезу);

T0, T1 – входы таймеров-счетчиков или тест-входы;

RD (инв), WR (инв) – сигналы управления чтения и записи (формируются при обращении к внешней памяти);

P0 - восьмибитный двунаправленный порт ввода-вывода информации: при работе с внешними ОЗУ и ПЗУ по линиям порта в режиме временного мультиплексирования выдается адрес внешней памяти, после чего осуществляется передача или прием данных.

Структурная организация и система команд МК - 51 хорошо описана в [1], и приведена на рисунке 2. Отметим, что микроконтроллер можно представить в виде следующих основных узлов:

- арифметико-логическое устройство (АЛУ);
- память данных и программ;
- аккумулятор;
- регистр слова состояния (регистр признаков);
- 8 - битный регистр - указатель стека;
- 16 - битный регистр - указатель данных;
- блок таймеров - счетчиков;
- последовательный порт;
- регистры специальных функций;
- устройства управления и синхронизации;
- порты ввода - вывода.

МК-51 работает от внутреннего генератора, который имеет внешние выводы для подключения кварцевого резонатора.

Все порты (0,1,2,3) МК-51 служат для побайтного ввода - вывода данных. Кроме того, порты 0 и 2 служат для вывода адреса. Выводы порта 3 могут быть использованы для реализации альтернативных функций. Порт 0 является двунаправленным, порты 1, 2, 3 - квазидвунаправленными: каждая линия порта может быть настроена для ввода или вывода данных. Нагрузочная способность портов 1, 2, 3 - один ТТЛ - вход, а порта 0 - два ТТЛ входа. PC - 16-разрядный программный счетчик.

Аккумулятор (A) служит для хранения как операнда, так и результата операции. Но МК-51 может выполнять ряд команд и без участия аккумулятора. Выполнение многих команд производится в арифметическо-логическом устройстве, а ряд признаков операций фиксируется в регистре слова состояния программы (PSW).

Арифметическо-логическое устройство (8-битное) может выполнять арифметические операции сложения, вычитания, умножения и деления; логические операции И, ИЛИ, исключающее ИЛИ, а также операции циклического сдвига, сброса, инвертирования и т.п. В АЛУ имеются программно недоступные регистры T1 и T2, предназначенные для временного хранения операндов, схема десятичной коррекции и схема формирования признаков.

Восьмиразрядный указатель стека (SP) может адресовать любую область памяти данных. Содержимое его уменьшается на единицу в ходе выполнения команд PUSH и CALL и увеличивается на единицу при выполнении команд POP и RET. При сбросе в SP автоматически загружается начальный код 07 H.

16-разрядный регистр-указатель данных (DPTR) служит для фиксации адреса при обращении к внешней памяти. Он может работать как два независимых 8-разрядных регистра DPH и DPL.

МК-51 имеет два независимых программируемых 16-разрядных таймера/счетчика, представленных в виде двух регистровых пар: TH0, TL0 и TH2, TL1, буфер последовательного порта SBUF - два независимых 8 - разрядных регистра: регистр приемника и регистр передатчика, регистры специальных функций с именами IP, IE, TMOD, TCON, SCON, PCON, позволяющие программировать схему прерывания, режимы работы таймеров-счетчиков и приемопередатчика.

Память программ и память данных, размещенные на кристалле МК51, физически и логически разделены, имеют различные механизмы адресации, работают под управлением различных сигналов и выполняют разные функции.

Память программ (ПЗУ или СППЗУ) имеет емкость 4 Кбайта и предназначена для хранения команд, констант, управляющих слов инициализации, таблиц перекодировки входных и выходных сменных и т.п. РПП имеет 16-битную шину адреса, через которую обеспечивается доступ из счетчика команд или из регистра-указателя данных. Последний выполняет функции базового регистра при косвенных переходах по программе или используется в командах, оперирующих с таблицами.

Память данных (ОЗУ) предназначена для хранения переменных в процессе выполнения прикладной программы, адресуется одним байтом и имеет емкость 128 байт. Кроме того, к адресному пространству РПД примыкают адреса регистров специальных функций (РСФ), которые перечислены в таблице 2.

Таблица 2 - Блок регистров специальных функций

Символ	Наименование	Адрес
* ACC	Аккумулятор	0E0H
* B	Регистр-расширитель аккумулятора	0F0H
* PSW	Слово состояния программы	0D0H
SP	Регистр-указатель стека	81H
DPTR	Регистр-указатель данных (DPH)	83H
	(DPL)	82H
* P0	Порт 0	80H
* P1	Порт 1	90H
* P2	Порт 2	0A0H
* P3	Порт 3	0B0H
* IP	Регистр приоритетов	0B8H
* IE	Регистр маски прерываний	0A8H
TMOD	Регистр режима таймера/счетчика	89H
* TCON	Регистр управления/статус таймера	88H
TH0	Таймер 0 (старший байт)	8CH
TL0	Таймер 0 (младший байт)	8AH
TH2	Таймер 1 (старший байт)	8DH
TL1	Таймер 1 (младший байт)	8BH
* SCON	Регистр управления приемопередатчиком	98H
SBUF	Буфер приемопередатчика	99H
PCON	Регистр управления мощностью	87H
Примечание. Регистры, имена которых отмечены знаком (*), допускают адресацию отдельных бит.		

В таблице 3 приводится перечень флагов ССП, даются их символические имена и описываются условия их формирования.

Таблица 3 - Формат слова состояния программы (ССП)

Символ	Позиция	Имя и назначение
C	PSW.7	Флаг переноса. Устанавливается и сбрасывается при выполнении арифметических и логических операций
AC	PSW.6	Флаг вспомогательного переноса. Устанавливается и сбрасывается при выполнении команд сложения и вычитания и сигнализирует о переносе или заем в бите 3
F0	PSW.5	Флаг 0. Может быть установлен, сброшен или проверен программой как флаг, специфицируемый пользователем
RS1 RS0	PSW.4 PSW.3	Выбор банка регистров. Устанавливается и сбрасывается программой для выбора рабочего банка регистров
OV	PSW.2	Флаг переполнения. Устанавливается и сбрасывается при выполнении арифметических операций
-	PSW.1	Не используется
P	PSW.0	Флаг паритета. Устанавливается и сбрасывается каждом цикле команды и фиксирует нечетное/четное число единичных бит в аккумуляторе

Примечание	RS1	RS0	Банк	Границы адресов
	0	0	0	00H-07H
	0	1	1	08H-0FH
	1	0	2	10H-17H
	1	1	3	18H-1FH

ПРАКТИЧЕСКАЯ РАБОТА №28. СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРА СЕМЕЙСТВА МК51.

Цель работы: изучить систему команд 8-разрядного микроконтроллера семейства МК51, а также программно-логическую модель МК-51 и работу с ней.

Теоретические сведения

Микроконтроллер имеет 111 базовых команд передачи данных, арифметических и побитных операций, передачи управления и операций с битами. Большинство команд выполняются за 1 или 2 машинных цикла при fтак=12Мгц и длительности машинного цикла 1 мкс. Первый байт команды всегда содержит код операции (КОП), второй и третий байты - либо адреса операндов, либо непосредственные операнды.

Перечень команд МК-51 приводится в таблице 4.

Таблица 4 - Система команд

Мнемокод	КОП	Мнемокод	КОП	Мнемокод	КОП
ACALL 0xxH	11	AJMP 5XXH	A1	DA A	D4
ACALL 1xxH	31	AJMP 6XXH	C1	DEC A	14
ACALL 2xxH	51	AJMP 7XXH	El	DEC ad	15
ACALL 3xxH	71	ANL A , ad	55	DEC R0	18
ACALL 4xxH	91	ANL A, R0	58	DEC R1	19
ACALL 5xxH	B1	ANL A, R1	59	DEC R2	1A
ACAlL 6xxH	D1	ANL A, R2	5A	DEC R3	1B
ACALL 7xxH	FI	ANL A, R3	SB	DEC R4	1C
ADD A, ad	25	ANL A, R4	5C	DEC R5	1D
ADD A, R0	28	ANL A, R5	5D	DEC R6	1E
ADD A , R1	29	ANL A, R6	5E	DEC R7	1F
ADD A, R2	2A	ANL A, R7	5F	DEC @R0	16
ADD A, R3	2B	ANL A, @R0	56	DEC @R1	17
ADD A, R4	2C	ANL A, @R1	57	DIV AB	84
ADD A, R5	2D	ANL A, #d	54	DJNZ ad, rel	D5
ADD A, R6	2E	ANL ad, A	52	DJNZ R0, rel	D8
ADDA, R7	2F	ANL ad, #d	S3	DJNZ R1, rel	D9
ADD A, @R0	26	ANL C, bit	82	DJNZ R2, rel	DA
ADD A, @R1	27	ANL C, /bit	BO	DJNZ R3, rel	DB
ADD A, #d	34	CJNE A, ad, rel	B5	DJNZ R4, rel	DC
ADDC A, ad	35	CJNE A, #d, rel	B4	DJNZ R5, rel	DD
ADDC A, R0	38	CJNE R0, #d, rel	B8	DJNZ R6, rel	DE
ADDC A, R0	39	CJNE R1, #d, rel	B9	DJNZ R7, rel	DF
ADDC A, RO	3A	CJNE R2, #d, rel	BA	INC a	04
ADDC A , R0	3B	CJNE R3, #d, rel	BB	INC ad	05
ADDC A, R0	3C	CJNE R4, #d, rel	BC	INC DPTR	A3
ADDC A, R0	3D	CJNE R5, #d, rel	BD	INC R0	08
ADDC A, R0	3E	CJNE R6, #d , rel	BE	INC R1	09
ADDC A , R0	3F	CJNE R7 , #d, ret	BF	INC R2	OA
ADDC A, @R0	36	CJNE @R0, #d, rel	B6	INC R3	OB
ADDC A, @R1	37	CJNE @R1, #d, rel	B7	INCR4	0C
ADDC A, #d	24	CLR A	E4	INC R5	0D
AJMP 0XXH	01	CLR bit	C2	INC R6	0E
AJMP 1XXH	21	CLR C	C3	INC R7	OF
AJMP 2XXH	41	CPL A	F4	INC @R0	06
AJMP 3XXH	61	CPL bit	B2	INC @R1	07
AJMP 4XXH	81	CPL C	B3	JB bit, rel	20
				JBC bit, rel	10

Продолжение таблицы 2

Мнемокод	КОП	Мнемокод	КОП	Мнемокод	КОП

JC rel	40	MOV ad , @R0	86	MOV R7 , ad	AF
JMP@A+DPTR	73	MOV ad, @R1	87	MOV R7, #d	7F
JNB bit , rel	30	MOV ad, #d	75	MOV @R0 , A	F6
JNC rel	50	MOV ad, ads	85	MOV@R0, ad	A6
JNZrel	70	MOV bit, C	92	MOV@R0, #d	76
JZ rel	60	MOV C, bit	A2	MOV @R1 , A	F7
LCALL ad16	12	MOV DPTR, #dl6	90	MOV@R1, ad	A7
LJMP ad 16	02	MOV R0, A	F8	MOV @R1, #d	77
MOV A , ad	E5	MOV R0, ad	A8	MOVC A, @+DPTR	93
MOV A , RO	E8	MOV R0, #d	78	MOVC A , @+PC	83
MOV A, R1	E9	MOV R1 , A	F9	MOVX A , @DPTR	EO
MOV A , R2	EA	MOV R1 , ad	A9	MOVX A, @R0	E2
MOV A , R3	EB	MOV Rl , #d	79	MOVX A, @R1	E3
MOV A , R4	EC	MOV R2, A	FA	MOVX @DPTR, A	F0
MOV A , R5	ED	MOV R2, ad	AA	MOVX @R0 , A	F2
MOV A , R6	EE	MOV R2, #d	7A	MOVX @R1, A	F3
MOV A , R7	EF	MOV R3 , A	FB	MUL AB	A4
MOV A , @R0	E6	MOV R3 , ad	AB	NOP	00
MOV A, @R1	E7	MOV R3 , #d	7B	ORL A , ad	45
MOV a , #d	74	MOV R4, A	FC	ORL A , R0	48
MOV ad , A	F5	MOV R4 , ad	AC	ORL A, R1	49
MOV ad , R0	88	MOV R4, #d	7C	ORL A, R2	4A
MOV ad , R1	89	MOV R5, A	FD	ORL A , R3	4B
MOV ad , R2	8A	MOV R5 , ad	AD	ORL A, R4	4C
MOV ad, R3	8B	MOV R5 , #d	7D	ORL A, R5	4D
MOV ad , R4	8C	MOV R6 , A	FE	ORL A, R6	4E
MOV ad , R5	8D	MOV R6, ad	AE	ORL A, R7	4F
MOV ad , R6	8E	MOV R6, #d	7E	ORL A, @R0	46
MOV ad , R7	8F	MOV R7 , A	FF	ORL A , @R0	47
ORL A, #d	44	RRC A	13	SUBB A , R7	9F
ORL ad , A	42	SETB bit	D2	SUBB A , @R0	96
ORL ad , #d	43	setb c	D3	SUBB A, @R1	97
ORL C , bit	72	SJMP rel	80	SWAP A	C4
ORL C, /bit	AO	SUBB A, ad	95	XCH A , ad	C5
POP ad	DO	SUBB A, R0	98	XCH A, R0	C8
PUSH ad	CO	SUBB A, R1	99	XCH A, R1	C9
RET	22	SUBB A , R2	9A	XCH A , R2	CA
RETI	32	SUBB A , R3	9B	XCH A , R3	CB

RL A	23	SUBB A, R4	9C	XCH A , R4	CC
RLC A	33	SUliB A , R5	9D	XCH A , R5	CD
RR A	03	SUBB A, R6	9E	XCH A , R6	CE
XCH A, R7	CF	XRL A, R1	69	XRL A , R7	6F
XCH A, @R0	06	XRL A,	6A	XRL A , @)R0	66
XCH A, @Rl	C7	XRL A, R3	6B	XRL A, @R1	67
XCHD A, @R0	D6	XRL A, R4	6C	XRL A, #d	64
XCHD A, @R1	D7	XRL A, R5	6D	XRL ad , A	62
XRL A, ad	65	XRL A , R6	6E	XRL ad, #d	63
XRL A, R0	68				

Программно-логическая модель МК-51 и работа с ней

Программно-логическая модель микроконтроллера K18I6BE51 реализуется с использованием PC. Программа SCM (Single-Chip Machine) представляет собой систему моделирования однокристальных микроконтроллеров.

Система моделирования Single-Chip Machine 1.22 предназначена для исследования поведения внутренних и внешних сигналов указанных микросхем.

Программа SCM (Single-Chip Machine) выполнена в виде независимого запускаемого модуля, работоспособного под управлением операционной системы MS Window NT/XP. SCM включает средства отладки и редактирования программ на ассемблере. Выполнение программы пользователя осуществляется с максимальным приближением к действительности с помощью имитационной модели. Кроме того, пользователю предоставляется такие средства, как временные диаграммы внутренних и внешних сигналов, имитация внешних сигналов, возможность изменения значений узлов МК в процессе работы модели и др.

Пользователь набирает программу в редакторе программ, затем нажимает кнопку “компиляция”. Текст программы переводится в машинные коды и записывается в одноименный файл (с исходным текстом) с расширением “.MPM”. Расширение “.MPM”, расшифровывается как Microcontroller Program Memory, однако существует стандартный формат представления памяти программ - так называемый формат HEX.

Программа обеспечивает: выполнение прикладной программы для ОЭВМ в пошаговом режиме, в режиме прогона с остановом по контрольным точкам; доступ ко всем внутренним ресурсам ОЭВМ, внешней памяти программ и данных.

Практическая часть.

1. Изучение программы эмулятора SCM.
2. Запуск программы. Знакомство и изучение основного меню программы, ознакомление с возможностями и способами редактирования внутренних и внешних ресурсов эмулятора.
3. Изучение процесса программирования микроконтроллера, ввода и отладки программ, и также их выполнения.
4. Запуск программы - эмулятора и практическое закрепление полученных знаний.

Контрольные вопросы

1. Архитектура микроконтроллера. Основные узлы и блоки.
2. Максимальный объем ОЗУ и ПЗУ, используемый данным контроллером.
3. Способы редактирования с помощью программы - эмулятора ОЗУ, ПЗУ, системных ресурсов.
4. Команды выполнения программы. Результаты выполнения.
5. Возможности отладки МПУ.
6. Между какими частями микроконтроллера осуществляется передача данных.
7. Методы адресации, используемые в ОЭВМ.

8. Типы портов микроконтроллера.
9. Как происходит адресация внешнего ОЗУ и ПЗУ?

ПРАКТИЧЕСКАЯ РАБОТА №29. ПРИЕМЫ РАБОТЫ С ВИДЕОПАМЯТЬЮ

Цель работы: изучение организации и основных приемов работы с видеопамятью ПК в текстовом режиме.

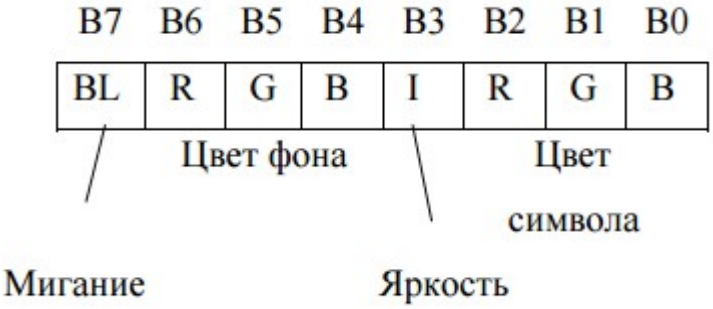
Теоретические сведения

В видеосистемах ПК для отображения информации на экране монитора используются два режима: текстовый и графический. В текстовом режиме на экран выводятся только символы, а в графическом можно строить любые сложные изображения в виде рисунков, чертежей и т. п. В текстовом режиме используется матричный метод формирования изображения. Суть метода заключается в том, что изображение на экране монитора строится из отдельных фрагментов выводимых символов (букв, цифр, математических знаков, графических элементов и др.). Каждый фрагмент представляется в виде матрицы, имеющей вид прямоугольника стандартных размеров и состоящей из определенного числа битов. Например, на рис. 2.1 показана матрица 8x8 для формирования на экране буквы “Н”. Матрицы символов хранятся в знакогенераторе (памяти типа ПЗУ) видеоадаптера. Каждый бит (0 или 1) подается в схему управления лучом электронно-лучевой трубки монитора, который формирует на экране точечное изображение. Единичное значение бита указывает на то, что соответствующая точка относится к изображаемому символу. Нулевое значение бита определяет точку фона символа на экране. В текстовом режиме экран дисплея делится на отдельные знакоместа, в каждое из которых может быть помещен символ. Экран разбивается на 25 строк по 80 знакомест в каждой строке, чем обеспечивается вывод одновременно до 2000 символов.

0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	**1**	0
0	**1**	0	0	0	0	**1**	0
0	**1**	**1**	**1**	**1**	**1**	**1**	0
0	**1**	0	0	0	0	**1**	0
0	**1**	0	0	0	0	**1**	0
0	**1**	0	0	0	0	**1**	0
0	0	0	0	0	0	0	0

В процессе работы по заданной программе коды символьных изображений, выдаваемых на экран монитора, записываются в специальную область оперативной памяти, которая называется видеопамятью. При работе в текстовом режиме для каждого символа в памяти следует хранить его код и указание, как его изображать (цвет изображаемого символа, цвет фона, яркость изображения). Поэтому каждому символу соответствуют два байта: в первом из них записывается, что следует изобразить (код символа), во втором - как это изобразить (код атрибута символа). Для каждого символа в цветном режиме работы видеоадаптера доступно 16 основных цветов и 8 цветов фона, а также возможность мерцания символа. Формат байта,

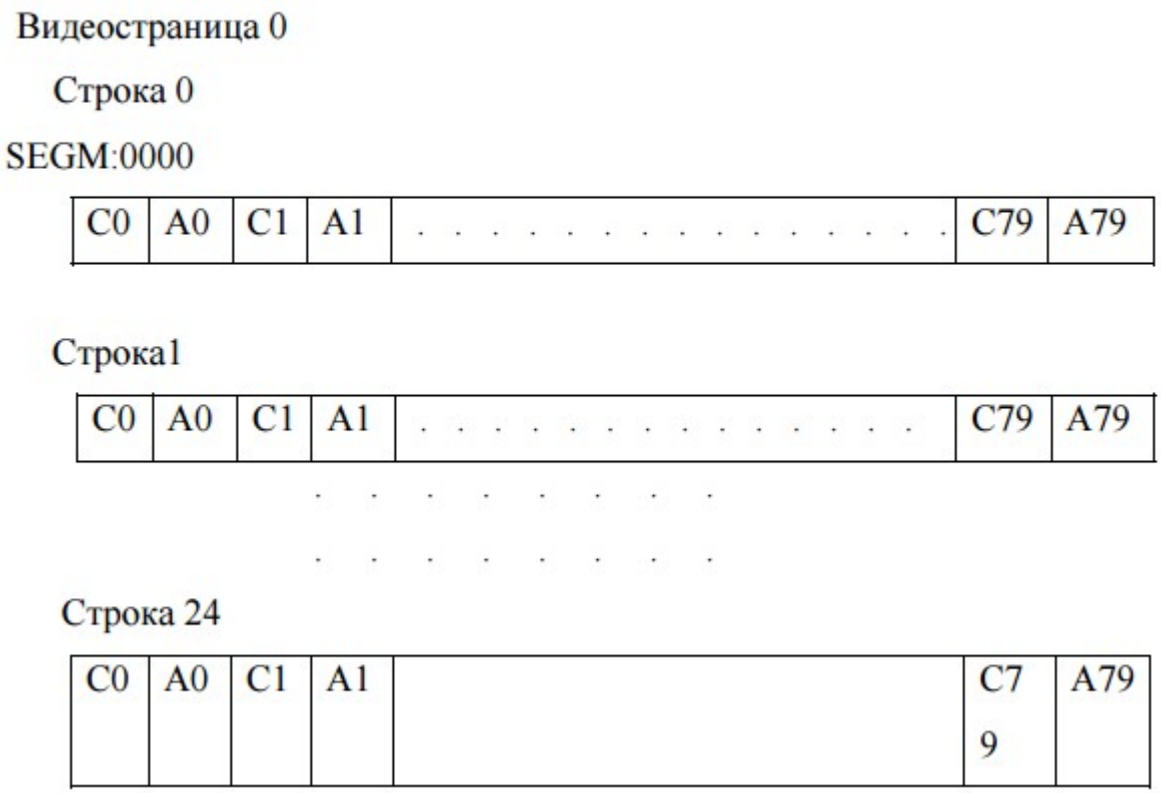
содержащего атрибут изображаемого символа, представлен на рис. 2.2. Отдельные биты атрибута образуют 4 группы: биты B0-B2 - цвет символа; бит B3 - яркость; биты B4-B6 - цвет фона; бит B7 - мерцание. Установка бита яркости в "1" делает цвет символа более светлым. С помощью битов B3-B0 можно получить 16 цветов символа. Если бит мерцания B7 установлен в "1", символ начинает мерцать с частотой приблизительно 4 раза в секунду.



Организация видеопамяти

В текстовом режиме в видеопамяти может храниться до 4 видеостраниц или экранов с номерами от 0 до 3. Переключение видеостраниц (т.е. вывод их содержимого на экран) осуществляется средствами BIOS. Большинство программ, в то числе и DOS, работают с 0 видеостраницей. В каждой видеостранице помещается равно один экран в 25 строк по 80 символов в каждой, причем каждый символ отображается двумя байтами - байтом с кодом символа и байтом с атрибутом символа. Порядок размещения этих кодов показан на рис. 2.3. Обозначим: адрес первой ячейки - SEGM:0000H, где SEGM-двухбайтовое значение сегмента видеопамяти; C0 и A0 -соответственно код символа в строке 0 и атрибут символа в строке 0 и т.д. Для формирования текстового изображения на экране из видеопамяти последовательно считывается информация, начиная с адреса SEGM:0000H. Считывание осуществляется непрерывно и циклически. Этот процесс называется сканированием памяти. По коду символа из знакогенератора выбирается соответствующая матрица для формирования изображения символа на экране. Первый символ (символ с номером C0 в строке 0) появляется в левом верхнем углу экрана с атрибутом A0. Следующие 2 байта видеопамяти изображают следующий символ справа от14 него и т. д. Значение смещения (SM) ячейки видеопамяти для любого символа, размещаемого в произвольной позиции (NS-строка, NK-колонка) на экране определяется по формуле:

$$SM = 2 * (NS * 80 + NK) .$$



Для видеосистем ПК разработан ряд стандартных видеорежимов. При работе в оболочке Far manadger видеосистема по умолчанию устанавливается в текстовый цветной режим № 3 с количеством цветов равным 16 и количеством знакомест 80 х 25. Это режим работы видеоадаптера CGA, для которого видеопамять имеет емкость 16 Кбайт и начинается с адреса B800:0000. Обычно в программах, работающих напрямую с видеопамятью, регистр DS либо ES микропроцессора загружается значением сегмента видеопамяти, а остальная часть программы работает со смещениями в видеопамяти.

Практическая часть

В этой практической работе необходимо написать программу, которая бы прямым доступом в видеопамять реализовала бы эффект "бегущей буквы" по экрану, например, движение буквы по верхней строке экрана. Алгоритм этой программы может быть, к примеру, следующий:

1. Инициализация ES, DI, CX.

2. Заполнение 0 страницы видеопамяти одинаковыми символами (например, кодом буквы "A" - 128 (80h) и одинаковыми атрибутами, причем такими, что бы цвет символа в них совпадал бы с цветом фона, т.е. символ был невидимым.

3. У символа на строке 0 и колонке 0 меняем атрибут на любой видимый, делаем задержку, после нее изменяем атрибут у следующего символа в строке и возвращаем старое значение атрибута предыдущему символу, т.е. гасим его. Эти действия продолжаем до 79 колонки 0-й строки.

4. Выход из программы. Таким образом, просто "зажигая" и "гася" записанные в видеопамять байты символов, мы получаем эффект бегущей по экрану буквы.

Варианты заданий на практическую работу.

Вариант 1. Движение символа по периметру экрана, начиная с позиции (0,0) слева направо.

Вариант 2. Движение символа по периметру экрана, начиная с позиции (0,0) сверху вниз.

Вариант 3. Движение символа по периметру экрана, начиная с позиции (24,79) справо налево.

Вариант 4. Движение символа по периметру экрана, начиная с позиции (24,0) слева направо.16

Вариант 5. Движение символа по периметру прямоугольника с позициями вершин (0,40), (24,40), (24,79), (0,79).

Вариант 6. Движение символа по периметру прямоугольника с позициями вершин (10,0), (24,0), (24,50), (10,50).

Вариант 7. Движение символа по периметру прямоугольника с позициями вершин (0,30), (15,30), (15,79), (0,79).

Вариант 8. Движение символа по периметру прямоугольника с позициями вершин (0,0), (0,40), (20,40), (20,0). Во время отладки программы необходимо поэкспериментировать с цветом символа и цветом фона.

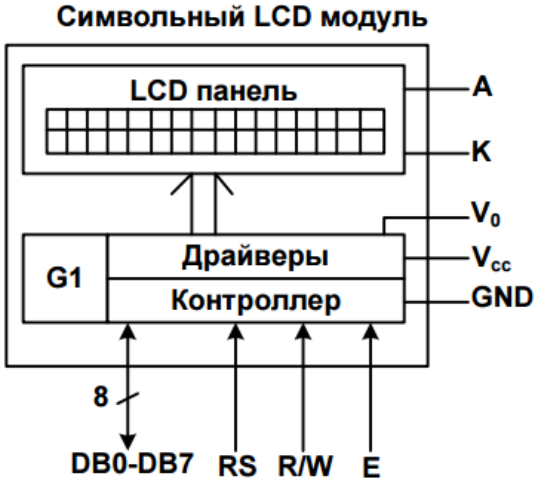
ПРАКТИЧЕСКАЯ РАБОТА №30. ВЗАИМОДЕЙСТВИЕ С ВНЕШНИМИ УСТРОЙСТВАМИ

Цель работы: научиться использовать ЖКИ совместно с микроконтроллером.

Краткие теоретические сведения

Жидкокристаллические индикаторы (ЖКИ) являются одними из основных средств вывода информации для современных цифровых систем. Представляют собой недорогое и удобное решение, позволяющее сэкономить время и ресурсы при разработке новых устройств. Обеспечивают отображение большого объема информации при хорошей различимости и низком энергопотреблении, благодаря чему широко используются в измерительных приборах, медицинском оборудовании, промышленном оборудовании, информационных системах, аппаратуре с автономным питанием.

Основу составляет специализированный контроллер, обычно выполненный в виде одной или двух «микросхем-капелек», реже − в виде фирменной SMD-микросхемы. Контроллер синхронизируется внутренним RC- генератором G1, имеющим частоту 250 ±50 кГц, и управляет интерфейсом, а драйвер "зажигает" сегменты. Напряжение подсветки подается через вы- воды A и K на светодиоды, которые освещают ЖК-панель с торца или об- ратной стороны корпуса. Светодиоды включены матрицей и соединены параллельно-последовательно. В связи с этим напряжение подсветки довольно высокое 4,0...4,2 В.



Электрический интерфейс ЖКИ состоит из 3-х шин: шины данных (DB0…DB7), шины управления (RS, R/W, E) и шины питания (Vss, Vdd, Vee). Выполняемые функции каждого вывода ЖКИ приведено в таблице 11.

Таблица 11 – Функции, выполняемые каждым выводом ЖКИ.

№ вывода	Обозначение	Выполняемая функция
1	Vee	Общий провод
2	Vcc	Питание +5В
3	Vdd	Напряжение фокусировки
4	RS	«0»- запись команд «1»-запись данных
5	R/W	«0» - запись в ЖКИ «1» - чтение из ЖКИ
6	E	Перепад уровня из «1» в «0» - ввод данных
7-14	DB0-DB7	Двунаправленная шина данных

В практической работе используется 2-строчный 8-символьный ЖКИ WH0802A-NGA-CT, который подключается к микроконтроллеру ATtiny2313 согласно схеме, представленной на рисунке 5. Для передачи данных используется 4-битная шина DB4-DB7, то есть байт данных будет передаваться в 2 этапа: сначала старший полубайт, затем − младший.

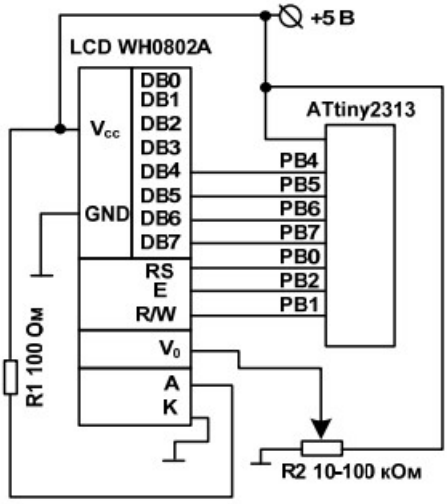


Рисунок 5 – Схема подключения микроконтроллера ATtiny2313 к ЖКИ

Микроконтроллер ATtiny2313 управляет работой WH0802A через си- стему команд. Все команды можно посмотреть в Datasheet ЖКИ. Основ- ные приведены в таблице 12.

Таблица 12 – Основные команды WH0802A

Команда ЖКИ	Hex-код
Очистка дисплея	0x01
Возврат курсора в начало	0x02
Сдвиг курсора влево	0x04
Сдвиг курсора вправо	0x06
Выключение дисплея	0x08
Выключение курсора	0x0C
Интерфейс 4 бита, 2 строки	0x28
Установка позиции курсора	0x80 – 0xC7

Для отправки команды ЖКИ необходимо, чтобы на выводах RS и R/W было напряжение низкого уровня. Для отправки данных требуется, чтобы на выводе RS было напряжение высокого уровня, а на выводе R/W – низкого. В обоих случаях разрешение записи осуществляется изменением состояния вывода E с высокого напряжения на низкое. Функция отправки команды в ЖКИ на языке СИ может выглядеть так:

```
void LCDsendCommand(uint8_t cmd)
 {
  LDP=(cmd&0xF0); //загружаем в порт данных старший полубайт команды
  LCP|=1<<LCD_E; //отправляем старший полубайт команды (E=1)
  _delay_ms(1);
  LCP&=~(1<<LCD_E); //E=0
  _delay_ms(1);
  LDP=((cmd&0x0F)<<4); //загружаем в порт данных младший полубайт команды
  LCP|=1<<LCD_E; //отправляем старший полубайт команды (E=1)
  _delay_ms(1);
  LCP&=~(1<<LCD_E); //E=0
  _delay_ms(1);
 }
```

Аргументом функции является 8-битная команда cmd, отправка которой происходит по полубайтам. Обязательным условием является передача данных контроллеру ЖКИ с паузами не менее 40 мкс, чтобы контроллер ЖКИ успел выгрузить полубайт и загрузить новый, так как частота работы ATtiny2313 в несколько раз превышает частоту работы контроллера ЖКИ; для этой цели в примере программы используется функция задержки _delay_ms().

Функция отправки символа в ЖКИ на языке СИ может выглядеть так:

```
void LCDsendChar(uint8_t ch)
  {
   LDP=(ch&0xF0); //загружаем в порт данных старший полубайт символа
   LCP|=1<<LCD_RS;  //настраиваем порт на отправку данных (RS=1)
   LCP|=1<<LCD_E; //отправляем старший полубайт символа(E=1)
   _delay_ms(1);
   LCP&=~(1<<LCD_E); //E=0
   LCP&=~(1<<LCD_RS); //RS=0
   _delay_ms(1);
   LDP=((ch&0x0F)<<4); //загружаем в порт данных старший полубайт символа
   LCP|=1<<LCD_RS; //настраиваем порт на отправку данных (RS=1)
   LCP|=1<<LCD_E; //отправляем старший полубайт символа(E=1)
_delay_ms(1);
   LCP&=~(1<<LCD_E); //E=0
   LCP&=~(1<<LCD_RS); // RS=1
   _delay_ms(1);
}
```

Как и в предыдущем случае передача байта символа осуществляется по полубайтам.

Используя ЖКИ, можно выводить результат вычислений над значениями в регистрах на дисплей. Однако следует помнить, что ЖКИ кодирует символы согласно таблице «ASCII» кода и символу 0 , например, соответствует десятичный код 48, а символу 9 – код 57.

Контрольные вопросы

1. Для чего используют жидкокристаллические индикаторы?
2. Какова структура ЖКИ?
3. Какие выводы ЖКИ используются для ввода данных от микро- контроллера?
4. Для чего требуется производить инициализацию ЖКИ перед выводом информации?
5. Какую кодировку использует ЖКИ при выводе символов на экран?

ПРАКТИЧЕСКАЯ РАБОТА №31. ПРОГРАММИРОВАНИЕ ПЕРИФЕРИЙНЫХ УСТРОЙСТВ, ДОСТУПНЫХ ЧЕРЕЗ ШИНУ I2C

Цель работы: получить навыки программирования периферийных устройств, подключенных к микропроцессорной системе по шине I 2C.

Теоретическая часть

Интерфейс I^2C

Шина I^2C (*Inter-Integrated Circuit*) — последовательная шина данных с шинной топологией соединения. Применяется для связи интегральных схем при помощи двух линий:

SDA (*Serial DAta*) — последовательная линия данных (SDATA);

SCL (*Serial CLock*) — линии синхронизации (SCLOCK).

Обе линии являются двунаправленными, т. е. ведущий и ведомый могут быть как приемником, так и передатчиком.

В основном варианте использования интерфейс соединяет одно ведущее устройство с одним или несколькими ведомыми устройствами (рис. 25). Стандарт допускает подключение нескольких ведущих к одной шине I 2C, описывает определение ошибок при одновременном обмене с участием нескольких ведущих (арбитраж). В стандартном режиме обеспечивается передача последовательных 8-битных данных со скоростью до 100 кбит/с, до 400 кбит/с в «быстром» режиме.

Вторая версия стандарта увеличила скорость передачи до 3,4 Мбит/с.

SCL SDA	SCL SDA	SCL SDA	SCL SDA
D R			

Типичная последовательность передачи данных по интерфейсу I_2C начинается с условия START, которое характеризуется переходом линии SDATA из 1 в 0, при этом на линии SCLOCK удерживается высокий уровень. Затем по линии SDATA ведущий передает байт информации, 7 старших бит содержат адрес ведомого, 8-й бит (младший) является разрядом чтения/записи (R/W), который определяет направление передачи. Если бит R/W очищен, то ведущий будет передавать данные выбранному ведомому устройству, если установлен — ожидает передачу информации от ведомого. Когда адрес ведомого устройства совпадает с запрашиваемым, оно возвращает сигнал подтверждения приема ACK (по линии SDATA низкий

уровень), иначе оно посылает сигнал отказа NACK (высокий уровень линии SDATA).

Все передаваемые сигналы по SDATA тактируются по линии SCLOCK.

После выбора ведомого происходит передача информационного байта. Во время передачи данных сигналы ACK и NACK всегда генерируются приемником. В таком случае девятый тактовый импульс, необходимый для этих сигналов, всегда генерируется ведущим. Передатчик должен освободить линию SDATA во время девятого тактового импульса.

Последовательность передачи данных заканчивается условием STOP, генерируемым ведущим устройством. STOP определяется переходом линии SDATA с низкого на высокий уровень, когда линия SCLOCK находится в высоком состоянии.

В расширенной версии устройство может иметь 10-битный адрес, что позволяет адресовать до 1008 периферийных устройств. При этом адрес ведомого передается двумя байтами. Использование 7-битного адреса допускает взаимодействие со 112 микросхемами, т. к. 16 адресов зарезервированы.

Основные преимущества шины I 2C:

использование двух проводников для подключения нескольких устройств;

возможна одновременная работа нескольких ведущих устройств, подключенных к одной шине I²C;

стандарт предусматривает «горячее» подключение и отключение устройств в процессе работы системы;

наличие достаточно подробного и жесткого стандарта на шину I2C обуславливает выпуск меньшего количества несовместимых I 2C-микросхем у различных производителей;

встроенный фильтр подавляет всплески, обеспечивая целостность данных.

Основные недостатки шины I 2C:

максимальное допустимое количество микросхем, подсоединенных к одной шине, ограничивается максимальной емкостью шины в 400 пФ;

сложность реализации режима с несколькими ведущими;

трудность локализации неисправности, если одно из подключенных устройств ошибочно устанавливает на шине состояние низкого уровня.

```
;*****************************************************************
; Пример организации программы для получения времени
; (4 байта) из часов по шине I2C (или для установки времени
; в часах путем отправки 4 байтов по шине I2C), где используются
; функции из библиотеки I2C.LIB для работы с шиной I2C
;*****************************************************************
extrn code (receiveblock, sendblock, getack); декларация сегмента
                                             ; программы
org 2000h                ; установка адреса начала программы
; начало программы
         mov dptr, #5000h ; адрес области в xdata для размещения
                          ; времени
         call gettime     ; получить 4 байта из часов
         call puttime     ; отправить 4 байта в часы
         jnb F0, done     ; F0=0, нет ошибки
; здесь можно разместить обработчик ошибки при F0=1
; команды обработчика
; конец обработчика ошибок по флагу F0=1
done:    sjmp $           ; бесконечная петля
```

Практическая часть

Программирование микросхемы часов реального времени

Изучите подпрограммы взаимодействия с часами через интерфейс I 2C.

Создайте новый проект в интегрированной среде Keil µVision IDE, подключите библиотеку I2C. LIB. Изучите технологию доступа к часам реального времени и разберитесь с форматом представления времени в часах. Напишите программу, позволяющую:

применяя подпрограмму PutTime, установить в часах текущее время

используя бесконечный цикл, на основе подпрограммы GetTime считывать содержимое часов и выводить полученное значение:

вариант 1 — на экран компьютера средствами Т2 в режиме эмуляции терминала;

вариант 2 — на ЖКИ стенда

Содержание отчета

В отчет по практической работе должны быть включены

титульный лист;

цель работы;

исходные тексты разработанных программ с комментариями к каждой строке или блоку, показывающие суть операций в контексте разработанной программы, а не фактическое действие с регистрами, константами и т. п.;

в практической части приведите фото ЖКИ стенда или экрана компьютера с выведенным временем;

заключение и выводы.

Контрольные вопросы

1. Какие линии связи используются в интерфейсе I $_2$C? Опишите формат передачи данных по шине I $_2$C.

2. Сколько ведомых и ведущих устройств позволяет подключать шина I $_2$C?

3. Укажите основные отличия интерфейсов I $_2$C и SPI.

4. Какие устройства подключены к шине I $_2$C в лабораторном стенде SDK-1.1?

5. Назовите регистры управления шиной I $_2$C. Какие режимы и параметры шины позволяют конфигурировать эти регистры?

6. Опишите предназначение, входные и выходные параметры процедур, содержащихся в библиотеке I2C.LIB. Поясните функционал процедур, используемых в работе для взаимодействия с часами реального времени (gettime и puttime).

7. В каком формате представляются данные в микросхеме часы-календарь типа PCF8583? Как осуществить вывод времени на ЖКИ, в порт УАПП?

ПРАКТИЧЕСКАЯ РАБОТА №32. ИЗУЧЕНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ТАЙМЕР-СЧЕТЧИКА МИКРОКОНТРОЛЛЕРА MCS-51.

Цель работы: изучить функциональные возможности таймер-счетчика микроконтроллера MCS-51.

Теоретическая часть

1. Таймер-счетчики микроконтроллеров семейства MCS-51

Базовые модели микроконтроллеров семейства MCS-51 содержат два программируемых многорежимных таймер-счетчика (0 и 1), предназначенных для подсчета внешних событий (выводы Т0 и Т1), организации программно управляемых временных задержек и измерения

временных интервалов. Кроме того, таймер 1 применяется для определения скорости передачи последовательного порта.

Таймер-счетчик может работать в режиме таймера или в режиме счетчика. В первом случае ведется подсчет тактов деленной системной частоты (определенный промежуток времени) и при переполнении выдается запрос прерывания. В каждом машинном цикле длительностью 12 тактов регистр таймера инкрементируется только один раз, поэтому скорость счета таймера равна $f_{OSC}/12$.

В режиме счетчика ведется подсчет количества поступивших импульсов на вход микросхемы, причем идентификация импульса производится по заднему фронту. При переполнении таймерного регистра таймер-счетчика выдается запрос прерывания. Распознавание спада внешнего сигнала занимает 24 периода тактовой частоты (2 машинных цикла), поэтому максимальная скорость счета равна $f_{OSC}/24$.

Управление режимами работы таймер-счетчиков и организация их взаимодействия с системой прерываний обеспечивается двумя регистрами специальных функций TMOD и TCON. Текущее значение таймер-счетчика, соответствующее количеству подсчитанных импульсов, хранится и изменяется в таймерных регистрах TH0, TL0 и TH1, TL1 соответственно для таймер-счетчика 0 и 1. В различных режимах разрядность таймер-счетчика составляет 8–16 бит, таким образом, для подсчета используются либо только регистры TL*x*, либо TH*x* и TL*x*, включенные последовательно, где TH*x* содержит старшие биты числа, а TL*x* — младшие, *x* — номер таймер-счетчика (0 или 1).

2. Регистр режима работы таймер-счетчика TMOD

Управление режимом работы таймер-счетчиков 0 и 1 осуществляет регистр TMOD:

Timer 1 (таймер-счетчик 1)				Timer 0 (таймер-счетчик 0)			
D7	D6	D5	D4	D3	D2	D1	D0
GATE	C/T	M1	M0	GATE	C/T	M1	M0

Поля регистра TMOD:

C/T — выбор функции: 0 — таймер, 1 — счетчик;

GATE — разрешение внешней блокировки. Если бит равен нулю, то включение и выключение соответствующего таймер-счетчика возможно только битом TR*x* регистра TCON. В случае, когда бит равен единице, включение таймер-счетчика зависит не только от бита TR*x*, но и от состояния на входе INT*x*, на который необходимо подать уровень логической единицы для активации работы соответствующего таймер-счетчика;

M [1:0] — код режима работы таймеров (табл. 1).

Таблица 1

Режим работы таймер-счетчика

M1	M0	Режим	Режим работы таймер-счетчика
0	0	0	13-битный таймер-счетчик. TH*x* — 8 бит, TL*x* — 5 (младших) бит
0	1	1	16-битный таймер-счетчик. TH*x* и TL*x* включен последова тельно
1	0	2	8-битный автоперезагружаемый таймер-счетчик. TH*x* хранит значение, которое должно быть перезагружено в TL*x* каждый раз по переполнению

1	1	3	Таймер-счетчик 0 и 1 работают по-разному

3. Регистр управления и статуса таймера TCON

Регистр TCON управляет запуском таймер-счетчиков, содержит флаги переполнения таймер-счетчика, также используется для настройки прерываний от внешних источников. Структура регистра TCON:

TCON.7	TCON.6	TCON.5	TCON.4	TCON.3	TCON.2	TCON.1	TCON.0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Поля регистра TCON:

TF*x* — флаг переполнения таймер-счетчика. Устанавливается аппаратными средствами при переполнении таймер-счетчика, т.е. в случае перехода из максимального состояния таймерного регистра в минимальное. Сбрасывается автоматически при передаче управления подпрограмме обработки прерывания;

TR*x* — бит управления таймер-счетчика. Для активации работы таймер-счетчика 0 или 1 в соответствующий бит необходимо записать единицу. Сброс бита выключает соответствующий таймер-счетчика;

IE*x* — флаг фронта прерывания. Устанавливается аппаратно при возникновении активного сигнала на внешнем входе INT*x* микроконтроллера (активный сигнал определяется битом IT*x*). Сбрасывается автоматически при обслуживании прерывания;

IT*x* — бит выбора типа активного сигнала на входе INT*x*. При IT*x* = 1 активным является переход из высокого в низкий, при IT*x* = 0 активным является низкий уровень сигнала.

4. Режимы работы таймер-счетчиков

Регистр TMOD позволяет выбрать один из четырех режимов работы для каждого таймер-счетчика, причем режимы работы 0, 1 и 2 одинаковы для обоих таймер-счетчиков, а режим 3 различен.

Схема функционирования таймер-счетчика в режиме 0 показана на рис. 1, где физические выводы микроконтроллера обозначены PIN.

Для этого режима разрядность таймерного регистра составляет 13 бит, из которых 8 старших битов текущего значения содержатся в регистре TH*x*, а 5 младших битов — в регистре TL*x*. Модуль счета (число различных устойчивых состояний счетчика) для данного режима составляет $2^{13} = 8192$. Например, если TH*x* = 29h и TL*x* = 15h, то в двоичной системе счисления этим числам соответствует запись: TH*x* = 00101001b и TL*x* = 00010101b, тогда значение в таймерном регистре можно считать равным 0010100110101b, что в шестнадцатеричной системе счисления составляет 535h, а в десятичной — 1333.

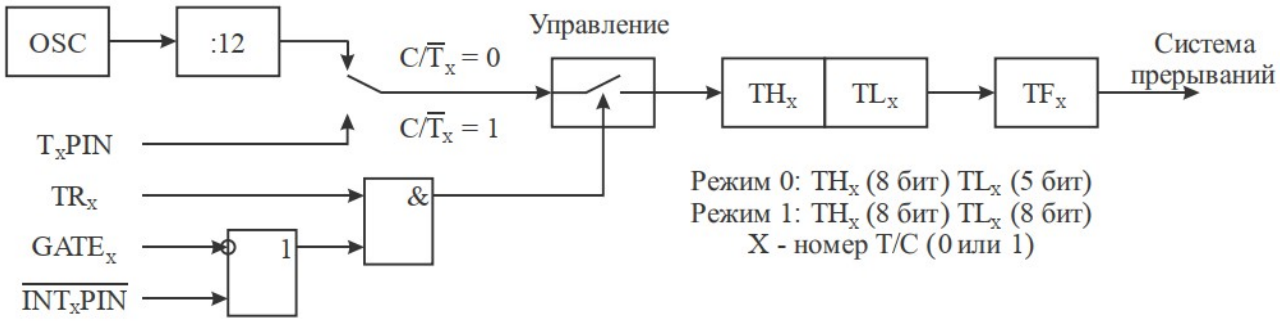


Рис. 1. Схема работы таймер-счетчика в режиме 0 и 1

При переполнении, т.е. переходе из максимального состояния, равного 8191, в минимальное — 0, в регистре TCON автоматически устанавливается флаг переполнения таймер-счетчика TF*x*.

Подсчет импульсов, поступающих с внешнего входа T*x* (см. рис. 1) или с генератора тактовой частоты через предделитель, осуществляется счетным узлом в двух случаях: когда управляющий бит TR*x* установлен и бит разрешения внешней блокировки GATE сброшен, либо когда TR*x* = 1, GATE = 1 и на внешнем входе микроконтроллера INT*x* присутствует уровень логической 1.

Функционирование любого таймер-счетчика в режиме 1 полностью совпадает с режимом 0 за исключением того, что таймерный регистр имеет разрядность 16 бит. В этом случае модуль счета будет равен 65536, а регистры TH*x* и TL*x* используются полностью и также включены последовательно.

Как видно из рис. 2, управление работой таймер-счетчика в режиме 2 осуществляется аналогично режимам 0/1. Первым отличием функционирования является использование для подсчета только регистра TL*x*, т. е. модуль счета равен 256. Вторым — при переполнении таймерного регистра, кроме установки флага переполнения, производится запись содержимого регистра TH*x* в регистр TL*x*, при этом значение TH*x* не изменяется. Таким образом можно уменьшить модуль счета с 256 до любого значения, предварительно записав соответствующую разницу в регистр TH*x*. Конечно, модуль счета может быть уменьшен и при использовании других режимов работы, но в данном случае перезапись определенного начального значения будет производиться автоматически.

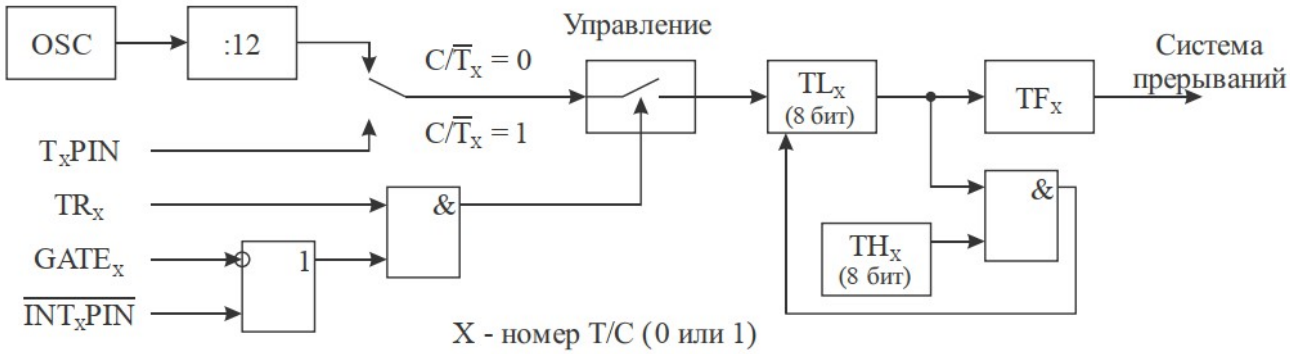


Рис. 2. Схема работы таймер-счетчика в режиме 2

Функционирование таймер-счетчиков 0 и 1 в режиме 3 различно (рис. 7). Таймер-счетчик 1 отключен, его таймерные регистры TH1 и TL1

сохраняют свое значение. Регистры TL0 и TH0 используются в качестве двух независимых таймерных регистров, причем TH0 может выполнять функции только таймера, а TL0 — таймера и счетчика.

Таймер с регистром TH0 управляется только битом TR1, соответственно его можно только включить или выключить, других настроек произвести нельзя. При переполнении TH0 устанавливается флаг прерывания TF1. Работа TL0 аналогична функционированию в режимах 0 и 1, отличием является разрядность таймерного регистра, здесь она составляет 8 бит, управление производится битами таймер-счетчика 0 (C/T0, GATE0, TR0), вход для внешней блокировки — INT0 и флаг переполнения — TF0.

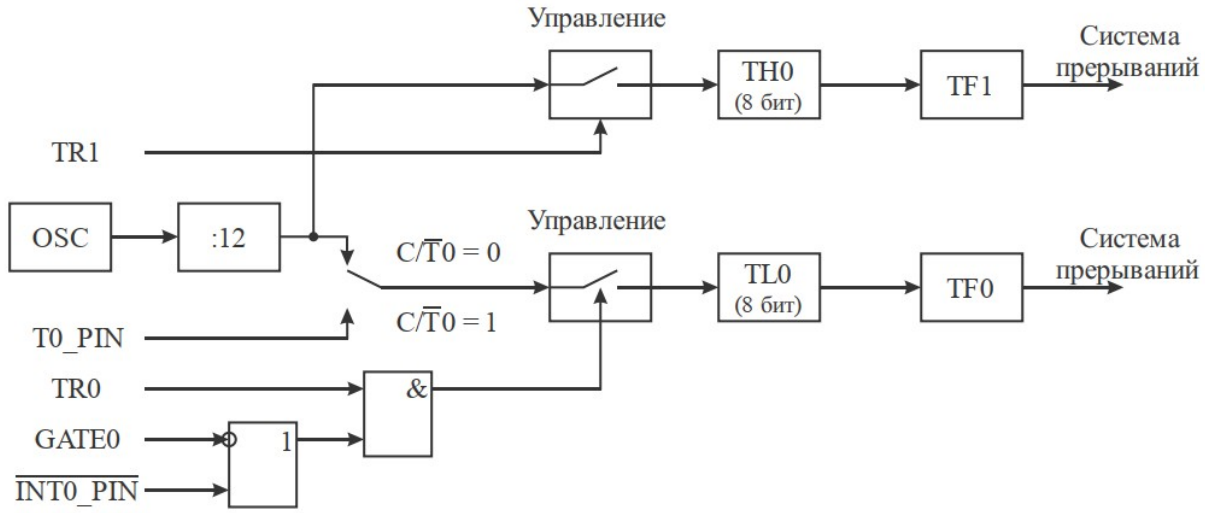


Рис. 3. Схема работы таймер-счетчика в режиме 3

В некоторых версиях микроконтроллеров семейства MCS-51 может присутствовать третий таймер-счетчик и (или) блок программных счетчиков PCA (Programmable Counter Array), которые также могут использоваться для отсчета временных интервалов.

Практическая часть

Используя систему команд выполните задания в системе моделирования микроконтроллера семейства MCS-51 «Single-Chip Machine».

1. Программа управления таймер-счетчиком.

Введите исходный код программы:

```
mov tmod, #02h
mov tl0, #E0h
mov tcon, #10h
L1:
jnb tf0, L1
```

Выполните в пошаговом режиме программу, определите назначение каждой команды и общую функцию программы.

2. Программирование таймер-счетчика.

Разработайте программу, организующую отсчет установленного числа итераций, с использованием указанного таймер-счетчика в определенном режиме. Функция таймер-счетчика — таймер. Варианты заданий приведены в табл. 2.

3. Программа расчета математической функции.

Разработайте программу, выполняющую математические операции. Варианты заданий приведены в табл. 2. Числа a, b, c выбирайте самостоятельно. Для выполнения операций необходимо использовать регистры R2 (число a), R3 (число b), R4 (число c) и A. В начале программы расположите команды загрузки исходных чисел в регистры.

Математические операции необходимо производить с выбранными регистрами. Последней командой разместите полученный результат в ячейку внутренней памяти данных с адресом 40h.

Содержание отчета

Отчет должен включать в себя:

титульный лист;

цель работы;

исходные тексты разработанных программ с комментариями к каждой строке, показывающие суть команды в контексте разработанной программы, а не фактическое действие с регистрами, константами и т. п.; к заданию А приведите комментарии к каждой строке и общее описание работы программы. К заданию 2 требуется, кроме комментариев, показать расчет начального значения, загружаемого в регистры таймер-счетчика. В задании 3 при написании комментариев показывайте суть команды в контексте данной программы, используя названия исходных переменных a, b и c;

заключение и выводы.

Контрольные вопросы

1. Какие основные блоки базового микроконтроллера семейства MCS-51 вы знаете?
2. Как организована система памяти изучаемого семейства микроконтроллеров?
3. Назовите функции и режимы таймер-счетчика, его регистры управления.
4. Для каких целей устанавливается начальное значение таймерсчетчика? Как его вычислить?
5. Какие команды можно использовать для задания режимов таймер-счетчика, установки начального значения, его запуска?
6. Укажите основные команды математических операций на языке ассемблера.
7. В каком регистре находится, а также с какой целью используется бит переноса C в математических операциях?

Таблица 2

Варианты заданий 2 и 3

Номер варианта	Задание 2			Задание 3
	Номер таймер-счетчика	Режим таймер-счетчика	Количество итераций	Математическая операция
1	0	0	10h	$(a+b)\cdot c-4$
2	1	0	20h	$c-a\cdot b+2$
3	0	1	30h	$b/c+a-3$
4	1	1	08h	$(a+4)/(b-c)$
5	0	0	04h	$(a+c-b)/2$
6	1	0	10	$a-c/b+3$
7	0	1	15	$a\cdot b+c-2$
8	1	1	25h	$(a-b)/c+5$
9	0	0	21	$c\cdot b+a-3$
10	1	0	40h	$2+a/c-b$
11	0	1	50	$15-a\cdot(b+c)$
12	1	1	32	$(b-2)\cdot a/c$
13	0	0	64	$a+c-b/4$
14	1	0	12	$b\cdot a+c-5$
15	0	1	05h	$(a+4)/(b-c)$
16	1	1	48	$c/(a-b)+2$

Номер варианта	Задание 2			Задание 3
	Номер таймер-счетчика	Режим таймер-счетчика	Количество итераций	Математическая операция
17	0	0	18h	$b-5\cdot c+a$
18	1	0	16	$a\cdot b-c+4$
19	0	1	28h	$(a+2)\cdot c-b$
20	1	1	11	$4+b\cdot c-a$
21	0	0	1Ch	$b+c\cdot a-3$
22	1	0	35h	$(4+a)\cdot c-b$
23	0	1	17	$(a+4-b)/c$
24	1	1	12	$c+b\cdot a-5$
25	0	0	13h	$b/a-c+7$
26	1	0	24	$(b+3)/(c-a)$
27	0	1	1Ah	$12/(a-b)+c$
28	1	1	19	$b-c/b+a$
29	0	0	2Bh	$a/b+c-1$
30	1	0	21	$(a-b/c)+4$
31	0	1	31h	$20-a\cdot(b+c)$
32	1	1	9h	$(b+c)\cdot c/a$

ПРАКТИЧЕСКАЯ РАБОТА №33. ИССЛЕДОВАНИЕ СИСТЕМЫ ПРЕРЫВАНИЙ МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА MCS-51.

Цель работы: исследование системы прерываний микроконтроллеров семейства MCS-51 с помощью персонального компьютера и программных средств отладки.

Практическая часть

1. Зафиксировать содержимое регистров, флагов и ячеек памяти микроконтроллера после загрузки эмулятора (avsim51 -c1 a). Чему равно содержимое указателя стека? Разрешены ли прерывания? На какой режим настроены таймеры? Какая установлена скорость выполнения программы?

2. Составить комментарий к программе преобразования двоичного числа, задаваемого на линиях порта P1, в двоично-десятичное содержимое регистра DPTR:

```
MOV   A,P1
MOV   B,#100
DIV   AB
MOV   DPH,A
MOV   A,#10
XCH   A,B
DIV   AB
SWAP  A
ORL   A,B
MOV   A,B
```

В режиме Patch Code ввести текст программы в эмулятор и проверить ее работу в пошаговом (F10) режиме. Как выполняется команда деления? Какие флаги PSW изменяются при выполнении программы?

3. Записать в первые две ячейки памяти программ программу, состоящую из одной команды SJMP 0 (80 FE), и запустить ее на выполнение в автоматическом режиме. Почему не работают таймеры T/C0 и T/C1?

Установив TR0=1, проверить работу T/C0 в режиме таймера (скорость счета изменяется клавишей F5) и счетчика событий (TMOD.2=1). Перепады на линии T0 (P3.4) формировать с помощью клавиши Insert. В каком диапазоне изменяется содержимое регистров TL0 и TH0 при работе T/C0 в режиме 0? Когда устанавливается флаг TF0?

Проверить работу T/C1 в режиме 1. Установив TR1=1 и GATE1=1, проверить возможность аппаратного управления работой таймера уровнем сигнала на входе INT1 (P3.3).

Перевести T/C0 в режим 2 (8-битный автоперезагружаемый таймер/счетчик). Установив (TH0)=0D5H, проследить работу T/C0 в режиме таймера и счетчика событий.

Перевести T/C0 в режим 3 (TL0 и TH0 функционируют как два независимых 8-битных счетчика). Возможно ли в этом режиме использование прерываний от T/C1?

4. В режиме Patch Code ввести в эмулятор текст программы, при реализации которой регистры R0, R1, R2, R3, R4 фиксируют число выполнения подпрограмм обслуживания прерываний от различных источников, а аккумулятор работает в режиме двоичного счетчика:

```
ORG  00H
INC  A
SJMP 0
ORG  03H
INC  R0
RETI
ORG  0BH
INC  R1
RETI
ORG  13H
INC  R2
RETI
ORG  1BH
INC  R3
RETI
ORG  23H
INC  R4
CLR  SCON.0
CLR  SCON.1
RETI
```

Разрешить прерывания по входу INT0, установив в режиме окна EA=1 и EX0=1. При работе программы в автоматическом режиме исследовать различие механизма обработки прерывания при IT0=0 и IT0=1 (по уровню и по срезу P3.2).

Разрешить прерывания и по входу INT1. При IT0=IT1=0 установить INT0=INT1=0 и запустить программу. Почему не выполняется подпрограмма обслуживания прерываний по входу INT1? Повторить работу программы, установив в регистре приоритетов прерываний PX1=1.

Разрешить все прерывания. Установить TR0=TR1=IT0=IT1=1. Запустить программу на выполнение. Убедиться, что периодически выполняются подпрограммы обслуживания прерываний по переполнению таймеров. Что происходит при изменении содержимого буферных регистров приемника и передатчика последовательного порта SBUF? Проимитировать внешние прерывания по входам INT0 и INT1.

Остановить выполнение программы. Установить все флаги прерываний (IE0, IE1, TF0, TF1, RI и TI). Продолжить выполнение программы в пошаговом режиме. Объяснить поведение микроконтроллера. В какой момент сбрасываются флаги IE0, IE1, TF0, TF1(при передаче управления подпрограмме обслуживания или по команде RETI)? Повторить эксперимент, установив в регистре приоритетов PS=1. Объяснить новую последовательность выполнения подпрограмм обслуживания прерываний. Что будет, если при выполнении подпрограммы

обслуживания прерываний пришел запрос прерываний с большим приоритетом? Нужно ли сбрасывать программно флаги TF0, TF1, RI и TI?

5. Испытать на эмуляторе работу следующей программы, формирующей в аккумуляторе двоично-десятичный код длительности импульса (единицы и десятые доли мс) на входе INT0:

```
        ORG  00H          ; RESET
        MOV  TH0,#156     ; Загрузка регистров T/C0
        MOV  TL0,#0
        MOV  TMOD,#0AH ; Настройка T/C0 на режим 2
        SJMP M1
        ORG  0BH          ; Вектор прерывания от T/C0
        ADD  A,#1         ;Подпрограмма обслуживания
        DA   A            ; прерываний
        RETI              ; Возврат из подпрограммы
M1:     CLR  A            ; Очистка аккумулятора
        MOV  IE,#82H      ;Разрешение прерываний от
                          T/C0
        SETB TR0          ; Запуск таймера T/C0
        SJMP $            ; Зацикливание программы
```

Начиная с адреса 0BH записана подпрограмма обслуживания прерываний по таймеру T/C0. После каждого переполнения таймера (т.е. через каждые 100 мкс при частоте кварца 12 МГц) содержимое двоично-десятичного счетчика, организованного в аккумуляторе, увеличивается на единицу. Основная программа начинается с нулевой ячейки, при выполнении обходит ячейки, занятые подпрограммой, и заканчивается командой SJMP $.

Таймер T/C0 настраивается на режим 8-разрядного счетчика с автоперезагрузкой и возможностью аппаратного запуска логической 1 на входе INT0 (перед запуском программы на этом входе надо зафиксировать логический 0). В регистр TH0 загружается дополнительный код числа минус 100.

Проимитировав на входе INT0 импульс длительностью 10 мс, измерить секундомером реальное время работы программы при наивысшей скорости (HI). Во сколько раз скорость воспроизведения программы с помощью эмулятора отличается от реального масштаба времени?

Контрольные вопросы

1. Разрешены ли прерывания после системного сброса?
2. Может ли быть прервано выполнение программы обработки прерывания с высоким уровнем приоритета?
3. Транслировать команду JB TF0,$+5.
4. Что происходит при выполнении команды CJNE A,#40,M1?
5. Какой флаг устанавливается после выполнения команды MOV SBUF,B?
6. Какими командами можно загрузить в T/C0 дополнительный код числа 10000?

ПРАКТИЧЕСКАЯ РАБОТА №34. РАЗРАБОТКА ПРОГРАММЫ ИЗМЕРЕНИЯ НАПРЯЖЕНИЯ

Цель работы: приобрести навыки разработки программ измерения напряжений при помощи АЦП микроконтроллера ADuC812.

Теоретическая часть

Блок АЦП включает в себя 8-канальный 5-микросекундный аналого-цифровой преобразователь с однополярным питанием, который содержит многоканальный мультиплексор, устройство выборки-хранения, встроенный источник опорного напряжения (ИОН), систему калибровок и собственно АЦП. АЦП построен по схеме стандартного конвертора последовательного приближения. Источник опорного напряжения (ИОН) откалиброван на напряжение 2,5 В. Также может использоваться внешний ИОН с напряжением 2,3 В — AV_{dd}.

Все компоненты блока управляются при помощи трех интерфейсных регистров специального назначения: ADCCON1, ADCCON2 и ADCCON3. Выходной код АЦП хранится в регистрах ADCDATAH и ADCDATAL. Для указания адреса в режиме прямого доступа к памяти (ПДП) используются регистры DMAL, DMAH и DMAP.

Регистр управления АЦП ADCCON1

Управление преобразованием, временем переключения, режимами преобразования и потреблением энергии в режиме управления АЦП ADCCON1 осуществляется программным путем. Для этого необходимо задать соответствующее управляющее слово в регистр ADCCON1, функциональное назначение бит которого отражено ниже:

ADCCON1.7	ADCCON1.6	ADCCON1.5	ADCCON1.4	ADCCON1.3	ADCCON1.2	ADCCON1.1	ADCCON1.0
MD1	MD0	CK1	CK0	AQ1	AQ0	T2C	EXC

Поля регистра ADCCON1:

MD1, MD0 — биты, определяющие режим АЦП

Режим работы АЦП

MD1	MD0	Режим
0	0	Отключен
0	1	Нормальный
1	0	Дежурный, если не выполняется цикл преобразования
1	1	Холостой, если не выполняется цикл преобразования

CK1, CK0 — биты, определяющие коэффициент деления тактовой частоты процессора для получения тактовой частоты АЦП. Получение одной выборки АЦП занимает 16 тактов (табл. 12)

Делитель тактовой частоты для АЦП

CK1	CK0	Делитель
0	0	1
0	1	2
1	0	4
1	1	8

AQ1, AQ0 — биты задержки переключения, выбирающие время, которое необходимо для перезарядки устройства выборки-хранения (УВХ) при переключении мультиплексора3.1.

Задержка запуска АЦП

AQ1	AQ0	Число задержки запуска АЦП
0	0	1
0	1	2
1	0	3

1	1	4

T2C — бит запуска преобразования от таймера 2. Если бит установлен, то флаг переполнения T2 используется для запуска преобразования;

EXC — бит разрешения внешнего запуска. Если он установлен, то вывод микроконтроллера 23 (CONVST) будет использован как сигнал запуска.

Регистр управления АЦП ADCCON2

Регистр ADCCON2 управляет выбором номера канала и режимами преобразования:

ADCCON2.7	ADCCON2.6	ADCCON2.5	ADCCON2.4	ADCCON2.3	ADCCON2.2	ADCCON2.1	ADCCON2.0
ADCI	DMA	CCONV	SCONV	CS3	CS2	CS1	CS0

Поля регистра ADCCON2:

ADCI — бит прерывания АЦП, устанавливается аппаратным способом после окончания работы однократного цикла преобразования АЦП или по окончании передачи блока в режиме ПДП. ADCI очищается аппаратно при переходе по вектору на подпрограмму обслуживания прерывания;

DMA — бит разрешения режима ПДП, устанавливается пользователем для начала операции ПДП со стороны АЦП;

CCONV — бит циклического преобразования, устанавливается пользователем для перевода АЦП в режим непрерывного циклического преобразования. В этом режиме АЦП выполняет преобразование в соответствии с типом синхронизации и конфигурацией каналов, которые выбраны в других регистрах управления АЦП;

SCONV — бит запуска однократного преобразования, устанавливается пользователем для однократного запуска АЦП. Бит сбрасывается автоматически по завершении преобразования;

CS3, CS2, CS1, CS0 — биты выбора входных каналов, позволяют осуществить выбор номера канала АЦП под управлением программы. В режиме ПДП выбор номера канала осуществляется из ID-канала, записанного во внешней памяти.

Выбор входных каналов АЦП

CS3	CS2	CS1	CS0	Режим
0	n2	n1	n0	Номер входного канала n2n1n0
1	0	0	0	Температурный сенсор
1	1	1	1	Останов ПДП

Регистр управления АЦП ADCCON3

Регистр управления АЦП ADCCON3 дает индикацию занятости АЦП для прикладных программ.

ADCCON3.7	ADCCON3.6	ADCCON3.5	ADCCON3.4	ADCCON3.3	ADCCON3.2	ADCCON3.1	ADCCON3.0
BUSY	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD

Поля регистра ADCCON3:

BUSY — бит занятости АЦП, только для чтения. Устанавливается на время преобразования или калибровки АЦП. Автоматически очищается в конце циклов преобразования или калибровки;

остальные биты RSVD зарезервированы, при чтении возвращают 0, их следует записывать только нулями.

Режимы работы АЦП

После настройки АЦП при помощи регистров ADCCON1–3 и запуска преобразования в выходных регистрах АЦП ADCDATAH и ADCDATAL начинает появляться 12-разрядный код, соответствующий напряжению на входе МК. В четырех старших разрядах ADCDATAH указан номер канала, с которого был преобразован сигнал:

ADCDATAH								ADCDATAL							
Номер канала				Старшие разряды результата				Младшие разряды результата							
D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0

Если для требуемой обработки сигнала, поступившего с АЦП, необходимо большее время, чем пауза между преобразованием, то используется режим ПДП. До включения режима ПДП размечают область внешней памяти данных, в которую будут сохраняться выборки. Разметка состоит в записи идентификаторов номеров каналов ID (четыре старших разряда) во внешней памяти.

Разметка внешней памяти для режима ПДП

000000 H	0	0	1	0	Преобразовывать канал 2
0	1	0	1	Преобразовывать канал 5	
1	0	0	0	Преобразовывать температурный сенсор	
0	1	0	0	Преобразовывать канал 4	
0	1	0	0	Преобразовывать канал 4	
00000A H	1	1	1	1	Остановка

После разметки указывается стартовый адрес блока во внешней памяти. Значение адреса заносится в регистры в следующем порядке:

DMAL (*low* — младшая часть адреса), DMAH (*high* — старшая) и DMAP (*page* — страница), например, 000000 H. Окончание таблицы ПДП обозначается записью «1 1 1 1» в поле выбора канала. Перед запуском преобразования необходимо сконфигурировать регистры управления АЦП, выбрав необходимые параметры преобразования.

Запуск АЦП в режиме ПДП осуществляется установкой бита разрешения (ADCCON2.6, DMA). По окончании преобразования установится бит прерывания АЦП ADCI (ADCCON2.7), и во внешней памяти данных будут доступны результаты работы АЦП в соответствии с предварительной разметкой памяти, как показано в табл. 16. При этом результаты разметки сохранятся.

Содержание внешней памяти после окончания преобразования

000000H	0	0	1	0	x	x	x	x	Канал 2
	x	x	x	x	x	x	x	x	x — результат преобразования канала 2
	0	1	0	1	x	x	x	x	Канал 5
	x	x	x	x	x	x	x	x	x — результат преобразования канала 5
	1	0	0	0	x	x	x	x	Температурный сенсор
	x	x	x	x	x	x	x	x	x — результат преобразования сенсора
	0	1	0	0	x	x	x	x	Канал 4
	x	x	x	x	x	x	x	x	x — результат преобразования канала 4
	0	1	0	0	x	x	x	x	Канал 4
	x	x	x	x	x	x	x	x	x — результат преобразования канала 4
00000AH	1	1	1	1	x	x	x	x	Остановка

Практическая часть

Рассмотрите пример инициализации и запуска преобразования АЦП. Создайте новый проект в интегрированной среде Keil µVision IDE (см.п. 3.3, с. 90). Разработайте программу, выводящую на экран ЖКИ стенда входное напряжение измеренное на канале 0 встроенного АЦП.

Предлагается следующая структура программы.

1) Основная программа:

а) инициализация АЦП — 1 режим (нормальный), делитель тактовой частоты для АЦП — 4 (т. к. оптимальный рабочий частотный диапазон АЦП составляет 400 кГц — 3 МГц), число задержки запуска АЦП — 3;

б) основной цикл программы — измерение кода АЦП, преобразование полученного кода в значение напряжения и вывод его на ЖКИ;

2) подпрограмма вывода одного байта на ЖКИ (PutChar).

В данном задании не требуется большой точности, поэтому достаточно использовать 8-битное значение измеренного напряжения, кроме того, такой подход облегчает математические операции. С этой целью необходимо полученное 12-битное число преобразовать в 8-разрядное, убрав наименее значимую часть — 4 младших бита исходного числа. Затем полученное 8-битное значение кода преобразовать в напряжение, учитывая, что источником опорного напряжения является Vref (2,5 В), и вывести на ЖКИ, не забыв перекодировать в ASCII последовательность. Обратите внимание: каждый канал АЦП имеет входной резистивный делитель на 2, выполненный при помощи двух последовательно соединенных резисторов номиналом 10 кОм.

Содержание отчета

Отчет о работе включает:

титульный лист;

цель работы;

исходные тексты разработанных программ с комментариями к каждой строке или блоку, показывающие суть операций в контексте разработанной программы, а не фактическое действие с регистрами, константами и т. п.;

в практической части описание алгоритма преобразования измеренного кода АЦП в напряжение;

заключение и выводы.

Контрольные вопросы

1. Назовите основные параметры АЦП микроконтроллера ADuC812.
2. Какие регистры позволяют управлять АЦП?
3. Опишите режимы работы, поддерживающие АЦП микроконтроллером ADuC812.
4. Предложите алгоритм перевода числа 0–255 в напряжение 0–2,5 В, а затем в ASCII-код для вывода на ЖКИ.

ПРАКТИЧЕСКАЯ РАБОТА №35. ИССЛЕДОВАНИЕ РАБОТЫ ТАЙМЕРОВ/СЧЕТЧИКОВ СОБЫТИЙ МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА MCS-51.

Цель работы: исследование работы таймеров/счетчиков событий микроконтроллеров семейства MCS-51 с помощью персонального компьютера и программных средств отладки.

Практическая часть

1. Апробировать программу, реализующую на микроконтроллере К1830ВЕ51 электронные часы с индикацией часов, минут и секунд реального времени. Все операции по

решению поставленной задачи выполняет подпрограмма обслуживания прерываний. Остальное время контроллер находится в режиме зацикливания основной программы.

```
        ; Начальная установка и запуск часов в 00 00 00

        ORG   00H
        MOV   P0,#0         ; Счетчик часов
        MOV   P1,#0         ; Счетчик минут
        MOV   P2,#0         ; Счетчик секунд
        MOV   R0,#100       ; Начальная загрузка
        MOV   R1,#100       ; счетчиков генератора
        MOV   TH1,#9CH      ; секундных импульсов
        MOV   TMOD,#20H     ; T/C1 в режиме 2
        MOV   IE,#88H       ; Разрешение прерываний от T/C1
        SETB  TR1           ; Старт таймера T/C1
MAIN:   SJMP  MAIN          ; Основная программа
; Подпрограмма обслуживания прерываний

        ORG   1BH           ; Вектор прерывания
        DJNZ  R0,EXIT       ; Задержка в одну
        MOV   R0,#100       ; секунду
        DJNZ  R1,EXIT
        MOV   R1,#100
        JNB   T0,M1         ; Коррекция минут
        JNB   T1,M2         ; Коррекция часов
        MOV   A,P2          ; Счетчик секунд
        ADD   A,#1
        DA    A
        MOV   P2,A
        CJNE  A,#60H,EXIT
        MOV
M1:     MOV   A,P1          ; Счетчик минут
        ADD   A,#1
        DA    A
        MOV   P1,A
        CJNE  A,#60H,EXIT
        MOV   P1,#0
M2      MOV   A,P0          ; Счетчик часов
        ADD   A,#1
        DA    A
        MOV   P0,A
        CJNE  A,#24H,EXIT
        MOV   P0,#0
EXIT:   RETI                ; Возврат из п/п прерываний
```

END

При отладке программы с помощью эмулятора ход часов замедлен. Для ускорения процессов рекомендуется в регистры R0 и R1 загружать число 3, а не 100.

После старта программы производится начальная загрузка регистров секундной задержки, а также счетчиков секунд, минут и часов. Таймер/счетчик T/C1 настраивается на работу в режиме 2, когда TL1 работает как 8-битовый автоперезагружаемый таймер, а TH1 хранит значение, которое перезагружается в TL1 каждый раз по переполнении. Разрешаются прерывания от T/C1, и после его запуска они происходят через каждые 100 машинных циклов (100 мкс при частоте кварца 12 МГц), вызывая выполнение подпрограммы обслуживания с начальным адресом 1BH. Через 10000 прерываний, которые подсчитывают счетчики на регистрах R0 и R1, т.е. ежесекундно, меняется содержимое порта P2, определяющее показания цифрового индикатора секунд.

Двоично-десятичный счетчик минут реализован с помощью порта P1, аналогичный счетчик часов с помощью P0. При включении контроллера счетчики сбрасываются и на цифровые индикаторы заносятся нули. Установка реального времени производится в определенной последовательности. Сначала держат лог. 0 на входе T1 до тех пор, пока индикаторы покажут требуемое число часов. Затем держат лог. 0 на входе T0 до тех пор, пока не высветятся нужные цифры минут. Коррекция осуществляется подачей секундных импульсов на счетчики часов и минут.

Оценить минимальное и максимальное время выполнения подпрограммы обслуживания прерываний.

Контрольные вопросы

1. Разрешены ли прерывания после системного сброса?
2. Может ли быть прервано выполнение программы обработки прерывания с высоким уровнем приоритета?
3. Транслировать команду JB TF0,$+5.
4. Что происходит при выполнении команды CJNE A,#40,M1?
5. Какой флаг устанавливается после выполнения команды MOV SBUF,B?
6. Какими командами можно загрузить в T/C0 дополнительный код числа 10000?

ПРАКТИЧЕСКАЯ РАБОТА №36. УСТАНОВКА ПЕРИФЕРИЙНЫХ УСТРОЙСТВ

Цель работы: ознакомиться с началами программирования периферийных USB устройств с использованием библиотеки libusb 1.0.

Теоретическая часть

USB – промышленный стандарт, определяющий интерфейс между компьютерами и подключаемыми к ним периферийными устройствами. В частности, он определяет механические параметры кабелей и разъемов, электрические параметры и протоколы передачи данных. Стандарт USB ввел единообразие в работе с широким спектром периферийных устройств. До его появления использовалось множество разных интерфейсов для подключения периферийных устройств (последовательный порт RS232C для модема, мыши, параллельный порт для принтера и дисководов IOMEGA ZIP, PS/2 для мыши и клавиатуры, специализированные интерфейсы для подключения сканеров и т. д.). Программирование взаимодействия с USB устройствами достаточно трудоемко. Для его упрощения была построена libusb – многоплатформенная библиотека для работы с USB устройствами из прикладной программы. Библиотека позволяет получать список подключенных к компьютеру устройств, определять их параметры, задавать режим их работы и обмениваться данными с устройствами. Существует несколько классов USB

устройств, например, устройства пользовательского интерфейса, устройства внешней памяти и коммуникационные устройства. С устройствами всех этих классов можно работать, используя интерфейс библиотеки libusb, но детали протокола взаимодействия у них существенно отличаются. Функции libusb, которые используются в практической работе:

libusb_init – инициализация работы с libusb,
libusb_exit – завершение работы с libusb,
libusb_set_debug – установка уровня подробности отладочных сообщений (рекомендуется использовать уровень 3),
libusb_get_device_list – получение списка подключенных к машине USB устройств,
libusb_free_device_list – освобождение памяти, выделенной в функции libusb_get_device_list дл хранения данных со списком устройств,
libusb_get_device_descriptor – получение дескриптора USB устройства,
libusb_get_config_descriptor - получение дескриптора конфигурации USB устройства,
libusb_free_config_descriptor - освобождение памяти, выделенной в функции,
libusb_ref_device – увеличение счетчика числа пользователей устройства на 1 (при первом вызове функции libusb_get_device_list после подключения устройства его счетчик устанавливается в 1),
libusb_unref_device – уменьшение счетчика числа пользователей устройства на 1 (если счетчик уменьшается до нуля, дескриптор устройства удаляется),
libusb_open – открыть устройство (начать работать с устройством) и получить дескриптор устройства, который далее можно использовать для ввода/вывода данных,
libusb_open_device_with_vid_pid - открыть устройство по его идентификаторам производителя и изделия,
libusb_close – закрыть устройство после его использования,
libusb_get_string_descriptor_ascii – получение атрибута дескриптора устройства в виде ANSI C строки,
lib_usb_error_name – преобразование кода ошибки библиотеки libusb в строковое сообщение об ошибке.
libusb_detach_kernel_driver – отключить драйвер ядра ОС Linux от устройства, заданного переданным дескриптором,
libusb_set_configuration – задание номера конфигурации устройства,
libusb_claim_interface – открыть интерфейс устройства,
libusb_control_transfer – передача данных между компьютером и устройством,
libusb_release_interface – закрыть интерфейс устройства.

Работа может выполняться как на сервере, так и на своем ПК под ОС Windows или GNU Linux. В последнем случае нужно самостоятельно установить libusb.

Рассмотрим последовательность действий, выполняемых простейшей программой, осуществляющей пересылку данных между компьютером и устройством пользовательского интерфейса (вариант для ОС Linux).

1. Инициализации библиотеки (вызов функции libusb_init).

2. Открытие устройства по заданным идентификатором производителя и модели (вызов функции libusb_open_device_with_pid_vid). Первый параметр установить в NULL.

3. Отключение драйвера устройства (libusb_detach_kernel_driver). Второй параметр установить в 0.

4. Выбор текущей конфигурации (вызов функции libusb_set_configuration). Номер конфигурации — 0.

5. Открытие интерфейса (вызов функции libusb_claim_interface). Номер интерфейса — 0.

6. Одна или несколько пересылок данных (функция libusb_control_transfer). В минимальном варианте нужно использовать следующий код:

```
#define CTRL_IN LIBUSB_ENDPOINT_IN | LIBUSB_REQUEST_TYPE_CLASS | \
                LIBUSB_RECIPIENT_INTERFACE
#define HID_GET_IDLE 0x02
...
result = libusb_control_transfer(
    handler_of_device,
    CTRL_IN,
    HID_GET_IDLE,
    buffer,
    length,
    5000);
```

7. Закрытие интерфейса (вызов функции libusb_release_interface).

8. Закрытие дескриптора устройства (libusd_close).

9. Деинициализация библиотеки (libusb_exit). Передаваемый параметр установить в NULL.

Практическая часть

1. Реализовать программу, получающую список всех подключенных к машине USB устройств на сервере portal.sscc.ru или на собственном компьютере, на языках ANSI C или C++ с использованием libusb. Для каждого найденного устройства напечатать его класс, идентификатор производителя и идентификатор изделия.

2. Изучить состав и характеристики обнаруженных с помощью реализованной программ USB устройств.

3. Написать программу, производящую пересылку типа HID_GET_IDLE от устройства пользовательского интерфейса к компьютеру.

Контрольные вопросы

1. Какие задачи решает libusb?

2. На какие группы можно разбить функции libusb?

3. Какая последовательность действий производится программой, получающей состав и конфигурацию USB устройств с помощью libusb?

ПРАКТИЧЕСКАЯ РАБОТА №37. ГЕНЕРАТОР ИМПУЛЬСНОГО СИГНАЛА

Цель работы: приобрести навыки программирования ЦАП микроконтроллера ADuC812 и освоить разработку программ генерации импульсных сигналов.

Теоретическая часть

Блок ЦАП

ADuC812 на кристалле содержит два 12-разрядных цифроаналоговых преобразователя. Один SFR управления и четыре SFR данных позволяют управлять работой ЦАП:

DAC0L/DAC1L — содержат 8 младших разрядов данных ЦАП, адреса регистров в пространстве SFR соответственно F9h и FBh;

DAC0H/DAC1H — содержат 4 старших разрядов данных ЦАП, адреса регистров соответственно FAh и FCh;

DACCON — содержат биты управления ЦАП общего назначения, адрес регистра FDh.

Изменение выходного напряжения каждого канала ЦАП производится после записи значения младшей части входного кода в соответствующий регистр DAC0L/DAC1L. Модуль ЦАП позволяет одновременно устанавливать выходное напряжения на обоих каналах.

Для этого используется бит SYNC регистра DACCON. В случае использования 8-разрядного режима работы число, записанное в регистры DACxL, переносится в верхнюю часть 12-разрядного регистра данных ЦАП.

Структура регистра DACCON:

DACCON.7	DACCON.6	DACCON.5	DACCON.4	DACCON.3	DACCON.2	DACCON.1	DACCON.0
MODE	RNG1	RNG0	CLR1	CLR0	SYNC	PD1	PD0

Поля регистра DACCON:

MODE — бит устанавливает режим работы обоих ЦАП. Если MODE равен 1, то 8-разрядный режим (запись восьми битов в DACxL SFR). Если равен 0, то 12-разрядный;

RNG1 — бит выбора диапазона ЦАП1. Если равен 1, то диапазон ЦАП1 0–Vdd. Если равен 0, то диапазон ЦАП1 0–Vref;

RNG0 — бит выбора диапазона ЦАП0. Если равен 1, то диапазон ЦАП0 0–Vdd. Если равен 0, то диапазон ЦАП0 0–Vref;

CLR1 — бит очистки ЦАП1. Если равен 1, то выход ЦАП1 соответствует коду. Если равен 0, то выход ЦАП1 равен 0 В;

CLR0 — бит очистки ЦАП0. Если равен 1, то выход ЦАП0 соответствует коду. Если равен 0, то выход ЦАП0 равен 0 В;

SYNC — бит синхронизации ЦАП0/1. Если равен 1, то выходы ЦАП изменяются, как только данные попадают в регистры DACxL SFR. Можно одновременно обновить выходы обоих ЦАП путем предварительной записи данных в DACxL/H при SYNC = 0.

Выходы ЦАП одновременно обновятся теперь при установке SYNC в 1;

PD1 — бит выключения ЦАП1. Если равен 1, то ЦАП1 включен.

Если равен 0, то ЦАП1 выключен;

PD0 — бит выключения ЦАП0. Если равен 1, то ЦАП0 включен.

Если равен 0, то ЦАП0 выключен.

Практическая часть

Изучите способы организации циклов на языке ассемблер, рассмотрите пример инициализации ЦАП.

Создайте новый проект в интегрированной среде Keil µVision IDE. В соответствии со своим вариантом разработайте программу для формирования импульсного сигнала.

При написании программы используйте 8-разрядный режим работы и 1 канал ЦАП, выберите оптимальный для вашего варианта источник опорного напряжения, задав соответствующее значение в регистре DACCON.

Содержание отчета

В отчете должны содержаться:

титульный лист;

цель работы;

параметры и форма сигнала с указанием напряжения и соответствующего кода в контрольных точках, исходные тексты разработанных программ с комментариями к каждой строке или блоку, показывающие суть операций в контексте разработанной программы, а не фактическое действие с регистрами, константами и т. п.;

фотография экрана осциллографа с полученным сигналом;

заключение и выводы.

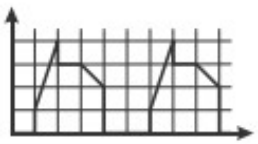
Контрольные вопросы

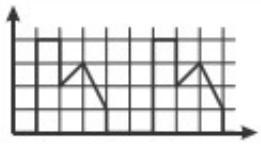
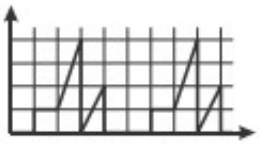
1. Назовите основные параметры ЦАП микроконтроллера ADuC812.
2. Какие регистры позволяют управлять ЦАП?

3. С какой целью в ЦАП микроконтроллера ADuC812 могут использоваться два источника опорного напряжения — Vref (2,5 В) и Vdd (5 В)? Как выбрать оптимальный источник опорного напряжения?
4. Как сформировать увеличение или уменьшение амплитуды сигнала в реальном времени на выходе ЦАП? Каким приемом можно воспользоваться для варьирования длительности сигнала без изменения его формы? Предложите способ программной генерации паузы между импульсами.
5. Подготовьте пример исходного кода программы, предназначенной для реализации линейного увеличения амплитуды выходного напряжения ЦАП в диапазоне 1–1,5 В за период времени 250 мкс.

Варианты задания

Номер варианта	Форма сигнала	Параметры сигнала	Номер варианта	Форма сигнала	Параметры сигнала
1		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 0{,}5$ В	19		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 0{,}5$ В
2		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 1$ В	20		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 1$ В

3		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 1{,}5$ В	21		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 1{,}5$ В
4		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 2$ В	22		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 2$ В
5		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 2{,}5$ В	23		$T = 1250$ мкс $\tau_{min} = 500$ мкс $U_{max} = 2{,}5$ В
6		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 3$ В	24		$T = 1250$ мкс $\tau_{min} = 500$ мкс $U_{max} = 3$ В
7		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 3{,}5$ В	25		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 3{,}5$ В
8		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 4$ В	26		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 4$ В
9		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 4{,}5$ В	27		$T = 1250$ мкс $\tau_{min} = 500$ мкс $U_{max} = 4{,}5$ В
10		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 5$ В	28		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 5$ В
11		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 0{,}5$ В	29		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 0{,}5$ В
12		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 1$ В	30		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 1$ В
13		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 1{,}5$ В	31		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 1{,}5$ В

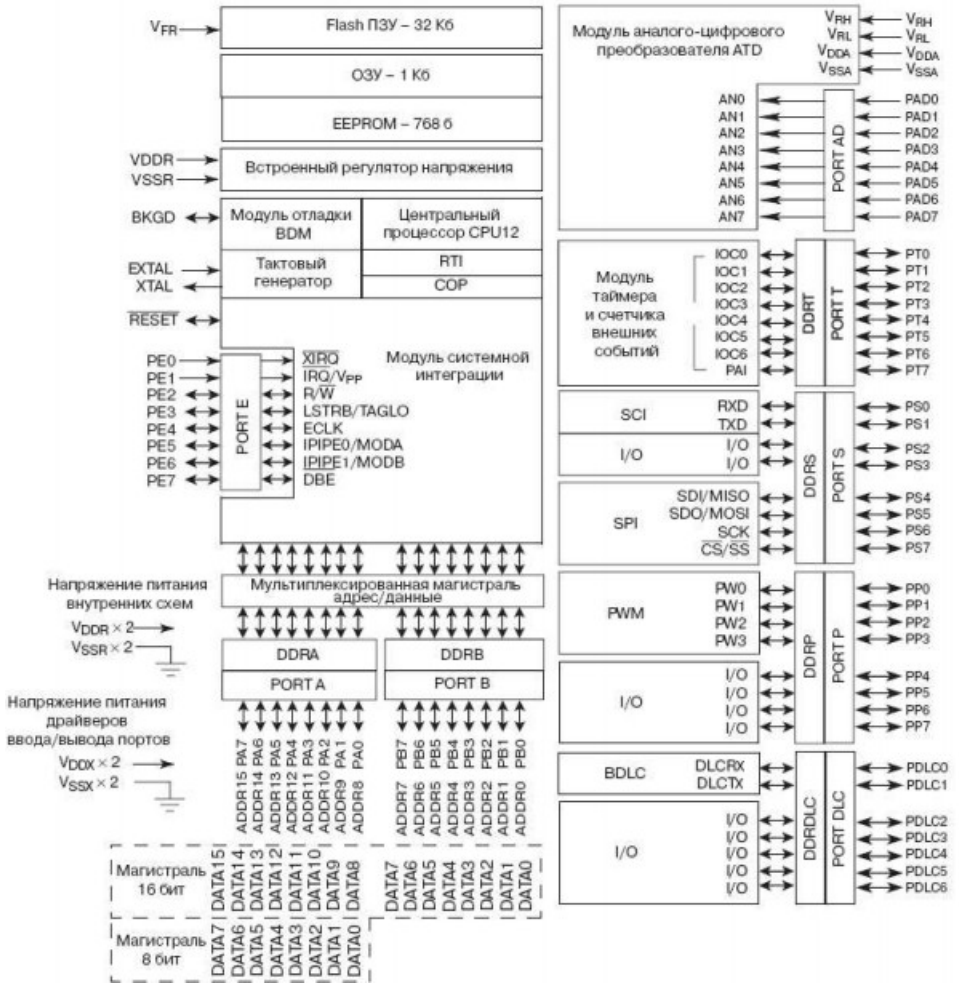
14		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 2$ В	32		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 2$ В
15		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 2,5$ В	33		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 2,5$ В
16		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 3$ В	34		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 3$ В
17		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 3,5$ В	35		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 3,5$ В
18		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 4$ В	36		$T = 1250$ мкс $\tau_{min} = 750$ мкс $U_{max} = 4$ В

ПРАКТИЧЕСКАЯ РАБОТА №38. ПРОГРАММНЫЕ МОДЕЛИ АППАРАТНЫХ СРЕДСТВ МИКРОПРОЦЕССОРНЫХ СИСТЕМ

Цель работы: ознакомление с работой основных аппаратных средств современных микроконтроллеров семейства Motorola.

Общие сведения

Все МК обладают процессорным ядром. Семейство объединяет ряд моделей. Отдельные модели в составе семейства различаются набором периферийных модулей, которые подключаются к внутренней межмодульной магистрали. Основные отличия между отдельными представителями семейства состоят в типе и объеме размещенной на кристалле резидентной памяти, количестве параллельных портов и контроллеров последовательных интерфейсов. Структура современных микроконтроллеров на примере МК MC68HC912B32 приведена на следующем рисунке:



Резидентное ОЗУ объемом 1024 байта (1Кб). Встроенное в кристалл микроконтроллера ОЗУ используется для хранения промежуточных результатов вычислений. Число ячеек, равное 1К, достаточно для большинства прогнозируемых для семейства 68HC12 применений. Резидентная энергонезависимая память данных типа EEPROM объемом 768 б. Этот тип энергонезависимой памяти (энергонезависимая память сохраняет содержимое после отключения питания) обычно используют для сохранения изменяемых констант прикладной программы. Например, в области EEPROM могут храниться коды доступа к данной модели устройства, или на основе ячеек EEPROM могут быть организованы счетчики аварий исполнительного механизма, которым управляет микропроцессорный контроллер. Энергонезависимая память типа EEPROM позволяет выполнять операции записи и перезаписи содержимого ячеек памяти в течение сеанса работы микропроцессорного устройства под управлением прикладной программы, а также чтение ячеек памяти в произвольном порядке.

Резидентная память программ типа Flash объемом 32 Кб. Этот тип памяти предназначен для хранения прикладной программы, которая функционально завершена, прошла отладку и тестирование на реальном объекте. Объем памяти программ МК B32 составляет 32 Кб, что позволяет разместить в ней достаточно большие программы. Использование Flash памяти в качестве памяти программ позволяет реализовать технологию программирования в системе ISP (In System Programming). Эта технология обеспечивает выполнение операций стирания и записи новых кодов в резидентное ПЗУ программ микроконтроллера без демонтажа МК с платы конечного изделия. Мультиплексированная шина адрес/данные для адресации внешней памяти и периферийных устройств. Число выводов корпуса, в котором размещается полупроводниковый кристалл микроконтроллера, ограничено. Причиной тому – стремление разработчика выпускать миниатюрные и относительно недорогие МК, в то время, как увеличение числа выводов корпуса увеличивает его размеры и стоимость. Одним из способов сокращения числа выводов корпуса МК при сохранении функций этих выводов является мультиплексирование линий магистралей

адреса и данных для сопряжения МК с внешней памятью. При мультиплексировании одни и те же выводы МК на протяжении одного временного интервала используются для передачи информации об адресе внешней ячейки памяти, а в течение другого временного интервала – для обмена данными с этой ячейкой. В микроконтоллере В функции мультиплексированных во времени магистралей адрес/данные выполняют линии портов Port A и Port B (см. рис.).

Аппаратные средства современных микроконтроллеров на примере МК Motorola

Многофункциональный таймер. Подсистема реального времени МК семейства 68HC12 включает несколько модулей, но основным является таймер с 16-разрядным счетчиком временной базы, программируемым делителем частоты тактирования и 8 каналами входного захвата IC (Input Capture) или выходного сравнения OC (Output Compare). Эти каналы могут быть сконфигурированы произвольно: любое число каналов из 8 настраивается на реализацию функции входного захвата IC, оставшиеся каналы – на функцию выходного сравнения OC. При этом возможны конфигурации, когда все каналы находятся в режиме IC или в режиме OC. Такая организация модуля таймера позволяет производить точные измерения временных характеристик входных сигналов МК, и генерировать многоканальные импульсные последовательности на его выходах.

Независимый 16 разрядный счетчик внешних событий. Этот модуль также принадлежит к подсистеме реального времени. Он предназначен для подсчета так называемых внешних событий, каждое из которых представляется импульсом на одном из входов МК. Модули контроллеров последовательных интерфейсов SPI (Serial Peripheral Interface) и SCI (Serial Communication Interface).

Микроконтроллеры семейства 68HC12 обладают достаточно мощными аппаратными средствами для обмена с другими устройствами в последовательном коде. Модуль контроллера SPI реализует обмен в синхронном режиме, в то время как модуль SCI предназначен для асинхронного последовательного обмена. Синхронный режим обмена в стандарте SPI характеризуется более высокими скоростями обмена по сравнению с стандартным асинхронным протоколом. Однако расстояние между взаимодействующими устройствами ограничено 20…30 см. Интерфейс в стандарте SPI часто используется для подключения к МК дополнительных интерфейсных компонентов, установленных на плате с МК. Например, МК семейства 68HC12 не имеют в своем составе цифроаналоговых преобразователей (ЦАП). Поэтому система с МК может быть дополнена внешней ИС ЦАП, подключенной к микроконтроллеру с использованием встроенного модуля контроллера SPI. Интерфейс асинхронного обмена SCI часто используется для обмена данным между двумя и более контроллерами, т.к на его основе созданы интерфейсы, сигналы которых могут передаваться на значительные расстояния.

Модуль аналого-цифрового преобразователя ATD (Analog To Digital convertion system). Большинство сигналов в системах, которыми управляют микроконтроллеры, имеют аналоговую природу. Например, температура и давление воздуха окружающей среды изменяются плавно, а не скачкообразно. Для работы с аналоговыми сигналами в микропроцессорной системе, эти сигналы должны быть предварительно преобразованы в цифровую форму. В русскоязычной литературе говорят, что эти сигналы должны быть оцифрованы. Для этой цели в составе МК B32 имеется модуль аналого-цифрового преобразователя. Модуль имеет 8 входов для одновременного подключения восьми измеряемых аналоговых сигналов. Однако оцифровка этих сигналов будет производиться последовательно. Измеряемые аналоговые сигналы будут подключаться ко входу одного аналого-цифрового преобразователя (АЦП) посредством встроенного в модуль ATD мультиплексора. Оцифрованный сигнал представляется в 8 и или 10 разрядном прямом коде без знака. Два дополнительных бита кода представления результата позволяют увеличить чувствительность АЦП с 19.53 до 4.88 мВ.

Модуль широтно-импульсного модулятора PWM (Pulse Width Modulation). Широтно-импульсная модуляция (ШИМ) – один из способов формирования импульсного сигнала с регулируемыми временными характеристиками. Способ широтно-импульсной модуляции часто используется для регулирования скорости вращения двигателей постоянного тока, а также для управления электрическими двигателями других типов. Для генерации ШИМ сигнала в МК B32

могут быть использованы аппаратные средства модуля многофункционального таймера TIM. Однако МК B32 оснащен специальным модулем ШИМ. Этот модуль позволяет генерировать четыре независимых импульсных последовательности с 8 разрядным разрешением для задания коэффициента заполнения, или две импульсные последовательности с 16 разрядным заданием коэффициента заполнения. Допускается комбинация этих режимов. Например, ШИМ сигнал используется для управления двигателем рулевого управления в игрушечных радиоуправляемых машинках. Для того, чтобы машинка повернула направо или налево, она должна получить импульсный сигнал, у которого частота следования импульсов постоянная, а длительность импульсов изменяется. Отношение длительности импульса к длительности периода сигнала называется коэффициентом заполнения. Машинка повернет налево, если коэффициент заполнения менее 50% , или направо, если коэффициент заполнения превышает 50%. Модуль PWM микроконтроллера B позволяет организовать импульсную последовательность с требуемыми значениями периода следования и коэффициента заполнения при использовании минимального числа команд прикладной программы.

Модуль контроллера CAN интерфейса msCAN12 (Motorola Scalable Controller Area Network) содержит в себе набор аппаратных средств для поддержки коммуникационного протокола промышленных сетей в стандарте CAN 2.0 A/B.

Практическая часть

Перед началом выполнения практической части необходимо ознакомиться с лабораторной установкой на базе платы Freescale и программным обеспечением CodeWarrior. Для того чтобы воспользоваться модулем аналого-цифрового преобразования ATD для измерения уровня напряжения на нескольких аналоговых входах МК, необходимо выполнить следующие действия:

1) Подключить источники стабилизированного напряжения ко входам опорного напряжения VRF и VRL . Необходимо помнить, что напряжение на входе высокого уровня опорного напряжения VRF не должно превышать 5,0 В, а на входе низкого уровня VRL напряжение должно быть не менее 0 В. Кроме того, разность напряжений на входах VRF и VRL, равная напряжению полной шкалы АЦП UREF = URH – URL , не должна быть менее 2,5 В;

2) Подключить источники измеряемых аналоговых сигналов ко входам AN0…AN7. Напряжение измеряемых сигналов должно находиться в диапазоне от 0 В до 5,0 В;

3) Осуществить внутреннюю коммутацию напряжения питания к модулю ATD. Для этого записать 1 в бит ADPU регистра управления ATDCTL2. Адрес регистра – $0062;

4) Выдержать паузу в 100 мкс для завершения переходных процессов в модуле ATD. В рассматриваемом ниже программном фрагменте мы покажем, как организовать такую задержку;

5) Назначить режим работы модуля ATD посредством записи необходимых слов инициализации в управляющие регистры модуля;

6) Запустить измерительную последовательность посредством записи в регистр управления ATDCTL5;

7) Контролировать ход преобразования, используя флаги регистра состояния модуля ATDSTAT;

8) Когда измерительная последовательность будет завершена, считать данные из регистров результата ADR0H…ADR7H в память МК.

Программный фрагмент voltmeter.c производит измерение аналогового сигнала на входе AN6. Измерительная последовательность состоит из четырех преобразований. Режим измерения однократный. Результаты четырех последовательных преобразований одного и того же сигнала располагаются в четырех регистрах результата ADR0H…ADR3H. Эти измерения усредняются, что позволяет снизить влияние шумов. Полученный 8 разрядный двоичный код преобразуется к истинному значению измеряемого напряжения, умноженному на 100. Умножение на нормирующий коэффициент (100 в десятичной системе счисления) необходимо, чтобы использовать в программе целочисленные форматы представления данных. Полученный результат содержит одну десятичную цифру целой части измеренного напряжения, это единицы

Вольт. А также две цифры десятичной дробной части. Это десятые и сотые доли Вольт в представлении результата. Промежуточные результаты исполнения программы, а также измеренное напряжение выводятся на экран персонального компьютера.

```
/* В этом примере реализован простейший цифровой вольтметр. */
/* Программа выполняет одно измерение и выводит данные на экран */
/* персонального компьютера. Для того, чтобы выполнить следующее измерение*/
/* следует перезапустить программу */
/*_____ */
/*подключаемые файлы*/
#include<912b32.h>
#include<stdio.h>
#define DECIMAL 0x2E /*определить код точки на экране*/
#define V 0x56 /*определить код символа "V" для экрана*/
/*используемые функции*/
void delay_100us(void);
void ADC_convert(void);
void delay_5ms (void);
void main(void)
{
printf("HELLO\n"); /*вывести приветствие на экран*/
ATDCTL2 = 0x80; /*включить питание модуля, запретить */
/*прерывания от модуля*/
printf ("ADC \n" );
delay_100us(); /*задержка 100мкс*/
printf ("warmed up\n");
ATDCTL3 = 0x00; /*обеспечить доступ к модулю в отладочном режиме*/
ATDCTL4 = 0x0l; /*установить 2 такта для времени выборки*/
/*и коэффициент деления, равный 4*/
printf ( "ready\n" );
ADC_convert(); /*реализовать цифровой вольтметр*/
}
/*_____ */
```

Контрольные вопросы

1) Назначение процессорного ядра микроконтроллеров.
2) Основные аппаратные средства современных МК.
3) Дополнительные аппаратные средства современных МК.
4) Особенности микроконтроллеров на примере МК семейства Motorola.
5) Многофункциональный таймер.
6) Модуль контроллера CAN интерфейса msCAN12 (Motorola Scalable Controller Area Network).
7) Модули контроллеров последовательных интерфейсов SPI (Serial Peripheral Interface) и SCI (Serial Communication Interface).
8) Модуль аналого-цифрового преобразователя ATD (Analog To Digital convertion system).
9) Модуль широтно-импульсного модулятора PWM (Pulse Width Modulation).

ПРАКТИЧЕСКАЯ РАБОТА №39. АППАРАТУРА И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ТИПОВОЙ МИКРОПРОЦЕССОРНОЙ СИСТЕМЫ

Цель работы: определение состава аппаратуры и программного обеспечения типовой микропроцессорной системы.

Теоретические сведения

В настоящее время все вычислительные средства любого уровня строятся с использованием элементной базы одного типа – больших и сверхбольших интегральных схем

(БИС и СБИС). Архитектурные особенности и мощное аппаратное обеспечение ПК типа IBM PC позволяет использовать такие ЭВМ в качестве управляющих или в качестве контроллеров в автоматических системах и под- системах, несмотря на концептуальную неприспособленность ЭВМ с «изолированной шиной» для управления объектами. Высокое быстродействие аппаратуры ПК и использование однозадачной операционной системы, например, MS DOS, обеспечивает малое время реакции всей вычислительной системы на события во внешней среде. Запросы прерываний и их восприятие и обработка являются единственными средствами взаимодействия управляющих ЭВМ и объектов управления во внешней, по отношению к ядру ЭВМ, среде. Малое время реакции на запросы прерываний, формирующихся по событиям во внешней среде, является единственным критерием выбора операционных систем (ОС) для работы в реальном времени (ОС РВ). По этому критерию ни одна из современных многозадачных ОС не может конкурировать с MS DOS. При включении электропитания ПК сначала выполняется программа само- тестирования POST, по результатам работы которой заполняется область дан- ных BIOS в оперативной памяти ЭВМ. Размещение в памяти разнообразных данных, описывающих аппаратуру и программное обеспечение ЭВМ, выполняется стандартным образом для всех поколений ПК.

Практическая часть

Программа, текст которой содержит Листинг 1, копирует в оперативную память содержимое некоторых элементов области данных BIOS и интерпретирует их в форме, доступной для восприятия человеком- оператором. Кроме того, для определения аппаратной конфигурации ПК ис- пользуются некоторые службы BIOS и DOS. Программа обращается к системной области памяти, что для непривилегированных пользователей запрещено в ОС Windows 2000/XP.

Листинг 1

```
#include  <stdio.h>
#include  <dos.h>
#include  <bios.h>
#include  <conio.h>

long a,mem;
unsigned int
b,c,d,i,j,typ_disk,col_disk,max_gol,max_sect,max_cilind;
char e,f,g,h,model;
```

```
void ver()                  //функция определения версии DOS
 {
  union REGS regs;
  regs.h.ah=0x30;           //30 hex служба
  int86(0x21,&regs,&regs);//21 hex прерывание
  e=regs.h.ah;              //Младшая часть версии
  f=regs.h.al;              //Старшая часть версии
 }

void width()                //Функция определения размера
                            //расширенной памяти
 {
  struct REGPACK regx;
  regx.r_ax=0x8800;         //88 hex служба
  intr(0x15,&regx);         //15 hex прерывание
  mem=regx.r_ax;            //размер расширенной памяти в килобайтах
 }

int pardisk(int NUM)
 {
  union REGS regs;
  regs.h.ah=0x8;
  regs.h.dl=NUM;
  int86(0x13,&regs,&regs);
  col_disk=regs.h.dl&0x7f;
  typ_disk=regs.h.bl;
  max_gol=regs.h.dh+1;
  max_sect=regs.h.cl;
  max_cilind=regs.h.ch;
  return 0;
 }
void main(void)
{
 i=20;
 d=1;
 model=peekb(0xf000,0xfffe);
 clrscr();
 gotoxy(i-15,d);
 printf("РЕЗУЛЬТАТЫ РЕВИЗИИ СИСТЕМНЫХ РЕСУРСОВ МИКРОПРОЦЕССОРНОГО"
        " УСТРОЙСТВА");
 d+=2;
 gotoxy(i,d);
 switch(model&0x00ff)
  {
   case 0xff:
       printf("Эта машина - динозаврик PC модели %X",model);break;
   case 0xfb:
   case 0xfe:
       printf("Эта машина - старушка XT модели %X",model);break;
   case 0xfd:
       printf("Эта машина - ужастик PCjr модели %X",model);break;
   case 0xfc:
```

```
        printf("Эта машина - AT модели %X",model);break;
    case 0xfa:
        printf("Эта машина - PS/2 модели %X",model);break;
    case 0xf9:
        printf("Эта машина - PC Convertible модели %X",model);break;
    case 0xf8:
        printf("Эта машина - PS/2 модели 80 %X",model);break;
    default:
        printf("Это вообще не ЭВМ модели %X",model);
  }
 ver();
 d+=1;
 gotoxy(i,d);
 printf("Операционная система: Версия %u.%u",f,e);
 d+=1;
 gotoxy(i,d);
 printf("Дата изготовления ROM BIOS: ");
 for(a=0xfff5;a<=0xfffc;a++)
    printf("%c",peekb(0xf000,a));
b=peek(0x40,0x10);
c=b&0x00c1;
d+=1;
gotoxy(i,d);
switch(c)
 {
  case 0x0000:
          printf("Нет ни единого дискетника");break;
  case 0x0001:
       {
        pardisk(0);
        printf("Установлен один дискетник ");
        switch(typ_disk)
         {
          case 0x01:
               printf("360 кБ, 5.25 дюйма");break;
          case 0x02:
               printf("1.2 МБ, 5.25 дюйма");break;
          case 0x03:
               printf("720 кБ, 3.5 дюйма");break;
          case 0x04:
               printf("1.44 МБ, 3.5 дюйма");
         }
       }break;
  case 0x0041:
       {
        pardisk(0);
        gotoxy(i-3*i/4,d);
        printf("Установлено два дискетника");
        switch(typ_disk)
         {
          case 0x01:
               printf("360 кБ, 5.25 дюйма и ");break;
```

```
         case 0x02:
              printf("1.2 МБ, 5.25 дюйма и ");break;
         case 0x03:
              printf("720 кБ, 3.5 дюйма и ");break;
         case 0x04:
              printf("1.44 МБ, 3.5 дюйма и ");
        }
       pardisk(1);
       switch(typ_disk)
        {
         case 0x01:
              printf("360 кБ, 5.25 дюйма");break;
         case 0x02:
              printf("1.2 МБ, 5.25 дюйма");break;
         case 0x03:
              printf("720 кБ, 3.5 дюйма");break;
         case 0x04:
              printf("1.44 МБ, 3.5 дюйма");
        }
      }break;
case 0x0081:
      {
 pardisk(0);
 gotoxy(1,d);
 printf("Установлено три дискетника");
 switch(typ_disk)
  {
   case 0x01:
        printf("360 кБ, 5.25 дюйма, ");break;
   case 0x02:
        printf("1.2 МБ, 5.25 дюйма, ");break;
   case 0x03:
        printf("720 кБ, 3.5 дюйма, ");break;
   case 0x04:
        printf("1.44 МБ, 3.5 дюйма, ");
  }
 pardisk(1);
 switch(typ_disk)
  {
   case 0x01:
        printf("360 кБ, 5.25 дюйма, ");break;
   case 0x02:
        printf("1.2 МБ, 5.25 дюйма, ");break;
   case 0x03:
        printf("720 кБ, 3.5 дюйма, ");break;
   case 0x04:
        printf("1.44 МБ, 3.5 дюйма, ");
  }
 pardisk(2);
 switch(typ_disk)
  {
   case 0x01:
        printf("360 кБ, 5.25 дюйма, ");break;
```

```
        case 0x02:
             printf("1.2 МБ, 5.25 дюйма, ");break;
        case 0x03:
             printf("720 кБ, 3.5 дюйма, ");break;
        case 0x04:
             printf("1.44 МБ, 3.5 дюйма, ");
       }
     }break;
case 0x00c1:
     {
      pardisk(0);
      gotoxy(1,d);
      printf("Установлено четыре дискетника");
      switch(typ_disk)
       {
        case 0x01:
             printf("360 кБ, 5.25 дюйма, ");break;
        case 0x02:
             printf("1.2 МБ, 5.25 дюйма, ");break;
        case 0x03:
             printf("720 кБ, 3.5 дюйма, ");break;
        case 0x04:
             printf("1.44 МБ, 3.5 дюйма, ");
       }
      pardisk(1);
      switch(typ_disk)
       {
        case 0x01:
             printf("360 кБ, 5.25 дюйма, ");break;
        case 0x02:
             printf("1.2 МБ, 5.25 дюйма, ");break;
        case 0x03:
             printf("720 кБ, 3.5 дюйма, ");break;
        case 0x04:
             printf("1.44 МБ, 3.5 дюйма, ");
  }
 pardisk(2);
 switch(typ_disk)
  {
   case 0x01:
        printf("360 кБ, 5.25 дюйма, ");break;
   case 0x02:
        printf("1.2 МБ, 5.25 дюйма, ");break;
   case 0x03:
        printf("720 кБ, 3.5 дюйма, ");break;
   case 0x04:
        printf("1.44 МБ, 3.5 дюйма, ");
  }
 pardisk(3);
 switch(typ_disk)
  {
   case 0x01:
        printf("360 кБ, 5.25 дюйма, ");break;
```

```
            case 0x02:
                 printf("1.2 МБ, 5.25 дюйма, ");break;
            case 0x03:
                 printf("720 кБ, 3.5 дюйма, ");break;
            case 0x04:
                 printf("1.44 МБ, 3.5 дюйма, ");
           }
         }
 }

pardisk(0x80);
d+=1;
gotoxy(i,d);
printf("Винчестеров %u",col_disk);
switch(col_disk)
 {
  case 0x00:
       printf(". Это очень печально!!!");break;
  case 0x01:
    {
     gotoxy(10,d);
     max_cilind=((max_cilind&0x00ff)|((max_sect&0x00c0)<<2))+1;
     printf("Винчестеров %u шт,типа %u,головок %u,цилиндров %u,"
            " секторов %u",
     col_disk,typ_disk,max_gol,max_cilind,max_sect&0x003f);
     clreol();
    }break;
  case 0x02:
    {
     gotoxy(i,d);
     printf("Винчестеров %u, первый - типа %u, головок %u, ",
             col_disk,typ_disk,max_gol);
     clreol();
     pardisk(0x81);
     d+=1;
     gotoxy(i+4,d);
     printf("второй - типа %u, головок %u",typ_disk,max_gol);
    }break;
  default:
      printf(", а это очень много!!!");
 }
c=b&0xc000;
d+=1;
gotoxy(i,d);
switch(c)
 {
  case 0x0000:
          printf("Нет принтеров");break;
  case 0xc000:
          printf("Установлен один принтер");break;
  case 0x8000:
          printf("Установлено два принтера");break;
  case 0x4000:
```

```
            printf("Установлено три принтера");
 }

c=b&0x1000;
d+=1;
gotoxy(i,d);
switch(c)
 {
  case 0x0000:
          printf("Игровой адаптер не установлен");break;
  case 0x1000:
          printf("Установлен игровой адаптер");
 }

c=b&0x0e00;
d+=1;
gotoxy(i,d);
switch(c)
 {
  case 0x0000:
          printf("Нет COM-портов");break;
  case 0x0200:
          printf("Установлен один COM порт");break;
  case 0x0400:
          printf("Установлено два COM порта");break;
  case 0x0600:
          printf("Установлено три COM порта");break;
  case 0x0800:
          printf("Установлено четыре COM порта");break;
  case 0x0a00:
          printf("Установлено пять COM портов");break;
  case 0x0c00:
          printf("Установлено шесть COM портов");break;
  case 0x0e00:
          printf("Установлено семь COM портов");
 }

c=b&0x0030;
d+=1;
gotoxy(i,d);
switch(c)
 {
  case 0x0000:
      printf("Начальный видеорежим не идентифицирован");break;
  case 0x0010:
      printf("Начальный текстовый видеорежим - цветной на 40"
             " колонок");break;
  case 0x0020:
      printf("Начальный текстовый видеорежим - цветной на 80"
             " колонок");break;
  case 0x0030:
      printf("Начальный текстовый видеорежим - моно на 80"
             " колонок");
```

```
 }
d+=1;
gotoxy(i,d);
printf("Процессор прикладными программами не определяется!");
c=b&0x0002;
d+=1;
gotoxy(i,d);
switch(c)
 {
  case 0x0000:
          printf("Копроцессор не установлен");break;
  case 0x0002:
          printf("Копроцессор установлен");
 }
a=peek(0x40,0x13);
d+=1;
gotoxy(i,d);
printf("Размер основной памяти %lu килобайт",a);
width();
d+=1;
gotoxy(i,d);
if(mem==0)
  printf("Процессор, вероятно, работает в реальном режиме\n\n");
 else
  printf("Размер расширенной памяти %lu килобайт\n\n",mem);
  printf("          ... Всё ли Вам ясно? ... Жми любую...");
bioskey(0);
clrscr();
gotoxy(25,12);
printf("НАДЕЮСЬ, что я Вам нравлюсь!!!\n\n"
    "                                    дМПУ \f\1\4\3");
bioskey(0);
clrscr();
} //Нажмите ENTER !!!
```

1. В интегрированной среде программирования (IDE) BorlandC++ вер.3.1 набрать текст программы, который содержит Листинг 1. По желанию студента текст программы можно модифицировать в соответствии с индивидуальным стилем программирования.

2. В тексте программы операторы подробно прокомментировать.

3. Выполнить трансляцию и отладку программы.

4. Выйти из IDE в ОС и выполнить программу.

5. Зафиксировать результаты выполнения программы для представления в отчёт по лабораторной работе.

Содержание отчёта

Отчёт должен содержать подробно прокомментированный листинг программы ревизии системных ресурсов ПК, описание результатов работы программы и выводы по результатам исследования.

ПРАКТИЧЕСКАЯ РАБОТА №40. СБРОС И ПРЕРЫВАНИЯ МИКРОКОНТРОЛЛЕРОВ В ПРОЦЕССЕ ВЫПОЛНЕНИЯ ПРОГРАММЫ УПРАВЛЕНИЯ

Цель работы: ознакомление с состояниями сброса и прерывания микроконтроллеров в процессе выполнения программы управления.

Общие сведения

В процессе исполнения прикладной программы МК реализует монотонную многократно повторяющуюся последовательность действий:

• Выборку кода команды из памяти программ в регистр команды центрального процессора;

• Дешифрацию кода команды;

• Выборку из памяти следующих байтов команды;

• Исполнение команды;

• Сохранение в памяти результатов исполнения команды.

Если исполняется линейная последовательность команд, то содержимое счетчика PC центрального процессора постоянно увеличивается на 1, обеспечивая выборку из памяти следующих команд прикладной программы. Линейная последовательность исполняемых команд может быть изменена под управлением самой программы, например инструкциями «jmp» или «branch». При этом в счетчик команд под управлением программы будет записано новое число, и начнется исполнение следующего линейного фрагмента программы из другого сегмента памяти программ.

Несмотря на явные различия механизмов формирования следующего за исполнением текущей операции значения счетчика команд PC, в обоих рассмотренных случаях это следующее значение PC определяется ходом вычислительного процесса и предсказывается программистом в ходе написания прикладной программы. В противоположность только что рассмотренному полностью предсказуемому потоку событий, который формируется самой микропроцессорной системой, существует еще поток внешних событий, очередность и моменты возникновения которых не синхронизированы с исполнением центральным процессором тех или иных команд прикладной программы. Однако МК должен реагировать на эти события, для чего необходимо изменить последовательность исполнения операторов программы в произвольный, непредсказуемый с точки зрения устройства управления центральным процессором момент времени.

Такое изменение реализуется принудительной записью нового значения в счетчик команд PC под управлением специальных аппаратных средств микроконтроллера, которые реагируют на внешние события. Включение в работу механизма принудительного изменения текущего значения счетчика команд нельзя считать аварийным состоянием микропроцессорной системы. Это лишь специальное состояние, которое позволяет организовать эффективное распределение ресурса одного центрального процессора для обслуживания нескольких устройств, генерирующих в реальном времени несвязанные между собой внешние события.

По способу обработки микроконтроллером исключения подразделяются на прерывания и сброс. В русскоязычной литературе термин «исключение» обычно не используется, и говорят просто о состоянии прерывания или о состоянии сброса микроконтроллера.

Состояния сброса и прерывания в микроконтроллерах семейства Motorola МК семейства 68HC12/HCS12 обладают мощной системой обработки исключений. По способу реакции микроконтроллера на возмущающие события исключения делятся на прерывание и на сброс.

• МК реализует прерывание или МК находится в состоянии прерывания;

• МК находится в состоянии сброса или в состоянии начального запуска.

Прерывания в свою очередь делятся на маскируемые и немаскируемые. Состояние сброса микроконтроллера Микроконтроллер семейства 68HC12/HCS12 переходит в состояние сброса по внешнему сигналу или при наступлении определенных внутренних событий. В состоянии сброса программный счетчик и часть битов регистра состояния центрального процессора, а также определенные в техническом описании регистры специальных функций периферийных модулей устанавливаются в начальное состояние. Это состояние однозначно определяет аппаратную конфигурацию микроконтроллера, с которого он начнет работу после включения питания. Поэтому второе название состояния сброса – состояние начального запуска. Состояние сброса МК использует также для восстановления работоспособного состояния после обнаружения внутренней аварийной ситуации.

Различают четыре источника событий, которые переводят МК в состояние сброса:

1) Внешний сброс (External reset). Все МК семейства 68HC12/HCS12 имеют специальный вывод корпуса RESET для подачи сигнала внешнего сброса. Активный уровень сигнала – логический 0. Пока на входе RESET удерживается низкий уровень сигнала, МК будет находиться в состоянии сброса. После перевода линии RESET в состояние логической 1 МК перейдет в активный режим работы по истечении задержки, которая составляет 4096 периодов системной магистрали МК.

2) Внутренний сброс по нарастанию напряжения питания (Power on reset – POR). Нарастание напряжения на входе VDD микроконтроллера вызывает состояние сброса. Таким образом реализуется начальный запуск МК с однозначно определенной аппаратной конфигурацией и с известным начальным адресом запускаемой на исполнение программы.

3) Внутренний сброс по сторожевому таймеру (Computer Operating Properly reset – COP). Логика работы сторожевого таймера позволяет микроконтроллеру выявлять перемежающиеся ошибки в исполнении прикладной программы, которые могут возникнуть в результате электромагнитных помех или при колебаниях напряжения питания микропроцессорной системы.

В процессе отладки работа сторожевого таймера запрещена. Работа модуля сторожевого таймера разрешается в конечном варианте прикладной программы, который используется при работе МК в системе. Сторожевой таймер – это счетчик, коэффициент счета которого настраивается пользователем при инициализации системы. Счетчик начинает счет внутренних тактовых импульсов в момент начала исполнения программы. Если счетчик переполнится, то МК перейдет в состояние сброса. Правильно исполняемая прикладная программа, в которой очередность исполнения операторов совпадает с предусмотренной программистом очередностью, должна постоянно сбрасывать сторожевой таймер. Тогда внутреннего сброса от него случаться не будет. Для сброса сторожевого таймера в МК семейства 68HC12/HCS12 необходимо в регистр COPRST записать сначала код $55, а затем код $AA.

При создании конечного кода прикладной программы разработчик должен разместить операции записи приведенной последовательности кодов так, чтобы исполнение программы по любому26 возможному пути обеспечивало бы выполнение команд сброса через меньшие интервалы времени, чем период переполнения сторожевого таймера.

4) Внутренний сброс по отклонению частоты тактовых импульсов МК (Clock Monitor reset). МК переводится в состояние сброса, когда модуль встроенного генератора тактирования обнаруживает выход частоты тактирования за заданные пределы или просто останов системы тактирования. Состояние прерывания микроконтроллера Немаскируемые прерывания. В соответствие со своим названием немаскируемые прерывания не могут быть отключены пользователем. Однако в предыдущем абзаце было упомянуто, что установка бита X в 1 запрещает немаскируемые прерывания. Значение бита X действительно равно 1 в состоянии сброса МК. Однако далее он может быть установлен в 0 под управлением программы инициализации, разрешая тем самым немаскируемые прерывания. Далее этот бит не может быть изменен под управлением программы, и в этом его отличие от бита глобальной маски прерываний I.

Три типа немаскируемых прерываний реализуются в МК 68HC12/HCS12:

1) Прерывание по внешнему запросу XIRQ. Все МК 68HC12/HCS12 имеют вывод внешнего немаскируемого прерывания XIRQ. Активный уровень сигнала для генерации запроса на прерывание – логический 0.

2) Прерывание по несуществующему коду команды. Каждая инструкция языка ассемблер МК имеет собственный код. В МК 68HC12/HCS12 коды операций могут быть однобайтовыми и двухбайтовыми. Но не все теоретически возможные коды использованы для кодирования реальных команд процессорного ядра CPU12. Если на этапе выборки кода команды из памяти произошло считывание несуществующего кода команды, то генерируется запрос на немаскируемое прерывание.

3) Программное прерывание – инструкция SWI. Система команд МК 68HC12/HCS12 имеет инструкцию программного прерывания, которая позволяет перейти к исполнению подпрограммы прерывания из прикладной программы.

Маскируемые прерывания.

1) Прерывание по внешнему запросу IRQ. Все МК 68HC12/HCS12 имеют вывод внешнего маскируемого прерывания IRQ. Активный уровень сигнала для генерации запроса на прерывание – логический 0. В некоторых приложениях требуется принять запросы от нескольких внешних источников сигналов. Для таких случаев следует использовать дополнительный логический элемент, который объединяет запросы от всех источников по логике ИЛИ Если запрос на вход IRQ поступил, и МК перешел к выполнению подпрограммы прерывания, то в этой подпрограмме следует опросить линии порта для того, чтобы установить, какой из источников вызвал прерывание. Обработка нескольких объединенных по ИЛИ запросов с программным поиском установившего запрос источника называется поллингом.

2) Прерывание по таймеру меток реального времени RTI. Таймер меток реального времени генерирует последовательность равноотстоящих во времени запросов на прерывание. Период повторения запросов настраивается программистом. Эти прерывания могут быть использованы для регулярного выполнения микроконтроллером некоторой задачи. Например, для измерения напряжения аккумуляторной батареи каждые три мин, чтобы сигнализировать о необходимости ее замены. Мы рассмотрим особенности прерываний RTI в главе 7 на примере управления скоростью вращения электрическим двигателем.

3) Прерывание по событию канала захвата/сравнения (IC/OC) таймера. Восемь одинаковых блоков в составе модуля таймера, которые именуют «каналами», 27 предназначены для контроля за уровнем сигнала на входе канала или для изменения в строго определенный момент времени логического уровня на выходе канала. Заданное программистом изменение входного или выходного сигнала канала рассматривается как событие, которое генерирует запрос на прерывание. Например, если канал настроен на слежение за перепадом входного сигнала из 1 в 0, то когда такое изменение произойдет, будет выставлен запрос на прерывание.

4) Прерывание по переполнению таймера. Основным блоком модуля таймера является 16_разрядный счетчик временной базы. Этот счетчик невозможно остановить. Также невозможно изменить его коэффициент счета, который составляет 216 = 65536. Поэтому регулярно счетчик временной базы изменяет свой код с $FFFF на $0000. Такое изменение кода называют переполнением счетчика. В момент переполнения по желанию программиста может генерироваться запрос на прерывание, в то время как счетчик продолжает считать дальше. Такие прерывания особенно удобны при необходимости измерения очень больших временных интервалов. Для этого в микроконтроллере производят подсчет, сколько переполнений счетчика произошло за этот временной интервал, и, зная период счета счетчика, определяют длительность исследуемого временного интервала.

5) Прерывание по переполнению счетчика внешних событий. Когда счетчик внешних событий переполняется, то может генерироваться запрос на прерывание точно так же, как и для счетчика временной базы.

6) Прерывание по событию на входе счетчика внешних событий. Этот запрос на прерывание формируется, если сигнал на входе счетчика внешних событий изменил свое значение. Характер изменения, т.е. перепад из 0 в 1, или из 1 в 0, или любое изменение логического уровня, определяется программистом.

7) Прерывание от модулей контроллеров последовательного ввода/вывода SCI и SPI. Каждый модуль последовательного ввода/вывода формирует целую группу запросов на прерывание: при завершении передачи слова, при приеме слова, при обнаружении различного рода нарушений в протоколе передачи информации.

8) Прерывание от модуля АЦП. Модуль аналого-цифрового преобразователя формирует запрос на прерывание, когда процесс оцифровки очередного сигнала завершен, и двоичный код сигнала может быть считан в память МК.

9) Прерывание при выходе МК из энергосберегающих режимов. Это прерывание позволяет вывести МК из состояния STOP или WAIT, в котором он находился с целью снижения потребляемой энергии. Такие прерывания очень полезны при объединении нескольких МК в информационную сеть.

Практическая часть

Перед началом выполнения практической части необходимо ознакомиться с лабораторной установкой на базе платы Freescale и программным обеспечением CodeWarrior. В данной практической работе мы рассмотрим основные особенности формирования исходного текста программы с прерываниями на Си. При программировании на Си Вы должны обязательно реализовать следующие этапы записи исходного текста:

1. При написании программы обработки прерывания на Си, имя подпрограммы обработки прерывания должно быть объявлено с использованием специальной директивы препроцессора. #pragma interrupt_handler 28 В поле следует записать имя подпрограммы прерывания, которое Вы будете далее использовать в тексте программы. Приведенная запись информирует компилятор о том, что функция с названным именем является подпрограммой прерывания.

2. Далее по тексту подпрограмма прерывания оформляется как обычная функция. Компилятор в процессе перевода исходного текста этой функции на Си в инструкции ассемблера автоматически подставит в конце подпрограммы команду возврата из прерывания RTI, потому что эта функция была объявлена подпрограммой прерывания (директива #pragma на этапе 1).

3. Для правильного функционирования МК в процессе прерывания необходимо инициализировать указатель стека. Его значение должно быть равно старшему адресу области оперативной памяти МК, увеличенному на единицу. Поскольку диапазоны памяти пользователя (как постоянной, так и оперативной) являются необходимыми установками в конфигурации компилятора, то функция инициализации указателя стека выполняется компилятором автоматически. Поэтому программист не должен записывать какой либо текст в программе для инициализации указателя стека. Зато следует проверить карту памяти в установках компилятора, которая обязательно должна совпадать с реальной проектируемой системой.

4. Подсистема прерывания будет функционировать корректно, если для нее сформирована таблица векторов прерывания. Мы уже обсуждали, что таблица векторов прерывания в МК B32 находится в области Flash памяти, которая защищена от перезаписи информации. Для того, чтобы пользователь имел возможность записать собственную таблицу векторов сброса и прерывания, в эту нестираемую область памяти записаны фиксированные вектора, которые передают управление по известным адресам в области перезаписываемой EEPROM памяти. По этим адресам программист должен вписать команду безусловного перехода JMP с адресом соответствующей подпрограммы обработки прерывания.

5. Каждый маскируемый источник запроса на прерывание должен быть разрешен установкой соответствующего бита в регистре управления периферийного модуля. Мы рассмотрим, как это записать на Си в последующих примерах.

6. После установки всех индивидуальных битов на разрешение прерывания, необходимо сбросить глобальную маску прерывания I. На ассемблере для этого используют команду CLI. При программировании на Си мы также воспользуемся этой командой, посредством следующих макросов:

```
#define CLI ( ) asm(«cli\n»); //разрешить маскируемые прерывания
#define SEI ( ) asm(«sei\n»); //запретить маскируемые прерывания
```

Далее по тексту программы, если необходимо разрешить прерывания, то следует ввести CLI ( ).

```
//объявление функции в модуле
void toggle_isr(void);
//директива #pragma для указания, что функция является подпрограммой
//обслуживания прерывания
#pragma interrupt_handler toggle_isr
//инициализация соответствующего вектора в таблице векторов прерываний
#pragma abs_address: 0xF7EA //B32 RAM_based vector address
void (*Timer_Channel_2_interrupt_vector[]) ()={toggle_isr};
#pragma end_abs_address
```

Контрольные вопросы

1) Последовательность действий МК при выполнении исполняемой программы.
2) Состояния сброса МК.
3) Состояния прерывания МК.
4) Маскируемые и немаскируемые прерывания.
5) Вектора исключений.
6) Система приоритета для исключений.
7) Регистры подсистемы прерывания.
8) Процесс перехода к системе прерывания.
9) Программная реализация систем сброса.
10) Программная реализация систем прерывания.

ПРАКТИЧЕСКАЯ РАБОТА №41. ТЕСТИРОВАНИЕ И ОТЛАДКА МИКРОПРОЦЕССОРНЫХ СИСТЕМ.

Цель работы: изучение аппаратных и программных средств микроконтроллера, ориентированных на обработку битовой информации; получение навыков работы с пакетом программных средств PK51–Eval, предназначенных для разработки и отладки программ микроконтроллера; ознакомление с принципами реализации микропроцессорной системы на основе универсального лабораторного стенда (УЛС); получение навыков работы с управляющей программой MCS51 для отладки микропроцессорной системы в составе УЛС.

Теоретическая часть

Важной отличительной чертой архитектуры микроконтроллеров семейства MCS-51 является мощная поддержка обработки одноразрядных данных. Тогда как поддержка простых типов данных при существующей тенденции к увеличению длины слова может, с первого взгляда, показаться шагом назад, это качество делает MCS-51 особенно удобными там, где наиболее оправданно применение однокристальных микроконтроллеров, т.е. в системах управления.

Алгоритмы работы последних по своей сути предполагают наличие входных булевых переменных, преобразуемых в выходные битовые сигналы. Такую обработку сложно проводить с помощью БИС универсальных микропроцессоров или однокристальных микроконтроллеров, не имеющих соответствующих программно-аппаратных средств. В микроконтроллерах семейства MCS-51 такая поддержка обеспечивается как на аппаратном, так и на программном уровнях.

Аппаратная поддержка включает в себя:

- АЛУ, допускающее обработку битовой информации;
- специальный битовый аккумулятор, входящий в состав АЛУ;
- память данных с побитовой адресацией;
- возможность адресации отдельных бит некоторых специальных регистров;

- индивидуальную поразрядную настройку линий портов ввода-вывода на ввод или вывод информации.

Система команд микроконтроллера позволяет активно манипулировать одноразрядными данными. Отдельные программно-доступные биты могут быть установлены, сброшены или проинвертированы, могут пересылаться и использоваться в логических вычислениях. Важной особенностью системы команд MCS-51, чрезвычайно эффективной в алгоритмах управления, является возможность в одной команде проанализировать состояние какой-либо битовой переменной (например, отдельной линии порта ввода-вывода) и выполнить переход в зависимости от результата этого анализа.

Все эти свойства в целом позволяют говорить об отдельном булевом процессоре, встроенном в состав микроконтроллеров семейства MCS-51.

Для выполнения практической работы необходимо ознакомиться с архитектурой микроконтроллера (однокристальной микроЭВМ) МК-51 (MCS-51) и изучить его систему команд.

Практикум выполняется на современной версии микроконтроллера (МК PCF80C552 семейства MCS-51), которая входит в состав микропроцессорной системы, реализованной на УЛС.

Постановка задачи и варианты ее решения

В практической работе задание предполагает разработку микропроцессорной системы на базе МК семейства MCS-51, ориентированного на обработку битовой информации.

МК получает от внешнего устройства считывания и согласования уровней показания трёх битовых датчиков, обрабатывает их в соответствии с заданной логической функцией и формирует управляющее воздействие, являющееся значением вычисленной логической функции. Значения входных переменных и соответствующее им значение функции записываются также во внутреннюю память данных МК.

В данной микропроцессорной системе работа МК и внешнего устройства никак не синхронизирована, каждое их них работает по своей программе. Для получения верного результата они (в этом частном случае) работают в режиме handshaking – синхронизируют свою работу при помощи двух осведомительных сигналов – «Разрешение» и «Подтверждение». МК может считывать значения входных логических переменных только поле того, как внешнее устройство установит их истинные значения. При этом внешнее устройство задает активный уровень сигнала «Разрешение». Когда МК дождётся установки этого уровня, он считывает значения логических переменных и оповещает об этом внешнее устройство установкой активного уровня сигнала «Подтверждение». Внешнее устройство после этого изменяет значение сигнала «Разрешение» – устанавливает пассивный уровень – и может готовиться к приёму результата. МК может выдать на выходе своего порта результат, но внешнее устройство может не заметить факт выдачи результата – новое значение может быть равно предыдущему. Единственная возможность у МК сообщить об этом – изменить уровень сигнала «Подтверждение».

В общем случае у МК для выполнения задачи обмена с внешними устройствами можно использовать любые порты. Выберем для примера порт P1, который имеется у всех микроконтроллеров семейства MCS-51.

Каждый из сигналов от датчиков поступает на МК по определенной входной линии: X – по P1.0, Y – по P1.1, Z – по P1.2. Сигнал разрешения чтения показаний датчиков поступает по линии P1.3, а сигнал подтверждения приема МК информации выдается по линии P1.6. Результат выводится по линии P1.7.

После завершения цикла работы управление передается на обработку очередных показаний, считываемых с датчиков.

Структурная схема микропроцессорного устройства приведена на рис. 26.1. Микроконтроллер представлен на рисунке лишь своим портом P1, распределение линий которого существенно для данной работы.

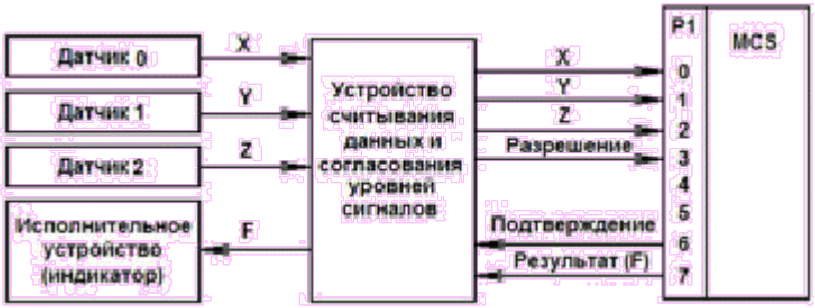


Рис. 26.1. Структурная схема микропроцессорной системы обработки битовой информации

Индивидуальные варианты заданий на выполнение практической работы различаются видом логической функции, вычисляемой МК по показаниям от датчиков, а также уровнями сигналов, которыми обмениваются микроконтроллер и внешнее устройство.

Для подготовки к выполнению собственно практической работы необходимо:

- изучить пример подготовки работы;
- составить в среде разработки µVision программу на языке Ассемблер МК с учетом реализации микропроцессорного устройства на универсальном лабораторном стенде;
- составить таблицу истинности логической функции, реализуемой МК.

Практическая часть

Порядок выполнения практической работы:

- отладить программу на всех наборах логических переменных с использованием программного симулятора/отладчика dScope;
- продемонстрировать работу отлаженной программы преподавателю;
- создать в системе Xilinx проект, обеспечивающий подключение источников и приемников битовых сигналов к контактам порта МК.

Пример подготовки к выполнению практической работы

Рассмотрим вариант задания, в котором МК вычисляет простую логическую функцию F= Z&Y&X. Разрешающий сигнал достоверности входных данных имеет высокий уровень. Микроконтроллер должен вырабатывать сигнал подтверждения приёма информации также высокого уровня. В программе должна быть реализована запись вычисленного значения функции вместе с переменными во внутреннюю память данных, начиная с ячейки, адрес которой находится в регистре R0.

Разработаем требуемую программу. Её текст с необходимыми комментариями приведен ниже. При написании программы учитывается, что пользовательская программа, загруженная в эмулятор ПЗУ МК, будет отображаться в памяти программ, начиная с адреса 8000h.

Программа

ORG 8000h

Эта макрокоманда Ассемблера указывает, что команда, расположенная после неё в тексте, будет записана в памяти программ, начиная с адреса 8000h.

BEGIN: MOV P1,#00111111b

Этой командой задаётся начальное значение порта P1: в разряды P1.0-P1.3 записываются «1» потому, что эти разряды порта предназначены для приёма входных сигналов и в соответствующих разрядах выходного «регистра-защелки» этого порта должны быть «1» – настройка на ввод. Линии P1.4 и P1.5 микроконтроллера не используются и они остаются в состоянии «1» («1» – это исходное состояние «регистров-защёлок» портов МК при подаче сигнала сброса). P1.6 = 0 – для данного варианта задания это пассивное значение сигнала «Подтверждение». P1.7 = 0 – исходное значение выходного сигнала результата (оно в начальный момент произвольное и может быть и равно «1»).

```
m       JNB P1.3, $ ; ожидание сигнала разрешения
```

```
metka:
        MOV C,P1.2 ; C=Z
        ANL C,P1.1 ; C=Z&Y
        ANL C,P1.0 ; C=Z&Y&X
        SETB P1.6 ; выдача сигнала подтверждения конца ввода
        переменных
        JB P1.3,$ ; ожидание снятия сигнала разрешения
        MOV P1.7,C ; выдача результата
        ANL A, #0Fh; в аккумуляторе выделяются значения
        входных переменных
        RLC A ; циклический сдвиг аккумулятора влево через C,
        при этом значение C записывается в младший разряд A
        MOV @R0, A ; запись цифры в массив по адресу,
        находящемуся в регистре R0
        INC R0
        CLR P1.6 ; сброс сигнала подтверждения
        AJMP metka ; переход к вычислению
        нового значения функции
        END
```

Примечание. В дальнейшем с учетом реализации микропроцессорного устройства на УЛС, программа будет модифицироваться.

Для подготовки к отладке программы составим таблицу истинности заданной функции (табл.26.1). Порядок переменных в столбцах таблицы может быть любым, но желательно, чтобы расположение повторяло расположение переменных на входах МК (ZYX).

Таблица 26.1

Таблица истинности

P1.2=Z	P1.1=Y	P1.0=X	P1.7=F(X,Y,Z)
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Рекомендации по разработке и отладке программы на программно-логической модели

Программа реализации задания может быть оформлена в текстовом виде или непосредственно создана (написана) в среде KeilPK51 –Eval. В состав этого пакета входят интегрированная среда разработки µVision и программный симулятор-отладчик dScope. Среда разработки программы для микроконтроллера µVision вызывается кликом на пиктограмме (рис. 26.2) на рабочем столе.

Рис. 26.2. Пиктограмма среды разработки µVision

Далее обратим внимание только на отдельные этапы, касающиеся данной практической работы.

На начальном этапе создаётся новый проект, для чего необходимо в меню-вкладке Project выбрать New Project (рис.26.3).

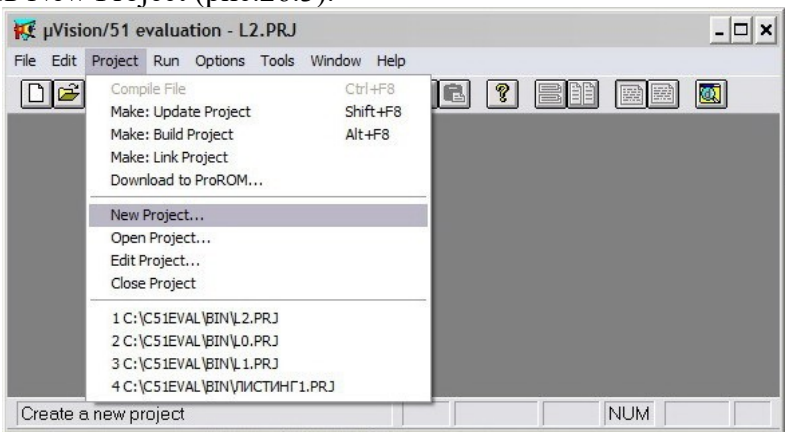


Рис. 26.3. Вкладка Project, создание нового проекта

После этого откроется меню создания нового проекта (рис. 26.4).

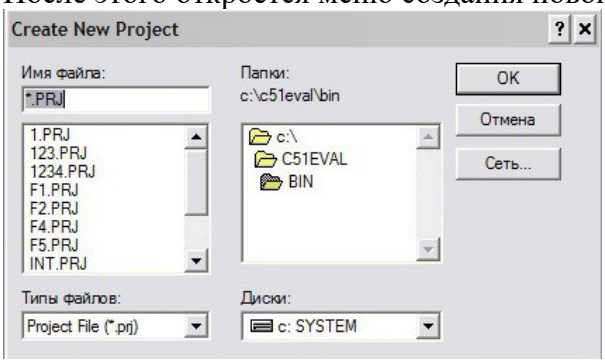


Рис. 26.4. Меню создания проекта Create New Project

В поле «Имя файла» вместо знака «*» нужно указать название проекта, не превышающее восьми символов, например LAB1 (но рекомендуется указать не более семи первых букв фамилии и цифру 1 на конце).

Проект должен располагаться в папке C:\C51EVAL\BIN.

Диск C – не сетевой, поэтому при выполнении работы в лаборатории при смене компьютера составленный проект придётся заново восстанавливать.

После заполнения всех полей и нажатия кнопки «OK», откроется меню настройки проекта. На начальном этапе нет необходимости менять значения установок данного меню. Для закрытия следует нажать «Cancel». Далее во вкладке «File» нужно открыть новый файл «New», в результате появится окно, в котором записывается текст программы (рис. 26.5).

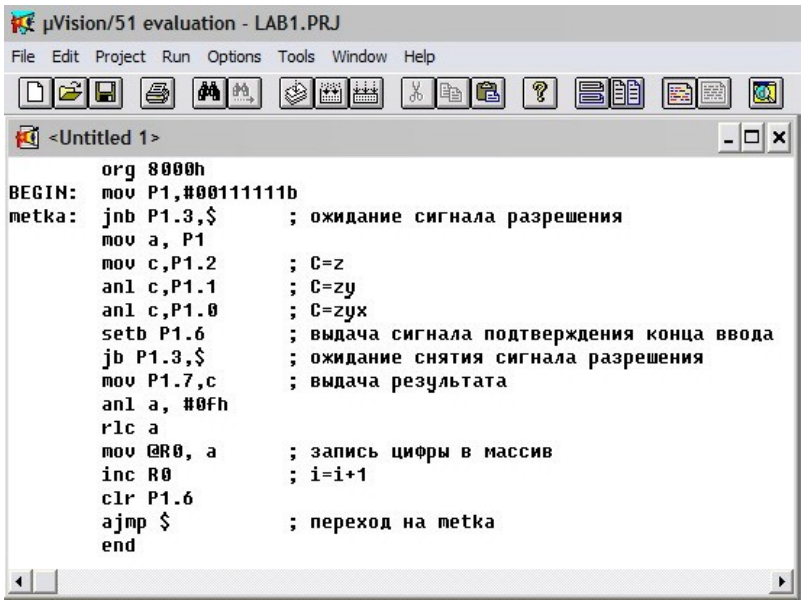


Рис. 26.5. Пример формирования программного кода

Созданному файлу необходимо присвоить имя, при этом имена ранее созданного проекта и файла должны совпадать. После выбора вкладки «File» и далее «Save As» в окне «Сохранить как» для указания типа файла нужно выбрать «Assembly Source (*.a51)». Далее с оставшимися полями проводим действия, как и при создании проекта.

Файл должен быть расположен в той же папке, где расположен проект, а именно в папке C:\C51EVAL\BIN.

Для завершения формирования проекта необходимо в меню настройки проекта добавить созданный файл. Для этого во вкладке «Project» выбираем «Edit Project» (рис. 26.6).

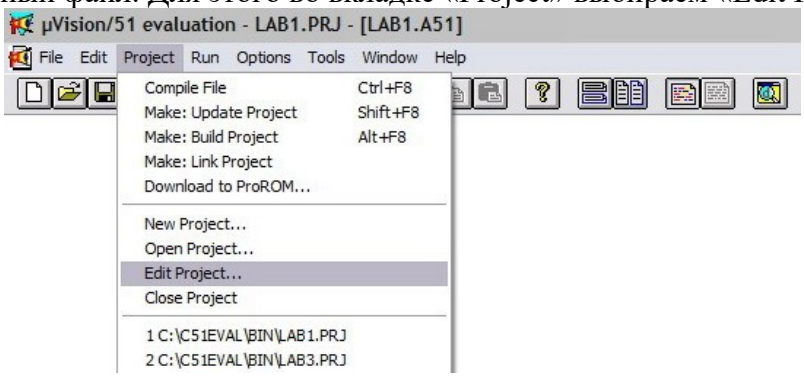


Рис. 26.6. Вкладка «Project», настройка проекта

Далее в меню настройки проекта нажимаем «Add» и находим созданный файл. Выбранный файл отображается в поле «Source Files». Для сохранения настроек нажимаем «Save».

На следующем этапе целесообразно проверить синтаксис написанной программы, для этого во вкладке «Project» выбираем «Compile File».

В случае успешной компиляции система выдаёт сообщение об этом. В случае ошибки система выдаёт сообщение о неудачной компиляции и подсвечивает место ошибки в программном коде. Также система даёт краткую расшифровку ошибки (рис. 26.7).

На рисунке показан вариант, когда в программе вместо P1.2 ошибочно использовано обозначение P4.2 Использование символических имен портов удобно при написании программ, но в имеющемся в настоящее время программном обеспечении поддерживаются только символические имена портов, имеющихся в базовой конфигурации МК (порты P0-P3).

Проверку синтаксиса рекомендуется проводить не только в конце, но и после ввода отдельных фрагментов текста программы.

До исправления ошибки дальнейшие действия не выполняются.

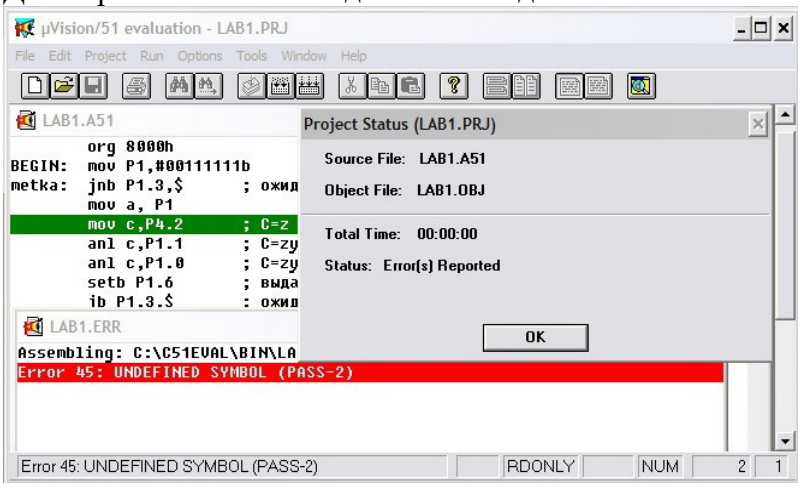


Рис. 26.7. Сообщение об ошибке компиляции программы

В случае успешной компиляции во вкладке «Project» нужно выбрать «Build Project», в результате чего должно быть получено сообщение системы об успешном создании файла *.hex (в данном случае LAB1.hex).

Далее отладка программы происходит в симуляторе dScope, предназначенном для отладки пользовательских программ на моделях различных микроконтроллеров. Окно симулятора вызывается нажатием на пиктограмму «Debug» (крайняя справа).

Пользовательский интерфейс симулятора изображен на рис. 26.8.

Окно отладки (debug window) отображает код загруженной программы с указанием адреса ячейки памяти, где расположено первое слово команды. В правой части формы находится панель, нажатие на кнопки которой приводит к вызову соответствующей функции отладки. Отметим только функции, необходимые для отладки программы в лабораторной работе:

Go – выполнение программы, начиная с текущего адреса счётчика команд;

GoTilCurs – выполнение программы, начиная с текущего адреса счётчика команд. Выполнение заканчивается при достижении команды, содержащейся в строке, на которую указывает курсор;

StepInto – выполнение следующей команды (если встречается команда вызова функции, то происходит вход в тело функции);

Stop – прекращение выполнения программы.

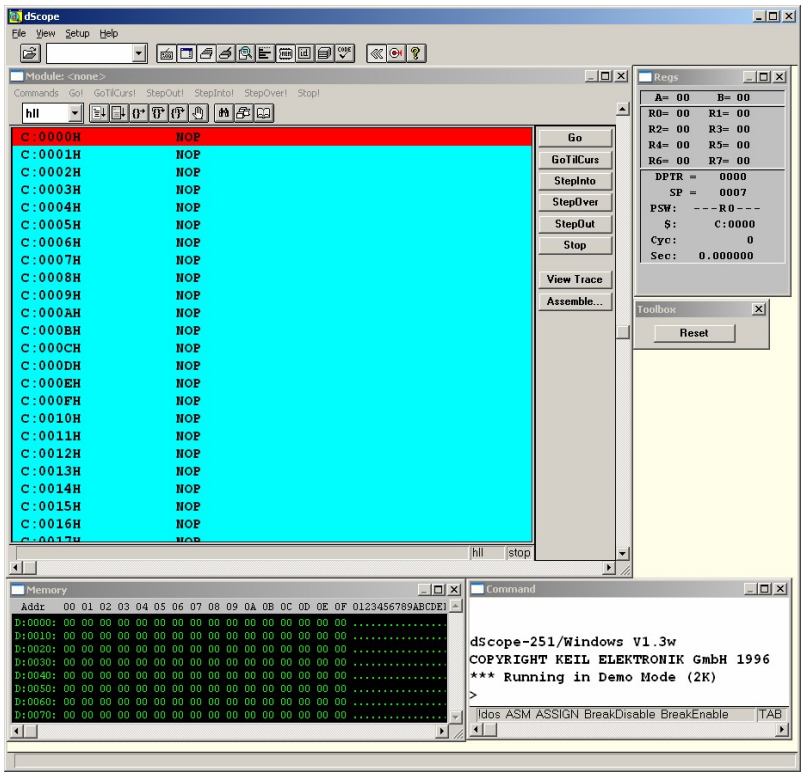


Рис. 26.8. Пользовательский интерфейс симулятора dScope

Окно памяти (Memory) может отображать как память программ, так и память данных МК. Переключение режимов осуществляется путём выполнения специальных команд. Для ввода команд предусмотрено специальное окно (Command).

Окно регистров (Regs) отображает содержимое аккумулятора (A), вспомогательного регистра (B), восьми регистров общего назначения R7-R0. Последние относятся к тому банку регистров, номер которого задан в разрядах 4 и 3 слова состояния процессора (PSW). На экране в регистре слова состояния в этих разрядах цифра справа от буквыRкак раз и означает номер используемого в данный момент банка регистров. При выполнении битовых операций важно контролировать состояние битового аккумулятора C, который является ничем иным как разрядом переноса в слове состояния (PSW.7). При нулевом состоянии этого разряда отображается символ «-», при единичном – символ «C».

В окне отображаются также состояния некоторых регистров специальных функций, выводятся значение счётчика команд, число выполненных циклов и общее время выполнения программы.

Для обращения к периферийным устройствам МК в процессе отладки необходимо настроить dScope на работу с микроконтроллерами данного вида. Для этого в основном окне симулятора необходимо выбрать драйвер 80552.dll для установления соответствия микроконтроллеру, используемому в УЛС, после чего на верхней панели основного окна появляется дополнительное меню Peripherals.

Для отладки программы необходимо в меню «File» выбрать «Loadobjectfile». В поле «Типы файлов» необходимо выбрать «Hex file (*.hex)», указав диск и папку, содержащую файл (C:\C51EVAL\BIN), в поле «Имя файла» необходимо выбрать файл из списка (в нашем примере –LAB1.HEX).

Обратим внимание на то, что код программы будет в отладчике располагаться с адреса 8000h, а в окне данные начинаются с адреса 0000h. Для того чтобы перейти на первую команду программы, достаточно один раз кликнуть левой кнопкой мыши на бегунок в правой части окна. Потом необходимо нажать кнопку GoTilCurs. Таким образом, сразу выполняются все команды, расположенные в памяти программ с адреса 0000h по 7FFFh. По умолчанию в начальный момент вся эта область заполнена командами NOP. Далее можно работать по шагам с помощью команды StepInto.

Пояснения. При отладке программы на программно-логической модели в окне отладки символические имена портов P3-P0 сохраняются, а номера разрядов портов отображаются иначе. В состав МК PCF80C552 входят шесть 8-разрядных портов ввода-вывода (P5-P0). Каждый разряд всех портов, кроме P5, для возможности организации двунаправленного обмена в своем составе имеет защёлку, входной буфер и выходной драйвер. Табл.26.2.иллюстрирует назначение разрядов порта P1 (используемого в рассматриваемой программе) микроконтроллера с учетом их альтернативных функций.

Таблица 26.2

Назначение разрядов порта p1

Вывод порта	Альтернативная функция
P1.0	CT0I С-вход регистра-защелки 0
P1.1	CT1I С-вход регистра-защелки 1
P1.2	CT2I С-вход регистра защелки 2
P1.3	CT3I С-вход регистра-защелки 3
P1.4	T2 Вход таймера T2
P1.5	RT2 Сброс таймера T2
P1.6	SCL Линия синхронизации I2C
P1.7	SDA Линия данных I2C

Именно в соответствии с приведённой таблицей в коде программы, загруженной в симулятор dScope, обозначения P1.i заменены на соответствующие CTiI и обозначения (0x90.i). К обозначению порта P1 добавляется адрес этого порта – (0x90) (рис. 26.9).

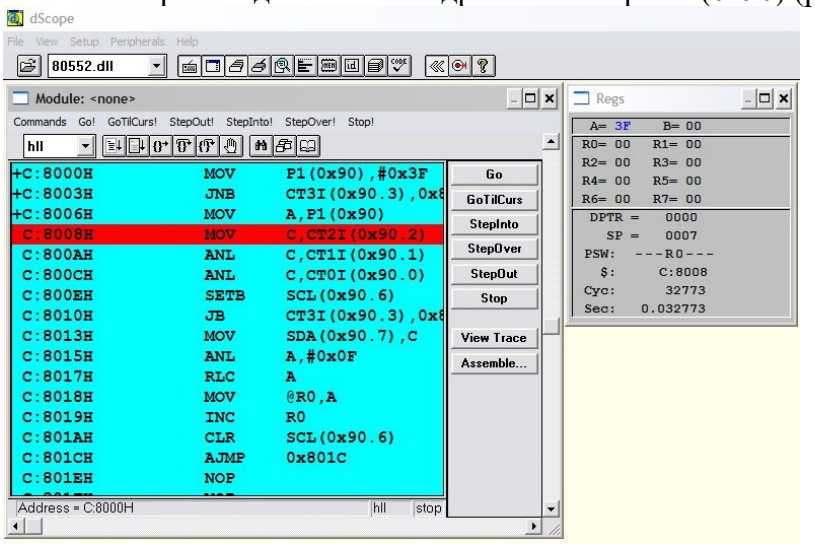


Рис. 26.9. Окно отладки программы, запись в аккумулятор

Выполняя программу по шагам, не только можно, но и нужно отслеживать изменение регистров и памяти. Так, результатом выполнения команды MOV A, P1, расположенной по адресу 8006H, является запись кода 00111111b(или 3Fh), находящегося в регистре P1, в регистр аккумулятора (см. рис. 1.9). Выполненные команды отмечаются знаком «+» в левой части строки. Красным цветом выделяется следующая по порядку выполнения команда.

Ход выполнения программы в МК зависит от значений сигналов от внешнего устройства. Для имитации изменения состояний сигналов от периферийного устройства на внешних контактах порта МК нужно во вкладке меню «Peripherals» выбрать «I/O-Ports», затем «Port1».

В исходном состоянии (до начала выполнения программы) в регистре порта P1 находятся одни «1», что соответствует настройке «по умолчанию» всех разрядов порта на ввод (рис. 26.10). После выполнения первой команды, расположенной по адресу 8000H, в окне P1 отображается

записанное в него значение 00111111, а после команды выдачи сигнала подтверждения ввода по адресу 800EH отображается код 01111111 (рис. 26.11).

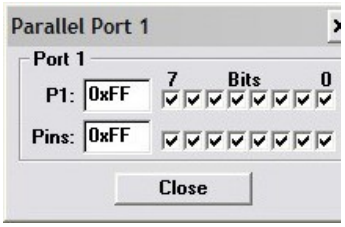


Рис. 26.10. Исходное состояние порта P1

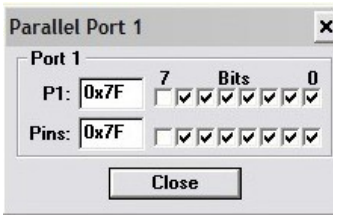


Рис. 26.11. Загрузка кода в порт P1

На рис. 26.12 показано окно кода программы в ситуации, когда после исполнения команды, расположенной по адресу 8010H, микроконтроллер ждёт снятия сигнала разрешения, т.е. установки нулевого значения на контакте P1.3.

Для изменения содержимого порта можно либо левой кнопкой мыши установить в нижней строке нужное (в данном случае нулевое) значение на соответствующем контакте порта, либо в окошке «Pins:» записать новое значение (рис. 26.13).

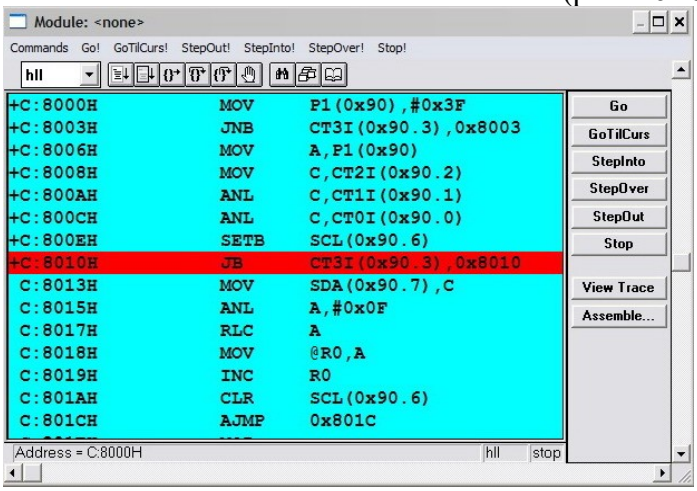


Рис. 26.12. Окно кода программы, ожидание снятия сигнала разрешения

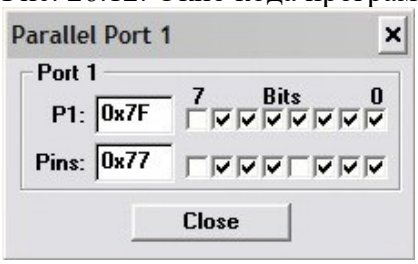


Рис. 26.13. Установка кода «0» на контакте 3 порта P1

Следует помнить о том, что задавать внешние сигналы для имитации работы внешнего устройства, подающего сигналы на порт, в верхней строке нельзя – значения регистра порта изменяются программой.

Теперь о том, как составить программу для того, чтобы её можно было выполнять на МК в составе УЛС.

Несколько усложним задание. Значения результатов вычисления функции от трёх переменных должны засылаться во внутреннюю память данных, начиная с адреса, находящегося в регистре R0 (илиR1). Эти регистры имеются во всех четырёх банках данных. Номер банка определяется значением битов 4 и 3 (S1 иS0) в слове состояния PSW. Программа должна выполняться на МК, входящим в состав УЛС. Внешнее устройство имитируется элементами, расположенными в ПЛИС и на стенде. Задать входные сигналы на стенде можно на регистрах 1-3 (12 сигналов), а для индикации имеется 16 светодиодов на индикаторах A, B, C, D, E и F.

Подключение этих элементов к некоторой схеме на ПЛИС выполняется с помощью макроэлементов, входящих в библиотеку Maket. Но в составе стенда имеется уже законченная микропроцессорная система с подключенными к микроконтроллеру различными устройствами. Из портов для связи с ПЛИС и через неё с регистрами и индикаторами стенда может использоваться только порт P4.

Для порта P4 можно сохранить выбранное ранее распределение разрядов для порта P1. Данные вычислений должны записываться в память по адресу, находящемуся в регистре R0 (или R1) заданного банка регистров. Номер банка, с которым нужно будет работать, будет задаваться на входах P4.5, P4.4 (рис. 26.14).

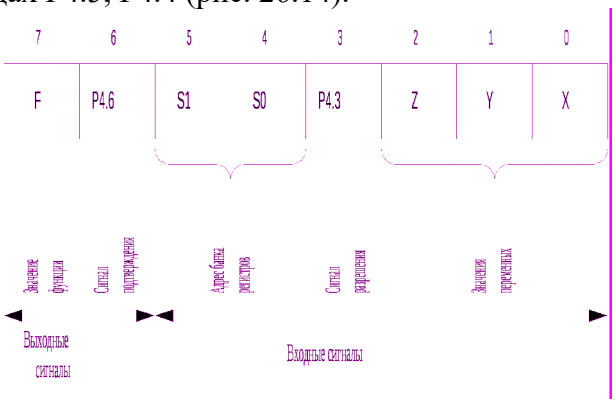


Рис. 26.14. Распределение разрядов порта P4 между входными и выходными сигналами

Чтобы видеть результат выполнения программы – запись в то или иное место памяти в зависимости от того, с каким банком работает программа, нужно в регистры R0 (или R1), относящиеся к разным банкам, записать разные начальные значения. Однако симулятор dScope не позволяет редактировать содержимое внутренней памяти данных. Потому необходимо в начале программы добавить команды начальной загрузки ячеек памяти, которые являются одновременно и регистрами R0 разных банков. РегистрR0 бан- ка 0 – это ячейка памяти с адресом 00h, регистрR0 банка 1 – ячейка с адресом 08h, регистрR0 банка 2 – 10h, регистрR0 банка 3 – 18h.

В имеющемся в настоящее время программном обеспечении поддерживаются только символические имена портов, имеющихся в базовой конфигурации МК (порты P0-P3). Для обращения к порту P4 необходимо использовать его адрес, а именно 0C0h. При этом адрес отдельного контакта этого порта, например первого, будет иметь вид 0C0h.1. С учётом сказанного программа будет выглядеть следующим образом (рис. 25.15).

F5.A51

```
        org 8000h
BEGIN:  mov 00h, #60d
        mov 08h, #20d
        mov 10h, #40d
        mov 18h, #50d
        mov 0C0h, #00111111b
metka:  jnb 0C0h.3,$
        mov a, 0C0h
        mov c, 0C0h.5
        mov psw.4, c
        mov c, 0C0h.4
        mov psw.3, c
        mov c,0C0h.2    ;C=z
        anl c,0C0h.1    ;C=zy
        anl c,0C0h.0    ;C=zyx
        setb 0C0h.6
        jb 0C0h.3,$
        mov 0c0h.7,c
        anl a, #0fh
        rlc a
        mov @R0, a
        inc R0
        ;clr 0C0h.4
        ajmp metka
        end
```

Рис. 26.15. Текст программы

В приведённом тексте программы нет комментариев, но и без этого нетрудно понять, каким образом номер банка регистров с входных контактов порта P4 передаётся в слово состояния. Возможен и иной вариант организации пересылки. Для банка 0 начальное содержимое регистра R0 следует задавать не 3d, а, например, 60d. Это вызвано тем, что ячейки памяти данных, начиная с адреса 08h, отводятся по стек.

Именно в таком виде и должна быть оформлена программа при подготовке к практической работе.

Программу следует сначала отладить на модели. При отладке необходимо будет отслеживать содержимое памяти. При демонстрации отлаженной программы следует показать преподавателю результаты выполнения каждой команды по показаниям состояний различных регистров, показать запись в последовательные ячейки памяти при выбранном банке регистров.

ПРАКТИЧЕСКАЯ РАБОТА №42. ПОЛУЧЕНИЕ НАВЫКОВ РАБОТЫ С УПРАВЛЯЮЩЕЙ ПРОГРАММОЙ ДЛЯ ОТЛАДКИ МИКРОПРОЦЕССОРНОЙ СИСТЕМЫ

Цель работы: ознакомление с принципами реализации микропроцессорной системы на основе универсального лабораторного стенда (УЛС); получение навыков работы с управляющей программой MCS51 для отладки микропроцессорной системы в составе УЛС.

Практическая часть

Порядок отладки микропроцессорной системы на стенде

Для отладки программы и проверки работы реального МК в составе УЛС следует создать в системе Xilinx проект соединения регистров и индикаторов на стенде с контактами порта P4 через контакты ПЛИС.

Выходы порта P4 соединяются с контактами ПЛИС, причем с теми же контактами, через которые могут выводиться сигналы на индикаторы A и B (табл. 27.1).

Соотношение контактов порта p4 и контактов плис

Контакты порта P4	Контакты ПЛИС	Индикаторы стенда
0	P14	A0
1	P9	A1
2	P6	A2
3	P3	A3
4	P39	B0
5	P36	B1
6	P27	B2
7	P24	B3

Для создания проекта кроме элементов из библиотеки Maket на схеме должны быть размещены обозначения выходных контактов ПЛИС – OPAD (для МК – это контакты порта P4,

по которым поступают входные сигналы), входных контактов ПЛИС – IPAD (для МК – это контакты порта P4, на которые поступают выходные сигналы). Этим контактам присвоены соответствующие имена. Между контактами, источниками и приемниками сигналов на ПЛИС располагаются соответствующие буферные элементы – IBUF и OBUF. Эти элементы всегда используются для вывода из ПЛИС или ввода в ПЛИС однонаправленного сигнала (Pad– контактная площадка, которая подсоединена к контакту корпуса ПЛИС – Pin).

Если какой-то сигнал (контакт порта P4) является для ПЛИС выходным (для МК – входным), то подключение выполняется следующим образом.

Из библиотеки элементов извлекается элемент OPAD необходимой разрядности. Этому контакту присваивается имя, номер контакта и тип (рис.27.1):

Reference – своё имя (возможно наименование сигнала или номер контакта порта P4, например P4_4);

Name – выбрать LOC;

Description – P39 (для контакта, связанного с контактом P4.4).

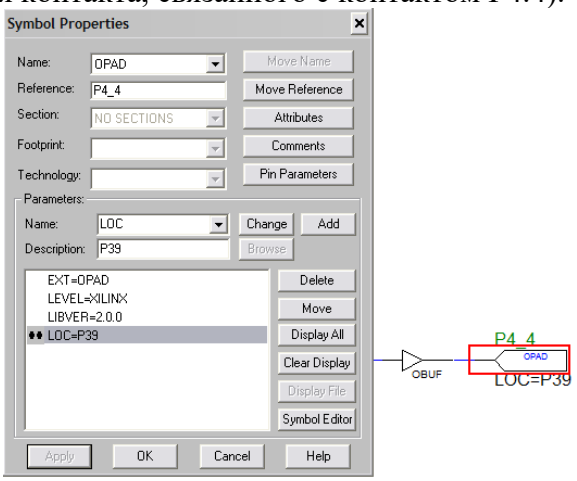


Рис. 27.1. Окно редактирования элемента OPAD

Следует помнить, что состояние источника сигнала при этом одновременно будет отображаться на соответствующем светодиоде индикатора A или B.

Для случая, когда какой-то сигнал (контакт порта P4) является для ПЛИС входным (для МК – выходным), из библиотеки элементов извлекается элемент IPAD и производятся действия, аналогичные как и для элемента OPAD.

Схема внешнего устройства – источника и приемника сигналов – показана на рис. 27.2.

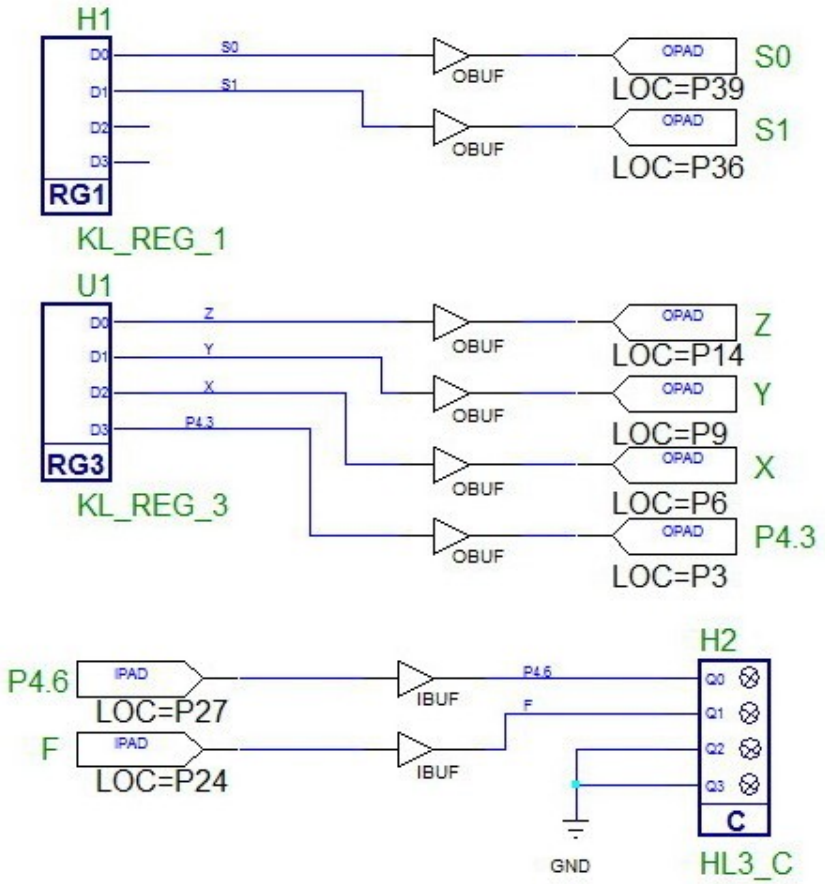


Рис. 27.2. Схема источника и приемника сигналов – проект, загружаемый в ПЛИС

Сигнал, соответствующий выходной функции, и сигнал подтверждения подключены к индикатору С. Этого можно было бы и не делать, так как состояние сигнала на контакте P4.7 отображается на светодиоде B3, а состояние сигнала на контакте P4.6 – на светодиоде B2.

Проект должен быть сохранён на сетевом диске U компьютера.

Для отладки программы на УЛС необходимо соблюдать строгую последовательность действий.

1. Выполнить размещение проекта Xilinx на кристалле (Impleventation).

2. Загрузить проект в УЛС.

Тумблер СТ на лицевой панели УЛС должен находиться в положении ВЫКЛ. Монитор-отладчик УЛС должен быть в исходном состоянии, для этого необходимо нажать на кнопку RESET на лицевой панели УЛС.

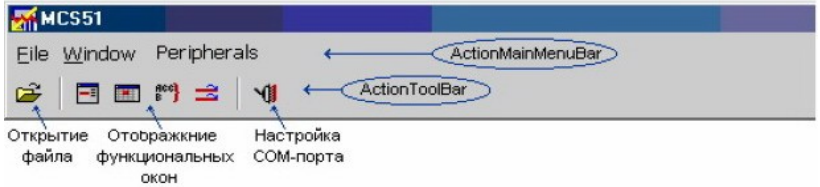
Генератор импульсов ГОИ1 на стенде должен быть включен в непрерывном режиме. Для созданного проекта на ПЛИС в этом нет необходимости, но от этого генератора работает и МК, для которого необходим именно этот режим.

3. Загрузить пользовательскую программу в УЛС.

На рабочем столе Windows дважды щёлкнуть мышью на пиктограмме системы:

Mcs.exe

Откроется главное окно управляющей программы – MCS51 (рис. 27.3).



По умолчанию при входе в систему открываются функциональные окна: программное окно (Program Window), окно памяти данных (Data Memory Window). Последнее в данной работе не требуется.

Открытие пользовательской программы доступно во вкладке меню «File», а также на панели быстрого доступа и с помощью «горячей клавиши».

Открытие программы и её отображение в программном окне осуществляется по файлу листинга – *.lst, однако код программы, загружаемый в МК, формируется из Hex-файла, в связи с чем необходимо проверить наличие Hex-файла в директории файла листинга!

Код выбранной программы отображается в программном окне (рис. 27.4).

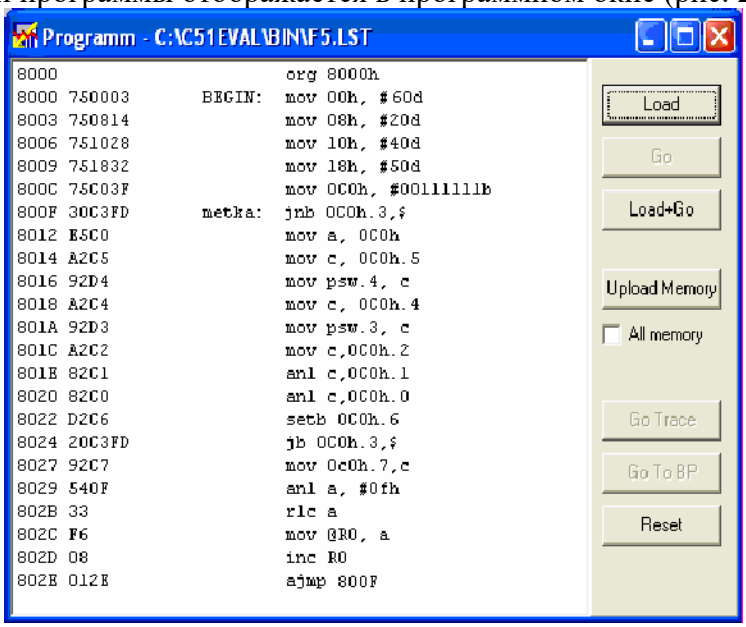


Рис. 27.4. Программное окно с загруженным кодом пользовательской программы

Отладка и выполнение программы, загруженной в МК стенда с помощью управляющей программы MCS51, могут выполняться в трёх режимах: «Основной», «Останов в КТ» и «Трассировка». Основной режим используется в тех случаях, когда необходимо наблюдать только конечный результат работы программы и в этом случае программа должна непрерывно выполняться (быть зацикленной). Этот режим и необходим при выполнении лабораторной работы.

После открытия программы пользователю становятся доступны кнопки Load и Load+Go.

При выполнении функции Load код, представленный в программном окне, загружается в эмулятор постоянного запоминающего устройства (ПЗУ), по результатам загрузки выдаются системные сообщения, становится доступной кнопка Go, при выборе которой происходит запуск на исполнение пользовательской программы, предварительно загруженной в эмулятор ПЗУ.

При выполнении функции Load+Go выполняется загрузка и автоматический запуск пользовательской программы.

Касаясь рассматриваемого примера, программный код которого представлен на рис. 27.4, обратим внимание на присутствие в коде ожидания сигнала разрешения и его снятия. В основном режиме выполнение программы будет приостановлено до момента выдачи сигнала разрешения (в нашем примере на регистре УЛС). После получения сигнала разрешения выполнение программы автоматически продолжается.

При отладке микропроцессорной системы и демонстрации её работы преподавателю необходимо показать, что после задания набора переменных на регистре УЛС и последующего изменения уровней сигнала разрешения – активный/пассивный – на индикаторе отображаются правильные изменения сигнала подтверждения и верное значение логической функции. Так как содержимое внутренней памяти данных МК при работе на стенде недоступно, то работа с различными банками регистров на УЛС не проверяется.

Варианты заданий

В табл.27.2 приведены варианты заданий, где даны обозначения:
в записи логической функции:
& – логическое “И”, v – логическое “ИЛИ”, ^ – отрицание;
уровня сигнала: L – низкий, H – высокий.

Таблица 27.2

Номер варианта	Логическая функция	Уровни сигналов управления	
		разрешение	подтверждение
1	X&(^Yv^Z)	L	L
2	^X&(Yv^Z)	L	H
3	X&Yv^Z	H	L
4	^XvY&Z	H	H
5	^(^X&^YvZ)	L	L
6	^X&(^YvZ)	L	H
7	^X&(YvZ)	H	L
8	^(^X&(YvZ))	H	H
9	^(XvY&^Z)	L	L
10	^X&^YvZ	L	H
11	^X&Yv^Z	H	L
12	^X&^Yv^Z	H	H
13	^Xv^Y&Z	L	L
14	^XvY&^Z	L	H
15	Xv^Y&^Z	H	L
16	^(Xv^Y&^Z)	H	H
17	^X&(^Yv^Z)	L	L
18	^(^X&(YvZ))	L	H
19	^Xv^Y&^Z	H	L
20	X&^YvZ	H	H
21	X&^Yv^Z	L	L
22	^X&YvZ	L	H
23	^XvY&Z	H	L
24	^(^XvY&Z)	H	H
25	XvY&^Z	L	L
26	X&(^YvZ)	L	H

Номер варианта	Логическая функция	Уровни сигналов управления	
		разрешение	подтверждение
27	^(X&(YvZ))	H	L
28	^(X&Yv^Z)	H	H
29	^(X&YvX&^Z)	H	H

Примечание. Функция должна иметь программную реализацию согласно виду, представленному в таблице вариантов, без каких-либо предварительных преобразований. Следует помнить, что при отсутствии скобок приоритет имеет операция логического умножения.

ПРАКТИЧЕСКАЯ РАБОТА №43. ТЕСТИРОВАНИЕ И ОТЛАДКА МИКРОПРОЦЕССОРНЫХ СИСТЕМ ПРИ РАЗРАБОТКЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Цель: изучение микропроцессорной системы для тестирования АЛУ.

Теоретическая часть

Рекомендации по подключению внешних устройств к системной шине и порту p4 микроконтроллера

Варианты подключения внешних устройств через системную шину

В МПС на основе УЛС все составные части системы подключаются к внешней системной шине. Шина представляет собой набор отдельных линий данных, адресов и сигналов управления. Оказывается, что шину в УЛС системной назвать нельзя, нельзя её называть и шиной данных. При ограниченном количестве задействованных контактов ПЛИС для подключения внешних устройств или средств визуализации используется ограниченный набор сигналов внешней системной шины: восемь разрядов шины данных DMK [7 0], стробы чтения и записи MKRD и MKWR соответственно и два адресных 7FFAh и 7FFBh, формируемых селектором адреса. Четыре последних сигнала – входные. Трехстабильная шина данных – это просто выходы порта P0 МК, где при выполнении программы или при обмене данными в режиме разделения времени выдаются младшие разряды адреса памяти (старшие в это время передаются через порт P2), а потом и байт данных.

Подключение схем на ПЛИС ко всем этим сигналам осуществляется через контакты ПЛИС, а собственно к контактам подключение осуществляется через IO Buffers – стандартные буферные элементы IBUF, OBUF, OBUFE, OBUFT.

Если просто нужно вывести из ПЛИС сигнал, не подключаемый к трёхстабильной шине данных, или ввести в ПЛИС какой-то сигнал с внешнего контакта, то на логической схеме это делается просто с помощью входных и выходных буферов IBUF, OBUF (так же, как это делается при подключении схемы в ПЛИС к порту P4 МК). Именно так подключаются стробы чтения и записи MKRD и MKWR и два адресных сигнала 7FFAh и 7FFBh.

Для трехстабильной двунаправленной шины данных всегда применяются контакты IOPAD. Подключение к внешней шине через контакты ПЛИС одного источника данных обычно никаких затруднений не вызывает. Для этого используются имеющиеся в библиотеке ПЛИС буферные элементы OBUFE с активным «высоким» состоянием разрешающего сигнала, как показано на рис. 28.3, или буферные элементы OBUFTc активным «низким» состоянием разрешающего сигнала.

Более сложная задача – подключение к выходным контактам нескольких внешних устройств – источников данных. Первый вариант решения – использование нескольких

буферных элементов OBUFE или OBUFT, объединённых у контакта. Для ПЛИС такой вариант непригоден.

Один источник подключается к контакту ПЛИС через элементы OBUFE, OBUFT. На схеме проекта эти элементы должны быть представлены, но при загрузке на кристалл они берутся из блока ввода-вывода, придаваемому каждому выводу корпуса ПЛИС. Аналогично элементы берутся из состава блока и в случае использования буферов IBUF, OBUF – минуя блок ввода-вывода выйти за пределы ПЛИС невозможно.

Блок ввода-вывода может быть конфигурирован как вход, выход или двунаправленный вывод. На рис. 29.1 показана структура блока ввода-вывода для контакта LOC=P80 (реализация в ПЛИС фрагмента схемы на рис. 29.2). Ни рис. 29.2 пунктиром отмечено, какие элементы и цепи из состава блока были использованы системой для реализации.

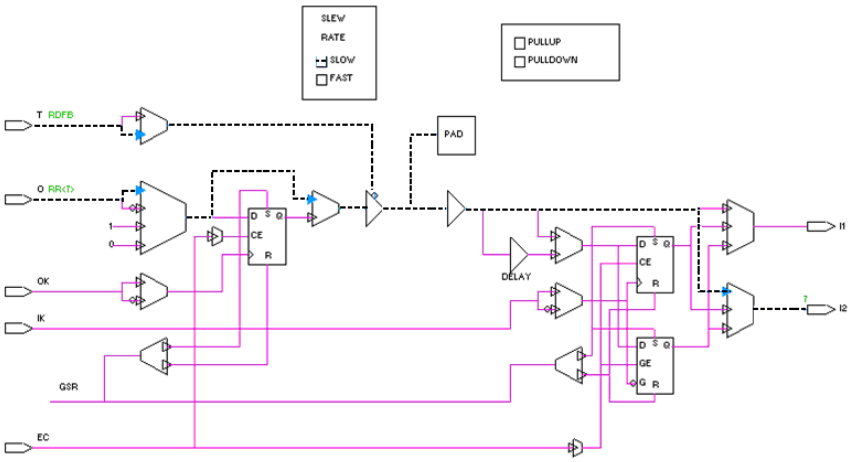


Рис. 29.1. Структура блока ввода-вывода в ПЛИС УЛС

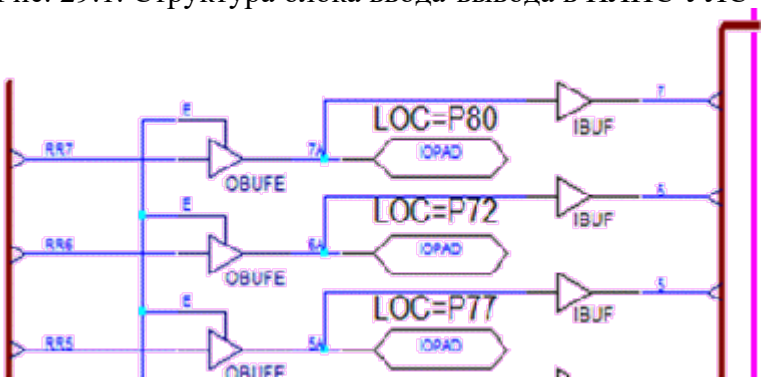


Рис. 29.2. Фрагмент схемы подключения к шине

В каждом блоке ввода-вывода может находиться только один выходной и только один входной буферные элементы. К единственному выходному буферному элементу несколько источников могут подключаться по-разному.

В ПЛИС необходимо организовать внутреннюю шину данных и подключить её к внешней шине. Можно использовать для этого из библиотеки элементов ПЛИС Tri-States – элементы с тристабильными выходами(BUFE, BUFT). Эти элементы находятся рядом с каждым конфигурируемым логическим блоком и вне него. Они через транзисторы-перемычки могут подключаться к общим шинам, проходящим вдоль всего кристалла.

Для подключения к шине, находящейся внутри ПЛИС, проще использовать мультиплексоры. Это позволяет избежать конфликтов и обеспечить более высокое быстродействие. Несмотря на это, в ПЛИС фирмы Xilinx всё-таки широко применяются шины с тремя состояниями, хотя это существенно повышает их себестоимость. Зато, во-первых, проще выполнить переход от проекта схемы на плате к проекту системы на кристалле. Во-вторых, устройство с общими шинами, к которым подключено несколько десятков источников, имеет аппаратные затраты в несколько раз меньше, чем такое же устройство, в котором шины заменены на эквивалентную схему из мультиплексоров.

В любом случае в практикуме необходимо делать выбор между реализацией на мультиплексорах или на буферных трёхстабильных элементах и решать вопрос с выбором источника управления буферными элементами или управления мультиплексором с учетом того, что МПС представляет для внешних устройств на ПЛИС ограниченное количество доступных адресов.

Рассмотрим различные схемные решения подключения к внешней шине двух и четырёх источников данных.

Варианты решения для двух источников данных

При реализации схемы подключения на мультиплексоре «2-в-1» для управления передачей данных необходим только один управляющий сигнал. Им может быть выходной битовый сигнал с одного из разрядов порта P4 (рис. 29.3). Подключение мультиплексора D1 к внешней шине выполняется через трёхстабильный буферный элемент OBUFEс активным «высоким» состоянием разрешающего сигнала. Разрешающий сигнал формируется на выходе элемента D2 при совпадении сигнала чтения MKRD и того или иного адресного сигнала.

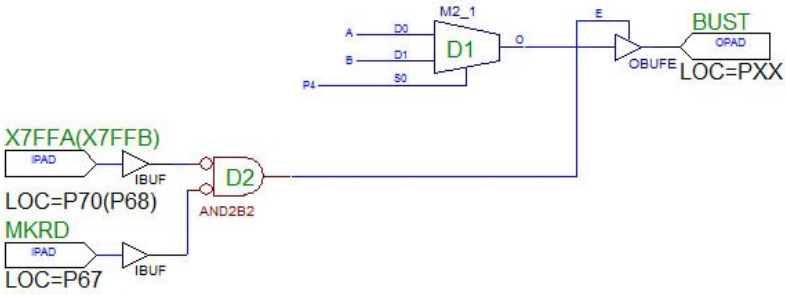


Рис. 29.3. Вариант подключения двух источников через мультиплексор (управление от порта P4)

При организации внутренней шины с использованием Tri States – элементов с трёхстабильными выходами – достаточно использовать либо два элемента BUFE с «активным» высоким состоянием управляющего сигнала и один инвертор, либо один элемент BUFE и один элемент BUFT с «активным» низким состоянием управляющего сигнала (рис. 29.4).

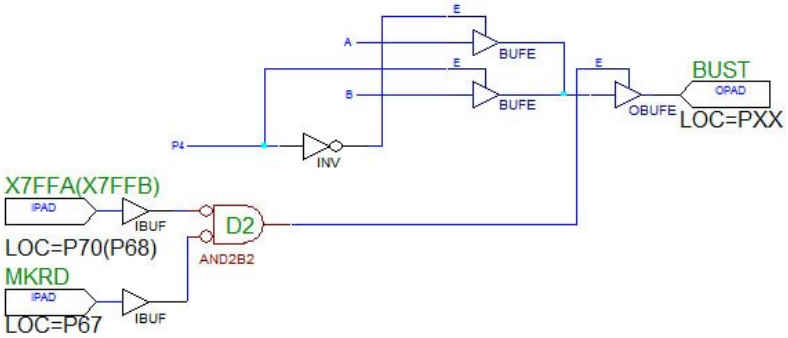


Рис. 29.4. Вариант подключения двух источников через буферные элементы (управление от порта P4)

Если все разряды порта P4 заняты, то источник управляющего сигнала может быть задан только при обмене по системной шине. При малом количестве адресов для обмена данными по шине приходится использовать решения, подобные тем, что применены в некоторых интерфейсных БИС, где для настройки БИС приходилось использовать строгую последовательность подачи определённых кодов с определёнными значениями в отдельных двоичных разрядах.

В схеме на рис. 29.5 используется дополнительный регистр адреса (для двух источников – один триггер). Перед чтением данных в этот триггер (D1) необходимо записать «1» или «0». Для выдачи данных с выбранного источника можно использовать команду чтения по тому же или

другому адресу. Схема подключения с использованием буферных элементов приведена на рис. 29.6.

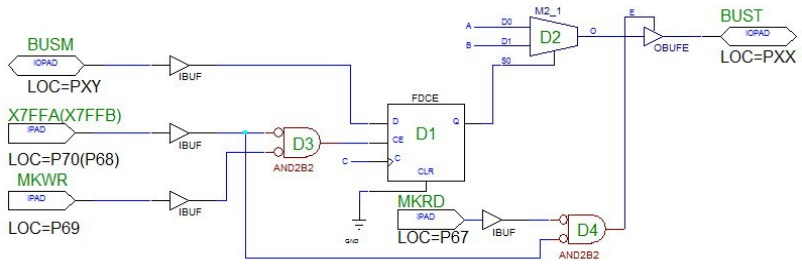


Рис. 29.5. Вариант подключения двух источников через мультиплексор

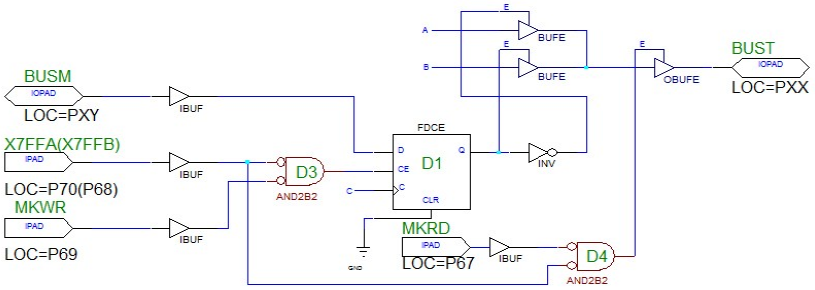


Рис. 29.6. Вариант подключения двух источников через буферные элементы

Варианты решения для четырёх источников данных

На рис. 29.7 показана одна из возможных схем подключения к шине нескольких внешних устройств – источников данных. В схеме используется дополнительный регистр адреса (RGADDR), код в который записывается микроконтроллером при обращении по адресу 7FFAh. В схеме в регистр записывается унитарный 4 разрядный код. При обращении МК по второму адресу (7FFBh) происходит запись данных с шины в один из четырёх регистров (внешних устройств – приёмников данных).

В зависимости от кода в регистре адреса подаётся разрешающий сигнал на один из четырёхбуферных элементов с «третьим» состоянием выхода. Для упрощения на схеме показано подключение к шине только одного (0-го) разряда источников данных.

При использовании унитарного кода к шине может быть подключено до восьми 8 разрядных источников данных и до восьми 8 разрядных приемников данных.

При использовании мультиплексора «4-в-1» для управления им требуются два сигнала: S0, S1. Поэтому в качестве регистра адреса используются два триггера (D1 иD2), и адрес источника представляется в виде двоичного кода (рис. 29.8).

При наличии двух свободных разрядов порта P4, которые можно использовать для управления выбором канала мультиплексора, схема подключения имеет вид, представленный на рис. 29.9.

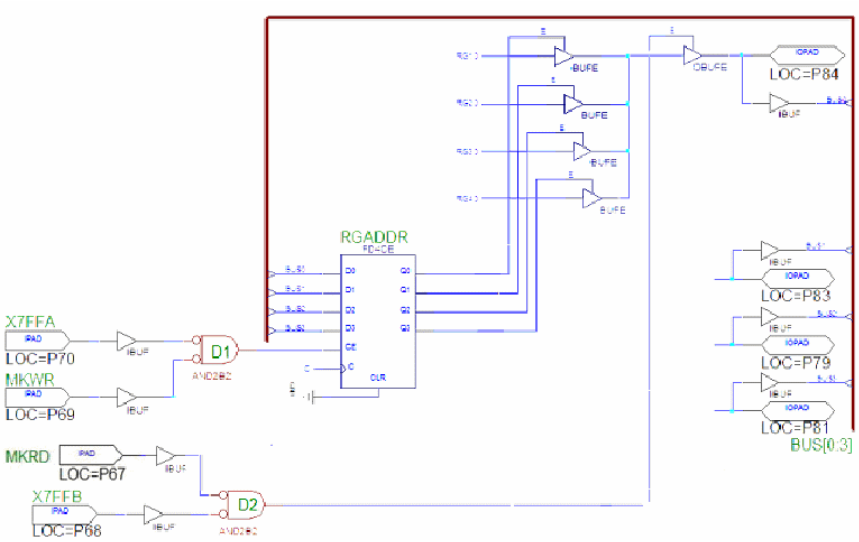


Рис. 29.7. Схема подключения к шине нескольких внешних устройств – источников данных (унитарное кодирование адреса)

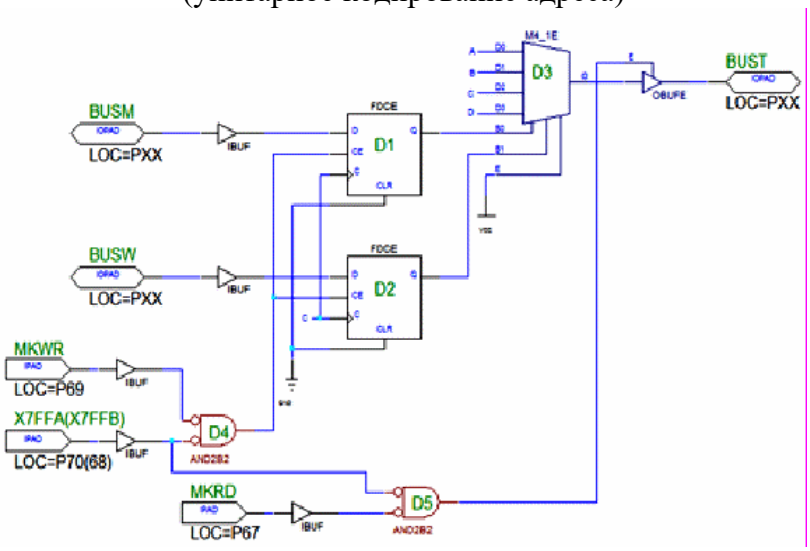


Рис. 29.8. Вариант подключения четырёх источников через мультиплексор (управление от регистра адреса)

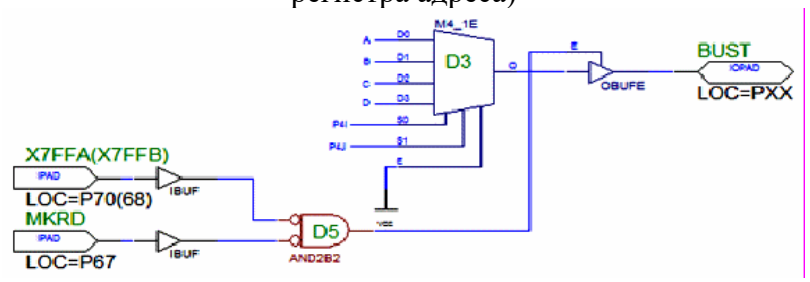


Рис. 29.9. Вариант подключения четырёх источников через мультиплексор (управление от порта P4)

Для схемы с использованием буферных элементов потребуется дешифратор (D3) для преобразования двоичного кода в унитарный (рис. 29.15).

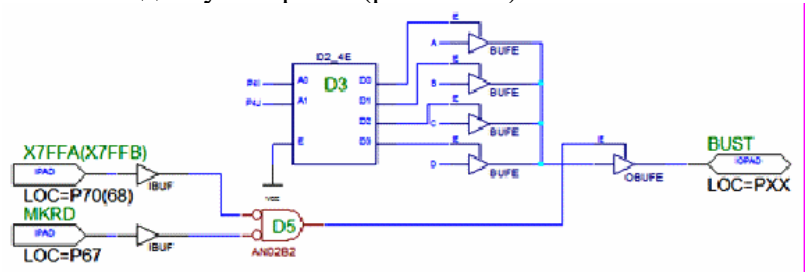


Рис. 29.10. Вариант подключения четырёх источников через буферные элементы (управление от порта P4)

Варианты решения для четырёх приёмников данных

На рис. 29.11 и 29.12 показаны возможные схемы подключения к шине четырёх внешних устройств – приёмников данных. Для упрощения на рисунках показана 4 разрядная шина. В схемах используется дополнительный регистр адреса (RGADDR), код в который записывается микроконтроллером при обращении, например, по адресу 7FFAh. В первой схеме в регистр записывается унитарный 4-разрядный код. При обращении МК по второму адресу (7FFBh) происходит запись данных с шины в один из четырёх регистров (внешних устройств – приёмников данных).

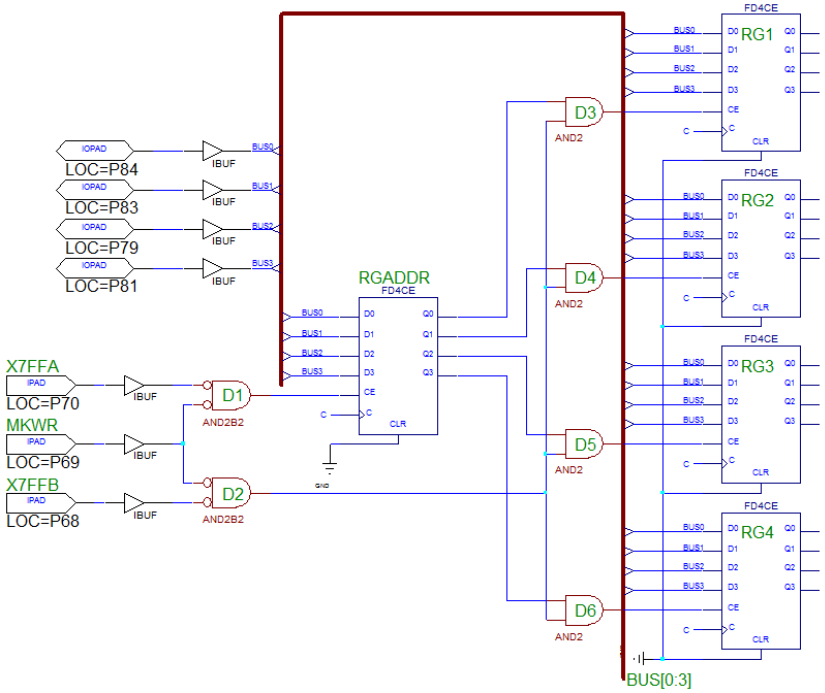


Рис. 29.11. Схема подключения к шине нескольких внешних устройств – приёмников данных (унитарное кодирование адреса)

В регистр адреса может быть записан и чисто двоичный код. К выходу регистра в таком случае подключается дешифратор. Использование дешифратора позволяет в предельном случае подключать к шине до 256-ти внешних устройств.

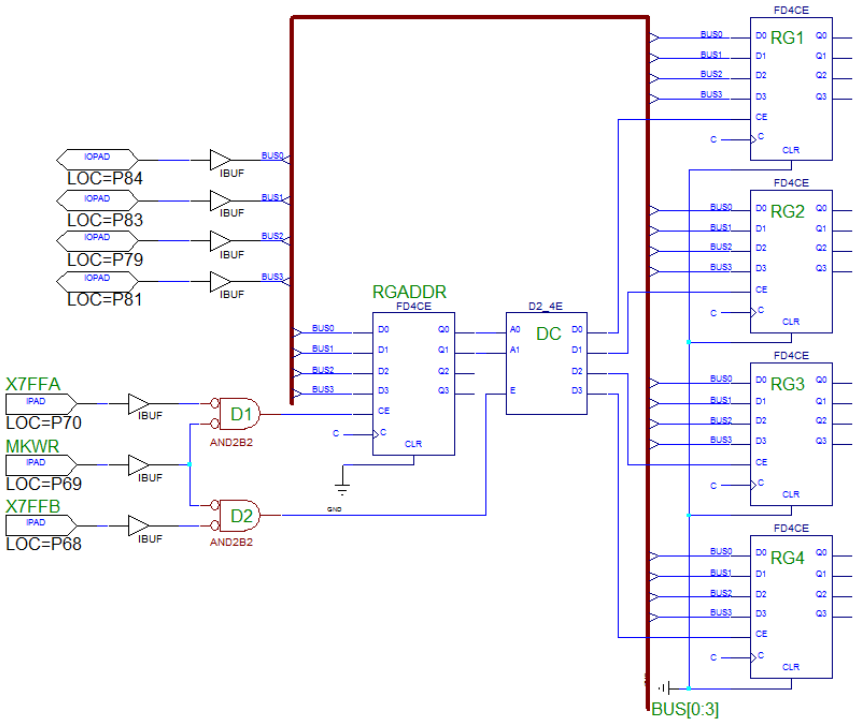


Рис. 29.12. Схема подключения к шине нескольких внешних устройств – приёмников данных (двоичное кодирование адреса)

Подключение внешних устройств к порту p4

На ПЛИС располагаются также внешние устройства, подключаемые к МК через порт P4. Если восьми разрядов одного порта оказывается недостаточно, то можно использовать следующие решения. Например, если МК требуется проанализировать состояния четырёх битовых сигналов, а свободно только три разряда порта, то при использовании мультиплексора сначала нужно задать код источника сигнала на адресных входах мультиплексора, а затем опросить выход мультиплексора (рис. 29.13).

Свободные четыре разряда порта P4 позволяют подключить к МК до восьми битовых сигналов. На адресные входы мультиплексора можно подавать код и с регистра адреса. В этом случае достаточно одного разряда порта P4. На рис. 29.14 для примера показана схема, которая позволяет при трёх свободных разрядах порта вести битовый сигнал в одну их четырёх точек схемы на ПЛИС.

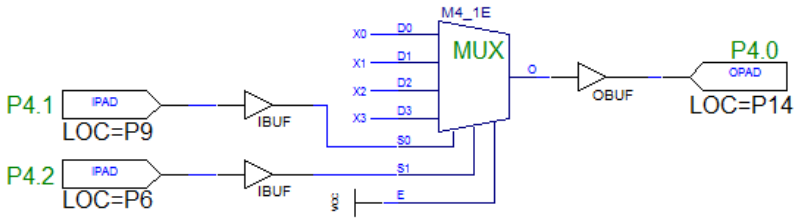


Рис. 29.13. Схема подключения к порту P4 нескольких битовых сигналов

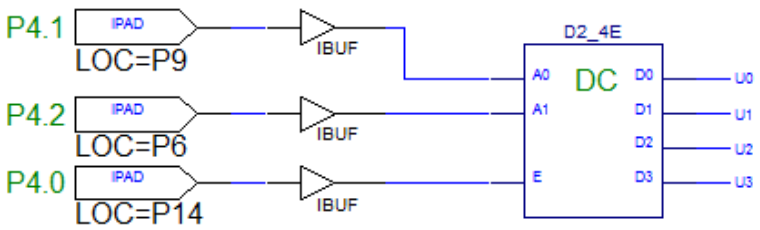


Рис. 29.14. Схема ввода в порт P4 нескольких битовых сигналов