МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт сервиса, туризма и дизайна (филиал) СКФУ в г.Пятигорске
Колледж Института сервиса, туризма и дизайна(филиал) СКФУ в г.Пятигорске

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

МДК.02.01 Микропроцессорные системы

Специальности СПО
09.02.01 Компьютерные системы и комплексы

Квалификация техник по компьютерным системам

Пятигорск 2020

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

Рассмотрено на заседании ПЦК ИСТиД (филиал) СКФУ в г. Пятигорске

Протокол №_8_от _12.03___2020 г.

Составитель: Н.А. Чернова

Директор З.А. Михалина

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.

Практическая работа №1. Ознакомление с работой учебного микропроцессорного комплекса.

Цель работы: ознакомиться с учебным микропроцессорным комплексом на базе восьмиразрядного CISC микропроцессора i8080 (отечественный аналог КР580ВМ80) с архитектурой фон Неймановского типа.

Теоретическая часть

Учебный микропроцессорный комплекс (УМК) представляет собой совокупность аппаратных и программных средств, которые позволяют изучать работу микропроцессора и других программируемых интегральных схем. Основа аппаратных и программных средств УМК - центральный блок, представляющий собой по структуре внутрисхемный эмулятор с программным монитором - вариант отладочного комплекса, используемого при создании программных средств для микропроцессорных контроллеров. Аппаратные средства центрального блока УМК представлены на Рис. 1. Программные средства (системный монитор) содержатся в постоянном запоминающем устройстве (ПЗУ1) центрального блока и занимают объем 1 килобайт с адреса 0 по адрес 03ffh. Системный монитор содержит программные средства, обеспечивающие:

- начальный запуск микропроцессора,
- работу в непрерывном или шаговом режиме,
- фиксацию точек останова с сохранением состояния процессора в стеке,
- работу клавиатуры и индикатора,
- выполнение некоторых встроенных процедур.

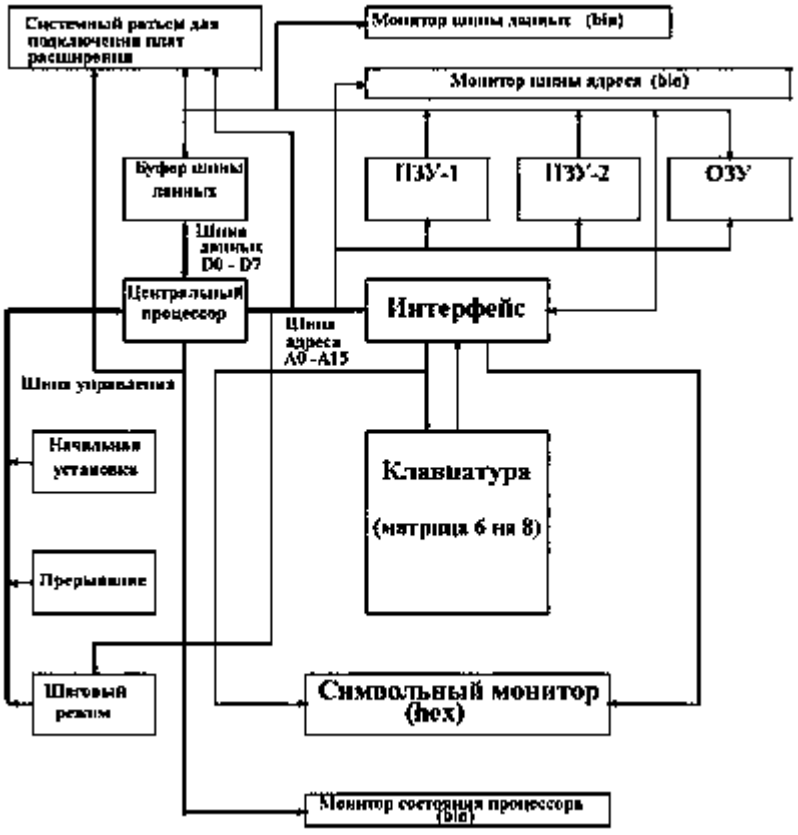


Рис. 1 Структурная схема центрального блока УМК

Кроме того предусмотрена прямая двоичная индикация состояния микропроцессора (PSW) и его шины данных и адреса.

ПЗУ2 объёмом 1К занимает адреса с 03ffh до 07ffh и зарезервировано для расширения системных возможностей УМК. Оперативное запоминающее устройство (ОЗУ или в англоязычной аббревиатуре RAM) статического типа (SRAM) объёмом 1К предназначено для хранения программ пользователя (прикладных программ), организации стека монитора и стека

пользователя, а также для поддержки работы системного монитора. Начальный адрес ОЗУ - 0800h.

В УМК предусмотрено расширение аппаратных возможностей за счет подключения к системному разъему центрального блока (см. Рис. 1) плат расширения, каждая из которых содержит комплект дополнительных аппаратных средств.

Кроме того, большинство плат расширения дает возможность создания произвольных аппаратных структур на макетном поле.

Перечень плат расширения :

- М1- параллельный интерфейс ;
- ПГМ - программатор РПЗУ и последовательный интерфейс;
- ПС - светодиодная матрица и дополнительная клавиатура;
- АЦА - аналоговый интерфейс ввода-вывода;
- М2 - расширение объема ЗУ и параллельный интерфейс;
- КОП – магистральный приборный интерфейс;
- ППИ - таймер с аппаратным и акустическим выходом.

Включение УМК

После включения кнопкой “ВКЛ” не должны светиться индикаторы перегрузки блока питания. В противном случае необходимо выключить УМК и после выдержки около 5 секунд повторить включение.

После включения питания нажать и отпустить кнопку “СБРОС”. В момент ее отпускания на входе “RESET” микропроцессора формируется сигнал высокого логического уровня для его начальной установки. На Рис. 2 представлены фрагменты электрической принципиальной схемы для цепи формирования сигнала “ RESET”. На рисунках сохранены все обозначения и нумерация элементов, принятые на электрических принципиальных схемах технического описания УМК завода изготовителя, поэтому порядок индексов отличается от того, который предписан ЕСКД.

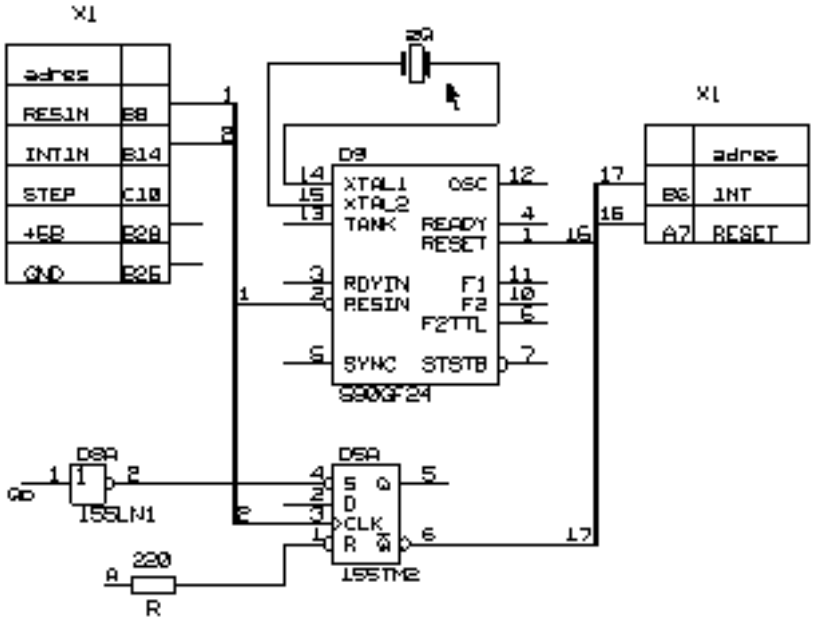


Рис. 2 Схема формирования сигнала “RESET”

Окончательное формирование системного сигнала “RESET” происходит в системном генераторе КР580ГФ24 (элемент D9 на Рис. 15). Для согласования контактной пары клавиши “СБРОС” с входом системного генератора используется стандартный набор аппаратных средств, основное назначение которых – исключение дребезга контактов при их размыкании и замыкании. Дребезг контактов – это переходный процесс, заключающийся в многократных коммутациях контактов при их соединении или разъединении. Длительность процесса составляет около 5 mc и зависит от упругости контактов и их поверхностных свойств.

Многократные коммутации приводят к ошибкам при вводе информации с любых контактных датчиков, в т.ч. клавиатуры. Для исключения ошибок используются программные и

схемные методы. В данном случае рассматривается наиболее распространенный схемный метод с использованием RS-триггера.

Схема, представленная на Рис. 15 содержит несколько элементов, расположенных в различных блоках УМК. Для их электрической связи используются разъёмы и межблочные соединения (межблочный монтаж). На электрических принципиальных схемах межблочные соединения, как правило, не приводятся, и наличие электрической связи подразумевается между одноименными контактами одноименных разъемов. Для упрощения графики сложных электрических принципиальных схем используются условные групповые линии, каждая из которых содержит необходимое число реальных соединительных линий. Каждая реальная линия при входе в канал нумеруется цифрами или буквенно-цифровым символом. Кроме номеров линий на принципиальной схеме приводятся номера выводов корпуса интегральной схемы и ее позиционный номер. Межблочные соединения приводятся на отдельной схеме соединений.

Встроенные процедуры

После прохождения сигнала “RESET”, сформированного аппаратным путем, происходит обнуление внутренних регистров МП. Устанавливается машинный цикл М1 – чтение кода первой операции из ячейки ЗУ с нулевым адресом. С нулевого адреса микропроцессор начинает выполнение программ системного монитора и продолжает их выполнение в циклическом режиме, поддерживая работу клавиатуры, индикатора и других элементов структуры. В этом режиме УМК ожидает нажатия оператором одной из функциональных клавиш для выполнения стандартной процедуры, входящей в состав программ системного монитора. В состоянии ожидания выбора процедуры в старшем разряде индикатора формируется знак "-".

Выполнение (“ВП”)

Процедура “ВП” (аналог “Enter” в компьютере) подтверждает принятые ранее установки оператора.

Пробел (“_”)

Нажатие клавиши “_” разделяет элементы вводимой с клавиатуры информации, например, адрес и данные или два разных адреса. При записи в память или чтении из памяти процедура, вызываемая клавишей “_” , выполняет операцию “инкремент” для кода адреса .

Обращение к внутренним регистрам (“РГ")

После нажатия клавиши "РГ" микропроцессор выполняет процедуру обращения к регистрам и находится в режиме ожидания имени регистра. Для ввода имени регистра используется основная шестнадцатеричная клавиатура УМК. Список регистров и их обозначения на клавиатуре:

- "A"- регистр аккумулятор, поддерживающий работу АЛУ;
- " B,C,D,E,H,L"- регистры общего назначения;
- " SL,SH"- регистры младшего и старшего байта счетчика стека;
- "PL,PH"- регистры младшего и старшего байта счетчика команд.

После ввода имени регистра, в двух младших разрядах индикатора УМК появляется содержимое данного регистра, представленное в шестнадцатеричной системе счисления. В старших разрядах - имя регистра.

Очередность действий при модификации содержимого регистров:

- нажатие "РГ" для входа в процедуру обращения к регистрам ввод имени регистра чтение содержимого регистра на светодиодном индикаторе;
- набор нового байта данных на шестнадцатеричной клавиатуре;
- нажатие клавиши "-" для записи нового байта данных в регистр или подтверждения существующего байта данных и перехода к режиму ожидания имени нового регистра;
- нажатие клавиши "ВП" для подтверждения сделанных изменений и выхода из процедуры.

Содержимое регистров сохраняется до нажатия клавиши “СБРОС”.

Обращение к ячейкам памяти (“П”)

Очередность действий при модификации содержимого ОЗУ:

- нажатие "П" для входа в процедуру обращения к памяти набор начального

адреса на шестнадцатеричной клавиатуре и ввод адреса нажатием клавиши "-";

- чтение содержимого данной ячейки ЗУ на светодиодном индикаторе;
- набор нового байта данных на шестнадцатеричной клавиатуре;
- нажатие клавиши "-" для записи нового байта данных или подтверждения существующего и перехода к следующему адресу массива (операция инкремент для кода адреса);
- нажатие клавиши "ВП" для подтверждения сделанных изменений в содержимом ячеек памяти и выход из процедуры.

Начать выполнение программы ("СТ")

Процедура "СТ" предназначена для запуска любой программы, расположенной в массиве ОЗУ или ПЗУ.

Очередность действий при запуске программы:

- нажатие "СТ" для входа в процедуру;
- набор начального адреса программы на шестнадцатеричной клавиатуре и ввод адреса нажатием клавиши "-";
- набор конечного адреса программы на шестнадцатеричной клавиатуре;
- ввод адреса и запуск программы пользователя нажатием клавиши "ВП".

В момент нажатия клавиши "ВП" микропроцессор выходит из программ системного монитора и выполняет программу пользователя. После окончания выполнения программы пользователя микропроцессор возвращается в системный монитор и в старших разрядах индикатора появляется адрес ячейки ЗУ, где расположен последний код выполненной программы. До нажатия клавиши "СБРОС" состояние всех регистров микропроцессора соответствует их состоянию на момент окончания программы пользователя. Состояние регистров сохраняется в стеке и восстанавливается при обращении к ним.

Подсчет контрольной суммы ("КС")

Очередность действий при запуске процедуры определения контрольной суммы:

- нажатие "КС" для входа в процедуру;
- набор начального адреса массива ЗУ на шестнадцатеричной клавиатуре и ввод адреса нажатием клавиши "-";
- набор конечного адреса массива ЗУ на шестнадцатеричной клавиатуре;
- ввод адреса и подсчет контрольной суммы при нажатии "ВП".

После выполнения директивы в младших разрядах индикатора выводится значение контрольной суммы. Директива используется для проверки правильности загрузки программ пользователя.

Запись константы ("ЗК")

Процедура предназначена для заполнения константой заданного массива ЗУ.

Очередность действий при запуске процедуры для записи константы:

- нажатие "ЗК" для входа в процедуру;
- набор начального адреса массива ЗУ на шестнадцатеричной клавиатуре и ввод адреса нажатием клавиши "-";
- набор конечного адреса массива ЗУ на шестнадцатеричной клавиатуре и ввод нажатием клавиши "-";
- набор однобайтовой константы на шестнадцатеричной клавиатуре;
- ввод и запись в массив нажатием клавиши "ВП".

Перемещение массива данных ("ПМ")

Очередность действий при запуске процедуры перемещения:

- нажатие "ПМ" для входа в процедуру;
- набор начального адреса перемещаемого массива ЗУ на шестнадцатеричной клавиатуре и ввод адреса нажатием клавиши "-";
- набор конечного адреса перемещаемого массива ЗУ на шестнадцатеричной клавиатуре и ввод адреса нажатием клавиши "-";
- набор начального адреса размещения на шестнадцатеричной клавиатуре;
- ввод адреса и выполнение перемещения нажатием клавиши "ВП".

При размещении программ в новом массиве ЗУ, например, при копировании в пространство ОЗУ системных программ, необходимо учитывать, что адреса ветвления и адреса, формируемые на основе заданных базовых адресов не будут модифицированы под новое адресное пространство, что приведет к ошибкам в работе программы.

Средства отладки программ, аппаратная и программная реализация шагового режима

Учебный микропроцессорный комплекс обладает набором средств для отладки и редактирования программ, представленных в машинных кодах. Для облегчения этого процесса при ограниченных возможностях УМК рекомендуется использовать модульный принцип при подготовке программ в ассемблере. Используемый в лабораторном курсе ассемблер не имеет средств для компоновки программы из отдельных модулей. При модульном принципе создания сложных программ каждый модуль должен быть скомпилирован в отдельную программу и отлажен средствами УМК. Для отладки программных модулей в УМК предусмотрен режим пошагового выполнения программы по машинным циклам и по командам. В шаговом режиме к шине данных и шине адреса микропроцессора подключаются двоичные индикаторы (см. Рис. 1). Двоичные индикаторы шины адреса и шины данных разбиты на тетрады для удобства восприятия двоичных чисел в шестнадцатеричной форме. Кроме того, предусмотрен двоичный индикатор кода состояния микропроцессора.

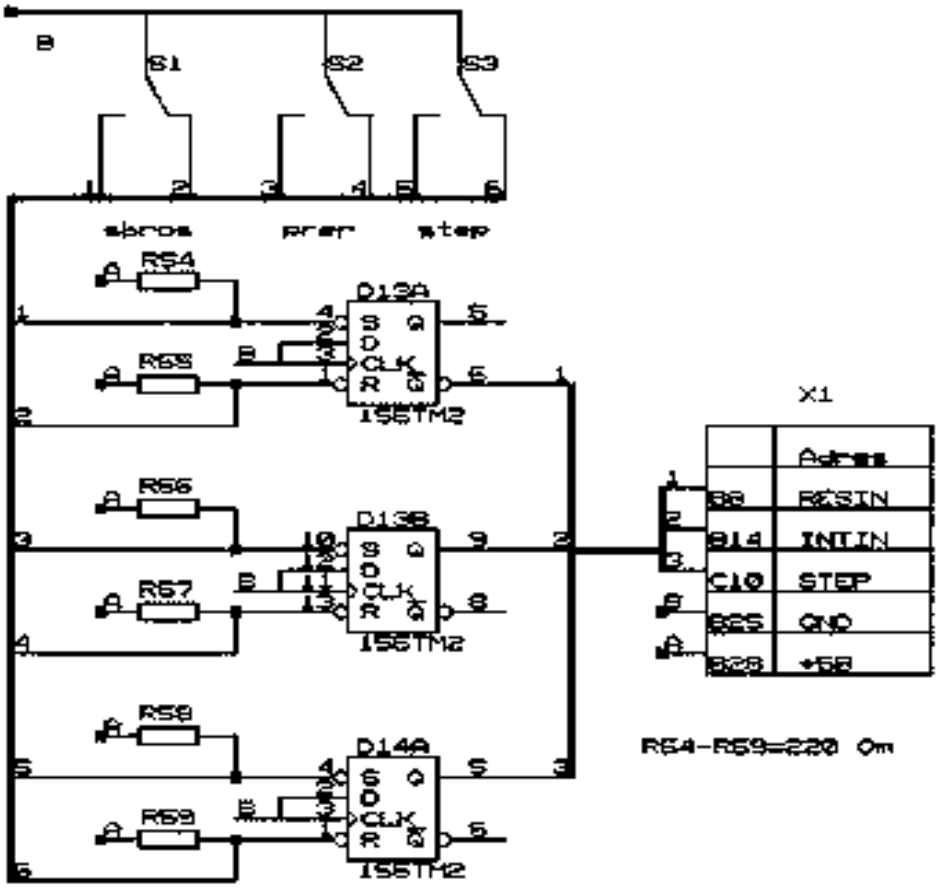


Рис. 3 Схема формирования сигнала "STEP" и устранения дребезга контактов аппаратными средствами

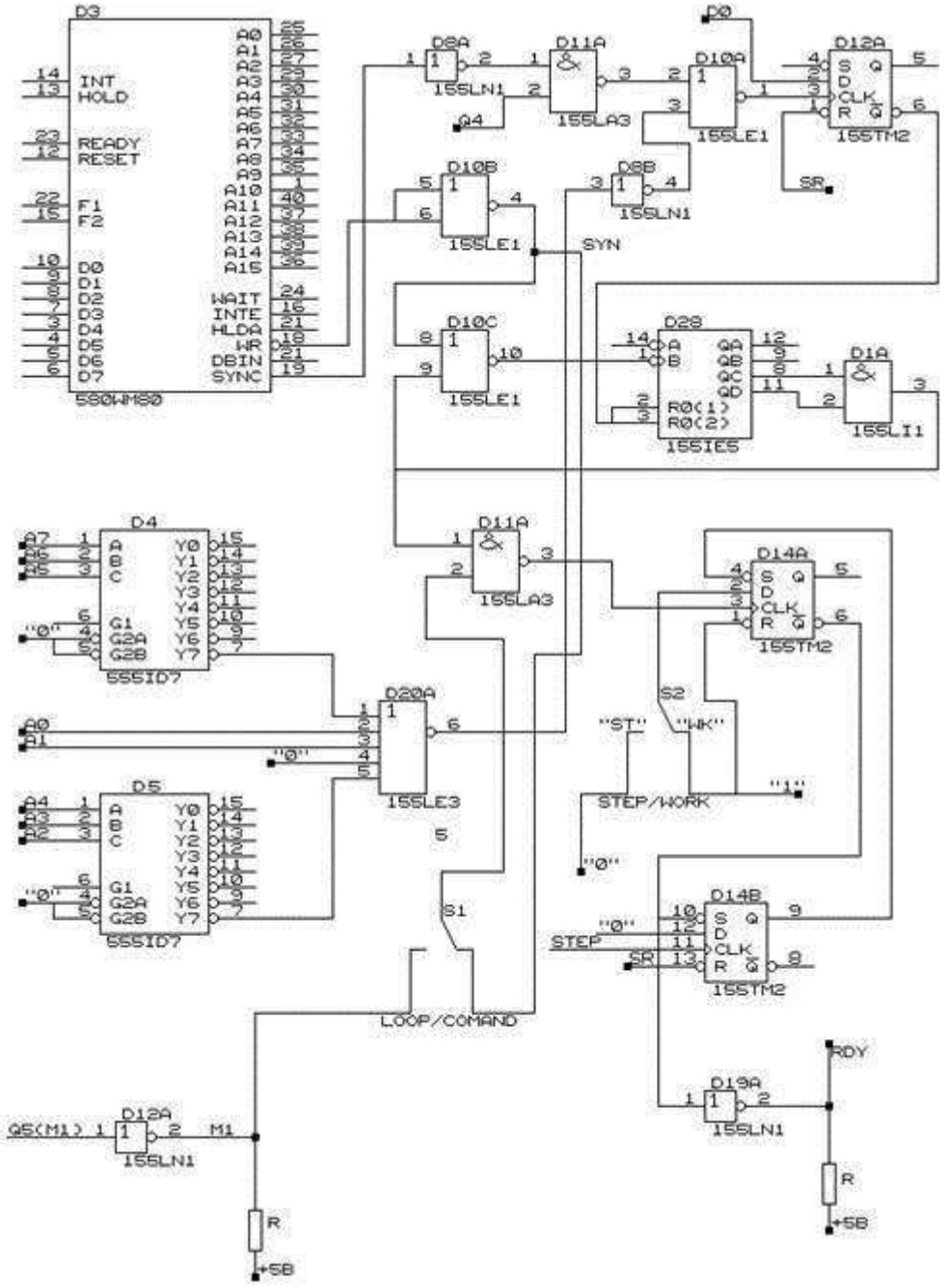


Рис. 4 Схема порта шагового режима

Шаговый режим выполнения программы устанавливается аппаратным путем при фиксации клавиши "РБ/ШГ" (Рис. 4) и может быть сброшен или восстановлен программным путем. Шаговый режим реализуется в микропроцессорных устройствах за счет использования внутреннего цикла ожидания микропроцессора. В цикл ожидания микропроцессор входит по сигналу на входе “RDY” (“ГОТОВНОСТЬ”). Окончательное формирование сигнала “ RDY” происходит в системном генераторе КР580ГФ24. Формирование входного сигнала “RD” Y для системного генератора выполнено программно-аппаратными средствами. На Рис. 17 приведен фрагмент схемы формирования сигнала “RDY” и порта шагового режима, доступного для программного управления.

В шаговом режиме переключатель S2 коммутирует низкий логический уровень на вход D-триггера D14-1. Низкий логический уровень на выходе инвертора D-19 появится при наличии стробирующего фронта по входу С триггера D14-1 (переход из 1 в 0) с выхода элемента D11. Стробирующие импульсы для записи информации в триггер выбираются переключателем S1 и соответствуют машинным или командным циклам микропроцессора. Условием прохождения

импульсов является наличие высокого уровня на верхнем входе элемента D11. Высокий уровень на верхнем входе элемента D11 образуется после прохождения через счетчик D13 определенного числа импульсов машинных циклов МП (импульсов с выхода SYN МП). Задержка включения шагового режима необходима для выполнения в непрерывном режиме процедуры "СТ" системного монитора при запуске программы пользователя в шаговом режиме.

Запрет счета импульсов машинных циклов, т.е. фактически сброс шагового режима, происходит при наличии высокого уровня на инверсном выходе триггера D12. Элементы D22,D23,D20, D12 образуют порт шагового режима. Выполнение машинного цикла или команды в шаговом режиме происходит при нажатии клавиши "ШАГ", инициирующей управляющий сигнал “ STEP”. Процесс формирования сигнала “STEP” можно проследить по схемам на рис.2 и рис.3. Сигнал “STEP” выводит микропроцессор из режима ожидания. Возврат в режим ожидания происходит в момент прихода очередного импульса начала машинного цикла или начала команды.

Процесс отладки программ значительно упрощается при использовании подпрограмм или процедур, в т.ч. имеющихся в составе системного монитора. Основой для использования подпрограмм является условие сохранения (восстановления) состояния микропроцессора при выполнении подпрограммы или целенаправленное изменение этого состояния по результату работы подпрограммы. В микропроцессоре ВМ80 автоматически (на микропрограммном уровне) сохраняется только содержимое программного счетчика (“ PC”), т.е. адрес возврата из подпрограммы. Для этого используется стековая память, организованная в массиве ОЗУ с помощью регистра стека “SP”.

Для сохранения содержимого других регистров МП используются программные средства обращения к стеку. При начальной установке микропроцессора в регистр стека загружается адрес, являющийся нижней границей ОЗУ. Каждое обращение к стеку для записи информации уменьшает содержимое регистра “SP” на единицу. Каждое обращение для чтения информации из стека увеличивает содержимое регистра “SP” на единицу. При выполнении операции “ RET” микропроцессор считывает в качестве адреса возврата текущие значения двух байтов из вершины стека.

Практическая часть

Задание 1. Построить временные диаграммы процесса формирования системных сигналов “ INT”, ”RESET”, “STEP”.

Задание 2. Обратиться к ячейке ЗУ с адресом 0000 h. В этой ячейке расположен код операции первой команды системного монитора. Обращаясь последовательно к ячейкам ЗУ, записать текст программы начальной установки УМК до адреса 0052h в машинных кодах. Выполнить дизассемблирование кодов и проанализировать полученную программу начального запуска МП. При записи программы необходимо учитывать, что не запрограммированные ячейки РПЗУ содержат код FFh.

Задание 3. С помощью процедуры "П" обратиться к адресу 035Bh. Это начальный адрес подпрограммы “DELAY” системного монитора. Назначение данной подпрограммы – формирование интервала времени (10 ms).

Выполнить дизассемблирование подпрограммы “ DELAY”, составить алгоритм подпрограммы и проанализировать принцип формирования интервала времени программным путем.

Составить и запустить программу формирования произвольного интервала времени, кратного 10 ms, с использованием подпрограммы “ DELAY” и регистров DE в качестве счетчика циклов обращения к подпрограмме.

Задание 4. Составить и запустить программу для определения нижней границы оперативной памяти УМК. Объяснить разницу между адресом, полученным в результате работы программы, и адресом в счетчике SP при начальной установке УМК (граница доступной памяти).

Задание 5. Составить электрическую принципиальную схему подключения одного разряда светодиодного индикатора к БИС интерфейса и схему формирования управляющих сигналов БИС интерфейса.

Задание 6. С помощью соединительных проводников создать модель канала

последовательной передачи данных в асинхронном режиме между двумя УМК. Все коммутации производить при отключенном питании УМК. Составить программу для обмена данными между двумя УМК в асинхронном режиме с заданной преподавателем скоростью передачи. Выполнить двухсторонний обмен информацией в асинхронном режиме.

Задание 7. С помощью соединительных проводников создать модель канала последовательной передачи данных в синхронном режиме между двумя УМК. Все коммутации производить при отключенном питании УМК. Составить программу для обмена данными между двумя УМК в синхронном режиме с заданными преподавателем параметрами передачи (вид синхронизации, число синхрослов, вид синхрослов). Выполнить двухсторонний обмен информацией в синхронном режиме.

Контрольные вопросы.

1. Что составляет основу аппаратных и программных средств УМК?
2. Какой объем имеет ПЗУ и ОЗУ?
3. За счет чего производится расширение аппаратных возможностей?
4. Как формируется сигнал RESET?

Практическая работа №2. Изучение принципов работы компьютеров в машинных кодах.

Цель работы: изучить принципы работы компьютеров в машинных кодах

Теоретическая часть

Двоичный код – основа функционирования компьютера

Вся информация хранится и обрабатывается в компьютере в двоичной системе. Сама программа обработки также представляет собой не что иное, как двоичный код. В настоящее время, когда пользователь отделен от компьютерного «железа» несколькими слоями программного обеспечения, данный факт не очевиден: вы вводите в электронную таблицу десятичное число, и в соседней ячейке ответ выводится также в десятичном виде. Так что тезис о том, что внутри компьютера вычисления были произведены в двоичном коде, во многом приходится принимать на веру.

Сейчас мы не будем пытаться доказать или продемонстрировать справедливость тезиса о двоичном способе хранения и обработки информации в компьютере. Будем ссылаться на данный тезис как на твердо установленный факт.

Итак, компьютер способен непосредственно обрабатывать только двоичную информацию. Принять бинарную информацию он может из устройств внешней памяти, а также от других компьютеров через посредство сети. Хотя устройства внешней памяти весьма разнообразны, их объединяет именно то, что они хранят любые данные в двоичном виде, т.е. в виде, пригодном для наиболее быстрого и оперативного использования. Что же касается человека, то он, разумеется, нуждается в специальных устройствах, которые преобразуют его информацию во внутреннюю компьютерную форму и обратно. Для ввода бинарных машинных кодов человек легко может использовать клавиатуру. Важно понимать, что прием данных с клавиатуры (даже если предположить, что некий оригинал вводит чисто двоичный код!) всегда происходит по определенной программе, которая преобразует поток набираемых символов в двоичные числа и отправляет последние в необходимое место ОЗУ. Замена громоздких двоичных чисел более компактными восьмеричными или даже шестнадцатеричными числами дела не меняет, лишь несколько усложняя программу кодирования.

Мы видим, что простейшим способом организовать ввод цифровых кодов в память является несложное преобразование последовательности символов, соответствующих допустимым значениям цифр, в двоичный код. Именно такая программа, называемая монитором, имелась в ПЗУ отечественных «ДВК-образных» машин (семейство ДВК, БК, УКНЦ). Монитор позволял вводить коды в виде восьмеричных цифр и записывать их в набранный таким же образом адрес ОЗУ. Еще он умел запустить машинную программу и перехватить ее завершение, а

также выполнял некоторые другие действия. Если внимательно подумать, то по своей сути такой монитор являлся непосредственным предшественником отладчиков типа Debug. Главной особенностью последнего по сравнению с монитором является возможность символьного представления машинных инструкций. Таков первый шаг от машинных кодов к ассемблеру.

Ассемблер: символьная запись команд

Цифровой ввод, легко распознаваемый машиной, довольно неудобен для человека. В целом большинству людей проще запомнить некоторое слово (даже на неродном языке, например, ADD или LOOP) чем комбинации цифр, им соответствующие. Аналогичным образом часто отказываются от обращения ко внутренним регистрам микропроцессора по номерам, заменяя их буквенными обозначениями (EAX, AX, AH, AL, BX, IP и т.д.) Подобная замена особенно уместна, если регистры микропроцессора не являются универсальными и имеют выделенное назначение (например, A – аккумулятор). Заметим, что неуниверсальность рабочих регистров и их жестким образом фиксированное использование, что свойственно семейству процессоров Intel, вовсе не единственный возможный вариант. Скажем, в семействе PDP логическая структура процессоров была более стройной: все регистры общего назначения РОН могли использоваться в машинных инструкциях на равных основаниях; в такой ситуации давать регистрам имена было менее удобно, чем просто пронумеровать.

Описанные символьные обозначения стали называться ***мнемониками*** (мнемонический значит облегчающий запоминание). Вот несколько мнемоник для записи машинных инструкций процессора Intel.

Таблица 1

hex-код	**мнемоника**	**комментарий**
B81200	MOV AX,12	занести число 12 в регистр AX
01D8	ADD AX,BX	сложить регистры AX и BX
8B1E1001	MOV BX,[110]	прочитать в BX ячейку ОЗУ с адресом 110

Примечание. В таблице жирным шрифтом выделен код, описывающий операцию и способы обращения к данным; обычным шрифтом набраны непосредственные данные и адреса ячеек ОЗУ. Обратите внимание, что использованные в командах двухбайтовые числа хранятся «задом наперед»: число 0110 лежит в памяти не как 01 10, а, наоборот, 10 01. Подобный способ хранения, принятый в IBM PC, носит название *обратный порядок хранения байтов*.

Думается, читатели согласятся, что мнемонический способ записи команд легче понимается и запоминается. Именно поэтому при работе с Debug обычно пользуются именно им. Мнемоническое представление команд является неотъемлемой частью ассемблера, но отнюдь не главной его частью. Наибольшее достоинство языка ассемблер состоит в возможности заменять символическими именами не только операции и регистры, но и конкретные адреса памяти. В последнем случае программист получает возможность освободиться от «привязки» к конкретным адресам памяти, что, в свою очередь, позволяет элементарно, как в обычном текстовом редакторе, удалять или дополнять команды программы. Такая замечательная возможность заслуживает более подробного обсуждения.

Ассемблер: идентификаторы и директивы

Имеется как минимум две ситуации, когда программа однозначно привязывается к конкретным адресам памяти. Во-первых, при обращении к некоторым переменным (данным), которые находятся не во внутренних регистрах микропроцессора, но хранятся в памяти. И, во-вторых, при переходах, без которых невозможно организовать разветвляющиеся и циклические программы. Хотя обе ситуации по смыслу отличаются (происходит обращение к данным или программе), с формальной точке зрения оба этих случая выглядят необычайно похоже: требуется сослаться на адрес некоторой ячейки памяти – неважно, что именно там находится.

Появление в программах конкретных адресов приводит к потере ее мобильности. Например, если по какой-либо причине потребуется изменить адрес переменной, то придется внести коррекцию во все команды, где на эту переменную содержится ссылка. Или при вставке всего одной машинной инструкции вся последующая часть программы сдвигается и приходится аналогичным образом модифицировать все переходы на ее адреса.

Чтобы читатели получили некоторое представление об описанных трудностях, реализуем в Debug простейшую программу.

Пусть мы хотим решить на компьютере простейшую задачу, которая состоит вычислении по формуле r = x + y. Напишем и реализуем программу решения, а затем посмотрим, что потребуется исправлять, если мы захотим добавить к нашей формуле еще одну операцию, например: r = x + y – z.

Договоримся, что все переменные являются двухбайтовыми целыми. Примем, что область хранения переменных будет находиться «в самом начале» – начиная с адреса 102. Поскольку по принятым в MS-DOS соглашениям программа стартует, начиная с адреса 100, придется предусмотреть «обход» области данных.

Таблица 2

адреса	**содержимое**	**комментарии**
100,101	jmp 108	на начало программы (обход данных)
102,103	3	x
104,105	7	y
106,107	0	r

Примечание. Размещением данных указанным способом имеет свои достоинства и недостатки. С одной стороны, требуется обход области данных, возникают трудности при дизассемблировании (см. ниже), зато с другой – данные сохраняются на диск вместе с программой единым массивом и адреса информации в момент написания программы уже определены. При работе с Debug последнее немаловажно.

Итак, память для данных спланирована и можно, постоянно заглядывая в табл. 2, набрать программу (см. следующий ниже протокол; директива dw в нем означает двухбайтовое число).

Добавим, что в протоколе жирным шрифтом показаны символы, которые нам пришлось набирать; все остальные выведены на экран отладчиком. Причем текст, выделенный при редактировании курсивом, может отличаться от того, который получился на компьютере автора, а многоточие заменяет выводимые на экран строки, содержимое которых для нашей задачи абсолютно несущественно.

```
-a
13E4:0100 jmp 108
13E4:0102 dw 3
13E4:0104 dw 7
13E4:0106 dw 0
13E4:0108 mov ax,[102]
13E4:010B add ax,[104]
13E4:010F mov [106],ax
13E4:0112 int 20
13E4:0114
-u
13E4:0100 EB06        JMP     0108
13E4:0102 0300        ADD     AX,[BX+SI]
13E4:0104 07          POP     ES
13E4:0105 0000        ADD     [BX+SI],AL
13E4:0107 00A10201    ADD     [BX+DI+0102],AH
13E4:010B 03060401    ADD     AX,[0104]
13E4:010F A30601      MOV     [0106],AX
13E4:0112 CD20        INT     20
...
-u 108
13E4:0108 A10201      MOV     AX,[0102]
13E4:010B 03060401    ADD     AX,[0104]
```

```
13E4:010F A30601       MOV     [0106],AX
13E4:0112 CD20         INT     20
...
-g
Программа завершилась нормально
-d 102
13E4:0100       03 00 07 00 0A 00-A1 02 01 03 06 04 01 A3     ..............
13E4:0110  06 01 CD 20 00 00 00 00-00 00 00 00 34 00 D3 13   ... ........4...
...
13E4:0180  00 00
```

Теперь проконтролируем набор директивой ***u***. Из протокола видно, что наличие данных посреди программы «сбивает» дизассемблер отладчика: пытаясь интерпретировать значения переменных как машинные команды, Debug «не попадает» на адрес 108. В результате первая команда программы выглядит неверно. Проверка директивой ***u 108*** подтверждает, что в памяти все сохранено правильно.

Примечание. Из последней распечатки запомним тот факт, что программа начинается с адреса 108 и завершается байтом 113. Эти сведения нам потребуются позднее при переделке программы.

Остается запустить нашу программу и посмотреть ответ, для чего вывести содержимое памяти командой ***d 102***. При расшифровке шестнадцатеричных чисел 3, 7, A (10_{10}) следует обязательно вспомнить о примечании к таблице 1 по поводу обратного хранения байтов в памяти. Для удобства анализа протокола результирующие байты переменной r в протоколе подчеркнуты.

Переходим теперь к наиболее интересной части нашего эксперимента. Что придется поменять, чтобы переделать нашу программу для расчета по исправленной формуле r = x + y – z. Увы, придется проделать довольно много операций, хотя каждая из них сама по себе несложная.

Начнем с расширения таблицы переменных. Новое распределение памяти приведено в табл. 3 (советуем читателям внимательно сравнить ее с предыдущей таблицей; все изменения в табл. 3 выделены цветом и подчеркнуты).

Таблица 3

адреса	**содержимое**	**комментарии**
100,101	jmp 10A	на начало программы (обход данных)
102,103	3	x
104,105	7	y
106,107	4	z
108,109	0	r

Теперь будем вносить необходимые изменения. Все требуемые действия зафиксированы в приведенном ниже протоколе, *который является продолжением предыдущего*.

```
-m 108 113 10a
-u 10a
13E4:010A A10201       MOV     AX,[0102]
13E4:010D 03060401     ADD     AX,[0104]
13E4:0111 A30601       MOV     [0106],AX
13E4:0114 CD20         INT     20
...
-a 111
13E4:0111 sub ax,[106]
13E4:0115 mov [108],ax
13E4:0118 int 20
13E4:011A
-a 100
```

```
13E4:0100 jmp 10a
13E4:0102
-e 106 4 0
-u 100
13E4:0100 EB08        JMP     010A
13E4:0102 0300        ADD     AX,[BX+SI]
13E4:0104 07          POP     ES
13E4:0105 0004        ADD     [SI],AL
13E4:0107 00A102A1    ADD     [BX+DI+A102],AH
13E4:010B 0201        ADD     AL,[BX+DI]
13E4:010D 03060401    ADD     AX,[0104]
13E4:0111 2B060601    SUB     AX,[0106]
13E4:0115 A30801      MOV     [0108],AX
13E4:0118 CD20        INT     20
...
-g
Программа завершилась нормально
-d 102
13E4:0100       03 00 07 00 04 00-06 00 A1 02 01 03 06 04    ..............
13E4:0110  01 2B 06 06 01 A3 08 01-CD 20 00 00 34 00 D3 13   .+....... ..4...
...
13E4:0180  00 00                                             ..
```

Передвинем основную часть программы, чтобы освободить 2 байта под новую переменную. Для этого наберем команду ***m 108 113 10a***, которая означает задание отладчику подвинуть (move) байты со 108 по 113 (эти значения мы запомнили из предыдущих экспериментов с программой). Проверим, что программа действительно теперь находится с адреса 10a. Далее можно было бы аналогичным образом освободить байты под команду вычитания и ввести ее, но проще набрать 3 последние команды заново, тем более, что в команде записи результата в переменную r все равно потребовалось бы изменить адрес. Так что введем эти команды с адреса 111, а затем не забудем поменять адрес в стартовой инструкции перехода. Далее командой ***e 106 4 0*** занесем в переменную z ее числовое значение (опять-таки в обратном порядке!) и все еще раз проверим. Наконец, запустим программу и убедимся в правильности получившегося результата: 3 + 7 – 4 = 6.

Таким образом, поработав с простейшей программой, мы убедились в том, что замена адресов и переменных, и переходов при модификации программы представляет собой весьма трудоемкую работу, требующую предельного напряжения внимания (но, несмотря на все усилия, ошибки неизбежно случаются). От указанных трудностей можно избавиться, если для реализации программы в командах процессора использовать ассемблер. Главное преимущество ассемблера перед программой Debug заключается в том, что ассемблер нигде не использует конкретные адреса ОЗУ – вместо них везде указываются символические имена, которые называются ***идентификаторы***. При трансляции текста программы ассемблер автоматически связывает идентификаторы с адресами ячеек памяти, в которых они будут располагаться. Как следствие, модификация текста программы не потребует никакого пересчета адресов, поскольку ассемблер при новой трансляции распределит их уже в соответствии с модифицированной программой.

Текст нашей программы на ассемблере будет выглядеть так.

```
    jmp start
x: dw 3
y: dw 7
z:  dw 4
r:    dw 0
start: mov ax,x
    add ax,y
```

sub ax,z
mov r,ax
int 20

Легко видеть, что вставка двух выделенных строк это все, что потребуется при рассмотренной выше модификации программы. К сожалению, **Debug не позволяет использовать идентификаторы**, а значит, заметно уступает ассемблеру в данном вопросе. Впрочем, никто и не обещал, что данное программное средство является полноценным ассемблером.

Язык ассемблер содержит целый ряд специальных управляющих операторов (обычно их называют *директивами*), из которых Debug поддерживает только некоторые. В частности, команда dw 3, определяющая в памяти некоторую константу, суть одна из таких поддерживаемых отладчиком директив. В то же время, ассемблер содержит целый ряд директив, которые отсутствуют в Debug [2]. Между прочим, в свете наличия дополнительных директив, некоторые из которых в ассемблерной программе приходится писать обязательно, программа для Debug вводится проще. Зато ассемблер засчет добавочных директив становится мощнее; например, отдельные участки ассемблерной программы можно помещать внутрь условий, так что в зависимости от их выполнения или невыполнения будет генерироваться различный код.

Таким образом, оказывается, что Debug не является в полном смысле слова ассемблирующей программой, хотя и позволяет вводить ассемблерные мнемоники команд. Следовательно, работая в Debug, мы не реализуем всех возможностей, которыми обладает настоящий язык ассемблер.

Отметим, что хорошее понятие об ассемблере дают школьные учебники [3-6], где описываются учебные ЭВМ «Малютка» и «Нейман» (в книгах [4,6] изложение проведено более подробно). Особенно полезна практическая работа с программными реализациями указанных моделей, в которых имеется поддержка ассемблера.

В качестве еще одного эксперимента предлагаем сконструировать свой собственный ассемблер. Главное достоинство идеи заключается, конечно, не в практической полезности полученного продукта, а в том, что в процессе написания программы мы лучше поймем принципы ассемблирования программ.

Таблица 4

Операция	**Код**		**Мнемоника**	**Комментарии**
	2 с/с	**8 с/с**		
перепись	000	0	MOV	A3 = A1
сложение	001	1	ADD	A3 = A1 + A2
деление нацело	010	2	DIV	A3 = A1 div A2
модуль разности	011	3	SUB	A3 = |A1 - A2|
переход по =	100	4	JE	при A1 = A2 – перейти к A3
умножение	101	5	MUL	A3 = A1 * A2
переход по >	110	6	JG	при A1 > A2 – перейти к A3
стоп и вывод	111	7	HLT	A1, A2, A3 на дисплей; стоп

Для уменьшения технической работы используем некоторые упрощения; они не повлияют на понимание принципов ассемблера, зато существенно ускорят программирование.

- Каждая команда занимает отдельную строку, пустые строки запрещены. Согласно устройству «Крохи», *любая* команда занимает *ровно одну* ячейку, значит, номер ячейки фактически совпадет с номером строки.
- Все метки (идентификаторы) условимся обозначать одной латинской буквой, после которой ставится двоеточие. Хотя это и несложно, проверять каждую новую метку на совпадение со старыми не будем; факт того, что это именно латинская буква, для простоты также не проверяется.
- Части команды (метка, операция и 3 адреса) разделены между собой произвольным

количеством пробелов (M: ADD X Y R).

- Константы (десятичные числа), заносимые ассемблером в ячейки, будем записывать после обозначения DN, например, DN 19. Правильность значения константы (для «Крохи» она должна быть целой и неотрицательной, не превышающей 4095) контролировать не будем.
- Так как практическая эксплуатация нашего программного продукта не предполагается, диагностику ошибок в нем предусматривать не будем (известно, что это часто наиболее трудоемкая часть программы!)
- Вместо длинных 12-разрядных двоичных кодов, которые вполне оправданы в учебнике для демонстрации принципов Неймана, везде будем пользоваться более короткими восьмеричными кодами; подчеркнем, что данная система прекрасно гармонирует с командами «Крохи», у которых и код операции, и адреса состоят именно из трех двоичных разрядов.

Благодаря принятым упрощениям ассемблирующая программа на Паскале получается совсем небольшой (ниже приводится ее листинг).

```
PROGRAM kroha_asm(INPUT,OUTPUT); {Демо-ассемблер для учебной ЭВМ «Кроха»}

CONST kop:ARRAY [0..7] OF STRING[3]= {мнемоники команд}
        ('MOV','ADD','DIV','SUB','JE','MUL','JG','HLT')
      prg:ARRAY [0..7] OF STRING=    {ассемблируемая программа!!!}
        ('B: MUL F K F',
         '   ADD K E K',
         '   JG  N K B',
         '   HLT F F F',
         'K: DN  1',
         'F: DN  1',
         'N: DN  3',
         'E: DN  1');
VAR   tab: ARRAY [0..15] OF RECORD sym:STRING[3]; num:BYTE END;
      {tab – это таблица всех идентификаторов (их имя и код), включая операции и метки}
      i,Nid,k,e,c:INTEGER; {рабочие переменные}

FUNCTION get_code(i:INTEGER; VAR k:INTEGER):BYTE;
{выделяет из строки с номером i начиная с позиции k очередной идентификатор
 и находит в tab его код}
VAR p,id:STRING; j,m,q:INTEGER; {рабочие переменные}
BEGIN p:=prg[i]+' '; {добавим пробел для удобства выделения последней метки}
      WHILE (k<LENGTH(p))AND(p[k]=' ') DO k:=k+1; {пропустим пробелы}
      {а теперь выберем все до следующего пробела, т.е. получим имя идентификатора}
      id:=''; WHILE p[k]<>' ' DO BEGIN id:=id+p[k]; k:=k+1 END;
      q:=255; {найдем, если идентификатор есть, в таблице и поместим его код в q}
      FOR j:=0 TO Nid DO IF id=tab[j].sym THEN q:=tab[j].num;
      get_code:=q {результатом функции является код, заносимый в программу}
END

BEGIN FOR i:=0 TO 7 DO {занесем в tab коды операций}
        BEGIN tab[i].sym:=kop[i];tab[i].num:=i
        END;
      tab[8].sym:='DN';tab[8].num:=8; Nid:=8; {добавим условный код DN}
      FOR i:=0 TO 7 DO {проход I – занесение в таблицу меток программиста}
        IF prg[i][2]=':' THEN {после метки двоеточие}
          BEGIN inc(Nid); {добавляем метку в таблицу; ее адрес=номеру строки!}
              tab[Nid].sym:=prg[i][1];tab[Nid].num:=i;
              prg[i][1]:=' '; prg[i][2]:=' '; {удалим метку из текста}
```

```
        END;
    FOR i:=0 TO Nid DO {для контроля выведем tab}
      WRITELN(tab[i].sym:4, tab[i].num:4);
    FOR i:=0 TO 7 DO {проход II – «генерация» кода на экран}
      BEGIN k:=1; c:=get_code(i,k); {код операции}
          CASE c OF {выделяем команды или DN}
          0..7: BEGIN WRITE(c); {<код операции}
                  WHILE k<=LENGTH(prg[i]) DO {3 адреса}
                     WRITE(' ',get_code(i,k))
              END;
          8:{DN}BEGIN {выделить 10-число и перевесит в 8 с/с}
                  VAL(COPY(prg[i],k,255),k,e);
                  WRITE('   '); e:=512;
                  REPEAT WRITE(k DIV e);
                       k:=k MOD e; e:=e DIV 8;
                  UNTIL e=0
              END
          END;
          WRITELN {конец вывода команды}
      END;
END
```

Программа работает следующим образом. Ядром ее данных служит таблица идентификаторов, которая содержит их имена в поле sym и коды, которые вместо этих имен надо подставлять в программу (поле num). Например, операция ADD имеет код 1 (см. табл. 4), поэтому tab[1].sym=’ADD’ и tab[1].num=1. Становится очевидным, что центральная идея ассемблера состоит в замене, пользуясь таблицей, каждого символьного имени соответствующим ему числовым кодом.

После занесения в строки таблицы с номерами от 0 до 7 мнемоник команд «Крохи», добавим далее оператор определения константы DN и все найденные в программе метки, которые поставил пользователь в тексте. Метки пользователя распознаются по наличию после них двоеточия; чтобы обработанные метки не мешали ассемблеру в дальнейшем, программа их просто стирает. Обработка меток в литературе носит название *первого прохода*, поскольку сначала надо «пройти» всю программу и подготовить **полную** таблицу идентификаторов, и только потом повторно просмотреть текст (*второй проход*), заменяя с помощью таблицы все идентификаторы их кодами.

При выполнении второго прохода наш ассемблер опирается на «мощную» функцию get_code, которая выделяет из текста программы очередной идентификатор. Несмотря на сложный вид, идея функции достаточно проста: в заданной строке, начиная с заданной позиции, она пропускает пробелы, а затем, наоборот, «собирает» все отличающиеся от пробела символы, формируя тем самым имя очередного идентификатора. Когда имя выделено, оно ищется в таблице и в качестве результата выдается его код (или 255, если имя в таблице отсутствует).

Примечание. Особо отметим, что алгоритм функции построен так, чтобы автоматически подготовить значение переменной k для нового вызова с целью выделения следующего в этой строке идентификатора.

Пользуясь описанной функцией, ассемблер выделяет первый идентификатор текущей строки программы и определяет его код. Если он попадает в диапазон от 0 до 7, то это машинная команда и в ней с помощью таблицы кодируются идущие следом 3 метки переменных (3 адреса). Если же обнаруживается, что код равен 8, то это определение числа. Последнее извлекается из строки и переводится в восьмеричную систему счисления. Все остальные случаи являются ошибкой и нашим ассемблером просто игнорируются.

Заметим, что итоговый восьмеричный код программы просто выводится на экран дисплея и нигде не сохраняется.

Для тестирования демо-ассемблера реализована традиционная задача – вычисление

факториала числа n. Обозначим k рабочую переменную, которая является текущим множителем для факториала и меняется от 1 до n. Тогда итоговая программа приобретает вид, приведенный в табл. 5.

Таблица 5

Адрес	**Команда**	**Расшифровка**	**Комментарий**
0	5545	(5) * (4) ==> (5)	n! * k ==> n!
1	1474	(4) + (7) ==> (4)	k + 1 ==> k
2	6640	если (6)>(4), перейти к 0	переход при k < n+1
3	7555	стоп; вывод (5), (5), (5)	вывод n!
4	k [задать 1]		рабочая ячейка
5	n! [задать 1]		результат
6	n + 1 [задать]		константа
7	0001	1	константа 1

Второй столбец данной таблицы может быть использован для проверки результатов ассемблирования, которые наша программа выводит на экран.

В качестве продолжения нашего эксперимента предлагаем читателям самостоятельно написать обратную программу – дизассемблер, которая по восьмеричному 4-разрядному коду восстанавливает ассемблерную мнемонику команд. Для решения задачи советуем воспользоваться таблицей, аналогичной tab, если потребуется, разделив ее на части. Разумеется, первоначальные имена меток по коду не восстановить, поэтому можно присваивать им по мере появления последовательные значения букв латинского алфавита.

А теперь подведем итоги того, что мы узнали.

В основе ассемблирования программы лежит относительно простая идея табличной замены имен идентификаторов соответствующими им кодами. В то же время реализация этой идеи ведет к замечательным последствиям: в программе исчезают **все** адреса и ее исправление становится не сложнее, чем исправление текста в текстовом редакторе: поскольку адреса переменных и команд распределяются теперь автоматически, следить за их изменением вовсе не нужно. Еще раз подчеркнем, что именно в этом, а не в замене кодов операций и регистров буквенными мнемониками, заключается истинная мощь ассемблера. В отладчике Debug, предназначенном в основном для отладки уже готовой программы, поддержка механизма меток не предусмотрена.

Подчеркнем, что ассемблер – это язык программирования, хотя и весьма низкого уровня, жестко ориентированный на систему команд того процессора, для которого он предназначается. Очень важно отметить, что каждая исполняемая строка ассемблерного текста в точности соответствует **одной** машинной команде, что является характерной особенностью ассемблера. Отказ от данного ограничения и разрешение автоматической замены строки программы **серией команд** открывает дорогу (через так называемый *макроассемблер*) к простейшим языкам высокого уровня. В последних текст программы уже больше не содержит прямых ссылок на инструкции процессора, следовательно, сам язык становится машинно-независимым. Генерацию конкретных машинных инструкций полностью принимает на себя программное обеспечение – транслятор языка, который в традиционных реализациях является машинно-зависимым

Контрольные вопросы.

1. Как хранится и обрабатывается информация в компьютере?
2. Что такое мнемоника?
3. Что такое директива?
4. Что лежит в основе ассемблирования программы?

Практическая работа №3. Изучение принципов работы ПК.

Цель работы: изучение общих принципов организации и работы компьютеров.

Теоретическая часть

Компьютер (англ. computer — вычислитель) представляет собой программируемое электронное устройство, способное обрабатывать данные и производить вычисления, а также выполнять другие задачи манипулирования символами [51] .

Существует два основных класса компьютеров:

- **цифровые компьютеры,** обрабатывающие данные в виде двоичных кодов;
- **аналоговые компьютеры,** обрабатывающие непрерывно меняющиеся физические величины (электрическое напряжение, время и т.д.), которые являются аналогами вычисляемых величин.

Поскольку в настоящее время подавляющее большинство компьютеров являются цифровыми, далее будем рассматривать только этот класс компьютеров и слово "*компьютер*" употреблять в значении "*цифровой компьютер*".

Основу компьютеров образует аппаратура (HardWare), построенная, в основном, с использованием электронных и электромеханических элементов и устройств. Принцип действия компьютеров состоит в выполнении программ (SoftWare) — заранее заданных, четко определённых последовательностей арифметических, логических и других операций.

Любая компьютерная программа представляет собой последовательность отдельных команд.

Команда — это описание операции, которую должен выполнить компьютер. Как правило, у команды есть свой *код* (условное обозначение), *исходные данные* (операнды) и *результат*.

Например, у команды "*сложить два числа*" операндами являются слагаемые, а результатом — их сумма. А у команды "*стоп*" операндов нет, а результатом является прекращение работы программы.

Результат команды вырабатывается по точно определенным для данной команды правилам, заложенным в конструкцию компьютера.

Совокупность команд, выполняемых данным компьютером, называется системой команд этого компьютера.

Компьютеры работают с очень высокой скоростью, составляющей миллионы — сотни миллионов операций в секунду.

Разнообразие современных компьютеров очень велико. Но их структуры основаны на общих логических принципах, позволяющих выделить в любом компьютере следующие главные устройства:

- память (запоминающее устройство, ЗУ), состоящую из перенумерованных ячеек;
- процессор, включающий в себя устройство управления (УУ) и арифметико-логическое устройство (АЛУ);
- устройство ввода;
- устройство вывода.

Эти устройства соединены каналами связи, по которым передается информация.

Основные устройства компьютера и связи между ними представлены на схеме (рис. 1). Жирными стрелками показаны пути и направления движения информации, а простыми стрелками — пути и направления передачи управляющих сигналов.

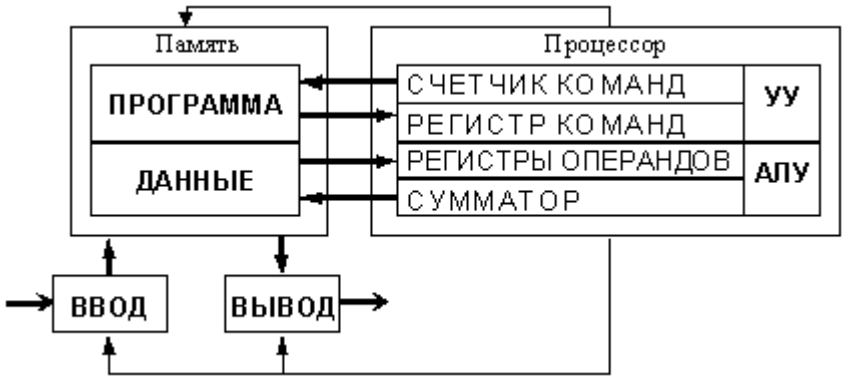


Рис. 1. Общая схема компьютера

Функции памяти:

- приём информации из других устройств;
- запоминание информации;
- выдача информации по запросу в другие устройства машины.

Функции процессора:

- обработка данных по заданной программе путем выполнения арифметических и логических операций;
- программное управление работой устройств компьютера.

Та часть процессора, которая выполняет команды, называется арифметико-логическим устройством (АЛУ), а другая его часть, выполняющая функции управления устройствами, называется устройством управления (УУ).

Обычно эти два устройства выделяются чисто условно, конструктивно они не разделены.

В составе процессора имеется ряд специализированных дополнительных ячеек памяти, называемых регистрами.

Регистр выполняет функцию кратковременного хранения числа или команды. Над содержимым некоторых регистров специальные электронные схемы могут выполнять некоторые манипуляции. Например, "вырезать" отдельные части команды для последующего их использования или выполнять определенные арифметические операции над числами.

Основным элементом регистра является электронная схема, называемая **триггером,** которая способна хранить одну двоичную цифру (разряд двоичного кода). Логическая схема триггера описана в разделе 5.7.

Регистр представляет собой совокупность триггеров, связанных друг с другом определённым образом общей системой управления.

Существует несколько типов регистров, отличающихся видом выполняемых операций. Некоторые важные регистры имеют свои названия, например:

- сумматор — регистр АЛУ, участвующий в выполнении каждой операции;
- счетчик команд — регистр УУ, содержимое которого соответствует адресу очередной выполняемой команды; служит для автоматической выборки программы из последовательных ячеек памяти;
- регистр команд — регистр УУ для хранения кода команды на период времени, необходимый для ее выполнения. Часть его разрядов используется для хранения кода операции, остальные — для хранения кодов адресов операндов.

В основу построения подавляющего большинства компьютеров положены следующие общие принципы, сформулированные в 1945 г. американским ученым Джоном фон Нейманом.

1. Принцип программного управления. Из него следует, что программа состоит из набора команд, которые выполняются процессором автоматически друг за другом в определенной последовательности.

Выборка программы из памяти осуществляется с помощью счетчика команд. Этот регистр процессора последовательно увеличивает хранимый в нем адрес очередной команды на длину

команды.

А так как команды программы расположены в памяти друг за другом, то тем самым организуется выборка цепочки команд из последовательно расположенных ячеек памяти.

Если же нужно после выполнения команды перейти не к следующей, а к какой-то другой, используются команды условного илибезусловного переходов, которые заносят в счетчик команд номер ячейки памяти, содержащей следующую команду. Выборка команд из памяти прекращается после достижения и выполнения команды *"стоп"*.

Таким образом, процессор исполняет программу автоматически, без вмешательства человека.

2. Принцип однородности памяти. Программы и данные хранятся в одной и той же памяти. Поэтому компьютер не различает, что хранится в данной ячейке памяти — число, текст или команда. Над командами можно выполнять такие же действия, как и над данными. Это открывает целый ряд возможностей. Например, программа в процессе своего выполнения также может подвергаться переработке, что позволяет задавать в самой программе правила получения некоторых ее частей (так в программе организуется выполнение циклов и подпрограмм). Более того, команды одной программы могут быть получены как результаты исполнения другой программы. На этом принципе основаны методы трансляции — перевода текста программы с языка программирования высокого уровня на язык конкретной машины.

3. Принцип адресности. Структурно основная память состоит из перенумерованных ячеек; процессору в произвольный момент времени доступна любая ячейка. Отсюда следует возможность давать имена областям памяти, так, чтобы к запомненным в них значениям можно было впоследствии обращаться или менять их в процессе выполнения программ с использованием присвоенных имен.

Компьютеры, построенные на этих принципах, относятся к типу фон-неймановских. Но существуют компьютеры, принципиально отличающиеся от фон-неймановских. Для них, например, может не выполняться принцип программного управления, т.е. они могут работать без "счетчика команд", указывающего текущую выполняемую команду программы. Для обращения к какой-либо переменной, хранящейся в памяти, этим компьютерам не обязательно давать ей имя. Такие компьютеры называются не-фон-неймановскими.

Команда — это описание элементарной операции, которую должен выполнить компьютер.

В общем случае, команда содержит следующую информацию:

- код выполняемой операции;
- указания по определению операндов (или их адресов);
- указания по размещению получаемого результата.

В зависимости от количества операндов, команды бывают:

- одноадресные;
- двухадресные;
- трехадресные;
- переменноадресные.

Команды хранятся в ячейках памяти в двоичном коде.

В современных компьютерах длина команд переменная (обычно от двух до четырех байтов), а способы указания адресов переменных весьма разнообразные. В адресной части команды может быть указан, например:

- сам операнд (число или символ);
- адрес операнда (номер байта, начиная с которого расположен операнд);
- адрес адреса операнда (номер байта, начиная с которого расположен адрес операнда), и др.

Рассмотрим несколько возможных вариантов команды сложения (англ. add — сложение), при этом вместо цифровых кодов и адресов будем пользоваться условными обозначениями:

- **одноадресная команда *add x*** (содержимое ячейки ***x*** сложить с содержимым сумматора, а результат оставить в сумматоре)

dd	

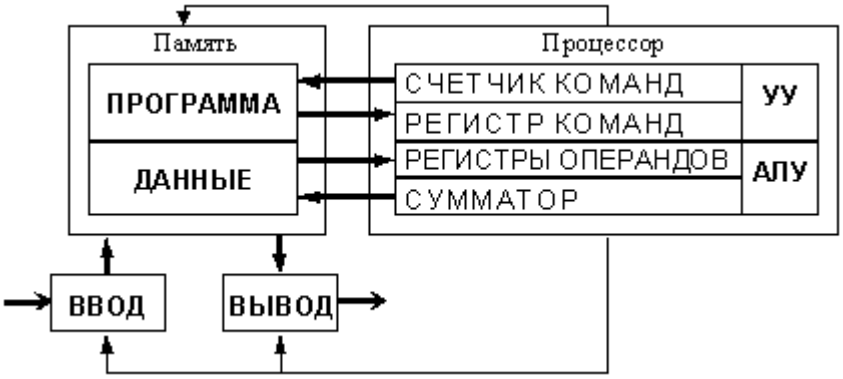
двухадресная команда *add x, y* (сложить содержимое ячеек ***x*** и ***y***, а результат поместить в ячейку ***y***)

dd		

трехадресная команда *add x, y, z* (содержимое ячейки ***x*** сложить с содержимым ячейки ***y***, сумму поместить в ячейку ***z)***

dd			

Выполнение команды можно проследить по схеме:



Общая схема компьютера

Как правило, этот процесс разбивается на следующие этапы:

- из ячейки памяти, адрес которой хранится в счетчике команд, выбирается очередная команда; содержимое счетчика команд при этом увеличивается на длину команды;
- выбранная команда передается в устройство управления на регистр команд;
- устройство управления расшифровывает адресное поле команды;
- по сигналам УУ операнды считываются из памяти и записываются в АЛУ на специальные регистры операндов;
- УУ расшифровывает код операции и выдает в АЛУ сигнал выполнить соответствующую операцию над данными;
- результат операции либо остается в процессоре, либо отправляется в память, если в команде был указан адрес результата;
- все предыдущие этапы повторяются до достижения команды “стоп”.

При рассмотрении компьютерных устройств принято различать их архитектуру и структуру.

Архитектурой компьютера называется его описание на некотором общем уровне, включающее описание пользовательских возможностей программирования, системы команд, системы адресации, организации памяти и т.д. Архитектура определяет принципы действия, информационные связи и взаимное соединение основных логических узлов компьютера: процессора, оперативного ЗУ, внешних ЗУ и периферийных устройств. Общность архитектуры разных компьютеров обеспечивает их совместимость с точки зрения пользователя.

Структура компьютера — это совокупность его функциональных элементов и связей между ними. Элементами могут быть самые различные устройства — от основных логических узлов компьютера до простейших схем. Структура компьютера графически представляется в виде структурных схем, с помощью которых можно дать описание компьютера на любом уровне детализации.

Наиболее распространены следующие архитектурные решения.

Классическая архитектура (архитектура фон Неймана) — одно арифметико-логическое устройство (АЛУ), через которое проходит поток данных, и одно устройство управления (УУ), через которое проходит поток команд — программа (рис. 1). Это однопроцессорный компьютер. К этому типу архитектуры относится и архитектура персонального компьютера с . Все функциональные блоки здесь связаны между собой общей шиной, называемой также системной магистралью.

Физически магистраль представляет собой многопроводную линию с гнездами для подключения электронных схем. Совокупность проводов магистрали разделяется на отдельные группы: шину адреса, шину данных и шину управления.

Периферийные устройства (принтер и др.) подключаются к аппаратуре компьютера через специальные контроллеры — устройства управления периферийными устройствами.

Контроллер — устройство, которое связывает периферийное оборудование или каналы связи с центральным процессором, освобождая процессор от непосредственного управления функционированием данного оборудования.

· **Многопроцессорная архитектура**. Наличие в компьютере нескольких процессоров означает, что параллельно может быть организовано много потоков данных и много потоков команд. Таким образом, параллельно могут выполняться несколько фрагментов одной задачи. Структура такой машины, имеющей общую оперативную память и несколько процессоров, представлена на рис. 2.3.

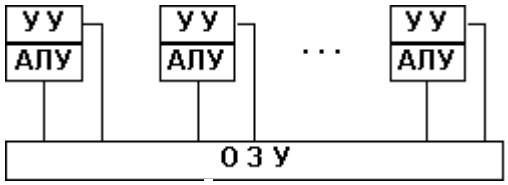


Рис..3. Архитектура многопроцессорного компьютера

Многомашинная вычислительная система. Здесь несколько процессоров, входящих в вычислительную систему, не имеют общей оперативной памяти, а имеют каждый свою (локальную). Каждый компьютер в многомашинной системе имеет классическую архитектуру, и такая система применяется достаточно широко. Однако эффект от применения такой вычислительной системы может быть получен только при решении задач, имеющих очень специальную структуру: она должна разбиваться на столько слабо связанных подзадач, сколько компьютеров в системе.

Преимущество в быстродействии многопроцессорных и многомашинных вычислительных систем перед однопроцессорными очевидно.

Архитектура с параллельными процессорами. Здесь несколько АЛУ работают под управлением одного УУ. Это означает, что множество данных может обрабатываться по одной программе — то есть по одному потоку команд. Высокое быстродействие такой архитектуры можно получить только на задачах, в которых одинаковые вычислительные операции выполняются одновременно на различных однотипных наборах данных. Структура таких компьютеров представлена на рис. 2.4.

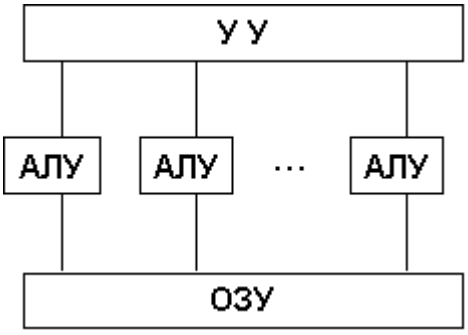


Рис. 2.4. Архитектура с параллельным процессором

В современных машинах часто присутствуют элементы различных типов архитектурных решений. Существуют и такие архитектурные решения, которые радикально отличаются от рассмотренных выше.

Контрольные вопросы

1. Какова роль аппаратуры (HardWare) и программного обеспечения (SoftWare) компьютера?

2. Какие основные классы компьютеров Вам известны?

3. В чём состоит принцип действия компьютеров?

4. Из каких простейших элементов состоит программа?

5. Что такое система команд компьютера?

6. Перечислите главные устройства компьютера.

7. Опишите функции памяти и функции процессора.

8. Назовите две основные части процессора. Каково их назначение?

9. Что такое регистры? Назовите некоторые важные регистры и опишите их функции.

10. Сформулируйте общие принципы построения компьютеров.

11. В чём заключается принцип программного управления? Как выполняются команды условных и безусловных переходов?

12. В чём суть принципа однородности памяти? Какие возможности он открывает?

13. В чём заключается принцип адресности?

14. Какие архитектуры называются "фон-неймановскими"?

15. Что такое команда? Что описывает команда?

16. Какого рода информацию может содержать адресная часть команды?

17. Приведите примеры команд одноадресных, двухадресных, трёхадресных.

18. Каким образом процессор при выполнении программы осуществляет выбор очередной команды?

19. Опишите основной цикл процесса обработки команд.

20. Что понимается под архитектурой компьютера?

21. Что понимается под структурой компьютера? Какой уровень детализации описания компьютера может она обеспечить?

22. Перечислите распространённые компьютерные архитектуры.

23. Каковы отличительные особенности классической архитектуры?

24. Что собой представляет шина компьютера? Каковы функции общей шины (магистрали)?

25. Какую функцию выполняют контроллеры?

26. Как характер решаемых задач связан с архитектурой компьютера?

27. Какие отличительные особенности присущи многопроцессорной архитектуре? Многомашинной архитектуре? Архитектуре с параллельным процессором?

28. Что такое центральный процессор?

29. Какие основные компоненты содержат в себе современные микропроцессоры?

Практическая работа №4. Изучение принципов работы микропроцессорных систем.

Цель работы: изучить структуру МПС и принцип ее работы.

Теоретическая часть.

Структура любой микропроцессорной системы является ***магистрально-модульной.*** Это означает, что в ней можно выделить набор модулей – устройств, подключенных к общим магистралям, называемых шинами. Под***шиной*** понимают набор линий связи, по которым передается информация определенного типа, осуществляется обмен информацией между различными модулями системы.

Обобщенная структура микропроцессорной системы представлена на рис. 1.

Любую микропроцессорную систему можно представить как микроЭВМ и набор ВУ. Под ВУ понимают устройства двух типов:

• устройства ввода/вывода информации, обеспечивающие вычислительный процесс и связь с оператором (монитор, клавиатура, внешние запоминающие устройства и т.д.);

• устройства, обеспечивающие управление техническими средствами технологического оборудования, станками и т.п.

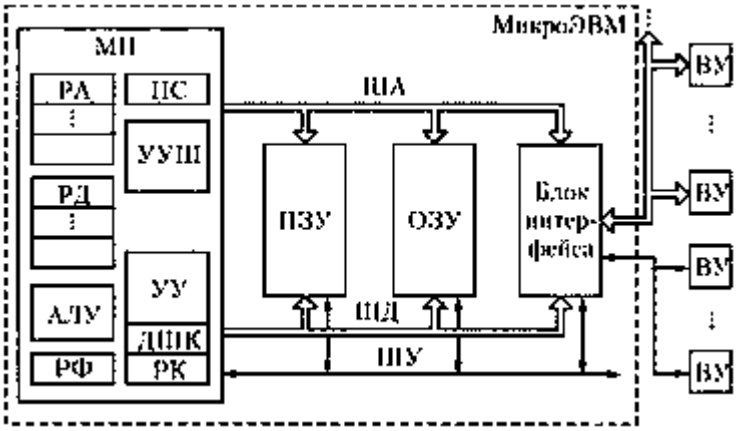


Рис. 1. **Структура микропроцессорной системы**

Кроме МП, который также называют ***центральным процессорным элементом,*** в состав микроЭВМ входят ПЗУ, ОЗУ и ***блок интерфейса.*** ПЗУ обеспечивает хранение неизменяемых программ работы системы. Если это универсальная система типа персонального компьютера, то в ПЗУ хранится программа базовой системы ввода/вывода, обеспечивающая функционирование и начальную загрузку системы – инициализацию. Если это специализированная система, типа устройства числового программного управления, то в ПЗУ заносится все программное обеспечение системы. ОЗУ предназначено для хранения информации, которая может изменяться в процессе работы системы. Это могут быть данные, промежуточные результаты вычислений и программы, исполняемые в текущий момент времени. В простых системах это только входная информация и промежуточные результаты.

Весь обмен информацией МП с ВУ осуществляется через блок интерфейса. ВУ передают данные из внешней среды в МП или ОЗУ или получают их из микроЭВМ. Для подключения ВУ к микропроцессорной системе его сигналы, скорость передачи информации, формат слов необходимо привести к стандартному виду, с которым работает МП. Все эти преобразования данных выполняются в интерфейсном блоке. Фактически блок интерфейса это набор различных узлов – ***адаптеров*** и ***контроллеров.*** Сложные ВУ, типа монитора или накопителей на магнитных дисках подключаются через контроллеры ВУ, которые обеспечивают не только преобразование данных, но и управление самими ВУ. Они на структурной схеме не показаны.

Взаимодействие узлов микроЭВМ между собой осуществляется с помощью трех шин: ***шины адреса*** (Ш А),***шины данных*** (ШД) и ***шины управления*** (ШУ). Чтобы МП мог однозначно выбрать нужную ячейку памяти или регистр ВУ, они имеют адреса. Адрес ячейки (регистра) передается от МП в память или интерфейсный блок по ША. ША однонаправленная, так как направление передачи информации по ней только одно – из МП. В отличие от нее ШД является двунаправленной, так как передача данных по ней осуществляется как из МП в память и интерфейс, так и наоборот. ША и ШД состоят из параллельных линий, передача информации по которым осуществляется одновременно для всех линий (поэтому на рисунке эти шины обозначены широкими стрелками). Число линий ШД определяется разрядностью МП, а ША – объемом памяти, т.е. разрядностью двоичного кода, необходимого для адресации всех ячеек. ШУ состоит из отдельных линий, по которым передаются те или иные управляющие сигналы. Естественно, что они передаются не одновременно, поэтому на рис. 4.9 ШУ обозначена узкими стрелками. В основном это сигналы, передаваемые из МП в остальные узлы, но некоторые имеют обратную направленность – в МП. Примером первых могут служить сигналы чтения и записи, указывающие, какую именно следует выполнять операцию с ячейкой, адрес которой выставлен на ША. Ко вторым относят осведомительные сигналы запроса обслуживания, поступающие от ВУ, а

также сигнал сброса МП в начальное (нулевое) состояние.

Внешние устройства в зависимости от способа передачи информации разделяются на две большие группы: устройства, обменивающиеся параллельными словами данных (на рис. 1 они подключены к параллельной шине), и устройства, обменивающиеся информацией в последовательном коде, т.е. последовательно, бит за битом (подключены к однопроводной шине, обозначенной узкой стрелкой).

Основными узлами МП являются ***устройство управления*** (УУ), ***регистр команд*** (РК), ***дешифратор команд*** (ДШК), ***арифметико-логическое устройство*** (АЛУ), ***регистр флажков*** (РФ), набор ***внутренних регистров,*** разделяемых на ***адресные регистры*** (РА) и ***регистры данных*** (РД), ***программный счетчик*** (ПС), ***устройство управления шинами*** (УУШ).

Координация работы всех узлов в соответствии с выполняемой командой осуществляется тремя узлами: УУ, РК и ДШК. РК обеспечивает хранение команды в течение всего цикла ее исполнения, а ДШК выполняет расшифровку кода этой команды. УУ вырабатывает серию импульсов, обеспечивающих последовательное и слаженное срабатывание узлов МП в соответствии с выполняемой командой. Для выработки управляющих импульсов на вход УУ поступают импульсы синхронизации от внешнего генератора. Такой генератор может быть также встроен в УУ. Кроме управления внутренними узлами, УУ обеспечивает прием и выдачу внешних управляющих сигналов.

АЛУ обеспечивает выполнение всех операций, с помощью которых осуществляется переработка данных в МП. Оно может выполнять несложные арифметические, логические и сдвиговые операции. Количество ***операндов,*** т.е. двоичных чисел, над которыми выполняются действия в АЛУ, может колебаться от одного до двух. Например, при инвертировании (логическое НЕ) АЛУ достаточно одного операнда, а для операции сложения двух чисел необходимо два операнда. Перечень операций, выполняемых АЛУ, зависит от типа МП. Для большинства МП в АЛУ выполняются следующие операции: сложение, вычитание, логические И, ИЛИ, НЕ, исключающее ИЛИ (сумма по модулю 2), сдвиг вправо, сдвиг влево, сложение с единицей (инкремент), вычитание единицы (декремент). Сложные арифметические операции, такие как умножение и деление, АЛУ не выполняет. В зависимости от результата операции АЛУ формирует признаки результата, называемые ***флажками.*** Эти признаки используются не в текущей, а в последующих командах, поэтому для их хранения в МП используется РФ.

Регистры – составная и очень важная часть МП. Каждый регистр МП можно использовать для временного хранения одного слова данных. Некоторые регистры имеют специальное назначение, другие – многоцелевое. Внутренние РА и РД являются внутренней памятью МП. РА используются для временного хранения двоичных чисел, с помощью которых МП вычисляет адреса ячеек памяти, к которым он обращается в процессе работы. РД используются как для непосредственного хранения операндов, так и для вычисления адресов ячеек ОЗУ, хранящих операнды. Через РД также осуществляется обмен информацией между МП и ВУ. Программный счетчик служит для хранения адреса ячейки памяти, в которой хранится очередная исполняемая команда программы.

Выполняя программу, МП обрабатывает команду за командой, которые обычно располагаются в ячейках памяти последовательно одна за другой. Команда задает выполняемую операцию и содержит сведения, где находятся операнды. Выполнение команды можно разбить на две фазы: фазу выборки команды и фазу ее исполнения. Первая фаза начинается с того, что МП выставляет на ША содержимое ПС, хранящего адрес ячейки памяти с очередной командой. Содержимое ячейки выставляется на ШД, МП считывает информацию с ШД и помещает команду в РК.

Вторая фаза заключается в собственно выполнении команды. При этом сначала МП должен подготовить операнды. Операнды могут храниться как в самом МП, так и в ОЗУ. В первом случае они хранятся в регистрах данных, и МП может переходить к непосредственному исполнению математической или логической операции в соответствии с кодом команды. Во втором случае МП должен сначала вычислить адрес ячейки ОЗУ, хранящей операнд, потом выставить этот адрес на ША и считать содержимое указанной ячейки ОЗУ, и только затем выполнить операцию. Выполнение операции осуществляется в АЛУ, после чего результат должен

быть помещен на место первого операнда. Если это один из внутренних регистров МП, результат сразу же переписывается в этот регистр, если это ячейка ОЗУ, требуется еще один цикл обращения к памяти. Таким образом время исполнения команды зависит от количества циклов обращения к памяти, и самыми короткими являются те команды, в которых операнды хранятся непосредственно в МП.

Во время выполнения команды при каждом обращении МП к памяти программ содержимое ПС автоматически увеличивается на единицу. Команды могут занимать не только одну ячейку памяти, а две и даже три, при этом, чтобы считать всю команду, МП должен несколько раз обратиться к памяти программ. В результате в конце выполнения команды в ПС уже хранится адрес следующей, и МП готов к выполнению очередной команды. Отсюда и название этого регистра – "программный счетчик".

Регистр ПС хранит адрес следующей выполняемой команды только в случае естественного порядка следования команд программы – команда за командой. В случае разветвления алгоритма в зависимости от выполнения или невыполнения заданного условия необходимо идти но одной из двух ветвей программы. Такие разветвления выполняются с помощью команд условного перехода. Для этого в команде условного перехода задается проверяемое условие и указывается адрес команды, подлежащей исполнению в случае выполнения условия. При невыполнении условия сохраняется естественный порядок следования команд, т.е. выполняется следующая по порядку команда. Так как адресация осуществляется через программный счетчик, то при выполнении заданного условия в ПС загружается адрес, указанный в команде, если же условие не выполняется, то адрес следующей команды оказывается уже сформированным в ПС. Проверка тех или иных условий в МП обычно заключается в анализе признаков результата, которые были сформированы при исполнении предыдущей команды и сохранены в регистре флажков.

В процессе работы МП постоянно обращается к ША и ШД. Передача информации внутри МП осуществляется по внутренним шинам, которые непосредственно не связаны с внешними шинами. Для передачи адресов и данных из МП во внешние шины и приема данных с ШД в МП необходимо буферное устройство, которым служит УУШ. В простейшем случае – это набор буферных регистров, управляемых УУ. Буферный регистр адреса принимает данные с внутренней шины и хранит его в течение цикла обращения к памяти или ВУ, при этом адрес через выходные каскады регистра выставляется на ША. Буферный регистр данных – двунаправленный и может как передавать данные с внутренней шины во внешнюю, так и принимать их с внешней ШД и передавать во внутреннюю. Эти регистры имеют третье состояние и переводятся в него, когда МП с ША и ШД не работает. В более сложных МП в состав УУШ, помимо буферных, входит набор внутренних регистров, некоторые адресные регистры и комбинационные схемы. Такое УУШ работает самостоятельно, обеспечивая взаимодействие МП с внешними шинами.

Контрольные вопросы

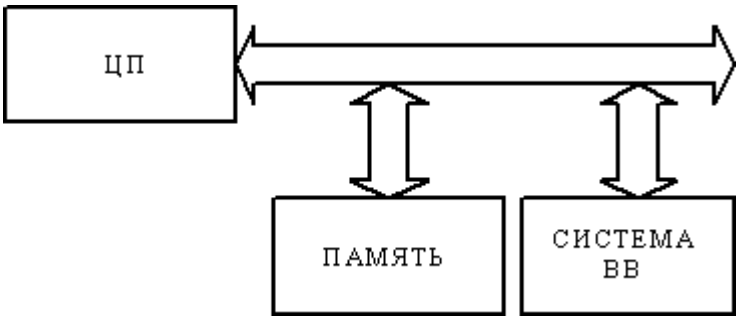
1. Какой вид имеет структура любой микропроцессорной системы?
2. Как осуществляется обмен информацией МП с ВУ?
3. Как подразделяются внешние устройства в зависимости от способа передачи информации?
4. Перечислите фазы исполнения команды.

Практическая работа №5. Архитектура микропроцессорных систем.

Цель работы: изучить существующие архитектурные решения микропроцессорных систем.

Теоретическая часть.

Базовая структура микропроцессорной системы имеем вид



Задача управления системой возлагается на **центральный процессор** (ЦП), который связан с **памятью** и **системой ввода-вывода** через каналы памяти и ввода-вывода соответственно. ЦП считывает из памяти команды, которые образуют программу, и декодирует их. В соответствии с результатом декодирования команд он осуществляет выборку данных из памяти м портов ввода, обрабатывает их и пересылает обратно в память или порты вывода. Существует также возможность ввода-вывода данных из памяти на внешние устройства и обратно, минуя ЦП. Этот механизм называется **прямым доступом к памяти** (ПДП). Каждая составная часть микропроцессорной системы имеет достаточно сложную внутреннюю структуру.

С точки зрения пользователя при выборе микропроцессора целесообразно располагать некоторыми обобщенными комплексными характеристиками возможностей микропроцессора. Разработчик нуждается в уяснении и понимании лишь тех компонентов микропроцессора, которые явно отражаются в программах и должны быть учтены при разработке схем и программ функционирования системы. Такие характеристики определяются понятием архитектуры микропроцессора.

Архитектура микропроцессора - это его логическая организация, рассматриваемая с точки зрения пользователя; она определяет возможности микропроцессора по аппаратной и программной реализации функций, необходимых для построения микропроцессорной системы. Понятие архитектуры микропроцессора отражает:

- его структуру, т. е. совокупность компонентов, составляющих микропроцессор, и связей между ними; для пользователя достаточно ограничиться регистровой моделью микропроцессора;
- способы представления и форматы данных;
- способы обращения ко всем программно-доступным для пользователя элементам структуры (адресация к регистрам, ячейкам постоянной и оперативной памяти, внешним устройствам);
- набор операций, выполняемых микропроцессором;
- характеристики управляющих слов и сигналов, вырабатываемых микропроцессором и поступающих в него извне;
- реакцию на внешние сигналы (система обработки прерываний и т. п.).

По способу организации пространства памяти микропроцессорной системы различают два основных типа архитектур.

Организация, при которой для хранения программ и данных используется одно пространство памяти, называется **фон Неймановской архитектурой** (по имени математика Джона фон Неймана (John von Neumann), предложившего кодирование программ в формате, соответствующем формату данных). Программы и данные хранятся в едином пространстве, и нет никаких признаков, указывающих на тип информации в ячейке памяти. Преимуществами такой архитектуры являются более простая внутренняя структура микропроцессора и меньшее количество управляющих сигналов. Примером микропроцессоров с такой архитектурой могут служить микропроцессоры для персональных компьютеров семейства x86.

Организация, при которой память программ CSEG (Code Segment) и память данных DSEG (Data Segment) разделены и имеют свои собственные адресные пространства и способы доступа к ним, называется **Гарвардской архитектурой** (по имени лаборатории Гарвардского Университета, предложившей ее). Такая архитектура является более сложной и требует дополнительных управляющих сигналов. Однако, она позволяет осуществлять более гибкие

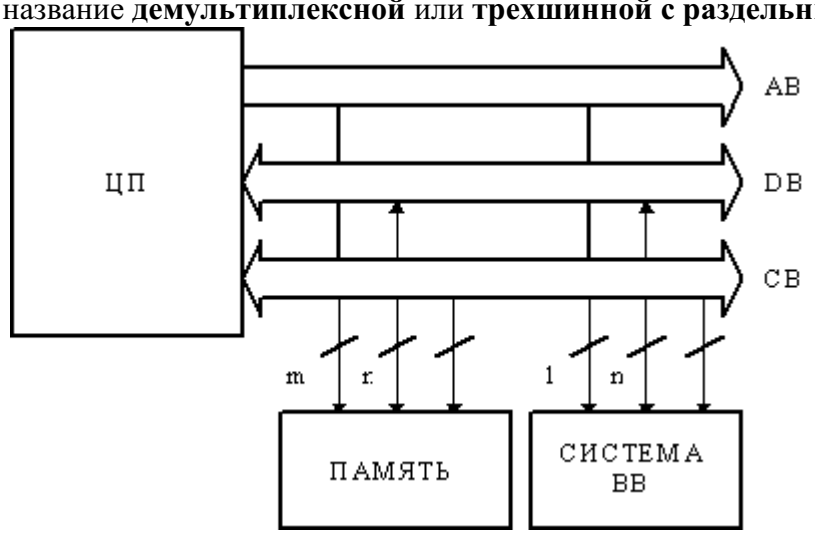
манипуляции информации, реализовывать компактно кодируемый набор машинных команд и, в ряде случаев, ускорять работу микропроцессора. Примером микропроцессоров с такой архитектурой могут служить микроконтроллеры семейства MCS-51.

В настоящее время выпускаются микропроцессоры со смешанной архитектурой, в которых CSEG и DSEG имеют единое адресное пространство, однако различные механизмы доступа к ним. Конкретным примером являются микропроцессоры семейства TMS320F2000 фирмы Texas Instruments.

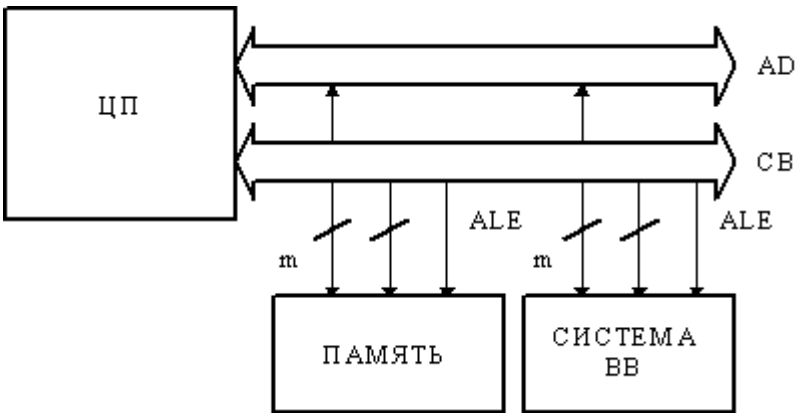
На физическом уровне микропроцессор взаимодействует с памятью и системой ввода-вывода через единый набор системных шин - **внутрисистемную магистраль**. Она, в общем случае состоит из:

- шины данных DB (Data Bus), по которой производится обмен данными между ЦП, памятью и системой ВВ;
- шины адреса АВ (Address Bus), используемой для передачи адресов ячеек памяти и портов ВВ, к которым осуществляется обращение;
- шины управления СВ (Control Bus), по которой передаются управляющие сигналы, реализующие циклы обмена информацией и управляющие работой системы.

Этот же набор шин применяется для организации канала ПДП. Магистраль такого типа носит название **демультиплексной** или **трехшинной с раздельными шинами адреса и данных**.

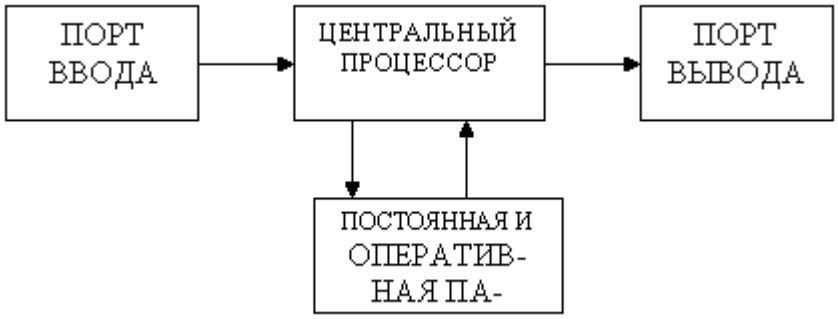


В некоторых микропроцессорах с целью сокращения ширины физической магистрали вводят **совмещеннную шину адреса-данных** AD (Address/Data Bus), по которой передаются как адреса так и данные. Этап передачи адресной информации отделен по времени от этапа передачи данных и стробируется специальным сигналом ALE (Address Latch Enable), который включен в состав СВ. Данную магистраль обычно называют **мультиплексной** или **двухшинной с совмещенными шинами адреса и данных**.

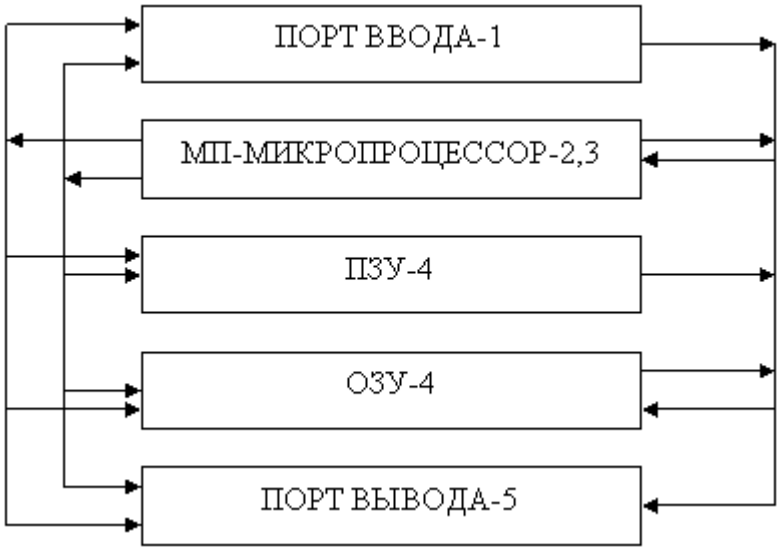


Архитектура МПС (функциональная схема) расшифровывает связи между устройствами цифровой техники и состоит из пяти основных элементов: 1 – устройства ввода, 2 – устройства

управления, 3 – устройства арифметических действий, которые входят в собственно микропроцессор, 4 – памяти, 5 – устройства вывода (рис. 1.2).



Как работает архитектура? МП контролирует все устройства и управляет ими посредством линий управления и контроля. Адресная шина (ША) однонаправленная выбирает ячейки памяти данных, команд, порты ввода или вывода данных.



Шина данных (ШД) является двунаправленной и служит для передачи данных в ЦП обработки информации или из него. Таким образом, видно, что ЦП может пересылать данные в память и получать их из нее посредством шины данных.

Когда программа помещается в памяти постоянно, она располагается в устройстве, называемом постоянным запоминающим устройством (ПЗУ). Ее невозможно изменить. Память изменяющихся данных, называемая оперативным запоминающим устройством (ОЗУ), также является отдельной интегральной схемой.

Программы пользователя, которые по своей природе изменяемы, также помещаются в ОЗУ вместе с данными.

Большинство микропроцессорных систем содержат эти характерные устройства, а также многие другие.

Контрольные вопросы.

1. В чем состоит различие архитектур МПС?
2. Как называется организация, при которой для хранения программ и данных используется одно пространство памяти?
3. Какая магистраль называется мультиплексной?
4. Как работает архитектура?

Практическая работа №6. Сегментная и страничная организация памяти. Функции системы.

Цель работы: изучить существующие методы распределения памяти.

Теоретическая часть

Структурно основная память состоит из пронумерованных ячеек. Процессору в произвольный момент времени доступна любая ячейка.

Память компьютера должна состоять из некоторого числа пронумерованных ячеек, в каждой из которых могут находиться или обрабатываемые данные, или инструкции программ. Все ячейки памяти должны быть одинаково легко доступны для других устройств компьютера.

Отсюда следует возможность давать имена областям памяти так, чтобы к сохраненным в них значениям можно было впоследствии обращаться или менять их в процессе выполнения программ с использованием присвоенных имен.

Сегментная, страничная и сегментно-страничная организация памяти

Методы распределения памяти, при которых задаче уже может не предоставляться сплошная (непрерывная) область памяти, называют разрывными. Идея выделять память задаче не одной сплошной областью, а фрагментами позволяет уменьшить фрагментацию памяти, однако этот подход требует для своей реализации больше ресурсов, он намного сложнее. Если задать адрес начала текущего фрагмента программы и величину смещения относительно этого начального адреса, то можно указать необходимую нам переменную или команду. Таким образом, виртуальный адрес можно представить состоящим из двух полей. Первое поле будет указывать на ту часть программы, к которой обращается процессор, для определения местоположения этой части в памяти, а второе поле виртуального адреса позволит найти нужную нам ячейку относительно найденного адреса. Программист может либо самостоятельно разбивать программу на фрагменты, либо можно автоматизировать эту задачу, возложив ее на систему программирования.

Сегментный способ организации виртуальной памяти

Первым среди разрывных методов распределения памяти был сегментный. Для этого метода программу необходимо разбивать на части и уже каждой такой части выделять физическую память. Естественным способом разбиения программы на части является разбиение ее на логические элементы — так называемые сегменты. В принципе, каждый программный модуль (или их совокупность, если мы того пожелаем) может быть воспринят как отдельный сегмент, и вся программа тогда будет представлять собой множество сегментов. Каждый сегмент размещается в памяти как до определенной степени самостоятельная единица. Логически обращение к элементам программы в этом случае будет состоять из имени сегмента и смещения относительно начала этого сегмента. Физически имя (или порядковый номер) сегмента будет соответствовать некоторому адресу, с которого этот сегмент начинается при его размещении в памяти, и смещение должно прибавляться к этому базовому адресу.

Преобразование имени сегмента в его порядковый номер осуществит система программирования. Для каждого сегмента система программирования указывает его объем. Он должен быть известен операционной системе, чтобы она могла выделять ему необходимый объем памяти. Операционная система будет размещать сегменты в памяти и для каждого сегмента она должна вести учет о местонахождении этого сегмента. Вся информация о текущем размещении сегментов задачи в памяти обычно сводится в таблицу сегментов, чаще такую таблицу называют таблицей дескрипторов сегментов задачи. Каждая задача имеет свою таблицу сегментов. Достаточно часто эти таблицы называют таблицами дескрипторов сегментов, поскольку по своей сути элемент таблицы описывает расположение сегмента.

Таким образом, виртуальный адрес для этого способа будет состоять из двух полей — номера сегмента и смещения относительно начала сегмента.

Итак, каждый сегмент, размещаемый в памяти, имеет соответствующую информационную структуру, часто называемую дескриптором сегмента. Именно операционная система строит для каждого исполняемого процесса соответствующую таблицу дескрипторов сегментов, и при размещении каждого из сегментов в оперативной или внешней памяти отмечает в дескрипторе текущее местоположение сегмента. Если сегмент задачи в данный момент находится в оперативной памяти, то об этом делается пометка в дескрипторе. Как правило, для этого используется бит присутствия P (от слова «present»). В этом случае в поле адреса диспетчер

памяти записывает адрес физической памяти, с которого сегмент начинается, а в поле длины сегмента (limit) указывается количество адресуемых ячеек памяти. Это поле используется не только для того, чтобы размещать сегменты без наложения друг на друга, но и для того, чтобы контролировать, не обращается ли код исполняющейся задачи за пределы текущего сегмента. В случае превышения длины сегмента вследствие ошибок программирования мы можем говорить о нарушении адресации и с помощью введения специальных аппаратных средств генерировать сигналы прерывания, которые позволят фиксировать (обнаруживать) такого рода ошибки.

Если бит присутствия в дескрипторе указывает, что сегмент находится не в оперативной, а во внешней памяти (например, на жестком диске), то названные поля адреса и длины используются для указания адреса сегмента в координатах внешней памяти. Помимо информации о местоположении сегмента, в дескрипторе сегмента, как правило, содержатся данные о его типе (сегмент кода или сегмент данный), правах доступа к этому сегменту (можно или нельзя его модифицировать, предоставлять другой задаче), отметка об обращениях к данному сегменту (информация о том, как часто или как давно этот сегмент используется или не используется, на основании которой можно принять решение о том, чтобы предоставить место, занимаемое текущим сегментом, другому сегменту).

При передаче управления следующей задаче операционная система должна занести в соответствующий регистр адрес таблицы дескрипторов сегментов этой задачи. Сама таблица дескрипторов сегментов, в свою очередь, также представляет собой сегмент данных, который обрабатывается диспетчером памяти операционной системы.

При таком подходе появляется возможность размещать в оперативной памяти не все сегменты задачи, а только задействованные в данный момент. Благодаря этому, с одной стороны, общий объем виртуального адресного пространства задачи может превосходить объем физической памяти компьютера, на котором эта задача будет выполняться; с другой стороны, даже если потребности в памяти не превосходят имеющуюся физическую память, можно размещать в памяти больше задач, поскольку любой задаче, как правило, все ее сегменты единовременно не нужны. А увеличение коэффициента мультипрограммирования ц, как мы знаем, позволяет увеличить загрузку системы и более эффективно использовать ресурсы вычислительной системы. Очевидно, однако, что увеличивать количество задач можно только до определенного предела, ибо если в памяти не будет хватать места для часто используемых сегментов, то производительность системы резко упадет. Ведь сегмент, находящийся вне оперативной памяти, для участия в вычислениях должен быть перемещен в оперативную память. При этом если в памяти есть свободное пространство, то необходимо всего лишь найти нужный сегмент во внешней памяти и загрузить его в оперативную память. А если свободного места нет, придется принять решение — на место какого из присутствующих сегментов будет загружаться требуемый. Перемещение сегментов из оперативной памяти на жесткий диск и обратно часто называют свопингом сегментов.

Итак, если требуемого сегмента в оперативной памяти нет, то возникает прерывание, и управление передается через диспетчер памяти программе загрузки сегмента. Пока происходит поиск сегмента во внешней памяти и загрузка его в оперативную, диспетчер памяти определяет подходящее для сегмента место. Возможно, что свободного места нет, и тогда принимается решение о выгрузке какого-нибудь сегмента и выполняется его перемещение во внешнюю память. Если при этом еще остается время, то процессор передается другой готовой к выполнению задаче. После загрузки необходимого сегмента процессор вновь передается задаче, вызвавшей прерывание из-за отсутствия сегмента. Всякий раз при считывании сегмента в оперативную память в таблице дескрипторов сегментов необходимо установить адрес начала сегмента и признак присутствия сегмента.

При поиске свободного места используется одна из вышеперечисленных дисциплин работы диспетчера памяти (применяются правила «первого подходящего» и «самого неподходящего» фрагментов). Если свободного фрагмента памяти достаточного объема нет, но, тем не менее, сумма этих свободных фрагментов превышает требования по памяти для нового сегмента, то в принципе может быть применено «уплотнение памяти», о котором мы уже говорили в подразделе «Разделы с фиксированными границами» раздела «Распределение памяти

статическими и динамическими разделами».

В идеальном случае размер сегмента должен быть достаточно малым, чтобы его можно было разместить в случайно освобождающихся фрагментах оперативной памяти, но достаточно большим, чтобы содержать логически законченную часть программы с тем, чтобы минимизировать межсегментные обращения.

Для решения проблемы замещения (определения того сегмента, который должен быть либо перемещен во внешнюю память, либо просто замещен новым) используются следующие дисциплины:

- правило FIFO (First In First Out — первый пришедший первым и выбывает);
- правило LRU (Least Recently Used — дольше других неиспользуемый);
- правило LFU (Least Frequently Used — реже других используемый);
- случайный (random) выбор сегмента.

Важнейшей проблемой, которая возникает при организации мультипрограммного режима, является защита памяти. Для того чтобы выполняющиеся приложения не смогли испортить саму операционную систему и другие вычислительные процессы, необходимо, чтобы доступ к таблицам сегментов с целью их модификации был обеспечен только для кода самой ОС. Для этого код операционной системы должен выполняться в некотором привилегированном режиме, из которого можно осуществлять манипуляции дескрипторами сегментов, тогда как выход за пределы сегмента в обычной прикладной программе должен вызывать прерывание по защите памяти. Каждая прикладная задача должна иметь возможность обращаться только к собственным и к общим сегментам.

При сегментном способе организации виртуальной памяти появляется несколько интересных возможностей.

Во-первых, при загрузке программы на исполнение можно размещать ее в памяти не целиком, а «по мере необходимости». Действительно, поскольку в подавляющем большинстве случаев алгоритм, по которому работает код программы, является разветвленным, а не линейным, то в зависимости от исходных данных некоторые части программы, расположенные в самостоятельных сегментах, могут быть не задействованы; значит, их можно и не загружать в оперативную память.

Во-вторых, некоторые программные модули могут быть разделяемыми. Поскольку эти программные модуля являются сегментами, относительно легко организовать доступ к таким общим сегментам. Сегмент с разделяемым кодом располагается в памяти в единственном экземпляре, а в нескольких таблицах дескрипторов сегментов исполняющихся задач будут находиться указатели на такие разделяемые сегменты.

Однако у сегментного способа распределения памяти есть и недостатки. Прежде всего для доступа к искомой ячейке памяти приходится тратить много времени. Мы должны сначала найти и прочитать дескриптор сегмента, а уже потом, используя полученные данные о местонахождении нужного нам сегмента, вычислить конечный физический адрес. Для того чтобы уменьшить эти потери, используется кэширование — те дескрипторы, с которыми мы имеем дело в данный момент, могут быть размещены в сверхоперативной памяти (специальных регистрах, размещаемых в процессоре).

Несмотря на то что рассмотренный способ распределения памяти приводит к существенно меньшей фрагментации памяти, нежели способы с неразрывным распределением, фрагментация остается. Кроме того, много памяти и процессорного времени теряется на размещение и обработку дескрипторных таблиц. Ведь на каждую задачу необходимо иметь свою таблицу дескрипторов сегментов. А при определении физических адресов приходится выполнять операции сложения, что требует дополнительных затрат времени.

Поэтому следующим способом разрывного размещения задач в памяти стал способ, при котором все фрагменты задачи считаются равными (одинакового размера), причем длина фрагмента в идеале должна быть кратна степени двойки, чтобы операции сложения можно было заменить операциями конкатенации (слияния). Это — страничный способ организации виртуальной памяти. Этот способ мы детально рассмотрим ниже.

Страничный способ организации памяти.

Как уже упоминалось, при страничном способе организации виртуальной памяти все фрагменты программы, на которые она разбивается (за исключением последней ее части), получаются одинаковыми. Одинаковыми полагаются и единицы памяти, которые предоставляются для размещения фрагментов программы. Эти одинаковые части называют страницами и говорят, что оперативная память разбивается на физические страницы, а программа — на виртуальные страницы. Часть виртуальных страниц задачи размещается в оперативной памяти, а часть — во внешней. Обычно место во внешней памяти, в качестве которой в абсолютном большинстве случаев выступают накопители на магнитных дисках (поскольку они относятся к быстродействующим устройствам с прямым доступом), называют файлом подкачки, или страничным файлом (paging file). Иногда этот файл называют swap-файлом, тем самым подчеркивая, что записи этого файла — страницы — замещают друг друга в оперативной памяти. В некоторых операционных системах выгруженные страницы располагаются не в файле, а в специальном разделе дискового пространства8.

Разбиение всей оперативной памяти на страницы одинаковой величины, причем кратной степени двойки, приводит к тому, что вместо одномерного адресного пространства памяти можно говорить о двухмерном. Первая координата адресного пространства — это номер страницы, вторая координата — номер ячейки внутри выбранной страницы (его называют индексом). Таким образом, физический адрес определяется парой (Pp, i), а виртуальный адрес — парой (Pv, i), где Pv — номер виртуальной страницы, Pp — номер физической страницы, i — индекс ячейки внутри страницы. Количество битов, отводимое под индекс, определяет размер страницы, а количество битов, отводимое под номер виртуальной страницы, — объем потенциально доступной для программы виртуальной памяти. Отображение, осуществляемое системой во время исполнения, сводится к отображению Pv в Pp и приписыванию к полученному значению битов адреса, задаваемых величиной i. При этом нет необходимости ограничивать число виртуальных страниц числом физических, то есть не поместившиеся страницы можно размещать во внешней памяти, которая в данном случае служит расширением оперативной.

Для отображения виртуального адресного пространства задачи на физическую память, как и в случае сегментного способа организации, для каждой задачи необходимо иметь таблицу страниц для трансляции адресных пространств. Для описания каждой страницы диспетчер памяти операционной системы заводит соответствующий дескриптор, который отличается от дескриптора сегмента прежде всего тем, что в нем нет поля длины — ведь все страницы имеют одинаковый размер. По номеру виртуальной страницы в таблице дескрипторов страниц текущей задачи находится соответствующий элемент (дескриптор). Если бит присутствия имеет единичное значение, значит данная страница размещена в оперативной, а не во внешней памяти, и мы в дескрипторе имеем номер физической страницы, отведенной под данную виртуальную. Если же бит присутствия равен нулю, то в дескрипторе мы будем иметь адрес виртуальной страницы, расположенной во внешней памяти. Таким образом и осуществляется трансляция виртуального адресного пространства на физическую память. Этот механизм трансляции иллюстрирует рис. 3.5.

Защита страничной памяти, как и в случае сегментного механизма, основана на контроле уровня доступа к каждой странице. Как правило, возможны следующие уровни доступа:

- только чтение;
- чтение и запись;
- только выполнение.

Каждая страница снабжается соответствующим кодом уровня доступа. При трансформации логического адреса в физический сравнивается значение кода разрешенного уровня доступа с фактически требуемым. При их несовпадении работа программы прерывается.

При обращении к виртуальной странице, не оказавшейся в данный момент в оперативной памяти, возникает прерывание, и управление передается диспетчеру памяти, который должен найти свободное место. Обычно предоставляется первая же свободная страница. Если свободной физической страницы нет, то диспетчер памяти по одной из вышеупомянутых дисциплин замещения (LRU, LFU, FIFO, случайный доступ) определит страницу, подлежащую расформированию или сохранению во внешней памяти. На ее месте он разместит новую

виртуальную страницу, к которой было обращение из задачи, но которой не оказалось в оперативной памяти.

Как и в случае с сегментным способом организации виртуальной памяти, страничный механизм приводит к тому, что без специальных аппаратных средств он существенно замедляет работу вычислительной системы. Поэтому обычно используется кэширование страничных дескрипторов. Наиболее эффективным механизмом кэширования является ассоциативный кэш. Именно такой ассоциативный кэш и создан в 32-разрядных микропроцессорах i80x86. Начиная с i80386, который поддерживает страничный способ распределения памяти, в этих микропроцессорах имеется кэш на 32 страничных дескриптора. Поскольку размер страницы в этих микропроцессорах равен 4 Кбайт, возможно быстрое обращение к памяти размером 128 Кбайт.

Итак, основным достоинством страничного способа распределения памяти является минимальная фрагментация. Поскольку на каждую задачу может приходиться по одной незаполненной странице, очевидно, что память можно использовать достаточно эффективно; этот метод организации виртуальной памяти был бы одним из самых лучших, если бы не два следующих обстоятельства.

Первое — это то, что страничная трансляция виртуальной памяти требует существенных накладных расходов. В самом деле, таблицы страниц нужно тоже размещать в памяти. Кроме того, эти таблицы нужно обрабатывать; именно с ними работает диспетчер памяти.

Второй существенный недостаток страничной адресации заключается в том, что программы разбиваются на страницы случайно, без учета логических взаимосвязей, имеющихся в коде. Это приводит к тому, что межстраничные переходы, как правило, осуществляются чаще, нежели межсегментные, и к тому, что становится трудно организовать разделение программных модулей между выполняющимися процессами.

Для того чтобы избежать второго недостатка, постаравшись сохранить достоинства страничного способа распределения памяти, был предложен еще один способ — сегментно-страничный. Правда, за счет увеличения накладных расходов на его реализацию.

щита страничной памяти, как и в случае сегментного механизма, основана на контроле уровня доступа к каждой странице. Как правило, возможны следующие уровни доступа:

- только чтение;
- чтение и запись;
- только выполнение.

ждая страница снабжается соответствующим кодом уровня доступа. При трансформации логического адреса в физический сравнивается значение кода разрешенного уровня доступа с фактически требуемым. При их несовпадении работа программы прерывается.

и обращении к виртуальной странице, не оказавшейся в данный момент в оперативной памяти, возникает прерывание, и управление передается диспетчеру памяти, который должен найти свободное место. Обычно предоставляется первая же свободная страница. Если свободной физической страницы нет, то диспетчер памяти по одной из вышеупомянутых дисциплин замещения (LRU, LFU, FIFO, случайный доступ) определит страницу, подлежащую расформированию или сохранению во внешней памяти. На ее месте он разместит новую виртуальную страницу, к которой было обращение из задачи, но которой не оказалось в оперативной памяти.

к и в случае с сегментным способом организации виртуальной памяти, страничный механизм приводит к тому, что без специальных аппаратных средств он существенно замедляет работу вычислительной системы. Поэтому обычно используется кэширование страничных дескрипторов. Наиболее эффективным механизмом кэширования является ассоциативный кэш. Именно такой ассоциативный кэш и создан в 32-разрядных микропроцессорах i80x86. Начиная с i80386, который поддерживает страничный способ распределения памяти, в этих микропроцессорах имеется кэш на 32 страничных дескриптора. Поскольку размер страницы в этих микропроцессорах равен 4 Кбайт, возможно быстрое обращение к памяти размером 128 Кбайт.

ак, основным достоинством страничного способа распределения памяти является

минимальная фрагментация. Поскольку на каждую задачу может приходиться по одной незаполненной странице, очевидно, что память можно использовать достаточно эффективно; этот метод организации виртуальной памяти был бы одним из самых лучших, если бы не два следующих обстоятельства.

Первое — это то, что страничная трансляция виртуальной памяти требует существенных накладных расходов. В самом деле, таблицы страниц нужно тоже размещать в памяти. Кроме того, эти таблицы нужно обрабатывать; именно с ними работает диспетчер памяти.

Второй существенный недостаток страничной адресации заключается в том, что программы разбиваются на страницы случайно, без учета логических взаимосвязей, имеющихся в коде. Это приводит к тому, что межстраничные переходы, как правило, осуществляются чаще, нежели межсегментные, и к тому, что становится трудно организовать разделение программных модулей между выполняющимися процессами.

Для того чтобы избежать второго недостатка, постаравшись сохранить достоинства страничного способа распределения памяти, был предложен еще один способ — сегментно-страничный. Правда, за счет увеличения накладных расходов на его реализацию.

Сегментно-страничный способ организации памяти.

Как и в сегментном способе распределения памяти, программа разбивается на логически законченные части — сегменты — и виртуальный адрес содержит указание на номер соответствующего сегмента. Вторая составляющая виртуального адреса — смещение относительно начала сегмента — в свою очередь может быть представлено состоящим из двух полей: виртуальной страницы и индекса. Другими словами, получается, что виртуальный адрес теперь состоит из трех компонентов: сегмента, страницы и индекса.

Этот способ организации виртуальной памяти вносит еще большую задержку доступа к памяти. Необходимо сначала вычислить адрес дескриптора сегмента и прочитать его, затем определить адрес элемента таблицы страниц этого сегмента и извлечь из памяти необходимый элемент и уже только после этого можно к номеру физической страницы приписать номер ячейки в странице (индекс). Задержка доступа к искомой ячейке получается, по крайней мере, в три раза больше, чем при простой прямой адресации. Чтобы избежать этой неприятности, вводится кэширование, причем кэш, как правило, строится по ассоциативному принципу. Другими словами, просмотры двух таблиц в памяти могут быть заменены одним обращением к ассоциативной памяти. Напомним, что принцип действия ассоциативного запоминающего устройства предполагает, что каждой ячейке памяти такого устройства ставится в соответствие ячейка, в которой записывается некий ключ (признак, адрес), позволяющий однозначно идентифицировать содержимое ячейки памяти. Сопутствующую ячейку с информацией, позволяющей идентифицировать основные данные, обычно называют полем тега. Просмотр полей тега всех ячеек ассоциативного устройства памяти осуществляется одновременно, то есть в каждой ячейке тега есть необходимая логика, позволяющая посредством побитовой конъюнкции найти данные по их признаку за одно обращение к памяти (если они там, конечно, присутствуют). Часто поле тегов называют аргументом, а поле с данными — функцией. В данном случае в качестве аргумента при доступе к ассоциативной памяти выступают номер сегмента и номер виртуальной страницы, а в качестве функции от этих аргументов получаем номер физической страницы. Остается приписать номер ячейки в странице к полученному номеру, и мы получаем адрес искомой команды или операнда.

Оценим достоинства сегментно-страничного способа. Разбиение программы на сегменты позволяет размещать сегменты в памяти целиком. Сегменты разбиты на страницы, все страницы сегмента загружаются в память. Это позволяет сократить число обращений к отсутствующим страницам, поскольку вероятность выхода за пределы сегмента меньше вероятности выхода за пределы страницы. Страницы исполняемого сегмента находятся в памяти, но при этом они могут находиться не рядом друг с другом, а «россыпью», поскольку диспетчер памяти манипулирует страницами. Наличие сегментов облегчает разделение программных модулей между параллельными процессами. Возможна и динамическая компоновка задачи. А выделение памяти страницами позволяет минимизировать фрагментацию.

Однако поскольку этот способ распределения памяти требует очень значительных затрат

вычислительных ресурсов и его не так просто реализовать, используется он редко, причем в дорогих мощных вычислительных системах. Возможность реализовать сегментно-страничное распределение памяти заложена и в семейство микропроцессоров i80x86, однако вследствие слабой аппаратной поддержки, трудностей при создании систем программирования и операционной системы практически в персональных компьютерах эта возможность не используется.

Практическая часть.

Задание 1. Определите, как распределяется оперативная память на компьютере, за которым Вы работаете. Для этого:

Откройте <Пуск>/<Настройка>/<Панель управления>/<Система>.

Запишите все сведения о системе.

Определите, на какой вкладке можно определить состояние системы (запишите эти данные),

Рассмотрите свойства файловой системы (какие устройства здесь рассматриваются? Запишите эти данные).

Обратите особое внимание на вкладку <Виртуальная память>. Запишите, какие возможности Вы имеете для настройки системы виртуальной памяти. Определить размер файла подкачки.

Нажмите комбинацию кнопок <ctrl>/ <alt>/ <del>.

Запишите какие задачи решаются в системе. Определите объем страницы для вашей системы. Понаблюдайте с помощью гистограмм и графиков изменение во времени загрузки процессора и использование виртуальной памяти.

Задание 2. Дать сравнительную характеристику сегментного и страничного способа организации виртуальной памяти. Перечислить достоинства и недостатки каждого. Данные оформить в виде таблицы.

Контрольные вопросы

1. Что такое виртуальная память?
2. Какие существуют методы распределения виртуальной памяти?
3. Почему размер страницы выбирается равным степени двойки? Можно ли принять такое же ограничение для сегмента?
4. На что влияет размер страницы?
5. Каковы преимущества и недостатки большого размера страницы

Практическая работа №7. Организация оперативной памяти. Изучение структуры, состава и принципа работы микропроцессорных систем

Цель работы: изучить организацию оперативной памяти. Изучить возможности программы-отладчика DEBUG как средства отображения памяти и приобрести практические навыки работы с ней.

Теоретическая часть

Оперативная память предназначена для хранения программ и данных. Ее можно рассматривать как конечную последовательность ячеек, имеющих размер 1 байт. Номер байта в этой последовательности называется физическим адресом (или просто адресом).

Физический адрес используется для получения доступа к конкретной ячейке памяти. Именно эта информация выставляется центральным процессором на шину адреса.

Для обеспечения доступа к оперативной памяти в процессоре Intel 8086 и процессорах последующих поколений, работающих в реальном режиме, используется так называемая сегментированная модель памяти.

В сегментированной модели вводится понятие "сегмента". Так называется любой участок памяти размером до 64 Кб и с начальным адресом, кратным 16. Физический адрес формируется процессором на основании этого начального адреса и смещения конкретной ячейки памяти относительно начала сегмента. Для хранения начального адреса сегмента применяются сегментные регистры процессора.

Процессор обеспечивает доступ к четырем сегментам одновременно. Эти сегменты называются сегментом кода, сегментом данных, сегментом стека и дополнительным сегментом данных.

Сегмент кода содержит команды программы. Для доступа к этому сегменту используется сегментный регистр *CS*. Он содержит адрес сегмента с машинными командами.

Сегмент данных содержит обрабатываемые программой данные. Для доступа к этому сегменту служит сегментный регистр *DS*, который хранит адрес сегмента данных текущей программы.

Сегмент стека – этот сегмент представляет собой область памяти, называемую стеком. Работу со стеком микропроцессор организует по следующему принципу: последний записанный в эту область элемент выбирается первым. Для доступа к этому сегменту служит сегментный регистр *SS*, содержащий адрес сегмента стека.

Дополнительный сегмент данных применяется в некоторых командах для организации обмена информации между этим сегментом и сегментом данных. Адрес дополнительного сегмента данных должен содержаться в сегментном регистре *ES*.

С представлением данных в памяти ЭВМ тесно связано понятие типа данных.

Понятие типа данных носит двойственный характер. С точки зрения размерности микропроцессор аппаратно поддерживает следующие основные типы данных.

1. Байт – восемь последовательно расположенных битов, пронумерованных от 7 до 0, при этом бит 0 является самым младшим значащим битом.
2. Слово – последовательность из двух байт, имеющих последовательные адреса. Размер слова – 16 бит; биты в слове нумеруются от 15 до 0. Байт, содержащий нулевой бит, называется младшим байтом, а байт, содержащий 15-й бит, – старшим байтом. Процессоры Intel имеют важную особенность – младший байт всегда хранится по меньшему адресу. Адресом слова считается адрес его младшего байта. Адрес старшего байта может быть использован для доступа к старшей половине слова.
3. Двойное слово – последовательность из четырех байт (32 бита), расположенных по последовательным адресам. Нумерация этих бит производится от 31 до 0. Слово, содержащее нулевой бит, называется младшим словом, а слово, содержащее 31-й бит, — старшим словом. Младшее слово хранится по меньшему адресу. Адресом двойного слова считается адрес его младшего слова. Адрес старшего слова может быть использован для доступа к старшей половине двойного слова.

Кроме трактовки типов данных с точки зрения их разрядности, процессор на уровне команд поддерживает логическую интерпретацию этих типов. С точки зрения логической интерпретации выделяют следующие типы данных.

1. Целый тип со знаком – двоичное значение со знаком, размером 8, 16 или 32 бита. Знак в этом двоичном числе содержится в 7, 15 или 31-м бите соответственно. Ноль в этих битах в операндах соответствует положительному числу, а единица – отрицательному. Отрицательные числа представляются в дополнительном коде. Числовые диапазоны для этого типа данных следующие:

– 8-разрядное целое – от – 128 до 127;

– 16-разрядное целое – от – 32 768 до 32 767;

– 32-разрядное целое – от -2^{31} до $2^{31} - 1$.

2. Целый тип без знака – двоичное значение без знака, размером 8, 16 или 32 бита. Числовой диапазон для этого типа следующий:

– байт – от 0 до 255;

– слово – от 0 до 65 535;

– двойное слово – от 0 до $2^{32} - 1$.

3. Указатель на память (адрес) бывает двух типов.

Ближний тип – 16-разрядный логический адрес, представляющий собой относительное смещение в байтах от начала сегмента (короткий адрес).

Дальний тип – 32-разрядный логический адрес, состоящий из двух частей: 16-разрядной сегментной части и 16-разрядного смещения (полный адрес).

4. Цепочка представляет собой некоторый непрерывный набор байтов, или слов максимальной длиной до 64 Кбайт.

5. Символ – байт, в который записывается код символа – целое от 0 до 255. В ЭВМ используется система кодировки *ASCII* (American Standard Code for Information Interchange).

6. Строка – последовательность символов, которая размещается в соседних байтах памяти, так, что код первого символа строки записывается в первом байте, код второго символа - во втором байте и т.п. Адресом строки считается адрес ее первого байта.

7. Неупакованный двоично-десятичный тип – байтовое представление десятичной цифры от 0 до 9. Неупакованные десятичные числа хранятся как байтовые значения без знака по одной цифре в каждом байте. Значение цифры определяется младшим полубайтом.

8. Упакованный двоично-десятичный тип представляет собой упакованное представление двух десятичных цифр от 0 до 9 в одном байте. Каждая цифра хранится в своем полубайте.

DEBUG – это системная программа, позволяющая выполнять просмотр и изменение состояний процессора и памяти компьютера, побайтное тестирование и побайтную обработку дисковых файлов, что обеспечивает возможность выполнения отлаживаемых программ небольшими порциями. При этом программа выполняется под "наблюдением" отладчика. Таким образом, основное назначение этой программы – отладка программ на уровне машинных кодов и языка ассемблера. Однако возможности, предоставляемые этой программой, делают ее удобным инструментом для изучения организации персональных компьютеров.

Запуск отладчика.

Чтобы запустить отладчик, в командной строке Windows (Меню Пуск – Выполнить) вводится команда debug (см. рис. 1). В качестве параметра команда запуска отладчика может включать имя обрабатываемого файла, например debug lab1.com

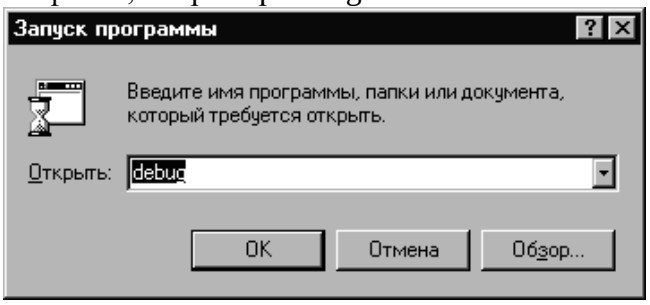


Рис. 1. Диалог "Запуск программы DEBUG.

Если обрабатываемый файл не записан в текущем каталоге, то в командной строке необходимо указать полный путь к нему.

После нажатия кнопки "*OK*" диалога запуска отладчик загружается в память машины и переходит в режим ожидания ввода команды.

Команды программы DEBUG

DEBUG – это программа, работающая по принципу "команда – действие", т.е., чтобы произвести некоторую операцию отладчик должен получить соответствующую команду. В качестве сигнала о готовности принять команду, отладчик посылает на экран стандартный запрос – дефис (-).

Для управления процессом отладки в DEBUG применяется набор команд, список которых можно получить, введя команду помощи (символ "Вопрос"). Окно отладчика, в котором выведен список поддерживаемых команд, приведено на рис. 2.

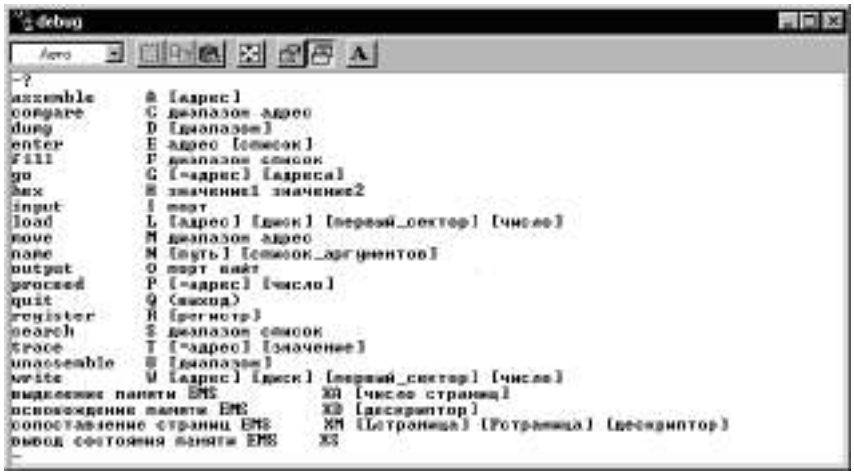


Рис. 2.2. Окно программы DEBUG

Общими замечаниями по вводу команд отладчика является следующее:

– все команды начинаются с буквы, заглавной или строчной;

– большая часть команд требует введения дополнительных параметров, часть из которых является необязательными;

– если два подряд расположенных параметра являются числами, то они разделяются пробелом или запятой (в противном случае параметры можно не отделять один от другого);

– все числа должны вводиться в шестнадцатеричном представлении;

– некоторые команды принимают в качестве параметра адрес, который может вводиться в двух формах: полный логический адрес – два шестнадцатеричных числа, записанные через двоеточие (первое число – сегментная компонента логического адреса, второе число – смещение) и короткий адрес – одно шестнадцатеричное число (смещение);

– при указании полного логического адреса допускается в качестве сегментной компоненты приводить имя сегментного регистра, из которого данная компонента будет выбираться.

С точки зрения изучения организации ЭВМ, набор команд, предлагаемый отладчиком, является избыточным. Минимально необходимый набор включает команды:

– (A)SSEMBLE – ассемблирование;
– (U)NASSEMBLE – дизассемблирование;
– (E)NTER – ввод данных в память;
– (D)UMP – вывод содержимого участка памяти на экран;
– (R)EGISTER – просмотр и изменение содержимого регистров;
– (T)RACE – пошаговое выполнение программы;
– (N)AME – задание имени файла программы;
– (L)OAD – загрузки файла в память;
– (W)RITE – запись области памяти в файл;
– (Q)UIT – выход из отладчика.

Перечисленные команды представляют для нас наибольший интерес. Рассмотрим их подробнее.

Команда ассемблирования (перевод мнемокода ассемблера в машинный код)

Отладчик DEBUG можно использовать для введения операторов ассемблера непосредственно в память машины. *Команду ASSEMBLE* можно использовать при составлении коротких программ на ассемблере, а также при внесении изменений в существующие программы. Эта команда позволяет вводить мнемокод ассемблера непосредственно в память, избавляя от необходимости транслировать (ассемблировать) программу. Вводимый текст не может включать метки перехода в чистом виде.

При введении команды, необходимо набрать "*a*" или "*A*" и, через пробел, необязательный параметр – адрес первой команды загружаемой программы. Если указан короткий адрес, то адрес сегмента выбирается из регистра *CS*. Если адрес не задан вообще, то машинный код будет помещаться в память, начиная с того места, где закончилась обработка предыдущей командой

ASSEMBLE. Если после старта отладчика команда вводится в первый раз и в командной строке отсутствует начальный адрес, то размещение машинного кода производится с адреса *CS*:0100.

После введения команды ассемблирования на экране появляется начальный адрес. Это сигнал на введение первой команды программы. Если команда введена без ошибок, на экран выдается адрес следующей команды и отладчик опять переходит в режим ожидания. В случае ошибки отладчик обозначает ее месторасположение. Если введены все команды программы, то нажимается *Enter* – команда ASSEMBLE заканчивает работу и возвращает управление отладчику.

Пример ассемблирования небольшой программы:

```
-a 0976:0100
0976:0100  MOV AL,2A
0976:0102  MOV DI,0200
0976:0105  MOV CX,001D
0976:0108  CLD
0976:0109  REPNZ STOSB
0976:010B  MOV AL,24
0976:010D  STOSB
0976:010E  PUSH ES
0976:010F  POP DS
0976:0110  MOV DX,0200
0976:0113  MOV AH,09
0976:0115  INT 21
0976:0117  INT 20
0976:0119        <---- Нажимается Enter
-
```

Команда дизассемблирования (перевод машинного кода в мнемокод ассемблера)

Команда UNASSEMBLE служит для перевода машинного кода на язык ассемблера. При введении команды необходимо набрать "*u*" или "*U*" и, через пробел, необязательные параметры – начальный адрес обрабатываемого кода, конечный адрес обрабатываемого кода или его размер.

В командной строке UNASSEMBLE можно не указывать начальный адрес обрабатываемого кода. Если указан короткий адрес, то адрес сегмента выбирается из регистра *CS*. Если адрес не задан вообще, то машинный код обрабатывается с того места, где закончилась обработка предыдущей командой UNASSEMBLE. Если после старта отладчика команда вводится в первый раз и в командной строке отсутствует начальный адрес, то обработка машинного кода производится с адреса *CS*:0100.

Обрабатываемый участок памяти можно определить начальным и конечным адресами. При этом, в не зависимости от формы начального адреса, конечный адрес должен быть коротким.

Другой вариант задания обрабатываемого участка памяти – задание его начального адреса и размера. Чтобы отличить размер от короткого конечного адреса перед ним вводится символ "*L*".

Если размер участка памяти, обрабатываемой командой UNASSEMBLE, не определен, то по умолчанию длина обрабатываемого участка равна 32 байтам.

Результатом выполнения команды дизассемблирования является листинг программы, сгруппированный в три колонки. В листинге слева (первая колонка) указывается полный логический адрес команды. Затем (вторая колонка) – значение составляющих команду байтов в машинном коде. В третьей колонке находится соответствующая этому коду инструкция ассемблера.

Пример дизассемблирования небольшой программы, введенной в предыдущем примере:

```
-u CS:0100 L19
0976:0100      B02A        MOV AL,2A
0976:0102      BF0002            MOV DI,0200
0976:0105      B91D00      MOV CX,001D
0976:0108      FC          CLD
0976:0109      F2          REPNZ
0976:010A      AA          STOSB
```

```
0976:010B      B024        MOV AL,24
0976:010D      AA          STOSB
0976:010E      06          PUSH ES
0976:010F      1F          POP DS
0976:0110      BA0002          MOV DX,0200
0976:0113      B409        MOV AH,09
0976:0115      CD21        INT 21
0976:0117      CD20        INT 20
-
```

Команда ввода данных в память.

Ввод данных осуществляется с помощью *команды ENTER*. Эта команда позволяет побайтно корректировать содержимое памяти. Команда состоит из буквы *e* (или *E*) и адреса первого байта корректируемого блока. Если указан короткий адрес, то адрес сегмента выбирается в регистре *DS*.

Вводимые данные также включаются в командную строку. Они представляют собой последовательность чисел в шестнадцатеричном представлении и/или символьных значений, разделенных пробелом или запятой. Символьные значения заключаются в апострофы.

Проиллюстрируем работу ENTER на следующем примере:

-e DS:0000 20 2A 44 41 54 41 20 'IS' 20 48 45 52 45 2A 20

Команда вводит 16 значений. Данные последовательно заполняют память (побайтно), начиная с адреса DS:0000. Четырнадцать байтов занимают числа в шестнадцатеричном формате, два байта отводятся под символьную константу 'IS'.

Команда ENTER может использоваться для отображения и, в случае необходимости, корректировки значения конкретного байта. В этом случае команда состоит из буквы *e* (или *E*) и следующего за ней адреса. При введении команды на экране появляется адрес байта и его значение:

-e DS:0000

0958:0000 20.

При нажатии на клавишу пробела на экране появляется значение следующего байта:

-e DS:0000

0958:0000 20. 2A.

Значение байта можно изменить. Для этого вводится новое шестнадцатеричное число. Однако символьные переменные в этом случае вводить нельзя.

Чтобы завершить выполнение команды, нажимается *Enter*. Появление дефиса (-) – стандартного запроса отладчика, свидетельствует о его готовности принять следующую команду.

Команда вывода содержимого участка памяти на экран.

Команда DUMP (*d* или *D*) служит для отображения на экране содержимого участка памяти. Полученный кусочек памяти – дамп, представляет собой последовательность значений байтов в шестнадцатеричном представлении, а также – в коде ASCII:

```
-d
0958:0100 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0958:0110 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0958:0120 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0958:0130 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0958:0140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0958:0150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0958:0160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
0958:0170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

Адрес Значения в шестнадцатеричном виде, в виде символов

Числа в первом столбце, разделенные двоеточием – это полный логический адрес памяти, следующие 16 столбцов – значения, последовательно содержащиеся в памяти, начиная с этого

адреса (в шестнадцатеричном виде), и, наконец, строка из 16 символов – символьное представление этих значений.

Значения, не имеющие символьного представления в коде *ASCII*, обозначаются символом "точка". В рассмотренном примере изображены только точки, поскольку в коде *ASCII* не существует печатных символов со значением 00. Дамп отображает содержимое 128 последовательно расположенных байтов. В приведенном выше примере начальный адрес дампа – 0958:0100, конечный – 0958:017F.

Если вводить команду "*d*" не указывая параметров, DEBUG будет последовательно выводить по 128 байтов памяти. То есть начальный адрес дампа будет на единицу превышать конечный адрес дампа, полученного при введении предыдущей команды "*d*". Если команда "*d*" вводится первоначально, то дамп выводится, начиная с адреса, по которому был загружен обрабатываемый файл.

Команда DUMP может использоваться с параметрами, которые задают начальный адрес дампа и его размер. Использование параметров аналогично использованию параметров в команде UNASSEMBLE. За тем исключением, что, если начальный адрес дампа задан в коротком формате, то сегментная компонента адреса выбирается из регистра *DS*.

Команда просмотра и изменения содержимого регистров.

Команда REGISTER (*r* или *R*) выводит на экран и корректирует значения регистров и флагов состояния процессора. Эта команда также выдает информацию о следующей выполняемой команде:

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0958 ES=0958 SS=0958 CS=0958 IP=0100    NV UP DI PL NZ NA PO NC
0958:0100  0000       ADD          [BX+SI],AL          DS:0000=CD
-
```

С помощью "*r*" можно изменить значение регистра. В этом случае в командной строке указывается его имя. Значение регистра выводится на экран. Теперь можно вводить новое число. Чтобы сохранить старое значение регистра, нажмите *Enter*.

```
-r CX
CX 0000
:245D
-r
AX=0000 BX=0000 CX=245D DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0958 ES=0958 SS=0958 CS=0958 IP=0100    NV UP DI PL NZ NA PO NC
0958:0100  0000       ADD          [BX+SI],AL          DS:0000=CD
-
```

Команда "rf" выводит на экран флаги состояния процессора. Получив значения флагов, их можно изменить. Для этого вводится одно или несколько новых значений.

Мнемонические обозначения состояний флагов приведены в табл. 2.2.

Таблица 2.2

Флаг	Установлен	Сброшен
Переполнение (есть/нет)	OV	NV
Направления (увеличение/уменьшение)	DN	UP
Прерывания (разрешение/запрещение)	EI	DI
Знака (минус/плюс)	NG	PL
Нуля (да/нет)	ZR	NZ
Дополнительного переноса (да/нет)	AC	NA
Паритета (чет/нечет)	PE	PO
Переноса (да/нет)	CY	NC

Символьные значения вводятся в любом порядке через пробел или вообще без разделителя. Установим, например, значения флагов переполнения, знака и переноса:

```
-rf
NV UP DI PL NZ NA PO NC   -OV  NG  CY <- Подчеркнутое вводит пользователь
-r
AX=0000 BX=0000 CX=245D DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0958 ES=0958 SS=0958 CS=0958 IP=0100    OV UP DI NG NZ NA PO CY
0958:0100  CD20        INT       20
-
```

Команда пошагового исполнения программы.

Команда TRACE (t или T) – трассировка осуществляет пошаговое выполнение программы в машинном коде. При трассировке после выполнения каждой команды производится останов работы программы и на экран выводятся регистры и флаги состояния процессора. Полученная картинка аналогична картинке, получаемой с помощью команды REGISTER. Разница заключается только в том, что при введении TRACE перед появлением картинки, выполняется одна команда отлаживаемой программы.

Проиллюстрируем работу TRACE на примере нашей программы. Если она не загружена в память, то запустим DEBUG и введем:

```
-e CS:0100 B0 2A BF 00 02 B9 1D 00 FC F2 AA B0 24
-e CS:010D AA 06 1F BA 00 02 B4 09 CD 21 CD 20
-
```

Чтобы узнать адрес программы, введем команду REGISTER:

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0976 ES=0976 SS=0976 CS=0976 IP=0100    NV UP DI PL NZ NA PO NC
0976:0100  B001        MOV           AL,2A
-
```

При введении "t" выполняется команда по адресу CS:IP. После этого на экран выводятся регистры и флаги состояния:

```
-t
AX=002A BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0976 ES=0976 SS=0976 CS=0976 IP=0102    NV UP DI PL NZ NA PO NC
0976:0102  BF0002      MOV           DI,0200
-t
AX=002A BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0200
DS=0976 ES=0976 SS=0976 CS=0976 IP=0105    NV UP DI PL NZ NA PO NC
0976:0105  B91D00      MOV           CX,001D
-
```

В командной строке TRACE можно указать адрес выполняемой команды. В этом случае после t набирается знак равенства (=) и нужный адрес. Если указан короткий адрес, то адрес сегмента выбирается из регистра CS:

```
-t=0100
AX=002A BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0200
DS=0976 ES=0976 SS=0976 CS=0976 IP=0102    NV UP DI PL NZ NA PO NC
0976:0102  BF0002      MOV           DI,0200
-
```

В этом случае выполнена команда по адресу CS:0100. Адрес следующей команды находится в регистрах CS:IP. Он равен 0976:0102.

Одной командой TRACE можно одновременно трассировать несколько команд отлаживаемой программы. Для этого при введении "t" просто указывается их количество. После выполнения каждой команды на экране появляется картинка с содержимым регистров и флагов

состояния. При заполнении экрана новые данные выводятся в нижней его части, сдвигая данные в верхней части за пределы экрана. Чтобы остановить движение данных вдоль экрана, нажимаются клавиши Ctrl-NumLock. Чтобы возобновить движение, нажимается любая клавиша.

При нажатии Ctrl-C трассирование прекращается и на экране появляется стандартный запрос отладчика.

Пример:

```
-t6
AX=002A BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0200
DS=0976 ES=0976 SS=0976 CS=0958 IP=0105    NV UP DI PL NZ NA PO NC
0976:0105  B91D00      MOV       CX,001D

AX=002A BX=0000 CX=001D DX=0000 SP=FFEE BP=0000 SI=0000 DI=0200
DS=0976  ES=0976 SS=0976 CS=0958 IP=0108    NV UP DI PL NZ NA PO NC
0976:0108  FC          CLD

AX=002A BX=0000 CX=001D DX=0000 SP=FFEE BP=0000 SI=0000 DI=0200
DS=0976 ES=0976 SS=0976 CS=0958 IP=0109    NV UP DI PL NZ NA PO NC
0976:0109  F2          REPNZ
0976:010A  AA          STOSB

AX=002A BX=0000 CX=001C DX=0000 SP=FFEE BP=0000 SI=0000 DI=0201
DS=0976 ES=0976 SS=0976 CS=0958 IP=0109    NV UP DI PL NZ NA PO NC
0976:0109  F2          REPNZ
0976:010A  AA          STOSB

AX=002A BX=0000 CX=001B DX=0000 SP=FFEE BP=0000 SI=0000 DI=0202
DS=0976 ES=0976 SS=0976 CS=0958 IP=0109    NV UP DI PL NZ NA PO NC
0976:0109  F2          REPNZ
0976:010A  AA          STOSB

AX=002A BX=0000 CX=001A DX=0000 SP=FFEE BP=0000 SI=0000 DI=0203
DS=0976 ES=0976 SS=0976 CS=0958 IP=0109    NV UP DI PL NZ NA PO NC
0976:0109  F2          REPNZ
0976:010A  AA          STOSB
-
```

Команда задания имени файла программы.

Команда NAME (n или N) присваивает имя обрабатываемому файлу. Затем этот файл загружается в память командой LOAD или записывается на диск командой WRITE. (LOAD и WRITE рассматриваются ниже.)

Чтобы идентифицировать файл, наберите "n" и, через пробел – имя файла. Воспользуемся NAME, чтобы присвоить нашей программе имя "mytest.pro":

```
-n mytest.pro
```

Команда загрузки файла в память.

Загрузка файла в память осуществляется, если в командной строке DEBUG указать имя файла. Другой способ – использование команды LOAD (l или L).

При использовании команды LOAD необходимо специфицировать файл с помощью команды NAME.

В командной строке LOAD можно указать начальный адрес, по которому загружается файл. Если указан короткий адрес, то адрес сегмента выбирается из регистра CS. При отсутствии начального адреса, загрузка производится по адресу CS:0100.

После загрузки отладчик запоминает количество занятой файлом памяти (в байтах) в регистрах BX (старшее слово) и CX (младшее слово).

К примеру, загрузим в память файл "mytest.pro" по адресу CS:0100:

```
-n mytest.pro
-L
-r
AX=0000 BX=0000 CX=00CF DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=0958 ES=0958 SS=0958 CS=0958 IP=0100    NV UP DI PL NZ NA PO NC
0958:0100  2A2A        SUB           CH,[BP+SI]           SS:0000=CD
-
```

В регистрах BX и CX находится значение 207 (000000CF). Это значит, что файл занял 207 байт. Тот же результат можно получить при введении спецификации файла в командной строке команды старта отладчика ("debug mytest.pro").

Команда записи области памяти в файл.

Команда WRITE (w или W) переписывает на диск данные, выбирая их из памяти. При этом спецификация создаваемого файла должна задаваться с помощью команды NAME.

Перед введением команды WRITE в регистры BX и CX записывается размер занимаемой файлом памяти в байтах (шестнадцатеричное число, занимающее 4 байта). Поэтому перед записью необходимо проверить содержимое этих регистров (с помощью REGISTER).

В командной строке WRITE можно указать начальный адрес памяти, по которому производится чтение данных с последующей записью их на диск. Если указан короткий адрес, то адрес сегмента выбирается из регистра CS.

Если начальный адрес не указан, то запись производится, начиная с адреса CS:0100.

Команда выхода из отладчика.

Чтобы выйти из отладчика и передать управление операционной системе, на его стандартный запрос вводится команда q:

```
-q
```

Содержание работы.

1. Ознакомиться с теоретическим материалом.
2. В соответствии со своим вариантом (пример для ассемблирования) опробовать команды DEBUG.

Варианты заданий.

1	MOV AL,20
	MOV BL,10
	ADD AL,BL
	SUB BL,3
	AND AX,BX
		14	INC DH
			MOV DL,00
			AND AX,DX
			NOT AX
			MOV BX,AX

2	MOV AX,1111
	MOV BX,2
	MUL BX
	INC AX
	XOR AX,FFFF
		15	MOV AL,03
			DEC AL
			CMP AL,00
			JNE 103
			MOV DX,0F0E

3	MOV BL,FF
	PUSH BX
	DEC BL
	POP AX
	SUB AX,BX
		16	MOV BX,100
			SUB BX,AX
			NEG AX
			INC BX
			ADD BX,AX

4	MOV AX,7
	CMP AX,4
	JNE 10C
	MOV BL,7
	DIV BL
		17	MOV BX,13
			DEC BX
			MOV AX,BX
			IDIV BL
			MOV BL,AL

5	MOV BX,FF00
	ROR BX,1
	NOT BX
	SUB BX,0F
	CMP AX,BX
		18	MOV DX,FFFF
			MOV AX,1111
			NOP
			AND DX,AX
			NOT DX

6	MOV AX,3003
	DEC AH
	INC AL
	AND AX,04
	XOR AX,FFFF
		19	INC AX
			CMP AX,FFFF
			JNE 100
			NEG AX
			MOV CX,AX

7	
	AND AX,40
	JNZ 109
	MOV BX,FFEE
	NOP
	CMP BX,FFEE
		20	
			STC
			MOV AL,F1
			MOV BL,0F
			ADC AL,BL
			CLC

8	MOV AX,000F
	ROR AX,1
	MOV BX,0330
	OR AX,BX
	XOR AX,0
		21	POP CX
			MOV AX,0FFF
			DIV CL
			ROL AX,CL
			PUSH AX

9	DEC AX NOT AX MOV CL,05 MUL CL NEG AX	22	NOT AX NOP RCR AX,1 ADC AX,2EA SUB AX,0A

10	POP BX
	POP AX
	SUB AX,BX
	MOV DX,AX
	NEG AX
		23	MOV CL,20
			MOV CH,20
			AND CH,CL
			SUB CX,20
			IMUL CX

11	XOR BH,BH
	MUL BH
	CMP AX,0
	JZ 10A
	NOP
	MOV AX,2000
		24	NEG BX
			MOV DX,300
			ADD BX,DX
			MOV AX,BX
			PUSH AX
			POP CX

12	MOV CL,15
	MOV CH,15
	PUSH CX
	ADD CH,CL
	POP AX
	SUB AX,CX
		25	NOP
			INC AX
			DEC BX
			ADD AX,BX
			AND AX,43
			NOT AX

13	LAHF STI CLC SAHF MOV DX,FFF0		

Практическая работа №8. Запись и выполнение простых команд: INR, DCRADD, ANA, ORA, XRA. Запись и выполнение простых команд: DAA, RAR, SUB, SBB.

Цель: изучение правил записи прикладных программ с использованием директив Ассемблера, ознакомление с программами Ассемблера и симулятора и приобретение навыков работы с кросс-средствами отладки.

Теоретическая часть.

Рассмотрим функционирование устройства пошагового выполнения программы. Устройство пошагового выполнения программы переводит ОУ в состояние ожидания после выполнения очередного шага. Вызов пошагового режима работы осуществляется переключателем **"РБ/ШГ",** выбор величины шага переключателем **"КМ/ЦК".** Для последующего шага необходимо нажать кнопку **"ШГ".** При этом после выполнения очередного шага на светодиодном индикаторе отображается состояние адресной вины, шины данных и регистра состояния ОУ в двоичном коде.

Команды выполняются по машинным циклам (1 ¸ 5 циклов в команде). Имеется десять типов машинных циклов и соответственно десять слов состояния (см. табл.1).

Таблица 1.

Состояние ОУ	***Разряды регистра состояния ОУ***							
	EM	**NR**	**I**	**UT**	**LTA**	**STACK**	**O/**	**NTA**
Выбор команды					0	0		
Чтение памяти					0	0		
Запись в память					0	0		
Чтение стека					0	1		
Запись в стек					0	1		
Ввод					0	0		
Вывод					0	0		
Прерывание					0	0		
Останов					1	0		
Прерывание в останове					1	0		

Перечислим органы управления**.**

Директивные клавиши служат для вызова директив и имеют следующие обозначения и назначения:

"П" (Память) – чтение и изменение содержимого ячеек памяти,

"РГ" (Регистр) – чтение и изменение содержимого регистров МП,

"СТ" (Старт) – запуск программы пользователя,

"КС" (Контрольная сумма) – определение контрольной суммы массива памяти: используется для контроля правильности ввода ранее отлаженной программы,

"ЗК" (Заполнение константой) – заполнение массива памяти константой,

"ПМ" (Перемещение массива) – перемещение массива в адресном пространстве памяти,

"ВП" (Выполнить) – означает конец работы с директивой,

"¾" (Пробел) – разделитель при вводе нескольких переменных.

Информационные клавиши служат для ввода адресов и данных в шестнадцатеричном коде и содержат шестнадцатеричные символы. Одновременно часть из них используется для ввода индентификаторов регистров МП:

A – аккумулятор,

B, C , D, E – одноименные регистры,

8/H – регистр H**,**

9/L – регистр L,

F – регистр флагов (признаков),

4/PH – старший байт счетчика команд,

5/PL – младший байт счетчика команд,

6/SH – старший байт указателя стека,

7/SL – младший байт указателя стека.

На лицевой панели находятся пять кнопок управления**:**

~ – включение/выключение УМК,

СБ (Сброс) – устанавливает нулевой адрес в счетчике команд,

ПР (Прерывание) – позволяет прервать выполнение программы,

РБ/ШГ (Работа/Шаг) – используется для перевода выполнения программы в пошаговом режиме,

ШГ (Шаг) – каждое нажатие на кнопку ведет к выполнению одного шага,

КМ/ЦК – переключатель дискреты шага (КМ – шаг соответствует выполнению одной команды, ЦК – шаг соответствует выполнению одного машинного цикла).

Дисплей содержит 6 разрядов: четыре левых отображают адрес, два правых информацию (в шестнадцатеричном коде).

При неправильной работе с клавиатурой в крайней левой позиции дисплея высвечивается символ "**?**". Нажмите кнопку ²СБ² для возвращения в исходное состояние.

Светодиодная индикация состоит из набора светодиодов (0 – светодиод не горит, 1 – горит).

Верхний ряд отображает состояние адресной шины (16 бит).

Второй ряд отображает состояние шины данных (8 бит).

Третий ряд отображает разряды регистра состояний (8 бит).

Подготовка УМК к работе

Установите кнопку ″~″ в отжатое состояние.

1. Подключите УМК к сети переменного тока 220 В.
2. Переключатель "РБ/ШГ" установите в отжатое состояние ″**РБ**″.
3. Включите УМК, нажав кнопку ″~″.

4. Нажмите кнопку **"СБ".** При этом в крайней левой позиции дисплея должен появиться знак "–". После этого УМК готов к работе.

Базовые рабочие процедуры

Внимание! Перед заданием директив с помощью управляющих клавиш необходимо нажать ″СБ″.

Индикация и изменение содержимого памяти

Если необходимо узнать содержимое ячейки памяти ОЗУ с определенным адресом, то последовательно нажимают клавиши:

"П" "АДРЕС" "ВП",

где "АДРЕС" – адрес в шестнадцатеричном коде. При этом в левых четырех разрядах дисплея высвечивается адрес, а в двух правых содержимое ячейки памяти.

Внимание! Пользователю доступен массив ОЗУ с адресами

0800 ¸ OBFF.

Если необходимо посмотреть содержимое следующей ячейки, то надо нажать клавишу **″—″.**

Если информацию, хранящуюся в просматриваемой ячейке памяти надо изменить, то новые данные вводят с помощью информационных клавиш. При этом новые значения высвечиваются в двух правых разрядах дисплея. После этого нажимают клавишу "ВП" или для перехода к следующей ячейке клавишу "—". Директива завершается нажатием клавиши ″ВП″.

Пример 1: – просмотрите и измените содержимое ячеек памяти:

0800 3A
на 00
0801 0B
– 2F
0802 32
– 01
0803 0B
– 76
0804

–
0805
–
0806
–
0807
–

Индикация и изменение содержимого регистров

Если надо узнать содержимое регистра, то последовательно нажимают клавиши: **"РГ" "X" "ЗП",**

где "X" - информационная клавиша, идентифицирующая регистр. На дисплее высвечивается идентификатор регистра, а в двух правых разрядах его содержимое.

Идентификаторы регистров:

A – аккумулятор;

B, C, Д, E, H, L – регистры общего назначения (РОН),

F – регистр условий,

SL - младший байт указателя стека,

SH - старший байт указателя стека,

PL - младший байт счетчика команд,

PH - старший байт счетчика команд.

Все регистры по 8 бит.Если содержимое просматриваемого регистра изменять не надо, то нажимают ″—″ и набирают идентификатор другого регистра. Если содержимое регистра надо изменить, то набирают новые данные в шестнадцатеричном коде (два символа) и нажимают клавишу "—". После этого можно набирать идентификатор следующего регистра.

Для завершения директивы нажимают клавишу "ВП".

Заполнение массива памяти константой

Константа представляется в шестнадцатеричном коде двумя символами, адреса ячеек ОЗУ четырьмя символами.

Последовательность нажатия клавиш:

"ЗК" "АДРЕС 1" "¾" "АДРЕС 2" "¾" "К" "ВП",

где АДРЕС 1 – адрес первой ячейки массива памяти;

АДРЕС 2 – адрес последней ячейки массива памяти;

"К" - константа в шестнадцатеричном коде(00 ¸ FF)

Внимание! Заполнять только массив 0800 ¸ OBFF.

Перемещение массива памяти в адресном пространстве

Последовательность нажатия клавиш:

"ПМ" "АДРЕС 1" "—" "АДРЕС 2" "—" "АДРЕС 3" "ВП",

где АДРЕС 1 – начальный адрес перемещаемого массива;

АДРЕС 2 – конечный адреса перемещаемого массива;

АДРЕС 3 – новый начальный адрес массива.

Старый и новый массивы не должны перекрываться, иначе теряется информация.

Определение контрольной суммы массива памяти

Последовательность нажатия клавиш:

"КС" "АДРЕС 1" ″—″ "АДРЕС 2" "ВП",

где АДРЕС 1 – начальный адрес массива памяти,

АДРЕС 2 – конечный адрес массива памяти.

После выполнения директивы на дисплее высвечивается контрольная сумма массива, представляющая собой сумму содержимого ячеек массива по модулю 256.

Запуск программы пользователя

Последовательность нажатия клавиш:

"СТ" "АДРЕС 1" "—" "АДРЕС 2" "—" "АДРЕС 3" "ВП",

где АДРЕС 1 – начальный адрес программы,

АДРЕС 2 и АДРЕС 3 – адреса выполнения прерывания программы (могут отсутствовать)

Состояние регистров микропроцессора при достижении адресов 2 и 3 сохраняется в ОЗУ, и управление передается программе "Монитор". Пользователь может задать любую директиву. Продолжение программы последует после повторного нажатия клавиши "ВП".

Внимание! Программа должна умещаться в пределах массива 0800 ¸ 0AFF (1 кбайт).

Пошаговое выполнение программы

Имеются две разновидности пошагового выполнения программы: поцикловый режим "ЦК" и покомандный режим "КМ".

Для вызова пошагового режима:

- установите переключатель "РБ/ШГ" в положение "ШГ" (кнопка утоплена), при которой происходит подключение световой индикации;
- переключателем "КМ/ЦК" выберите один из режимов работы;
- передайте управление выполняемой программе.

После этого световая индикация отобразит начальный адрес программы, данные по этому адресу и содержимое регистра состояния. Программа выполняется путем нажатия кнопки "ШГ". Для выхода из пошагового режима установите переключатель "РБ/ШГ" в состояние "РБ" (кнопка отжата) и нажмите кнопку "ШГ".

Прерывание выполнения программы

Для прерывания выполнения программы необходимо нажать клавишу "ПР". После этого пользователь может вызвать выполнение любой из существующих директив. Выполнение прерванной программы возможно, начиная с адреса останова или любого другого адреса.

Запись и выполнение программы

Микропроцессор БИС КР580ВМ80 имеет фиксированный набор команд. Программа записывается последовательно в массив памяти ОЗУ 0800 ¸ 0AFF. Для записи данных исследуемых программ используют ячейки ОЗУ с адресами 0B00 ¸ 0BFF.

Система команд микропроцессора КР580ВМ80А приведена в Приложении 1.

Время выполнения команды измеряют в машинных тактах, равных периоду синхронизации (0,5 мкс при $f_т$ = 2 МГц).

Рассмотрим простейшую программу (программа 1), извлекающую число из ячейки памяти с адресом 0B00, инвертирующую его и записывающую результат в адрес памяти 0B01.

Программа 1 (в мнемокодах)

Мнемокод	Комментарий
LDA 0B00	Получить число из ячейки с адресом 0B00
CMA	Инвертировать число
STA 0B01	Записать результат по адресу 0B01
HTL	Прервать выполнение программы

При записи программ все числа представляются в шестнадцатеричной системе счисления.

Для записи программы в память необходимо перевести мнемокоды команд в машинные коды. Команды в программе могут быть одно–, двух– или трехбайтные и должны в памяти занимать соответственно один, два или три адреса.

Программа 1 (размещение по адресам памяти)

Адрес	Число	Комментарий
0800	3A	Код команды LDA
0801	00	Младший байт адреса
0802	0B	Старший байт адреса
0803	2F	Код команды CMA
0804	32	Код команды STA
0805	01	Младший байт адреса
0806	0B	Старший байт адреса
0807	76	Код команды HLT

В программе 1 используется прямой способ адресации. Приведем аналогичную программу с косвенной адресацией (программа 2).

Программа 2 (общий вид записи)

Адрес	*Машинный код*	*Мнемокод*	Комментарий
0800	21 00 0B	LXI H 0B00	Записать в регистры H, L число 0B00.
0803	7E	MOV A, M	Получить число из адреса указанного в регистрах H, L.
0804	2F	CMA	Инвертировать число в аккумуляторе.
0805	23	INX H	Увеличить на 1 число в регистрах H,L.
0806	77	MOV M, A	Записать число из аккумулятора по адресу указанному в регистрах H, L. (0B01)
0807	76	HLT	Прервать выполнение программы.

Задание для домашней подготовки

1. Изучить устройство УМК и порядок работы с ним.
2. Ознакомьтесь с языком программирования и структурой команд МП БИС КР580.
3. Изучите команды пересылки и загрузки, команды арифметических и логических операций.
4. Определите результат выполнения программы 1 при записи различных однобайтовых команд по адресу 0803, и занесите их в таблицу

Команда записанная по адресу 0803	Число записанное по адресу 0B00.	Число записанное по адресу 0B01.
CMA ADD A SUB A ANA A XRA A ORA A CMP A INR A DCR A		

Примечание – по адресу 0B00 запишите число соответствующее вашему порядковому номеру в журнале.

Разработайте программы:

Программа 3: увеличение на 5 числа, записанного по адресу 0B00, и записи результата по адресу 0B01 (программа 3);

Программа 4: сложение чисел, записанных по адресам 0B00 и 0B01, и запись результата по адресу 0B02 (программа 4);

Программа 5: сравнение чисел, записанных по адресам 0B00 и 0B01, и записи большего из них в ячейку по адресу 0B02 (программа 5).

Практическая часть

1. Изучите по рекомендованной литературе правила записи программ на языке Ассемблера, назначение и использование директив Ассемблера, их ввод и отладку с помощью

кросс-средств, обратив особое внимание на формальный синтаксис языка Ассемблер, запись директив Ассемблера, команды симулятора.

2. Пользуясь системой команд, составьте программу сложения двух однобайтовых чисел без знака:

а) числа последовательно вводятся с внешнего устройства с адресом 2;

б) числа находятся в ОЗУ по выбранным адресам.

В обоих случаях предусмотрите вывод результата на внешнее устройство с адресом 3 и сохранение в ячейке ОЗУ

Запишите в шестнадцатеричной системе счисления вводимые числа и результат, а также содержимое используемых РОН, аккумулятора и регистра флагов по завершении программы. Рассмотрите два случая: сумма не превышает значения 255 и сумма превышает 255. Программу оформите в виде таблицы.

Ход работы

1. Создайте свой каталог и скопируйте в него программы **av.bat, a85.com, avsim85.exe, avsim85.hlp** и подкаталог **error**.

Данную и все последующие лабораторные работы проводите только в своем каталоге!

2. В подкаталоге **error** записаны короткие программы, содержащие ошибки. Проассемблируйте и отладьте несколько примеров по указанию преподавателя.

3. В текстовом редакторе (Shift/F4) наберите подготовленную дома программу. Проассемблируйте ее и исправьте возможные ошибки.

4. Вызовите моделирующую программу Avsim85, загрузите прикладную программу и выполните ее в пошаговом режиме, отмечая изменения содержимого регистров МП и памяти. Загрузку и выполнение прикладной программы проведите для всех рассмотренных дома вариантов.

Содержание отчета

Отчет должен содержать домашнюю подготовку согласно пункту 2 подготовки к работе, результаты исправления примеров с ошибками (ошибочная и исправленная строка), результаты отладки и моделирования составленных программ.

Практическая работа №9. Процессоры общего назначения.

Цель: Ознакомление с эмулятором микропроцессора КР580, со структурой МП, с назначением выводов МП, со структурой памяти МП-системы, с форматами команд МП.

Руководство пользования эмулятором *КР580*

Эмулятор микропроцессора КР580ВМ80/КР580ВМ80А (МП КР580) является приложением для операционных систем Windows ’95/98/Me/NT, поэтому для него характерны свойства, присущие всем приложениям данных операционных систем. А именно: возможность управления размерами окон стандартными кнопками, возможность параллельной загрузки других приложений, возможность изменения цветов и параметров окон в зависимости от настройки интерфейса операционной системы, также реализованы всплывающие подсказки и пр.

Единственным файлом программы эмулятора МП КР580ВМ80 является файл emKP580.exe. Все необходимые данные для работы программы "вшиты" в этот файл. Это сделано для удобства переноса программы из одного компьютера на другой.

Создание программы на Ассемблере осуществляется в два этапа:

- ✓ Создание или считывание программы на Ассемблере для данного процессора. Этот этап выполняется в текстовом редакторе эмулятора, окно которого появляется после запуска приложения. После завершения написания программы в текстовом редакторе ее необходимо проверить на наличие ошибок. После проверки можно переходить ко второму этапу. Подробное описание редактора изложено в *п. 1.1.1., Текстовый редактор.*
- ✓ Эмуляция (симуляция) выполнения написанной программы для микропроцессора КР580. При этом можно отслеживать любое изменение состояний регистров, флагов и т. п. при выполнении как отдельной команды, так и группы команд. Если обнаруживается, что

выполняемые действия программы не удовлетворяют требованиям, необходимо вернутся на предыдущий этап и изменить текст программы. Подробное описание эмулятора изложено в *п. 1.1.3, Эмулятор программы.*

Текстовый редактор.

Текстовый редактор предназначен для ввода программы на Ассемблере для микропроцессора КР580. Заголовок редактора состоит из названия текущего файла и названия приложения. Предусмотрено изменение размеров окна редактора.

Как и другие приложения Windows, редактор имеет меню, в котором содержатся все выполняемые им действия. Большинство команд в меню имеют горячие клавиши, при нажатии соответствующей комбинации клавиш происходит автоматический вызов соответствующего пункта меню. Также редактор содержит панель инструментов с быстрыми кнопками, которые наиболее часто в нем используются. Кнопки дублируют соответствующие команды меню, поэтому их описание здесь не приводится. Каждая кнопка имеет смысловую иконку и подсказку, что облегчает ее понимание.

Статусная строка, находящаяся внизу окна редактора, показывает текущую строку, текущий столбец, был ли файл изменен и подсказку выполняемого действия при выборе того или иного пункта меню.

Меню редактора содержит следующие пункты:

▷ **Файл**. Сгруппированы все действия, которые можно выполнять над файлом. Сюда входит следующее пункты:

⇨ *Создать*. Создать новый файл. Создается новый файл под именем "Безимени". При этом перед созданием файла предлагается сохранить текущий файл.

⇨ *Открыть*. Открыть ранее созданный файл. Вызывается окно открытия файла, указывающее на текущую папку. В этом окне имеется возможность показа файлов с расширением "asm", файлов с расширением "txt", файлов с расширением "rtf" и файлов с любым расширением (*.*). Файлы с расширениями "asm" и "txt" должны содержать данные в обычном текстовом формате, а файлы с расширением "rtf" должны содержать данные в формате RTF.

⇨ *Сохранить*. Сохранить текущий файл. Если файл имеет имя "Безимени", то предлагается сохранить его под другим именем. Файл может быть сохранен в текстовом формате (с расширениями "asm" или "txt") либо в формате RTF (с расширением "rtf"). При этом по умолчанию, файлу присваивается расширение "asm". Если файл был открыт и изменен или сохранен под именем, отличным от "Безимени", и изменен, то просто выполняется сохранение, при этом старое содержимое файла теряется.

⇨ *Сохранить как…*.Сохранить текущий файл под другим именем. Здесь предлагается сохранить файл с некоторым именем под другим именем. Файл может быть сохранен в текстовом формате (с расширениями "asm" или "txt") либо в формате RTF (с расширением "rtf"). При этом по умолчанию, файлу присваивается расширение "asm".

⇨ *Печать*. Вывод текущего файла на принтер. Вызывается стандартное окно печати, где можно настроить параметры печати.

⇨ *Выход*.Завершение работы с приложением.

Дополнительно в меню "Файл" появляются четыре подпункта меню, которые показывают названия файлов, использованных в текущем сеансе работы с эмулятором. Названия файлов располагаются в порядке увеличения времени их открытия (первым располагается файл, открытый в последний раз). Текущий файл, если он еще не внесен в список, при открытии в этот список не заносится, а заносится только при его закрытии.

▷ **Правка**. Сгруппированы все действия, которые можно выполнять над текстом.

⇨ *Отмена*. Отмена в тексте последнего выполненного действия. При этом отменяется только одно действие, и повторное нажатие вернет выполненное действие.

⇨ *Вырезать*. Забирает выделенный фрагмент текста в карман.

⇨ *Копировать*. Копирует выделенный фрагмент текста в карман.

⇨ *Вставить*. Вставляет текст из кармана в текущую позицию курсора.

⇨ *Шрифт…*. Вызывает стандартное окно выбора шрифта и его параметров. При

отсутствии выделенного фрагмента текста изменения накладываются на весь текст.

▷ **Ассемблер**. Сгруппированы действия проверки на ошибки и запуска окна эмуляции программы.

⇨ *Ассемблировать*. Производится проверка на различные ошибки. Ограничения и правила приведены в пункте "Синтаксис редактора". Здесь весь текст проверяется на ошибки и в случае возникновения ошибки появляется поле ошибок, в котором указаны номер строки и вид ошибки. Курсор переводится на начало той строки, где возникла первая ошибка. Если ошибок не обнаружено, то поле ошибок не отображается и не выдается никаких сообщений. После этого можно запустить эмулятор.

⇨ *Эмулировать*. Отличается от предыдущего пункта тем, что после проверки на ошибки запускается эмулятор.

▷ **Помощь**.

⇨ *Помощь*. Вывод помощи текстового редактора.

⇨ *О программе…*. Вывод некоторой информации о программе и о ее разработчике.

Синтаксис редактора.

Число строк и длина строки в редакторе не ограничены. Ограничение накладывается на размер кода откомпилированной программы и не может превышать 64 Кбайта. Шрифт, его размер и другие параметры можно изменять по своему усмотрению. При записи программы на Ассемблере возможно применение как строчных, так и заглавных букв. В тексте допускается присутствие пустых строк. Разделители (пробелы и символы табуляции) можно ставить в любом месте, кроме тех случаев, которые указаны ниже. Количество последовательности разделителей не ограничено, то есть эту последовательность можно считать как один разделитель (пробел или символ табуляции). Часть строки, находящаяся за символом точка с запятой ";", считается пояснением команды, и не участвует в формировании машинного кода процессора.

Метка должна стоять в начале строки. Название метки может содержать любые символы кроме пробела и символа табуляции, однако рекомендуется задавать названия меток, используя только латинские символы и арабские цифры, причем для избежания непредсказуемой работы программы не следует начинать название метки с цифры. Конец метки определяется по символу двоеточие ":". Разделители перед меткой и после метки допускаются в любом количестве. В программе метка должна быть описана только один раз, количество же ссылок на эту метку не ограничено. Ссылка в команде на некоторую метку осуществляется путем написания в качестве операнда названия этой метки. Метка может находиться как до, так и после ссылки на нее.

Команда должна быть написана в соответствии с языком Ассемблера данного микропроцессора. Разделители перед командой и после команды допускаются в любом количестве. Между мнемоникой команды и первым операндом обязательно должен стоять хотя бы один разделитель (пробел или символ табуляции), разделяющий мнемонику команды от операнда. Если в команде присутствует второй операнд, то его необходимо отделить от первого операнда запятой. Разделители перед запятой и после нее допускаются в любом количестве. После команды не допускается присутствие других символов, кроме пояснения и разделителей.

В тексте программы имеется возможность задавать численные значения операндов в десятичной, шестнадцатиричной и двоичной системах счисления. При этом однобайтное число должно лежать в диапазоне от 0 до 255 включительно, а двухбайтное – от 0 до 65535 также включительно. При задании числа в шестнадцатиричной системе счисления сразу после числа ставится буква "h". При задании числа в двоичной системе счисления сразу после числа ставится буква "b". При отсутствии этих символов подразумевается десятичная система счисления.

Эмулятор программы.

Эмулятор программы предназначен для пошагового выполнения или выполнения написанной программы целиком. При этом можно отслеживать любое изменение состояний регистров, флагов, памяти, портов, счетчика команд и указателя стека при выполнении, как отдельной команды, так и группы команд. Заголовок окна состоит из названия текущего файла и названия приложения. Изменение размерами окна эмулятора не предусмотрено.

Эмулятор имеет меню, в котором содержатся все доступные действия. Все команды в меню имеют горячие клавиши, при нажатии соответствующей комбинации клавиш происходит

автоматический вызов соответствующего действия. Также эмулятор содержит панель инструментов с быстрыми кнопками, которые наиболее часто используются в его среде. Эти кнопки дублируют соответствующие команды меню, и поэтому их описание здесь не приводится. Каждая кнопка имеет смысловую иконку и подсказку, что облегчает ее применение.

Меню эмулятора содержит следующие пункты:

▷ **Команды**. Список всех команд вызываемых при просмотре или отладки работы программы.

⇨ *Выполнить до курсора*. Выполнение программы до выделенной строки в поле команд (*F4*).

⇨ *Выполнить одну команду*. Выполнение одной команды программы (*F7*).

⇨ *Выполнить процедуру*. Выполнение одной команды, не заходя в процедуры (*F8*). Процедурами считаются команды условного и безусловного перехода к подпрограммам, а также команды обработки прерываний.

⇨ *Запустить программу*. Запуск программы до ее завершения (*F9*).

⇨ *Прервать*. Остановка выполнения программы (*Esc*) при запуске ее до завершения. Это единственное действие, которое доступно во время отработки команд программы, оно позволяет в любой момент прерват выполнение программы процессором. При остановке выполнения программы курсор команды будет показывать на следующую выполняемую команду.

⇨ *Установить точку*. Установка точки останова в выделенной строке в поле команд (*Ctrl + F8*). При этом возможно установление только одной точки останова. Установка точки останова в строке, где она уже установлена, приведет к удалению из этой строки точки останова. Если точка останова устанавливается в другой строке, то предыдущая точка удаляется.

⇨ *Выход*. Выход из эмулятора в текстовый редактор (*Alt + x, Alt + F4*).

▷ **Помощь**. Содержит единственное подменю *Помощь*, которое выводит помощь по работе с эмулятором.

Эмулятор содержит следующие поля:

✓ *Поле команд*. В первом столбце указывается курсор выполняемой команды. Также здесь указывается точка останова (буквой "B"). Во втором столбце отображается адрес команды. В третьем столбце отображается код команды. Наконец, в последней строке отображается сама команда, при этом все числа и адреса представлены в шестнадцатиричной системе счисления. В этом поле нельзя произвести никаких изменений.

✓ *Поле регистров*. В первом столбце содержатся названия регистров микропроцессора. В последующих столбцах отображается содержимое регистров в шестнадцатиричной (h), десятичной (d) и в двоичной (b) системах счисления.

✓ *Поле указателей*. В первом столбце содержатся два названия: счетчик команд (PC) и указатель стека (SP). В последующих столбцах отображается содержимое указателей в шестнадцатеричной и десятичной системах счисления.

✓ *Поле флагов*. В первой строке содержатся названия флагов, а во второй – их значения. Флаги могут принимать только значения 0 или 1.

✓ *Поле параметров*. В первом столбце содержатся два названия: стартовый адрес и задержка. В последующих столбцах отображается содержимое параметров в шестнадцатеричной и десятичной системах счисления. Стартовый адрес изменяет счетчик команд на указанный адрес, а задержка нужна для замедления выполнения команды.

✓ *Поле портов*. В первом столбце содержатся номера портов в шестнадцатеричном виде. В последующих столбцах отображается содержимое портов в шестнадцатеричной, десятичной и в двоичной системе счисления.

✓ *Поле памяти*. В первом столбце содержится адрес базовой ячейки памяти строки в шестнадцатеричном виде. Эти адреса нумеруются через восемь значений. В последующих столбцах отображается содержимое ячеек памяти в шестнадцатеричном виде. Адрес ячейки памяти определяется путем сложения адреса базовой ячейки памяти данной строки и значения, указанного в первой строке в квадратных скобках. Таким образом, адрес ячейки вычисляется путем смещения данной ячейки относительно базовой ячейки в данной строке.

Все поля кроме первого, являются редактируемыми. В них можно изменять значение

нужного параметра. Редактирование значений возможно в любой, имеющейся в данном поле, системе счисления. При этом происходит соответствующее изменение содержимого других систем счисления.

Ввод всех редактируемых значений, кроме значений флагов регистра признаков, осуществляется следующим образом:

1. Сначала левой кнопкой мыши либо клавишей табуляции (*Tab*, *Shift* + *Tab*) и клавишами перемещения курсора (←, →, ↑, ↓) выбирается редактируемый параметр поля.
2. Затем нажимается клавиша *Enter* или левая кнопка мыши, после чего осуществляется ввод значения редактируемого параметра.
3. Ввод завершается только нажатием клавиши *Enter*. После ввода значения производится проверка. Если ввод неправильный, то остается старое (последнее, правильно введенное) значение.

Редактирование флагов производится иначе. Здесь для изменения значения некоторого флага с 0 на 1 или наоборот, с 1 на 0, достаточно нажать левую кнопку мыши на этом флаге. При этом не следует пользоваться клавишами ←, →, ↑ и ↓, т. к. их нажатие приводит не только к выбору редактируемого флага, но и к его изменению на противоположное значение, что затрудняет процесс редактирования данным способом.

Задания для самоподготовки

1. Ознакомиться с руководством пользования эмулятором emКР580 (*п. 1.1*).
2. Изучить типовую структуру МП КР580, назначение выводов МП, структуру памяти МП-системы, форматы команд МП по вводному разделу практикума и по [1, 2, 3].
3. Изучить работу программы *Прогр. 1.1*.

Прогр. 1.1. Первая программа на языке Ассемблер.

```
        LXI H,  ;В HL адрес ячейки памяти.
        0100h
        MVI M, 120  ;Записать байт данных в ячейку с
                    адресом в
                    ;  HL.
        MVI B, 24   ;Записать вВ байт данных.
        MOV A, B    ;Записать содержимое регистра В в Акк.
        ADD M       ;Сложить Акк. с байтом по адресу в HL.
        JPO LAB_1   ;Если не установлен флаг Р, то идти на
                    ;  LAB_1.
        DCR B       ;Декрементация регистра В.
        INX H       ;Инкрементация регистровой пары HL.
        CALL        ;Переход на подпрограмму PROC_1.
        PROC_1
        MOV M, B    ;Записать содержимое регистра В в
                    память по
                    ;  адр. в HL.
        SUB B       ;Вычесть из Акк. байт в В и занести
                    ;  результат в Акк.
        HLT         ;Выход из программы.
LAB_
1:
      P MOV A, B    ;Записать содержимое регистра В в Акк.
ROC_
1:
        RLC         ;Сдвиг Акк. влево.
        MOV B, A    ;Записать содержимое Акк. в регистр В.
        RET         ;Возврат из подпрограммы.
```

Задание к практической работе.

Задание 1.1. Отладка и исследование первой программы на языке Ассемблер.

Порядок выполнения задания:

1. Ввести программу *Прогр. 1.1* в текстовом редакторе эмулятора. Сохранить программу (*Ctrl + s*).
2. Выполнить ассемблирование программы (*F9*). Если в результате ассемблирования эмулятор выдает сообщения об ошибках, то необходимо найти и исправить ошибки в программе, после чего повторить процедуру ассемблирования.
3. Запустить эмулятор программы (*Ctrl + F9*). Ознакомиться с адресным пространством памяти и устройств ввода/вывода (портов) эмулятора.
4. Проанализировать работу программы в пошаговом режиме (*F7*). Выполнить программу за один прием (*F9*). Выполнить программу в пошаговом режиме без захода в подпрограмму (*F8*).
5. Проверить работу программы при различных числовых значениях адреса и данных (строки 1 – 3). Проверить правильность выполнения программой арифметических и логических операций.
6. В 6-й строке поставить другое условие перехода (переход по другому признаку, по выбору студента). Повторить *п.* 5 данного задания.
7. Осуществить непосредственную запись данных в ячейку памяти, порт ввода/вывода, РОН (ручной ввод).

Содержание отчета

Отчет должен содержать:

1. Название работы, фамилию и инициалы студента, номер группы, цель работы.
2. Структурную схему МП КР580.
3. Описание выводов МП КР580.
4. Структурную схему памяти МП КР580.
5. Форматы команд МП КР580.
6. Команды эмулятора emКР580 (кратко, в том числе команды его текстового редактора).
7. Программу *Прогр. 1.1*с комментариями и в печатном виде.
8. Результаты выполнения задания к лабораторной работе.
9. Выводы по лабораторной работе.

Контрольные вопросы

1. Какие бывают форматы 1- 2-х и 3-хбайтных команд?
2. Назначение выводов процессора.
3. Какова структура памяти МП КР580?
4. Как осуществить в МП-системе обмен данными с портом ввода/вывода, с ячейкой памяти?
5. Что такое и для чего нужна десятичная коррекция данных.
6. Для чего предназначены механизмы прямого доступа к памяти и обработки прерываний?
7. Назначение общих регистров процессора (РОН), указателя стека, счетчика команд, регистра флагов.
8. Какие поля содержит эмулятор emКР580 во время симуляции процесса выполнения программы?
9. Работа программы по командам, т. е. что делает каждая команда программы?

Практическая работа №10. Исследование процессора (регистровая память).

Цель работы: изучить состав и назначение регистров процессора.

Краткие сведения из теории

Процессор содержит ряд узлов, выполняющих различные функции, такие, как арифметическо-логическое устройство (АЛУ), используемое для осуществления операций над данными, дешифратор команд (ДК) и устройство управления (УУ), которые анализируют команды, поступающие из программной памяти, и генерируют необходимые импульсы для выполнения этих операций, а также ряд регистров. Процессор КР580 имеет семь регистров общего назначения, обозначаемых A, B, C, D, E, H, L. Они предназначены для хранения как постоянных, так и переменных данных и адресов. В структуре процессора предусмотрена реализация команд пересылки данных из одного регистра в другой, из регистра в память и наоборот, или же команд для выполнения арифметических или логических операций над содержимым двух регистров (в этом случае один из регистров обязательно должен быть регистром A, например, нельзя суммировать содержимое регистров B и C).

Регистр A особый, называется аккумулятором, т. к. в нем аккумулируются результаты ряда арифметических операций.

Регистр команд (РК) хранит команду, которая выполняется. Он соединен с дешифратором команд, который анализирует команду и определяет вид обработки. Команда хранится и используется в течение всего времени ее выполнения.

Буферные регистры данных и адреса предназначены для временного хранения данных и адреса с целью обеспечения нормальной работы других узлов микро-ЭВМ.

Регистры счетчик команд (СК, PC), указатель стека (УС, SP) и регистр флагов (F), играют очень важную роль в программировании микропроцессора.

Все пять бит регистра признаков (F) устанавливаются по результату выполнения операции в АЛУ:

- ✓ Признак переноса (C) устанавливается в 1, если при выполнении команд появляется единица переноса из старшего разряда.
- ✓ Дополнительный признак переноса (AC) устанавливается в 1, если при выполнении команд возникает единица переноса из третьего разряда числа. Состояние разряда может быть проанализировано лишь командой десятичной коррекции.
- ✓ Признак знака (S) устанавливается в 1, если седьмой (старший) разряд числа равен 1. Машинное слово представляется числом от –128 до +127. Седьмой разряд числа указывает на знак числа. Если он равен 0, то число положительное, если 1, то отрицательное.
- ✓ Разряд признака нулевого содержимого аккумулятора (Z) устанавливается в 1, если при выполнении команды результат равен нулю.
- ✓ Разряд признака паритета (P) устанавливается в 1, если число единичных битов аккумулятора четно, в противном случае этот разряд будет установлен в нулевое состояние.

Счетчик команд (СК) содержит текущий адрес памяти, к которому обращается программа. Его содержимое автоматически изменяется в течение каждого цикла команды.

Указатель стека (УС) содержит адрес стековой памяти, начиная с которого ее можно применять для хранения и восстановления содержимого программно доступных регистров МП.

Выполнение каждой команды производится МП в строго определенной последовательности, определяемой кодом команды, и синхронизируется во времени сигналами C_1 и C_2 тактового генератора. Период синхросигналов C_1 или C_2 называют машинным тактом (МТ). Длительность машинного такта от 0,5 до 2 мкс.

Машинный цикл (МЦ) – время, требуемое для извлечения 1 байта информации из памяти или выполнения команды, определяемой одним машинным словом. Машинный цикл может состоять из 3 – 5 машинных тактов. Время выполнения команды – время получения, дешифрации и отработки команды процессором. В зависимости от вида команды, это время может состоять из 1 – 5 машинных циклов. Существуют десять различных типов машинных циклов:

1. Извлечение кода команды.
2. Чтение данных из памяти.
3. Запись данных в память.
4. Извлечение из стека.
5. Запись данных в стек.
6. Ввод данных из внешнего устройства.

7. Запись данных во внешнее устройство.
8. Цикл обслуживания прерывания.
9. Останов.
10. Обслуживание прерывания при работе в режиме останова.

Первым машинным циклом при выполнении любой команды является извлечение кода команды. На первом такте каждого машинного цикла МП указывает тип выполняемого цикла с помощью 8-разрядного слова состояния, выдаваемого на ШД. Отдельные разряды слова состояния используются для формирования шины управления.

Способы адресации памяти.

- ✓ Прямая адресация.

При прямой адресации команда содержит адрес памяти. Команда занимает три байта памяти, причем второй и третий байты содержат адрес.

Пример:

Загрузить содержимое ячейки 1F2A в аккумулятор.

Мнемоника команды	Запись в памяти			
		3	2	1
LDA 1F2A		A	A	F

- ✓ Косвенная адресация.

При косвенной адресации байт адресуется через пару регистров, т. е. адрес ячейки памяти может быть определен с помощью содержимого пары регистров. Для большинства команд используется регистры H и L. Регистр H содержит старший байт адреса, регистр L – младший.

Пример:

Загрузить аккумулятор содержимым ячейки 1F2A.

Память			Регистры							
	E						F	A		

MOV A, M

- ✓ Непосредственная адресация.

Команда с непосредственной адресацией содержит в одном из своих полей операнд. Команды с непосредственной адресацией не обращаются к памяти для извлечения операнда – они сами содержат операнд.

Пример:

Загрузить в аккумулятор величину 2A.

Память

	E	A	

MVI A, 2A

- ✓ Адресация через указатель стека.

Здесь адресация может осуществляться через 16-разрядный регистр – УС. Существуют две операции со стеком: запись в стек (PUSH) и выборка из стека (POP). Запись в стек используется для пересылки 16 бит данных из пары регистров или из программного счетчика (СК) в область памяти, отведенную под стек. Извлечение из стека используется для пересылки 16 бит из

стековой области памяти, в любую пару регистров или в программный счетчик. Подробнее эти операции будут рассмотрены позже.

Практическая часть

1. Изучить способы адресации МП КР580.
2. Ознакомиться с языком программирования Ассемблер и с программированием в машинных кодах для МП БИС КР580.
3. Ознакомиться с системой команд МП КР580.
4. Рассмотреть особенности выполнения команд пересылки данных, арифметических и логических команд.
5. Самостоятельно составить программу сравнения двух 8-разрядных чисел, находящихся в ячейках памяти с адресами 0100h и 0101h, и записи большего из них в память по адресу 0102h.

Задание 1. Исследование программы по выполнению арифметических и логических операций с одним операндом (над содержимым аккумулятора).

Порядок выполнения задания:

1. Ввести программу *Прогр. 2.1*, исправить ошибки, осуществить пуск.
2. Исследовать процесс выполнения программы в пошаговом режиме (обратить особое внимание на изменения регистра флагов).
3. Заменяя во второй строке программы число, записываемое в регистр B, исследовать программу при различном значении этого числа.
4. Занести результаты выполнения программы в таблицу*Табл. 2.2,* числа в таблицу заносить в 16-ричной системе счисления).

Контрольные вопросы

1. Способы адресации МП КР580.
2. Что такое машинный такт и машинный цикл?
3. Какие бывают в МП КР580 машинные циклы?
4. Назначение общих регистров процессора (РОН), указателя стека, счетчика команд, регистра флагов.

Практическая работа № 11. Исследование процессора К580 (система команд).

Цель работы: изучить систему команд процессора К580.

Теоретические сведения.

Команды МП можно разделить на 4 категории:

- ✓ Команды пересылки данных (см. *Табл. 0-4 – Табл. 0-7*).

Эти команды приводят к перемещению элементов информации между регистрами или между регистром и ячейкой памяти (портом ввода/вывода). Общее число таких команд велико, т. к. имеется много видов адресации, а также большое число ограничений на форматы команд, обусловленное небольшой длиной слова процессора. Это приводит к усложнению архитектуры.

- ✓ Арифметические и логические команды (см. *Табл. 0-8 – Табл. 0-10*).

К ним относятся команды, объединяющие два элемента данных арифметически или логически, или выполняют операции над одним числом. Для этого используется регистр А.

- ✓ Команды передачи управления (см. *Табл. 0-11 – Табл. 0-13*).

Они включают в себя команды разветвления, перехода к подпрограммам и возврата из подпрограмм.

- ✓ Команды различного назначения (специальные команды, см. *Табл. 0-14*).

К данному классу относятся команды, которые нельзя точно классифицировать.

Команды пересылки данных.

Команды пересылки "регистр-регистр" – MOV R1, R2. Формат этих команд имеет вид 01 DDD SSS, где DDD – регистр приемник, SSS – регистр источник. Регистры определяются номерами B = 0, C = 1, D = 2, E = 3, H = 4, L = 5, A =7. Таким образом, команда 4C (01001100) означает "переслать содержимое регистра H в регистр C". Если значение DDD или SSS равно 110, то приемником или источником является ячейка памяти, адрес которой хранится в паре регистров HL (косвенная адресация).

Команда MVI служит для пересылки одного байта в любой из 7 РОН. Формат команды 00 DDD 110. Если DDD = 0, 1, ..., 5, 7, то пересылается константа, следующая в программе за этой командой в регистры B, C, ..., L, A. Если поле DDD = 6, то константа пересылается в ячейку памяти, адрес которой хранится в паре регистров H, L.

Для быстрой загрузки регистров H, L существует команда LXI H, пересылающая следующие за командой LXI H два байта (в программе) в указанную пару регистров. Таким же образом команда LXI D загружает два байта в пару регистров D и E, а команда (LXI B) – два байта в регистры B и C. Эти команды называются командами непосредственной загрузки.

Существуют команды, выделяющие регистр A для специальных целей. Команды STAX и LDAX служат для передачи данных между регистром A и ячейкой памяти, адрес которой находится либо в паре регистров BC, либо в паре регистров DE, причем STAX B – пересылает содержимое A в ячейку памяти, адрес которой находится в BC, а LDAX D – осуществляет обратную передачу содержимого ячейки памяти с адресом, записанным в DE в регистр A.

В выше приведенных командах доступ к ячейке памяти становится возможным только после помещения адреса в пару регистров. Это эффективно, если неоднократно используется один и тот же адрес. При одноразовом использовании адреса используют метод абсолютной адресации.

К этой же группе команд могут быть отнесены команды ввода/вывода. Команды ввода/вывода IN и OUT осуществляют передачу данных между процессором и каналом ввода/вывода (портом ввода/вывода), номер которого приведен вслед за командой.

Арифметические и логические команды.

К арифметическим командам относятся: команды сложения и вычитания с учетом (ADD, ADI, SUB и SUI) и без учета (ADC, ACI, SBB и SBI) флага переноса, команды инкрементации и декрементации (INR, DCR, INX и DCX), команда десятичной коррекции аккумулятора (DAA) и команда DAD, выполняющая увеличение регистровой пары HL на величину, содержащуюся в другой регистровой паре (BC или DE).

К логическим командам относятся команды логического И, ИЛИ, исключающего ИЛИ, НЕ; команды сдвига влево и вправо с учетом и без учета переноса; команда инверсии аккумулятора (CMA); команды сравнения (CMP, CPI). Сюда же можно отнести несколько команд изменения содержимого триггера переноса в регистре флагов: STC – установить в единицу триггер переноса; CMC – инвертировать содержимое триггера переноса.

Команды передачи управления.

Команды передачи управления делятся на три типа: ветвления, обращения к подпрограммам и возврата из подпрограмм. Адреса команд ветвления хранятся за 1-м байтом команды.

Для команд перехода к подпрограммам характерно наличие адресов возврата, хранимых в стеках, а команды возврата из подпрограмм не имеют ассоциированных с ними адресов, они получают адреса возврата из стека.

Каждый из трех видов перехода может быть безусловным и условным в соответствии с одним из двоичных разрядов регистра признаков. Команды JMP, CALL, RET предназначены для осуществления безусловных переходов.

Команда PCHL приводит к тому, что содержимое регистров HL передается в программный счетчик (PC). Это переход на ячейку памяти, указанную парой HL (безусловный переход с косвенной адресацией). Команда RST – команда начального запуска прерывания программы, которая является специально командой перехода, используемой совместно с процедурами прерывания.

Есть и другие команды, относящиеся к данной группе (см. *Табл. 0-11 – Табл. 0-13*).

Специальные команды.

К ним относятся команды EI и DI – разрешающие и, соответственно, запрещающие прерывания; NOP – пустая команда, не выполняющая никакой операции, но включаемая в программу для коррекции времени выполнения или последовательности команд и HLT – команда останова.

Все вышеприведенные команды, а также и некоторые неупомянутые другие команды приведены в*Табл. 2.1*.

Табл. 2.1. Команды микропроцессора КР580.

	Мнемоническое обозначение	Описание команды	2-й или 16-й код	Число тактовых импульсов
		Передача, загрузка и хранение		
	MOV r1r2	Передать содержимое одного регистра в другой.	01DDDSSS	5
	MOV M r	Передать содержимое регистра в память.	01110SSS	7
	MOV r M	Передать содержимое памяти в регистр.	01DDD110	7
	MVI r	Загрузить регистр вторым байтом команды.	00DDD110	7
	MVI M	Загрузить память вторым байтом команды.	00110110	10
	LXI B	Загрузить пару регистров BC вторым и третьими байтами команды.	00000001	10
	LXI D	Загрузить пару регистров DE вторым и третьими байтами команды.	00010001	10
	LXI H	Загрузить пару регистров HL вторым и третьим байтами команды.	00100001	10
	STAX B	Записать содержимое аккумулятора в память по адресу, указанному в паре регистров BC.	00000010	7
0	STAX D	Записать содержимое аккумулятора в память по адресу, указанному в паре регистров DE.	00010010	7
1	LDAX B	Загрузить аккумулятор содержимым ячейки, адрес которой указан в паре регистров BC.	00001010	7
2	LDAX D	Загрузить аккумулятор содержимым ячейки, адрес которой указан в паре регистров DE.	00011010	7

3	STA	Загрузить содержимое аккумулятора в память по адресу, указанному во втором и третьем байтах команды.	00110010	3
4	LDA	Загрузить аккумулятор содержимым ячейки, адрес которой указан во втором и третьем байтах команды.	00111010	3
5	SHILD	Записать в память содержимое пары регистров HL по адресу, указанному во втором и третьем байтах команды.	00100010	6
6	LHLD	Загрузить пару регистров HL содержимым ячейки, адрес которой указан во втором и третьем байтах команды.	00101010	6
7	XCHG	Поменять местами содержимое пар регистров DE и HL.	11101011	4
		Операции со стеком		
8	PUSH B	Записать содержимое пары регистров BC в стек.	11000101	1
9	PUSH D	Записать содержимое пары регистров DE в стек.	11010101	1
0	PUSH H	Записать содержимое пары регистров HL в стек.	11100101	1
1	PUSH PSW	Записать содержимое аккумулятора и регистра признаков в стек.	11110101	1
2	POP B	Загрузить пару регистров BC из стека.	11000001	0
3	POP D	Загрузить пару регистров DE из стека.	11010001	0
4	POP H	Загрузить пару регистров HL из стека.	11100001	0
5	POP PSW	Загрузить аккумулятор и регистр признаков из стека.	11110001	0
6	XTHL	Поменять местами содержимое верхней ячейки стека и пары регистров HL.	11100011	8
7	SPHL	Передать содержимое пары регистров HL в указатель стека.	11111001	5
8	LXI SP	Загрузить указатель стека вторым и третьими байтами команды.	00110001	0
9	INX SP	Увеличить на 1 содержимое указателя стека.	00110011	5

0	DCX SP	Уменьшить на 1 содержимое указателя стека.	00111011	5
		Переходы		
1	JMP	Безусловный переход.	11000011	0
2	JC	Условный переход по единице триггера переноса.	11011010	0
3	JNC	Условный переход по нулевому значению триггера переноса.	11010010	0
4	JZ	Условный переход по нулевому значению результата.	11001010	0
5	JNZ	Условный переход по ненулевому значению результата.	11000010	0
6	JP	Условный переход по положительному значению результата.	11110010	0
7	JM	Условный переход по отрицательному значению результата.	11111010	0
8	JPE	Условный переход по четности кода результата.	11101010	0
9	JPO	Условный переход по нечетности кода результата.	11100011	0
0	PCHL	Передать содержимое пары регистров HL в программный счетчик.	11101001	5
		Вызовы		
1	CALL	Безусловный переход к подпрограмме.	11001101	7
2	CC	Переход к подпрограмме по единичному значению триггера переноса.	11011100	1/17
3	CNC	Переход к подпрограмме по нулевому значению результата.	11010100	1/17
4	CZ	Переход к подпрограмме по нулевому значению результата.	11001100	1/17
5	CNZ	Переход к подпрограмме по ненулевому значению результата.	11000100	1/17
6	CP	Переход к подпрограмме по положительному значению результата.	11110100	1/17
7	CM	Переход к подпрограмме по отрицательному значению результата.	11111100	1/17
8	CPE	Переход к подпрограмме по четности кода результата.	11101100	1/17

9	CPO	Переход к подпрограмме по нечетности кода результата.	11100100	1/17
		Возвраты		
0	RET	Возврат из подпрограммы.	11001001	0
1	RC	Условный возврат из подпрограммы по единичному значению триггера переноса.	11011000	/11
2	RNC	Условный возврат из подпрограммы по нулевому значению триггера переноса.	11010000	/11
3	RZ	Условный возврат из подпрограммы по нулевому значению результата.	11001000	/11
4	RNZ	Условный возврат из подпрограммы по ненулевому значению результата.	11000000	/11
5	RP	Условный возврат из подпрограммы по положительному значению результата.	11110000	/11
6	RM	Условный возврат из подпрограммы по отрицательному значению результата.	11111000	/11
7	RPE	Условный возврат из подпрограммы по четности кода результата.	11101000	/11
8	RPO	Условный возврат из подпрограммы по нечетности кода результата.	11100000	/11
		Рестарт		
9	RST	Начальный запуск прерывающей программы.	11AAA111	1
		Увеличение и уменьшение.		
0	INR	Увеличить содержимое регистра на единицу.	00DDD100	
1	DCR	Уменьшить содержимое регистра на единицу.	00DDD101	
2	INR M	Увеличить содержимое памяти на единицу.	00110100	0
3	DCR M	Уменьшить содержимое памяти на единицу.	00110101	0
4	INX B	Увеличить на единицу содержимое пары регистров BC.	00000011	
5	INX D	Увеличить на единицу содержимое пары регистров DE.	00010011	

6	INX H	Увеличить на единицу содержимое пары регистров HL.	00100011	5
7	DCX B	Уменьшить на единицу содержимое пары регистров BC.	00001011	5
8	DCX D	Уменьшить на единицу содержимое пары регистров DE.	00011011	5
9	DCX H	Уменьшить на единицу содержимое пары регистров HL.	00101011	5
		Сложение		
0	ADD	К содержимому аккумулятора прибавить содержимое регистра.	10000SSS	4
1	ADC	К содержимому аккумулятора прибавить содержимое регистра и триггера переноса.	10001SSS	4
2	ADD M	К содержимому аккумулятора прибавить содержимое памяти.	10000110	7
3	ADC M	К содержимому аккумулятора прибавить содержимое памяти и триггера переноса.	11000110	7
4	ADI	К содержимому аккумулятора прибавить второй байт команды.	11000110	7
5	ACI	К содержимому аккумулятора прибавить второй байт команды и содержимое триггера переноса.	11001110	7
6	DAD B	К содержимому пары регистров HL прибавить содержимое пары регистров BC.	00001001	10
7	DAD D	К содержимому пары регистров HL прибавить содержимое пары регистров DE.	00011001	10
8	DAD H	К содержимому пары регистров HL прибавить содержимое пары регистров DE.	00101001	10
9	DAD SP	К содержимому пары регистров HL прибавить содержимое указателя стека.	00111001	10

Практическая часть

Прогр. 2.1. Программа по выполнению арифметических и логических операций с одним операндом.

```
        LHLD 0100h  ;В HL адрес ячейки памяти (начало массива).
        MVI B, 1Eh  ;Записать число в регистр B (исходное число).
        MOV A, B    ;    Записать исходное число в Акк.
        CMA         ;1. Инверсия Акк.
        CALL        ;    Переход на подпрограмму PROC_1.
        PROC_1
        INR A       ;2. Инкрементация Акк.
        CALL        ;    Переход на подпрограмму PROC_1.
        PROC_1
        DCR A       ;3. Декрементация Акк.
        CALL        ;    Переход на подпрограмму PROC_1.
        PROC_1
        ADD A       ;4. Умножить Акк. на 2
        CALL        ;    Переход на подпрограмму PROC_1.
        PROC_1
        ANA A       ;5. Выполнить над Акк. операцию лог. И.
        CALL        ;    Переход на подпрограмму PROC_1.
        PROC_1
        ORA A       ;6. Выполнить над Акк. операцию лог. ИЛИ.
        CALL        ;    Переход на подпрограмму PROC_1.
        PROC_1
        CMP A       ;7. Сравнить Акк с самим собой.
        CALL        ;    Переход на подпрограмму PROC_1.
        PROC_1
        DAA         ;8. Выполнить десятичную коррекцию над Акк.
        CALL        ;    Переход на подпрограмму PROC_1.
        PROC_1
        HLT         ;Завершение программы.
      P MOV M, A    ;    Записать содержимое Акк в ячейку по
ROC_                ;     адресу из HL.
1:
        INX H       ;    Инкрементация регистровой пары HL.
        MOV A, B    ;    Записать исходное число в Акк.
        RET         ;Возврат из подпрограммы.
```

Табл. 2.2. Результаты выполнения программы Прогр. 2.1.

		MA	NR	CR	DD	NA	RA	MP	AA
	Eh	1h	Fh	Dh	Ch	Eh	Eh	Eh	0
	..	..	..	..	..	..	..	..	..
..									

Задание 2.2. Исследование программы по выполнению арифметических и логических операций с двумя операндами (один из операндов – аккумулятор).

Порядок выполнения задания:

1. Ввести программу *Прогр. 2.2*, исправить ошибки, осуществить пуск.
2. Исследовать процесс выполнения программы в пошаговом режиме (обратить особое внимание на изменения регистра флагов).
3. Заменяя в третьей строке программы число, записываемое в регистр C,

исследовать программу при различном значении этого числа.

4. Занести результаты выполнения программы в таблицу*Табл. 2.3* (на стр. 39), числа в таблицу заносить в 16-ричной системе счисления).

Прогр. 2.2. Программа по выполнению арифметических и логических операций с двумя операндами.

```
        LHLD 0100h  ;В HL адрес ячейки памяти (начало
                    массива).
        MVI B, 7Ch  ;Записать число в регистр B (первый
                    ; операнд).
        MVI C, 1Eh  ;Записать число в регистр C (второй
                    ; операнд).
        MOV A, B    ;   Записать 1-й операнд в Акк.
        ADD C       ;1. Сложение операндов.
        CALL        ;   Переход на подпрограмму PROC_1.
        PROC_1
        SUB C       ;2. Вычитание 2-го операнда из 1-го.
        CALL        ;   Переход на подпрограмму PROC_1.
        PROC_1
        ANA C       ;3. Выполнение над операндами
                    операции
                    ;   лог. И.
        CALL        ;   Переход на подпрограмму PROC_1.
        PROC_1
        ORA C       ;4. Выполнение над операндами
                    операции
                    ;   лог. ИЛИ.
        CALL        ;   Переход на подпрограмму PROC_1.
        PROC_1
        XRA C       ;5. Выполнение над операндами
                    операции
                    ;   лог.исключающего ИЛИ.
        CALL        ;   Переход на подпрограмму PROC_1.
        PROC_1
        CMP C       ;6. Сравнение операндов.
        CALL        ;   Переход на подпрограмму PROC_1.
        PROC_1
        HLT         ;Завершение программы.
    P MOV M, A      ;   Записать результат (Акк) в ячейку по
ROC_                ;    адресу из HL.
1:
        INX H       ;   Инкрементация регистровой пары
                    HL.
        MOV A, B    ;   Записать исходное число в Акк.
        RET         ;Возврат из подпрограммы.
```

Табл. 2.3. Результаты выполнения программы Прогр. 2.2.

			DD	UB	NA	RA	RA	MP
	Ch	Eh	Ah	Eh	Ch	Eh	2	C
	Ch	..	..	..	..	..	..	..
..	Ch							

Задание 2.3. Исследование программы сравнения двух чисел из памяти и записи большего из них обратно в память.

Порядок выполнения задания:

1. Ввести программу, разработанную при выполнении *п.* 5 задания для самоподготовки (см. выше, стр. 37).
2. Осуществить пуск программы и проверить результат сравнения следующих чисел: 33h и 2Eh, D8h и 25h, 98h и A5h.

Контрольные вопросы.

1. Какие бывают команды пересылки, арифметические команды, логические команды, команды передачи управления, специальные команды?
2. Как работают команды CMP, CPI?

Практическая работа №12. 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.

Таблица 1

Тест проверки функционирования ОЗУ

Адрес	0 1 2 3 4 5 6 7 8 9 A B C D E F
800	31 EO 83 21 EO 80 AF 37 17 77 BE CA 12 80 CD CE
801	02 76 2F 77 BE C2 OE 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 № 13. Принцип организации ввода-вывода в микропроцессорной системе.

Цель: изучение режимов работы и принципов программирования БИС периферийного адаптера К580ВВ55А.

Основные сведения

Программируемые периферийные адаптеры (ППА) предназначены для организации связи между внешними устройствами и микропроцессором (МП). Наиболее часто такие адаптеры используются для реализации сложных интерфейсов, когда логика обмена заранее не известна, или когда характеристики процедур обмена должны меняться во время работы микропроцессорной системы. Номенклатура БИС ППА довольно обширна, но наиболее часто используются следующие БИС: i8255A – ППА фирмы Intel (отечественный аналог КР580ВВ55А) и MC6820 – ППА фирмы Motorola.

Рассмотрим основные принципы организации и функционирования ППА на примере БИС КР580ВВ55А (ВВ55А). Это однокристальное программируемое устройство параллельного ввода/вывода информации произвольного формата. Адаптер позволяет осуществлять параллельный обмен данными в режиме программного управления или по прерываниям. При этом обеспечивается организация как однонаправленного, так и двунаправленного ввода/вывода. Определение и переопределение типа интерфейса выполняется программными методами с помощью специальных процедур инициализации.

Структурная схема программируемого периферийного адаптера ВВ55А приведена на рис. 1. В состав ППА входят три двунаправленных 8-разрядных порта (А, В и С), разбитых на две группы, два устройства управления группами портов и интерфейсная логика для согласования с системной магистралью. Порты содержат буферные регистры и шинные формирователи с тремя состояниями. Схема управления содержит регистр управляющего слова CW, который доступен только для записи и определяет режимы работы ППА. Программный доступ к такому регистру дает возможность оперативно управлять работой адаптера и изменять характеристики интерфейса в соответствии с текущей необходимостью.

Обмен информацией между МП и внутренними регистрами ППА осуществляется через двунаправленный шинный формирователь и управляется сигналами CS, A0, A1, RD и WR в соответствии с требованиями к шине Microbus. Адресные сигналы (A0 и A1) выбирают один из внутренних регистров, а стробы RD и WR управляют направлением передачи согласно табл. 1. Вход RESET служит для аппаратного сброса БИС в исходное состояние, при этом все внутренние регистры ППА, включая регистр управляющего слова CW, обнуляются.

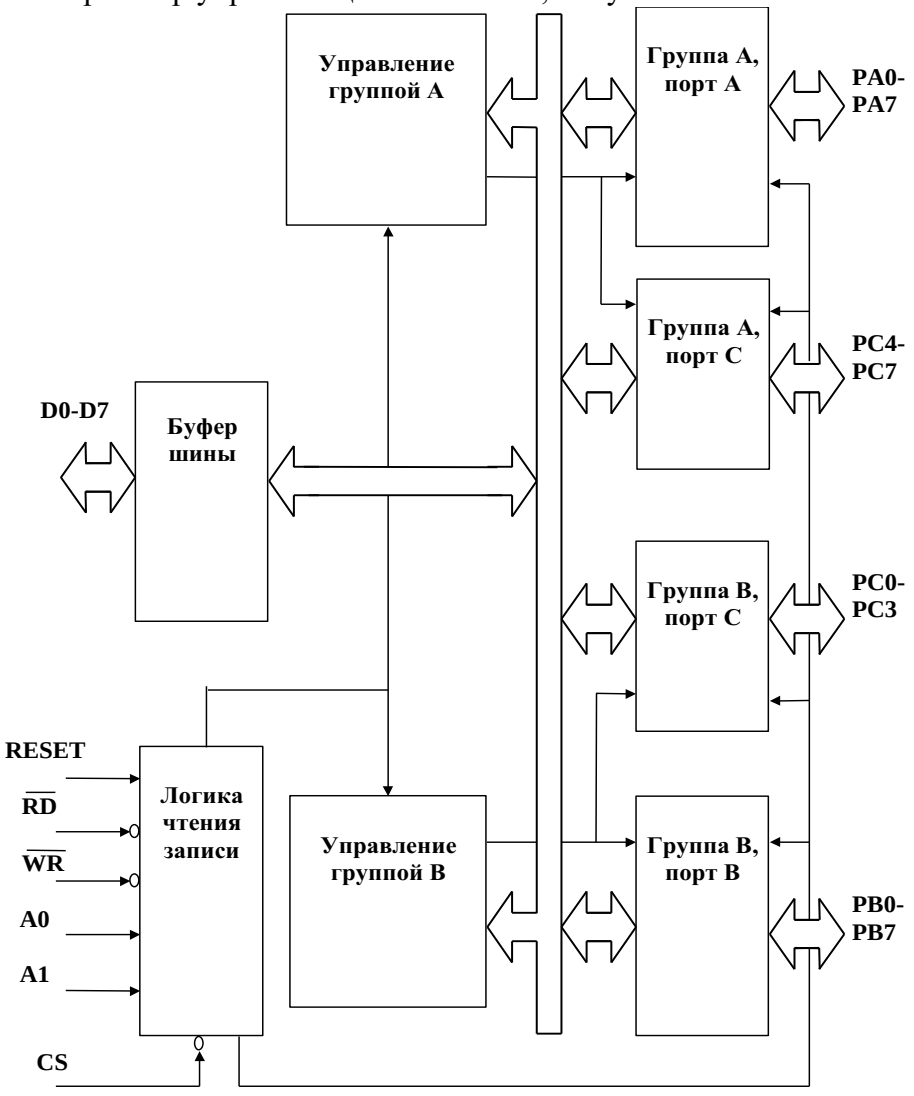


Рис. 1. Структурная схема БИС ППА К580ВВ55А

Настройка ППА выполняется программно с помощью специального управляющего слова MS (Mode Selection), которое назначает режим работы каждому каналу. Эти режимы могут быть изменены в любое время. Для хранения MS используется 7-разрядный регистр управляющего слова CW, запись в который осуществляется при передаче в ППА управляющего слова с установленным в «1» битом D7. Управляющее слово определяет режим работы каждого канала ВВ в соответствии с форматом, приведённым на рис. 2.

Таблица 1

$\overline{1}$	$\overline{0}$	$\overline{D}$	WR	S	ОПЕРАЦИЯ
			1		D← Порт A
			1		D← Порт B
			1		D← Порт C
			1		Не допустимо
			0		Порт A←D
			0		Порт B←D
			0		Порт C←D
			0		Управление←D
			1		Нет операции
			X		Нет операции

Устройства управления групп A или B воспринимают только свою часть управляющего слова. При записи нового управляющего слова все буферные регистры портов обнуляются.

Адаптер поддерживает три режима работы портов:

режим 0 - однонаправленный ВВ без квитирования (применим к любому из трех портов). Код режима - “00”;

режим 1 - однонаправленный ВВ с квитированием (применим к портам A и B). Код режима – “01”;

режим 2 - двунаправленный ВВ (допускается только для порта A).

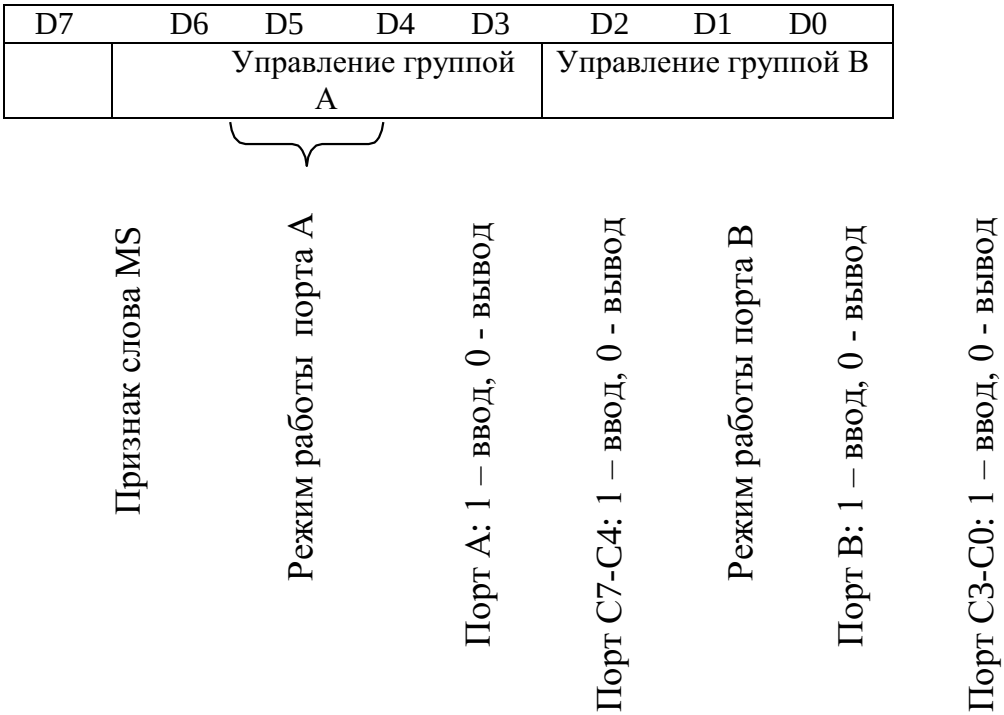


Рис. 2. Формат регистра CW

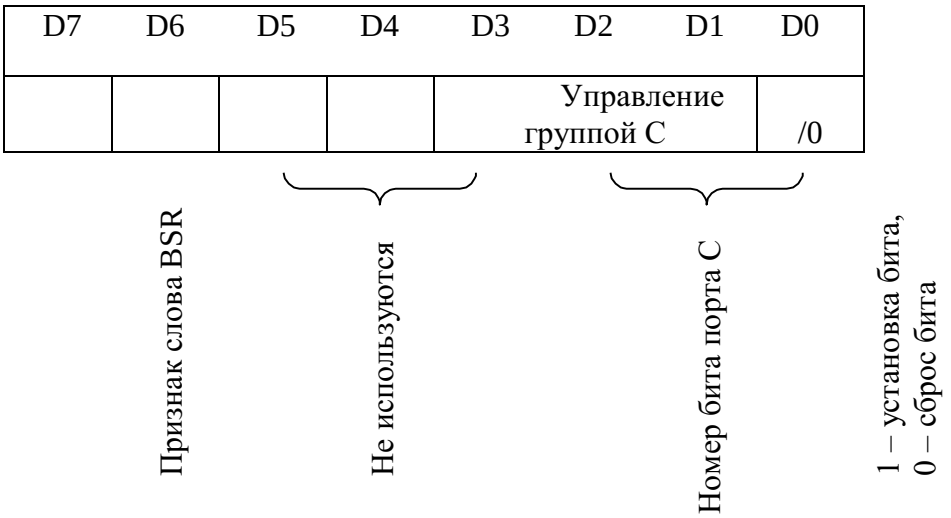


Рис. 3. Формат регистра BSR

При работе портов А и В в режимах 1 и 2 часть линий порта С из соответствующей группы используется для управления обменом с периферийным устройством (ПУ). Код режима “10” или “11”.

При сбросе бита D7 управляющее слово носит название BSR (Bit Set/Reset) и применяется для независимой установки (сброса) разрядов выходного порта С при его использовании для управления ПУ. Формат слова BSR представлен на рис. 3.

Порты А, В и С для работы в том или ином режиме программируются независимо друг от друга. Так, если порт В запрограммирован для ввода данных в режиме 1, то порт А может выполнять любую другую операцию обмена из числа возможных. Свободная от управления часть порта С может программироваться для ввода или вывода в режиме 0, причем младшая половина порта независимо от старшей.

Применение ППА ВВ55А позволяет организовать связь микроЭВМ с приемниками и передатчиками цифровой информации в параллельном коде, а также строить схемы адаптеров радиального параллельного интерфейса (ИРПР).

При работе портов в режимах 1 и 2 управление обменом информацией с ПУ осуществляется дополнительными сигналами управления. Для генерации сигналов управления используются отдельные линии порта С. При разработке следует учитывать, что усложнённый алгоритм управления вводом-выводом, свойственный этим режимам, затрудняет построение аппаратной части периферийного устройства и программную реализацию.

В режиме 0 осуществляется прямой однонаправленный ввод/вывод через любой из трёх портов без каких-либо сигналов сопровождения. В этом режиме интерфейс можно представить как набор параллельных линий ВВ, организованных в две байтовые и две 4-разрядные шины, причём каждая может быть применена либо для ввода, либо для вывода независимо от других.

Входная информация адаптером не запоминается и читается при низком уровне напряжения сигнала на входе RD. Выходная информация защелкивается в выходной буферный регистр выбранного порта по срезу сигнала WR и остаётся на выходе порта до нового цикла вывода или изменения режима.

Описание лабораторного стенда

Основой стенда является БИС КР580ВВ55А. Вывод информации осуществляется через порт А. Для индикации информации, выводимой в порт А, используются светодиоды HL1-HL8. Для имитации сигналов от периферийных устройств служат кнопки К1 и К2 (биты С7 и С6 соответственно).

После включения питания или после нажатия на кнопку «СБРОС» УМК все порты ППА переходят в высокоомное состояние и светодиоды HL1-HL8 загораются.

Адресация микросхемы производится при выполнении команд IN port и OUT port. При этом по шине управления на соответствующие выводы микросхемы подаются сигналы RD и WR. В качестве дешифратора адреса используется дешифратор двоичного кода в позиционный (К555ИД7). Подключая его входы к различным разрядам шины адреса, можно варьировать распределение регистров ППИ в пространстве ввода-вывода МП К580ВМ80. Для задания базового адреса микросхемы необходимо установить перемычку между соответствующим выводом дешифратора и выводом 6 БИС К580ВВ55. Соответствие между выходами дешифратора и базовым адресом ППА приведено в табл. 2.

Таблица 2

Соединение		Базовый адрес КР580ВВ55А
Номер вывода К555ИД7	Номер вывода КР580ВВ55А	
15	6	80Н
14	6	84Н
13	6	88Н
12	6	8СН
11	6	90Н
10	6	94Н
9	6	98Н
7	6	9СН

Для адресации внутренних регистров К580ВВ55 используются два младших разряда шины адреса. Например, если выбрать в качестве базового адрес 80, то при выполнении команд IN 80 , IN 81 , IN 82 или IN 83, на выводе 15 дешифратора вырабатывается логический «0», разрешающий работу ППА.

Практическая часть

1. Вывести в порт двоичное число, используя базовый адрес микросхемы 90. Произвести проверку правильности выдачи по светодиодному индикатору.

2. Используя порты А и С, написать программу вывода данных в циклическом режиме. Тип выдаваемых данных, скорость выдачи и базовый адрес микросхемы по вариантам указаны в табл. 3.

Таблица 3

Вариант	Задержка	Эффект	Базовый адрес	Запуск	Останов
1	0.1 с	Бегущий огонь	80Н	К1	К2
2	0.5 с	Бегущая тень	84Н	К2	К1
3	1 с	Бегущая цепь	88Н	К1	К2
4	0,2	Бегущая тень	8СН	К2	К1
5	1,5	Бегущий огонь	90Н	К1	К2
6	0,5 с	Бегущая цепь	94Н	К2	К1

Практическая работа № 14. 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».

Краткие 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

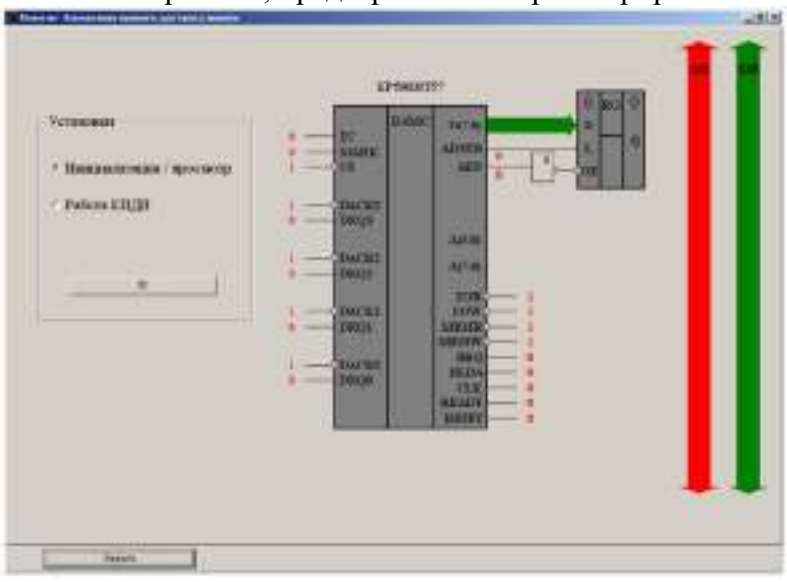
Порядок выполнения работы

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

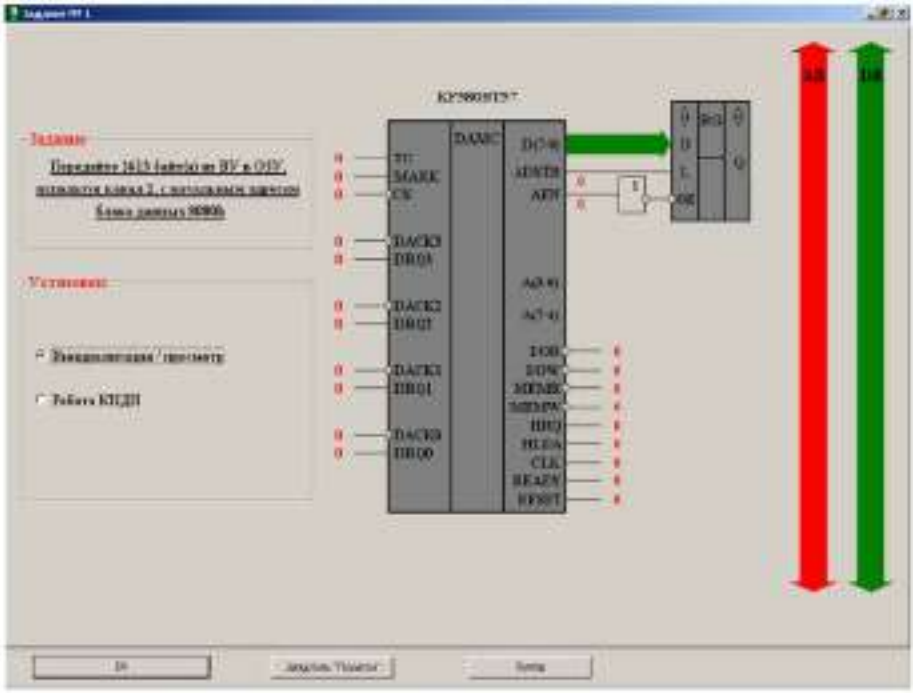
Инициализация осуществляется загрузкой исходных данных (начальные адреса областей ОЗУ, размер блоков данных и код ы режимов) в адресуемые регистры и регистр режима контроллера. Содержимое любого регистра и счетчика, кроме регистра режима, можно считать.

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 №15. Интерфейсы микропроцессорных систем.

Цель работы: изучить режимы работы программируемых интерфейсных компонентов, изучить построение параллельных и последовательных портов ввода-вывода МПС на примере микросхем программируемого параллельного интерфейса КР580ВВ55 (Intel 8255) и универсального синхронно-асинхронного приемо-передатчика КР580ВВ51 (Intel 8251).

Теоретические сведения.

Взаимодействие МП с памятью и УВВ требует выбора способа обращения к устройствам памяти и ввода – вывода, разработки системы адресации и внутреннего интерфейса микропроцессорной системы.

В архитектуре магистрального типа важное значение приобретает интерфейс. На рис. 1 показаны интерфейсы МП, системы памяти и системы ввода – вывода (ВВ). В узком смысле интерфейсом (от англ. interfase – сопрягать, согласовать) называют устройство сопряжения; в широком смысле под интерфейсом понимают совокупность аппаратных, программных и конструктивных средств, обеспечивающих взаимодействие функциональных модулей системы.

Таким образом, для представленной на рис. 1 микропроцессорной системы необходимым условием высокой эффективности использования является совместимость интерфейсов МП, системы памяти и системы ВВ.

Работа рассматриваемой системы синхронизируется генератором тактовых импульсов (ГТИ).

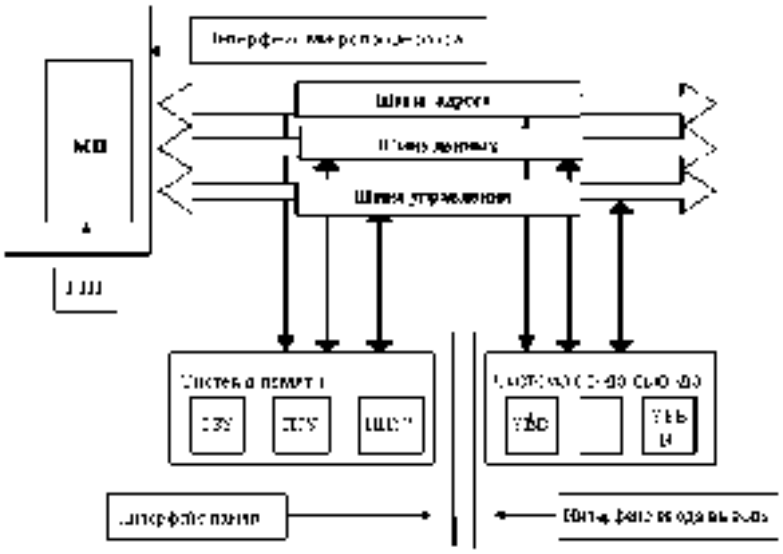


Рис. 1. Обобщенная структура МПС

Наиболее важными факторами, характеризующими интерфейс, являются:

• простота связи,

• скорость,

• адаптивность обмена.

Опыт свидетельствует, что, создавая интерфейс, трудно одновременно удовлетворить всем этим требованиям. Так, программно управляемый интерфейс наиболее прост в аппаратном смысле и обеспечивает высокую адаптивность обмена, однако при этом неизбежны небольшая скорость и потери машинного времени.

Интерфейсы по каналам прямого доступа и с прерываниями значительно повышают скорость обмена, однако возрастает сложность интерфейса как в аппаратном, так и в программном отношении, и, как следствие, снижается возможность оперативной модификации обмена.

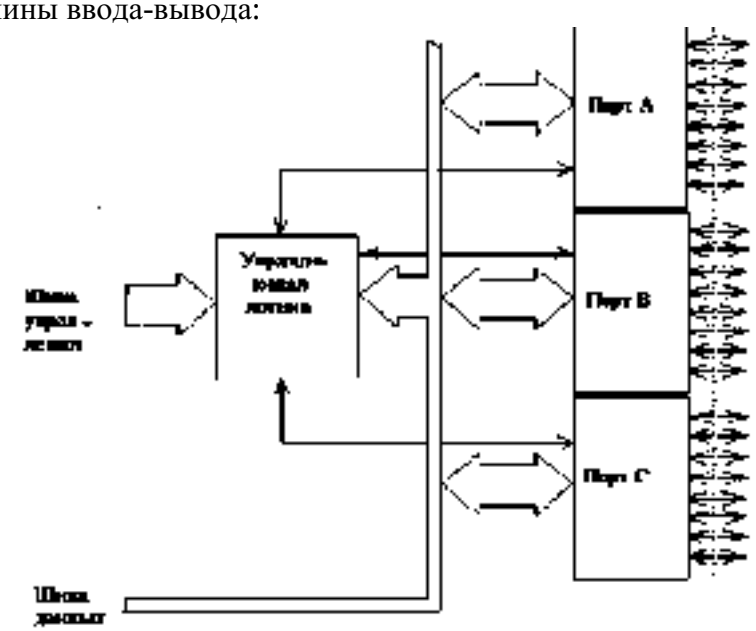
В настоящее время наибольшее распространение получили микро-ЭВМ с магистральным интерфейсом (рис. 1). Линии связи между процессором, памятью и внешними устройствами разделяют на группы (шины): адреса, данных и управления.

Электронная схема, входящая в состав ВУ и обеспечивающая подключение к интерфейсу, называется контроллером. Нередко конструктивно интерфейс оформлен в виде отдельной платы, которая вставляется в электронный блок микро-ЭВМ и кабелем соединяется с внешним устройством. Через контроллер процессор обменивается информацией с ВУ, а также управляет его работой (перемоткой ленты, пуском, остановкой) и опрашивает его состояние (готовность, наличие ошибок).

Обычно в контроллере для этих целей имеются два отдельных регистра: регистр данных (РД) и регистр состояния (РС). Каждый разряд регистра состояния имеет свое назначение, определяемое форматом регистра состояния. Например, отдельный бит может использоваться для проверки готовности ВУ: если в нем при чтении РС процессор обнаружил “1”, значит, ВУ готово к вводу-выводу , если же в этом разряде “0” – значит, ВУ не готово (например, выполняет предыдущую операцию ввода-вывода). Так как в системе может быть несколько различных контроллеров, то каждому из них присваивается свой адрес. Если в контроллере имеются РД и РС, то каждому из этих регистров присваиваются отдельные адреса. В начале каждой операции ввод –вывода процессор формирует адрес требуемого ВУ, контроллеры анализируют этот адрес,

сравнивая его с собственным адресом. Контроллер, принявший свой адрес, участвует в операции ввода-вывода, остальные контроллеры не мешают ее выполнению.

С использованием разнообразных БИС (кристаллов ввода-вывода) упрощается сопряжение, уменьшается количество компонентов, уменьшается стоимость и увеличивается эффективность системы. Самым простым из таких кристаллов является программируемый параллельный интерфейс (ППИ БИС КР580 ВВ55,Intel8255). Он представляет собой выводную ИС, содержащую 3 шины ввода-вывода:



Каждая из шин используется как входная или выходная шина. Направление каждой шины управляется регистром на кристалле. Программа инициализации, находящаяся в ПЗУ системы, устанавливает управляющий регистр в нужную комбинацию входных и выходных шин. Такие БИС являются очень гибкими, так как каждая шина может изменяться программой. Кроме того, они обычно включают управляющую логику для синхронизации работы.

Другой общий тип интерфейсных кристаллов, обеспечивающих последовательные входы и выходы (рис. 2), – универсальные асинхронные передатчики и приемники. Они принимают байт данных с МП, а затем выдают его по одному биту, т. е. действуют подобно регистру параллельно-последовательного ввода-вывода. Кроме того, могут автоматически вводиться биты старта, остановка и другие синхронизирующие и управляющие сигналы.

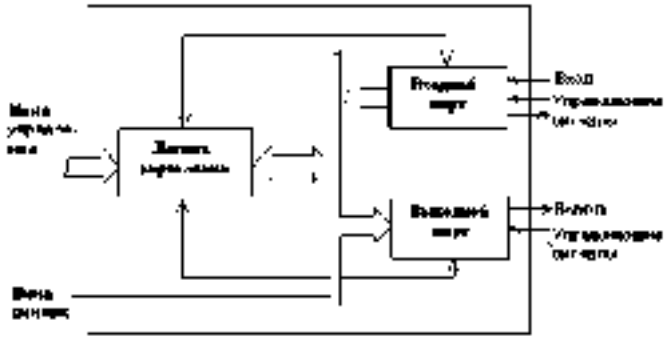


Рис. 2. Интерфейс, обеспечивающий последовательную связь между двумя системами.

Формат управляется управляющим регистром, подобным рассмотренному для ППИ.

Рассматриваемые интерфейсы также могут обрабатывать данные в другом направлении, преобразовывая поток последовательных битов в параллельную форму, подходящую для непосредственного использования МП. Обычно для связи между микропроцессорной системой и периферийным устройством, таким как ЭЛТ или телетайп, используется последовательный ввод-вывод. Благодаря тому, что информация имеет последовательный формат, для соединения устройств необходимо только 2 провода.

Имеются также разнообразные специализированные интерфейсные кристаллы, включающие контроллеры гибких дисков, контроллеры ЭЛТ, контроллеры ЗУ с прямым доступом, контроллеры клавиатур и дисплея.

Структура и типы интерфейсов контрольно-измерительной системы на базе микроконтроллера и ПЭВМ

Структура контрольно-измерительной системы в общем случае приведена на рис.3. Она состоит из измерительного датчика, микроконтроллера, осуществляющего управление всей системой, персональной ЭВМ, осуществляющей обработку полученной информации.

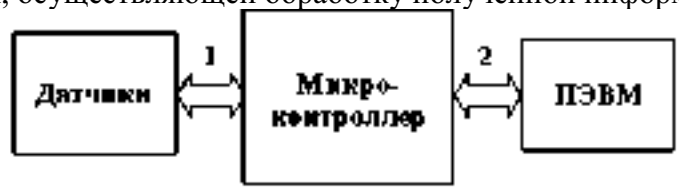


Рис. 3. Структура контрольно-измерительной системы.

В частном случае, когда микроконтроллер в состоянии сам осуществить обработку получаемой от датчика информации и отобразить результаты, структурная схема несколько видоизменяется (рис. 4).

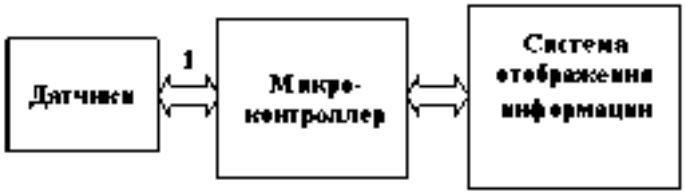


Рис. 4. Структура автономной контрольно-измерительной системы.

Система индикации может представлять собой жидкокристаллический индикатор либо любой другой, предназначенный для использования в микропроцессорной системе.

Цифрой 1 обозначен интерфейс, посредством которого происходит обмен информацией между датчиками и микроконтроллером. Требования к этому интерфейсу следующие: скорость передачи информации должна соответствовать скорости работы датчиков.

Цифрой 2 обозначен интерфейс, позволяющий связать микроконтроллер с персональным компьютером. Требования к этому интерфейсу: скорость передачи информации, достаточная для связи компьютера с выбранным микроконтроллером, а также простота реализации процесса передачи информации.

Параллельный и последовательный порты ПЭВМ (интерфейсы Centronics и RS-232).

Параллельный, последовательный – это наиболее распространенные порты ввода/вывода. Изначально каждый из этих портов разрабатывался для определенного применения. Параллельные предназначались для соединения компьютеров с принтерами, последовательные – для подключения принтеров, модемов и мыши. Однако они могут использоваться и для других приложений, связанных с сопряжением компьютера с внешними устройствами. Периферийные устройства, созданные для этих портов, легко подключаются к IBM PC-совместимому компьютеру. Принципиальные схемы отличаются мобильностью и могут применяться для решения проблем сопряжения с любым оборудованием, которое оснащено указанными портами. Таким образом, полезно узнать, как они работают и как обеспечивается наиболее эффективное их использование.

Параллельный интерфейс Centronics

Порт “Centronic”, или параллельный, – это промышленный стандарт для подсоединения принтеров к компьютеру. Компьютер имеет по крайней мере один такой порт, встроенный в материнскую плату или представляющий собой отдельную интерфейсную карту ввода/вывода. Увеличить количество параллельных портов просто и недорого, можно установить параллельных

порта с логическими именами от LPT1 до LPT3. Разъемы порта для компьютера и принтера отличаются друг от друга. Первый – это 25-контактная розетка D-типа, а второй – 36-контактная розетка параллельного типа. Для соединения компьютера с принтером используется принтерный кабель длиной не более 5 м. Персональный компьютер работает максимум с тремя параллельными портами, которые в MS-DOS имеют логические имена: LPT1, LPT2, LPT3. В адресном пространстве компьютера резервируются базовые адреса этих портов: 3BCh, 378h, 278h.

Первый адрес обычно используется, если принтерный порт находится на плате графического адаптера Hercules или EGA. На материнской плате (motherboad) адрес LPT1 – 378h, а LPT2 – 278h. Для принтерного порта LPT1 предусмотрено аппаратное прерывание IRQ7, а для LPT2 – IRQ5, хотя на практике они используются очень редко.

Начиная с базового адреса, каждый адаптер принтера имеет в адресном пространстве три адреса. При этом первый адрес соответствует регистру данных, посылаемых от компьютера к принтеру. В случае использования TTL-микросхем этот регистр бывает реализован на микросхеме 74LS374. Чтение установленных битов данных можно осуществить по тому же адресу. Физически чтение данных происходит через буфер данных, выполненный, например, на микросхеме 74LS244.

Следующий адрес (базовый адрес плюс единица) позволяет читать регистр статуса адаптера (расположенный, конечно, в принтере) через буферную микросхему (часто 74LS240 или 367–368). В регистре статуса биты с 3 по 7 позволяют определить состояние некоторых сигналов Centronics:

bit 3 = 0:Error,
bit 4 = 1:Select,
bit 5 = 1:Paper Out,
bit 6 = 0:Acknoweledge,
bit 7 = 0:Busy.

Чтение регистра статуса имеет смысл при передаче данных на принтер для определения состояния принтера и процесса передачи данных.

Адрес третьего порта (базовый адрес+2) соответствует регистру управления интерфейса. Этот регистр (read only – только для чтения), для которого может использоваться микросхема 74LS174, позволяет определить следующие состояния принтера:

bit 0 = 0: сигнал Data Strobe активен,
bit 1 = 0: сигнал Auto Line Feed включен,
bit 2 = 0: INIT инициализация принтера,
bit 3 = 1: Select Input принтер выбран,
bit 4 = 1: IRQEN прерывание разрешено.

Таким образом, порт состоит из двух регистров на запись и трех на чтение (регистр данных, статуса и управления).

Некоторые из линий регистров аппаратно инвертируются, так что логический сигнал на разъеме противоположен программно-установленному. На рис.10 места инверсии обозначены кружками.

Таким образом, на разъем порта выведены 8 выходных линий данных, 5 входных регистра статуса и 4 выходных линии регистра управления. Сигнал IRQEN, не выведенный на разъем, разрешает выработку прерываний при поступлении сигнала /ACK.

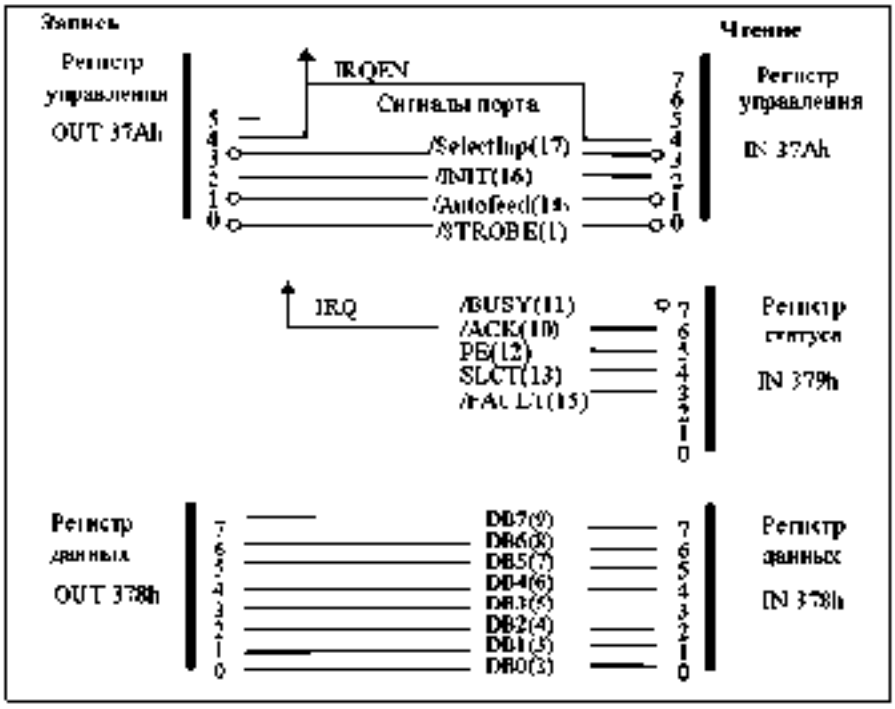
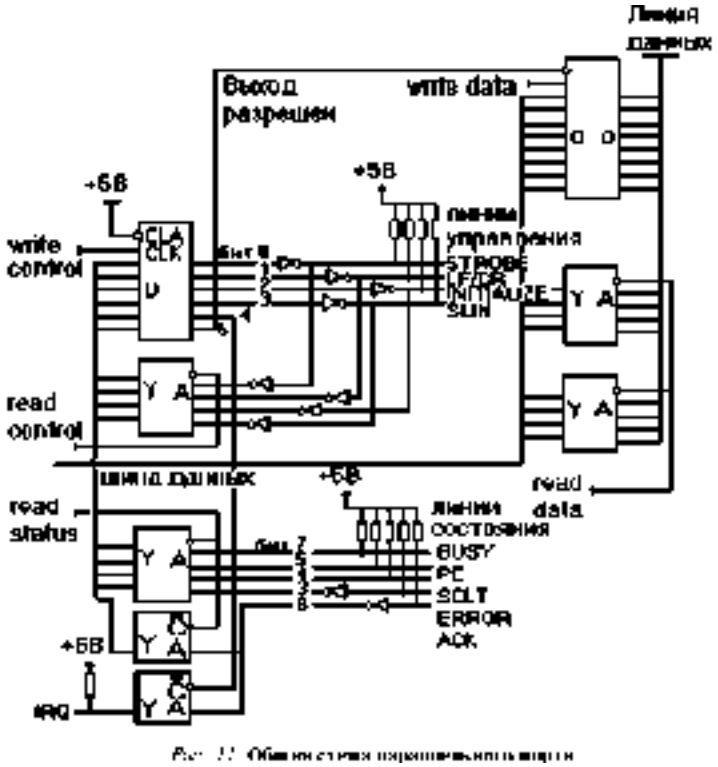


Рис.10. Сигнал интерфейса Centronics

Общая схема параллельного порта внутри ПК представлена на рис. 11. Восьмибитовые данные заносятся в DD1 во время записи в регистр с адресом “базовый адрес” + 0. Операция осуществляется командой WRITE_DATA. Эти данные образуют группу. Они считываются компьютером из того же регистра, через DD2 с помощью команды READ_DATA. Во время чтения выход DD1 должен иметь высокий уровень сопротивления, что достигается подачей на контакт 1 (выход разрешен) DD1 высокого уровня напряжения. Шестибитовое управляющее слово записывается в DD3 через регистр с адресом “базовый адрес” + 2 при помощи команды WRITE_CONTROL. Биты с 0 по 3 подаются на выход разъема и образуют группу управления. Некоторые биты инвертируются микросхемами с открытыми коллекторами на выходе (DD6 и DD7). Все выходные линии подключены к питанию +5В через резисторы 4,7 кОм. Состояние этих линий считывается через регистр с адресом, “базовый адрес” + 2 через DD4 посредством команды READ_CONTROL. Четвертый бит управляющего байта разрешает прерывание, а пятый бит открывает или закрывает выход DD1. Состояние пяти контактов разъема порта (группа состояния) компьютер считывает через DD4 с помощью команды READ_STATUS через регистр с адресом “базовый адрес” + 1. Входы линии подключены к питанию +5В через резисторы 4,7 кОм, два входа инвертируются. В первых конструкциях IBM PC контакт “выход” разрешен DD1 соединялся с “землей” для постоянного открывания выходов. Это была однонаправленная версия параллельного порта. Начиная с IBM PS/2, указанный контакт соединили с пятым битом регистра управления DD3, и порт стал двунаправленным. Следует отметить, что параллельные порты, поставляемые со встроенными картами ввода/вывода, двунаправленные. Для любого контакта следует избегать короткого замыкания и/или соединения с шиной питания.



Скорость передачи данных через параллельный порт превышает 1 Мб/с. На рис. 12 представлена логическая структура параллельного порта.

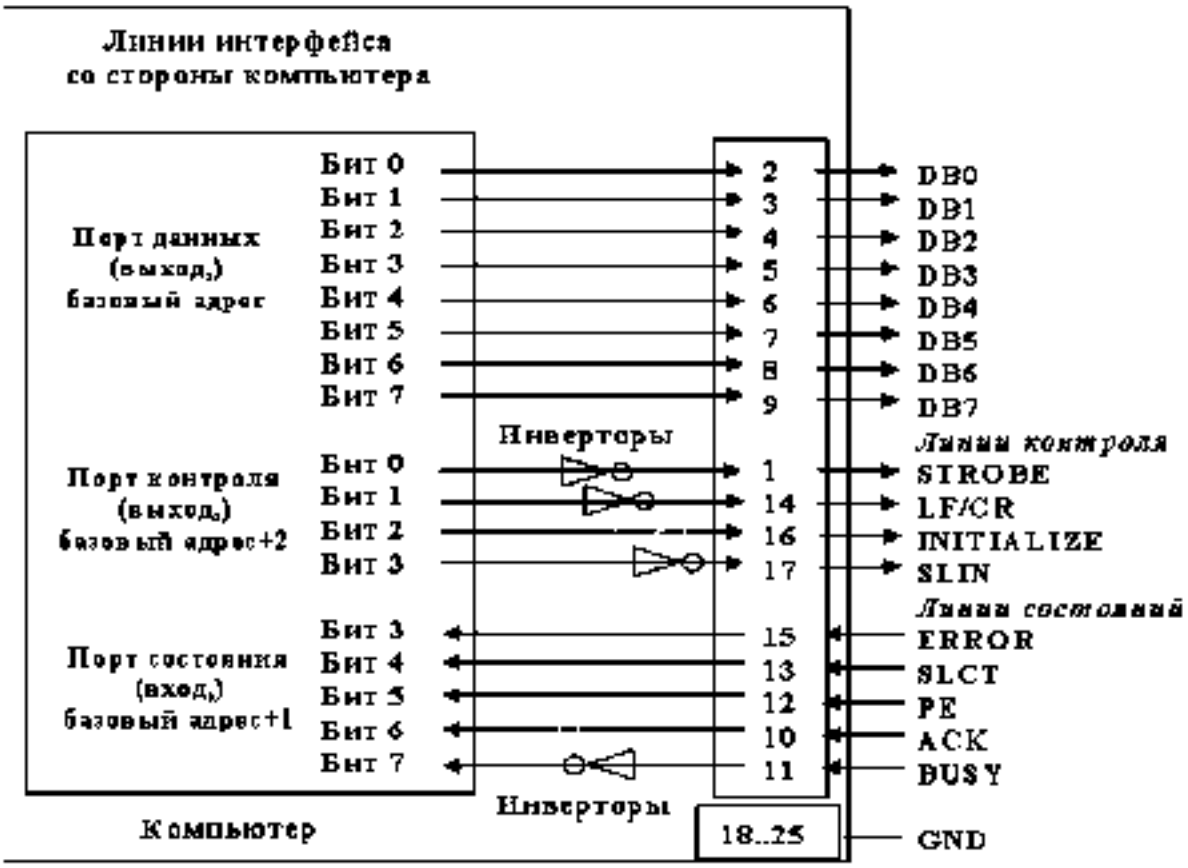


Рис. 12. Логическая структура параллельного порта

Рассмотрим кратко расширенные режимы работы параллельного порта (табл. 1).

Таблица 1.

Расширенные режимы работы параллельного порта

Совместимый протокол		Нибловый протокол		Байтовый протокол		Протокол EPP		Протокол ECP	
Сигнал	*Направление*	*Сигнал*	*Направление*	*Сигнал*	*Направление*	*Сигнал*	*Направление*	*Сигнал*	*Направление*
/STROBE	0	-	-	HostClk	0	/Write	0	HostClk	0
BUSY	1	PtrBusy	1	PtrBusy	1	/Wait	1	Periph-Ack	1
/ACK	1	PtrClk	1	PtrClk	1	Intr	1	Periph-Clk	1
SELECT	1	Xflag	1	Xflag	1	Xflag	1	Xflag	1
PERROR	1	AckDataReq	1	AckDataReq	1	AckDataReq	1	/Ack-Reverse	1
/FAULT	1	/Data-Avail	1	/Data-Avail	1	/Data-Avail	1	/Periph Request	1
/INIT	0	-	-	-	-	/Init	0	/Reverse Request	0
/AUTOFD	0	Host-Busy	0	Host Busy	0	/DStrb	0	Host-Ack	0
	0	-	-	Data 1...8	1/0	Data 1...8	1/0	Data 1...8	1/0
/SELECT-IN	0	-	-		-	/AStrb	0	ECP-Mode	0

Совместимый протокол моделирует простейший принтерный порт. Компьютер защелкивает данные на выходах PD0...PD7 сигналом /STROBE. Принтер выдает сигнал /ASK в подтверждение приема данных и готовности к приему новых данных. Принтер выдает сигнал BUSY, если он не готов к приему данных, а также сигналы PERROR (Paper Error) и /FAULT при возникновении ошибок. Компьютер выдает сигнал /INIT для аппаратного сброса периферийного устройства. Для выбора внешнего устройства компьютер выдает сигнал /SELECTIN. Принтер выдает сигнал SELECT для указания, что он включен. Сигнал AUTOFD (Auto Feed Extension) выдается компьютером для автоматического перевода строки в принтере на каждый символ возврата каретки.

Нибловый протокол служит для ввода в компьютер битов 0...3 и 4...7 по очереди. Внешнее устройство сопровождает каждую порцию данных сигналом PtrClk. Этот сигнал вызывает прерывание в компьютере, если это разрешено. В ответ компьютер сбрасывает сигнал HostBusy, который затем устанавливается в подтверждение приема данных. Биты 0 и 4 передаются по линии DataAvail, 1 и 5 – по Xflag, 2 и 6 – по AckDataReq, 3 и 7 – по PtrBusy. Байтовый протокол описывает двунаправленную передачу по линиям данных. Внешнее устройство выдает сигнал DataAvail, указывающий на наличие данных к передаче, и сигнал PtrClk вместе с каждым байтом данных. Сигнал HostClk сбрасывается компьютером в подтверждение приема данных. Заметьте, что периферийное устройство не должно интерпретировать этот сигнал как строб данных, передаваемых от компьютера. Xflag выдается внешним устройством и указывает, что оно включено (on-line). Сигналы HostBusy и PtrBusy указывают на занятость компьютера и принтера соответственно при приеме данных. Сигнал AckDataReq сбрасывается внешним устройством в подтверждение HostBusy.

Протокол EPP(Enhanced Parallel Port) описывает двунаправленную передачу через порт адресов и данных. Сигнал /Write выдается компьютером и устанавливает направление передачи. Сигнал /Wait выдается внешним устройством, когда оно не готово. Когда этот сигнал сброшен в “0”, порт снимает сигнал IOCHRDY на шине ISA, чтобы удлинить цикл чтения-записи. По окончании передачи адреса или данных сигнал /Wait устанавливается в “1”. Передача данных стробируется сигналом /DStrb, а адреса – /Astrb. Сигнал Intr выдается внешним устройством для выработки прерывания в компьютере.

Назначение остальных сигналов аналогично совместимому протоколу.

Протокол ECP (Extended Capabilities Port) обеспечивает двунаправленную передачу адресов, данных или RLE (Run Length Encoded) информации. Направление передачи задается компьютером с помощью сигнала /ReverseRequest (если неактивен – от компьютера к внешнему устройству, т.е. в прямом направлении). Внешнее устройство выдает /PeriphRequest для запроса на передачу в обратном направлении. В ответ на /ReverseRequest оно выдает сигнал /AckReverse.

При передаче в прямом направлении сигнал HostClk стробирует данные. Этот сигнал снимается при обнаружении сигнала PeriphAck от внешнего устройства и не может быть выдан вновь, пока PeriphAck не будет снят. При передаче в обратном направлении сигнал HostClk не используется, а сигнал PeriphAck обычно сброшен в“0” (“1” указывает на передачу RLE данных).

Передача в обратном направлении стробируется сигналом PeriphClk и подтверждается сигналом HostAck аналогичным образом.

При передаче в прямом направлении сигнал HostAck указывает на передачу адреса или RLE (“1”) либо данных (“0”).

При работе в режиме ECP сигнал ECP Mode установлен в “1”.

Отметим, что конкретная плата портов в вашем компьютере может поддерживать далеко не все из указанных режимов. Тем не менее зачастую их удается реализовать программным способом хотя бы частично.

Интерфейс RS-232C

Широко используемый последовательный интерфейс синхронной и асинхронной передачи данных определяется стандартом EIA RS-232-C и рекомендациями V.24 CCITT. Он изначально создавался для связи компьютера с терминалом и в настоящее время используется в самых различных применениях. Параметры интерфейса приведены в табл. 2.

Таблица 2

Стандарт	EIA RS-232-C, CCITT V.24
Скорость передачи	115 Кбит/с (максимум)
Расстояние передачи	15 м (максимум)
Характер сигнала	Несимметричный по напряжению
Количество драйверов	1
Количество приемников	1
Схема соединения	полный дуплекс, от точки к точке

Интерфейс RS-232-C соединяет два устройства. Линия передачи первого устройства соединяется с линией приема второго, и наоборот (полный дуплекс). Для управления соединенными устройствами используется программное подтверждение (введение в поток передаваемых данных соответствующих управляющих символов). Возможна организация аппаратного подтверждения путем организации дополнительных RS-232 линий для обеспечения функций определения статуса и управления. Описание выводов интерфейса Centronics приведено в табл. 3.

Сигналы интерфейса RS-232C подразделяются на следующие классы:

Последовательные данные: – например, TXD, RXD.

Интерфейс RS-232C обеспечивает два независимых последовательных канала данных: первичный (главный) и вторичный (вспомогательный). Оба канала могут работать в дуплексном режиме.

Управляющие сигналы квитирования: – например, RTS, CTS.

Сигналы квитирования – это средство, с помощью которого обмен сигналами позволяет DTE начать диалог с DCE до фактических передачи или приема данных по последовательной линии связи.

Сигналы синхронизации: – например, TC, RC.

В синхронном режиме (в отличие от более распространенного асинхронного) между устройствами необходимо передавать сигналы синхронизации, которые упрощают контроль целостности сигнала в целях его декодирования.

Таблица 3

Описание выводов интерфейса Centronics

Наименование	Направление	Описание	Контакт (25-контактный разъем)	Контакт (9-контактный разъем)
DCD	IN	Carrie Detect (Определение несущей)	8	1
RxD	IN	Receive Data (Принимаемые данные)	3	2
TxD	OUT	Transmit Data (Передаваемые данные)	2	3
DTR	OUT	Data Terminal Ready (Готовность терминала)	20	4
GND	-	System Ground (Корпус системы)	7	5
DSR	IN	Data Set Ready (Готовность данных)	6	6
RTS	OUT	Request to Send (Запрос на отправку)	4	7
CTS	IN	Clear to Send (Готовность приема)	5	8
RI	IN	Ring Indicator (Индикатор)	22	9

Интерфейс RS-232C предназначен для подключения к компьютеру внешних устройств (принтера, сканера, модема, мыши и др.), а также для связи компьютеров между собой.

Основными преимуществами использования RS-232C по сравнению с Centronics являются возможность передачи на значительно большие расстояния и гораздо более простой соединительный кабель. В то же время работать с ним несколько сложнее. Данные в RS-232C передаются в последовательном коде побайтно. Каждый байт обрамляется стартовым и стоповыми битами. Данные могут передаваться как в одну, так и в другую сторону (дуплексный режим). Компьютер имеет 25-контактный (DB25P) или 9-контактный (DB9P) разъем для подключения RS-232C. Назначение контактов приведено в табл. 3. Наиболее часто используется трех- или четырехпроводная связь (для двунаправленной передачи). Схема соединения для четырехпроводной линии связи показана на рис. 13.

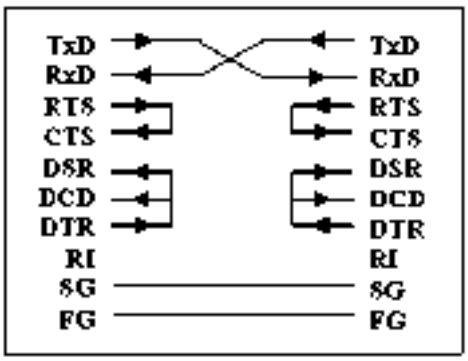


Рис. 13. Схема 4-проводной линии связи для RS-232C

Для двухпроводной линии связи в случае только передачи из компьютера во внешнее устройство используются сигналы SG и TxD. Все 10 сигналов интерфейса задействуются только при соединении компьютера с модемом. Формат передаваемых данных показан на рис. 14.

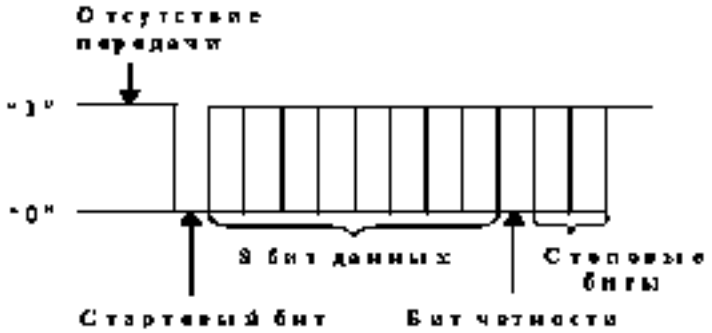


Рис. 14. Формат данных RS-232C

Собственно данные (5, 6, 7 или 8 бит) сопровождаются стартовым битом, битом четности и одним или двумя стоповыми битами. Получив стартовый бит, приемник выбирает из линии биты

данных через определенные интервалы времени. Очень важно, чтобы тактовые частоты приемника и передатчика были одинаковыми (допустимое расхождение – не более 10 %).

Скорость передачи по RS-232C может выбираться из ряда: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 бит/с.

Все сигналы RS-232C передаются специально выбранными уровнями, обеспечивающими высокую помехоустойчивость связи (рис. 15.).

Отметим, что данные передаются в инверсном коде (логической единице соответствует низкий уровень, логическому нулю – высокий уровень).

Рис. 15. Уровни сигналов RS-232C на передающем и принимающем концах линии связи

Для подключения произвольного УС к компьютеру через RS-232C обычно используют трех- или четырехпроводную линию связи (см. рис. 1.1), но можно задействовать и другие сигналы интерфейса. Обмен по RS-232C осуществляется с помощью обращений по специально выделенным для этого портам COM1 (адреса 3F8h...3FFh, прерывание IRQ4), COM2 (адреса 2F8h...2FFh, прерывание IRQ3), COM3 (адреса 3F8h...3EFh, прерывание IRQ10), COM4 (адреса 2E8h...2EFh, прерывание IRQ11).

Интерфейс RS-232C/CCITT V24, определенный стандартом Ассоциации электронной промышленности (EIA), подразумевает наличие оборудования двух видов: терминального DTE и связного DCE.

Терминальное оборудование, например компьютеры, может посылать и (или) принимать данные по последовательному интерфейсу. Оно как бы оканчивает (terminate) последовательную линию.

Связное же оборудование понимается как устройства, которые могут упростить последовательную передачу данных совместно с терминальным оборудованием (например, модем).

Различие между терминальным и связным оборудованием довольно расплывчато. Произведя незначительные изменения в линиях интерфейса RS-232, можно заставить связное оборудование функционировать как терминальное.

DTE – Data Terminal Equipment (COM-порт, принтер, плоттер) – разъем “папа” (male).

DCE - Data Communication Equipment (модемы) – разъем “мама” (female).

Общее правило соединения (бывают исключения):

Кабель DTE-DCE: “мама” – “папа”, соединяются одноименные цепи. Переходники 9-25 и 25-9 – аналогично.

Кабель DTE_DTE (Zero-Modem Cable) – “нуль-модем”: соединяются земли, RD и TD перекрестно, остальные сигналы соединяются по одной из вышеприведенных схем (рис. 2.5).

Наборы передаваемых сигналов могут сокращаться в зависимости от протокола квитирования (Flow Control). Входы портов буферизованы триггерами Шмитта 1489 (2 шт.) с порогами переключения – 3В и +3В, выходы – передатчиками 1488 с выходными уровнями –12...-5В и +5...+12В, током короткого замыкания до 20 мА.

На современных системных платах порт COM2 может конфигурироваться на использование IR (Infra Red) – беспроводной инфракрасной связи на скорости 115 Кбит/с в стандарте Hewlett Packard IR или Amplitude Shifted Keyed IR. IR-интерфейс имеют блокнотные

ПК, некоторые принтеры. Внешний приемопередатчик – “красный глаз” – устанавливается на лицевой панели компьютера и подключается к разъему IR-Connector системной платы.

Программирование последовательного порта компьютера

Контроллер последовательного интерфейса обеспечивает асинхронный обмен по стандарту RS-232C.

Порты COM1 - COM4 поддерживаются BIOS INT 14h. Функции прерывания INT 14h:

00h – инициализация (110-9600 бит/с, 5, 7 или 8 бит данных, 1-2 стоп-бита, паритет);

01h – вывод символа (без прерываний);

02h – ввод символа (без прерываний);

03h – опрос состояния модема и линии.

Используемые ячейки BIOS DATA AREA:

0:0400, 0402, 0404, 0406 – адреса портовCOM1-COM4;

0:047C, 047D, 047E, 047F – тайм-аут COM1-COM4.

Стандартные базовые адреса и прерывания:

COM1: 3F8h-IRQ4;

COM2: 2F8h-IRQ3;

COM3: AT-3E8h, 3E0h, 338h-IRQ4;

PS/2-3220h-IRQ3;

COM4: AT-2E8h, 2E0h, 238h-IRQ3;

PS/2-3228h-IRQ3;

COM5-COM8: PS/2-4220h, 4228h, 5220h, 5228h -IRQ3;

8250/16450/16550 UART

COM-порты XT/AT базируются на микросхеме UART (Universal Asynchronous Receiver-Transmitter – универсальный асинхронный приемопередатчик), совместимой с i8250-8250A/16450/16550.

Основные отличительные особенности членов семейства8250:

•8250 имеет ошибки, учтенные в XT BIOS;

•8250A – ошибки исправлены, потеряна совместимость с BIOS. Работает в некоторых моделях AT, но не на скорости 9600 бит/с.

•8250B – исправлены ошибки 8250 и 8250A; восстановлена ошибка в прерываниях – совместима с XT BIOS. В AT работает под DOS, кроме скорости 9600 бит/с.

•16450 – высокоскоростная версия 8250 для AT. Ошибок 8250 и полной совместимости с XT BIOS не имеет. Минимум, требуемый для OS/2.

•16550A – имеет работающие 16-байтные FIFO-буферы приема и передачи и возможность использования DMA. Должен применяться в AT при интенсивных обменах на скоростях 9600 бит/с и выше без потери данных.

Назначение регистров 8250:

0R/W - DATA – регистр данных (DLAB=0).

0W – DLL – младший байт делителя (DLAB=0).

1W – DL H – старший байт делителя (DLAB=1), делитель 115200/V, где V – скорость бит/с.

1W – IER – регистр разрешения прерываний (1 = разрешить прерывание):

Биты 7..4 0. Бит 3 – Mod IE – по изменению состояния модема (любой из линий CTS, DSR, RI, DCD).

Бит 2 – RxL IE – по обрыву/ошибке линии.

Бит 1 – TxD IE – по завершению передачи.

Бит 0 – RxD IE – по приему символа.

2R – IIR – регистр идентификации прерывания:

Биты 2..1 – причина прерывания:

11 = ошибка/обрыв линии; сброс – чтением регистра состояния линии.

10 = принят символ; сброс чтением данных.

01 = передан символ; сброс записью данных.

00 = изменение состояния модема; сброс – чтением регистра состояния модема.

Бит 0 – Interrupt Pending – 1 = нет запроса прерывания.

3 R/W – LCR – регистр управления линией:
Бит 7 – DLAB – доступ к делителю (регистрам #0,1).
Бит 6: 1= обрыв линии (посылка нулей).
Бит 5=0 – отмена постоянной четности.
Бит 4: 0 = нечетность, 1 = четность.
Бит 3: 1= контроль паритета разрешен.
Бит 2: 0=1, 1=2 (1.5 для 5-битного кода) стоп-бит.
Биты 1-0: 00=5, 01=6, 10=7, 11=8 бит/симв.
4W – MCR – регистр управления модемом:
Биты 7..5 = 8.
Бит 4 –Loopback – диагностика.
Бит 3 – OUT2 – выходной сигнал (используется для разрешения IRQ).
Бит 2 – OUT1 – выходной сигнал (свободен).
Бит 1 –RTS: 1=активен (-).
Бит 0 –DTR: 1=активен (-).
5R – LSR – регистр состояния линии:
Бит 7 = 0.
Бит 6 – TEMPT – регистр передатчика пуст, нет передаваемых данных.
Бит 5 – THRE – регистр передатчика пуст, готов принять байт для передачи.
Бит 4 – B1 – индикатор обрыва линии.
Бит 3 – FE – ошибка кадра (стоп-бит).
Бит 2 – PE – ошибка четности.
Бит 1 – OE – переполнение (потеря символа)
Бит 0 – RxRdy – данные приняты; сброс – чтением приемника.
6R – MSR – регистр состояния модема:
Бит 7 – состояние линии DCD.
Бит6 – состояние линииR1.
Бит 5 – состояние линии DSR.
Бит 4 – состояние линии CTS.
Бит 3 – DDCD – изменение состояния линии DCD.
Бит 2 – TER1 – изменение огибающей R1.
Бит 1 – DDSR – изменение состояния линии DSR.
Бит 0 – DCTS – изменение состояния линии CTS.
7 W – Scratch-Pad Register – в 8250 отсутствует.

Для программирования приемопередатчика компьютера необходимо знать его базовый адрес в адресном пространстве портов ввода-вывода компьютера. Для его определения следует обратиться к таблице базовых адресов COM-портов (COM – логическое имя последовательного интерфейса). Эта таблица находится по адресу[$40:0] в памяти компьютера. Таблица состоит из слов – адресов портов ввода-вывода. В компьютере может быть до четырех COM-портов, соответственно таблица состоит из восьми байт. Если порт отсутствует, в таблице стоит вместо адреса нуль.

Для адресации регистров управления COM-портов используются три адресные линии, но нужно адресовать десять регистров, поэтому бит D7 порта 3 (3D7) используется для переключения функций портов 0 и 1.

В табл. 4 приведен список регистров управления.

Таблица 4

Обозначение регистров последовательного порта

Ад-рес	3D7	Чтение/ Запись	Обозначение	Наименование регистра
0	0	Чтение	THR Transmitter Holding Register	Буфер приема
0	0	Запись	RBR Receiver Buffer Register	Буфер передачи
0	1	Чт/зп	DLL Divisor Latch LSB	Делитель частоты, младший байт
1	1	Чт/зп	DLM Divisor Latch MSB	Делитель частоты, старший байт
1	0	Чт/зп	IER Interrupt Enable Register	Регистр разрешения прерываний
2	0	Чт/зп	IIR Interrupt Identification Register	Идентификатор прерывания
3	0	Чт/зп	LCR Line Control Register	Управление линией
4	0	Чт/зп	MCR Modem Control Register	Управление модемом
5	0	Чт/зп	LSR Line Status Register	Состояние линии
6	0	Чт/зп	MSR Modem Status Register	Состояние модема

Для передачи байт-данных записывается в буфер передачи, прием обеспечивается чтением буфера приема. Также возможен вызов прерывания по приему или передаче данных, ошибке при обмене данными или изменению состояния модема (изменению состояния линий DCD,DSR,CTS,RI).

Регистр 1 (при 3D7=0) служит для разрешения/запрещения прерываний. Причину прерывания можно узнать, прочитав регистр 2 – идентификатор прерывания.

Регистр 3 служит для управления форматом передачи (кроме бита7).

Регистры 4 и 6 – регистры линий управления модемом (DCD,DTR,DSR,RTS,CTS).

Регистры 0 и 1 (при 3D7=1) управляют скоростью передачи данных.

Теперь рассмотрим последовательность этапов инициализации COM-порта на примере программы DEMO68HC.PAS (см. приложение 1):

1. Запись в регистр 3 управляющего байта с единицей в 7 разряде: {p-базовый адрес порта};

Port [P+3]:=$83.

2. Программирование делителя частоты:

{BR–Baud Rate – скорость передачи данных в бод, обычное значение – 9600 };

BT:=Round(115200.0/BR); {BT – коэффициент деления};

Port[P+0]:=Lo(BT);

Port[P+1]:=Hi(BT).

3. Запись в регистр 3 управляющего байта с нулем в 7 разряде:

Port[P+3]:=$03;{$03-8 бит данных ,1 стоповый бит, без бита четности}.

4. Запрещение прерываний:

Port[P+1]:=$00.

Для того чтобы данные передавались (принимались) правильно, при чтении или записи буфера данных (регистра 0) необходимо ждать приема данных или полной передачи предыдущего байта. Для этого используются биты 5 и 0 регистра 5 – регистра состояния линии:

1. Передача байта.

repeat until Port[P+5] and $20=$20; { бит 5 регистра 5 равен 1 – буфер передачи пуст}

Port[P+0]:=a; {передача байта данных в a}

2. Прием байта.

{Ticks:Longint absolute $40:$6C – системный таймер с приращением на единицу каждую 1/18,9 с. Используется для выхода из цикла ожидания через 5 с (при отсутствии получаемых данных).

ST := Ticks;

Repeat Until (Port[P+5] a

nd 1=1) or (Ticks-ST>94);

If Port[P+5] and 1=1 then a:=port[P];

{бит 1 регистра 5 равен 1 – получен байт данных a}.

Последовательный интерфейс и его стандарты RS (в сокращениях типа RS-232, RS-485, RS-422) – это всего-навсего Recommended Standard (рекомендованный стандарт). Слово “рекомендованный” означает, что эти стандарты никогда никем не были приняты (в противоположность таким стандартам, как IEEE-1284 или IEEE-1394), они были просто “рекомендованы”. Естественно, это позволяет производителям допускать часто определенные вольности (например, питание по 9-му пину в RS-232 вовсе не оговорено стандартом, однако широко используется).

Далее все RS-протоколы можно приблизительно разделить на полудуплексные (half-duplex) и дуплексные (full-duplex). Правда, деление такое не совсем точно, так как тот же RS-485 может быть и полудуплексным (два провода), и дуплексным (четыре провода), они так и называются: 2-wire (2-проводный) RS-485 и 4-wire (4-проводный) RS-485.

Есть еще такой вид протоколов, как симплексный (simplex), но ввиду ряда причин в компьютерной технике не применяется. Чем эти виды протоколов различаются? Симплексные протоколы позволяют передавать данные только в одну сторону, т.е. только с передатчика на приемник, но не обратно. Хороший пример симплексного протокола– FM радио или телевидение. Применяется он в тех случаях, когда надо просто передать информацию какому-либо устройству без необходимости подтверждения и обратной связи.

Полудуплексные протоколы снимают главное ограничение симплексных протоколов – одностороннюю связь. Они позволяют двум устройствам обмениваться информацией, причем оба устройства могут быть и приемниками, и передатчиками, но не одновременно, т.е. каждое устройство может либо передавать, либо принимать (классический / рекомендованный/ RS-485 именно полудуплексный).

Дуплексные протоколы наиболее совершенные. Применение дуплексного протокола позволяет вести и прием, и передачу информации одновременно, т.е. оба устройства могут быть и приемником, и передатчиком одновременно.

Вот сравнительная таблица для рекомендованных протоколов (надо сказать, что нынешние интерпретации протоколов, особенно RS-232, весьма далеки от рекомендованных):

Сравнительная характеристика последовательных интерфейсов

	RS-232	RS-422	RS-485
Соединения	Одиночный провод	Одиночный провод/много соединений допустимо	Много соединений допустимо
Количество устройств	1 передатчик 1 приемник	5 передатчиков 10 приемников на 1 передатчик	Много передатчиков и приемника
Вид протокола	Дуплексный	дуплексный	полудуплексный
Макс. длина провода	~15.25 м при 19.2Kbps	~1220 м при 100Kbps	~1220 м при 100Kbps
Макс. скорость передачи	19.2Kbps для 15 м	10Mbps для 15 м	10Mbps для 15 м
Сигнал	Небалансный	балансный	балансный
Двоичная 1	-5В минимум -15В максимум	2В мин (B>A) 6В макс. (B>A)	1.5В мин (B>A) 5В макс. (B>A)
Двоичный 0	5В мин. 15В макс.	2В мин (A>B) 6В макс. (A>B)	1.5В мин (A>B) 5В макс. (A>B)
Мин. входное напряжение	(+/-) 3В	0.2В диф.	0.2В диф.
Выходной ток	500мА	150мА	250мА

На рис.16 приведены схемы соединения приемников и передатчиков и показаны их ограничения на длину линии (L) и максимальную скорость передачи данных (V).

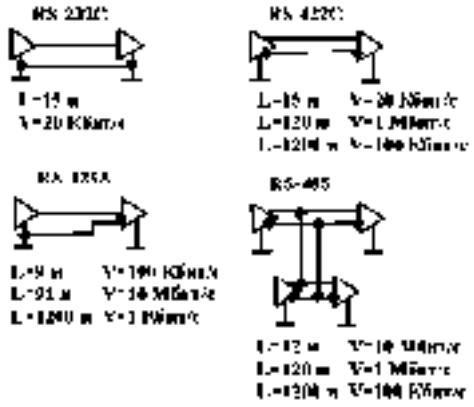


Рис.16. Стандарты асинхронного интерфейса

Наибольшее распространение в PC получил простейший из этих – стандарт RS-232C. В промышленной автоматике широко применяется RS-485, а также RS-422A, встречающийся в некоторых принтерах.

Интерфейс RS-232C предназначен для подключения аппаратуры, передающей или принимающей данные (ООД – оконечное оборудование данных, или АПД – аппаратура передачи данных), к оконечной аппаратуре каналов данных (АКД).

Интерфейс позволяет исключать канал удаленной связи вместе с парой устройств DCE (модемов), соединив устройства с помощью нуль-модемного кабеля (рис. 17).

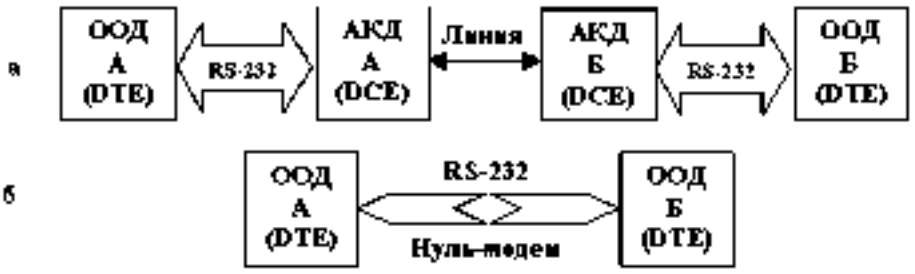


Рис. 17 **Полная схема соединения:**
а – **RS-232C;**
б – **нуль-модемное соединение**

RS-485/RS-422 используют экранированную витую пару, экран в качестве сигнальной земли. Хотя сигнальная земля обязательна, она не используется для определения логического состояния линии. Устройство, управляющее сбалансированной линией (balanced line driver), может (для RS-485 – обязательно, для RS-422 – нет) также иметь входной сигнал “Enable” (Разрешен), который используется для управления выходными терминалами устройства. Если сигнал “Enable” выключен, то это значит, что устройство отключено от линии, причем отключенное состояние устройства обычно называется “tristate” (т.е. третье состояние, вдобавок к двоичным 1 и 0).

Стандарт на RS-485 предусматривает только 32 пары передатчик/приемник, но производители расширили возможности RS-485 протокола, так что теперь он поддерживает от 128 до 255 устройств на одной линии, а используя репитеры, можно продлевать RS-485/RS-422 практически до бесконечности.

Также стандарт на RS-485 предусматривает использование двухжильной экранированной витой пары, так называемой 2-wire RS-485, но возможно использование и четырехпроводной витой пары (4-wire RS-485); тогда получается полный дуплекс.

Стандарт на RS-422 изначально предусматривает использование четырехжильной экранированной витой пары, но допускает соединения только от одного устройства к другим (до пяти драйверов и до десяти ресиверов на каждый драйвер).

RS-422 был придуман для замены RS-232 в тех случаях, когда RS-232 не удовлетворяет по скорости и дальности передачи. RS-422 использует строго разделенные две (или больше) пары проводов: одну пару для приема, одну для передачи (и еще по одной на каждый сигнал контроля/подтверждения (control/handshake)).

RS-485, благодаря наличию третьего состояния (“tristate”), позволяет обойтись одной парой проводов, что снижает общую стоимость системы при обеспечении связи на большие расстояния.

Практическая часть.

1. Исследовать с помощью задающих воздействий, подаваемых с тумблерных регистров универсального стенда для исследования БИС, работу параллельного интерфейса К580ВВ55(i8255) и универсального синхронно-асинхронного приемопередатчика К580ВВ51.
2. Реализовать заданные режимы работы микросхем параллельного интерфейса К580ВВ55(i8255) и универсального синхронно-асинхронного приемопередатчика К580ВВ51.
3. Составить карту адресов внешних устройств для учебной микро-ЭВМ “Микролаб”.

Контрольные вопросы

1. Дать определение порта ввода-вывода.
2. Сколько 8-разрядных портов включает в себя БИС КР580ВВ55А?
3. Описать формат управляющего слова, задающего режим работы программируемого параллельного интерфейса.
4. Описать основные режимы работы ППИ.
5. С помощью каких сигналов происходит обмен информацией между портами ввода-вывода и шиной данных?

6.Описать отличия последовательного порта ввода-вывода и параллельного порта.

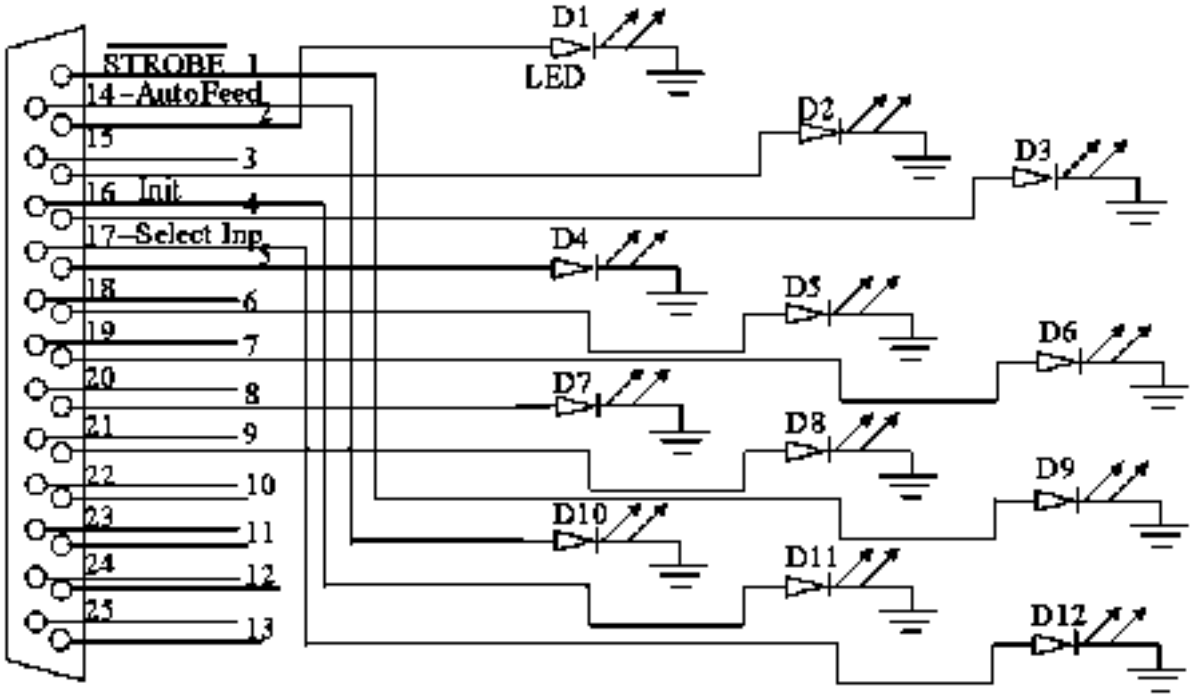
7.Назначение буфера шины данных в ППИ.

Практическая работа №16. Интерфейсы микропроцессорных систем (интерфейс Centronics).

Цель работы: изучить интерфейс Centronics, ознакомиться с работой параллельных портов ПЭВМ.

Порядок выполнения практической работы.

1. Подключить к ПЭВМ к порту LPT схему для определения правильного прохождения сигналов по шине данных в стандарте Centronics.



2. Написать на языке Ассемблера программу тестирования параллельного порта, которая обеспечивает режим бегущей единицы на выходных линиях LPT-порта (минимальное время горения одной лампочки – 3 с).

Содержание отчета

1.Программная и логическая модель LPT-порта компьютера.

2.Схема тестирования LPT-порта компьютера.

Контрольные вопросы.

1.Опишите структуру LPT-портов PC.

2. Опишите режимы работы современных LPT-портов компьютера.

Практическая работа №17. Интерфейсы микропроцессорных систем (интерфейс RS-232).

Цель работы: изучить программирование интерфейса RS-232 в составе лабораторного комплекса.

Порядок выполнения практической работы.

1.Собрать лабораторный комплекс в соответствии с рисунком. Соединить учебную микроЭВМ “Микролаб” и ПЭВМ с использованием “нуль-модемного” (Null Modem) кабеля. (Тип монитора – клавиатурный).

2.Написать программы для ПЭВМ и “Микролаб”, обеспечивающие передачу байта через COM-порт со стороны ПЭВМ, его прием учебной микроЭВМ, инвертирование и передачу назад в ПЭВМ. ПЭВМ должна принять инвертированный байт назад, сложить его с переданным и результат вывести на экран ПЭВМ. Программу для ПЭВМ написать с использованием языка Паскаль, для “Микролаб” – языка Ассемблера. Формат передачи данных: длина символа 8 бит, стоповый бит, без бита четности, скорость 9600 , коэффициент пересчета скорости для КР580ВВ51А равен 64.

Содержание отчета

1. Программная модель COM-порта компьютера.
2. Тексты программ, разработанных в ходе выполнения практической работы.
3.Схема соединения компьютеров с использованием нуль-модемного кабеля.

Контрольные вопросы.

1.Опишите структуру COM-портов PC.

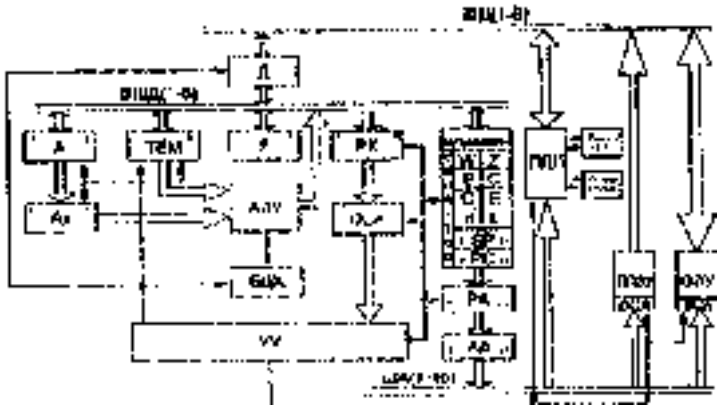
2.Опишите скорости передачи информации через COM-порты современных компьютеров.

Практическая работа №18. Ознакомление с учебной микро-ЭВМ (проектирование микропроцессорных систем)

Цель работы: изучить общую структуру микро-ЭВМ «Электроника-580», принцип работы микро-ЭВМ и представление информации в микроЭВМ.

Теоретическая часть.

Современные микро-ЭВМ строятся на базе интегральных схем с высокой степенью интеграции элементов. Их структура и принцип функционирования существенно определяются типом используемого микропроцессора. Широкое распространение получили ЭВМ с микропроцессором фирмы Intel. Основы их функционирования можно изучить, используя работы, где описывается ЭВМ на базе 8-разрядного однокристального параллельного микропроцессора КР 580 ВМ 80 А или его аналога Intel 8080. Для изучения принципа работы ЭВМ и отладки программ для постоянных запоминающих устройств (ПЗУ) предназначено УОУ " Электроника -580", блок-схема которого представлена на рисунке 1.



УОУ содержит следующие блоки:

АЛУ - арифметико-логическое устройство микропроцессора;

А - аккумулятор - специальный 8-разрядный регистр, используемый для передачи в АЛУ операндов, приема результатов вычислений из АЛУ и временного хранения операндов;

ТЕМ - программно-недоступный регистр, хранящий промежуточные результаты выполнения операций;

F- регистр признаков (флажков), предназначенный для фиксации значений логических переменных, изменяющихся в результате выполнения команды;

БДК - блок двоично-десятичной коррекции результата операции;

УУ - устройство управления микропрограммного типа, формирующее последовательность управляющих сигналов в соответствии с обрабатываемой командой;

РК - однобайтный регистр команд, предназначенный для приема первого байта команды;

DCK - дешифратор команд;

DCA - дешифратор адреса ячейки памяти;

W,Z - программно-недоступные регистры краткосрочного хранения информации во время выполнения команды;

B, C, D, E, H, L - 8-разрядные регистры общего назначения(РОН);

SP - указатель стека, 16-разрядный счетчик адреса операндов и команд, находящихся в стековой памяти;

РС - счетчик команд, предназначенный для определения адреса следующей команды;

РА - регистр адреса, хранящий адрес следующей команды или операнда, который передается на шину адреса;

ПЗУ,ОЗУ- полупостоянное и оперативное запоминающее устройства соответственно;

ПДП - узел прямого доступа , осуществляющий выборку информации для дисплея(8-разрядного буквенно-цифрового индикатора) из ячеек ОЗУ и ввод информации с клавиатуры с помощью специальной программы-монитора, хранящейся в ППЗУ;

Ак, Ад, Д- буферы (регистры) аккумулятора, адреса и данных соответственно, предназначенные для увеличения нагрузочной способности шин;

ША, ШД, ВШД - шины адреса, данных и внутренняя шина данных соответственно(16- ,8- , 8- разрядные);

УОУ расположено в корпусе, внешний вид которого представлен на следующем рисунке:

На передней панели УОУ расположены:

- тумблер "Сеть" и сигнальная лампочка включения питания 220В;
- тумблер "Прогон-отладка", изменяющий режим работы УОУ ;
- лампочки "C","Z", характеризующие состояния разрядов регистра F;
- дисплей (2) ;
- таблицы команд микропроцессора КР580(1);
- клавиатура (3), состоящая из 25 управляющих клавиш ;
- лампочки "Вход" и "Выход" , характеризующие работу магнитофона (МФ) с УОУ.

Работу микро-ЭВМ рассмотрим на основе блок-схемы, представленной на рисунке 1. Микро-ЭВМ содержит:

– микропроцессор КР 580 ВМ 80А, выполненного на одной БИС, в которую включено АЛУ, БДК, А, ТЕМ, Ак, Д, Ад, РК, РОН, РА, ДСК, SP, PC;
- программируемого параллельного интерфейса КР580 ВВ 55;
- шинного формирователя К589 АП16;
- многорежимного буферного регистра К589 ИР12;
- статического ОЗУ на микросхемах К585РУ2А (емкость каждой 1024х1 бит);
- ППЗУ К573РФ2 (2048х8 бит) и микросхемы серии К 155.

БИС КР580ВМ80А представляет собой 8-разрядный микропроцессор, включающий в себя операционное и управляющее устройства. УУ содержит управляющую память, в которую записаны микропрограммы выполнения операций. Каждая из микропрограмм, предназначенная для реализации определенной команды, содержит от 4 до 17 микрокоманд, управляющих приемом из ОЗУ операндов, выполнением над ними простейших действий, выдачей в ОЗУ операндов и извлечением из ОЗУ очередной команды. В КР580 ВМ 80А не предусмотрена возможность изменения содержимого управляющей памяти, поэтому он относится к числу немикропрограммируемых с фиксированной системой команд. С использованием управляющей памяти микропроцессор реализует первый уровень управления ЭВМ - микропрограммное управление, предназначенное для обработки каждой команды, располагаемой в РК.

Второй уровень управления ЭВМ - командный, организуется хранимой в ОЗУ программой. Программа занимает определенную область ОЗУ, имеет обязательную начальную и конечную команды. Обращение к начальной команде с пульта управления ЭВМ или с помощью другой программы способствует передаче ее в РК и циклической обработке следующих команд, поступающих друг за другом в РК (естественный порядок) до извлечения конечной команды программы. Последняя команда заканчивает процесс программного управления ЭВМ, устанавливает блоки ЭВМ в начальное состояние и осуществляет прерывание вычислений по данной программе.

Каждая команда содержит поле кода операции (КОП) и поле адреса устройств (РОН, ячейки ОЗУ или ПЗУ, А и др.), участвующих в операции. Если устройством является один из РОН, то для указания адреса любого из них достаточно поля КОП длиною в 3 двоичных разряда. Если операция выполняется с участием ячейки ОЗУ, то необходимая разрядность поля адреса для определения номера ячейки характеризуется емкостью памяти ППЗУ и ОЗУ. При максимальной их размерности 64К ячеек для определения номера любой из них требуется 16-разрядное поле адреса. Поэтому в зависимости от разновидности операции и участвующих в ней устройств в микро-ЭВМ используются одно-, двух- и трехбайтные команды. В однобайтной команде 5 разрядов отводится под КОП и 3 разряда - под адрес устройства микропроцессора. В двух- и трехбайтных командах под поле КОП отводится первый байт команды. Вторые и третьи байты используются или под операнд (непосредственная адресация), или под код адреса (прямая адресация) ячейки в ПЗУ или ОЗУ.

Первый байт команды всегда располагается в РК, другие - в РОН, в которые они заносятся в последующие такты извлечения команды. После извлечения команды счетчик РС инкрементируется (РС=РС 1) для указания адреса следующей команды, при изменении естественного порядка в РС устанавливается код адреса из двух РОН (W, Z) при выполнении команд управления, реализующих условный (безусловный, отличный от естественного порядка) переход. Иногда содержимое 2-го и 3-го байтов команды заносится в H и L в качестве адреса операнда, используемого следующей командой при выполнении операции.

Взаимодействие устройств микропроцессора удобнее всего рассмотреть на примере выполнения какой-либо команды. Так, выполнение команды сложения содержимого А с ячейкой памяти ОЗУ осуществляется в следующей последовательности:

- передача текущего адреса из РС в РА;
- инкрементация РС, т.е. установление адреса следующей команды;
- поступление адреса команды из РА через ША на адресные входы схемы памяти (ДСА), что обеспечивает выборку адресуемой ячейки памяти и передачу ее содержимого через ШД в буфер Д;
- занесение из буфера Д команды в РК;
- дешифрирование поля КОП команды ДСК и поступление унитарного кода в УУ, запуск в нем соответствующей микропрограммы, с помощью которой формируются управляющие сигналы, обеспечивающие настройку микропроцессора на выполнение заданной в команде операции (сложения);
- поступление адреса операнда из РОН (H и L) в РА, передача операнда в АЛУ и сложение его с А, занесение результата в А;
- извлечение следующей команды по адресу, расположенному в РС.

Заметим, что на выполнение команды сложения микропроцессору требуется два машинных цикла. Во время первого цикла происходит обращение к памяти за командой и ее обработка, во время второго - извлечение из памяти операнда и выполнение операции сложения. После выполнения команды в РС устанавливается адрес следующей команды, который на единицу больше предыдущей. Тем самым после выполнения операции сложения будет извлечена следующая команда, расположенная в естественном порядке в ОЗУ. Для извлечения команды, расположенной по другому адресу, в программу включают команды управления. Особенности выполнения этих команд рассмотрим на примере трехбайтной команды безусловного перехода.

Она осуществляется в такой последовательности:

- выдача текущего адреса из РА в ДСА;
- увеличение РС на единицу (инкрементация РС);
- выбор первого байта команды из памяти по коду адреса в РА;
- передача его через ШД, буфер Д, ВШД в РК;
- дешифрация в ДСК поля команды КОП и формирование в УУ последовательности управляющих знаков;
- извлечение второго байта команды (младших 8 разрядов кода адреса следующей команды) по адресу в РС, занесение его в Z, инкрементация РС;
- извлечение третьего байта команды безусловного перехода (старших 8 разрядов кода адреса следующей команды) по адресу РС и занесение его в W;
- передача содержимого группы регистров W,Z в РС и РА.

Заметим, что команда безусловной передачи управления выполняется за 10 тактов и при частоте тактового генератора 2 мГц выполняется за 5,0 мкс. В результате выполнения команды в РС устанавливается адрес следующей команды, отличный от естественного порядка, равный коду адреса, расположенного в третьем и втором байте команды безусловного перехода.

Отличительной особенностью команды условного перехода от команд управления безусловного перехода является проверка значения логического условия в соответствии с типом команды перед последней микрооперацией передачи содержимого группы регистров W и Z в РС. Если условие выполняется, то содержимое из W,Z передается в РС ; если нет, то РС инкрементируется. Содержимое РС затем передается а РА.

Таким образом, для организации процесса вычислений в ЭВМ с использованием команд необходимо разместить в ОЗУ программу и операнды, к которым команды будут обращаться для выполнения предписанных операций. Операнды и программы размещаются в разных рабочих областях ОЗУ в зависимости от структуры памяти ЭВМ. Рабочая (не заштрихованная) область памяти УОУ представлена на рисунке 3.

Ячейки памяти с адресами 0000-:-03FF (16) расположены в ППЗУ. В эти ячейки для облегчения процесса занесения программы и данных в ОЗУ, а также для контроля и организации вычислений введены команды программы Монитор, являющейся простейшей операционной

системой. Монитор в процессе своей работы использует оперативную информацию, которую хранит в ячейках ОЗУ с адресами 83E0-:-8400. Поэтому эта область, хотя и является рабочей областью памяти, программистом не используется. Область памяти с адресами 0400-:-7FFF и 87FF-:-FFFF является нерабочей, так как по данным адресам отсутствуют ячейки памяти.

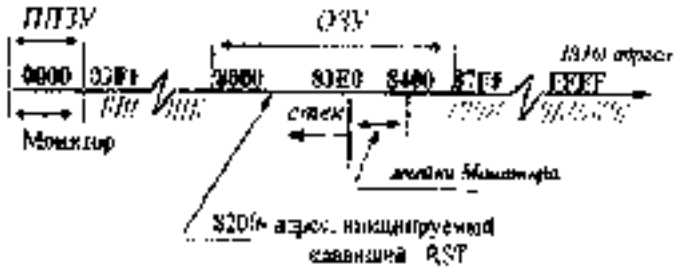


Рисунок 3

Данные и команды в ЭВМ кодируются в двоичной системе счисления и имеют обычно формат, кратный одному байту. Для кодирования чисел запятую чаще всего фиксируют после младшего разряда, знак ("1"("-"), "0"(" ")) размещают в позиции самого старшего разряда. При этом все дробные числа, с учетом предварительного сдвига влево, представляют целыми, а величину, на которую они сдвигаются (коэффициент фиксации), в дальнейшем используют для получения правильного результата вычислений. Так, если числа размещаются в восьмиразрядных ячейках памяти и имеют вид B= - b6b5b4b3b2b1b0,0...0 с учетом коэффициента фиксации, то в разрядной сетке ячейки памяти они представляются следующим образом:

ст мл

b7	b6	b5	b4	b3	b2	b1	b0

где разряд b7 является старшим (ст) и в нем размещается знак числа. Запятая располагается после младшего (мл) разряда b0, а все нули после запятой отбрасываются. Например, число $B=-7{,}51_{(10)}$ для представления в разрядной сетке сначала кодируется в двоичной системе счисления в прямой код числа $B=-7{,}51_{(10)}=-111{,}100110011\ldots$, затем представляется "целым" - путем сдвига на один (на величину коэффициента фиксации), разряд влево $B^*=-1111{,}0_{(2)}$, затем размещается в разрядной сетке

ст мл

1	0	0	0	1	1	1	1

Для упрощения операции сложения чисел с разными знаками или замены операции вычитания чисел с одинаковыми знаками числа в ЭВМ чаще всего представляются (переводятся программно) в дополнительном коде. Дополнительный код положительного числа B совпадает с прямым [B]пр, а отрицательного числа - равен результату, получаемому путем инвертирования всех разрядов числа B, кроме знакового, и прибавления единицы в разряд с весом 2^0 .Так, число

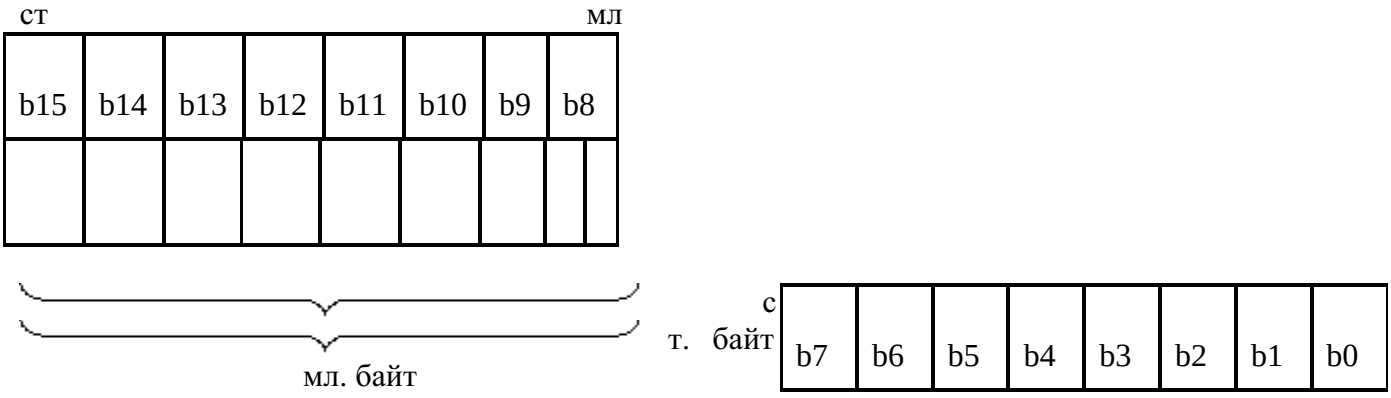
[B*]пр=1000 1111;

[B*]д =1111 0000 2^0 =1111 0001.

Если в старшем разряде стоит 0, подобная запись представляет положительное число, для которого прямой и дополнительный коды идентичны.

В одном байте ячейки памяти могут размещаться числа, представленные в дополнительном коде, в диапазоне от $1000\ 0000_{(2)}$=-128 до $0111\ 1111_{(2)}= 127_{(10)}$. Если такой диапазон представления мантиссы числа недостаточен для получения необходимой точности вычислений, так как в восьми разрядах можно разместить только семь разрядов мантиссы, а остальные младшие разряды отбрасываются, то для представления числа следует использовать две или более ячейки памяти. Так, при размещении числа в двух ячейках памяти ЭВМ работает с "целыми" числами следующего формата:

$$B^* = \sum_{i=0}^{14} b_i * 2^i, 0...0$$



При этом диапазон чисел, представленных в дополнительном коде, равен -32768 -:- 32767 с учетом знакового разряда числа, имеющего вес -2^{15} .

Кроме двоичной системы, в ЭВМ часто используется шестнадцатеричная система счисления. Более компактная запись чисел в этой системе позволяет уменьшить вероятность ошибок и увеличить скорость ввода информации, уменьшить число индикаторов контроля правильности преобразования информации и расширить диапазон представления чисел. В этой системе алфавит состоит из 16 символов: 0,...,9,A,B,C,D,E,F, каждый из которых может быть размещен в одной из позиций числа. Ввиду кратности оснований систем счисления правила перевода чисел из двоичной системы счисления в шестнадцатеричную и обратно весьма просты. Сначала двоичное число разбивается на тетрады, затем каждая тетрада заменяется соответствующим шестнадцатеричным символом. Например, разбиение числа 11110111 на тетрады справа налево даст 1111 0111, а в результате замены шестнадцатеричным эквивалентом получаем число $F7_{(16)}$. Обратное преобразование шестнадцатеричного числа осуществляется заменой символов 0,1,,,...,F тетрадами двоичного эквивалента. Например,

$6FD_{(16)} = 0110\ 1111\ 1101_{(2)}$.

Практическая работа №19. Ознакомление с учебной микро-ЭВМ «Электроника-580» (абстрактное представление)

Цель работы: изучить инструкцию по эксплуатации УОУ "Электроника-580"и работу программы-эмулятора «МикроЭВМ-580».

Теоретическая часть.

Для работы с микро-ЭВМ используется 25 клавиш, из которых верхний ряд и правый столбец являются командными. Нажатие командных клавиш способствует вызову программы Монитор и выполнению соответствующих действий УОУ. Остальные 16 клавиш (клавиши данных) служат для ввода в УОУ шестнадцатеричных цифр (0,...,F). Назначение клавиш следующее:

RST (сброс) - способствует установке УОУ в исходное состояние;

ADDR (адрес) - устанавливает УОУ в режим задания адреса ячейки памяти;

MEM (память) - служит для перевода УОУ в режим записи данных в ячейку памяти;

NEXT (следующий) - увеличивает на 1 адрес, инициируемый на дисплее ячейки памяти, или обращается к следующему регистру УУ;

CLR (восстановление) - восстанавливает начальное значение адреса или данных, если после их ввода не задействовались командные клавиши;

REG (регистр) - инициирует содержимое регистра УУ;

STEP (шаг) - осуществляет пошаговый (ручной) режим выполнения команд;

RUN (прогон) - запускает выполнение программы в автоматическом режиме до команды останова либо до введенной контрольной точки;

BRK (контрольная точка) - служит для задания адреса контрольной точки в программе.

С учетом включения командных клавиш клавиши данных могут использоваться для задания имен регистров и регистровых пар микропроцессора:

A,B,C,D,E,8/H,9/L,F - для обозначения регистров A-:-L и F;

I/P - для указания стека SP;

2/T - для обозначения содержимого вершины стека SP.

Индикатор адреса и данных.

Индикатор адреса и данных состоит из восьми разрядов. Каждый разряд является семисегментной ячейкой на светодиодах и отображает цифры 0-9; буквы A,C,E,F и буквы B,D,R. При отображении ячейки памяти в разрядах 1-4 индикатора в шестнадцатеричной системе счисления высвечивается адрес, в разрядах 7,8 - данные, хранящиеся по этому адресу. В других случаях в разрядах 1-:-4 отображается, например, содержимое счетчика команд, а в разрядах 7,8 - очередная команда либо содержимое регистра микропроцессора. В последнем случае в пятом разряде инициируется наименование регистра. Например, при чтении содержимого ячейки памяти B9 (16) с адресом 817A (16) мы увидим на индикаторе 817A B9, где 817A отображает состояние счетчика команд, а B9 - содержимое ячейки при этом состоянии. Если при этом состоянии необходимо инициировать содержимое какого-либо регистра, то на индикаторе высвечивается 817A B-A6.

Ввод команд и данных в ОЗУ.

Программа размещается в рабочей незанятой области ОЗУ (рисунке 3) и содержит последовательность закодированных в двоичной системе команд. Каждая программа имеет начальную и конечную команды.

Последовательность команд внутри программы определяется естественным порядком. Поэтому для записи программы нужно определить ее объем в ячейках памяти и выбрать адрес начальной команды. При свободной ОЗУ программу размещают с первой рабочей ячейки ОЗУ, т.е. с адреса $8000_{(16)}$; если эта ячейка занята другой программой или данными, выбирают другую свободную область ОЗУ и начинают запись программы с ее начального адреса.

Если программа может быть размещена в любой области ОЗУ, то данные непременно размещаются по адресам, указанным в программе.

Рассмотрим ввод программы и данных на примере. Пусть фрагмент программы состоит из 3-х последовательных команд CNZ 8200, SBB D, MOV A, M, закодированных C4, 00, 82, 9A, 7E. Причем для реализации первой команды требуется отвести ячейку 8200. Следовательно, ячейка под номером $8200_{(16)}$ будет занята и программу надо разместить в свободной области. Выберем область памяти для фрагмента программы начиная с ячейки памяти под номером $820A_{(16)}$ и введем последовательно команды в ОЗУ, для чего подключим монитор в режим записи символов в ОЗУ. Такой режим обеспечивается нажатием клавиши ADDR (фиксация адреса ячейки, в которой будет производиться запись). Так как адрес ячейки 820A, то последовательно нажимаем клавиши 8,2,0,A. При этом после каждого нажатия этих клавиш на индикаторе в разрядах 1-:-4 будет высвечиваться xxx8, xx82, x820, 820A, где x - произвольный символ предыдущего состояния индикатора. Таким образом, монитор подключился к ячейке памяти 820A, однако, чтобы происходила запись в эту ячейку, необходимо нажатием клавиши MEM включить режим записи. При срабатывании этого режима в шестом слева индикаторе дисплея высвечивается символ "," (запятая). Теперь можно первый байт команды CNZ 8200 вводить в ячейку 820A. Введем его нажатием клавиш C,4. При этом на индикаторе дисплея разрядах 7,8 последовательно появится информация xC, C4. Таким образом, ячейка 820A занимается КОП команды, которая должна быть продолжена в последующих двух ячейках (команда трехбайтная). Для повышения скорости ввода в мониторе предусмотрена подпрограмма автоматического подключения следующей (предыдущей) ячейки памяти нажатием клавиши NEXT или подключения предыдущей ячейки памяти нажатием клавиши MEM. При этом в режиме записи информации в ячейку должна обязательно быть запятая в 6-м разряде индикатора.

Нажмем клавишу NEXT, счетчик адреса увеличится на единицу; и на индикаторе дисплея в разрядах 1-:-4 появится 820B. Теперь введем второй байт команды нажатием клавиши 0,0. На индикаторе в двух разрядах справа последовательно появится x0, 00. Нажмем вновь NEXT, в

разрядах 1-:-4 индикатора отобразится 820C. Введем 8,2. Затем аналогично введем коды 9A и 7F в ячейки 820D и 820E соответственно. Таким образом , наша программа разместилась в ячейках 820A-820E. Содержимое программы можно проверить многократным нажатием клавиши MEM, при этом ячейки будут просматриваться в обратном порядке. При наличии неправильного кода в ячейке и "," в 6-м разряде индикатора код можно исправить путем ввода необходимых символов.

Ввод данных выполняется аналогично записи команд программы, только данные размещаются по адресам, указанным в программе. В нашем случае программа работает с операндом по адресу 8200. Нажимаем клавишу ADDR, клавиши 8,2,0,0, клавишу MEM и вводим число в шестнадцатеричной системе счисления.

Если при вводе данных или команд допущена ошибка, ее можно исправить повторным правильным нажатием клавиш. При этом в ячейке останется информация, отображаемая на дисплее, соответствующая нажатию двух последних клавиш. Нажатие клавиши CLR восстанавливает в ячейке содержимое первоначальной записи, если другие командные клавиши не нажимались.

При попытке ввести информацию в ячейку без предварительного нажатия клавиши MEM, а также при адресе ячейки ППЗУ либо нерабочей ячейки ОЗУ на дисплее отображается сигнал ошибки "E r r".

Чтение и запись информации в регистры

Для записи данных в регистры микропроцессора необходимо нажать последовательно клавиши REG и Ri, где Ri - клавиша обозначения регистра.

После нажатия клавиш в разряде 5-го индикатора отобразится имя регистра, а в разрядах 7,8 - его содержимое (осуществляется режим чтения). Пусть необходимо осуществить запись информации в аккумулятор, например, рассмотрим число $A8_{(16)}$. Нажимаем клавишу REG , затем клавишу A, при этом на дисплее отобразится REG A-xx. Нажимаем клавишу A, затем 8, на экране последовательно инициируется REG A-xA, REG A-A8. Нажатие клавиши NEXT устанавливает адрес следующего регистра. При этом в регистре с новым именем можно прочитать или изменить содержимое нажатием клавиш данных.

При чтении содержимого регистровой пары надо нажать последовательно три клавиши ADDR, RP, MEM, где RP∈{I/P, 8/H, B, D, 2/T}. При нажатии клавиши I/P, 8/H, B, D, 2/T в регистрах 5-, 6-го индикатора отобразится имя регистровой пары SP, HL, BC, DE, SГ соответственно, а в разрядах 1-:- 4 - ее содержимое

Чтение содержимого ОЗУ или ППЗУ

Чтение содержимого ячейки памяти по адресу $NIKM_{(16)}$ осуществляется нажатием командной клавиши ADDR и последовательным нажатием клавиш данных N, I, K, M. При этом на экране сигнализируется адрес ячейки и ее содержимое (XX): NIKM xx. Для чтения содержимого следующей ячейки нажимается клавиша NEXT, предыдущей - MEM.

Индикация ошибок

При неверных действиях оператора на индикаторе появится код ошибки Err xxxx. Он высвечивается в следующих случаях:

- при попытке записи в несуществующую ячейку ОЗУ или в ППЗУ, а также если была блокирована возможность ввода данных в память (не нажата клавиша MEM);
- при попытке установить несуществующее имя регистра;
- при попытке нажать клавишу RP П {B, D, 8/H, 1/P, 2/T}, отличную от обозначений символов регистровых пар для операций ADDR RP MEM или ADDR RP BRK (RP П {B, D, 8/H, 1/P, 2/T});
- при попытке запустить программу на выполнение клавишами STEP или RUN, если введено меньше четырех символов адреса после нажатия клавиши ADDR.

Если появится сигнал ошибки Err, то нажатием CLR или ADDR можно восстановить предыдущее состояние счетчика команд и саму команду. Нажатием клавиши MEM восстанавливается предыдущее значение ячейки памяти и ее адрес.

Отличия программы “Эмулятор МикроЭВМ-580” от устройства Электроника-580

Внешний вид программы “Эмулятор МикроЭВМ-580” версия 1.00 представлен на рисунке 4.

Рисунок 4

По сравнению с устройством Электроника-580 в эмулятор добавлены следующие клавиши:

“Остановка” – остановка программы в момент ее исполнения с помощью “Run”. Служит для остановки зациклившихся программ.

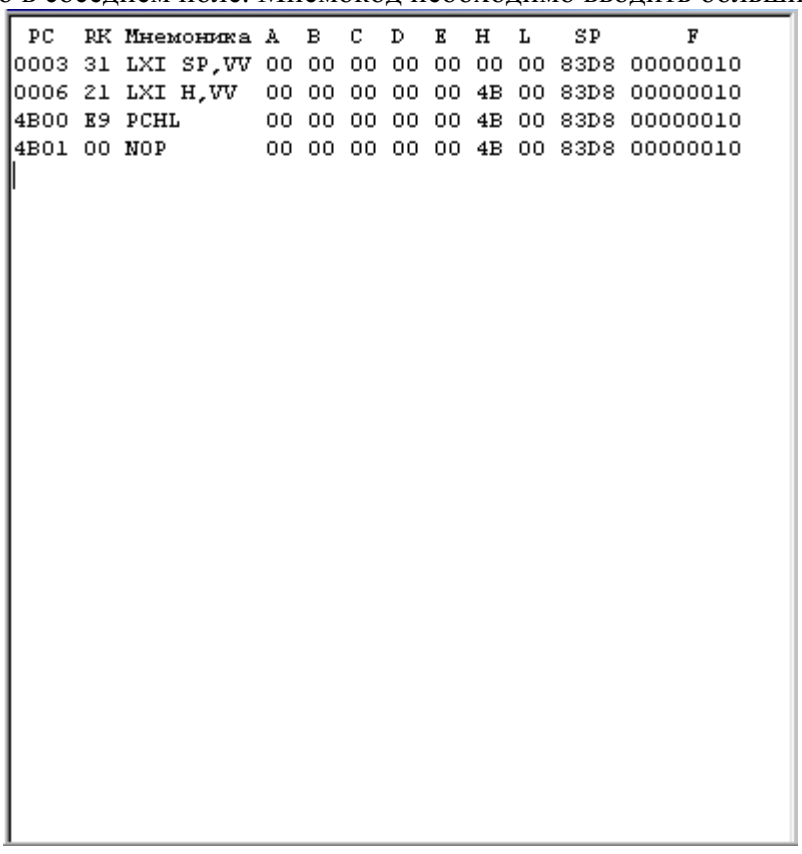
“Текст трассировки” – вызывает окно с текстом трассировки (рисунок 5) в котором в последней строке отображается текущее состояние регистров. информация обновляется по мере выполнения команд. В регистре RK находится команда, которая была выполнена последней, рядом отображается ее мнемокод. При достижении 500 строк в окне “Текст трассировки” первые строка удаляется для избежания перегрузки вашего ПК.

“Просмотр памяти” – вызывает окно (рисунок 6), в которое осуществляется считывание памяти виртуальной Электроники-580.

“Сохранить” – сохраняет содержимое памяти и регистров виртуальной Электроники-580 в файл.

“Загрузить ” – загружает содержимое памяти и регистров виртуальной Электроники-580 из файла.

В эмулятор добавлена эффективная система поиска команд: поиска кода по мнемокоду и поиска мнемокода по коду. Достаточно набрать в соответствующем поле код или мнемокод и искомое будет выведено в соседнем поле. Мнемокод необходимо вводить большими буквами.



```
 PC   RK Мнемоника A  B  C  D  E  H  L   SP      F
0003 31 LXI SP,VV 00 00 00 00 00 00 00 83D8 00000010
0006 21 LXI H,VV  00 00 00 00 00 4B 00 83D8 00000010
4B00 E9 PCHL      00 00 00 00 00 4B 00 83D8 00000010
4B01 00 NOP       00 00 00 00 00 4B 00 83D8 00000010
```

Рисунок 5

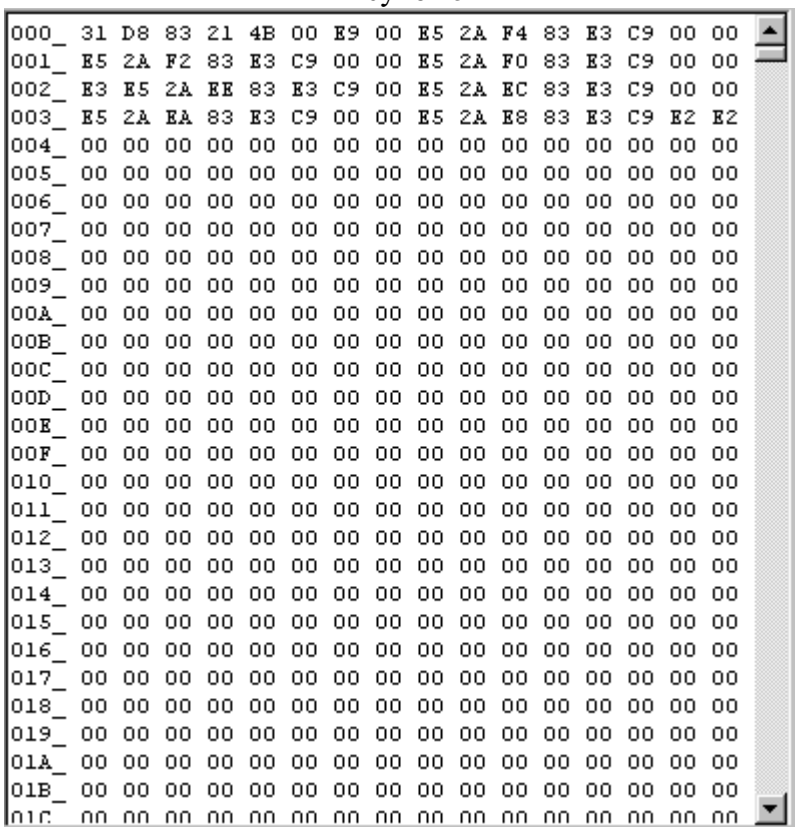


Рисунок 6

В программе "Эмулятор МикроЭВМ-580" рабочей является вся область памяти 0000-:-FFFF. В области 0000-:-003F записаны микропрограммы вызываемые микрокомандой RST.

В программе "Эмулятор МикроЭВМ-580" при вводе любых символов (адрес, данные) в соответствующем поле не возникает символов “x”. Происходит лишь сдвиг старых символов. Таким образом исключаются все возможные ошибки Электроники-580.

Индикация регистров “C” и “Z” производится лишь при выполнении команд в окне трассировке.

При выполнении команды RST программа переспрашивает пользователя о необходимости сохранения результатов работы и необходимости сброса.

Практическая работа №20. Ознакомление с учебной микро-ЭВМ «Электроника-580»(этапы проектирования)

Цель работы: изучить система команд микро-ЭВМ «Электроника-580».

Теоретическая часть.

Последовательность (закон) преобразования информации в ЭВМ задается программой. Программа размещается в ОЗУ в виде команд, представленных в машинных кодах. Представление программ в машинных кодах требует много времени на ввод команд и их проверку. Поэтому для уменьшения времени на программирование, вероятности ошибок кодирования, повышения наглядности представления команд для задания закона преобразования информации используется машинно-ориентированный язык Ассемблер. Поскольку различные микро-ЭВМ отличаются структурой и конструктивными особенностями, изменяется содержание и языка Ассемблер. Однако эти изменения имеют преемственность, а представление программы на языке Ассемблер позволяет получать наиболее короткие программы, занимающие меньший объем памяти по сравнению с программами, написанными на языках высокого уровня (БЕЙСИК, СИ). Программа, представленная в мнемонике Ассемблера, также должна быть оттранслирована, т.е. переведена и размещена в машинных кодах в ОЗУ . Для такого перевода необходима

программа-транслятор, которой в мониторе УОУ не имеется. Поэтому, хотя в данной работе и изучается язык Ассемблер и команды представляются в его мнемонике, запись команд в ОЗУ осуществляется в шестнадцатеричной системе счисления вручную с клавиатуры УОУ.

Система команд микропроцессора КР 580 ВМ 80А приведена в таблице. 1. Команды условно разбиты на пять групп, в первом столбце таблицы представлены наименования команд в мнемонике Ассемблера, где

Ri, Rj ∈ {A ,B ,C ,D , E , H , L , M };

V - 8-битное данное [$V_{(16)}$] ;

aa - 16-битный адрес памяти [$aa_{(16)}$] ;

RP- регистровая пара B,C; D,E; H,L или указатель стека SP ;

ap - 8-битный адрес порта [$ap_{(16)}$].

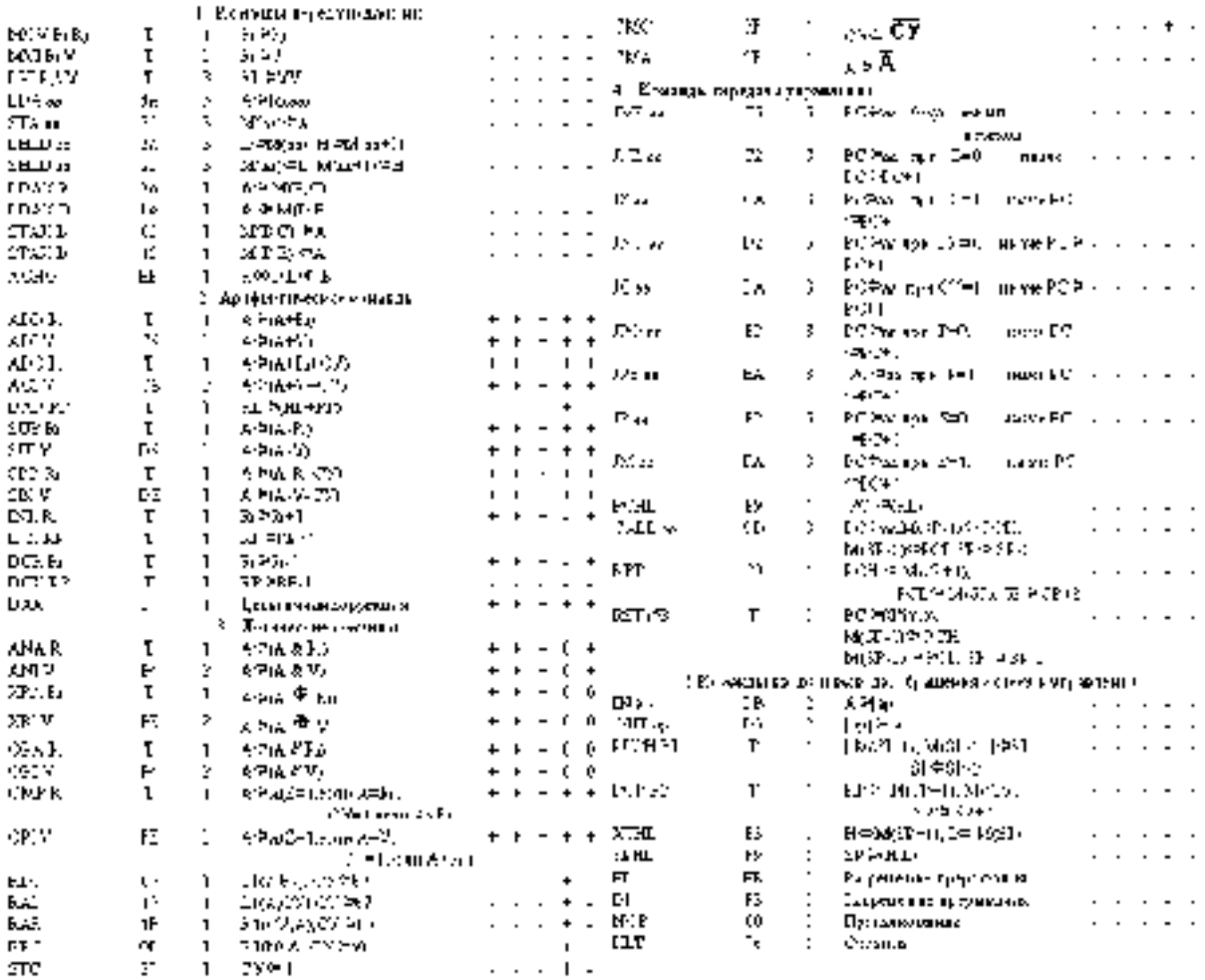
Во втором столбце таблицы указано содержание первого байта команды (ПБ), где T означает ссылку на дополнение к таблице 1, в котором ПБ конкретизирован в зависимости от типа команды. В третьем столбце таблицы 1 размещена размерность команды в байтах, в четвертом - её функциональное назначение, в последующих столбцах указано содержание поименованных разрядов регистра F после выполнения команды :

" " - возможно изменение в состояние, зависящее от результата выполнения команды ;

0 " - " - состояние не меняется ;

1 " 0 ", " 1 " - состояние нуля или единицы .

Таблица 1



Rj \ Ri	A	B	C	D	E	H	L	M
				MOV RJ, Ri				
A	7F	78	79	7A	7B	7C	7D	7E
B	47	40	41	42	43	44	45	46
C	4F	48	49	4A	4B	4C	4D	4E
D	57	50	51	52	53	54	55	56
E	5F	58	59	5A	5B	5C	5D	5E
H	67	60	61	62	63	64	65	66
L	6F	68	69	6A	6B	6C	6D	6E
M	77	70	71	72	73	74	75	------
ADD Ri	87	80	81	82	83	84	85	86
ADC Ri	8F	88	89	8A	8B	8C	8D	8E
ANA Ri	A7	A0	A1	A2	A3	A4	A5	A6
CMP Ri	BF	B8	B9	BA	BB	BC	BD	BE
DCR Ri	3D	05	0D	15	1D	25	2D	35
INR Ri	3C	04	0C	14	1C	24	2C	34
MVI Ri,V	3E	06	0E	16	1E	26	2E	36
ORA Ri	B7	B0	B1	B2	B3	B4	B5	B6
SUB Ri	97	90	91	92	93	94	95	96
SBB Ri	9F	98	99	9A	9B	9C	9D	9E
XRA Ri	AF	A8	A9	AA	AB	AC	AD	AE

DAD B	09	INX B	03	POP B	C1	
DAD D	19	INX D	13	POPD	D1	
DAD H	29	INX H	23	POPH	E1	
DAD SP	39	INX SP	33	POP PSW	F1	
DCX B	0B	LXI B,VV	01	PUSH B	C5	
DCX D	1B	LXI D,VV	11	PUSH D	D5	
DCX H	2B	LXI H,VV	21	PUSH H	E5	
DCX SP	3B	LXI SP,VV	31	PUSH PSW	F5	

I	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
RST I*8	C7	CF	D7	DF	E7	EF	F7	FF

1. Команды передачи данных

Первая группа команд в таблице 1 обеспечивает выполнение операций размещения, обмена, загрузки и перемещения данных. Число команд - 84 . Наибольшее число команд пересылки типа MOV Ri,Rj. Команды однобайтные, предназначены для пересылки операндов из одного регистра в другой или обмена информацией между РОН и ОЗУ.

Например, команда MOV A,B имеет код 78 $_{(16)}$ (дополнение к таблице1) адресация - регистровая, выполняет операцию передачи содержимого регистра B в регистр A (A⇐B).

Команды MOV M, Rj осуществляют передачу содержимого одного из РОН в ячейку памяти с адресом ,указанным в регистровой паре H, L. Поэтому перед выполнением данных команд необходимо предварительно поместить младший байт адреса ячейки M в регистр L, старший байт адреса в регистр H. Такая операция выполняется командой LHLD aa, где в ячейке с адресом aa располагают будущее содержимое L, а в ячейке aa 1 содержимое H. Например, после загрузки этой командой регистра H содержимым $82_{(16)}$, L - $40_{(16)}$ выполняется команда MOV M, A: код команды $77_{(16)}$ загружается в РК и из аккумулятора информация передается в ячейку с адресом $8240_{(16)}$.

Команды MOV Ri, M осуществляют передачу содержимого ячейки памяти M с адресом , указанным в регистровой паре H,L, в один из РОН .

Рассмотрим назначение других команд :

MVI Ri, V, осуществляют непосредственную передачу операнда, находящегося во втором байте команды , в РОН или ячейку памяти .

MVI B,V код - $06_{(16)}$, загружает в регистр B второй байт команды, равный V ;

MVI M,V код - $36_{(16)}$, осуществляет непосредственную передачу V в ячейку с адресом , указанным в регистровой паре H, L .

Команда LDA aa осуществляет прямую загрузку аккумулятора содержимым ячейки памяти с адресом aa , причем второй байт команды - младшие разряды адреса , третий байт - старшие .

STA aa осуществляет передачу содержимого аккумулятора в ячейку с адресом aa, т.е. M(байт3,байт2) ⇐A.

LHLD aa загружает регистровую пару H,L содержимым ячейки памяти с адресом aa, aa 1, т.е. H⇐ M(aa 1), L⇐ M(aa) .

LDAX B, LDAX D загружают аккумулятор содержимым ячейки по адресу, находящемуся в регистровой паре B,C; D,E соответственно .

STAX B,STAX D передают содержимое аккумулятора в ячейку памяти по адресу, находящемуся в регистровой паре B,C;D,E соответственно.

XCHG осуществляет обмен данными между регистрами H и D , L и E .

SHLD aa загружает две соседние ячейки памяти содержимым регистров H,L : M(байт3,байт2=aa(16)) ⇐ L , M(aa 1) ⇐ H.

2. Арифметические команды

Эти команды предназначены для выполнения операции сложения, сложения с переносом, вычитания, вычитания с заёмом, инкрементирования, декрементирования, десятичной коррекции аккумулятора. В большинстве случаев эти команды изменяют содержимое разрядов регистра флажков F, который содержит информацию следующего вида :

S	Z	0	AC	0	P	1	СУ
b7	b6	b5	b4	b3	b2	b1	b0

где разряд S - индикатор знака, Z - нуля,

AC - вспомогательного переноса,

P - четности, СУ - переноса.

Состояние регистра F используется последующими командами для выполнения и организации управления вычислительным процессом. Рассмотрим команды этой группы:

ADD Ri осуществляет сложение в прямых кодах содержимого аккумулятора и операнда, находящегося в РОН или ячейке памяти.

ADD B код- $80_{(16)}$, выполняет операцию A=A B, результат сложения остается в A .

ADD M код- $86_{(16)}$, складывает содержимое аккумулятора с операндом ячейки памяти по адресу, указанному в регистровой паре H,L.

ADI V код $C6_{(16)}$ выполняет сложение в АЛУ аккумулятора с операндом в команде, результат сложения заносит в A. В данной команде в первом байте располагается КОП, во втором - операнд V.

ADC Ri складывает содержимое A с РОН или ячейкой памяти и в младший разряд сумматора АЛУ прибавляет содержимое младшего разряда регистра F.

ADC M код- $8E_{(16)}$ складывает содержимое аккумулятора с ячейкой памяти с адресом, указанным в регистровой паре H,L, и к результату сложения прибавляет СУ. Результат операции передается в A.

ACI V осуществляет сложение аналогично ADI V, кроме того, к младшему разряду сумматора АЛУ прибавляет значение СУ регистра F(сложение с переносом).

DAD RP выполняет двухбайтное сложение регистровой пары RP с содержимым регистровой пары H,L .При наличии переноса Рст из сумматора -СУ=1 ,если его нет , то СУ=0 . Результат заносится в H,L.

DAD B код- $09_{(16)}$ выполняет операцию HL= BC HL; СУ=Рст.

SUB Ri позволяет непосредственно вычесть из аккумулятора содержимое РОН или ячейки памяти (A=A-Ri) и результат вычитания занести в A (сложить в дополнительном коде A=A [($\overline{R_i}$) 1] без учета знака вычитаемого) эта команда не выявляет переполнения и имеет другие разновидности:

SUI V - вычитание непосредственно из A операнда.

SBB Ri - вычитание с заемом A=A [($\overline{R_i}$) 1] [- СУ]д.

SBBV - вычитание непосредственно операнда с заемом A = A - V - СУ.

SBB M осуществляет вычитание A=A-M-СУ посредством сложения содержимого аккумулятора с инвертированным значением операнда, содержащимся в ячейке M, при СУ=1 и A=A $\overline{M}$ 1 при СУ=0. Рассмотрим эту операцию подробнее:

Пусть необходимо вычесть с заёмом из A содержимое ячейки $8204_{(16)}$.

Пусть $A=F8_{(16)}$, $M=30_{(16)}$, заема не было ; СУ=0.

Тогда в АЛУ под действием команды произойдет сложение: F 8 $_{(16)}$ 3 0 $_{(16)}$1111 1000 A = [-$8_{(10)}$] M= 0011 0000 = $48_{(10)}$1101 0000 [M]= [-$48_{(10)}$]Рст= 1 1100 1000 C 8 $_{(16)}$ A = [-56($_{(10)}$] 9 2

После сложения $F = 92_{(16)}$, т.е. 1001 0010

Из состояния разрядов регистра флажков видно, что S =1 ,т.е. в результате сложения получено отрицательное число (разряд 7 результата =1); Z =0(результат не равен 0); AC=0 (при сложении младшей тетрады нет переноса); P=0 (в результате сложения получилось число с нечетным числом "1"); СУ=0 (не нужен заём при вычитании из старшего байта; при вычитании СУ=, при выполнении сложения типа ADD СУ=Рст - перенос в старший байт при сложении с повышенной точностью). Если перед вычитанием СУ=1, то, выполняя команду SBB M, получим A=C7, $F=92_{(16)}$. Заметим, что до выполнения команды в H занесен код 82(16), в L- 04(16),а в ячейку $8204_{(16)}$ записан операнд $30_{(16)}$.

INR Ri, INR RP являются разновидностями команды ADD, осуществляют прибавление "1" к РОН, ячейке памяти или содержимому регистровой пары .

DCR RI , DCX RP декрементируют содержимое Ri или RP .

DAA корректирует результат сложения, хранящийся в A в двоичном коде, в двоичнодесятичный код <8421>:

при AC =1 или $b3b2b1b0_{(2)}$ >9 прибавляет 6 к младшей тетраде байта,

при СУ=1 или $b7b6b5b4_{(2)}$ >9 прибавляет 6 к старшей тетраде .

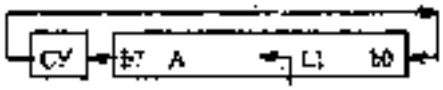
Команда DAA после ADD позволяет выполнить сложение в коде D1 .

3. Логические команды

Задачей этих команд является выполнение поразрядной конъюнкции, дизъюнкции , сложения по модулю 2 , сравнения содержимого A и Ri или A и V, а также циклического сдвига операндов в A (таблица 1). Например, команда XRA M(код- $AE_{(16)}$) осуществляет сложение по модулю 2 содержимого ячейки памяти M, расположенной по адресу, находящемуся в H,L, с аккумулятором, результат операции размещает в A.

```
1010 1011 =A
⊕ 1100 1100 =M
  0110 0111 ⇨A
```

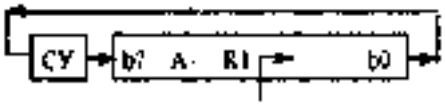
Команды сдвига осуществляют сдвиг чисел на один разряд циклически через разряд СУ регистра F

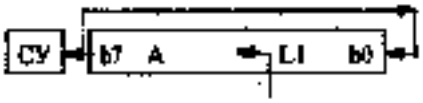


RAL L1(A,СУ), СУ п b7

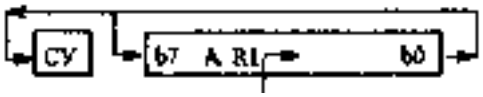
RAR R1(СУ,A), СУ п b0

или с занесением крайних цифр A в разряд СУ регистра





RLC L1(A,b7), СУ п b7RRC R1(b0,A), СУ п b0



При выполнении команд сравнения операндов CMP Ri , CPI V разряды Z и СУ регистра F устанавливаются в значения аналогично операции вычитания.

STC осуществляет присвоение СУ=1;

CMC инвертирует значение переноса СУ=СУ

CMA инвертирует все разряды A= $\overline{A}$.

4. Команды передачи управления

Четвертая группа команд содержит команды перехода, вызова, возврата и повторного запуска. Эта группа предназначена для изменения естественного порядка следования команд.

JMP aa - безусловный переход к команде, находящейся в ячейке (байт3, байт2 = aa). Второй и третий байты команды загружаются в W,Z и PC, а через регистр адреса, команда извлекает очередную необходимую команду программы.

JNZ aa извлекает команду аналогично JMP только при Z=0, в противном случае извлекается следующая по порядку команда, PC=PC 1. Таким же образом выполняются команды условного перехода JN, JNC, JPO, JPE, JP, JM в зависимости от состояний разрядов Z ,СУ, P, S регистра F.

PCHL передает содержимое регистровой пары H,L в счетчик команд, тем самым следующие команды извлекаются по адресу в PC.

CALL aa вызывает начальную команду подпрограммы, расположенной по адресу aa(16)= байт3 байт2 команды. При этом адрес следующей команды запоминается занесением в стековую память: в ячейку памяти M(SP-1) заносятся старшие разряды PC(PCH), в ячейку M(SP-2)- младшие PCL, значение указателя стека SP уменьшается на 2. Данная команда осуществляет безусловный переход к подпрограмме, однако часто требуется переход к подпрограмме в зависимости от состояния регистра F. Для этой цели используются следующие разновидности команды CALLaa : CNZ aa(C4), CZ aa(CC), CNC aa(D4), CC aa(DC), CPO aa(E4), CPE aa(EC), CP aa(F4), CM aa(FC).

В этих командах проверяются условия аналогично командам условного перехода и при истинности условий реализуют функции CALLaa , т.е. осуществляют переход к подпрограмме по адресу aa с занесением содержимого PC в стековую память. Если условие не выполняется, реализуется следующая команда PC 1 основной программы .

RET - безусловный возврат к команде основной программы : её адрес загружается из вершины SГ стековой памяти, к указателю стека прибавляется 2. Возможен также возврат по условию с использованием команд: RNZ(C0), RZ(C8), RNC(D0), RC(D8), RPO(E0), RPE(E8), RP(F0), RM(F8).

RSTi*8 - повторный пуск (рестарт) осуществляет прерывание выполнения основной программы, адрес команды основной программы передается в стековую память аналогично командам CALLaa. Счетчик команд загружается фиксированным адресом ППЗУ. Так, команда RST7*8 загружает PC адресом $0038_{(16)}=8*7=56_{(10)}$. Команды RST позволяют использовать подпрограммы монитора для организации процесса вычислений .

5. Команды ввода и вывода, обращения к стеку и управления микропроцессором Последняя группа команд выполняет операции помещения в стек и извлечения из него, ввода и вывода данных, обмена данными, подтверждения и неподтверждения прерываний, управления отсутствием операций и останова и установления маски прерываний.

INap - вводит данные в A из порта периферийного устройства, адрес которого определяется вторым байтом команды.

OUTap - выводит данные из A в порт, адрес которого определяется вторым байтом команды.

PUSH RP - помещает содержимое регистровой пары RP в стек. В ячейку M(SP-1) помещает значение старшего регистра, в M(SP-2) - значение младшего регистра пары, указатель стека дважды декрементируется SP=SP-2. Разновидностью этой команды является команда занесения в стек слова состояния процессора PUSH PSW, которая помещает в M(SP - 1) содержимое A, а в ячейку M(SP - 2) содержимое F, причем разрядам ячейки M(SP - 2) присваиваются b0⇐СУ, b2⇐ P, b4⇐AC, b6⇐Z, b7⇐S.

POP RP извлекает из стека в регистровую пару содержимое двух ячеек памяти, в младший регистр загружаются данные из M(SP), в старший - из M(SP 1). К указателю стека прибавляется 2. Разновидностью этой команды является POP PSW , которая из ячейки M(SP) загружает регистр F по следующему правилу : флагу СУ⇐b0,P⇐b2, AC⇐b4, Z⇐b6, S⇐b7; A⇐M(SP 1).

XTHL осуществляет обмен содержимого H,L с двумя ячейками вершины стека: L⇔M(SP), H⇔M(SP 1).

SPHL передает в указатель стека содержимое регистровой пары H,L.

EI разрешает прерывания. При выполнении EI прерывания не признаются.

DI - после выполнения этой команды микропроцессор игнорирует запросы на прерывания до появления команды DI.

HLT - останов вычислений.

Подпрограмма и стек

Использование стековой памяти оказалось весьма эффективным при построении компилирующих и интерпретирующих программ: программ, использующих многократные вычисления алгебраических функций и нестандартные арифметические операции; программ с обработкой операндов с последовательным размещением.

Принцип работы стековой памяти поясняется рисунком 7.

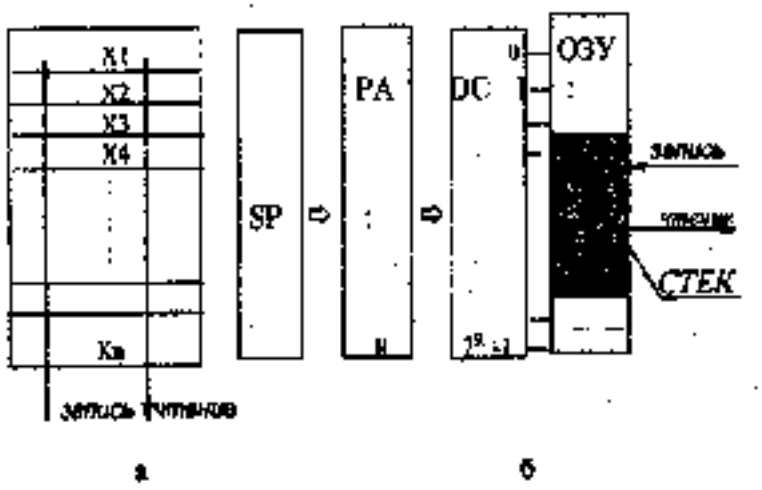


Рисунок 7

Работа стековой памяти осуществляется по правилу: "последним записан - первым считан". Так, вначале записывается в стек (рисунок 7а) число X1. Затем число X1 сдвигается вверх при записи числа X2 в стек. Тем самым любое записываемое число Xi сдвигает массив "вверх" и размещается в нижней ячейке. При чтении считывается содержимое самой "нижней "ячейки, а весь оставшийся массив чисел сдвигается "вниз". Размещая и считывая двоичные коды в указанном порядке, легко организовать поиск последнего записанного кода или организовать последовательную обработку кодов Xn,. . .,X1. Однако микрооперация сдвига массива требует значительных аппаратных затрат. С целью упрощения стековой памяти используют специальный реверсивный счетчик-указатель стека SP. При использовании SP не требуется микрооперация сдвига массива, достаточно перемещения возбуждения "вверх" или "вниз" выходных шин с дешифратора адреса. Так, при записи счетчик декрементируется SP-1 и число через шину данных записывается "вверх"(в сторону младших номеров шин DC), при чтении число сначала считывается, затем устанавливается адрес следующего операнда (SP 1) для возможного считывания.

При записи (считывании) двухбайтных операндов в восьмиразрядную память последовательно инкрементируется (декрементируется) дважды SP 2. При этом запись (считывание) осуществляется побайтно в две соседние ячейки : в M(SP),M(SP 1) - при считывании, в M(SP-1), M(SP-2) - при записи . Содержимое ячеек M(SP) и M(SP 1) определяет информацию в вершине стека (SГ), которая загружается в РОН при чтении стековой памяти.

Использование стековой памяти.

Часто в основной программе одни и те же операции (умножение, деление, сортировка массива и др.) выполняются многократно с различными данными. Целесообразно также нестандартные операции оформлять в виде программ, которые в основной программе использовать в виде подпрограмм, в свою очередь , подпрограмма так же может использовать при работе другие программы выполнения некоторых операций и т.д. В результате закон (порядок) работы ЭВМ может включать несколько вложенных друг в друга программ.

Для организации вычислительного процесса с применением подпрограмм используется стековая область ОЗУ. Эта область резервируется программистом для осуществления прерываний при вызове подпрограмм. При этом "низ" стека фиксируется установкой его адреса в SP командой SPHL (в УОУ монитор устанавливает автоматически SP=83E0) и, начиная с этого адреса, "вверх" (SP-1) размещаются промежуточные данные и коды адресов команд основных программ, которые должны выполняться вслед за выполнением подпрограмм. Для обращения к подпрограмме используется команда CALL аа и её модификации. Эти команды загружают в РС адрес аа начальной команды подпрограммы, а текущий адрес команды основной программы запоминается стековой памятью. Для возвращения вычислений из подпрограммы к следующей за CALL команде в подпрограмме используется команда RET и её модификации. Эта команда возвращает из стековой памяти в РС код адреса этой команды .

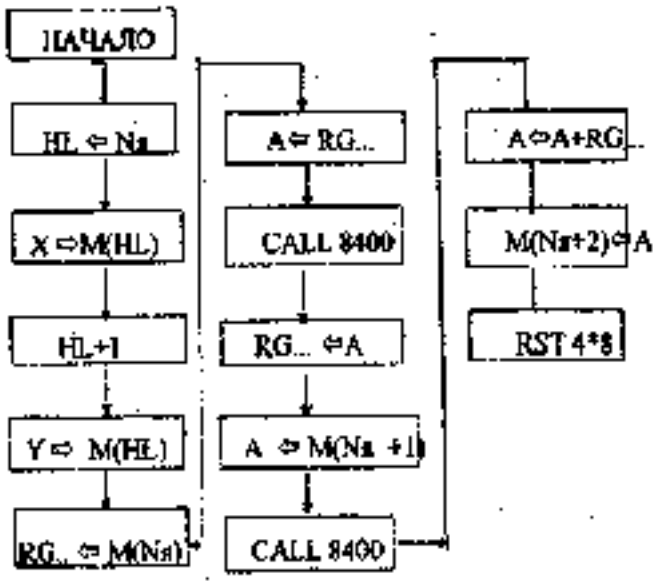


Рисунок 8.

Подпрограмма перевода числа в аккумуляторе в дополнительный код

8400	RAL	17
8401	INC 8404	D2
8402		04
8403		84
8404	CMA	2F
8405	RAR	1F
8406	INR A	3C
8407	JMP 8400	C3
8408		00
8409		84
840A	RAR	1F
840B	RET	C9

Следует отметить, что подпрограмма выполняет вычисления в процессоре с использованием набора тех же РОН, что и программа, поэтому, если данные основной программы, полученные в РОН перед выполнением подпрограммы, необходимы для вычислений после выполнения подпрограммы, их следует запомнить и восстановить, для чего перед обращением к подпрограмме данные заносятся в стековую память командами PUSH RP, а после её выполнения восстанавливаются командами POP RP.

Практическая работа №21. Ознакомление с учебной микро-ЭВМ «Электроника-580» (проектироваочные программы)

Цель работы: изучить правила составления и записи программ на микро-ЭВМ «Электроника-580».

Теоретическая часть.

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.

Пример 1. Программа загрузки и инвертирования содержимого аккумулятора.

Допустим, необходимо загрузить аккумулятор числом B4 в H-коде, затем инвертировать это число и разместить результат в ячейку памяти с адресом 8200.

Программа CMA1 на языке Ассемблер приведена в табл. 1. Последняя в программе команда HLT предназначена для останова при работе в пошаговом режиме. Для работы в непрерывном режиме эта команда должна быть заменена на команду RST4.

Программа CMA1 в машинных кодах и на языке Ассемблер приведена в табл.2. Из неё видно, что двухбайтовая команда загрузки аккумулятора MVI B4 размещается в ячейках памяти 8201 и 8202. Трёхбайтовая команда пересылки CTA размещена в трёх ячейках 8204, 8205 и 8206, причём в двух последних ячейках записывается адрес ячейки 8200, в которую отсылается результат выполнения операции инвертирования.

Заметим, что запись этого адреса в последовательно расположенные ячейки 8205 и 8206 производится начиная с младшего байта, т.е. в ячейку 8205 записывается байт 00, а затем в ячейку 8206 – байт 82.

Таблица 1

Программа CMA1 загрузки и инвертирования содержимого аккумулятора на языке ассемблер (вариант 1)

М-код	Операция	Комментарий
MVI	A, B4	Загрузить в аккумулятор число B4 в H-коде
CMA		Инвертировать содержимое аккумулятора
STA	8200	Поместить содержимое аккумулятора в ячейку памяти с адресом 8200
HLT		Остановить

Таблица 2

Адрес	H-код	М-код	Операнд	Комментарий
8200				Ячейка памяти для приёма результата
8201	3E	MVI	A,B4	Загрузить в аккумулятор число B4 из второго байта команды
8202	B4			
8203	2F	CMA		Инвертировать содержимое аккумулятора
8204	32	STA	8200	Поместить содержимое

				аккумулятора в ячейку памяти 8200
8205	00			
8206	82			
8207	76	HLT		Остановить МП

В рассмотренной выше программе при записи в аккумулятор (команда MVI) и память (команда STA) используется прямой способ адресации, когда адрес данных, участвующих в операции, следует в команде за кодом операции.

В целях освоения способов косвенной адресации полезно рассмотреть другой вариант программы решения этой же задачи, приведенный в табл.3. Второй вариант содержит большее число команд и, кроме того, он требует предварительной записи исходных данных в выделенную для этого ячейку памяти 8208.

Команда LXI H? 8208 в программе CMA2 использует фактически два адреса. Первый адрес (символ H) включён непосредственно в команду и определяет регистровую пару HL. Однако в этой регистровой паре нет операнда, как при прямой адресации, а содержится адрес ячейки памяти, в которой находится операнд (в данном случает B4). Аналогичный способ адресации применен в командах MOV A, M и MOV M,A.

Таблица 3

Программа CMA2 загрузки и инвертирования содержимого аккумулятора (вариант 2)

М-код	Операнд	Комментарий
LXI	H, 8208	Записать в регистровую пару HL число 8208 (адрес данных B4)
MOV	A,M	Загрузить в аккумулятор число из ячейки памяти с адресом, указанным в HL
CMA		Инвертировать содержимое аккумулятора
INX	H	Увеличить на 1 адрес в HL
MOV	M,A	Переслать число из аккумулятора по адресу, указанному в HL
HLT		остановить МП

Таблица 4

Программа CMA2 в машинных кодах и на языке Ассемблер

Адрес	Н-код	М-код	Операнд	Комментарий
8208				Ячейка памяти для предварительной записи числа B4
8209				Ячейка памяти для приёма результата
820A	21	LXI	H, 8208	Загрузить в регистровую пару HL число 8208
820B	08			
820C	82			
820D	7E	MOV	A,M	Загрузить в аккумулятор число из ячейки памяти с адресом 8208
820E	2F	CMA		Инвертировать содержимое A
820F	23	INX	H	Инкремент адреса в HL
8210	77	MOV	M,A	Переслать число из A в память по адресу 8209
8211	76	HLT		Остановить МП

Таблица 5

Программа СЛАМ сложения трёх чисел на Ассемблере

М-код	Операнд	Комментарий
LXI	H, 8212	Загрузить 8212 (адрес первого слагаемого) в HL
MOV	A,M	Поместить первое слагаемое из ячейки памяти в аккумулятор
INX	H	Инкрементировать содержимое пары HL
ADD	M	Сложить число из ячейки памяти, адрес которой указан в HL, с содержимым аккумулятора
INX	H	Инкрементировать содержимое пары HL
ADD	M	Сложить число из ячейки памяти, адрес которой указан в HL, с содержимым аккумулятора
INX	H	Инкрементировать содержимое пары HL
MOV	M,A	Поместить сумму, содержащуюся в аккумуляторе, в ячейку памяти, адрес которой указан в паре HL
HLT		Остановить МП

Таблица 6

Программа СЛАМ в машинном коде и на Ассемблере

Адрес	Н-код	М-код	Операнд	Комментарий
8212				Адреса ячеек памяти для предварительной записи слагаемых OF, 09 и 06
8213				
8214				
8215				Ячейка памяти для приёма суммы
8216	21 12 82	LXT	H, 8212	Загрузить адрес слагаемого OF, равный 8212, в пару HL
8217	7E	MOV	A,M	Поместить первое слагаемое OF в A
8218	23	INX	H	Инкрементировать пару HL до 8213
8219	86	ADD	M	Сложить число 09 из ячейки 8213 с содержимым A, равным OF
8220	23	INX	H	Инкрементировать пару HL до 8214
8221	86	ADD	H	Сложить число 06 из ячейки 8214 с содержимым A, равным сумме OF+09
8222	23	INX	H	Инкрементировать пару HL до 8215
8223	77	MOV	M,A	Поместить сумму OF+09+06 из A в ячейку памяти с адресом 8215
8224	76	HLT		Остановить МП

Таблица 7

Варианты заданий для составления программ СМА1, СМА2 и СЛАМ

№	Начальный адрес программы			Данные для СМА1,2	Слагаемые для СЛАМ		
	СМА1	СМА2	СЛАМ		1	2	3
1	8400 8428 8450 8478	8408 8430 8458 8480	8412 843A 8462 848A	6A 5 4E 59	07 44 B6 CE	1B AO 05 6C	C2 09 61 43

	84AO	84A8	84B3	A4	3B	29	OC
	84C8	84DO	84DA	E8	A4	05	27
	8500	8508	8512	4A	O8	C1	14
	8528	8530	853A	B7	1C	OA	54
	8550	8558	8562	CB	2A	4B	OE
	8578	8580	858A	76	4B	01	B2
	85AO	85F8	85B3	9A	07	B5	1C
	85C8	85DO	85DA	C7	3A	1B	09
	8600	8608	8612	A6	08	7B	24
	8628	8630	863A	E9	6C	CB	3A
	8650	8658	8662	3B	2E	1A	C3
	8678	8680	868A	87	17	09	У4
	86AO	85A8	86B3	C9	3A	45	C4
	86C8	86DO	86DA	B4	07	6B	15
	8700	8708	8712	49	5B	06	A2
	8728	8730	873A	E5	05	A6	22
	8750	8758	8762	OC	1E	06	5A
	8778	8780	878A	95	4C	2E	09
	87AO	87A8	87B3	C6	07	B5	16
	87C8	87DO	87DA	B9	15	AO	3E

Примечание: адреса и исходные данные для программ представлены в шестнадцатиричном коде; при составлении программ исходные данные и результаты следует размещать в начале выделенного для данной программы части адресного поля.

П Р И М Е Р 2. Программа сложения трёх чисел.

Рассмотрим пример сложения содержимых трёх последовательных ячеек памяти и размещения их суммы в четвёртой. Версия на Ассемблере представлена в табл.5, а запись программы в машинных кодах – в табл.6.

Исходные данные для этой программы предварительно заносятся в ячейки памяти с адресами 8212,8213,8214 (числа OF, 09 и 06 соответственно), а для результата резервируется ячейка с адресом 8215. Для приведенных трёх чисел сумма в Н-коде равна EI.

Порядок выполнения работы

В процессе домашней подготовки.

1.Выбрать из табл. 7 исходные данные для составления программы СМА1, СМА2 и СЛАМ.

2.Составить программы СМА1, СМА2 и СЛАМ на Ассемблере и в машинном коде, используя для кодирования команд табл.2

В учебной лаборатории.

1.Подготовить УОУ «Электроника-580 к работе, соблюдая следующие правила:

-запрещается работать с УОУ при снятом кожухе и задней панели;

-запрещается включать УОУ в сеть, не подключив заземления к клеммам 2 земля»;

-нажимать на клавиши следует без чрезмерных усилий, чётко фиксируя фазы нажатия и отпускания клавиши.

2.Произвести загрузку программы СМА1 по табл.4 из примера 1 в режиме ОТЛАДКА.

Для записи первой команды MVI A, B4 в ячейки 8201 и 8202 требуется нажать клавиши

ADDR 8201 MEM 3E NEXT B4

В указанной последовательности нажатие клавиши NEXT приводит к переходу от адреса 8202 к адресу 8203. Дальнейший ввод данных согласно последовательности

NEXT 2F NEXT 32 NEXT 00 82 NEXT 76 завершает загрузку программы.

3.Выполнить программу СМА1 в пошаговом режиме, для чего

3.1.Установить начальный адрес программы 8201 последовательным нажатием клавиш ADDR 8201, контролируя запись по индикаторам АДРЕС и ДАННЫЕ;

3.2.Нажатием клавиши STEP выполнить первую команду программы; после этого на индикаторах АДРЕС байта следующей команды СМА (т.е. 8203), а на индикаторах ДАННЫЕ отобразится Н-код этой команды;

3.3.Повторить п.3.2 для выполнения остальных команд программы, наблюдая за показаниями индикаторов;

3.4.Проверить результат выполнения операции инвертирования величины В4, для чего произвести чтение информации в ячейке памяти 8200 нажатием клавиш ADDR 8200.

4.Загрузить и выполнить собственную программу СМАI.

5.Произвести загрузку и выполнить программу СМА2 по табл.6 из примера 1.

6.Загрузить и выполнить собственную программу СМА2

7.Произвести загрузку и выполнить программу СЛАМ из примера 2.

8.Загрузить и выполнить собственную программу СЛАМ. Предварительно необходимо в первые три ячейки выделенного для этой программы адресного поля записать значения слагаемых из табл.7.

Содержание отчёта

1.Исходные данные и программы СМА1, СМА2 и СЛАМ на Ассемблере и машинном языке.

2.Тест-программа проверки функционирования ОЗУ на Ассемблере.

Контрольные вопросы

1.С какой целью осуществляется мультиплексирование ЩД?

2.Перечислите все программно-недоступные регистры МП с указанием их назначения.

3.Составьте программу загрузки регистра В константой 5А с последующим сложением с содержимым 9Е аккумулятора.

4.Какую информацию содержит первый байт команды? Второй? Третий?

5.Объясните работу стека в МП-системе.

6.Какую функцию выполняет схема десятичной коррекции в АЛУ?

7.Объясните назначение всех управляющих выводов в МП.

8.С какой целью используется ПДП?

9.Каким образом осуществляется адресация РОН?

10.Составьте программу логического сложения чисел В9 и 4Е.

11.Объясните работу всех командных клавиш УОУ.

12.Какую информацию и с какой целью фиксирует регистр признаков АЛУ?

13.Объясните характер выполнения всех возможных операций в АЛУ.

14.В каком случае изменяется содержимое программного счётчика.

15.Почему передача управляющих сигналов в МП-системе осуществляется с квитированием?

16.Составьте программу вычисления (в пошаговом режиме) арифметической прогрессии, если разность прогрессии равна 2А, а первый член ряда – 8000 в Н-коде.

17.Какая наибольшая ёмкость памяти может быть адресована 16 адресными линиями?

18.Как изменится содержимое FF регистровой пары HL после инкрементирования?

19.Определите слово состояния программы и слово состояния процессора.

20.В чём отличие косвенной адресации от прямой?

21.Составьте программу приёма в МП константы 6А из порта 04 и выдачи в порт В5 утроенного значения этой константы.

Практическая работа №22. Работа на учебной микро-ЭВМ (разработка архитектуры МП).

Цель работы: написать на языке ассемблера микропроцессора КР580ВМ80А программу, реализующую вычисления по заданному арифметическому выражению. Оттранслировать программу в машинные коды и выполнить ее отладку на УОУ "Электроника-580".

Практическая часть.

Пример программы к практической работе.

Пусть необходимо составить программу, реализующую следующую функцию:

$$Z = \max\{(2 * X + 17), |Y - 25|\}$$

Так как ассемблер является языком низкого уровня, т.е. его команды оперируют непосредственно со структурными элементами микропроцессорной системы, программист перед началом составления программы вычислений должен знать, в какой форме будут представлены входные и выходные переменные, по каким адресам ОЗУ они будут размещены. Из задания к лабораторной работе видно, что исходные переменные X, Y и результат вычислений Z являются числами со знаком, поэтому они представляются в дополнительном коде. Длина их не превышает одного байта. Расположим переменные X и Y в ОЗУ по адресам 8300H и 8301H соответственно. Результат Z после окончания вычислений разместим в ОЗУ по адресу 8300H, т.е. на месте исходной переменной X.

Создание программы начнем с составления алгоритма, соответствующего порядку вычислений при нахождении значения заданной функции. В общем случае алгоритмы решения задач на компьютере могут составляться в несколько этапов – со все более подробной их детализацией. На заключительном этапе алгоритм составляется с учетом особенностей его конкретной реализации на той или иной микро-ЭВМ, с учетом особенностей системы команд микропроцессора. В нашем случае задача очень простая и сразу же может быть разработан максимально подробный алгоритм, ориентированный на систему команд микропроцессора КР580ВМ80А.

Адрес 8300H используется в ходе выполнения программы несколько раз, сначала – для считывания переменной X, а после окончания вычислений – для занесения значения Z. Поэтому для упрощения обращения к ячейке памяти с этим адресом занесем его в регистровую пару HL, наиболее удобную для организации косвенной регистровой адресации.

Умножение на целое число в функции реализуется путем многократного сложения.

Так как данные представляются в дополнительном коде, то для получения абсолютной величины значения Y-25, в случае, если оно отрицательно, используются операции инверсии и добавления единицы.

После составления алгоритма записывается соответствующая программа на языке ассемблера. Полученные алгоритм решения задачи и текст программы с необходимыми комментариями приведены ниже. Команды, на которые в программе есть переходы, помечены метками ADR1, ADR2 и FIN.

При отладке программы на УОУ “Электроника-580” завершающей программу командой является RST 4, которая выполняет функцию “возврат к монитору”.

Получение машинных кодов команд программы при работе на УОУ “Электроника-580” осуществляют вручную с помощью таблицы команд. Альтернативный вариант трансляции – использование программы кросс-ассемблера AVMAC85.

Листинг трансляции рассматриваемой программы также приведен ниже. В нем показан пример введения и использования символьной константы CH1, которой присваивается с помощью команды транслятора EQU значение 17. Начальный адрес размещения рассматриваемой программы в памяти программ отладочного устройства определяется равным 8200H с помощью команды транслятора ORG. Длина полученного кода программы составляет 28 байт. Первая команда LXI H, 8300H трехбайтная и занимает три ячейки памяти, причем во второй ячейке размещается младший байт, а в третьей – старший байт числа 8300H. Далее следуют две однобайтные команды, потом – двухбайтная команда ADI CH1, занимающая ячейки 8205H и 8206H, причем во второй ячейке размещается константа CH1=17=11H. Далее опять следует однобайтная команда и т.д.

В командах переходов при трансляции вместо символических адресов –меток подставляются соответствующие им конкретные физические адреса памяти – 8212H, 821AH и 821BH. Так как переходы в примере выполняются вперед по программе, то при ручной трансляции этих команд физические адреса переходов еще неизвестны, и необходимо

зарезервировать две ячейки памяти под второй и третий байты команды перехода. Затем, когда процесс трансляции дойдет до команды, на которую выполняется переход, и становится известным физический адрес ее размещения в памяти, зарезервированные под этот адрес байты могут быть заполнены.

После ввода в память УОУ программы в машинных кодах необходимо подготовить и ввести в соответствующие ячейки памяти исходные переменные примера. Далее проводится отладка программы – проверка правильности ее функционирования. Отладка программы может быть также выполнена с помощью отладчика-симулятора AVSIM85.

Для рассмотренной программы были получены следующие результаты контрольного примера:

При X1=-4=FCH, Y1=-8=F8H - Z=21H=33;

При X2=21=15H, Y2=-8=F8H - Z=3BH=59.

ЛИСТИНГ ТРАНСЛЯЦИИ ПРОГРАММЫ

```
                          1          DEFSEG LAB1, ABSOLUTE=0000
                          2          SEG LAB1
                          3
=0011                     4     CH1: EQU  17
                          5
=8200                     6          ORG  8200H
                          7
8200 21 00 83             8     LAB1: LXI  M,8300H
8203 7E                   9           MOV  A,M
8204 87                  10           ADD  A
8205 C6 11               11           ADI  CH1
8207 47                  12           MOV  B,A
8208 3A 01 83            13           LDA  8301H
820B D6 19               14           SUI  25
820D F2 12 82            15           JP   ADR1
8210 2F                  16           CMA
8211 3C                  17           INR  A
8212 B8                  18     ADR1: CMP  B
8213 F2 1A 82            19           JP   ADR2
8216 70                  20           MOV  M,B
8217 C3 1B 82            21           JMP  FIN
821A 77                  22     ADR2: MOV  M,A
821B 00                  23     FIN:  NOP
                         24           END
```

Варианты заданий к практической работе.

Вар	Функция	A(10)	B(10)	C(10)	X1(10)	X2(10)	Y1,2(10)	Адр X(16)	Адр Y(16	Адр. Z(16)
1	Z=max{|A*X-Y+B|, C}	3	23	77	-40	11	-10	8300	8301	8301
2		4	17	62	-31	5	-20	8300	8301	8302
3		5	31	84	-22	9	-25	8301	8300	8300
4	Z=min{(|A*X-Y|+B), C}	3	14	58	-16	21	-24	8301	8300	8301
5		4	19	79	-24	6	-32	8301	8300	8302
6		5	9	93	-11	16	-17	8300	8301	8300
7	Z=max{|A*X-B|, (Y+C)}	3	33	100	-18	29	-41	8300	8301	8301
8		4	10	81	-12	25	-15	8300	8301	8302
9		5	21	119	-14	23	-27	8301	8300	8300
10	Z=min{(A*|X|-B), (Y+C)}	3	49	50	-35	3	-80	8301	8300	8301
11		4	55	64	-10	26	-67	8301	8300	8302
12		5	60	104	-23	10	-71	8300	8301	8300
13	Z=max{A*(|X-B+Y|), C}	3	65	44	-9	32	-57	8300	8301	8301
14		4	16	27	-13	24	-22	8300	8301	8302
15		5	50	99	-29	85	-59	8301	8300	8300
16	Z=min{A*(|X-Y|+B), C}	3	17	105	-38	2	-21	8301	8300	8301
17		4	1	82	-28	11	-9	8301	8300	8302
18		5	7	96	-13	12	-5	8300	8301	8300
19	Z=max{A*|X-B|, (Y+C)}	3	22	91	-4	34	-43	8300	8301	8301
20		4	8	69	-7	37	-2	8300	8301	8302
21		5	13	83	-6	14	-12	8301	8300	8300
22	Z=min{A*(|X|-B), (Y+C)}	3	37	54	-46	20	-66	8301	8300	8301
23		4	48	74	-38	63	-85	8301	8300	8302
24		5	61	109	-70	72	-60	8300	8301	8300
25	Z=max{A*|X+Y-B|, C}	3	20	70	-15	41	-6	8300	8301	8301
26		4	18	110	-1	52	-11	8300	8301	8302
27		5	15	90	-7	24	-3	8301	8300	8300
28	Z=min{A*(|X+Y-B|), C}	3	34	13	-53	61	-19	8301	8300	8301
29		4	26	68	-79	71	-31	8301	8300	8302
30		5	11	35	-52	66	-42	8300	8301	8300

Практическая работа №23. Работа на учебной микро-ЭВМ (системное проектирование).

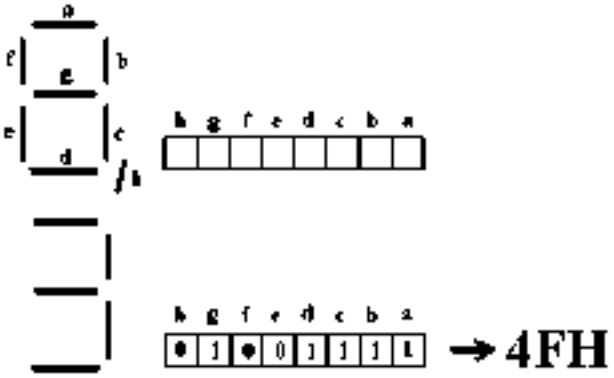
Цель работы: написать на языке ассемблера микропроцессора КР580ВМ80А программу, осуществляющую вывод на 1 и 2 индикаторы (крайние левые) УОУ "Электроника-580" заданной бесконечной последовательности шестнадцатеричных цифр. Вывод значений должен выполняться с заданной периодичностью. Оттранслировать программу в машинные коды и выполнить ее отладку.

Практическая часть.

Пример программы к практической работе.

Цель работы состоит в создании программы по управлению аппаратными средствами, в данном случае - собственными аппаратными средствами отладочного устройства. Программа должна с определенной периодичностью менять на индикаторах УОУ заданную последовательность символов (шестнадцатеричных цифр), подчиняющуюся той или иной закономерности.

Принцип формирования изображения какого-либо символа на семисегментном индикаторе состоит в том, что каждому сегменту индикатора соответствует определенный разряд посылаемого на этот индикатор двоичного кода. Если сегмент нужно зажечь, то в этот разряд записывается "1", в противном случае - "0" Соответствие между сегментами индикатора и разрядами двоичного кода, а также пример получения кода цифры "3" проиллюстрированы на следующем рисунке:



Для того, чтобы информация появилась на нужном индикаторе, необходимо ее записать в буфер индикации. Буфер индикации представляет собой область ОЗУ учебно-отладочного устройства, из которой записанные там коды передаются на соответствующие индикаторы. Адреса ячеек буфера индикации и их соответствие конкретным индикаторам на передней панели УОУ приведены в таблице 11.

Таблица 11. Буфер индикации

Адрес	Индикатор
83F8H	1-й индикатор (левый)
83F9H	2-й индикатор
83FAH	3-й индикатор
83FBH	4-й индикатор
83FCH	5-й индикатор
83FDH	6-й индикатор
83FEH	7-й индикатор
83FFH	8-й индикатор (правый)

Необходимые для работы индикаторов семисегментные коды шестнадцатеричных символов в принципе могут быть получены путем аппаратной дешифрации соответствующих четырехбитных значений от 0H до FH. В УОУ используется программный способ получения кодов, при котором в ПЗУ по адресам 02B3H - 02C2H помещена таблица семисегментных кодов всех символов (табл. 12), и адрес ячейки с нужным кодом может быть вычислен путем сложения соответствующего значения с базовым адресом таблицы. Например, адрес, по которому хранится семисегментный код цифры "3", равен 02B3H + 3H = 02B6H.

Таблица 12. Семисегментные коды символов

Адрес	Символ	Код
02B3	0	3F
02B4	1	06
02B5	2	5B
02B6	3	4F
02B7	4	66
02B8	5	6D
02B9	6	7D
02BA	7	07
02BB	8	7F
02BC	9	6F
02BD	A	77
02BE	B	5D
02BF	C	39
02C0	D	5E
02C1	E	79
02C2	F	71

Требуемая временная задержка может быть обеспечена за счет выполнения необходимого числа раз некоторого программного цикла. В простейшем случае в цикл включаются лишь команды отсчета числа этих циклов. Для определения времени выполнения цикла подсчитывается количество машинных тактов, за которое выполняются команды цикла. Время выполнения одного машинного такта на УОУ "Электроника - 580" в режиме "прогон" составляет 0,5 мкс (тактовая частота 2 МГц). Зная время выполнения одного цикла и величину необходимой задержки, легко определить необходимое количество циклов повторения. Число машинных тактов, которое включает в себя каждая команда микропроцессора КР580ВМ80А приведено в описании его системы команд.

Рассмотрим пример подпрограммы временной задержки TIMER, перед обращением к которой (по команде CALL TIMER) необходимо загрузить число циклов задержки в регистровую пару BC.

```
TIMER: DCX B ; число машинных тактов равно 5
       MOV A,B ; 5
       ORA C; 4
       JNZTIMER; 10
       RET
```

В каждом из циклов содержимое регистровой пары BC уменьшается на единицу. Выход из программы осуществляется тогда, когда значение в регистровой паре станет равным нулю. Так как при выполнении команды DCX B признаки не устанавливаются, то для проверки обнуления регистровой пары используется операция ИЛИ между содержимым обоих регистров. Результат этой операции равен нулю только в том случае, когда содержимое каждого из регистров B и C

равно нулю. Четыре команды цикла выполняются за 24 такта или за 12 мкс. Таким образом, при коде временной задержки FFFFH, что соответствует 65535 циклам, будет получена задержка 12 * 65535 мкс = 786,42 мс (около 0,79 с). Для получения задержки, например, 0,35 с необходимо разделить это значение на длительность одного цикла: 0,35 с / 12 мкс = 29167 = 71EFH.

Рассмотрим далее в качестве примера программу, реализующую вывод на индикацию (на индикаторы 1 и 2) последовательности символов: 0_, _F, 1_, _E, 2_, _D, ... , E_, _1, F_, _0, 0_, _F, 1_, _E, ...

Логика данной последовательности состоит в том, что оба индикатора работают поочередно, на первом цифры сменяются в порядке возрастания от 0 до F, а на втором - в порядке убывания от F до 0. Для реализации последовательности в общем случае необходимо организовать цикл с соответствующим изменением переменных цикла и вычислением необходимых адресов таблицы семисегментных кодов для вывода их содержимого на индикаторы. Однако для упрощения программы можно непосредственно оперировать адресами таблицы семисегментных кодов, изменяя один от 02B3H до 03C2H, а другой от 03C2H до 02B3H. В качестве первого указателя таблицы семисегментных кодов будем использовать регистровую пару HL, а в качестве второго - регистровую пару DE.

Текст программы на языке ассемблера, реализующей заданную последовательность изменения символов с периодом 0,35 с (с использованием вышеописанной подпрограммы TIMER), приведен ниже.

```
START:  LXI H, 02B3H     ;     начальная установка указателей
        LXI D, 02C2H     ;     таблицы семисегментных кодов
CYCLE:  MOV A, M         ; загрузка кода символа по указателю 1
        STA 83F8H        ; вывод символа на индикатор 1
        XRA A            ; обнуление аккумулятора
        STA 83F9H        ; гашение индикатора 2
        INX H            ; перевод указателя 1 вперед по таблице кодов
        LXI B, 71EFH     ; загрузка кода временной задержки
        CALL TIMER       ; выполнение временной задержки
        STA 83F8H        ; гашение индикатора 1
        LDAX D           ; загрузка кода символа по указателю 2
        STA 83F9H        ; вывод символа на индикатор 2
        DCX D            ; перевод указателя 2 назад по таблице кодов
        LXI B, 71EFH     ; загрузка кода временной задержки
        CALL TIMER       ; выполнение временной задержки
        MOV A, E         ;     проверка, не вышел ли указатель 2
        CPI B2H          ;     за начало таблицы кодов
        JNZ CYCLE        ; если не вышел, то переход на начало цикла
        JMP START        ; иначе – переход на начальную установку
```

В приведенной программе для получения нулевого кода в аккумуляторе с целью гашения индикатора в первый раз используется команда XRA A (операция ИСКЛЮЧАЮЩЕЕ ИЛИ, когда оба операнда берутся из аккумулятора). Во второй раз используется тот факт, что при выходе из подпрограммы TIMER аккумулятор обязательно обнулен. В конце программы проверяется содержимое регистра E на равенство B2H, что является признаком выхода второго указателя (регистровой пары DE) за пределы таблицы семисегментных кодов (содержимое регистра D всегда равно 02H).

Основную программу длиной 44 байта и подпрограмму TIMER длиной 7 байт можно разместить в памяти в произвольном порядке - одну вслед за другой, общая длина кода программы получится равной 51 байт. Например, при расположении программы с адреса 8200H последний байт ее попадет по адресу 8232H.

Варианты заданий к практической работе.

Вар.	Последовательность изменения цифр на индикаторах	Δt, с
1	01, 23, 45, 67, …, CD, EF, 01, 23, …	0,4
2	FE, DC, BA, 98, …, 32, 10, FE, DC, …	0,45
3	01, 12, 23, 34, …, DE, EF, F0, 01, 12, …	0,5
4	FE, ED, DC, CB, …, 21, 10, 0F, FE, ED, …	0,55
5	08, 19, 2A, 3B, …, 6E, 7F, 08, 19, …	0,6
6	7F, 6E, 5D, 4C, …, 19, 08, 7F, 6E, …	0,65
7	00, 11, 22, 33, …, EE, FF, 00, 11, …	0,7
8	FF, EE, DD, CC, …, 11, 00, FF, EE, …	0,75
9	0_, _1, 2_, _3, 4_, _5, …, C_, _D, E_, _F, 0_, _1, 2_, _3, …	0,4
10	F_, _E, D_, _C, B_, _A, …, 3_, _2, 1_, _0, F_, _E, D_, _C, …	0,45
11	_0, _1, _2, …, _6, _7, 8_, 9_, A_, …, E_, F_, _0, _1, _2, …	0,5
12	_F, _E, _D, …, _9, _8, 7_, 6_, 5_, …, 1_, 0_, _F, _E, _D, …	0,55
13	_0, _1, _2, …, _E, _F, 0_, 1_, 2_, …, E_, F_, _0, _1, _2, …	0,6
14	_F, _E, _D, …, _1, _0, F_, E_, D_, …, 1_, 0_, _F, _E, _D, …	0,65
15	0_, 1_, 2_, 3_, _4, _5, _6, _7, 8_, …, B_, _C, …, _F, 0_, 1_, …	0,7
16	F_, E_, D_, C_, _B, _A, _9, _8, 7_, …, 4_, _3, …, _0, F_, E_, …	0,75
17	00, FF, 11, EE, 22, DD, …, EE, 11, FF, 00, 00, FF, 11, EE, …	0,8
18	0F, 1E, 2D, 3C, …, E1, F0, 0F, 1E, …	0,85
19	01, FE, 23, DC, 45, BA, …CD, 32, EF, 10, 01, FE, 23, DC, …	0,9
20	0_, 1_, _F, _E, 2_, 3_, _D, _C, …, E_, F_, _1, _0, 0_, 1_, _F, _E, …	0,95
21	0_, 1_, 2_, …, D_, E_, F_, _F, _E, _D, …, _2, _1, _0, 0_, 1_, 2_, …	1,0
22	0_, _1, 2_, _3, …, E_, _F, F_, _E, D_, _C, …, 1_, _0, 0_, _1, 2_, …	1,05
23	00, 01, 11, 12, 22, 23, 33, …, EE, EF, FF, F0, 00, 01, 11, 12, 22, …	1,1
24	FF, EF, EE, DE, DD, CD, CC, …, 11, 01, 00, F0, FF, EF, EE, …	1,15
25	00, 01, 02, 03, …, 0E, 0F, 1F, 2F, 3F, …, DF, EF, FF, 00, 01, 02, …	1,2
26	FF, EF, DF, CF, …, 1F, 0F, 0E, 0D, 0C, …, 02, 01, 00, FF, EF, …	1,25
27	00, 01, 02, 03, 14, 15, 16, 17, 28, …, 2B, 3C, …, 3F, 00, 01, 02, …	1,3
28	FF, EF, DF, CF, BE, AE, 9E, 8E, 7D, …, 4D, 3C, …, 0C, FF, EF, …	1,35
29	00, 01, 02, 03, 44, 45, 46, 47, 88, …, 8B, CC, …, CF, 00, 01, 02, …	1,4
30	FF, EF, DF, CF, BB, AB, 9B, 8B, 77, …, 47, 33, …, 03, FF, EF, …	1,45

Практическая работа №24. Работа на учебной микро-ЭВМ.

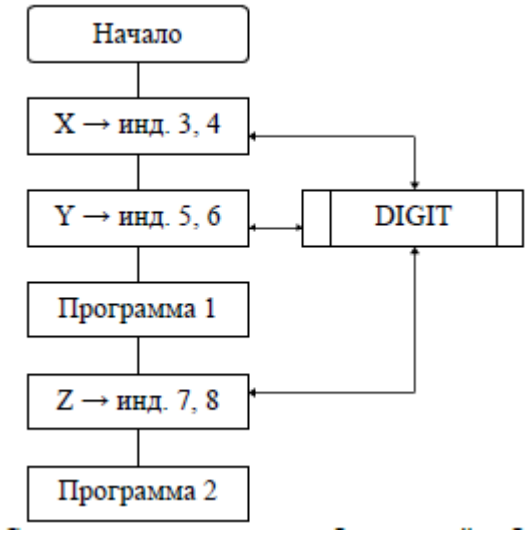
Цель работы: преобразовать программы, созданные в процессе выполнения практических работ №22 и №23 в единую программу, совмещающую функции ранее разработанных программ, а именно:

- реализующую вычисления по заданному арифметическому выражению, причем исходные данные X и Y, а также результат вычислений Z должны выводиться в шестнадцатеричной форме соответственно на 3-4, 5-6 и 7-8 индикаторы УОУ "Электрони ка-580";
- осуществляющую вывод на 1-2 индикаторы УОУ "Электроника-580" заданной последовательности шестнадцатеричных цифр (с заданной периодичностью).

Практическая часть.

Пример программы к практической работе.

Структура программы, совмещающей функции двух ранее разработанных программ с выводом значений X, Y и Z на индикацию, представлена на следующем рисунке:



Для реализации однотипных действий по выводу байта данных на индикацию здесь предусмотрено использование подпрограммы DIGIT, текст которой на языке ассемблера приведен ниже. Подпрограмма имеет два входа – DIGIT1 для вывода в виде шестнадцатеричного символа старшей тетрады байта, помещенного в аккумулятор, и DIGIT2 – для вывода младшей тетрады байта.

Соответствующий значению тетрады адрес таблицы семисегментных кодов формируется в регистровой паре HL. Адрес нужной ячейки буфера индикации перед обращением к подпрограмме должен быть помещен в регистровую пару BC. Длина подпрограммы – 14 байт.

```
DIGIT1:  RRC          ;      перемещение
         RRC          ;      старшей тетрады
         RRC          ;      в байте на место
         RRC          ;      младшей тетрады
DIGIT2:  ANI 0FH      ; обнуление старшей тетрады
         ADI B3H      ;      получение младшего байта адреса
         MOV L, A     ;      таблицы семисегментных кодов
         MVI H, 02H   ; загрузка старшего байта адреса
         MOV A, M     ; загрузка кода символа из таблицы
         STAX B       ; вывод символа на индикатор
         RET          ; возврат из подпрограммы
```

Например, для вывода с помощью подпрограммы DIGIT на индикаторы 3 и 4 значения X, считываемого из ячейки памяти 8300H, может быть использован следующий фрагмент программы.

```
LXI B, 83FAH    ; загрузка адреса индикатора 3
LDA 8300H       ; загрузка числа X
MOV D, A        ; дублирование X в регистре D
CALL DIGIT1     ; вывод старшей цифры X
INX B           ; получение адреса индикатора 4
MOV A, D        ; восстановление X в аккумуляторе
CALL DIGIT2     ; вывод младшей цифры X
```

Аналогичным образом могут быть выведены на индикацию значения Y и Z. Общая длина программы к лабораторной работе № 3 получится при этом около 130 - 140 байт (28 байт – пример программы 1, 51 байт – пример программы 2, 14 байт – подпрограмма DIGIT плюс дополнительные команды по выводу X, Y и Z на индикацию).

Практическая работа №25. Работа на учебной микро-ЭВМ (проектирование и формализация требований).

Цель работы*:* составить и отладить программу, выполняющую действия в соответствии с заданным вариантом.

Практическая часть.

Ниже в качестве примера приведена подпрограмма умножения двух однобайтных целых чисел без знака MUL88. Она реализует алгоритм умножения “младшими разрядами вперед со сдвигом частичной суммы”. Множимое помещается в регистр D, множитель в регистр C, произведение – двухбайтное число - образуется в регистрах B и C. При выполнении умножения последовательно анализируются биты множителя, начиная с младшего, помещаемые по команде RAR (циклический сдвиг вправо через перенос) в разряд переноса CY. Если очередной бит множителя ненулевой, то множимое добавляется к старшему байту частичной суммы, находящемуся в регистре B. Далее выполняется сдвиг этого байта вправо (вторая команда RAR), при этом его младший бит вытесняется в разряд переноса CY. При сдвиге в следующем цикле содержимого регистра C значение CY заносится в его старший бит, а в CY помещается очередной бит множителя. Таким образом в регистре C множитель постепенно вытесняется младшим байтом частичной суммы, которая после выполнения восьми циклов дает результат умножения. Длина описанной программы - 19 байт.

```
MUL88:  MVI  B, 00H     ; сброс старшего байта произведения
        MVI  E, 08H     ; установка числа бит
NXBIT:  MOV  A, C       ; множитель занести в аккумулятор
        RAR             ; очередной бит во флажке “CY”
        MOV  C, A       ; возвращение множителя
        DCR  E          ; декремент счетчика бит
        RM              ; умножение закончено?
        MOV  A, B       ; старший байт произведения
        JNC  NOADD      ; бит множителя равен нулю
        ADD  D          ; суммирование множимого
NOADD:  RAR             ; сдвиг частичной суммы
        MOV  B, A       ; возвращение старшего байта
        JMP  NXBIT      ; умножение на следующий бит
```

Варианты заданий к практической работе.

Вар.	Содержание программы
1	Перевод 8-битного числа из двоично-десятичного кода в двоичную систему счисления с выводом на
2	Перевод 8-битного числа из двоичного вида в двоично-десятичный код с выводом на индикацию
3	Перевод 16-битного числа из двоично-десятичного кода в двоичную систему счисления
4	Перевод 16-битного числа из двоичного вида в двоично-десятичный код
5	Сложение 24-битных чисел с выводом на индикацию
6	Вычитание 24-битных чисел с выводом на индикацию
7	Сложение 48-битных чисел
8	Вычитание 48-битных чисел
9	Сложение 16-битных чисел в двоично-десятичном коде с выводом на индикацию
10	Сложение 24-битных чисел в двоично-десятичном коде
11	Суммирование массива 8-битных чисел из N элементов (N – 8-битное число) с выводом на индикацию
12	Суммирование массива 16-битных чисел из N элементов (N – 8-битное число)
13	Подсчет количества нечетных 8-битных чисел в массиве из 16 элементов с выводом на индикацию
14	Подсчет количества нечетных 8-битных чисел в массиве из N элементов (N – 8-битное число)
15	Подсчет количества 8-битных чисел, кратных 8, в массиве из 16 элементов с выводом на индикацию
16	Подсчет количества 8-битных чисел, кратных 8, в массиве из N элементов (N – 8-битное число)
17	Подсчет количества 8-битных чисел в массиве из 16 элементов, попадающих в заданный диапазон, с
18	Подсчет количества 8-битных чисел в массиве из N элементов (N – 8-битное число), попадающих в за-
19	Определение диапазона значений 8-битных чисел в массиве из 16 элементов с выводом на индикацию

Контрольные вопросы.

1. Почему в системах управления в настоящее время применяются преимущественно цифровые методы обработки сигналов?
2. Какова должна быть минимальная длина командного слова микропроцессора для реализации 2500 различных команд?
3. Сколько машинных слов необходимо для представления в 8-битном микропроцессоре чисел в диапазоне от 1 до 1000000 (в формате с фиксированной запятой).
4. Сколько двоичных разрядов (без учета знака) содержит:
5. а) сумма двух целых n-битных чисел;
6. б) разность двух целых n-битных чисел;
7. в) произведение двух целых n-битных чисел?
8. Перечислить основные виды периферийных устройств микро-ЭВМ.
9. Сколько страниц прямо адресуемой памяти имеет микропроцессор с 20-битной шиной адреса при длине страницы памяти 4096 машинных слова?
10. По каким основным параметрам и качествам классифицируются ми-
11. кропроцессоры?
12. Какова должна быть разрядность регистра страницы памяти при использовании 16-битного микропроцессора с 16-разрядной шиной адреса и требуемом объеме памяти 1Мбайт?
13. Что входит в понятие архитектуры микропроцессора и микро-ЭВМ?
14. Как функционирует и для чего используется стековая память?

15. Опишите порядок выполнения команды микропроцессором.
16. Каковы основные различия между управляющими и персональными микро-ЭВМ?
17. Какова требуемая длина двоично-десятичного кода для представления чисел в диапазоне от 1 до 3000?
18. Какие основные операции выполняют арифметико-логические устройства микропроцессоров?
19. Определить значения признаков CY (перенос), AC (полуперенос), S (знак), Z (нуль) и P (четность) в микропроцессоре КР580ВМ80А после выполнения команды ADD B, если до ее выполнения A = 9BH, B = 36H.
20. Перечислить основные виды внутренних регистров микропроцессоров.
21. В какие ячейки памяти будет произведена запись информации и чему будет равно их содержимое, а также содержимое указателя стека SP после выполнения в микропроцессоре КР580ВМ80А команды PUSH B, если до ее выполнения B = 8AH, C = 15H, SP = 2304H?
22. Перечислите основные параметры БИС ЗУ, которые влияют на их выбор при организации блока памяти.
23. Какие машинные циклы реализуются в микропроцессоре КР580ВМ80А при выполнении команды SHLD ADR и какова длительность этой команды?

Практическая работа №26. 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).

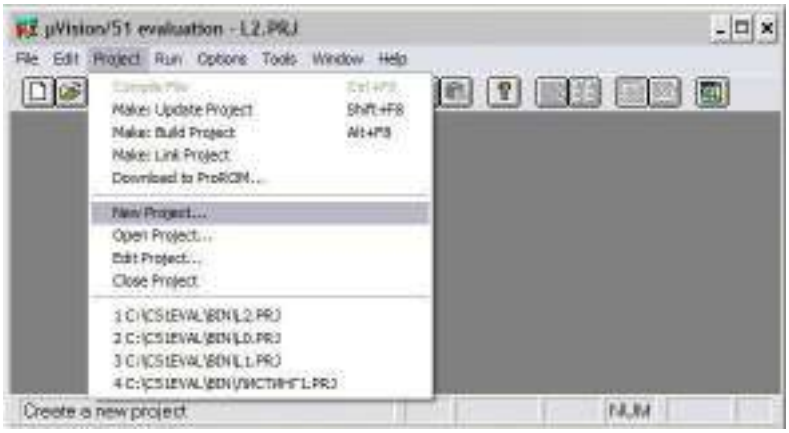


Рис. 26.3. Вкладка Project, создание нового проекта

После этого откроется меню создания нового проекта (рис. 26.4).

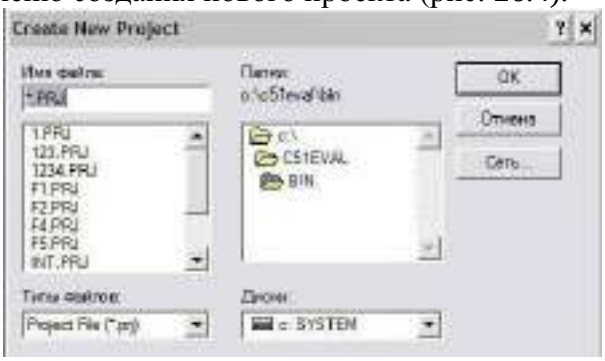


Рис. 26.4. Меню создания проекта Create New Project

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).

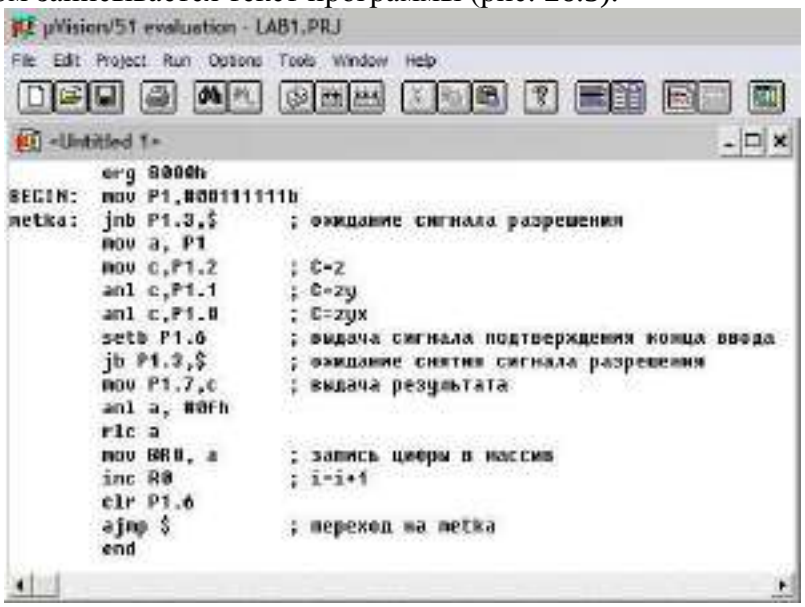


Рис. 26.5. Пример формирования программного кода

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).

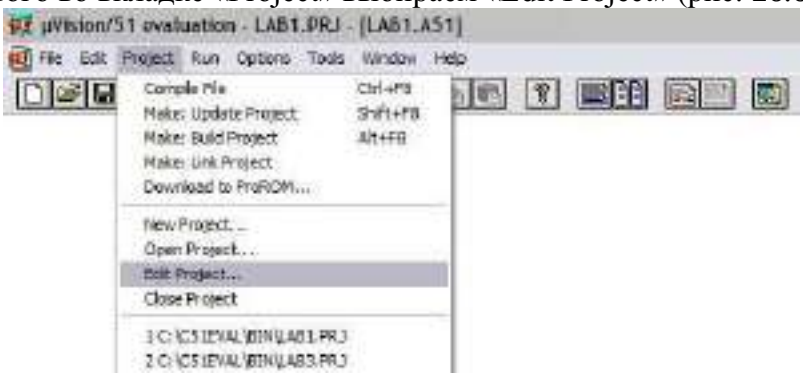


Рис. 26.6. Вкладка «Project», настройка проекта

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.

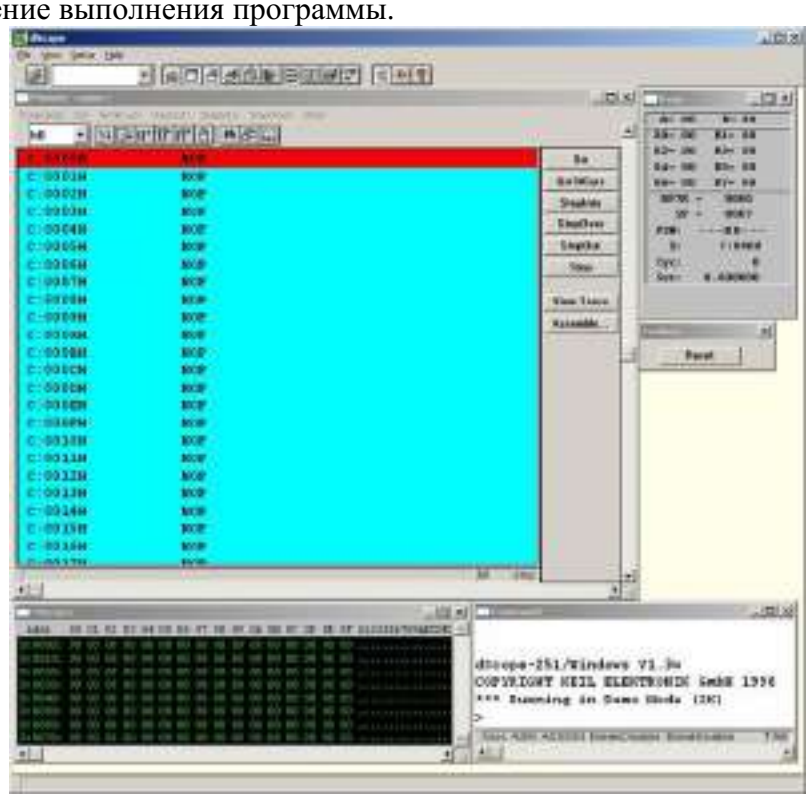


Рис. 26.8. Пользовательский интерфейс симулятора dScope

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 C-вход регистра-защелки 0
P1.1	CT1I C-вход регистра-защелки 1
P1.2	CT2I C-вход регистра защелки 2
P1.3	CT3I C-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).

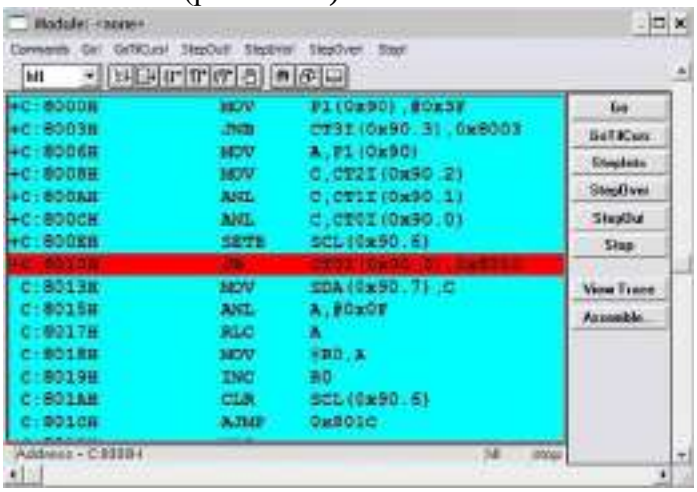


Рис. 26.12. Окно кода программы, ожидание снятия сигнала разрешения

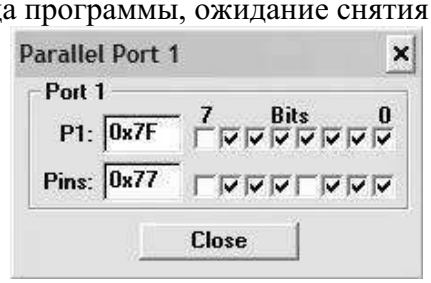


Рис. 26.13. Установка кода «0» на контакте 3 порта P1

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).

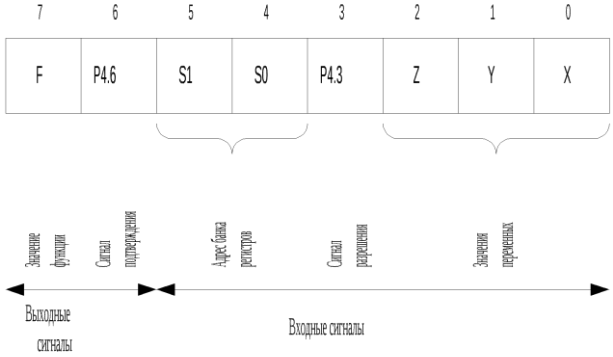


Рис. 26.14. Распределение разрядов порта P4 между входными и выходными сигналами

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).

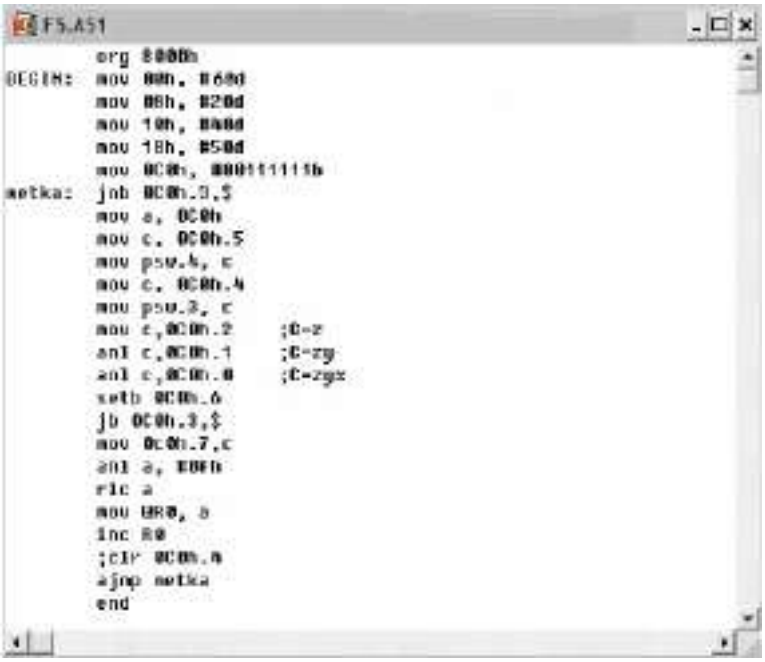


Рис. 26.15. Текст программы

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 №27. Тестирование и отладка микропроцессорных систем при разработке.

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 выполнения практической работы:

- загрузить проект в ПЛИС универсального лабораторного стенда;
- загрузить программу в память микропроцессорной системы (МПС) стенда;
- отладить работу спроектированного микропроцессорного устройства на УЛС с использованием управляющей программы MCS51;
- продемонстрировать работу отлаженной программы преподавателю;
- ответить на вопросы преподавателя.

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).

Таблица 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).

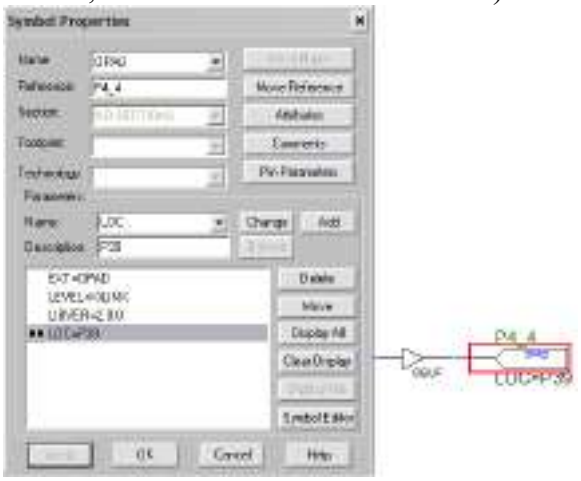


Рис. 27.1. Окно редактирования элемента OPAD

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.

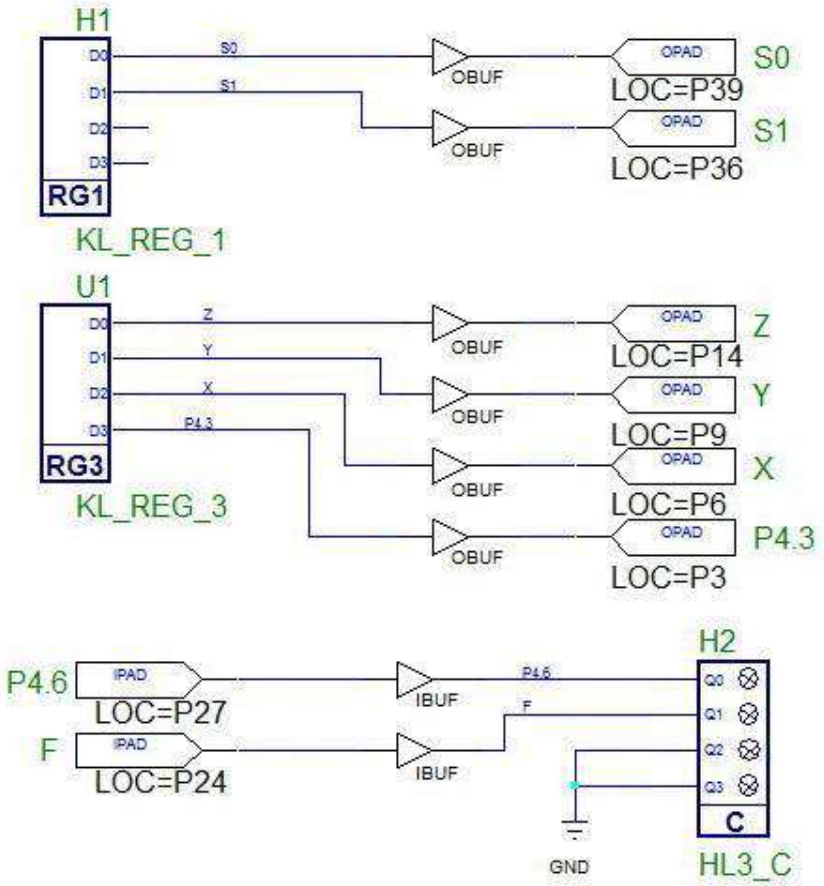


Рис. 27.2. Схема источника и приемника сигналов – проект, загружаемый в ПЛИС

Сигнал, соответствующий выходной функции, и сигнал подтверждения подключены к индикатору C. 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:

Откроется главное окно управляющей программы – MCS51 (рис. 27.3).

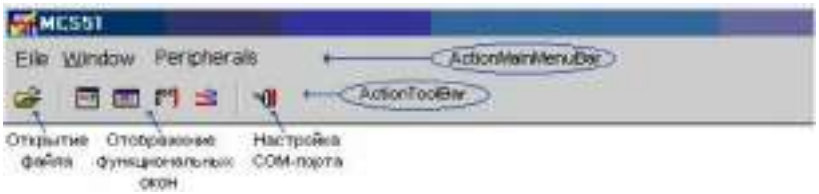


Рис. 27.3. Главное окно управляющей программы

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).

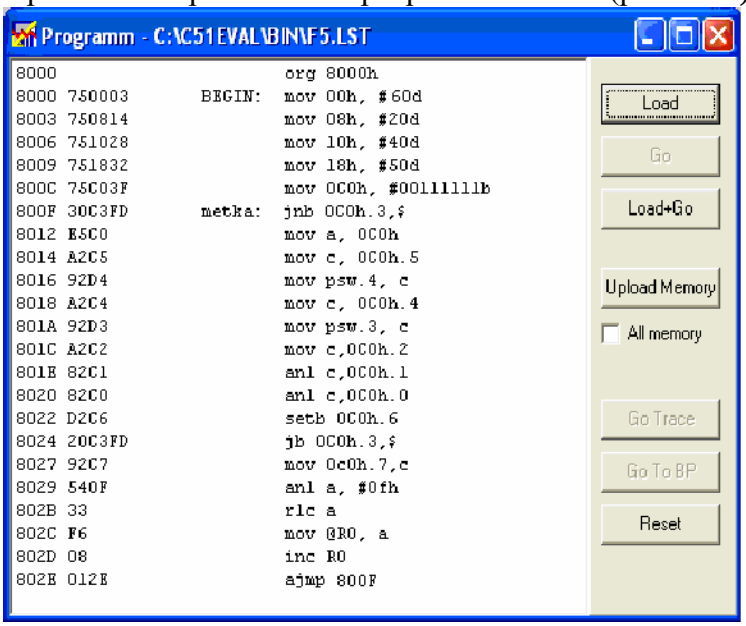


Рис. 27.4. Программное окно с загруженным кодом пользовательской программы

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

Номер варианта	Логическая функция	Уровни сигналов управления	
		разрешение	подтверждение
26	X&(^YvZ)	L	H
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 №28. Тестирование и отладка микропроцессорных систем при работе с МП.

Цель: изучение микропроцессорной системы для тестирования АЛУ.

Практическая работа предполагает создание на основе универсального лабораторного стенда микропроцессорной системы, выполняющей тестирование АЛУ из состава процессора тестирование АЛУ, модель которого написана на языке VHDL. МК в составе МПС должен формировать в определённом порядке операнды для операции умножения и короткой операции, подавать их на АЛУ, выдавать сигнал начала операции и после её окончания считывать результат операции, записывая его во внешнюю память или выдавая на индикацию.

Тестирование выполняется для каждой операции отдельно.

Перед тестированием необходимо составить программу, которая определяет результаты операций, заданных заданием на проектирование процессора, для всех возможных сочетаний операндов. Следует вспомнить, для каких чисел (дробных или целых) выполняются операции и в каком коде числа представлены.

Программа должна формировать текстовый файл, содержащий значения операндов, результатов и признаков (если их формирование предусмотрено) для каждой операции. Распечатка предоставляется для просмотра преподавателю и будет служить для проверки результатов тестирования.

Программа должна также позволять формировать тестовый файл (со значениями операндов, заданных вариантом) для записи его во внешнюю память МК, если это требуется вариантом задания.

Порядок выполнения практикума может быть следующим.

Определяется состав схем, размещаемых в ПЛИС, состав регистров и индикаторов, а также состав средств МПС, встроенных в УЛС и используемых для ввода или индикации данных. Параллельно должна быть определена структура программного обеспечения МПС, составлены алгоритмы работы программ МК.

Составляются и отлаживаются программы в среде Keil PK51 – Eval.

Затем следует в системе Xilinx составить проект, включающий в себя АЛУ со схемами подключения к элементам УЛС и МК. Необходимо взять проект всего разработанного ранее процессора, сохранить его под другим именем, удалить из него всё, кроме АЛУ, и на новом листе создать схему подключения.

После тестирования схемного АЛУ оно из проекта удаляется и на его место вставляется схема АЛУ, описанная на языке VHDL.

Пример подготовки к выполнению практической работы

Схемы подключения АЛУ к элементам стенда зависят от варианта задания. Общим в них является то, что обмен с МК может выполняться только через порт P4 и через системную шину. Выходы порта P4 соединяются с контактами ПЛИС, причем с теми же контактами, через которые могут выводиться сигналы на индикаторы A и B.

В качестве примера рассмотрим вариант тестирования некоторого АЛУ. Над целыми числами в обратном коде АЛУ выполняет операцию умножения (результат – 8-разрядный) и операцию сложения с формированием признаков: 0 – сумма равна нулю, 1 – сумма меньше нуля, 2 – сумма больше нуля, 3 – переполнение.

На ПЛИС размещается схема АЛУ, имеющая внешние входы операндов A и B, вход кода операции (KOP), вход сигнала начала операции (SNO), тактовый вход и вход начальной установки (RESET). Выходами АЛУ кроме результата операции и признаков выполнения операции сложения является также сигнал конца операции (SKO).

Порт P4 используем только для ввода-вывода битовых сигналов. Система будет иметь два режима работы: режим тестирования по периодическому анализу сигнала готовности (запуска) и режим автоматического тестирования.

МК должен выдавать на ПЛИС (АЛУ) сигнал SN0, реагировать на сигнал готовности и на сигнал SKO. Особенность создаваемой системы состоит в том, что и схема на ПЛИС, и сам МК – синхронные схемы, работающие от тактового сигнала. Хотя на УЛС обе части системы работают от одного генератора, но в ПЛИС длительность такта равна одному периоду сигнала C с генератора, а в МК частота генератора делится на 12 и длительность такта там равна 12 периодам сигнала C.

Сигнал SNO формируется, естественно, в МК битовыми командами setb и clr, и длительность его не может быть меньше одного такта синхронизации МК. А для АЛУ этот сигнал должен по длительности равняться одному такту C и быть активным в момент переднего фронта сигнала C. Для привязки сигнала SNO к таковым сигналам используется готовая схема, показанная на рис. 28.1.

При поступлении входного («длинного») сигнала SNOIN (P4x) от МК триггер T1, работающий как T-триггер, по заднему фронту C два раза изменяет свое состояние, формируя на своём выходе сигнал SNO длительностью в 1 такт. Триггер T2 устанавливается в «1» при наличии этого сигнала в момент заднего фронта C и далее запрещает прохождение входного сигнала через элемент D1 на триггер T1.

Возврат схемы в исходное состояние можно было бы осуществлять сигналом SKO через элемент D3. Однако АЛУ может выработать этот сигнал раньше, чем закончится входной сигнал, или же сигнал SKO на каком-то наборе операндов может не выработаться. Поэтому для сброса схемы следует формировать сигнал в микроконтроллере и выдавать его на ПЛИС через контакт порта P4. Для сброса можно было использовать и сигнал чтения результата АЛУ.

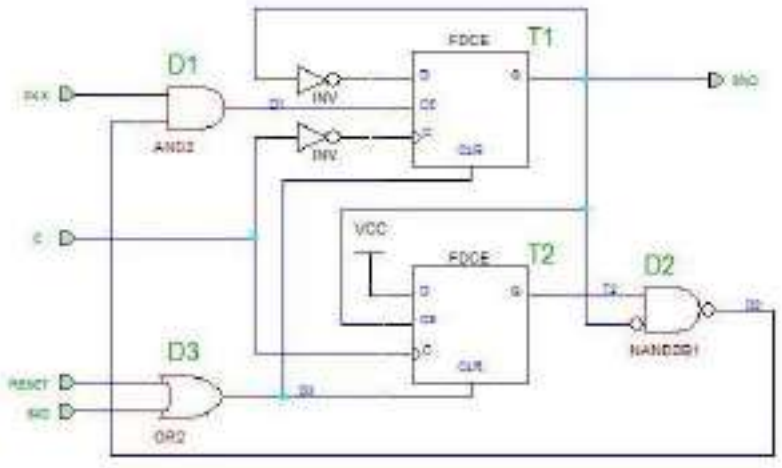


Рис. 28.1. Схема привязки сигнала SNO к тактовым сигналам

Результат моделирования схемы приведен на рис. 2.2.

Рис. 28.2. Результаты моделирования схемы привязки сигнала SNO

Схема привязки сигнала SNO может быть оформлена как подсхема и включена в проект.

Чтение результата выполнения операции в АЛУ микроконтроллером по шине должно выполняться после окончания работы АЛУ, т.е. после приёма сигнала SKO. Опрашивать его как сигнал готовности на контакте порта P4 микроконтроллер не может, так как он не успеет определить момент равенства сигнала «1» или же определить переход «0/1», а затем «1/0» вследствие малой длительности (1 такт) сигнала SKO.

Использовать сигнал SKO как сигнал запроса прерывания (при запуске по срезу сигнала) тоже невозможно, так как у сигнала запроса прерывания длительность состояний «1» и «0» должна быть более 1 такта МК (12 периодов С).

Можно воспользоваться тем, что длительность выполнения операций в АЛУ не превышает 20 тактов для любой операции и любых операндов. Поэтому можно сделать программную задержку на 2-3 такта работы микроконтроллера и потом считывать уже готовый результат. Ясно, что эта задержка может быть и большей.

Если необходимо использовать сигнал готовности (запуска), то к нему сказанное выше тоже применимо. Для реализации этого режима нужно каким-то сигналом (импульсом) устанавливать в «1» триггер готовности, который сбрасываться в «0» будет микроконтроллером.

Запуск-установку триггера в «1» проще всего выполнить сигналом с генератора, который работает в одиночном режиме – CLR.

Назначение разрядов порта P4 для рассматриваемого варианта показано в табл. 2.1.

Таблица 28.1

Назначение разрядов порта P4

Разряд	**P4.i**	**Назначение**	**Индикация**
0	1	Режим 1	A0
1	1	Режим 2	A1
2	0	–	A2
3	1	Сброс готовности	A3
4	0	Сброс SNO	B0
5	1	Готовность	B1
6	0	Окончание	B2
7	0	SNOIN	B3

Сигнал «Окончание» может быть использован при необходимости для индикации окончания выполнения микроконтроллером одного или заданного количества тестов. Значение сигнала «Режим 1», равное «1», соответствует режиму периодического анализа сигнала готовности, значение сигнала, равное «0», – автоматическому режиму. Сигнал «Режим 2» – резервный.

Не следует точно копировать это распределение сигналов между разрядами порта P4. Некоторые из сигналов могут не понадобиться, некоторые придется ввести дополнительно.

За исключением порта P4 внешние устройства, входящие в состав УЛС, и дополнительные (требуемые вариантом задания) внешние устройства размещаются в адресном пространстве внешней памяти МК. Такие внешние устройства называются «отображёнными на память». Для внешних устройств, входящих в состав УЛС и подключённых к шине, в адресном пространстве выделен ряд отдельных адресов, а для дополнительных внешних устройств, размещаемых на ПЛИС, отведено всего два адреса – 7FFAh и 7FFBh. На ПЛИС выведены только сигналы дешифрации этих адресов – X7FFA (контакт P70) и X7FFB (контакт P68) и сигналы записи или чтения по этим адресам – MKWR (контакт P69) и MKRD (контакт P67). Все эти сигналы имеют нулевые активные уровни.

Данные между системной шиной и ПЛИС передаются через восемь внешних контактов (табл. 28.2).

Таблица 28.2

Соответствие разрядов системной шины контактам ПЛИС

Контакты шины	**Контакты ПЛИС**
0	84
1	83
2	79
3	81
4	78
5	77
6	72
7	80

Схема части проекта (без АЛУ), реализованная на ПЛИС, показана на рис. 28.3. Для выбранного варианта средств МПС, встроенных в УЛС (клавиатуры, семисегментных индикаторов и дисплея на жидкокристаллических индикаторах – ЖКИ), не требуется. В качестве внешних устройств – приемников данных в ПЛИС необходимы два 4-разрядных буферных регистра RG1 и RG2 на входах A и B АЛУ. Они необходимы, так как данные на шине не хранятся. Запись кодов операндов в них может производиться одновременно 8-разрядным словом при обращении по адресу 7FFAh. Так как в тестируемом АЛУ у операндов разряд с индексом 0 – знаковый, то при подключении регистров к шинам A и B разряды приходится менять местами. В качестве внешнего устройства-источника возьмём только регистр результата без признаков. Чтение результата выполняется по адресу 7FFB.

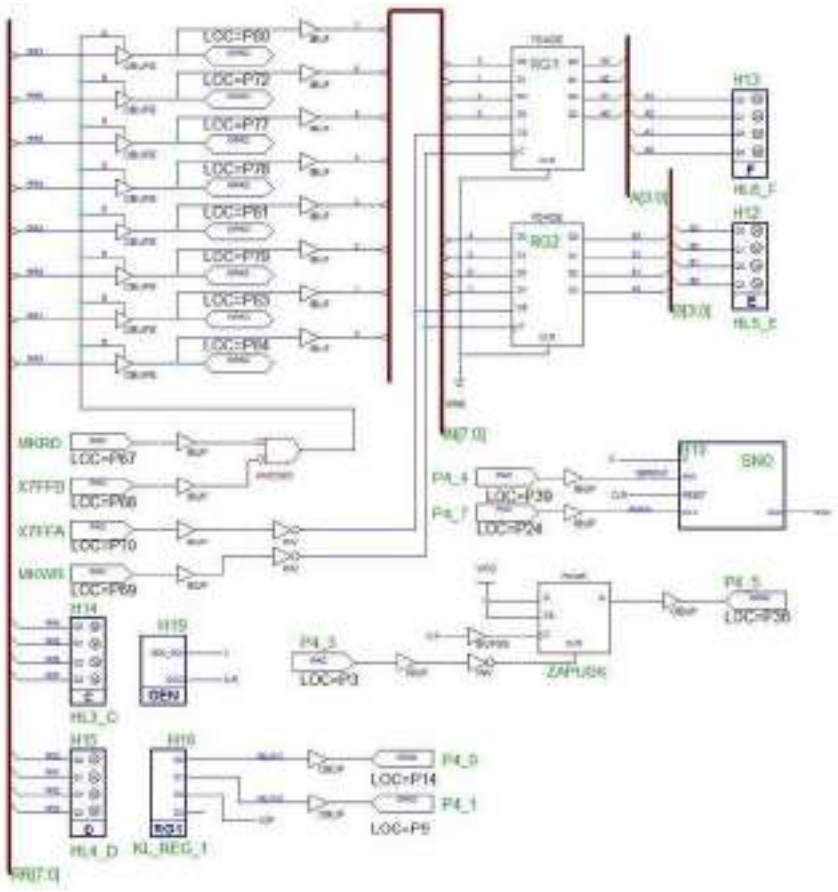


Рис .28.3. Схема подключения АЛУ к МК

Для подключения регистра результата к шине использованы буферные элементы с «третьим» состоянием выхода – OBUFE. Так как обмен данными по шине осуществляется в двух направлениях (от ПЛИС у МК и от МК к ПЛИС), используются двунаправленные входы-выходы – IOPAD.

В этом примере индикация операндов A и B выполняется на индикаторах E и F, результата – на индикаторах C и D. Индикатор окончания тестирования не нужен, так как этот сигнал выдаётся через контакт P4.6 и будет виден в разряде 1 индикатора B.

Далее дан пример программы тестирования АЛУ. Программа работает с внешней памятью, в которой записаны (в режиме редактирования внешней памяти данных или переписаны из файла) наборы операндов и заранее вычисленные результаты. Количество таких наборов – N. Величина N задается в программе.

В одиночном режиме («Режим 1» = 1) по сигналу готовности микроконтроллер читает пару операндов из памяти, подает их на АЛУ и результат записывает в ячейку памяти, соседнюю с ранее вычисленным результатом. Далее величина N уменьшается на 1 и МК вновь ожидает сигнала готовности. После перебора всех N наборов зажигается индикатор «Окончание», и программа зацикливается на одной команде.

В автоматическом режиме сигнал готовности опрашивается микроконтроллером только один раз. Далее после записи результата в память МК сразу переходит к чтению новой пары операндов. Пример загрузки памяти (для Nmax= 12) для операции умножения приведен в табл. 28.3.

Для каждого набора в первой ячейке памяти указаны значения операндов A и B, во второй – результат операции умножения, в следующей – результат второй операции. Значком «*» в таблице отмечены ячейки, в которые записывается результат той или иной операции.

Таблица 28.3

Таблица загрузки внешней памяти

Адрес	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
8000h	22	04	04	*	24	08	06	*	66	24	OC	*	77	21	0E	*
8010h	25	0F	07	*	62	0C	08	*	54	14	09	*	65	1E	0B	*
8020h	DD	04	FB	*	DB	08	F9	*	99	24	F3	*	88	31	F1	*

Четвертый набор – 7x7 имеет результатом 31h, в памяти записан ошибочный результат.

В последних четырех наборах записаны те же операнды, что и в первых четырёх наборах, но с другим знаком (в рассматриваемом варианте АЛУ числа представляются в обратном коде).

Блок-схема программы показана на рис. 28.4, текст программы – на рис. 28.5.

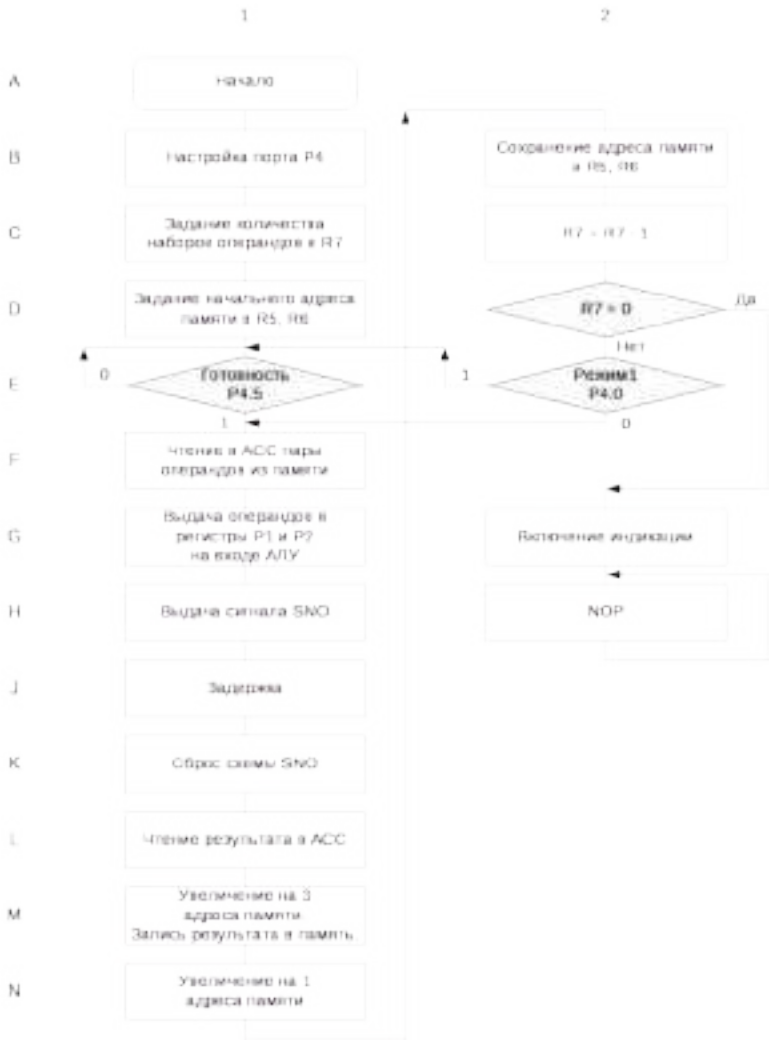


Рис. 28.4. Блок-схема программы

```
ALY3.A51
ORG 8000h
        MOV IE, #81h
        MOV IP, #01h
        SETB IT0
        MOV 0C0h, #00010101b    ; настройка порта P4
        MOV R7, #0Ch            ; задание N
        MOV R5, #00h            ; задание начального адреса памяти,
        NOP
        MOV R6, #8000h          ; равного 8000h
L1:     JNB 0C0h.5, $           ; ожидание готовности
        CLR 0C0h.3              ; сброс триггера готовности
        SETB 0C0h.3
L3:     MOV DPH, R6             ; загрузка в DPTR адреса ячейки памяти
        MOV DPL, R5
        MOVX A, @DPTR           ; ACC = 8,A
        MOV DPTR, #7FFAh        ; адрес записи в P1 и P2
        MOVX @DPTR, A           ; выдача операндов
        SETB 0C0h.7             ; выдача SMD
        CLR 0C0h.7
        NOP                     ; задержка
        NOP
        SETB 0C0h.4             ; сброс схемы SMD
        CLR 0C0h.4
        MOV DPTR, #7FFBh        ; адрес чтения результата
        MOVX A, @DPTR           ; ACC - результат
        MOV DPH, R6             ; загрузка в DPTR адреса ячейки памяти
        MOV DPL, R5
        INC DPTR                ; подготовка адреса результата
        INC DPTR
        INC DPTR
        NOP
        MOVX @DPTR, A           ; запись результата в память
        MOV A, R7               ; ACC = B A
        INC DPTR                ; увеличение адреса памяти на 1
        MOV R6, DPH             ; сохранение адреса памяти
        MOV R5, DPL
        DJNZ R7, L2             ; R-1, если N не равно 0, то анализ режима
        SETB 0C0h.6             ; если N-0, то включение индикатора
        AJMP $                  ; зацикливание
L2:     JB 0C0h.0, L1           ; анализ режима. Если «1», то ожидание готовности
        AJMP L3                 ; если «0», то чтение новых A и B
        END
```

Рис. 2.5. Текст программы для тестирования АЛУ

Практическая работа №29. Тестирование и отладка микропроцессорных систем при разработке программного обеспечения.

Цель: изучение микропроцессорной системы для тестирования АЛУ.

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 OBUFTс 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.

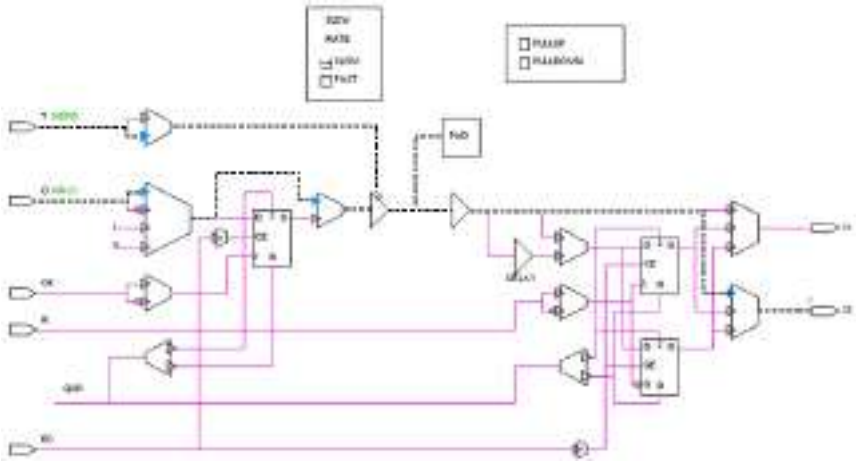


Рис. 29.1. Структура блока ввода-вывода в ПЛИС УЛС

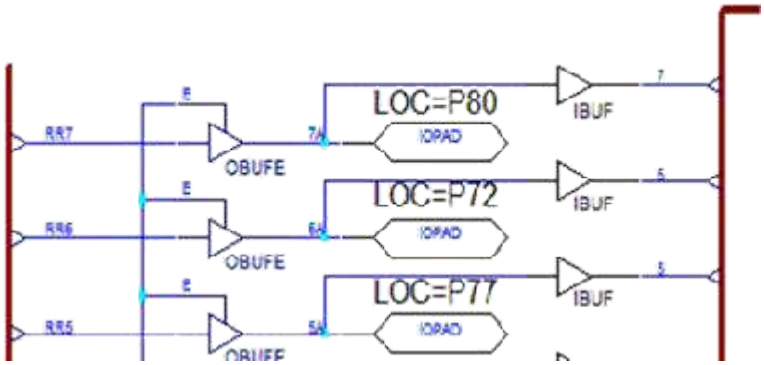


Рис. 29.2. Фрагмент схемы подключения к шине

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.

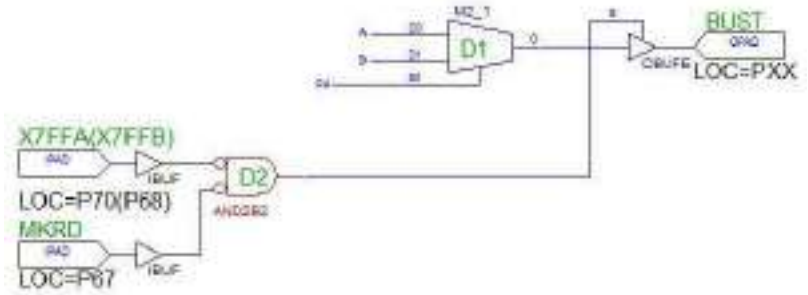


Рис. 29.3. Вариант подключения двух источников через мультиплексор (управление от порта P4)

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).

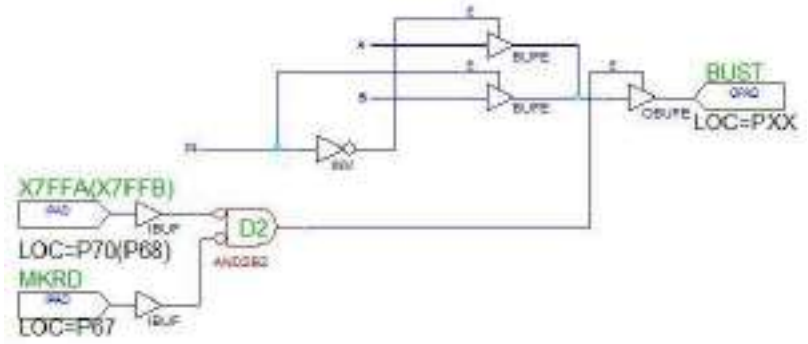


Рис. 29.4. Вариант подключения двух источников через буферные элементы (управление от порта P4)

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.

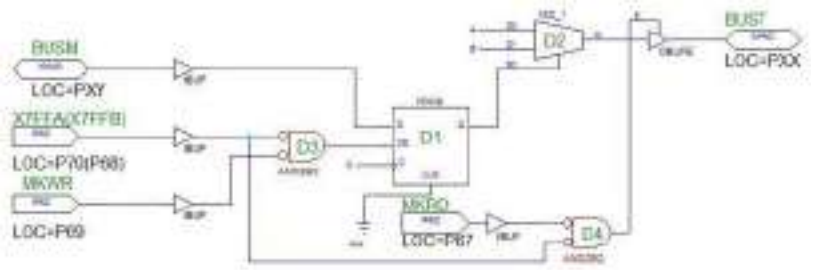


Рис. 29.5. Вариант подключения двух источников через мультиплексор

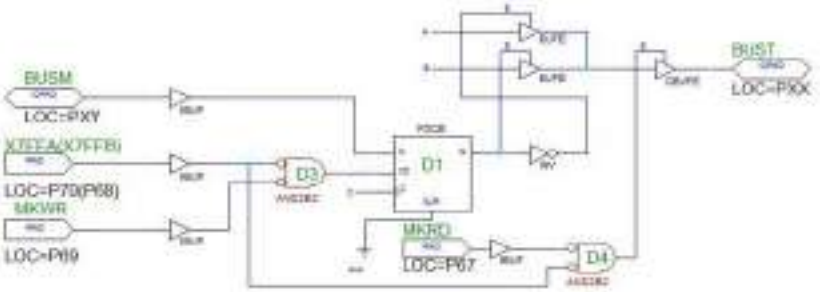


Рис. 29.6. Вариант подключения двух источников через буферные элементы

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.

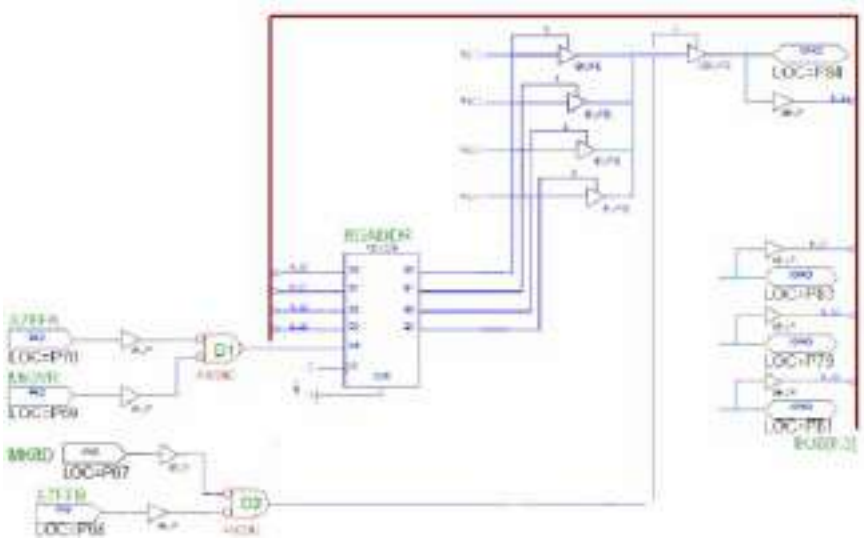


Рис. 29.7. Схема подключения к шине нескольких внешних устройств – источников данных (унитарное кодирование адреса)

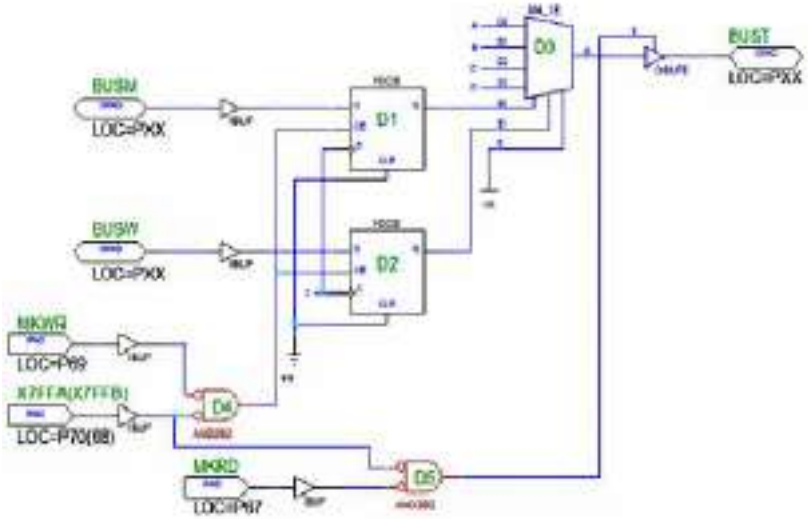


Рис. 29.8. Вариант подключения четырёх источников через мультиплексор (управление от регистра адреса)

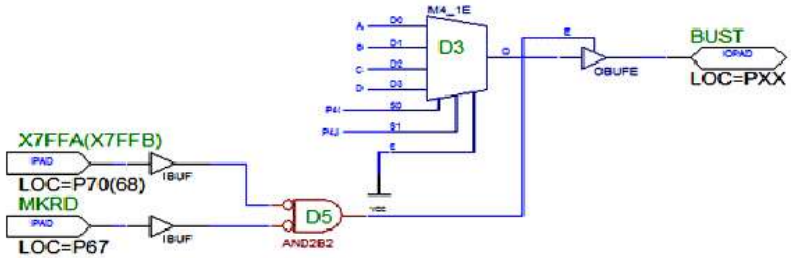


Рис. 29.9. Вариант подключения четырёх источников через мультиплексор (управление от порта P4)

Для схемы с использованием буферных элементов потребуется дешифратор (D3) для преобразования двоичного кода в унитарный (рис. 29.15).

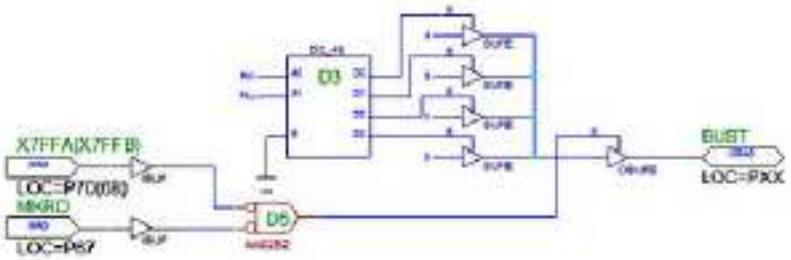


Рис. 29.10. Вариант подключения четырёх источников через буферные элементы (управление от порта P4)

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).

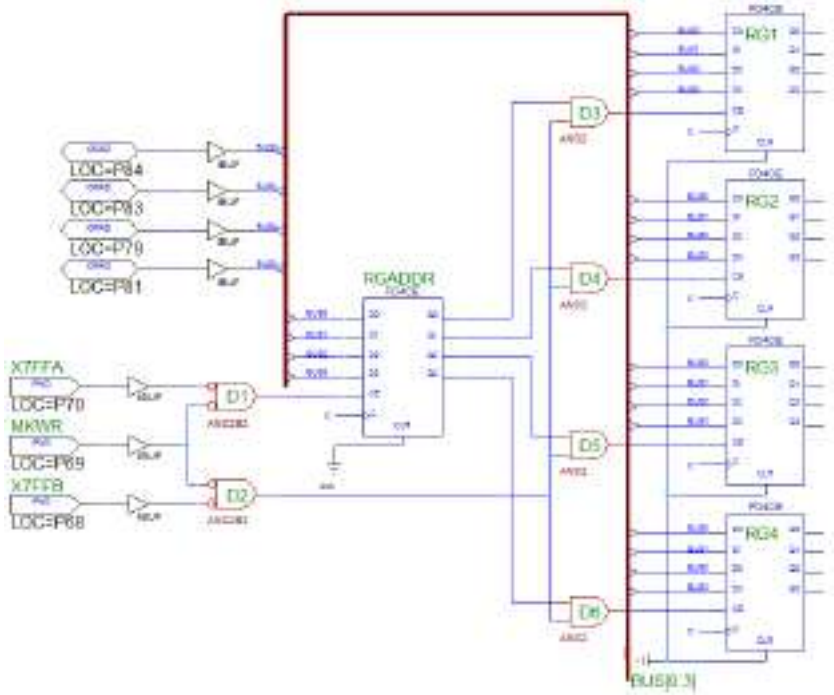


Рис. 29.11. Схема подключения к шине нескольких внешних устройств – приёмников данных (унитарное кодирование адреса)

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.

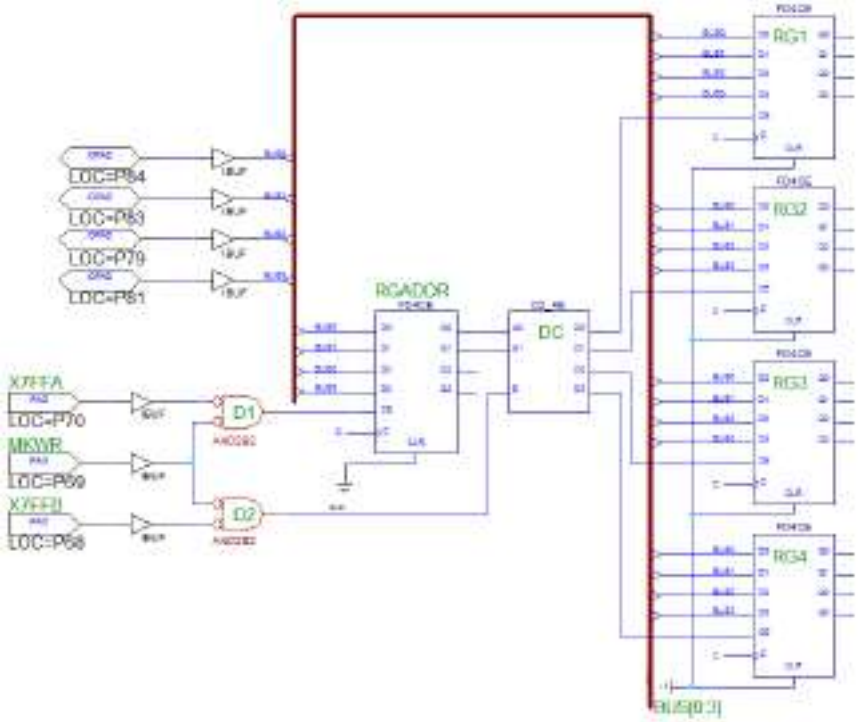


Рис. 29.12. Схема подключения к шине нескольких внешних устройств – приёмников данных (двоичное кодирование адреса)

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.

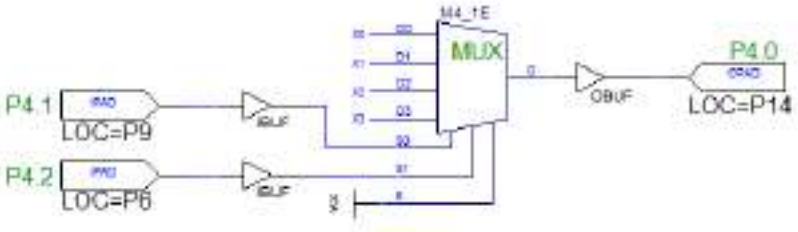


Рис. 29.13. Схема подключения к порту P4 нескольких битовых сигналов

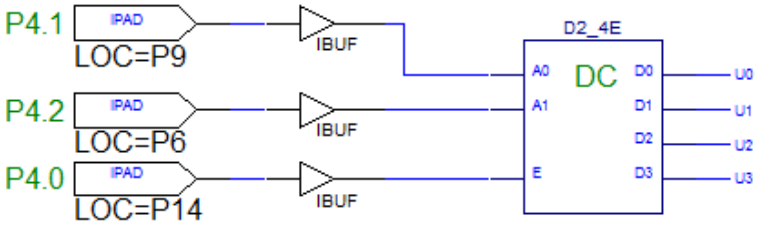


Рис. 29.14. Схема ввода в порт P4 нескольких битовых сигналов

Практическая работа №30. Тестирование и отладка микропроцессорных систем (АЛУ).

Цель: изучение последовательности и способов отладки микропроцессорной системы для тестирования АЛУ.

Теоретическая часть.

Тестирование АЛУ выполняется в двух режимах:

- одиночном при каждом сигнале «Готовность»;
- непрерывном.

И в том, и в другом режиме можно было бы запустить программу и увидеть конечный результат. Но это почти никогда не получается из-за допущенных ошибок (или не работает программа, или АЛУ выдаёт не тот результат).

Ошибки могут возникнуть при проектировании АЛУ (их можно было не заметить ранее из-за тестирования на малом наборе операндов). При их обнаружении придётся исправлять схему АЛУ или её описание на языке VHDL. Для схемного варианта АЛУ можно вывести на индикацию состояние каких-то регистров. Но выполнить проверку работы АЛУ по тактам невозможно.

Ошибки могут быть допущены в программе работы МК, как синтаксические, так и связанные с неверными вычислениями и с неверным формированием сигналов, выдаваемых на внешние устройства.

Программные ошибки, прежде всего, нужно искать с использованием программного симулятор dScope, который для этого и предназначен. При этом обнаруживаются синтаксические ошибки, можно наблюдать состояние внутренних регистров МК, но невозможно видеть, что и в какие внешние устройства отправлено и невозможно работать с реальными данными с внешних устройств.

Возможны и схемные ошибки, например такие:

- неправильно составлена схема подключения АЛУ и регистров УЛС к системной шине;
- перепутаны разряды шины;
- перепутаны разряды операндов A и B;
- не с тех разрядов регистра результата взят результат для передачи в МК.

За ошибку иногда принимаются случаи несовпадения результата операции над дробными числами с результатом, вычисленным при помощи другой программы (сказываются погрешности округлений), а также неправильная интерпретация результатов тестирования, когда студенты забывают о том, с какими числами (дробными или целыми) работает АЛУ.

Иногда ошибочная работа системы МПС–АЛУ вызывается неправильным формированием в программе управляющих сигналов – заданием перепадов уровней (0/1 или 1/0) вместо задания импульса (0/1/0 или 1/0/1).

Для выявления этих ошибок и определения места неисправности нужно убедиться, что:

- операнды выданы в буферные регистры на входе АЛУ верно;
- операнды записались в регистры АЛУ правильно;
- результат получается в регистре результата в нужных разрядах;
- выполняется именно та операция, которая задана на пульте УЛС;
- выполняется программа обработки прерываний.

Это невозможно проверить с помощью программы dScope.

Единственным выходом является отладка МПС на стенде с использованием управляющей программы MCS51. Можно использовать следующие режимы выполнения программы: «Основной» и «Останов в КТ» (КТ – контрольная точка).

Режим «Основной» используется в тех случаях, когда необходимо наблюдать только конечный результат работы программы на индикаторах стенда.

Отладка МПС, реализующей программу тестирования АЛУ в ПЛИС, должна выполняться в режиме «Останов в КТ», где можно видеть пошаговое исполнение программы (останов в заданной КТ), наблюдать изменение состояния регистров, аккумулятора, а самое главное отслеживать запись операндов и результата по конкретному адресу. Содержимое внутренней, внешней памяти, а также регистров выводится в соответствующие окна.

Для работы в режиме «Останов в КТ» пользовательская программа должна быть определенным образом подготовлена.

- Необходимо перед каждой командой, которую пользователь хочет отметить как точку контроля, установить пустую команду (NOP). (Естественно, что это ведёт к увеличению объёма программы и изменению времени её работы.)

- В теле программы нужно добавить команды, разрешающие прерывание от внешнего источника (Int 0) и устанавливающие для этого запроса высший приоритет, с помощью вставки в программу следующего кода:

```
mov ie, #81h
mov ip, #01h
setb it0
```

- Для доступа к данному режиму необходимо установить не менее двух контрольных точек.

- Для удобства работы следует отметить контрольные точки на блок-схеме алгоритма.

Отметим одну особенность системы отладки. Она использует таймер Т1 МК, и настройки этого таймера нельзя менять при работе с отладчиком в режиме КТ. В программе МК можно использовать лишь таймер Т0. Как только запускается монитор отладки, он настраивает режим работы таймера Т1 с помощью регистра управления TMOD. Изменить состояния этого регистра путём загрузки в него только значений разрядов, относящихся к таймеру Т0, нельзя. Нужно считать код регистра TMOD в аккумулятор А, логически сложить с требуемым кодом режима таймера Т0, потом отправить содержимое аккумулятора А в регистр.

Далее необходимо выполнить загрузку программы в УЛС (Load) и установить КТ, дважды кликнув левой кнопкой мыши на строке в окне «Program Window». Появление в выделенной строке символов «ВР» можно наблюдать на рис. 30.1. Следует помнить, что точкой останова программы является именно комбинация вставленной команды NOP и установленной КТ. Одна команда NOP не может быть контрольной точкой.

В данном режиме доступны следующие функции отладки:

- загрузка пользовательской программы (Load);
- запуск программы, в режиме «Останов в КТ» (Go To ВР);
- переход к следующей КТ (Next ВР).

Запуск программы осуществляется нажатием кнопки «Go To ВР». Выполнение программы прерывается после выполнения команды, помеченной как точка контроля. Эта команда выделяется в программном окне (рис. 30.2). После этого пользователю становятся доступны все функции работы с внутренней и внешней памятью данных, а также с памятью регистров.

Как видно из рис. 30.2, после останова в КТ появляется кнопка Next ВР. После нажатия данной кнопки выполнение пользовательской программы будет продолжено до очередной КТ. Содержимое памяти регистров обновляется в каждой КТ.

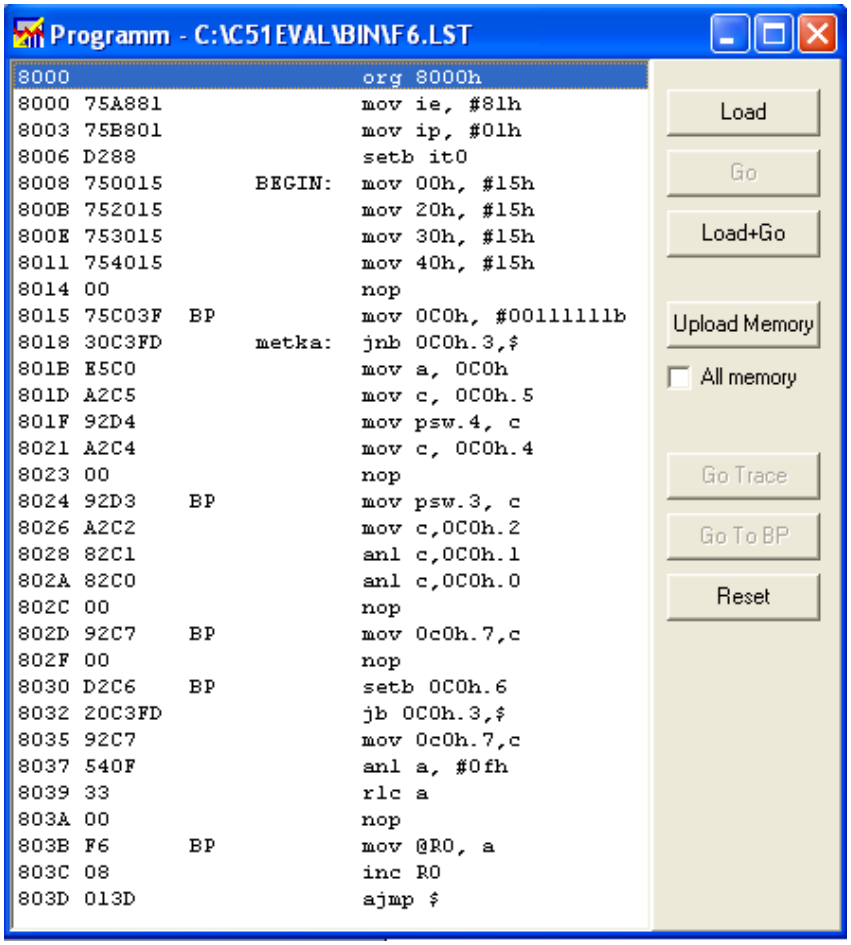


Рис. 30.1. Программное окно с установленными КТ

Использование индикаторов стенда при отладке программы и собственно тестировании АЛУ необязательно, однако их подключение в том или ином виде позволяет убедиться, что операнды выданы в буферные регистры на входе АЛУ верно, операнды переписаны в регистры АЛУ правильно, результат получается в регистре результата в нужных разрядах. При тестировании схемного АЛУ индикаторы можно подключать и к внутренним регистрам блока операций или устройства управления.

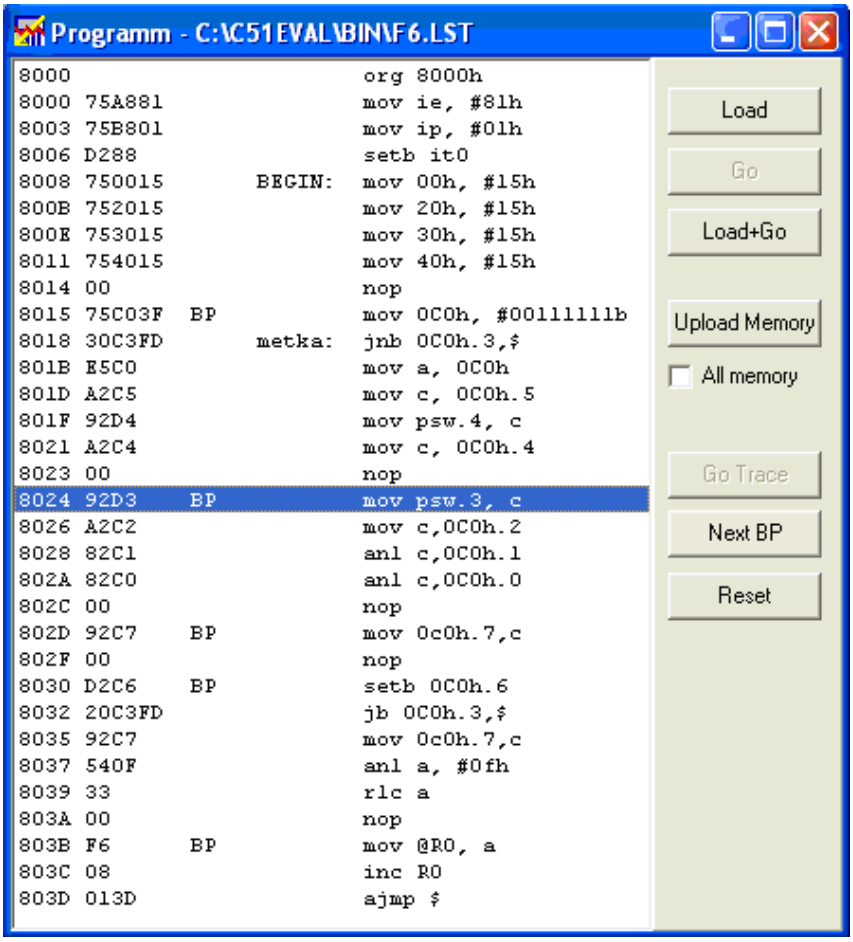


Рис. 30.2. Программное окно при останове в КТ, вызвавшей прерывание

Можно программу разбить на отдельные участки с номерами 1, 2,… и при входе в каждый участок выводить на индикацию (либо светодиод, либо 7-сегментгный индикатор, либо ЖКИ) номер участка.

Отлаживать программу не совсем просто из-за того, что постоянно изменяются комбинации операндов. Поэтому можно кроме основной программы создать еще несколько тестовых. Это могут быть программы:

- выполнения операций над выбранной парой операндов несколько раз;
- повторения какого-то участка основной программы;
- реализации некоторого упрощенного алгоритма;
- циклического выполнения некоторой программы;
- выполнения иных программ (вывод на индикаторы или ЖКИ).

Можно загружать каждую программу в память МПС по очереди, но это только удлиняет процесс тестирования. Лучше сразу все тестовые программы учесть в одной программе, в которой ввести команды анализа режима работы (тестирования).

Приведём возможные способы задания нескольких режимов работы системы и её отладки:

- задание режимов на регистре УЛС с вводом кода режима через порт P4;
- задание режимов на регистре УЛС с вводом кода режима через системную шину;
- иной способ.

Ввод через P4 маловероятен из-за малого количества разрядов, ввод через системную шину приводит к усложнению схемы подключения к системной шине.

Рассмотрим вариант с использованием клавиатуры. Она уже подключена к системной шине, нажатие одной из цифр вызывает прерывание, после чего обработчик прерывания считывает и распознаёт скан-код нажатой кнопки.

При использовании клавиатуры могут быть различные способы решения. Один из них может быть таким.

Используем какой-либо флаг, например F0, для определения порядка работы программы. Исходное состояние этого флага или состояние после набора какого-то символа, например, «0»: F0=0.

Если при чтении иного скан-кода обработчик обнаруживает, что F0=0, то происходит переход на участок программы ввода параметра, необходимого для выполнения основной программы (кода времени или начального значения операнда). После нажатия символа «F» флаг F0 устанавливается равным «1». В таком случае при чтении иного скан-кода обработчик переходит на участки программы, соответствующие различным тестам в зависимости от введенного символа. Заметим, что коды «0» и «F» вряд ли понадобятся при задании кода времени или начального значения операнда В. Такое решение использования прерывания от клавиатуры для выбора режима тестирования представляет собой подобие разделяемого прерывания (shared interrupts).

Практическая часть.

Рассмотрим пример программы обработки прерывания от клавиатуры при допущениях.

В спроектированной МПС дополнительными внешними устройствами-приемниками информации, подключёнными к системной шине, являются:

- регистры операндов А и В (для записи в них отведён адрес 7FFAh);
- регистр адреса внешних устройств – приёмников информации (для записи в него отведён адрес 7FFBh).

К шине внутри ПЛИС подключено четыре источника:

- регистр результата – код в регистре адреса 00h;
- регистры признаков – 01h;
- ПЗУ знакогенератора – 02h;
- регистр УЛС для задания операнда – 03h.

Для чтения с шины внутри ПЛИС выделен адрес 7FFBh.

Для формирования сигнала SNOIN используется разряд порта P4.7, для сброса – P4.4.

При инициализации МК флаг F0 устанавливается в «0».

При выполнении основной программы тестирования АЛУ клавиатура используется для ввода величины Т (период мигания индикаторов) в регистр R7.

Для запуска тестовых программ используется нажатие следующих цифр на клавиатуре:

«1» – программа ввода операнда А с регистра УЛС в регистр R3;

«2» – программа ввода операнда В с регистра УЛС в регистр R4;

«3» – программа отображения кода в А на четырёх 7 сегментных индикаторах (ввод слева) при реализации знакогенератора во внутренней памяти;

«4» – умножение операнда в R3 на операнд в R4 (отображение операндов и результата на подключённых шкалах индикаторов).

Текст программы обработки прерывания от клавиатуры

```
ORG8013h;обработчик прерыванияINT1
MOVDPTR,#7FFFh
MOVA,#40h
MOVX@DPTR,A;          разрешение чтения FIFO клавиатуры
JBF0,LAB1 ;           проверка режима работы клавиатуры
MOVDPTR, #7FFEh;      клавиатура - для ввода Т
MOVXA,@DPTR;          чтение скан-кода
CJNEA, #00h,KK1;      проверка на «0»
LJMP EXIT
KK1: CJNE A, #01h, KK2;       проверка на «1»
MOVR7,#01h;                   вывод вR7 кода «1»
LJMP EXIT
KK2: CJNE A, #02h, KK3;       проверка на «2»
MOVR7,#02h;                   вывод вR7 кода «2»
LJMP EXIT
```

```
KK3: CJNE A, #03h, KK4;          проверка на «3»
MOVR7,#03h;                      вывод вR7 кода «3»
LJMP EXIT
KK4: CJNE A, #04h, KK5 ;         проверка на «4»
MOVR7,#04h;                      вывод вR7 кода «4»
LJMP EXIT
KK5: CJNE A, #05h, KK6;          проверка на «5»
MOVR7,#05h;                      вывод вR7 кода «5»
LJMP EXIT
KK6: CJNE A, #06h, KK7;          проверка на «6»
MOVR7,#06h;                      вывод вR7 кода «6»
LJMP EXIT
KK7: CJNE A, #07h, KK8;          проверка на «7»
MOVR7,#07h;                      вывод вR7 кода «7»
LJMP EXIT
KK8: CJNE A, #08h, KK9 ;         проверка на «8»
MOVR7,#08h;                      вывод вR7 кода «8»
LJMP EXIT
KK9: CJNE A, #09h, KKA ;         проверка на «9»
MOVR7,#09h;                      вывод вR7 кода «9»
LJMPEXIT
KKA:CJNEA, #0Ah,KKB;             проверка на «A»
MOVR7,#0Ah;                      вывод вR7 кода «A»
LJMP EXIT
KKB: CJNE A, #0Bh, KKC ;         проверка на «B»
MOVR7,#0Dh;                      вывод вR7 кода «B»
KKC: CJNE A, #0Ch, KKD ;         проверка на «C»
MOVR7,#0Ch;                      вывод вR7 кода «C»
LJMP EXIT
KKD: CJNE A, #0Dh, KKE ;         проверка на «D»
MOVR7,#0Dh;                      вывод вR7 кода «D»
LJMP EXIT
KKE: CJNE A, #0Eh, KKF ;         проверка на «E»
MOVR7,#0Eh;                      вывод вR7 кода «E»
LJMP EXIT
KKF: CJNE A, #0Fh, EXIT ;        проверка на «F»
SETBF0 ;                         установка признака задания тестаF0=1
LJMP EXIT
                                 ; Установлен режим задания теста
LAB1:CJNEA, #00h,KKK1 ;          проверка на «0»
CLRF0 ;установка режима ввода теста
LJMPEXIT
KKK1:CJNEA, #01h,KKK2 ;          проверка на «1»
LJMPPART1
KKK2:CJNEA, #02h,KKK3 ;          проверка на «2»
LJMPPART2
KKK3:CJNEA, #03h,KKK4 ;          проверка на «3»
LJMPPART3
KKK4:CJNEA, #04,EXIT;    проверка на «4»
                                 ;Реализация программы 4
MOV A, R3
SWAPA
ORLA,R4 ; ACC = A B
```

```
MOV DPTR, #7FFAh ;          адрес записи в P1 и P2
MOVX @DPTR, A ;             выдача операндов
SETB 0c0h.7 ;               выдача SNOIN
CLR 0c0h.7
NOP ;                       задержка
NOP
SETB 0c0h.4 ;               сброс схемы SNO
CLR0c0h.4
LJMPEXIT
                            ; Реализация программы 1
PART1: MOV DPTR, #7FFBh ;   адрес записи в регистр адреса
MOVA, #03h;                 адрес регистра УЛС
MOVX @DPTR, A ;             запись адреса в регистр адреса
MOVX A, @DPTR ;             чтение операнда A
MOVR3,A
LJMPEXIT
                            ; Реализация программы 2
PART2: MOV DPTR, #7FFBh ;   адрес записи в регистр адреса
MOVA, #03h;                 адрес регистра УЛС
MOVX @DPTR, A ;             запись адреса в регистр адреса
MOVX A, @DPTR ;             чтение операнда B
MOVR4,A
LJMPEXIT
                            ; Реализация программы 3
PART3:MOVR7,A               ;сохранение значения A
MOV DPTR,#7FFFh
MOV A,#01h
MOVX@DPTR,A                 ;ввод символов слева
MOVDPTR,#7FFFh
MOVA,#90h                   ;автоинкрементирование адреса
MOVX @DPTR,A
MOV 70h, #0F3h
MOV 71h, #60h
MOV 72h, #0B5h
MOV 73h, #0F4h
MOV 74h, #66h
MOV 75h, #0D6h
MOV 76h, #0D7h
MOV 77h, #70h
MOV 78h, #0F7h               ;запись в память кодов символов
MOV 79h, #0F6h
MOV 7Ah, #77h
MOV 7Bh, #0C7h
MOV 7Ch, #93h
MOV 7Dh, #0E5h
MOV 7Eh, #097h
MOV 7Fh, #17h
MOV DPTR,#7FFEh
MOV A,#00h
MOVX @DPTR,A
MOVX @DPTR,A                ;гашение индикаторов
MOVX@DPTR,A
MOVX@DPTR,A
```

```
MOVA,R7                 ;возврат A
                        ;Вывод A в крайнюю левую ячейку дисплея
CallVivod
                        ;Вывод A во вторую слева ячейку
CallVivod
CallVivod
                        ;Вывод A в две правые ячейки
Call Vivod
Vivod:
MOV DPTR,#7FFEh
ADD A, #70h
MOV R0, A
MOVA, @R0
MOVX @DPTR, A           ;вывод цифры на индикацию
RET
EXIT:RETI
END
```

Оформление отчета о проделанной работе

Отчет о работе должен содержать:

1) техническое задание;

2) анализ задания;

3) программу получения тестовых значений (с комментариями), распечатку текстовых значений;

4) оценку количества внешних устройств и их назначение;

5) схему подключения внешних устройств к МК через порт P4 и системную шину;

6) структуру программного обеспечения МПС;

7) блок-схемы алгоритмов программ;

8) тексты программ (с комментариями);

9) результаты тестирования и анализ результатов (методику тестирования, копии экранов с содержимым внешней памяти, анализ допущенных ошибок, анализ несовпадения результатов с вычисленными значениями, анализ несовпадения результатов тестирования схемного АЛУ и АЛУ, написанного на языке VHDL).

Практическая работа №31. Тестирование и отладка микропроцессорных систем при программировании.

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)

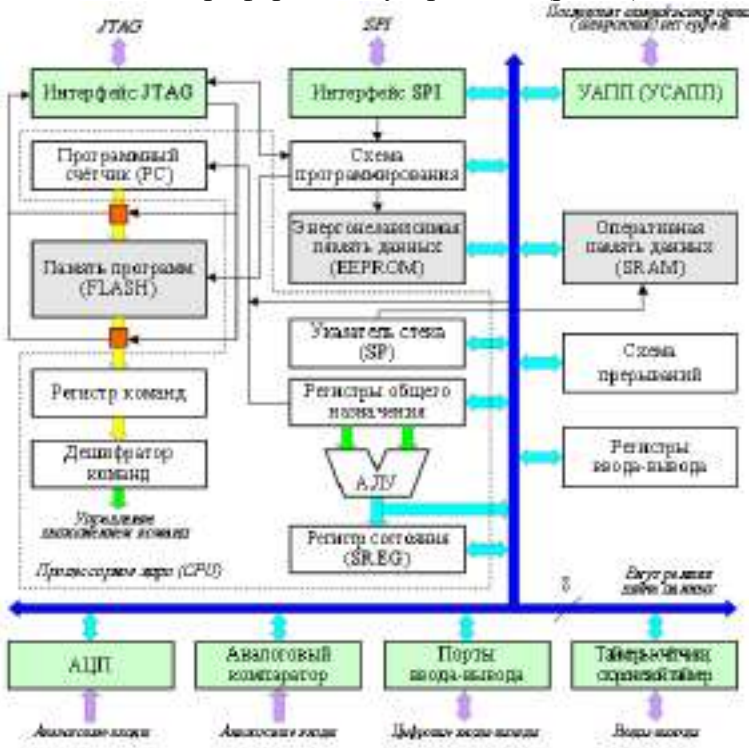


Рис. 2. Архитектура микроконтроллеров семейства AVR

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 128К 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 16К байт) и внешней (до 64К 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).

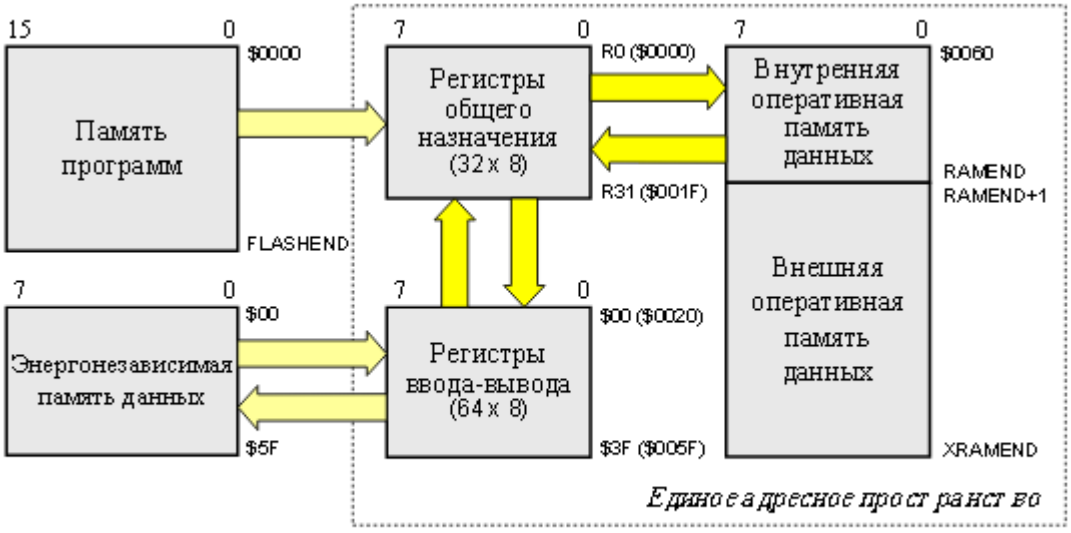


Рис. 3. *Программная модель AVR-микроконтроллеров*

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).

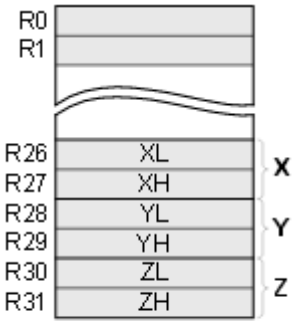


Рис. 4. *Регистры общего назначения*

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.

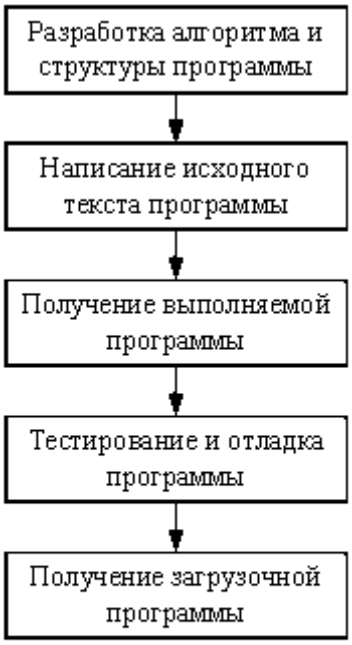


Рис. 5. *Последовательность разработки ПО для микроконтроллеров*

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 №32. Тестирование и отладка микропроцессорных систем при работе с данными.

Цель работы: изучение этапов разработки ПО для встраиваемых микропроцессоров; приобретение навыков работы в среде AVR Studio.

Теоретические сведения

Работа в среде AVR Studio. В состав среды **AVR Studio** входит редактор исходных текстов, транслятор с языка ассемблера, отладчик и симулятор.

Транслятор работает с исходными программами на языке ассемблера, содержащими метки, директивы, команды и комментарии. Метка представляет собой символическое обозначение адреса (последовательность символов, заканчивающаяся двоеточием). Метки используются для указания места в программе, в которое передаётся управление при переходах, а также для задания имён переменных. Директивы являются инструкциями для транслятора и не заносятся в исполняемый код программы. Директивы могут иметь один или несколько параметров. Команды записываются в программе в виде мнемонического обозначения выполняемой операции и могут иметь один или несколько *операндов,* т. е. аргументов, с которыми они вызываются. Транслятор позволяет указывать операнды в различных системах счисления: десятичной (по умолчанию, например, 15, 15 4), шестнадцатеричной (префикс 0x или $, например, 0x0f, $0f, 0x9a, $9a), восьмеричной (префикс - нуль, например, 017, 02 32) и двоичной (префикс 0b, например, 0b00001111, 0b10011010). Строка программы должна быть не длиннее 120 символов и может иметь одну из четырёх форм:

[метка:] .директива [параметры] [;Комментарий]
[метка:] команда [операнды] [;Комментарий]
[;Комментарий]
[Пустая строка]

Позиции в квадратных скобках необязательны. Текст после точки с запятой и до конца строки является комментарием и транслятором игнорируется. Включение в текст программы комментариев является признаком хорошего стиля программирования и облегчает её сопровождение. Кроме того, улучшению читаемости также способствует форматирование текста программы. При программировании на ассемблере выполнение этих правил особенно важно, т. к. программы на языке ассемблера отличаются существенной неудобочитаемостью.

Указать тип микроконтроллера, для которого транслируется программа, позволяет директива . device, например:

.device ATmega8535 ; программа для микроконтроллера ATmega8535

При наличии в программе команд, не поддерживаемых указанным в директиве микроконтроллером, транслятор выдаёт соответствующее предупреждение.

Входным для транслятора является файл <имя_файла>.asm с текстом программы на языке ассемблера. Транслятор создаёт четыре новых файла: файл листинга (<имя_файла>.^^, объектный файл (<имя_файла>.obj), файл- прошивку памяти программ (<имя_файла>.hex) и файл-прошивку энергонезависимой памяти данных (<имя_файла>.eep).

Файл листинга - это отчёт транслятора о своей работе. На рис. 6 приведена часть листинга трансляции программы, в которой числа 2, 5 и 19 заносятся соответственно в регистры **R17, R18** и **R19;** вычисляется произведение и сумма содержимого регистров **R17** и **R18;** из суммы содержимого регистров **R17** и **R18** вычитается содержимое регистра **R19.** Листинг содержит исходный текст транслируемой программы, каждой команде которой поставлены в соответствие машинные коды (правый столбец чисел) и адреса ячеек памяти программ, в которых они будут размещены (левый столбец чисел). Машинные коды и адреса приводятся в шестнадцатеричной системе счисления. Например, строка листинга с командой add содержит следующую информацию: 0f12 - машинный код команды; 0 0 0 0 0 4 - адрес размещения данной команды в памяти программ.

```
000000 e012    ldi R17, 2     ; загрузка числа 2 в регистр R17
000001 e025    ldi R18, 5     ; загрузка числа 5 в регистр R18
000002 e133    ldi R19, 19    ; загрузка числа 13 в регистр R19

000003 9f12    mul R17, R18   ; умножение R17 на R18, результат в R1:R0
000004 0f12    add R17, R18   ; сложение R17 и R18, результат в R17
000005 1b31    sub R19, R17   ; вычитание R17 из R19, результат в R19

000006 cfff    met: rjmp met  ; бесконечный цикл (для отладки)
```

Рис. 6. *Пример листинга трансляции*

Объектный файл имеет специальный формат и используется для отладки программы с помощью симулятора-отладчика среды **AVR Studio.** Файл прошивки памяти программ служит для занесения отлаженной программы в память программ микроконтроллера. Файл прошивки EEPROM-памяти данных предназначен для загрузки информации в энергонезависимую память данных. Операции загрузки памяти программ и энергонезависимой памяти данных выполняются с помощью специальных аппаратных средств (программаторов).

Практическая часть

Составить программу вычисления произведения и суммы двух чисел **A** и **B,** находящихся в РОН. Из суммы **A** и **B** вычесть число **C.** За основу взять программу, приведённую на рис. 6. Числа изменить в соответствии с заданным вариантом (табл. 1). В начало программы поместить директиву . device для микроконтроллера **ATmega8535** (здесь и далее предполагается использование микроконтроллера **ATmega8535).** В комментариях указать фамилию и номер группы.

Выполнить разработку программы в среде **AVR Studio**[1]**,** проделав следующие операции.

Создать новый проект, воспользовавшись командой New Project меню Project. В появившемся диалоговом окне в поле **Project Name** ввести имя создаваемого проекта без расширения (информация о проекте сохраняется в файле с расширением **.aps).** В поле **Location** указать место размещения файлов проекта на диске (путь); в поле **Project type** выбрать пункт **AVR Assembler** (исходные тексты программ разрабатываются на языке ассемблера). Для создания файла исходной программы установить флажок **Create initial File.** Задать имя файла программы, отличающееся от имени проекта, можно в поле **Initial File** (расширение **.asm** файлов программ на ассемблере устанавливается автоматически). Создание каталога для хранения файлов проекта обеспечивается установкой флажка **Create Folder.** Рекомендуется каждый проект размещать в отдельном каталоге. Нажать кнопку «Next»

Таблица 1

№ варианта	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
B	100	99	98	97	96	95	94	93	92	91	90	89	88	87	86
C	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64

Продолжение табл. 1

№ варианта	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
A	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
B	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71
C	65	66	67	68	69	79	71	72	73	74	75	76	77	78	79

2. В группе **Select debug platform and device** в поле **Debug Platform** указать способ отладки создаваемого проекта - **AVR Simulator** (симулятор- отладчик), в поле **Device** выбрать тип микроконтроллера, для которого создаётся программа **(ATmega8535).** Нажать кнопку «Finish». На экране появится дерево иерархии проекта (окно **Workspace,** закладка **Project)** и окно редактора исходных текстов программ.

Если при создании проекта не был установлен флажок **Create initial file,** создать файл исходного текста программы можно командой New File меню File.

3. Ввести и отредактировать текст программы. Сохранить файл, воспользовавшись командой Save меню File. Если файл с исходным текстом программы уже существует, его можно включить в проект командой Add existing File меню Project или контекстного меню окна иерархии проекта при выделенной группе **Assembler.**

4. Провести трансляцию созданной программы, воспользовавшись командой Build and run меню Project (или сочетанием клавиш **Ctrl+F7** на клавиатуре). Перед трансляцией убедиться, что

[1] Запуск программы **AVR Studio** производится из группы **ATMEL AVR Tools** меню **Пуск Программы.**

установлен флажок **List file** в диалоговом окне **AVR Assembler,** вызов которого осуществляется командой AVR Assembler Setup меню Project (это необходимо для создания листинга трансляции). По окончании трансляции в окне **Output** на закладке **Build** появится информация о результатах трансляции. Открыть файл листинга можно из дерева иерархии проекта (окно **Workspace,** закладка **Project).** После этого с помощью команды Save Project меню Project сохранить изменения в файле проекта.

1. Некоторые команды меню программы **AVR Studio** доступны из панели инструментов (см. Приложение 4).

3. Распечатку программ и листингов трансляции обеспечивает команда Print меню File, предварительный просмотр - команда Print Preview меню File, настройку параметров печати - команда Print Setup меню File.

Содержание отчета

Отчёт должен содержать: титульный лист с указанием номера и названия практической работы, номера группы и фамилий выполнивших работу; цель работы; схему программной модели **AVR**-микроконтроллера; перечень этапов разработки прикладного ПО для встраиваемых МП (МК); распечатку листинга трансляции созданной программы с расшифровкой одной из строк.

Контрольные вопросы

1. Назначение однокристальных микроконтроллеров.

2. Особенности архитектуры однокристальных микроконтроллеров.

3. Архитектура и программная модель **AVR**-микроконтроллеров.

4. Этапы разработки ПО для встраиваемых микропроцессоров.

5. Формат строки программы на ассемблере для **AVR**-микроконтроллеров.

Состав листинга трансляции.

Практическая работа №33. Тестирование и отладка микропроцессорных систем при работе с операторами.

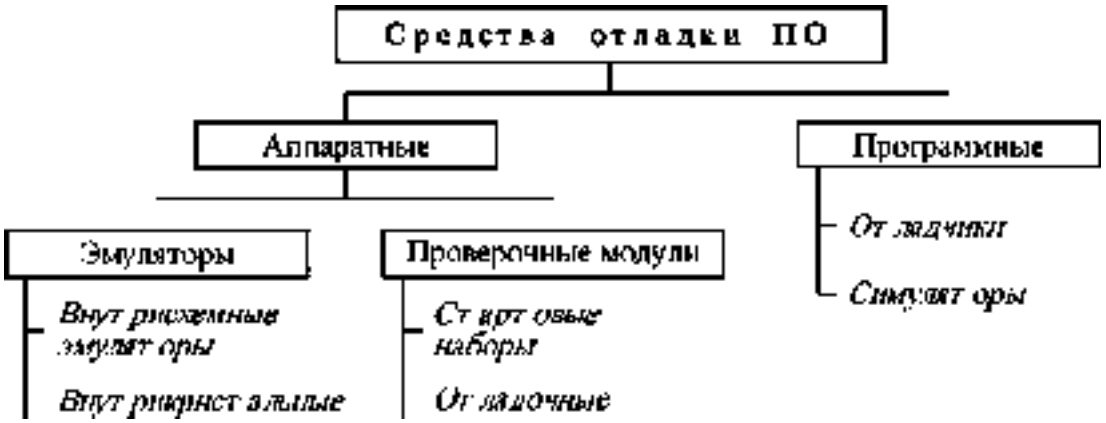
Цель работы: изучение команд отладчика среды **AVR Studio;** приобретение навыков отладки программ под управлением отладчика.

Теоретические сведения.

Особенность отладки ПО устройств на базе встраиваемых МП (в том числе однокристальных микроконтроллеров) состоит в отсутствии в их составе развитых средств для реализации пользовательского интерфейса и ограниченных возможностях системного ПО . В то же время, именно для встраиваемых микропроцессорных систем этап отладки является чрезвычайно ответственным, так как для них характерна тесная взаимосвязь работы ПО и аппаратных средств.

Взаимодействие микропроцессора (микроконтроллера) с датчиками и исполнительными устройствами происходит путём передачи данных через регистры периферийных устройств (регистры ввода-вывода). Отдельные разряды таких регистров задают режимы работы периферийных устройств, имеют смысл готовности к обмену, завершения передачи данных и т. п. Состояние этих разрядов может устанавливаться как программно, так и аппаратно. При отладке ПО часто приходится переходить на уровень межрегистровых передач и проверять правильность установки отдельных разрядов. Кроме того, на этапе отладки может производиться оптимизация алгоритма, нахождение критических участков кода и проверка надёжности разработанного ПО.

Для решения указанных задач применяются аппаратные и программные средства отладки ПО (рис. 33.1).



К **аппаратным средствам отладки** относятся аппаратные эмуляторы и проверочные модули.

Аппаратные эмуляторы предназначены для отладки программного и аппаратного обеспечения микропроцессорных систем в режиме реального времени. Они работают под управлением «ведущего» компьютера, оснащённого специальным ПО - программами-отладчиками (см. ниже). Основными видами аппаратных эмуляторов являются:

- внутрисхемные эмуляторы или эмуляторы-приставки, замещающие микропроцессор в отлаживаемой системе;
- внутрикристальные эмуляторы, представляющие собой одно из внутренних устройств микропроцессора.

Внутрисхемный эмулятор (In-Circuit Emulator, ICE) - это устройство, содержащее аппаратный имитатор процессора и схемы управления имитатором. При отладке с помощью эмулятора микропроцессор извлекается из отлаживаемой системы, на его место подключается контактная колодка, количество и назначение контактов которой идентично выводам замещаемого микропроцессора (рис. 8). С помощью гибкого кабеля контактная колодка соединятся с эмулятором. Управление процессом отладки осуществляется с персонального компьютера. Эмуляторам-приставкам присущи следующие недостатки: высокая стоимость, недостаточная надёжность, высокое энергопотребление, влияние на электрические характеристики цепей, к которым подключается эмулятор.

Внутрикристальные эмуляторы (On-Chip Emulator) позволяют проводить отладку программ без извлечения микропроцессора из системы. При этом осуществляется непосредственный контроль за выполнением программы, так как средства внутрикристальной отладки обеспечивают прямой доступ к регистрам, памяти и периферии микропроцессора.

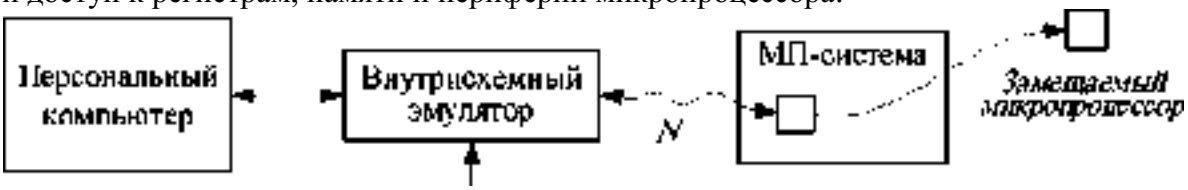


Рис. 33.2. *Отладка с помощью внутрисхемного эмулятора* N - *количество выводов процессора*

Наиболее распространённым средством внутрикристальной отладки является последовательный интерфейс **IEEE 1149.1,** известный как **JTAG** (Joint Test Action Group - Объединённая рабочая группа по автоматизации тестирования). Последовательный отладочный порт **JTAG** микропроцессора с помощью специального устройства сопряжения подключается к компьютеру, чем обеспечивается доступ к отладочным средствам процессора (рис. 33.3). Такой способ отладки также называют *сканирующей эмуляцией.* Достоинствами этого способа является возможность выполнения различных действий на процессоре без его изъятия из системы, использование малого числа выводов процессора и поддержка его максимальной производительности без изменения электрических характеристик системы.

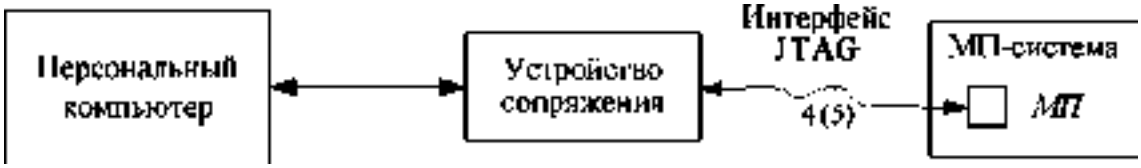


Рис. 33.3. *Отладка с помощью внутрикристального эмулятора*

Проверочные модули предназначены для быстрой отладки программного обеспечения в реальном масштабе времени. Проверочные модули бывают двух видов: стартовые наборы и отладочные платы.

Стартовые наборы (Starter Kit) предназначены для обучения работе с конкретным микропроцессором. Стартовый набор позволяет изучить характеристики микропроцессора, отладить не слишком сложные программы, выполнить несложное макетирование, проверить возможность применения микропроцессора для решения конкретной задачи. В состав стартового набора входит плата, ПО и комплект документации. На плате устанавливается микропроцессор, устройство загрузки программ, последовательные или параллельные порты, разъёмы для связи с внешними устройствами и другие элементы. Плата подключается к компьютеру через параллельный или последовательный порт. Стартовые наборы удобны на начальном этапе работы с микропроцессором.

Отладочные платы (Evaluation Board) предназначены для проверки разработанного алгоритма в реальных условиях. Они позволяют проводить отладку и оптимизацию алгоритма с использованием установленной на плате периферии, а также изготовить на базе платы законченное устройство. Обычно на плате размещается микропроцессор, схемы синхронизации, интерфейсы расширения памяти и периферии, схема электропитания и др. Плата подключается к компьютеру через параллельный или последовательный порт или непосредственно устанавливается в слот **PCI.**

Основными **программными средствами отладки** являются симуляторы и отладчики.

Симуляторы (simulator) или *симуляторы системы команд* представляют собой программы, имитирующие работу того или иного процессора на уровне его команд. Симуляторы обычно используются для проверки программ или её отдельных частей перед испытанием на аппаратных средствах.

Отладчики (debugger) представляют собой программы, предназначенные для анализа работы созданного программного обеспечения. Можно указать следующие возможности отладчиков.

1. *Пошаговое выполнение.* Программа выполняется последовательно команда за командой с возвратом управления отладчику после каждого шага.

2. *Прогон.* Выполнение программы начинается с указанной команды и осуществляется без остановки до конца программы.

3. *Прогон с контрольными точками.* При выполнении программы происходит останов и передача управления отладчику после выполнения команд с адресами, указанными в списке контрольных точек.

4. *Просмотр и изменение содержимого регистров и ячеек памяти.* Пользователь имеет возможность выводить на экран и изменять (модифицировать) содержимое регистров и ячеек памяти.

Отладчики ПО встраиваемых микропроцессоров обычно используются совместно с внутрисхемными или внутрикристальными эмуляторами, а также могут работать в режиме симулятора. Некоторые отладчики позволяют также выполнять *профилирование,* т. е. определять действительное время выполнения некоторого участка программы. Иногда функцию профилирования выполняет специальная программа - *профилировщик* (profiler).

Средства отладки ПО AVR-микроконтроллеров. Аппаратные средства отладки программного обеспечения **AVR**-микроконтроллеров представлены внутрисхемным эмулятором **ICE50,** внутрикристальным эмулятором **JTAG ICE,** а также стартовым набором **STK500.**

К программным средствам отладки ПО **AVR**-микроконтроллеров относятся отладчик и симулятор, входящие в состав среды **AVR Studio.** Отладчик среды **AVR Studio** позволяет проводить отладку программ как в исходных кодах (например, ассемблера), так и в кодах дизассемблера (оттранслированной или скомпилированной программы, записанной с помощью мнемоник ассемблера). Вызов окна с кодом дизассемблера производится командой Disassembler меню View или командой Goto Disassembly контекстного меню редактора исходного текста. Обратное переключение в окно исходного текста осуществляется командой Goto Source контекстного меню окна **Disassembler.**

Отладчик среды **AVR Studio** может использоваться с внутрисхемным эмулятором **ICE50,** внутрикристальным эмулятором **JTAG ICE,** отладочной платой **STK500** или симулятором. Указание способа отладки производится при создании проекта. Симулятор среды **AVR Studio** предназначен для предварительной отладки программ без применения аппаратных средств. В дальнейшем в настоящем лабораторном практикуме для отладки создаваемых программ предполагается применение отладчика среды **AVR Studio** в режиме симулятора.

Отладка ПО в среде AVR Studio. Команды отладчика в программе **AVR Studio** находятся в меню Debug.

Переход в режим отладчика в среде **AVR Studio** осуществляется автоматически при использовании для трансляции программы команды Build and Run или командой Start Debugging меню Debug при использовании для трансляции команды Build. Выход из режима отладчика производится командой Stop Debugging меню Debug.

Пошаговое выполнение программы задаётся командами Step Into, Step Over меню Debug. Команда Step Into позволяет выполнить одну команду программы (в т. ч. команду вызова подпрограммы). Для завершения выполнения подпрограммы может использоваться команда Step Out. Команда Step Over также выполняет одну команду программы, но если это команда вызова подпрограммы, последняя полностью выполняется за один шаг. Следующая выполняемая команда (команда, адрес которой содержится в программном счётчике) обозначается символом ■=[> в окне исходного текста программы. Сброс выполнения программы осуществляется с помощью команды Reset.

Прогон (запуск или продолжение выполнения) программы осуществляется командой Run. Для остановки выполнения программы служит команда Break.

Контрольные точки представляют собой специальные маркеры для программы-отладчика и могут быть трёх типов: точки останова, точки трассировки и точки наблюдения.

Точки останова задаются командой Toggle Breakpoint меню Debug или контекстного меню редактора исходного текста программы. Точка останова обозначается в редакторе исходного текста символом О слева от помечаемой строки. Просмотреть заданные точки останова можно на закладке **Breakpoints** окна **Output;** там же точки останова могут быть запрещены (путём сброса флажка напротив точки останова) и разрешены (путём установки флажка). При достижении точки останова во время прогона программы её выполнение приостанавливается. Повторный вызов команды установки точки останова на той же строке программы приводит к удалению точки останова. Удалить все заданные точки останова позволяет команда Remove Breakpoints меню Debug или команда Remove all Breakpoints контекстного меню закладки **Breakpoints** окна **Output.** Параметры точки останова задаются в диалоговом окне **Breakpoint Condition,** вызов которого осуществляется командой Breakpoints Properties контекстного меню редактора исходного текста программы. Установка флажка **Iterations** позволяет задать количество итераций (повторных выполнений) команды до останова прогона программы. При установке флажка **Watchpoint** по достижению точки

останова производится только обновление значений регистров и ячеек памяти в окнах просмотра. Флажки **Iterations** и **Watchpoint** не должны устанавливаться одновременно. Установка флажка **Show message** обеспечивает отображение сообщений о достижении точки останова на закладке **Breakpoints** окна **Output.** Вызов диалогового окна задания свойств и удаление точки останова может быть произведено из контекстного меню закладки **Breakpoints** окна **Output.**

Точки трассировки предназначены для контроля выполнения программы в режиме реального времени. Трассировка позволяет отслеживать так называемую трассу программы - изменение содержимого регистров и ячеек памяти при выполнении определённых команд (команд, по адресам которых заданы точки трассировки). В среде **AVR Studio** функция трассировки может использоваться только при отладке программы с применением внутрисхемного эмулятора; при работе в режиме симулятора функция трассировки недоступна.

Точки наблюдения задаются командой Add to Watch контекстного меню редактора исходного текста программы. Точки наблюдения представляют собой символические имена регистров или ячеек памяти, содержимое которых необходимо отслеживать. При выполнении команды Add Watch на экране появляется окно **Watches,** разделённое на четыре столбца: **Name** (символическое имя точки наблюдения), **Value** (значение), **Type** (тип), **Location** (местонахождение). Новая точка наблюдения может быть также задана в выделенной ячейке столбца **Name** окна **Watches** или командой Quickwatch в окне редактора исходного текста программы (при этом курсор должен находиться на имени регистра или ячейки памяти). Значения, отображаемые в столбце **Value,** обновляются при изменении содержимого соответствующего регистра или ячейки памяти. Удалить заданные точки наблюдения можно их окна **Watches.**

Отладчик среды **AVR Studio** также обеспечивает следующие функции: выполнение до курсора (команда Run to Cursor меню Debug) и последовательное выполнение команд с паузами между ними (команда Auto Step меню Debug).

Для удобства использования в процессе отладки ряд команд отладчика доступен с клавиатуры (табл. 2).

Таблица 2

Команда отладчика	Клавиша	Команда отладчика	Клавиша
Run	F5	Step Into	F11
Break	Ctrl+F5	Step Out	Shift+F11
Reset	Shift+F5	Step Over	F10
Run to Cursor	Ctrl+F10	Toggle Breakpoint	F9

Для просмотра и изменения содержимого регистров и ячеек памяти служат команды *Registers, Memory, Memory 1, Memory 2, Memory 3* меню *View.*

По команде Registers на экране отображается окно **Registers,** в котором приводятся шестнадцатеричные представления содержимого РОН. Изменение (модификация) содержимого регистров производится путём двойного щелчка мышью. Наблюдение за содержимым РОН может быть также произведено с помощью дерева устройств микроконтроллера, находящегося на закладке **I/O** окна **Workspace.** Для этого необходимо раскрыть объекты **Register 0-15** и **Register 16-31** щелчком мыши по знаку «+».

Команды Memory, Memory 1, Memory 2, Memory 3 обеспечивают вызов окон **Memory,** служащих для отображения содержимого ячеек оперативной и энергонезависимой памяти данных, памяти программ, регистров ввода-вывода и РОН. Выбор типа памяти, отображаемой в окне **Memory,** производится с помощью списка, расположенного в панели управления окна **(Data** - оперативная память данных, **Eeprom** - энергонезависимая память данных, **I/O** - регистры ввода-вывода, **Program** - память программ, **Register** - РОН).

Для наблюдения за состоянием процессора необходимо раскрыть объект **Processor** закладки **I/O** окна **Workspace.** При этом будет отображена следующая информация: содержимое программного счётчика **(Program Counter);** содержимое указателя стека **(Stack Pointer),** количество тактов, прошедших с начала выполнения **(Cycle Counter);** содержимое 16-разрядных регистров- указателей **X, Y** и **Z;** тактовая частота **(Frequency);** затраченное на выполнение время **(Stop Watch).**

Для наблюдения содержимого регистров ввода-вывода необходимо раскрыть объект **I/O *** закладки **I/O** окна **Workspace,** где ***** - тип микроконтроллера. Регистры ввода-вывода, входящие в объект **I/O,** сгруппированы по типам периферийных устройств.

Модифицированные значения содержимого регистров и ячеек памяти действуют только во время текущего сеанса отладки, в исходный текст программы изменения не заносятся.

Практическая часть.

Провести отладку созданной в практической работе №32 программы с помощью симулятора-отладчика среды **AVR Studio,** проделав следующие операции.

1. Выполнить программу в пошаговом режиме, отслеживая изменение содержимого используемых в программе регистров. Обратить внимание на изменение содержимого программного счётчика. Сравнить содержимое программного счётчика при выполнении команд с их адресами в памяти программ, приведёнными в листинге трансляции и окне памяти программ.

2. Выполнить прогон программы. Проверить правильность результата работы программы.

3. Задать точку останова на команде загрузки в РОН числа **B.** Включить режим отображения сообщений о достижении точки останова. Выполнить прогон программы с контрольными точками. Задать точку останова на команде умножения. Выполнить прогон программы с контрольными точками. Удалить заданные точки останова.

4. Задать точки наблюдения в используемых РОН. Выполнить программу в пошаговом режиме, отслеживая изменение их содержимого.

Содержание отчета.

Отчёт должен содержать: титульный лист с указанием номера и названия практической работы, номера группы и фамилий выполнивших работу; цель работы; краткие теоретические сведения (классификацию средств отладки ПО, перечень основных функций программ-отладчиков); список использованных команд отладчика **AVR Studio** с указанием их назначения.

Контрольные вопросы.

1. Классификация средств отладки прикладного ПО встраиваемых МП.

2. Виды и особенности аппаратных средств отладки ПО.

3. Основные функции программных средств отладки ПО.

4. Пошаговое выполнение программы и его возможности.

5. Особенности прогона программы с контрольными точками.

6. Контрольные точки: типы, назначение, использование.

Практическая работа №34. 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	Объем резидентной	Тактовая частота,	Ток потребления,

		памяти программ, Кбайт	памяти данных, байт	МГц	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 P0, P1, P2, P3);
- 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 0000H -0FFFH 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;

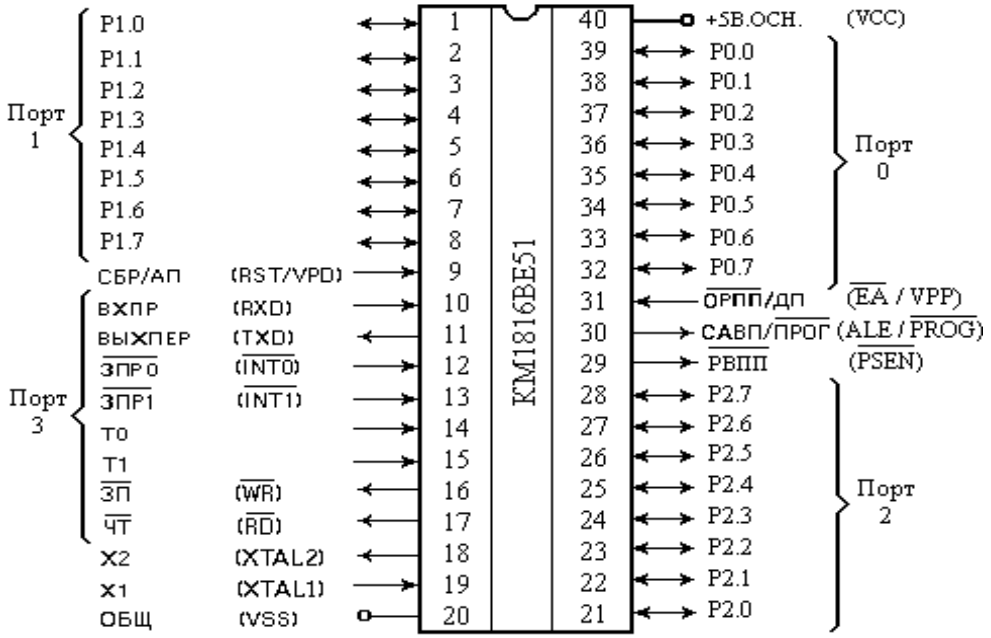


Рис. 3.1. Цоколёвка корпуса МК51 и наименование выводов

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 Т1 и Т2, 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 Н.

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 №35. Микроконтроллеры семейства МК51 (типы операндов).

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.

Ход работы:

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 №36. Исследование логических элементов (система прерываний).

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 №37. Параллельный интерфейс.

Цель работы: изучение схем динамической и статической индикации

Краткие теоретические сведения

В состав интегральной микросхемы К580ВВ55 (ИМС) программируемого параллельного интерфейса входит:

1. устройство управления вводом-выводом;
2. двухнаправленнй буфер данных;
3. порты A и B, 8-ми разрядные регистры с шинными формирователями;
4. порт C, два 4-ех разрядных регистра с шинами формирователями;
5. регистр управляющего слова.

Порт A и старшая половина порта C может объединятся в группу A. Порт B и младшая половина порта C может объединятся в группу B. Для обеспечения работоспособности ИМС необходимо записать управляющее слово.

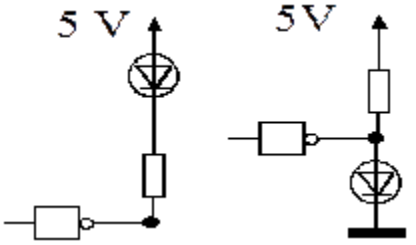
Формат управляющего слова инициализации.

Управляющее слово инициализации определяет режим работы и направление передачи данных.

Бит	

D7	Определяет либо установку режимов работы каналов D7-"1", либо работу ППИ в режиме сброса/установки отдельных разрядов порта С											
D6		Режим 0		Режим 1		Режим 2	Группа А					
D5												
D4		Вывод PA(7-0)		Ввод PA(7-0)								
D3		Вывод PC(7-4)		Ввод PC(7-4)								
D2		Режим 0		Режим 1	Группа В							
D1		Вывод PB(7-0)		Ввод PB(7-0)								
D0		Вывод PC(3-0)		Ввод PC(3-0)								

Простейшими приборами отображения информации в цифровых устройствах являются светодиоды и цифровые индикаторы.



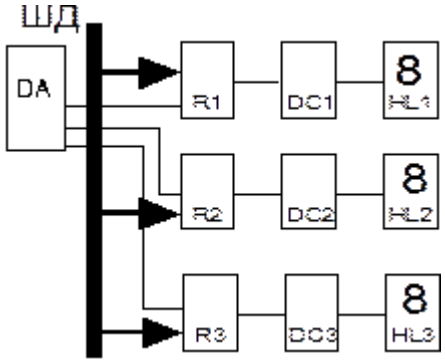
Включение единичных индикаторов.

В полупроводниковых светодиодах используется свойство p-n перехода излучать свет в видимой части спектра при протекании через него прямого тока ($I_{пр}$=5-20мА, $U_{пр}$=2-3В). Варианты включения индикаторов на рис5.

Для отображения цифровой информации наибольшее распространение получили семи сегментные индикаторы, в которых изображение цифры составляют из семи линейных светодиодных сегментов расположенных в виде цифры 8.

На основе светодиодов и семи сегментных индикаторов строятся подсистемы отображения информации. При построении подсистем отображения информации различают два подхода, динамическая и статическая индикация.

Статическая индикация заключается в постоянной подсветке индикаторов HL1-n от одного источника информации рис.6.



DA – дешифратор адреса, необходим для выборки соответствующего регистра

R1-R3- регистры, в которых временно хранится значение кода числа для отображения (соответствующий регистр выбирается DA).

DC1-DC3 – семи сегментные дешифраторы, преобразующие двоичный код в семи сегментный код.

HL1-HL3 – семи сегментные индикаторы.

ШД – шина данных, по ней осуществляется передача данных на индикацию.

Структурная схема статической индикации.

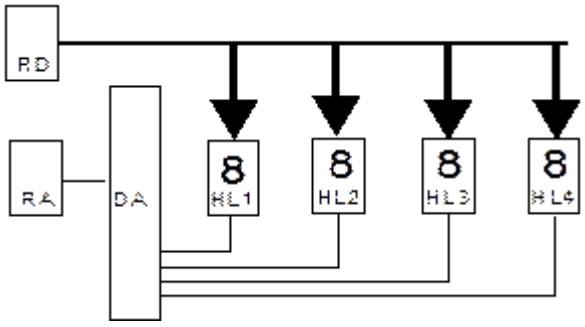
В такой системе каждый индикатор HL1-n подключен через собственный дешифратор DC1-n и регистр-защелку RG1-n к шине данных, выборка регистров RG1-n производится при помощи селектора адреса CA. Аппаратные затраты при такой организации составляют n пар регистр + дешифратор при n десятичных разрядов индикатора.

В стенде адресация реализована с помощью дешифратора микросхему КР555ИД7, выходные сигналыCS2, CS3, т.к. адреса индикаторов находятся в поле адресации внешней памяти данных, то для выборки используется сигнал WR с процессора. Микросхемы индикации КР490ИП2 содержат в себе регистр и семи сегментный дешифратор.

Сущность динамической индикации заключается в поочередном циклическом подключении каждого индикатора HL1-n к источнику информации через общую шину данных, рис 7.

Выборка индикатора осуществляется дешифратором DA. В регистре RD хранится цифровой код, предназначенный для отображения. В регистре RA хранится адрес индикатора.

Структурная схема динамической индикации



RD- регистр данных для временного хранения отображаемого числа либо символа.

RA- регистр адреса для временного хранения двоичного кода адреса выбираемого индикатора.

DA-для преобразования адреса задаваемого двоичным кодом а позиционный код.

HL1-HL4- семи сегментные индикаторы.

При таком включении значительно уменьшаются аппаратные затраты. Но необходимо обеспечить достаточное время свечения одного индикатора, для того чтобы не уменьшалась яркость. Также необходимо обеспечить такую частоту перебора индикаторов, чтобы не было заметно мерцание. Преимущества такого способа заметны при количестве десятичных разрядов индикации больше 5.

На стенде статическая индикация реализована на четырех статических семи сегментных двоично-десятичных индикаторах (HG1 – HG4). Обращение к ним производится, как к ячейкам памяти с адресами A000h , B000h. Передача данных на индикаторы осуществляется с ОЭВМ по шине данных AD(0-7). Сигналы выборки микросхем индикации CS2, CS2.

Динамическая индикация реализована с помощью двух семи сегментных индикаторов HG2, HG3. Управление динамической индикацией осуществляется с помощью элементов DD4, DD3.4, DD3.5 (линия данных A,B,C,D,E,F,G,H, -PB0, PB1, PB2, PB3, PB4, PB5, PB6 ,PB7) сигналы поступают с порта PB микросхемы параллельного приемопередатчика DD10 (см. схему электрическую стенда приложение 1) , сигналы выборки соответствующего индикатора поступают от линии порта PC микросхемы DD10 к транзисторам VT2 и VT3.

Также на плате расширения установлены матрица 5x7 светодиодов HG1, и линейка светодиодов HL1-HL8. Светодиоды зажигаются записью логических единиц в соответствующие разряды порта PA микросхемы параллельного приемопередатчика DD10.

Управление светодиодной матрицей осуществляется по линиям PA0-PA4 параллельного приемопередатчика DD10 и P1.0-P1.6. Например для того чтоб засветить точку с координатами [1;1] необходимо вставить уровень логической единицы на линии PA0, и уровень логического нуля на линии P1.0.

Пример программы для статической индикации

С частотой 1 Гц отобразить на статическом индикаторе число 04:

```
CSEG
ORG 0
Continue:
mov A,#04h ;записать в Акк. число 04
mov DPTR,#0A000h ;устан.в DPTR адрес Инд.DD15,DD16
movx @DPTR,A ;засветить на Инд. DD15,DD16 Ч 04
mov DPTR,#0B000h ;установить в DPTR адрес Инд.DD17,DD18
movx @DPTR,A ;засветить на Инд. DD17,DD18 Ч 04
mov R1,#0FFh ;временная задержка на 2 регистрах
C1: mov R2,#0FFh ;с декрементом во вложенном цикле,
C3: djnz R2, C3 ;определяет время свечения
djnz R1, C1 ;индикаторов
mov A,#0FFh ;записать в Акк. число FFh
mov DPTR,#0A000h ;установить в DPTR адрес Инд.D15,DD16
movx @DPTR,A ;потушить индикаторы DD15,DD16
mov DPTR,#0B000h ;установить в DPTR адрес Инд. DD17,DD18
movx @DPTR,A ;потушить индикаторы DD17,DD18
mov R1,#0FFh ;временная задержка на 2 регистрах
C2: mov R2,#0FFh ;с декрементом во вложенном цикле,
C4: djnz R2, C4 ;определяет время гашения
djnz R1, C2 ;индикаторов
jmp Continue ;переход на начало программы
END
```

Практическая часть

Порядок выполнения работы.

1) Изучить принцип работы различных методов отображения.

2) Разработать алгоритм для выполнения индивидуального задания до начало лабораторного занятия

3) Разработать программу для выполнения индивидуального задания до начало лабораторного занятия с использование подпрограмм.

4) Ввести программу индивидуального задания на персональном компьютере.

5) Изучить программно отладочные средства (ПОС) для КР1816ВЕ31.

6) С помощью ПОС проанализировать выполнение индивидуальной программы

7) Загрузить программу в стенд ОЭВМ. Убедиться в правильном выполнении индивидуального задания, при отрицательном результате осуществить изменение алгоритма либо программы. Повторить загрузку программы в стенд ОЭВМ

8) Распечатать листинг правильно работающей программы.

9) Ответить на контрольные вопросы преподавателя

Варианты индивидуальных заданий

№.	*Текст индивидуального задания*
	Занести в рег. R1 XXH, отнимая от числа единицу отображать на Д_Инд. HG2, HG3 полученное значение до нуля с частотой 1 сек . С частотой 0,5 Гц. Переключать Свет. HL3, HL4
	Занести в В ДД число X0, в рег. R1 XXH, число из В. отображать на С_Инд. HL0 с частотой 1 Гц, число из R1 отображать на Д_Инд. HG2, HG3 с частотой 0,25 Гц.
	Вкл. Свет. HL1. Занести в рег. В ДД число 0X, в рег.R5 X0, два разряда суммы (десятки и единицы) поочередно отображать на С_Инд. HG1, HG0 и на Д_Инд. HG3, HG2 с частотой 1 Гц..
	Занести в рег. R6 ДД число XXH, в рег.R5 ДД XX, в рег.R0 ДД XX, последовательно отображать эти числа по одному разряду на на Д_Инд. HG2, HG3, С_Инд. HG1, HG2, HG3, HG4.
	Попеременно вкл. Свет. HL8-HL7. Занести в рег. R2 ДД число 0X, в рег.R1 0X, сумму чисел отобразить на Д_Инд. HG2, HG3.
	Занести в рег. А ДД число 0X, в рег.R2 X0, число из А отобразить на С_Инд. HG3, число из рег. R2 отображать на Д_Инд. HG3 с частотой в 0.25 Гц.
	Занести в Ак. ДД число XX, в рег. R1 XX, младшие два разряда суммы чисел отобразить на Д_Инд. HG2, HG3, старшую тетраду С_Инд. HG3, осуществить плавное загорание HL8.
	Занести в рег. R6 ДД число XXH преобразовать его в двоично десятичное отобразить его на С_Инд. HG4- HG1.
	Занести в рег. В ДД число XX, в рег. R3 XX, разность чисел отобразить на Д_Инд. HG2, HG3.
	Вкл. Свет. HL5. Занести в Акк. ДД число XX, в рег. R5 X0, число из Акк. Отобразить на С_Инд. HG1, HG2, число из R5 отобразить на Д_Инд. HG2, HG3.
	Занести в рег. R0 ДД число XX, попеременно отображать мл. и ст. тетраду на Д_Инд. HG2, HG3 с частотой 0,25 Гц.
	Занести в рег. R2 ДД число 0X, в рег.R5 0X, сумму чисел отобразить на Д_Инд. HG2, HG3.
	Занести в рег. В ДД число, с частотой 2 Гц выводить это число на С_Инд. HG1, HG2 и одновременно на Д_Инд. HG2, HG3.
	Попеременно вкл. Свет. HL4-HL6. Занести в ячейку с адресом 0010h внешней памяти ОЭВМ ДД число 0X, в рег.R3 XXH, сумму чисел отобразить на Д_Инд. HG2, HG3.
	Занести в рег. R1 ДД число 0X, в рег.R3 XX, младшие два разряда суммы отобразить на Д_Инд. HG2, HG3 с медленным (в течение 5 сек.) затуханием этого числа, старшую на С_Инд. HG1.

Контрольные вопросы

1. Назначение, программирование параллельного интерфейса КР580ВВ55. Назначение и сфера применения устройств с статическим и динамическим методом отображения.

2. Расчет времени регенерации для динамического метода отображения.

3. Обоснование необходимости применения различных методов отображения

4. Схемотехнические решения для построения схем отображения информации
5. Схемы включения единичных индикаторов .
6. Включение жидкокристаллических индикаторов
7. Включение газоразрядных индикаторов

Практическая работа №38. Исследование последовательного интерфейса (асинхронный порт).

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 Т/С1 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 Т/С1, 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 Р2, определяющее показания цифрового индикатора секунд.

Двоично-десятичный счетчик минут реализован с помощью порта Р1, аналогичный счетчик часов ? с помощью Р0. При включении контроллера счетчики сбрасываются и на цифровые индикаторы заносятся нули. Установка реального времени производится в определенной последовательности. Сначала держат лог. 0 на входе Т1 до тех пор, пока индикаторы покажут требуемое число часов. Затем держат лог. 0 на входе Т0 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 Т/С0 дополнительный код числа 10000?

Практическая работа №39. Исследование последовательного интерфейса.

Цель работы: Изучение принципа работы COM-портов PC.

Теоретические сведения

Последовательная передача данных

При последовательной передаче данных биты, образующие коды символов, пересылаются по очереди один за другим по единственному проводнику, возвратным проводом является земля. Персональные компьютеры (PC) часто имеют два коммуникационных порта (обычно называемых COM-портами). Именно эти порты используются для асинхронной последовательной передачи

данных путем посылки и получения ASCII кода. Этот протокол передачи данных известен как стандарт RS232C. На рисунке 1 показано, как выглядит передаваемый сигнал при пересылке символа R. Каждому символу предшествует стартовый бит (логический 0), который передается перед стоповым битом. Длина стопового бита может составлять 1, 1.5 или 2 обычных бита. Число изменений состояния канала передачи данных в секунду называется скоростью передачи в бодах. В системе с двумя возможными состояниями скорость передачи в бодах равна числу битов, передаваемых за секунду (бит/с). При пересылке данных между компьютерами (или между компьютером и периферийным устройством) оба компьютера должны работать в одном и том же режиме так, чтобы во время передачи какого- либо символа тактовый генератор на принимающем компьютере работал на той же частоте, что и на передающем компьютере. Такая система является асинхронной, так как минимальное число символов, передаваемых в секунду, определяется скоростью работы передатчика, т. е. передатчик может сделать паузу любой длительности между символами. Во время любых пауз линия находится в состоянии логической единицы, поэтому естественно в качестве стартового бита при передаче стартового бита при передаче использовать логический ноль. Один из наиболее распространенных способов пересылки битов в системе асинхронной передачи данных показан на рисунке 1. Здесь логический 0 представлен уровнем напряжения, превышающем +3В(обычно 10В), а логическая 1 - уровнем напряжения, меньшим – 3В(обычно – 10В). Нулевое напряжение указывает на неработающую систему или на разрыв на линии. Режим работы ИС приемника передатчика может быть запрограммирован путем использования: 1) пакета программ; 2) режима командной строки MS-DOS (рисунок 2); 3) некоторой команды в СИ программе.

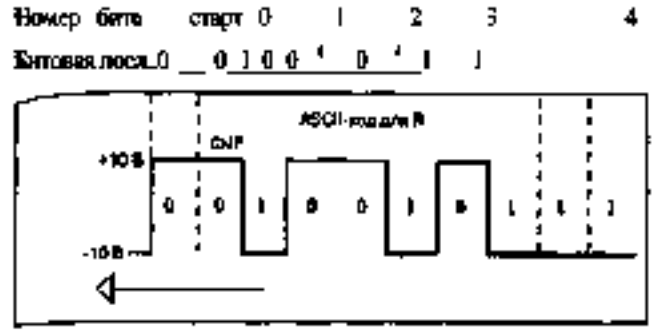
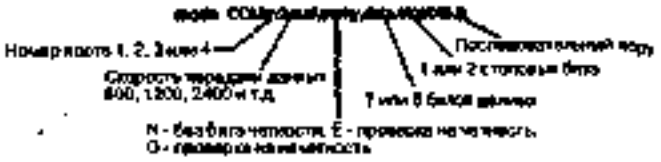


Рисунок1.-Сигнал стандартаRS232при передаче символа «R».



Пример Напечатайте>mode com1:9600,n,8,1,P.

Рисунок 2 - Режим командной строки MS-DOS.

Аппаратные средства последовательного интерфейса.

Специальные ИС, называемые универсальными асинхронными приемопередатчиками - UART, преобразуют данные из параллельного (TTL) формата, получаемого из шины данных микропроцессора, в последовательный (RS232). Эти устройства можно рассматривать как устройства с двумя внутренними секциями: преобразователем параллельного формата в последовательный, называемый передатчиком, и преобразователем последовательного формата в параллельный, называемый приемником. Передатчик представляет бит четности, стартовый и стоповый биты. Приемник проверяет формат и четность полученных данных. Приемник и передатчик могут работать одновременно, такой способ передачи называют полным дуплексом (передача только в одном направлении называется симплексной связью, передача в обоих направлениях, называется полудуплексной связью). Скорости передачи и приема данных

контролируются генератором тактовых импульсов, работающим на частоте, кратной скорости передачи данных. Внутренний регистр состояния, который хранит флажки состояний приема передачи и ошибок, может быть прочитан программным способом. Стандартным соединителем является 25-штырьковый разъем д типа (штепсельная часть). Однако в некоторых случаях применяется 9-штырьковый разъем, который пригоден для соединения наиболее часто используемых управляющих линий. Назначение выводов 25-штырькового разъема показано на рисунке 3.

Рисунок 3.- Назначение контактов Сом порта.

Практическая часть.

В начале работы следует проверить адрес последовательного порта используемого компьютера. Типичные адреса COM –портов ПК для пересылки и получения данных приведены ниже: 1 последовательный порт: 3F8 hex 2 последовательный порт: 2F8 hex.

Задание 1 Необходимо отослать некий символ с вывода «Передаваемые данные» (вывод 2) и получить на выводе «Принимаемые данные» (вывод 3) того же порта. Для этого теста нужен лишь один компьютер. Вывод 2 «Передаваемые данные» соединен с выводом 3 «Принимаемые данные» того же самого порта. Структурная схема программного обеспечения к этому заданию показана на рисунке 4.

Рисунок 4.- Структурная схема ПО.

```
/*Serial.c Тестирование последовательного порта*/
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
int Address=0x3f8; /*Адрес порта COM1,заменить на 0x2f8 для COM2*/
void main()
{
char menu='x';
system("cls"); /*Очистка экрана*/
printf("\n Передача данных через последовательный порт");
printf("\n R для получения данных");
printf("\n S для отсылки данных");
printf("\n Q для выхода из программы\n");
while(1)
{
menu=getch();
switch(menu)
{
case 'S': printf("\n Введите символ с клавиатуры для его пересылки");
          outp(Address,getch());
          printf("\nS,R,Q");
          break;
case 'R': printf("\n Полученный символ %c",inp(Address));
          printf("\n S,R,Q");
          break;
case 'Q': exit(1);
}
fflush(stdin);
}
}
```

Тестирование

Соедините выводы 2 и 3 COM-порта.

Нажмите на клавиатуре S, а затем введите символ.

1. Нажмите R: тот же самый символ должен быть получен на экране.

2. Отсоедините выводы 2 и 3. При нажатии R на экране должен быть получен другой символ.

Контрольные вопросы.

1. Почему каждому символу предшествует стартовый бит?

2. Приведите два важных преимущества использования напряжения противоположной полярности для передачи логических состояний, а не обычных TTL-уровней 5 и 0в.

3. Какое логическое состояние передается между символами?

4. За какое время будет передан 7-разрядный символ с дополнительным битом четности при скорости передачи 1000бод?

5. Каким образом принимающее устройство узнает о том, что передающее устройство не подключено, если используется трехпроводная система, в которой соединены выводы 2, 3 и 7?

Практическая работа №40 Разработка программного обеспечения.

Цель работы*:* ознакомление с возможностями среды разработки AVR Studio и практическом освоении режимов её работы.

Общие сведения

AVR Studio – это интегрированная отладочная среда разработки приложений (IDE) для микроконтроллеров AVR компании Atmel (рис. 1). IDE AVR Studio содержит:

средства создания и управления проектом;

редактор кода на языке Ассемблер;

транслятор языка Ассемблера (Atmel AVR);

отладчик (Debugger);

программное обеспечение верхнего уровня для поддержки внутрисхемного программирования (In-SystemProgramming, ISP) с использованием стандартных отладочных средств Atmel AVR.

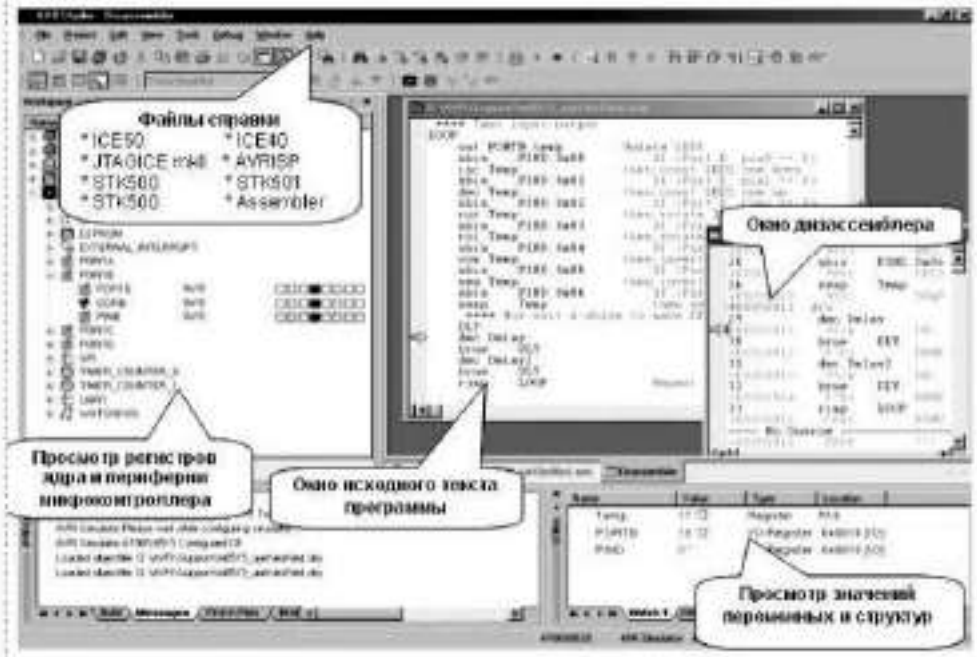


Рис. 1. Интерфейс интегрированной среды разработки AVR Studio

4

Работа с AVR Studio начинается с создания проекта. Сначала необходимо указать используемый микроконтроллер (МК) и платформу, на которой будет производиться отладка программы.

Написание программы производится в окне редактора текста программы. Для использования символических имен регистров специального назначения вместо их адресов необходимо подключить (директива .include) к проекту файл определения регистров специального назначения (например, m16def.inc для ATmega16). Включаемые файлы входят в прикладное программное обеспечение AVR Studio и при инсталляции помещаются в папку Appnotes в директории установки AVR Studio.

Написание программы в AVR Studio производится на языке Ассемблер. Последние версии AVR Studio содержат AVR Ассемблер второй версии, который в дополнение к стандартному Ассемблеру поддерживает новые директивы Ассемблера, Си - подобные директивы препроцессора, создание переменных определенного типа.

В результате трансляции создается выходной файл в формате HEX (расширение .hex). Если не выдается сообщений об ошибках, можно приступать к отладке проекта.

Отладчик AVR Studio поддерживает все типы МК AVR и имеет два режима работы: режим программной симуляции и режим управления различными типами внутрисхемных

эмуляторов (In-CircuitEmulators) производства фирмы Atmel. Важно отметить, что интерфейс пользователя не изменяется в зависимости от выбранного режима отладки.

Отладочная среда поддерживает выполнение программ как в виде ассемблерного текста, так и в виде исходного текста языка Си, загруженного в объектном коде.

Управление отладкой производится командами меню DEBUG либо соответствующими иконками на панели инструментов AVR Studio.

Пользователь может выполнять программу полностью в пошаговом режиме, трассируя блоки функций или выполняя программу до того места, где стоит курсор. В дополнение можно определять неограниченное число точек останова, каждая из которых может быть включена или выключена. Точки останова сохраняются между сессиями работы.

В AVR Studio для отладки программы предусмотрены две команды пошагового режима: Step Over и Trace into. Разница между ними в том, что при выполнении команды Step Over выполнение подпрограмм происходит до полного окончания без отображения процесса выполнения. К командам шагового режима также относится команда Auto Step.

С помощью команд пошагового режима можно проследить изменения значений в переменных, регистрах ввода/вывода, памяти и регистрового файла. Для этого предназначены раздел I/O рабочей области AVR Studio (см. рис. 1), окно Watch (меню Debug \ Quick watch).

Интегрированная среда разработки AVR Studio также поддерживает просмотр (меню View / Memory) ячеек памяти программ, памяти данных, EEPROM и регистров портов ввода/вывода в ходе исполнения. Выпадающее меню диалогового окна позволяет выбрать один из четырех массивов ячеек памяти: Data, IO, EEPROM, Program Memory. Для одновременного просмотра нескольких областей окно Memory может быть открыто несколько раз. Информация в диалоговом окне может быть представлена в виде байтов или в виде слов в шестнадцатеричной системе счисления, а также в виде ASCII-символов.

В процессе отладки пользователь может инициализировать внутреннее ОЗУ, или EEPROM МК (например, данными, содержащимися в полученном при трансляции файле .eep), или сохранить содержимое ОЗУ и EEPROM в виде файлов в формате Intel Hex (меню File ->Up/Download Memory).

Помимо шагового режима, возможна отладка программы с использованием точек останова (меню Breakpoints ->Toggle Breakpoint). Командой Start Debugging запускается исполнение программы. Программа будет выполняться до остановки пользователем или до обнаружения точки останова.

Для наблюдения за работой программы можно открыть несколько окон, отображающих состояние различных узлов МК (см. рис. 1). Окна открываются нажатием соответствующих кнопок на панели инструментов или при выборе соответствующего пункта меню View. Если в процессе выполнения программы в очередном цикле значение какого-либо регистра изменится, то этот регистр будет выделен красным цветом. При этом если в следующем цикле значение регистра останется прежним, то цветовое выделение будет снято. Такое же цветовое выделение реализовано в окнах устройств ввода/вывода, памяти и переменных.

Состояние встроенных периферийных устройств МК, а также состояния программного счетчика, указателя стека, содержимого регистра статуса SREG и индексных регистров X, Y и Z отображено в окне I/O View. В этом окне отражаются все функциональные блоки микроконтроллера. Любой блок может быть раскрыт нажатием на его значок. При раскрытии блока в окне отражаются адреса и состояния всех его регистров и отдельных, доступных для модификации, битов. Каждый доступный для модификации бит может быть установлен или сброшен как программой по ходу ее исполнения, так и пользователем вручную (указав курсором мыши нужный бит и щелкнув левой кнопкой мыши, пользователь может изменить значение бита на обратное), а в режиме программной симуляции это является способом имитации входного воздействия на МК.

Порядок выполнения работы

Последовательность действий, которую нужно выполнить:

-создание проекта;

-создание файла программы;

-компиляция;

-симуляция.

1.Запустить программу: из меню Пуск->Все программы->AtmelAVR Tools->AVRStudio 4.

2.В появившемся окне выберите New Project или то же самое можно сделать из меню Project/New Project.

3.Укажите имя и расположение проекта на жестком диске, например, как на рис. 2 введено имя My project и указано его расположение

D:\AVR\ Project. Нажмите Next.

Рис. 2. Окно для указания имени и месторасположения проекта

4. В появившемся окне (рис. 3) выберите AVR Simulator и ATmega8515. Нажмите Finish.

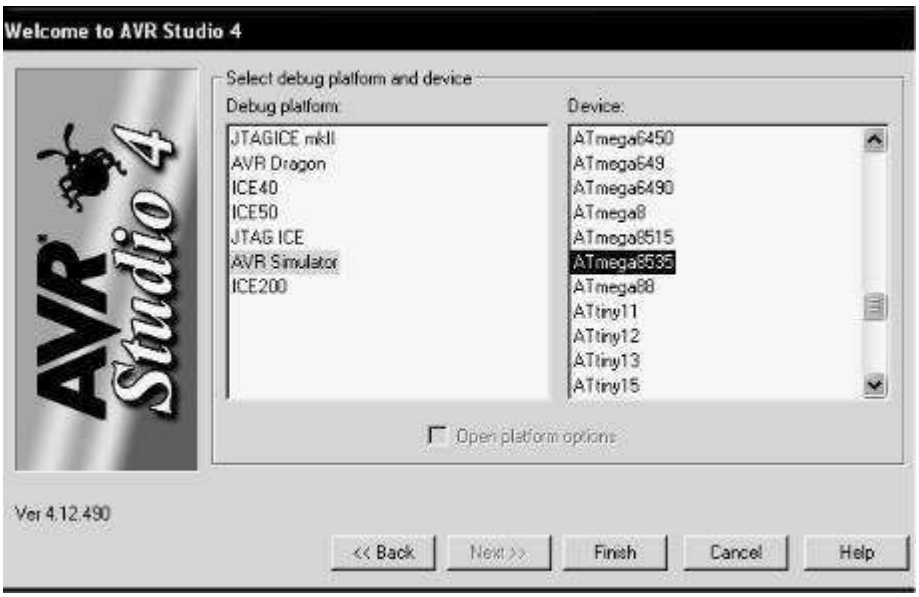


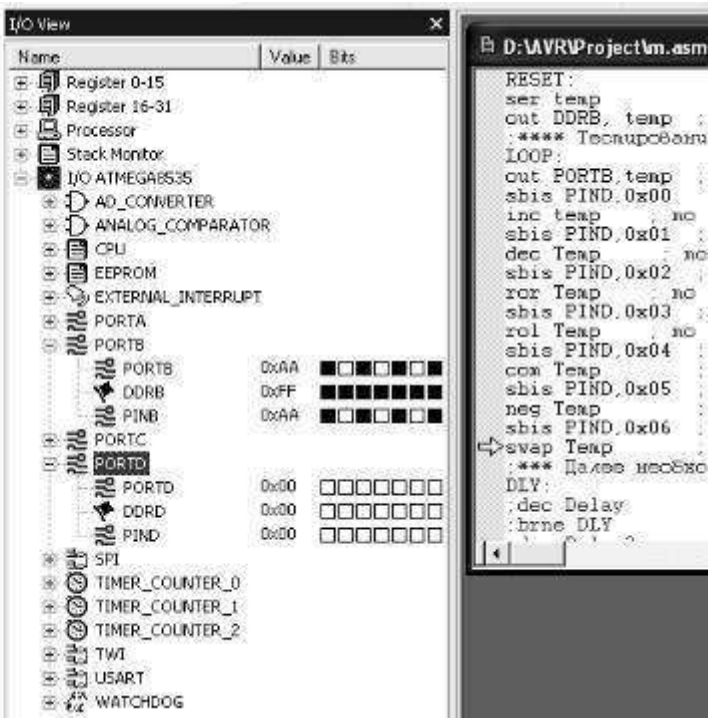
Рис. 3. Окно для выбора платформы для отладки и марки МК

5. В открывшемся окне редактора наберите исходный текст программы (комментарии на русском языке можно не набирать):

```
.include "8515def.inc"
.def Temp =r16 ; Регистр хранения временных данных
.def Delay =r17 ; Переменная 1 для генерации задержки
.def Delay2 =r18 ; Переменная 2 для генерации задержки ;***** Инициализация
RESET:
```

```
ser temp
out DDRB, temp     ; Настройка порта B (PORTB) на вывод
;**** Тестирование ввода/вывода
LOOP:
out PORTB,temp     ; Обновление состояния светодиодов
sbis PIND,0x00     ; Если PortD.0 = 0,
                   ; то уменьшение на 1 двоичного кода, формируемого свечением
inc temp           светодиодов
sbis PIND,0x01     ; Если PortD.1 = 0,
                   ; то увеличение на 1 двоичного кода, формируемого свечением
dec Temp светодиодов
sbis PIND,0x02     ; Если PortD.2 = 0,
ror Temp ; то циклический сдвиг состояния светодиодов на 1 вправо
sbis PIND,0x03     ; Если PortD.3 = 0,
rol Temp ; то циклический сдвиг состояния светодиодов на 1 влево
sbis PIND,0x04     ; Если PortD.4 = 0,
com Temp           ; то инверсия состояния всех светодиодов
sbis PIND,0x05     ; Если PortD.5 = 0,
neg Temp           ; то инверсия состояния и прибавление 1
sbis PIND,0x06     ; Если PortD.6 = 0,
swap
Temp               ; то обмен тетрадами светодиодов
;*** Далее необходима задержка, чтобы сделать проделанные изменения видимыми
DLY: dec Delay
```

8




```
brne DLY
dec Delay2
brne DLY
                   ;     Повторение
rjmp LOOP цикла заново
```

В практической работе эта программа будет запрограммирована в МК ATMega8515 и будет управлять включением светодиодов на плате STK 500. Так как симуляция проекта выполняется во много раз медленнее, чем выполнение программы реальным МК, на время

симуляции отключите задержку, которая записана в последних строчках программы. Для этого поставьте знак точка с запятой «;» в начале четырех строчек программы между строчками *DLY* и*rjmp LOOP*.

6.Запустите компиляцию проекта командой Build/Build или клавишей F7. При появлении сообщения об ошибке наведите на него указатель мышки и щелкните левой кнопкой. Строка с ошибкой в тексте программы будет указана синей стрелкой. Если ошибок нет, выполните симуляцию проекта.

7.Запустите отладку из меню Debug/Start Debugging. В окне I/O View раскройте меню I/O ATMEGA8515, а затем PORT B и PORT D (рис. 4). Запустите программу на выполнение из меню

Debug/Auto Step или кнопкой на панели инструментов. Строка программы, которая выполняется в текущее время указывается в окне редактора желтой стрелкой (см. рис.4). Имитировать нажатие кнопок можно щелчком кнопки мыши, когда её указатель наведен на соответствующий бит

врегистре PIN D.

8.Продемонстрируйте работу в режиме отладки преподавателю.

Контрольные вопросы

1.Чем отличается микроконтроллер от микропроцессора?

2.Чем отличается симуляция от эмуляции?

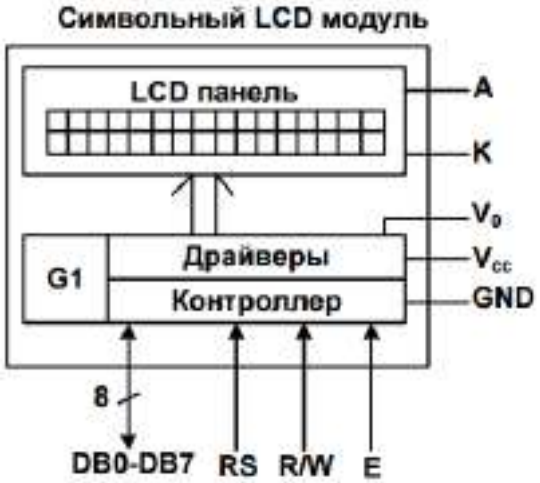
Практическая работа №41. Программирование микроконтроллера на языке ассемблера.

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.

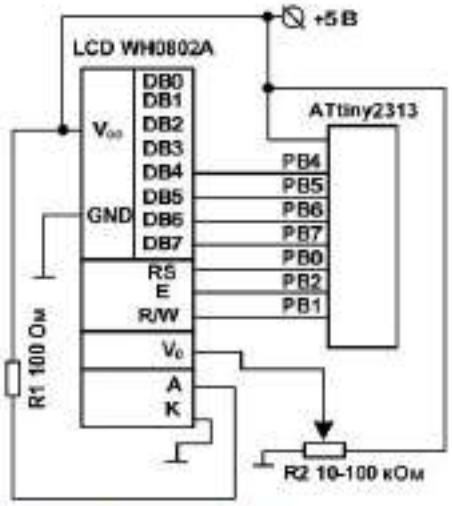


Рисунок 5 – Схема подключения микроконтроллера ATtiny2313 к ЖКИ

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 №42.Электронное сопряжение компонентов схем.

Цель работы: ознакомление с программой Electronics Workbench фирмы Interactive Image Technologies Ltd, приобретение навыков моделирования электрических схем.

Основные теоретические сведения.

Моделирующие программы широко используются в процессе проектирования радиоаппаратуры предприятиями, производящими современную электронную технику. Наибольшее распространение в мире получила программа **PSpice** фирмы **MicroSim,** ставшая де-факто стандартом профессиональной моделирующей программы для **ПЭВМ.** Эта программа, использующая адекватные математические модели компонентов электронных схем и совершенные вычислительные алгоритмы. Позволяет с высокой достоверностью оценить поведение схемы во временной и частотной областях с учетом технологического разброса параметров компонентов и влияния температуры окружающей среды. В ряде задач, например при проектировании больших интегральных схем, моделирующая программа становится единственным средством анализа работоспособности устройства ввиду очевидной невозможности подключения измерительного оборудования к внутренним структурам кристалла.

Однако мощные и многофункциональные программы такого уровня требуют от пользователя определенной квалификации уже при подготовке исходных данных для моделирования. Поэтому для учебных целей в качестве инструмента анализа электронных **схем** предлагается использовать программу **Electronics Workbench** фирмы **Interactive Image Technologies Ltd.,** отличающуюся наглядностью и простотой использования даже новичками. Обладая меньшей точностью и функциональными возможностями по сравнению с **PSpice,** эта программа позволяет решать основные задачи, возникающие при проектировании схем, и получать результаты в виде показаний виртуальных измерительных приборов: мультиметра, осциллографа и т.д.

Принцип компьютерного макетирования

Система **Electronics Workbench** представляет собой комплекс программ, функционирующих на **ПЭВМ IBM PC** под управлением **WINDOWS и** предназначена для проектирования аналоговых и цифровых электронных схем с визуализацией исходных данных и результатов проводимых анализов.

Процесс проектирования и анализа происходит во взаимодействии студента с диалоговым интерфейсом программы. Объекты выполняемой лабораторной работы имитируют свои реальные прототипы как по внешнему виду, так и по способу манипуляции их органами управления. Так, создаваемая с помощью встроенного графического редактора принципиальная схема собираемого из радиодеталей устройства, уже является достаточной информацией для ее моделирования. Номиналы и типы радиоэлементов (компонентов схемы) выбираются в процессе создания **схемы** и автоматически проставляются рядом с их условными графическими обозначениями **(УГО).** Для визуализации результатов анализа схемы на монтажном столе - рабочем поле экрана размещается нужные **виртуальные** (не существующие реально) измерительные приборы: осциллограф, вольтметр и т.д. Виртуальные приборы воспроизводят свои реальные прототипы.

В результате моделирование как таковое скрыто от пользователя программы, а процесс выполнения лабораторной работы сводится к привычной для лабораторной практики сборке

испытуемого макета схемы, подключению и настройке приборов и, наконец, проведению измерений. Отличие состоит в том, что работа выполняется не на физических, а на виртуальных объектах на экране монитора **ПЭВМ.** Таким образом, экран представляет собой как бы поверхность монтажного стола с размещенными, а ней макетом и набором измерительных приборов, подключенных к макету.

Состав монтажного стола

После запуска программы **Electronics Workbench,** на экране появляется изображение монтажного стола (рис.1).

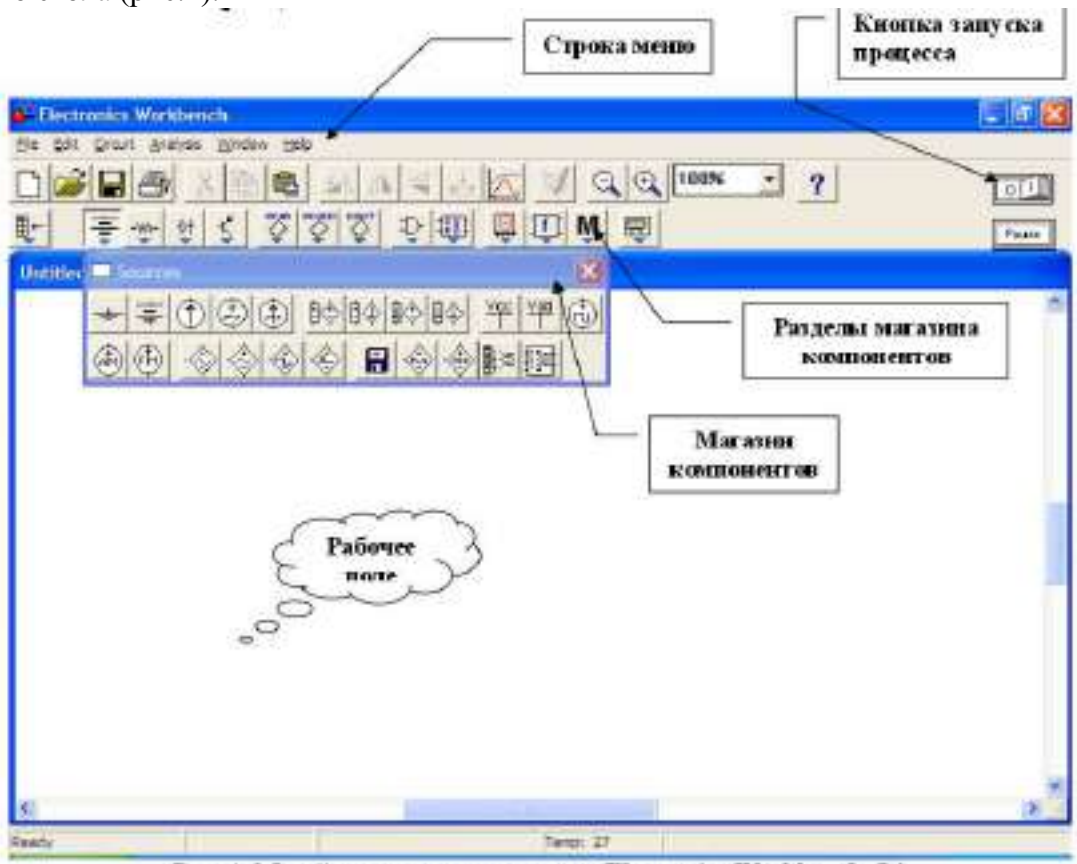


Рис. 1 Общий вид монтажного стола Electronics Workbench 5.0

Монтажный стол содержит все, что может понадобиться для построения макета схемы и его испытания. Основные составные части стола перечислены ниже :

1 рабочее поле - большая центральная часть экрана, на которой размещаются компоненты схемы и измерительные приборы;

2 магазин компонентов и панель измерительных приборов, расположенный над рабочим полем;

3 разделы магазина компонентов;

4 строка меню в верхней части экрана, предназначенная для выбора операций со схемой (загрузка, сохранение, удаления, правка и т.д)

5 клавиша включения питания в правом верх углу экрана, которой запускается процесс моделирования.

Рабочее поле монтажного стола используется для сборки подключения приборов. Рабочее поле представляет собой окно, которое можно перемещать, изменять его размеры и просматривать подобно другим окнам операционной среды **WINDOWS.**

1.3. Магазин компонентов

Магазин компонентов появляется после запуска программы в виде вертикальной полосы с условным графическими обозначениями **(УГО)** компонентов электрических схем. Кроме того, непосредственно над рабочим полем появляются клавиши с пиктограммами разделов магазина, один из которых является активным. Активизация раздела осуществляется щелчком на клавише. Разделы магазинов приведены на рис. 2

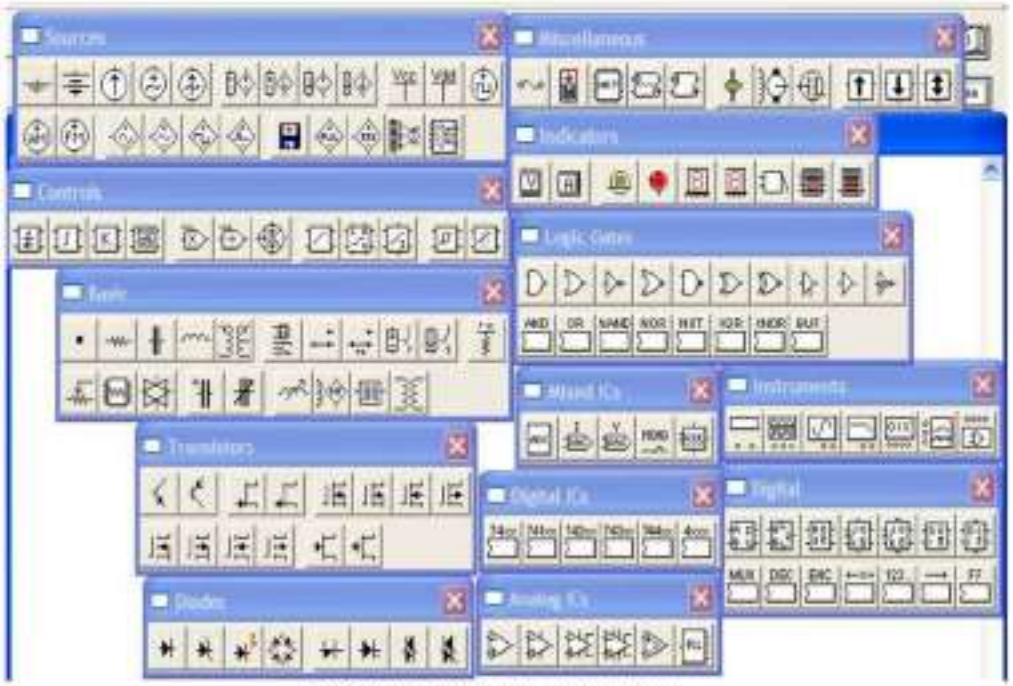


Рис.2 .Разделы магазина компонентов.

Разделы содержат:

1 заказные компоненты, которые могут формироваться пользователем, например путем создания подсхем (функциональных узлов, состоящих из компонентов других разделов и изображаемых в виде единого блока). Этот раздел на рис.2 не показан;

2 пассивные компоненты (резисторы, конденсаторы, катушки индуктивности и пр.);

3 активные компоненты (диоды ,транзисторы , операционные усилители и пр.);

4 управляемые компоненты , значения параметров которых можно изменять в процессе моделирования (потенциометром, управляемые переключатели и пр.);

2 гибридныемикросхемы , главным образом цифро-аналоговые и аналого-цифровые преобразователи;

3 элементы индикации;

4 логические элементы цифровых схем;

5 комбинационные цифровые узлы (мультиплексоры, дешифраторы и пр.);

6 последовательные цифровые узлы (триггеры, счетчики, регистры и пр.);

7 конкретные микросхемы из типовых серий микросхем.

Некоторые особенности размещения и обозначения аналоговых и цифровых компонентов схем рассмотрены в разделе 2.

Разделы **(Basic** и **Sources)** содержит главным образом пассивные компоненты электронных схем. Основные представители раздела перечислены ниже:

соединитель в виде круглой точки используется для соединения проводников и создания контрольных точек в схеме. Соединитель появляется автоматически при выполнении соединения, когда перемещаемый конец линии связи совмещается с изображением уже имеющегося проводника. Соединитель можно также перетащить мышью из магазина компонентов на рабочее поле. Каждый соединитель обеспечивает подключение до четырёх проводников к одному с каждой из ортогонально ориентированных сторонисточников назначаются после двойного щелчка на их изображении. При установке параметров напряжения назначаются среднеквадратические значения тока и напряжения **I** и **U,** связанные с амплитудными значениями **Ia , Ua** отношением

$$U=U\mathbf{a}/2$$

Следует иметь в виду , что все источники в схеме , включая внешний измерительный генератор, должны иметь одну и ту же частоту. Поэтому назначение частоты одного источника приводит к автоматической установке частоты остальных источников;

резистор, конденсатор и катушка индуктивности являются наиболее распространенными пассивными компонентами схем. Сопротивление резистора (Ом) конденсатора (Ф) и индуктивность катушки (Гн) назначаются после двойного щелчка на их изображениях.

трансформатор: представляет собой пару связанных между собой общим магнитопроводом катушек индуктивности. Коэффициент n трансформации напряжения характеризует отношение амплитуд переменных напряжений в выходной и входной обмотках и назначается после двойного нажатия на изображении трансформатора. Направления намотки провода на катушках трансформатора отмечаются точками рядом с началом обмотки. Если обмотки трансформатора ориентированы согласно, то его коэффициент трансформации положителен, в противном случае коэффициент трансформации отрицателен;

плавкий предохранитель предназначен для защиты схемы, как правило по входу, выходу и цепи питания, от токовой перегрузки.

Раздел **Sources** содержит также стандартный источник питания 5В для питания цифровых микросхем и переменные резистор, конденсатор и катушку индуктивности.

Разделы **(Basic, Diodes, Transistors)** включает распространенные полупроводниковые приборы, используемые для построения аналоговых схем:

диод, представляющий собой p-n переход, пропускающий ток только в одном направлении. По умолчанию предполагается идеальный диод, однако после двойного щелчка можно назначите модель реального диода из числа имеющихся моделей в библиотеке программы;

стабилитрон, или диод Зенера, использующий участок стабильного напряжения на обратной ветви вольт-амперной характеристики диода. Напряжение стабилизации можно назначить в пределах от 2,4 В до 200 В или же выбрать модель реального стабилитрона;

светодиод, который применяется для излучения света при протекании через него тока. Для идеального светодиода назначается пороговый ток, после превышения, которого начинается свечение светодиода. С помощью диалогового окна можно также задать модель реального светодиода;

биполярные транзисторы n-p-n и p-n-p типов. По умолчанию программа предлагает модели идеальных транзисторов, выбор модели реального транзистора из числа имеющихся в библиотеке осуществляется посредством диалогового окна после двойного щелчка на его изображении;

операционные усилители - это усилители с гальванической связью выхода со входами В магазине компонентов содержатся два вида моделей операционного . усилителя, вторая из которых учитывает цепи его питания. Входным сигналом операционного усилителя служит разность его входных напряжений, т.е.

Uвых=K(U++U-)

По умолчанию программа предлагает модель идеального операционного реального прибора из числа имеющихся в библиотеке. Основными параметрами модели, которые можно корректировать, являются; коэффициент усиления, напряжение смещения нуля, разность и абсолютные значения входных токов, частота единичного усиления (частота входного синусоидального напряжения усилителя без обратной связи, при которой коэффициент усиления уменьшается до единицы).

Данный раздел включает также разновидности тиристоров и типовых диодных схем. Раздел (**Transistors)** включает модели полевых транзисторов с каналами n- и p-типов с управляющим p-n переходом и с изолированным затвором. Конкретный тип полевого транзистора назначается посредством диалогового окна. По умолчанию программа предлагает модели идеальных транзисторов Раздел (**Basic)** содержит модели управляемых компонентов:

механические переключатели на два направления, способные переключаться нажатием клавиши на клавиатуре ПЭВМ. Клавиша, управляющая могут использоваться для сопряжения аналоговых схем с цифровыми схемами, а также для управления исполнительны ми органами электромеханических систем. Дня данных переключателей посредством диалогового окна назначаются пороги включения ($У_{оп}$ или I_{on}) и выключения (Y_{off} или I_{off}). Порог включения определяет условие замыкания переключателя, а порог выключения - условие его размыкания.

Порог включения выбирается, как правило, выше порога выключения, что обеспечивает гистерезисную передаточную характеристику и надежность работы переключателя;

источники напряжения и тока, управляемые напряжением или током позволяют построить макромодели сложных электронных узлов с привлечением минимального числа компонентов их эквивалентных схем. Посредством диалогового окна для каждого из четырех разновидностей управляемых источников задается коэффициент пропорциональности значений выходного и входного параметров:

для источника тока, управляемого током, назначается коэффициент усиления по току

Р=1вых/1вх (А/А)

для источника тока, управляемого напряжением, задается крутизна

G = 1вых/1вх (А/В);

для источника напряжения, управляемого током, устанавливается транзисторный коэффициент

И=$У_{в}$ых/1вх (В/А);

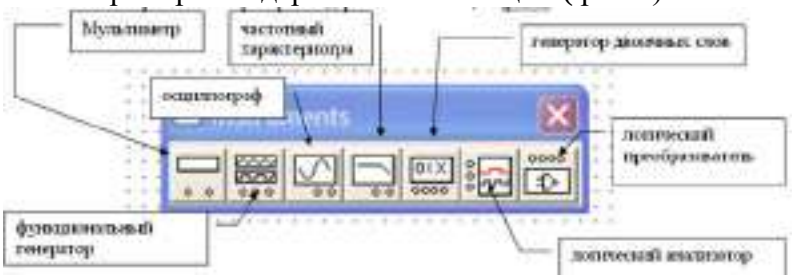
источник напряжения, управляемый напряжением, характеризуется коэффициент усиления по напряжению

Е=Увых/**У** вх **(В/В).**

Коэффициенты **F** и **E** - безразмерные, **G** имеет размерность **сименс,** а **H - Ом;** сложной функциональной зависимостью от нескольких аргументов. Остальные разделы магазина компонентов содержат элементы и узлы цифровых схем, а также аналого-цифровые и цифро-аналоговые преобразователи для сопряжения аналоговых и цифровых схем.

1.4. Панель инструментов

Панель измерительных приборов содержит семь позиций (рис.3)



Кроме того, имеются вольтметр и амперметры, расположенные в магазине компонентов.

Мультиметр предназначен для измерения напряжения, тока, сопротивления или ослабления сигнала в децибелах между двумя контрольными точками

Порядок манипуляции органами управления мультиметра и других описан в разделе 2.

Функциональный генератор является источником напряжения, который вырабатывает аналоговый сигнал синусоидальной, прямоугольной или треугольной формы. Органами управления этого прибора можно настроить частоту, коэффициент заполнения (отношение длительности сигнала к периоду), амплитуду и постоянную составляющую выходного напряжения.

Осциллограф используется для визуального наблюдения электрических сигналов в функции времени, измерения их амплитудных и временных параметров (длительности, времени задержки, периода повторения и пр.). Осциллограф снабжен двумя входными зажимами (каналы А и В) и позволяет наблюдать одновременно два сигнала в двух различных узлах схемы.

Частотный характериограф позволяет измерять амплитудно- и фазочастотную характеристики (АЧХ и ФЧХ) схемы (получать *диаграммы Боде).* **АЧХ** представляет собой отношение напряжений в двух контрольных точках в функции изменяющейся частоты, причем отношение напряжений представляется в логарифмическом масштабе (в децибелах). ФЧХ представляет собой зависимость от частоты разности фаз двух напряжений в градусах.

Генератор слов применяется для исследования цифровых схем. Он вырабатывает восемь последовательностей цифровых двоичных сигналов, периодически повторяющихся каждые шестнадцать тактов, а также тактовые импульсы, которые можно использовать для

синхронизации проверяемой схемы. Информационное содержание последовательностей можно задавать по желанию пользователя.

Логический анализатор служит для развертки во времени восьми цифровых сигналов при исследовании цифровых схем. Этот прибор подобен 8-канальному осциллографу для одновременного наблюдения цифровых сигналов в восьми различных узлах схемы.

Логический преобразователь, входящий в состав панели приборов, в зависимости от назначенного его органами управления режима способен преобразовывать логические функции из одной формы представления в другую и синтезировать схемы комбинационной логики по заданной булевой функции.

1.5. Опорное меню

Строка опорного меню **Electronics Workbench** расположена в верхнем левом углу монтажного стола (рис. 1). Опорное меню включает пункты: **File** Файл

Edit Редактирование
Circuit Схема
Analysis Анализ
Window Окно
Help Помощь

Выбрать пункт из меню можно либо щелчком на его имени, либо одновременным нажатием клавиши **ALT** и подчеркнутого символа в имени пункта (например, чтобы открыть пункт Circuit следует ввести **ALT+C).** Выбор пункта приводит к бокса команд этого пункта, выбор команды осуществляется аналогично. *Если имя команд тусклое , то команда недоступна.*

Пункт **File** включает команды для работы с файлами:

New (Ctrl+N)	**Создать новый файл**
Open... (Ctrl-Ю)	**Открыть файл ...**
Save (Ctrl+S)	**Сохранить**
Save As...	**Сохранить как...**
Revert to Saved...	**Восстановить**
Print... (Ctrl+P)	**Печать...**
Print Setup...	**Настройка печати ...**
Exit(Alt+F4)	**Выход**

Перечисленные команды являются обычными командами **WINDOWS.** Команда **Print Setup** дает дополнительную возможность назначения альтернативного принтера, не указанного в панели управления **WINDOWS,** а также выбора ориентации печатаемого изображения, размера бумаги и насыщенности распечатки с помощью диалогового окна.

Команда **Activate** запускает программу моделирования, «включая» питание схемы. Запуск можно также осуществить щелчком на выключателе питания в верхнем правом углу монтажного стола.

Команда **Stop** используется для остановки моделирования. К тому же результату приводит щелчок на выключателе питания.

Команда **Pause/Resume** предназначена дня временного прерывания моделирования и его возобновления с точки останова.

Команда **Label** применяется для позиционного обозначения предварительно выделенного компонента посредством диалогового окна. После введения позиционного обозначения диалоговое окно закрывается щелчком на клавише **ОК.**

Пункт **Circuit и Analysis** открывают команды, с помощью которых построение схемы и задаются директивы моделирования

Команда **Value** предназначена для указания значения параметра предварительно выделенного щелчком на его изображении компонента с помощью появляющегося диалогового окна. После того, как значение будет напечатано, необходимо щелкнуть на клавише **Accept.**

Команда **Model** устанавливает тип модели компонента посредством диалогового окна Обычно тип компонента выбирается из числа имеющихся в библиотеке моделей, однако имеется возможность создания собственных моделей.

Команда **Zoom** предназначена- для увеличения размеров изображения предварительно выделенного прибора или раскрытия структуры подсхемы. Команде Zoom соответствует также двойной щелчок на изображении прибора.

Команда **Rotate** позволяет изменять ориентацию компонента, поворачивая его изображение на 90° по часовой стрелке. Вольтметр и амперметр этой командой не вращаются, однако изменяется ориентация их выводов.

Команда **Fault** создает дефектный компонент для обучения поиску неисправностей в схеме. После выбора этой команды следует указать посредством диалогового окна характер дефекта: **Open circuit** (холостой ход -разрыв цепи) или **Short circuit** (короткое замыкание). Устранение дефекта достигается выбором режима **None** (Нет) из меню данной команды.

Команда **Subcircuit** предназначена для объединения части собранной схемы в самостоятельный функциональный узел, подобный интегральной микросхеме, который будет размещен в разделе Customs магазина компонентов.

Для создания подсхемы сначала необходимо собрать ее из имеющихся компонентов на рабочем поле монтажного стола, выделить компоненты подсхемы, а затем выбрать команду Subcircuit. В появившемся диалоговом окне печатается имя подсхемы и одновременно указывается режим копирования компонентов в подсхему (с изменением или сохранением взаимного расположения компонентов), после чет изображение подсхемы появляется в разделе Customs магазина, а изображения входящих в подсхему компонентов на рабочем поле заменяются изображением подсхемы. При этом все внешние подключения компонентов подсхемы становятся выводами подсхемы.

В дальнейшем подсхема может корректироваться после двойного щелчка на ее **УГО** совершенно аналогично редактированию основной схемы на рабочем поле. Новые выводы подсхемы образуются при перетаскивании проводника от внутреннего компонента подсхемы к краю подсхемы, при этом клавиша мыши отпускается после появления на краю подсхемы маленького кружочка.

Команда **Wire color** устанавливает цвет предварительно выделенного проводника (см. п.2.2).

Команда **Preferences** позволяет настроить рабочее поле монтажного стола и внешний вид схемы. Данная команда предусматривает следующие директивы:

Show grid - щелчок на этой опции показывает точечную координатную сетку рабочего поля;

Use grid - щелчок на этой опции включает использование координатной

Show labels - включает отображение на рабочем поле позиционных обозначений компонентов, которые появляются рядом с их **УГО;**

Show values и Show models применяются для отображения на рабочем поле значений параметров и типов моделей компонентов. Следует отмстить, что данные опции не выполняются, если в наборе команд Restrictions пункта Circuit назначено «Hide values and models restrictions» («Скрыть значения параметров и модели»)

Пункт **Analysis options** включает команды, устанавливающие виды выполняемых анализов, точность расчетов и особенности отображения результатов (см. п.п. 3.1, 3.2).

Пункт **Restrictions** содержит команды ограничения:

Hide values find models скрыть значение компонентов. Данный прием может использоваться идентификацию скрытых величин путем тестирования схемы;

Hide subcircuits - скрыть все подсхемы. Данная команда превращает подсхемы в «черные ящики» с неизвестной структурой, восстановление которой может составить суть задания студенту;

Hide parts bin - скрыть магазин компонентов - ограничивает доступ к магазину компонентов, предлагая студенту для выполнения задания ограничиться набором компонентов, уже имеющемся на рабочем поле;

Hide unused instruments - скрыть неиспользуемые приборы. Команда не позволяет доставать с панели приборов дополнительные объекты и предлагает ограничиться при проведении измерений теми из них, которые уже размещены на рабочем поле;

Hide faults - скрыть дефекты. Команда отменяет выделение ранее «испорченного» компонента, после чего внешне этот компонент не будет отличаться от исправных;

Password - пароль. Команда предотвращает несанкционированную отмену ограничений кем- либо, кроме автора ограничений. После первого же сохранения схемы доступ к диалоговому окну Restriction посредством ввода пароля.

Пункт **Window** опорного меню предлагает команды управления окнами программы Electronics Workbench:

Arrange (Ctrl+W) Упорядочивание
1 Circuit Схема
3 Description (Ctrl+D) Описание

Команда **Arrange** позволяет аккуратно расположить на рабочем столе поле монтажного стола открытые окна программы (раскрытые приборы, описание). Для этого окна перемещаются примерно туда, где их хотелось бы видеть, после чего выбирается команда Arrange.

Команда **Circuit** выводит рабочее поле на передний план.

Команда **Description** позволяет разместить текстовый комментарий к схеме на английском языке в окне описания.

Пункт **Help** предоставляет быструю контекстно-зависимую справку по использованию программы.

2.МОНТАЖ СХЕМЫ

2.1. Размещение и обозначение компонентов

Манипуляции с виртуальными компонентами осуществляются с помощью мыши и клавиатуры. Количество однотипных компонентов, извлекаемых из магазина компонентов, не ограничено.

Для размещения на рабочем поле монтажного стола нужного компонента указатель подводится мышью к пиктограмме соответствующего раздела магазина компонентов (рис.2) и выполняется щелчок левой клавишей мыши. Это приводит к активизации раздела и появлению его содержимого в магазине компонентов слева от рабочего поля. После этого указатель подводится к **УГО** нужного компонента и нажимается левая клавиша мыши. Далее , не отпускается клавиши **УГО** переносится на рабочее поле, где клавиша отпускается, когда выделяется красным цветом. Для выделения более одного компонента при нажатой клавише **SHIFT** на клавиатуре выполняется щелчок на всех выделяемых компонентах. К такому же результату приводит щелчок правой клавишей мыши без использования клавиатуры. Для выделения группы объектов можно также, установив указатель слева и выше самого левого объекта в группе, переместить указатель по диагонали вправо и вниз. По мере перемещения появится расширяющийся прямоугольник. Когда прямоугольник захватит все объекты группы, левую клавишу мыши можно отпустить. Выделение компонентов используется для указания их типономиналов, удаления, перемещения.

После размещения компонента в желаемой позиции его изображение можно ориентировать с помощью команды вращения для того, чтобы получить более читаемую конфигурацию **схемы** без излишних пересечений проводников. Вращение выделенного объекта или группы компонентов на 90^0 выполняется командой **Rotate** (Вращение) из меню **Circuit** (Схема). Этой же команде соответствует нажатие двух клавиш - **Ctrl и R (Ctrl+R).**

Для указания позиционного обозначения и типономинала выделенного компонента используются команды **Model** (Модель), **Label** (Обозначение) и **Value** (Значение) из меню **Circuit.** Командам **Model u Value** эквивалентны двойные щелчки на изображении компонента или с клавиатуры **Ctrl+M и Ctrl+U** соответственно. Если обозначения компонентов до указанных команд не были заданы. то сначала появляется диалоговое окно **Label.** Для компонентов имеющих сложные модели (например, для транзисторов), по умолчанию устанавливаются идеальные модели. Замена модели компонента производится с помощью диалогового окна модели, которое появляется после команды **Model** (Clrl+M). Диалоговое окно предлагает

перечень моделей компонентов, из которого выбирается нужный и нажимается клавиша **ОК.** Таблицы соответствия зарубежных типов компонентов отечественным приведены в **Приложениях А...Д.**

После назначения типономиналов компонентов рядом с их **УГО** на рабочем поле монтажного стола появляются их позиционные обозначения и номиналы (или типы для активных компонентов). Если эти данные на рабочем поле не обязательны, то их можно скрыть командой **«Hide values and models»** («Скрытие значений и моделей») в диалоговом окне **Restrictions** (Ограничения) из меню **Circuit.**

Выделенный компонент можно удалить с рабочего поля или нажатием клавишей **DEL** или перетащив **УГО** компонента назад в магазин компонентов.

- **Выполнение межсоединений**

Когда все нужные компоненты окажутся на рабочем поле монтажного стола, их выводы необходимо соединить друг с другом в соответствии с электрической принципиальной схемой. Соединение выводов двух компонентов выполняется указателем с помощью мыши. Для этого указатель подводится к выводу одного компонента (при этом вывод выделится), нажимается левая клавиша мыши и при нажатой клавише указатель подводится по кратчайшему нуги к нужному выводу другого компонента. По мере перемещения указателя за ним следует линия подводимого проводника. Когда указатель окажется на нужном выводе, последний также выделится. Тогда клавиша отпускается и линия связи автоматически трассируется между двумя выделенными выводами компонентов. При необходимости, например если линии располагаются слишком близко друг к другу или проходят над **УГО** компонентов, проводник можно переместить, указав проводник щелчком мыши и перетащив его после появления на месте указателя двунаправленной стрелки.

Для соединения двух проводников и создания контрольных точек в схеме используется **УГО** соединителя (точки), представляющее собой зачерненный кружок в разделе пассивных компонентов. Соединитель имеет четыре ортогонально ориентированных вывода, которые можно использовать для подключения четырех проводников по одному с каждой стороны. Соединители создаются автоматически при перетаскивании проводника от какого-либо вывода компонента к данному проводнику. Клавиша мыши отпускается после появления на изображении проводника кружочка в точке подключения.

Проводники можно выполнить линиями разного цвета командой **Wire color** (Цвет проводника) из меню **Circuit,** либо выбором цвета после двойного щелчка на изображении проводника. Если данным проводником к контрольной точке подключается вход виртуального осциллографа или логического анализатора, то этим же цветом будет окрашена соответствующая сигналограмма на экране прибора.

Если необходимо удалить проводник, то следует указать вывод компонента, куда подключен этот проводник, так, чтобы этот вывод выделился, и, нажав левую клавишу мыши, оттащить проводник от вывода и отпустить клавишу. Если убрать компонент или прибор с рабочего поля в магазин компонентов или на панель приборов, то все линии связи, подключенные к этому объекту также исчезнут.

Линия связи может получиться неровной из-за неточного по горизонтали и вертикали взаимного позиционирования компонентов и приборов. Для выравнивания линий связи можно выделить объект и скорректировать его позицию с помощью мыши.

Один из узлов схемы, потенциал которого принимается равным нулю, должен быть

заземлен путем подключения к нему **УГО Ground** (Земля) из раздела пассивных компонентов магазина.

- **Подключение и настройка измерительных приборов**

Измерительные приборы, точнее их пиктограммы, хранятся на панели приборов выше рабочего поля монтажного стола. Для подключения нужного прибора его пиктограмма с помощью мыши переносится на рабочее поле. Зажимы прибора подключаются к контрольным точкам схемы аналогично остальным соединениям в схеме. Каждый из приборов имеется на панели в единственном экземпляре, поэтому при переносе его на рабочее поле место на панели остается свободным.

Кроме того, в магазине компонентов имеются вольтметры и амперметры, количество которых не ограничено.

После подключения каждого прибора выполняется его настройка под конкретный режим анализа. С этой целью изображение прибора увеличивают путем выделения прибора и выполнения команды **Zoom** (Лупа) из меню **Circuit.** К тому же результату приводит двойной щелчок на изображении прибора. На увеличенном изображении прибора устанавливают в необходимые положения органы его управления. Все приборы, перечисленные в разделе 1, имеют собственные органы управления, однако для их настройки используются одни и те же принципы:

1. для выбора органа управления, выполненного в виде клавиши или кнопки, и его установки используется щелчок мышью после вывода указателя на клавишу или кнопку;

2. чтобы изменить значение и единицу размерности вырабатываемой или измеряемой величины, производится щелчок на стрелках «вверх» и «вниз», расположенных сбоку от обозначения единицы размерности или ее значения. Иногда можно щелкнуть в свободном поле и нажать стрелочные клавиши «вверх» или «вниз» на клавиатуре для изменения значения величины. На некоторых приборах можно просто напечатать новое значение величины.

Мультиметр (рис.4) применяется для измерения напряжения, тока, сопротивления или ослабления сигнала в децибелах между двумя контрольными узлами схемы.

Рис.4 Мультиметр

Режим работы мультиметра задается клавишами. Выбор клавиши осуществляется щелчком мыши на клавише верхнего ряда:

A - измерение тока (режим амперметра). Для измерения тока мультиметр должен быть включен своими зажимами «+» и «-» в разрыв цепи, где измеряется ток. Щелчком на клавишах «~» или «-» задается режим измерения либо переменного (его среднего значения), либо постоянного тока. По умолчанию мультиметр в режиме амперметра обладает внутренним сопротивлением 1 мОм (10^{-3} Ом), однако сопротивление можно изменить щелчком на клавише SETTINGS (Установки) мультиметра. Следует иметь в виду, что включение низкоомного прибора в высокоомную схему может привести к снижению математической точности моделирования;

V - измерение напряжения (режим вольтметра). Для измерения напряжения между двумя точками схемы к ним необходимо подключить соответствующие прибора. Режим измерения и внутреннее сопротивление прибора (по умолчанию 1 Мом =10^{6}) можно задать щелчком на клавишах «~», «-» и SETTINGS;

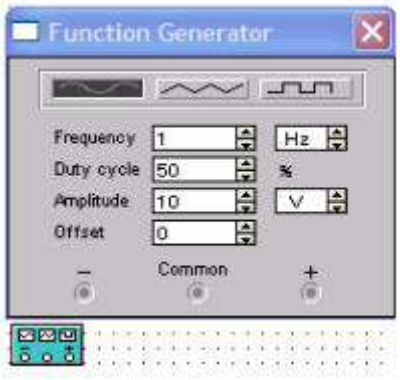
П - измерение сопротивления (режим омметра) выполняется между двумя точками схемы , к которым подключены зажимы прибора. Правильный результат измерения получается , если задан режим в децибелах между двумя точками«-», а измеряемая цепь не заземлена и не соединена с источником напряжения;

dB - измерение ослабления напряжения в децибелах между двумя точками схемы в соответствии с выражением

dB=20lg((V**n**-V**m**)/Vст)

где $\mathbf{V_n}$ и $\mathbf{V_m}$ - напряжения в точках n и m , $\mathbf{V_{CT}}$ - стандартное напряжение (по умолчанию 1В)

Функциональный генератор (рис. 5) является источником напряжения специальной формы Форма сигнала задаётся клавишами верхнего ряда. Частота сигналов, коэффициент заполнения , амплитуда и смещение базовой линии (постоянной составляющей напряжения) устанавливаются клавишами , расположенными ниже.



Изменение значений указанных параметров производится стрелочными кнопками сбоку от задаваемых значений:

FREQUENCY (Частота) выходных сигналов может устанавливаться в пределах от 1 Гц (Hz) до 999 МГц;

DUTY CYCLE (Коэффициент заполнения) задает для прямоугольного сигнала отношение в % длительности импульса с высоким рабочим уровнем к периоду, а для треугольного сигнала - отношение длительности поло: линейно нарастающего фронта к периоду. Коэффициент заполнения 50% соответствует симметричной форме сигнала. Форма синусоидального сигнала не изменяется с изменением данного коэффициента;

AMPLITUDE (Амплитуда) задает максимальное значение выходного напряжения, отсчитываемое от базовой линии (смещения по постоянному току) в предположении, что выходной сигнал снимается с зажимов **COM** (Общий) и «+» (или COM и «-»). Если выходной сигнал снимается с зажимов «+» и «-», то амплитуда удваивается;

OFFSET (Смещение) задает величину постоянного смешения выходного сигнала - его базовой линии в пределах от **-999** кВ до **+999** кВ.

Отметим, что режим источника напряжения, реализуемый функциональным генератором, предполагает, что генерируемый сигнал свободен от искажений, шумов и пульсаций, свойственных реальным генераторам с реальной нагрузкой. Параметры генерируемого сигнала в режиме источника напряжения не зависят от величины подключенной нагрузки. Один из выходных зажимов генератора обычно подключается к земле.

Осциллограф имеет два канала **A** и **B** и может использоваться для визуального наблюдения двух сигналов одновременно. Осциллограф можно также использовать в режиме , когда один сигнал откладывается в качестве аргумента по оси абсцисс, а второй в качестве функции по оси ординат. Кроме входных зажимов каналов **A** и **B** осциллограф снабжен клеммой заземления **(GROUND)** и входным зажимом запуска развертки **TRIGGER** (Запуск) (рис.6).

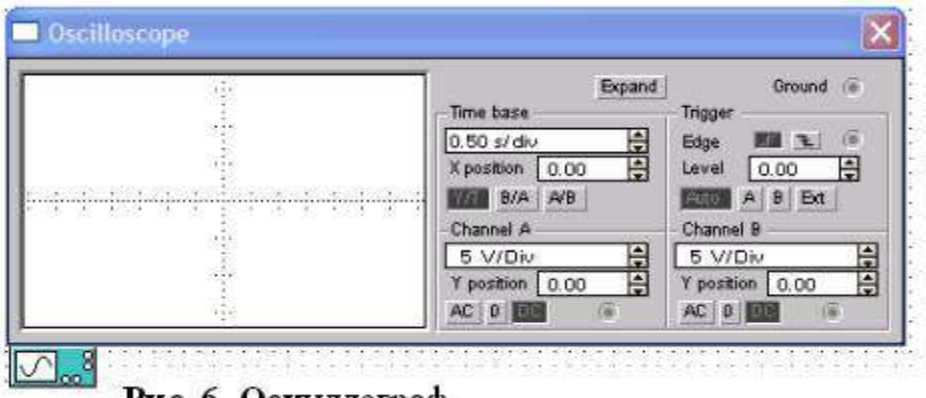


Рис. 6 Осциллограф

Управление осциллографом заключается в установке режимов его работы с помощью клавишных переключателей, на которых для их выбора выполняется щелчок мышью.

Щелчок на одной из кнопок **Y/T, B/A, A/B** определяет оси **X** и **Y** В режиме **Y/T** по оси **X** откладывается время , а по оси **Y—** напряжение сигналов **A** и **B.** Масштабирование оси **X** осуществляется в этом режиме кнопками «Т» и «4-» сбоку от табло текущего масштаба **TIME BASE** (s/div - - с/дел). Скорость развертки можно изменять от 0,1 не до 0,5 с на одно деление шкалы. Масштаб по оси **Y** устанавливается для каждого канала отдельно соответствующими кнопками сбоку от указателей текущего масштаба и может изменяться от 0,01 мВ/дел до 50 В/дел. Можно также задать смещение начальной точки развертки по осям X и Y, причем смещение по оси X определяется кнопками **X POS** для обоих каналов одновременно. Если **X POS** установлено 0.00, то развертка начинается от начала экрана осциллографа, положительное ненулевое значение **X POS** сдвигает начало развертки вправо, а отрицательное - влево. Смещение сигналов по оси **Y** задается органами **Y POS** обоих каналов в пределах от -3 до 3 делений и используется для того, чтобы раздвинуть по вертикали изображения сигналов **A** и **B.** В режимах **B/A** и **A/B** шкала по оси **X** определяется установленным масштабом по каналу **A** и **B** соответственно.

Исследуемые входные сигналы могут подаваться в каналы А и В через открытые для постоянной составляющей входы и через закрытые входы. Выбор типа входа осуществляется щелчком на одной из клавиш **AC, 0** или **DC:**

AC (Alternating Current - переменный ток) игнорирует постоянную составляющую сигнала. Данный режим эквивалентен подаче сигнала на вход канала через конденсатор большой емкости;

O (нуль) заземляет вход канала. Этот режим используется для коррекции положения луча на экране осциллографа в отсутствии входного сигнала;

DC (Direct Current - постоянный ток) показывает на экране сигнал с учетом его постоянной составляющей. Следует иметь в виду, что схемное подключение разделительного конденсатора между контрольной точкой и входом осциллографа является некорректным, так как конденсатор воспринимается программой как компонент с неподключенным выводом.

Елок **TRIGGER** осциллографа определяет режим запуска горизонтальной развертки. Кнопки **EDGE** (фронт) **«Г»** и **«|_»** устанавливают запуск развертки соответственно по фронту и спаду запускающего сигнала. **LEVEL** (уровень) определяет пороговое напряжение, превышение которого приводит к запуску развертки. Кнопки **AUTO, A, B** и **EXT** указывают, от какого сигнала производится запуск развертки:

AUTO - автоматический запуск;

A, B - запуск от входных сигналов **A и B** соответственно;

EXT - запуск от внешнего сигнала, подаваемого на зажим **TRIGGER.** При подключении осциллографа его зажим **GROUND** следует подключать к общей точке (земле) схемы.

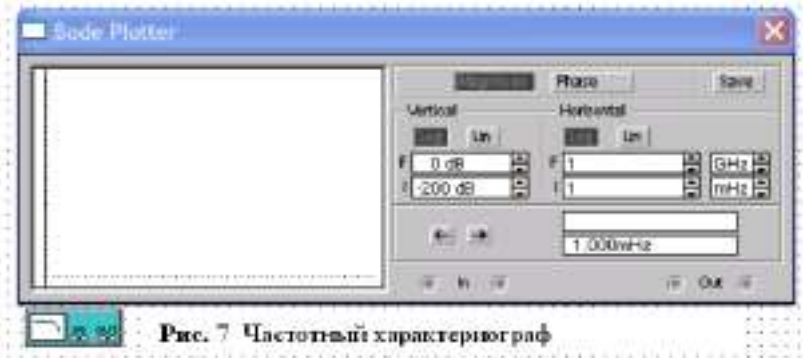


Рис. 7 Частотный характериограф

Частотный характериограф (Bode plotter - построитель диаграмм Боде) служит для вывода результатов моделирования в частотной области, изображая амплитудно-частотную характеристику (АЧХ) и фазо-частотную характеристику (ФЧХ). Прибор генерирует гармоническое напряжение с линейно изменяющейся частотой на своем зажиме **IN** и измеряет отношение напряжений на подключенных к схеме зажимах **OUT** (Выход) и **IN** (Вход), или же разность их фаз. Частоты всех источников переменного тока в схеме в процессе расчета игнорируются, однако схема должна включать хотя бы один такой источник.

Характериограф выводит на свой экран в функции частоты либо отношение амплитуд, выраженное в относительных единицах или децибелах, или разность их фаз в градусах. Режим работы прибора задается клавишами и стрелочными кнопками «Т», «4» (рис.7):

щелчок на клавише **MAGNITUDE** или **PHASE** устанавливает режим измерения АЧХ или ФЧХ соответственно;

щелчок на клавишах **LOG** или **LIN** устанавливает логарифмический или линейный масштаб вертикальной **VERTICAL** и горизонтальной **HORIZONTAL** осей графика на экране прибора;

шкалы осей задаются начальными **(I)** и конечными **(F)** значениями величин. По оси **X** всегда откладывается частота в Гц, а но оси **Y** откладывается безразмерная величина **(LIN)** или децибелы **(LOG)** для отношения напряжений и градусы для разности фаз;

для точного отсчета результата измерений можно использовать визир прибора, управляя его позицией с помощью кнопок с изображениями горизонтальных стрелок «^», «^» в нижней части лицевой панели характериографа. Визир можно также перемещать с помощью мыши, для чего визир (в виде крестика в левом нижнем углу экрана прибора) следует перетащить в нужную точку на графике АЧХ или ФЧХ. Отсчет измеряемой величины появляется на цифровом табло в правой нижней части прибора.

Логический преобразователь не имеет физического прототипа и является чисто компьютерным инструментом для выполнения преобразований цифровой схемы и ее логического описания. Изображен прибор (рис. 8) шаблон содержит таблицы истинности в левой части, клавиши управления в правой части и дисплей для отображения логического выражения в нижней части лицевой панели. Анализируемая или синтезируемая цифровая схема может содержать до восьми входных переменных (аргументов) и один выход (функцию). Аргументы обозначаются буквами **A..H,** а функция - **OUT.** Режим преобразования назначается клавишами:Преобразование схемы в таблицу истинности. В этом режиме входы схемы подключить к зажимам **A...H,** а выход - к зажиму OUT. После щелчка на клавише шаблон таблицы истинности заполняется

значениями переменных **(0 и 1).** Всего в таблице образуется 2^n строк, где n- число аргументов;

5. преобразование таблицы истинности в логическое выражение. Для создания таблицы истинности (если она еще не образована предыдущим преобразованием) необходимо щелкнуть на нескольких входных зажимах, соответствующих аргументам A...H,

■+ TofT

Появляется полный набор комбинаций аргументов, против которых в колонке **OUT** функции надо напечатать значения функции **(O, 1** или **X,** причем **X** обозначает неопределенное значение функции). После щелчка на клавише в нижней части прибора появляется алгебраическая запись функции, в которой вместо обычного знака отрицания в виде черты над символом используется знак «'», проставляемый за символом;

6. преобразование таблицы истинности в минимизированное логическое выражение. Минимизация в **ПЭВМ** проводится методом Мак-Класки, допускающим по сравнению с методом карт

Карно большее число аргументов. Следует заметить, что минимизация может потребовать значительного объема памяти. Результат минимизации выводится в нижней строке прибора;

7. преобразование логического выражения в таблицу истинности. Перед щелчком на этой на этой клавише следует ввести

алгебраическое выражение в нижней строке прибора. Результат преобразования автоматически записывается в шаблон таблицы истинности ;

8. преобразование логического выражения в цифровую комбинационную схему. Напечатанное предварительно выражение после щелчка на этой клавише преобразуется в схему

на рабочем поле монтажного стола;

НАМИ * создание цифровой схемы в функциональном базисе **И, НЕ,**

NAN U распространенном в инженерной практике.

Измерение постоянных токов и напряжений предусмотрено в программе возможностью подключения амперметров и вольтметров из магазина компонентов. Их количество не ограничено, поэтому при перемещении этих приборов на рабочее поле монтажного стола их изображения сохраняются в исходных позициях.

Практическая часть.

Вся информация программы **Electronics Workbench,** выводимая на экран **ПЭВМ,** располагается в окнах, которые можно перемещать, прокручивать (просматривать), изменять размеры и закрывать подобно другим окнам операционной среды **WINDOWS.** В окнах размещаются рабочее поле, магазин компонентов, панель инструментов, описание схемы и пр.

Если нужное в данный момент окно накрыто другими окнами, можно вывести его поверх остальных щелчком на титульной верхней полосе окна. Окно можно перетащить в другое место экрана ,установив указатель на титульной, полосе, нажав левую клавишу мыши, переместив окно и отпустить клавишу , когда окно окажется на новом месте. Двойной щелчок на кнопке в верхнем углу окна приводит к его закрытию. Для изменения размера окна следует перетащить его сторону или угол после появления на месте указателя двунаправленной стрелки. Для прокрутки окна можно перетащить кнопку, расположенную на правой стороне окна или полоску в нижней части окна. Щелчок внутри области частично скрытого окна делает это окно активным и выводит его наверх. Следует иметь в виду, что общая площадь рабочего поля монтажного стола в четыре раза больше той, которая выводится на экран, и для того, чтобы разместить дополнительные компоненты или просмотреть полностью рабочее поле надо использовать кнопки прокрутки, расположенные с правого и нижнего краев окна.

Открытие ранее созданного и сохраненного схемного файла осуществляется командой **Open** (Открыть) из меню **File** (Файл). При этом появляется диалоговое окно, из которого выбирается имя схемного файла, после чет производится щелчок на кнопке **ОК.** К такому же результату приводит **Ctrl+O.**

В необходимых случаях на рабочем поле можно поместить текстовую информацию: позиционные обозначения, типы и значения параметров компонентов; краткое описание схемы; значения величин при настройке измерительных приборов. В большинстве случаев вставка текста назначается автоматически после вывода указателя на нужную позицию и щелчка мышью. Для перемещения указателя можно также использовать стрелочные клавиши и клавишу табуляции **Tab** на клавиатуре **ПЭВМ.**

Информация или комментарии к схеме печатается в окне описания **(Description Window).** Окно описания открывается командой Description из меню Window или сочетанием клавиш **Ctrl+D.** Если нужно показать окно описания следует перетащить его к верхнему или нижнему краю рабочего поля и выбрать команду **Arrange** (Упорядочивание) из меню **Window** Эта команда (или сочетание клавиш **Ctrl+W)** используется для упорядочивания составных частей монтажного стола. Указанная команда заставляет все открытые окна увеличиваться , насколько это возможно в конкретной ситуации. Для изменения порядка расположения частей монтажного стола следует перетащить эти части примерно туда , где их хотелось бы видеть, и выбрать вновь команду **Arrange** из меню **Window.** Способ упорядочивания окон зависит от их текущей позиции.

Перед проведением анализа собранную схему можно вывести на передний план командой **Circuit** из меню **Window**

После создания схемы и подключения приборов можно назначить директивы анализа - опции (по умолчанию набор опций устанавливается автоматически, и зависят от типов подключенных измерительных приборов). Директивы анализа задаются командой **Analysis Options (Ctrl+Y)** и предназначены для определения видов проводимых программой расчетов. Указанная команда выводит на экран диалоговое окно со следующими опциями; **Analysis Type**

	Вид анализа
Transient	Переходной процесс
Steady state	Установившийся режим
Active component	Активный элемент

Assume linear operation Линеаризованный режим
Oscilloscope Display Экран осциллографа
Pause after each screen Пауза после каждого экрана
Store results for all nodes Запоминание результатов для всех узлов
Tolerance Погрешность
Points per cycle Точек на период
Bode plotter points per cycle Точек частотной характеристики **Temporary file size for simulation** Размер временного файла для моделирования

Перечисленные опции относятся к моделированию аналоговых электронных схем.

Исследование цифровых схем

Исследование цифровых схем, состоящих из логических комбинационных и последовательных узлов, проводится на функциональном и принципиальном уровнях. При функциональном анализе все цифровые компоненты полагаются по умолчанию идеальными, не обладающими задержками переключения и не требующими источников питания. Часто такого анализа оказывается достаточным для оценки функционирования спроектированного устройства. Однако, если необходим более детальный анализ поведения схемы, или если схема включает наряду с цифровыми и аналоговые компоненты, то следует проводить моделирование на принципиальном уровне с указанием конкретных типов микросхем и цепей их питания.

Исследование обычно проводится во временной области с использованием осциллографа **и/или** логического анализатора. Проверка комбинационных схем без элементов памяти может осуществляться также с помощью логического преобразователя.

Синтез комбинационной схемы начинается, как правило, с таблицы истинности, в которой каждому набору аргументов ставится в соответствие то или иное значения **(0** или **1)** булевой функции. Таблица истинности заполняется на экране раскрытого изображения логического преобразователя. Количество в таблице определяется количеством аргументов, которое назначается на кнопках, расположенных над формируемой таблицей истинности, и равно 2^n, где **n-** число аргументов. Для каждого набора аргументов после позиционирования указателя с помощью клавиатуры вводится значение функции.

Таблица истинности может быть преобразована в алгебраическое выражение или в минимизированное алгебраическое выражение щелчком на соответствующей клавише раскрытого изображения преобразователя .

Образовавшееся под таблицей истинности выражение, в котором знак отрицания изображается знаком «'» за соответствующим символом, можно далее реализовать на логических элементах **И, ИЛИ, НЕ,** или же только на логических Элементах **И, НЕ** щелчком на соответствующей клавише преобразователя Автоматически синтезированная схема появляется на рабочем поле монтажного стола.

Возможно также обратное преобразование алгебраического выражения или схемы в таблицу истинности, например при проверке правильности проведенного формального синтеза схемы. Эти преобразования назначаются также щелчками на соответствующих клавишах после прорисовки комбинационной схемы на рабочем поле монтажного стола.

Цифровая схема создается на монтажном столе в последовательности, описанной в разделе 2. Для извлечения компонента выбирается тот или иной раздел магазина компонентов и нужный объект перетаскивается на рабочее поле с помощью мыши.

Если перетаскивается компонент в виде корпуса микросхемы, то перед выполнением межсоединений двойным щелчком на изображении корпуса

посредством открывающегося диалогового окна назначается тип микросхемы по ее функциональному признаку. Вторично произведенный двойной щелчок вызывает следующее диалоговое окно, предлагающее выбрать

раздел библиотеки

микросхем **(library).** Всего имеется четыре раздела: **«cmos» (КМОП), «default»** (по умолчанию), **|misc»** (смешанные), **«ttl»** (ТТЛ). По умолчанию компонент полагается идеальным.

Выбор разделов **«cmos»** и **«ttl»** предлагает уточнить через диалоговое окно технологическую модификацию микросхемы (напряжение питания, наличие мощного выхода **«BUF»,** открытый коллектор **«OC»** или сток **«OD»,** скоростные свойства - **«HC»** (High-speed CMOS), мощная **ТТЛШ - «LS»** (Low-power tky)). Вызов функции **«HELP»** ЛОЩЬ) клавишей **F1** или выбором меню выдает дополнительную рмацию по выделенному оненту.

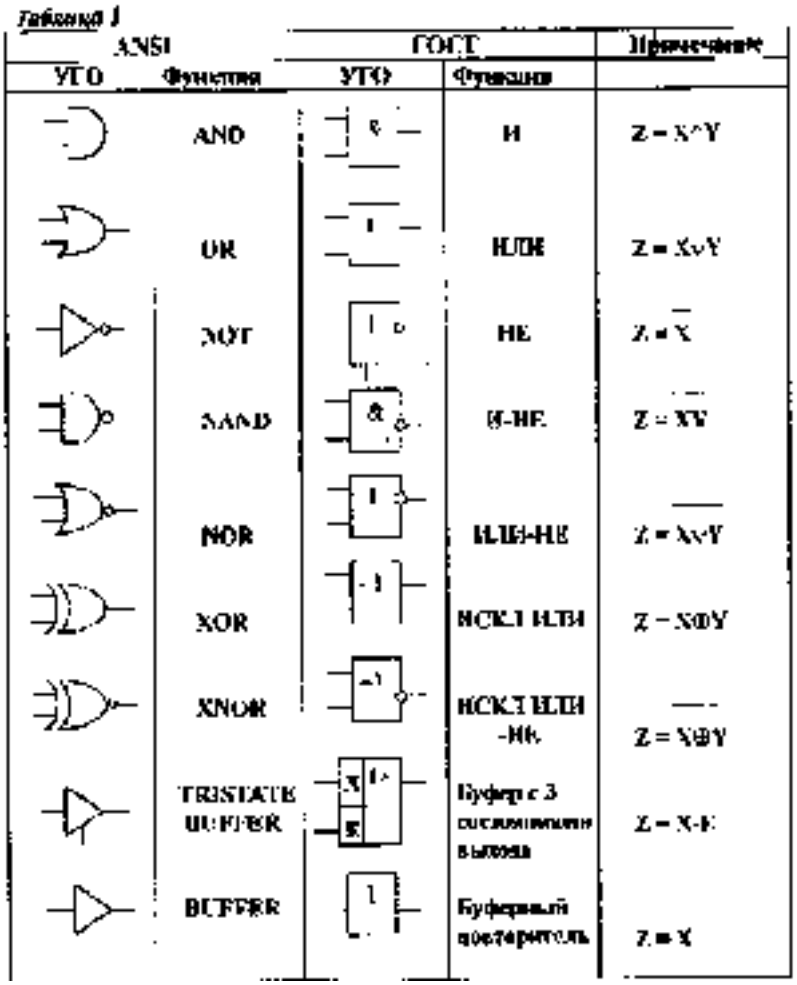
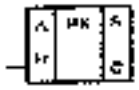
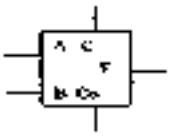
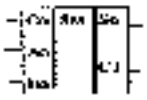
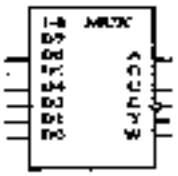
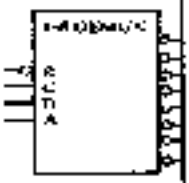
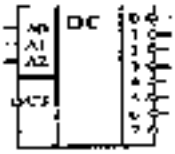
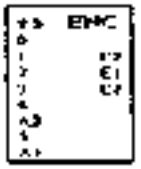
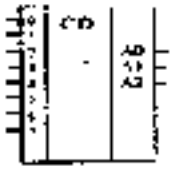
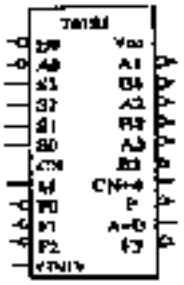


Таблица 2

ANSI		ГОСТ		Примечания
УГО	Функция	УГО	Функция	
	Half Summer (HS)		Полусумматор	A,B - операнды Σ - сумма Co - перенос
	Summer (SM)		Сумматор	A,B – операнды C – вход переноса Σ - сумма Co - выход переноса
	Multiplexer (MUX)		Мультиплексор	D0 - D7 - информационные входы A,B,C - адресные входы E – вход разрешения Y - выход W - инверсный выход
	Decoder/ DemulNiplexer (DEC/DEMUX)		Дешифратор	A,B,C – адресные входы E - вход разрешения
	Encoder (ENC)		Шифратор приоритета	0 – 7 - информационные входы E0 - E2 - входы разрешения A0 - A2 - выходы
	Arithmetic logic unit (ALU)		Арифметико-логическое устройство	Аналог К155ИП3

Electronics Workbench располагает обширной библиотекой моделей цифровых компонентов, **УГО** которых **могут** изображаться двояко: либо в соответствии со стандартом **ANSI** (American National Standard Institute), либо в соответствии с европейским стандартом **DIN** (второй весьма близок к **ГОСТам РФ).** Система обозначений назначается при инсталлирования программы.

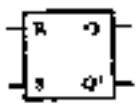
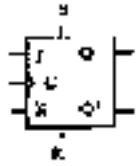
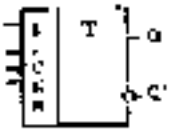
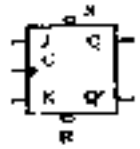
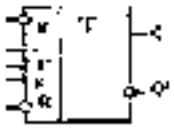
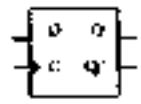
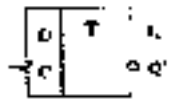
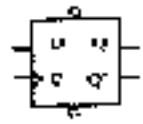
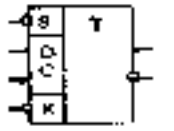
Раздел **Logic Gates** включает **УГО** вентилей (логических элементов) (табл.1) и изображения корпусов микросхем с обозначением функции.

Двойной щелчок на **УГО** вентиля приводит к появлению запроса о количестве его входов.

Раздел **Digital** предлагает комбинационные функциональные узлы, назначение которых обозначено на изображениях корпусов микросхем: **MTJX** (Мультиплексоры), **DEC/DEMUX** (Дешифраторы/Демультиплексоры), **ENCODERS** (Шифраторы), **ARITHMETIC**

(Арифметические узлы). Кроме того имеются **УГО** полусумматора **(HS)** и полного одноразрядного сумматора **(SM).** В разделе магазина также сосредоточены **УГО** различных триггеров, а также изображения корпусов микросхем с обозначением функции. Двоимой щелчок на изображении корпуса предлагает перечни: **FLIP-FLOPS** -триггеров, **COUNTERS** - счетчиков, либо **SHIFT REGS** - сдвигающих регистров, из которых выбираются конкретные типы и серии микросхем. В табл.2 приведены типовые представители данного раздела цифровых компонентов а в табл.3 **УГО** триггеров .

Таблица 3

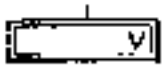
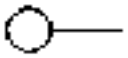
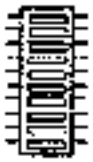
ANSI		ГОСТ		Примечание
УГО	Функция	УГО	Функция	
	RS flip-flop		RS-триггер	
	JK flip-flop		JK-триггер	Рабочие уровни асинхронных входов S,R - - высокие
	JK flih-flop		JK-триггер	Рабочие уровни S,R – низкие
	D flip-flop		D-триггер	
	D flip-flop		D-триггер	Имеются асинхронные входы S и R с низкими рабочими уровнями

Разделы **IC** предлагает к выбору серии микросхем **("74xxxx - ТТЛ, 4xxxx -КМОП,** знаком «x» обозначены переменные позиции обозначения микросхем). Таблицы соответствия отечественных микросхем и их зарубежных аналогов даны в приложениях В и Г.

Раздел **Indicators** магазина компонентов содержит **УГО** приборов индикации, световой и звуковой сигнализации в соответствии с табл. **4.**

В разделе гибридных (смешанных) микросхем имеются цифро-аналоговые и аналого-цифровые преобразователи, а также мультивибратор и таймер **555** (аналог отечественной микросхемы **КР1006ВИ1)** , которые можно использовать для создания схем в встроенными тактовыми генераторами и схем для связи цифровых узлов с аналоговыми. Анализ цифровых схем проводится преимущественно с помощью генератора двоичных слов

Таблица 4

УГО	Функция	Примечание
	Вольтметр	После двойного щелчка устанавливается сопротивление и шкала вольтметра и амперметра
	Амперметр	
	Лампа накаливания	
	Световой пробник	Устанавливается цвет свечения: красный, зеленый, синий
	Семисегментный дисплей	
	Декодирующий семисегментный дисплей	Преобразует 4-разрядный двоичный код в символ шестнадцатиричного кода (0...9, A...F)
	Запоминающее устройство	Записывает значения напряжений на своих 8-ми входах в функцию времени в файл, имя которого назначается после двойного щелчка
	Головка громкоговорителя	После двойного щелчка назначается частота звука, порог срабатывания и выходная мощность. Срабатывает при превышении порога срабатывания
	Полосковый дисплей	Содержит матрицу из 10 светодиодов (аноды - слева, катоды - справа). Каждый светодиод включается при протекании тока
	Декодирующий полосковый дисплей	Имеет встроенный АЦП. В зависимости от напряжения между выводами светится младшие n светодиодов. Каждый светодиод с номером n включается при превышении напряжением порогового значения U_n: $U_n = U_1 + \frac{U_9 - U_1}{9}(n-1)$, где U_1 и U_9 – напряжения включения младшего и старшего сегментов (по умолчанию 1В и 10В)

(Word generator) и логического анализатора **(Logic analyzer)** , который не был описан. Генератор слов позволяет задать по тактам повторяющиеся с периодом 16 тактов 8 различных входных логических сигналов для проверяемой схемы. Логический анализатор позволяет наблюдать до восьми логических сигналов одновременно в восьми узлах схемы, каковыми могут быть

входы и выходы схемы, а также любые внутренние узлы в схеме. Если назначить цвета проводников, которыми входы подключены к схеме (см. п.2.2), то именно этими цветами будут окрашены соответствующие сигналограммы на экранах этих приборов. Анализ схемы при низкой

тактовой частоте может проводиться также при помощи световых пробников , подключаемых к интересующей пользователя узлами схемы . В отдельных случаях может оказаться полезным подключение головок громкоговорителей, падающих звуковой сигнал при превышении напряжением в контролируемом узле их порогов срабатывания.

Документирование практической работы

Документирование практической работы заключается в создании текстового файла отчета о работе, включающего сведения о цели работы, описание аналитической части работы (проектирование и расчет схемы), описание лабораторной установки (схемы и набора измерительных приборов на монтажном столе), экспериментальные данные и выводы. В отдельных случаях требуется выполнить твердую копию отчета путем распечатки файла.

В программе **Electronics Workbench** можно задать распечатку любого или всех окон монтажного стола (окнами являются схема, каждый из раскрытых приборов, описание схемы и пр.). Задание на печать назначается с помощью диалогового окна, вызываемого командой. **Print** из меню **File.** Диалоговое окно предлагает к выбору всю доступную информацию о моделировании, составляющие которой отмечаются щелчком в соответствующей строке. Печать инициируется щелчком на клавише **ОК.**

Можно также включить в файл отчета общий вид экрана **ПЭВМ** после выполнения анализа и отражения результатов на экранах измерительных приборов. Для этого нажатием клавиши **Print Screen** экран **ПЭВМ** копируется в буфер, из которого затем производится его вставка в текстовый файл отчета с помощью панели инструментов редактора **Microsoft Word.**

Практическая работа №43. 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-битныхмикроконтроллеров 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 16-битных 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 портыввода/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.

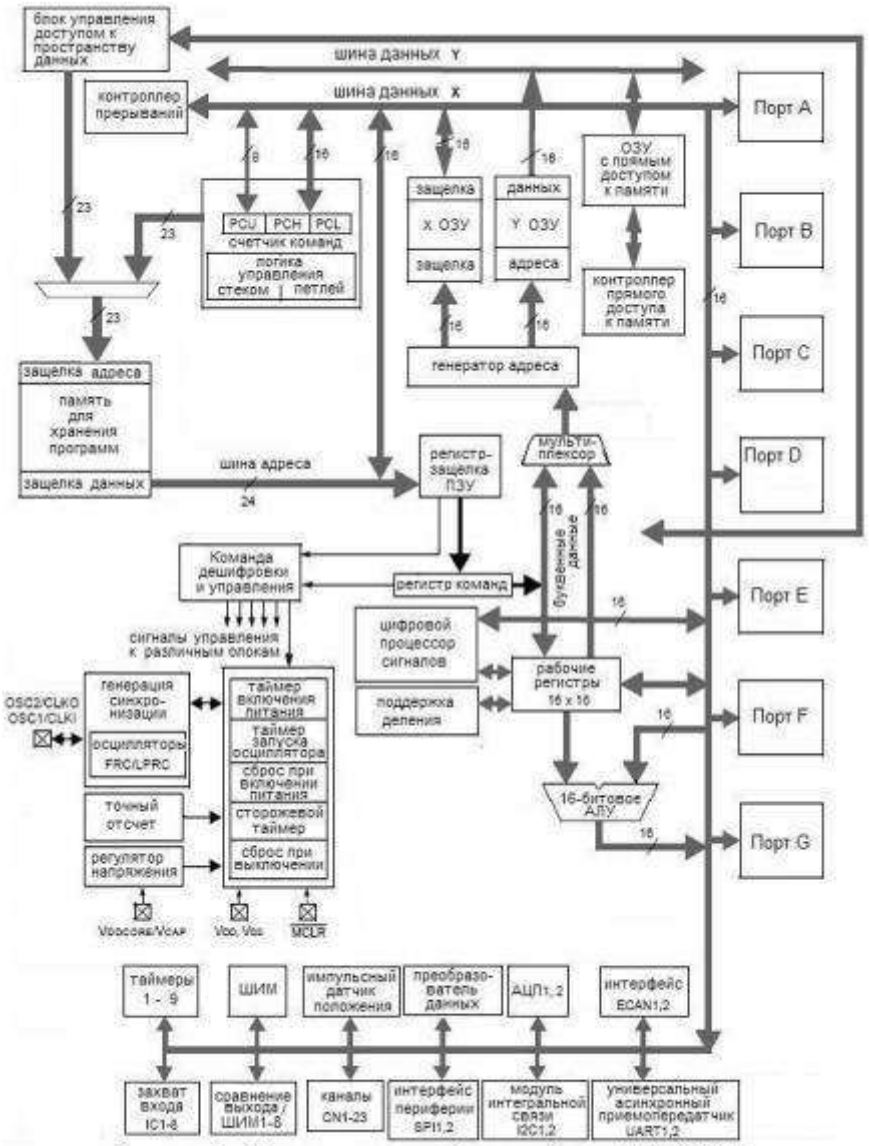


Рисунок 1 - Общая схема устройств семейства dsPIC/PIC24

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.*

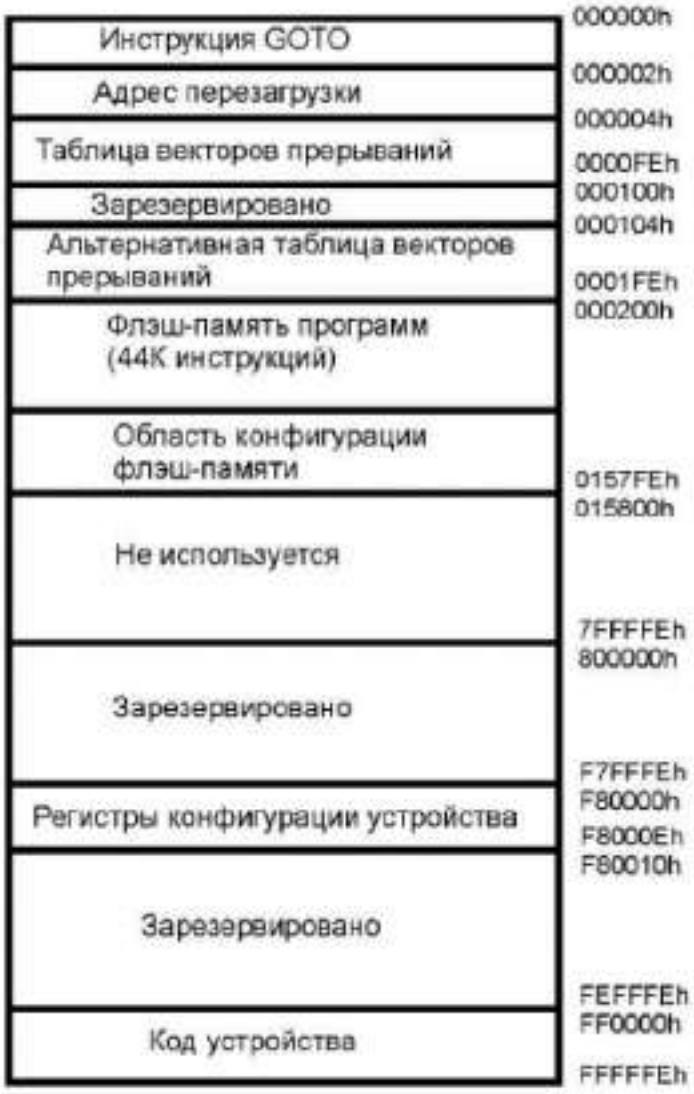


Рисунок 2 - Организация памяти программ ЦСП семейства *dsPIC33*

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 SpaceVisibility*) - 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 №44. Коммуникационные микроконтроллеры (цифровые сигнальные процессоров (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.. >.

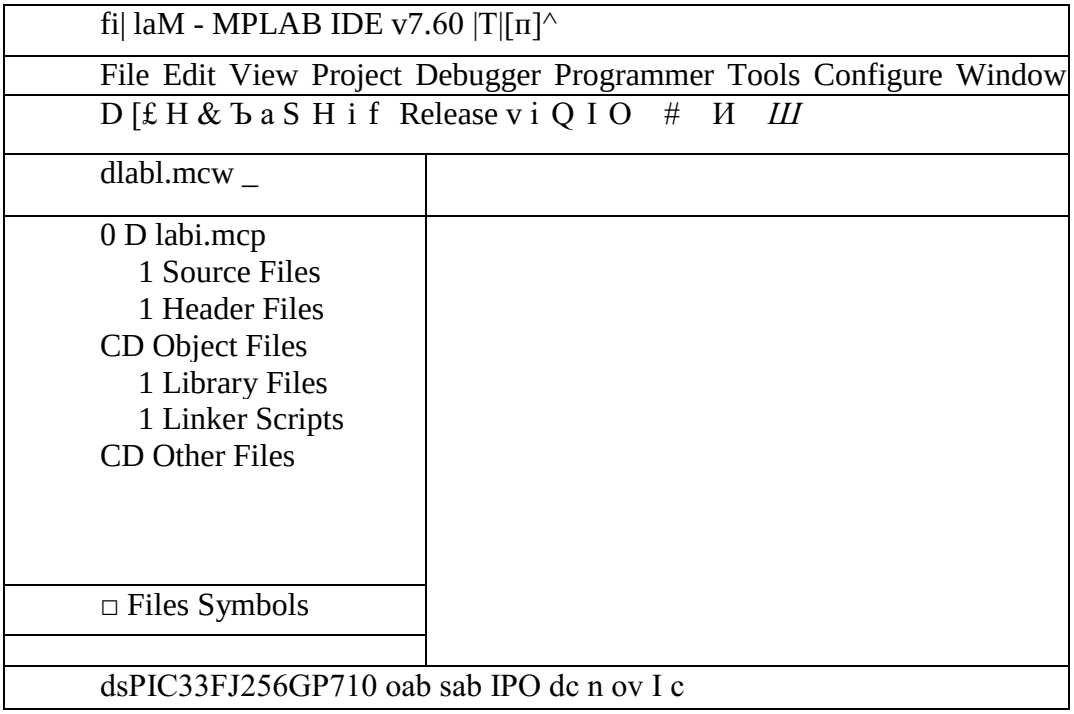


Рисунок 4 - Вид программы после создания нового проекта

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.

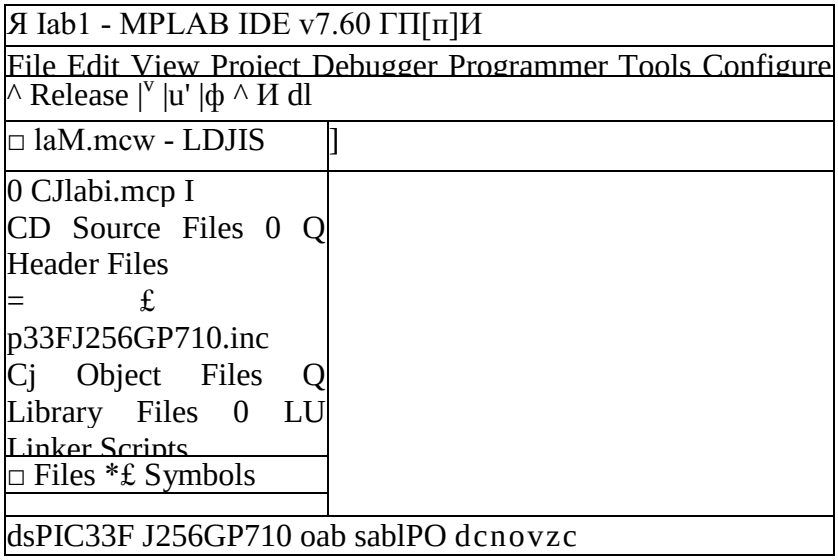


Рисунок 5 - Вид программы после подключения библиотек и упаковочных файлов процессора

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:      .glob       reset
        al
            .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.*

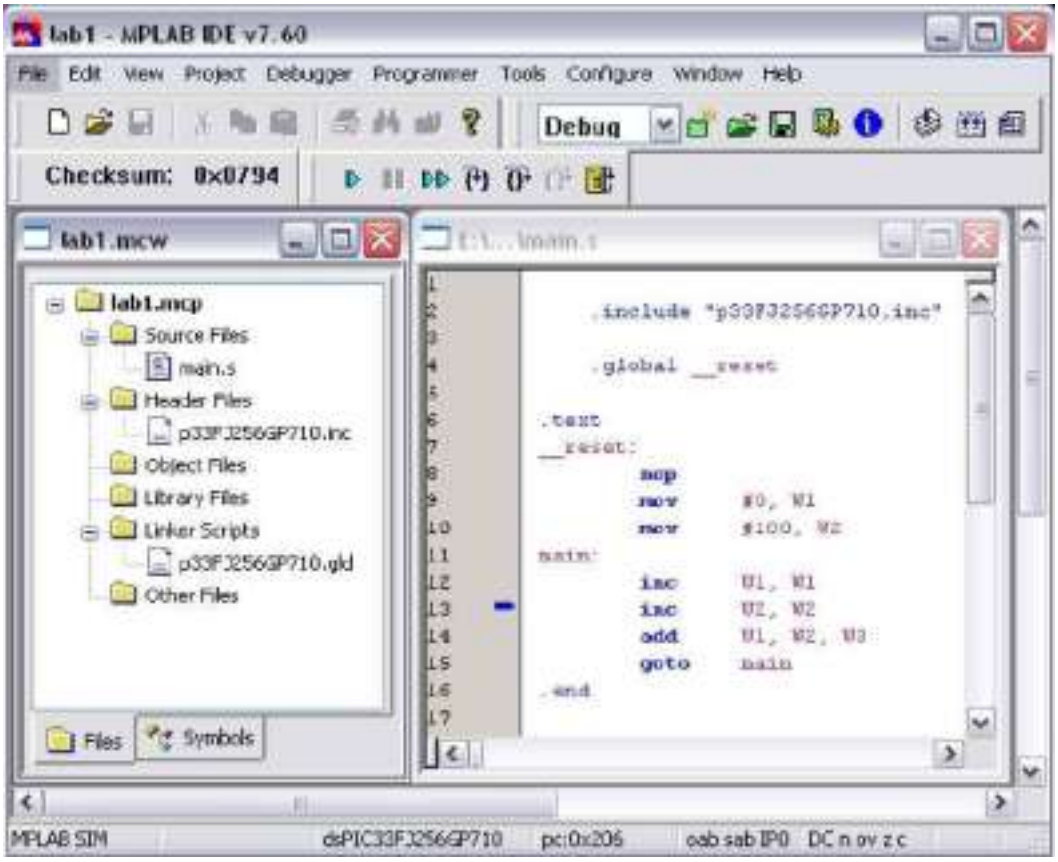


Рисунок 6 - Создание файла с исполняемым кодом

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 №45. 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.

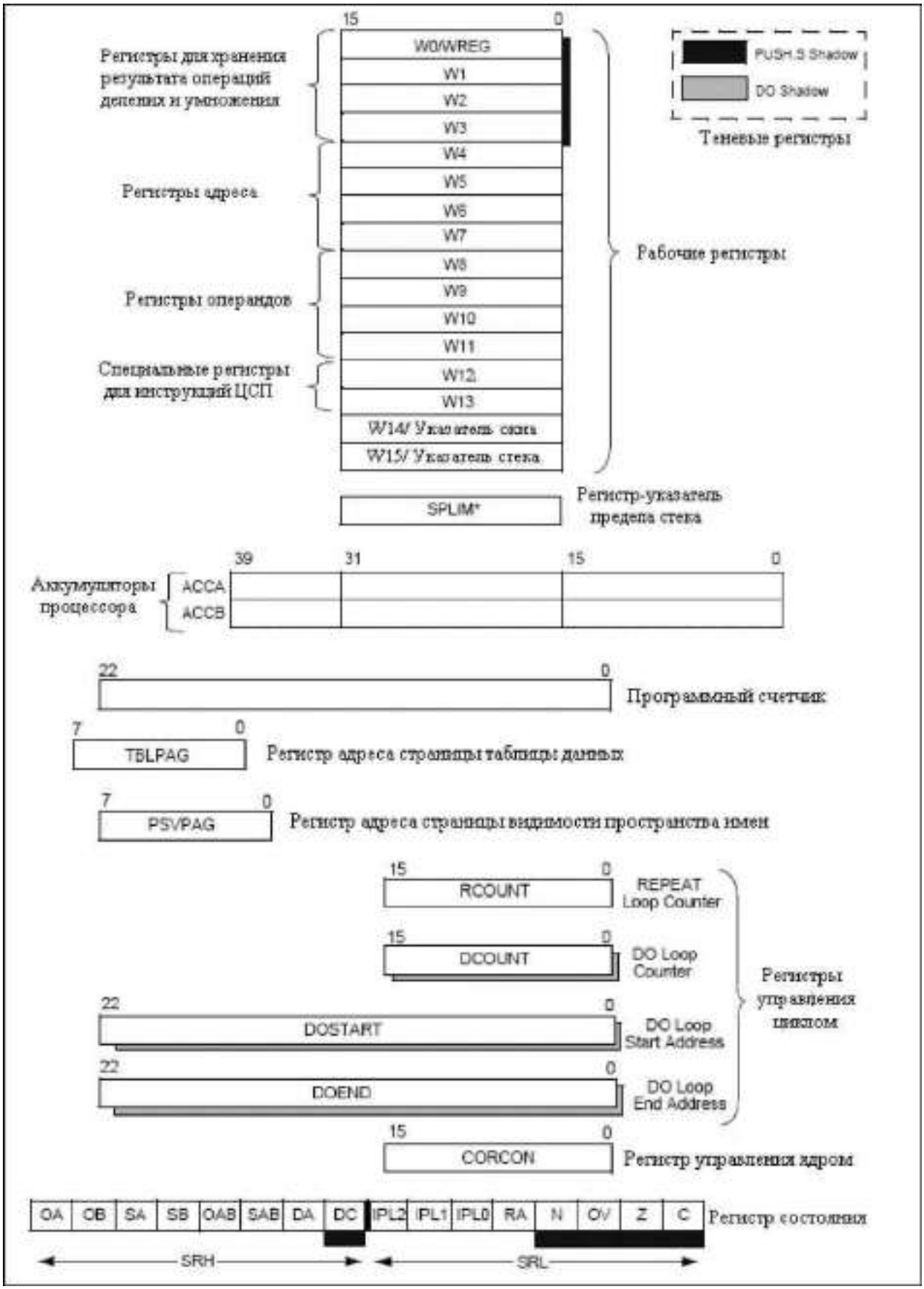


Рисунок 8 - Регистры микропроцессора

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
--
                mo          #0x1111, W1
                mo          W1, a
                mo          #0x2222, W1
                mo          W1, b
                mo          #0x3333, W1
                mo          W1, c

 ; - Реализация логической функции

                mo           a, W1
                mo          #0xF00F, W2
                an          W1, W2, W3
                mo          b, W1
                mo          c, W2
                an          W1, W2, W4
                ior         W3, W4, W1
                mo          W1, d
                no
 .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 №46. Коммуникационные микроконтроллеры (параметры).

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, [Ws], [W0... [W0... [W0...

Операнды: *Wb* Ws Wd

[Ws++], [Ws--], [+ + Ws], [-W15] W15] W15]

Wd [Wd] -Ws], [Wd++] [Wd--] [+ + Wd] Wd [--Wd]

Операция: Wb +Ws ^

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 №47. Коммуникационные микроконтроллеры (виды).

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 №48. Программирование микроконтроллеров (использование ВУ).

Цель работы: приобрести практические навыки работы со стеком.

Теоретическая часть.

Стеком называют область программы для временного хранения произвольных данных. Данные можно сохранять и в сегменте данных, однако в этом случае для каждой сохраняемой на время единицы данных необходимо заводить отдельную именованную ячейку памяти, что увеличивает размер программы и количество используемых имен. Удобство стека заключается в том, что его область используется многократно. Стек традиционно используется, например, для сохранения содержимого регистров, используемых программой, перед вызовом подпрограммы, которая, в свою очередь, будет использовать эти же регистры процессора. Исходное содержимое регистров извлекается из стека после возврата из подпрограммы. Отличительной особенностью стека является своеобразный порядок выборки содержащихся в нем данных: в любой момент времени в стеке доступен только верхний элемент, т.е. элемент, загруженный в стек последним. Выгрузка из стека верхнего элемента делает доступным следующий элемент.

Элементы стека располагаются в области памяти, отведенной под стек, начиная со дна стека (т.е. с его максимального адреса) по последовательно уменьшающимся адресам. Адрес верхнего, доступного элемента (вершины стека) хранится в регистре-указателе стека *W15*. При запуске программы по умолчанию регистр-указатель стека инициализируется адресом начинала сектора ОЗУ общего назначения *0x800*. Указатель на стек может быть изменен программно, что позволяет реализовывать гибкие системы реального времени.

К основным операциям работы со стеком относятся.

- *PUSH* - помещает на вершину стека *(TOS)* значение операнда. Предыдущее значение, находящееся на вершине стека перемещается на один уровень вниз.
- *POP* - извлекает значение из вершины стека в операнд, перемещает все содержимое стека на один уровень вверх.

Синтаксис написания арифметических инструкций приведен в таблице 9 приложении 1.

Помещение в стек

Примеры помещения в стек:

```
PUSH W1;
```

Помещает в вершину стека значение регистра *W1* (16-ти битный режим).

```
PUSH.D W1;
```

Помещает в вершину стека значения регистров *W1:W2* (32-х битный режим).

Извлечение из стека

Примеры извлечения из стека:

```
POP W1;
```

Извлекает значение из вершины стека в регистр *W1* **POP.D W1;**

Извлекает значения из вершины стека в регистры *W1:W2* (двойное извлечение).

Пример программы, реализующий арифметическую функцию:

*e = (a + b) * (c-1) / (d+2)* с ограничением числа используемых регистров.

```
      .include "p33FJ256GP710.inc"
.equ a, 0x810
.equ b, 0x812
.equ c, 0x814
.equ d, 0x816
.equ e, 0x818
.text
.global ____ reset
_ reset:
      ; - Инициализация переменных
      mov #1, W2
      mov W2, a                  ; a:=1
      mov #2, W2
      mov W2, b                  ; b:=2
      mov #3, W2
      mov W2, c                  ; c:=3
      mov #4, W2
      mov W2, d
      ; - Реализация       арифметической функции
      mov a, W2                  ; W2:=a
      mov b, W4                  ; W4:=b
      add W2, W4, W2             ; W2:=W2+W4
      push W2                    ; поместить в вершину стека значение          W2
      mov c, W2                  ; W2:=c
      mov #1, W4                 ; W4:=1
      sub W2, W4, W2             ; W2:=W2-W4
      pop W4                     ;извлечь из вершины стека значение в          W4
      mul.ss W2, W4,      W2 ; W2:W3=W2*W4
      push.d W2                ;поместить в вершину стека значение W2          :W3
      mov d, W2                  ; W2:=d
      mov #2, W4                 ; W4:=2
      add W2, W4, W4             ; W4:=W2+W4
      pop.d W2                   ; извлечь из стека значение в W2:W3
      repeat #17
```

```
div.s W2, W4          ; W0:W1=W2/W4
mov W0, e             ; e:=W0
nop
.end
```

В рассмотренной программе операции производятся с использованием двух регистров *W2* и *W4*. Использование дополнительных регистров *W0* и *W3* обусловлено спецификой команд MUL.SS *и* DIV.S.

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*. Выберете в меню <*Debugger\Select Tool\MPLAB SIM*>.
4. Создайте схему алгоритма и напишите программу, выполняющую заданную арифметическую функцию, с использованием минимального количества регистров.
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) + 1234
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) + 12345

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. Для чего нужен стек?
2. Что такое вершина стека?
3. Где хранится информация о вершине стека?
4. Назовите основные инструкции работы со стеком?
5. Назовите количество тактов, за которые выполняются операции работы со стеком.
6. В чем отличие операций *PUSH* и *PUSH.D?*
7. Приведите пример неявной (подразумеваемой) адресации в вашей программе.

Практическая работа №49. Программирование микроконтроллеров.

Цель работы: приобрести практические навыки работы с арифметическими командами, командами передачи управления.

Теоретическая часть

Понятие рекуррентной зависимости.

Основным назначением цифровых процессоров является выполнение задач по обработке информации, в частности реализация цифровой фильтрации сигнала. Цифровые фильтры реализуются и выполняются в микропроцессоре по разностному уравнению. В данной работе предложены различные рекуррентные зависимости, которые являются разновидностью разностных уравнений. В программировании рекурсия - вызов функции (процедуры) из неё же самой, непосредственно (простая рекурсия) или через другие функции (сложная или косвенная рекурсия). Основными инструкциями, которыми нужно будет пользоваться в данной лабораторной работе, являются: циклы, операции перемещения, арифметические инструкции. Они были рассмотрены в предыдущих работах.

Расчет зависимости.

Рассмотрим пример программы вычисляющий следующую рекуррентную зависимость:

a(k) = a(k-1) + 2*a(k-2) + a(k-3), k = 1, 2, 3, ... *здесь* a(k) *значение последовательности на* k *шаге,* a(k-1) - *на* предыдущем шаге выполнения программы и т.д.

Код программы:

```
include "p33FJ256GP710.inc"
Определим адреса последовательности a[k]
a[k] a[k-1] a[k-2] a[k-3]
.equ .equ .text
                .global __reset      - 0x800
reset:                               - 0x802
   Определим                         - 0x804
        назначение                   - 0x806
        рабочих
        регистры W2 a_begin, 0x800
        W3 = a(k-1) a_end, 0x806     = a(k)
        W4 = a(k-2)
        W5 = a(k-3)
        W6 - указатель на массив a[k]
```

#a_begin, W6 mov

Инициализация последовательности mov #0, W0

mov	**W0,**	**[W6++]**	**a[k] =**	**0**
mov	**#7,**	**W0**		
mov	**W0,**	**[W6++]**	**a[k-1]**	**= 7**
mov	**#3,**	**W0**		
mov	**W0,**	**[W6++]**	**a[k-2]**	**= 3**
mov	**#1,**	**W0**		
mov	**W0,**	**[W6++]**	**a[k-3]**	**= 1**

```
;       Определяем количество итераций работы        алгоритма
            do          #15, lab_B
;       Чтения из памяти
            mov             #a_begin, W6
            mov         [++W6],   W3
            mov         [++W6],   W4
            mov         [++W6],   W5
            ; Вычисление рекуррентной зависимости mul.su
                        W4,#2, W0
            add         W3, W0,   W1
            add         W1, W5,   W2
;       Запись нового значения в память
            mov             W2, a_begin
            ; Сдвиг информации в памяти mov
                            #a_end, W6
            do          #2,     lab_A
            mov          [—W6], [++W6]
            dec2        W6,       W6
lab_A: nop
lab_B: nop
.end
```

Данную программу можно записать более компактно:

Код программы:

```
.include "p33FJ256GP710.inc"
.equ a_begin, 0x800
.text
.global ____reset

 _ reset:

     ;инициализация, в W4 - адрес будущей прогрессии
          mov #a_begin, W4
     ;Задаем первые члены прогрессии, W1<-0
          mov #0, W1
          mov W1, [W4++] ;[0x800]<-0,                    W4<-0x802
          mov #1, W1
          mov W1, [W4++] ;[0x802]<-1,                    W4<-0x804

     ;основной цикл вычисления прогрессии

          do #15, cyc_end
     ; W5 указывает на текущий элемент прогрессии        в памяти
          mov W4, W5
     ;Помещаем в W2 -1й элемент
          mov [--W5], W2
     ;Помещаем в W1 -2й элемент
          mov [--W5], W1
     ;W3<-W1+W2 вычисляем новый элемент прогрессии
          add W1, W2, W3
     ;и записываем его в память
          mov W3, [W4++]
     ;конец цикла
          cyc_end: 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. Создайте схему алгоритма и разработайте программу, вычисляющую *n* первых элементов ряда заданного рекуррентной зависимостью.

.

№ Варианта	Задание
1	*а(к) = а(к-1) mod3 + а(к-2) mod* 5, *к* = 1, 2, 3, ...
2	*а(к) = [а(к-2) + а(к-1)] div* 3, *к* = 1, 2, 3, ...
3	*а(к) = а(к-1) div 7 + а(к-2) mod* 5, *к* = 1, 2, 3, ...
4	*а(к) = 12 а(к-2) + а(к-1) mod7*, *к* = 1, 2, 3, ...
5	*а(к) = а(к-1) mod 7- а(к-2) div* 7, *к* = 1, 2, 3, ...
6	*а(к)* = [7 *а(к-2) - а(к-1)]* div 2, *к* = 1, 2, 3, ...
7	*а(к)* = 3 *а(к-1) - а(к-2) mod 13, к* = 1, 2, 3, ...
8	*а(к) = [а(к-2) + а(к-1)] mod* 17, *к* = 1, 2, 3, ...
9	*а(к) = а(к-1) mod5 + а(к-2) div* 4, *к* = 1, 2, 3, ...
10	*а(к) = а(к-1) div 3 - а(к-2) mod* 12, *к* = 1, 2, 3, ...
11	*а(к) = [а(к-2) div 7- а(к-1)] mod* 11, *к* = 1, 2, 3, ...
12	*а(к) = 7 а(к-1) - а(к-2) div* 3, *к* = 1, 2, 3, ...

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.
7. Программа должна определять возникновение ошибки переполнения при расчёте произведения элементов.
8. Составьте таблицу результатов работы рекуррентного алгоритма до *n = 10.* В шапке таблицы должны быть отражены значения: *к, а(k), а(к-1), а(к-2).*

Контрольные вопросы

1. Что такое рекурсия?
2. Назовите способы адресации, используемые в командах вашей программы.
3. Можно ли использовать цикл *REPEAT,* если в теле цикла несколько команд?
4. Можно ли использовать цикл *DO,* если в теле цикла одна команда?
5. Можно ли использовать циклы *DO* и *REPEAT,* если заранее неизвестно количество итераций цикла?
6. Назовите основные арифметические инструкции.

Практическая работа №50. Программирование микроконтроллеров (современные микроконтроллеры).

Цель работы: приобрести практические навыки по преобразованию строк, познакомиться с ASCII-кодировкой, используемой для представления текстовой информации.

Теоретическая часть.

Таблица ***ASCII***

Хранение текстовой информации и отображение её на мониторе компьютера осуществляется следующим образом. Каждый символ кодируется числом от 0 до 255.

Таким образом, строка символов в памяти компьютера храниться в виде одномерного массива байт, каждый элемент массива хранит код одного символа. При отображении строки на экране используется знакогенератор, входящий в состав графического адаптера, который по

коду символа определяет, что должно отображаться на мониторе.

С целью стандартизации для кодирования символов в ЭВМ используется Американский Национальный Стандартный Код для Обмена Информацией - *ASCII (American National Standard Code for Information Interchange,* читается “аски”). Наличие стандартного кода облегчает обмен данными между различными устройствами компьютера (рисунок 9). Восьмибитовый расширенный ASCII-код, используемый в ЭВМ, обеспечивает представление 256 символов, включая символы для национальных алфавитов.

В семействе *dsPIC/PIC24* для работы с памятью программ используется два регистра - 8-битный регистр указания на страницу *TBLPAG* (который адресует блоки по 64 кБ) и любой из 16-битных регистров общего назначения *W0-W15,* адресующий слово в выбранном 64-кБ блоке. Величины двух этих регистров формируют так называемый «эффективный адрес» *(Effective Address - EA).* Регистр *PSVPAG (Program Memory Visibility Page Address Pointer)* позволяет пользователю отобразить страницу памяти программ размером 2 Кбайт на верхние 32 Кбайт адресного пространства памяти данных.

	Most Significant Character							
	Hex	**0**	**1**	**2**	**3**	4	**5**	**6**	**7**
	0	**NUL**	**DLE**	**Space**	**0**		**P**	'	**P**
	1	**SOH**	**DC1**	**!**	**1**	**A**	**Q**	**a**	**q**
	2	**STX**	**DC2**	II	2	**B**	**R**	**b**	**r**
	3	**ETX**	**DC3**	**#**		**C**	**3**	**c**	**s**
	4	**EOT**	**DC4**	**$**	**4**	**D**	**T**	**d**	**:**
	5	**ENG**	**NAK**	*%*	5	**E**	u	**e**	**u**
Least	**6**	**AC** K	**SYN**	**&**	**6**	**F**	V	**f**	V
Significant	**7**	**Bell**	**ETB**	**i**	**7**	G	**w**	g	**w**
Character	**f!**	**BS**	**CAN**	t	a	**H**	**X**	**h**	**X**
	9	**HT**	**EM**	>	**&**	**I**	**Y**	**i**	*'J*
	A	**LF**	**SJB**	*****	-	**J**	**z**	**j**	**z**
	B	**VT**	**ESC**	+	F	**K**	**[**	к	**{**
	C	**FF**	**FS**	**i**	<	**L**	\	**i**	**I**
	D	**CR**	**GS**		=	**M**	**]**	**m**	**}**
	E	**SO**	4S		>	**N**	Л	**n**	-
	F	**SI**	JS	*I*		0	_	0	**DEL**

Рисунок 9 - Таблица ASCII символов

Инструкции сдвига

Различают арифметический, логический и циклический сдвиги.

При логическом сдвиге уходящий бит исчезает, не влияя на оставшиеся биты, а на месте появившегося бита записывается бит 0.

Арифметический сдвиг при сдвиге влево ведёт себя как логический сдвиг, при сдвиге вправо уходящий бит исчезает, не

влияя на оставшиеся биты, а на месте появившегося бита устанавливается бит, соответствующий знаку.

При циклическом сдвиге уходящий бит появляется на месте появившегося свободного на другом конце числа.

Синтаксис инструкций сдвига приведен в таблице 5 приложения 1.

Задание: Разработать программу, формирующую новую строку их заданной строки. Каждый символ из заданной строки в новой строке удваивается.

```
.include "p33FJ256GP710.inc"
.equ msg_out, 0x800                    ;начальный адрес новой строки
.text
msg_in:          .ascii   "Cogito  ergo  sum!"  ;  исходная строка
.global          __reset
__reset:
; определяем номер страницы, в которой лежит ; исходная
строка
      mov              #tblpage(msg_in), W0
; помещаем номер страницы в регистр-указатель ; на
текущую строку
      mov          W0, PSVPAG
; определяем смещение местоположения исходной строки ; от начала
страницы и получаем эффективный адрес ; начала исходной строки,
помещаем его в W6 mov              #tbloffset(msg_in),    W6
; помещаем адрес начала новой        строки в W7
      mov          #msg_out, W7
; задаем параметры цикла (за один цикл обрабатывается ; по 2
символа)
      do               #7, end
; считываем 2 байта в регистры W2 и W3 tblrdl.b [W6++], W2 tblrdl.b
[W6++], W3 ; сдвигаем влево на 8 бит содержимое W2 и помещаем в
W4 sl             W2, #8, W4
```

```
; удваиваем        первый считанный символ
          ior
                                                    W4, W2, W4
    ; помещаем удвоенный символ в новую строку
          mov        W4, [W7++]
; делаем аналогичное удвоение второго считанного символа
          sl              W3, #8, W4
          ior
                                                    W4, W3, W4
          mov        W4, [W7++]
    end: nop
nop
    .end
```

Строка *msg_in* располагается в памяти программ. Для того чтобы

ее прочитать/записать, необходимо воспользоваться специальными командами: *TBLRDL, TBLRDH, TBLWTL, TBLWTH* - первые из них предназначены для чтения младшего слова и старшего слова, две последние для записи соответственно. Для корректной работы данных инструкций сначала необходимо записать старшую часть адреса в регистр *PSVPAG*, и младшую часть адреса строки в любой рабочий регистр (как показано в примере). Для чтения/записи слова (16 бит) используют синтаксис <*TBLRDL*>, для чтения/записи байта (8 бит) используют синтаксис <*TBLRDL.B*>. Команды в примере: *SL, IOR, MOV* - выполняют требуемое задание, т.е. удваивают символ и записывают в новую строку.

Задание для самостоятельного выполнения

1. Создайте новый проект. Процессор - *dsPIC33FJ256GP710.*
2. Подключите необходимые библиотеки.

3. Подключите отладчик (симулятор), встроенный в среду *MPLAB IDE*.
4. Запустите программу из примера и проверьте в пошаговом режиме ее выполнение.
5. Осуществите вызов окна *File Register* в меню *View* и проверьте содержимое области памяти, начиная с ячейки с адресом *0800.*
6. Разработайте схему алгоритма и программу, выполняющую преобразование введённой строки по алгоритму указанному в таблице в соответствии с номером варианта

№ Варианта	Задание
1	Первый символ копируется один раз, второй - дублируется, третий - утраивается, четвёртый - копируется четыре раза, пятый - копируется пять раз, шестой - один раз, седьмой - два раза и т.д.
2	Все символы «пробел» удаляются из исходной строки, остальные символы - копируются.
3	Первый символ копируется четыре раза, второй - три раза, третий - два раза, четвёртый - один раз, пятый - четыре раза, шестой - три раза, седьмой - два раза и т.д.
4	Все символы с нечётными индексами копируются три раза, остальные - копируются один раз.
5	Все символы цифр от нуля до девяти утраиваются, остальные - копируются один раз.
6	Каждый третий символ строки удваивается, остальные - копируются один раз.
7	Все символы цифр от трех до восьми копируются два раза, остальные - копируются один раз.
8	Все символы с чётными индексами удаляются из строки, остальные - копируются один раз.
9	Все символы цифр от нуля до девяти удаляются из строки, остальные - копируются один раз.
10	Все прописные буквы латинского алфавита заменяются строчными буквами. Остальные символы копируются.
11	Каждый четвёртый символ строки копируется четыре раза, остальные - копируются один раз.
12	Все символы заглавных букв латинского алфавита утраиваются.

7. Откройте окно *Watch* и внесите в него все регистры, которые используются в коде. Для наглядности в окне *Watch* необходимо с помощью щелчка правой кнопки мышки в шапке таблицы сделать видимым столбец *Char.*
8. В пошаговом режиме отладьте код, контролируя изменение регистров в окне *Watch.* После отладки программы, покажите код и результаты работы программы преподавателю.
9. Подготовьте отчет в соответствие с требованиями на стр. 21.

Контрольные вопросы

1. Назовите способы адресации, используемые в командах вашей программы.
2. Назовите назначение каждой команды в коде вашей программы.
3. Назовите основные инструкции сравнения.
4. Назовите основные инструкции условного и безусловного переходов.
5. Почему в примере цикл копирования повторяется 8 раз?
6. Для чего служит регистр *PSVPAG?*
7. Какие инструкции смещения используются в примере?

Практическая работа №51. 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 №52. Установка периферийных устройств (использование МК).

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.

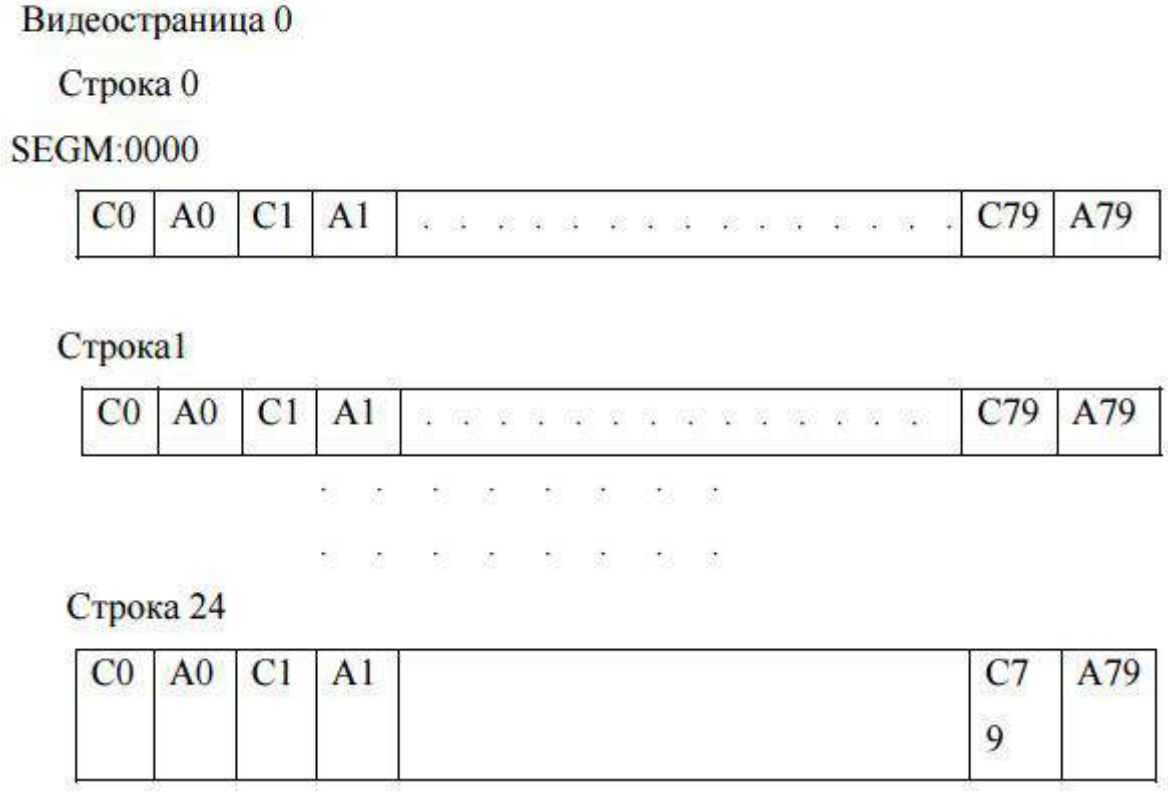
0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	**1**	0
0	**1**	0	0	0	0	**1**	0
0	**1**	**1**	**1**	**1**	**1**	**1**	0
0	**1**	0	0	0	0	**1**	0
0	**1**	0	0	0	0	**1**	0
0	**1**	0	0	0	0	**1**	0
0	0	0	0	0	0	0	0

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

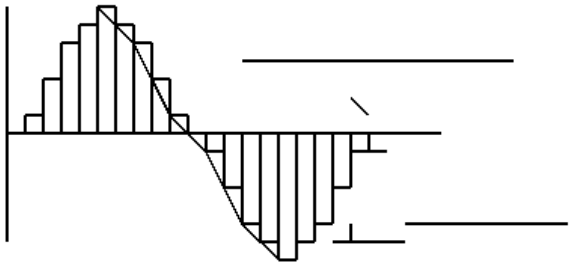
Практическая работа №53. Интерфейсы встраиваемых микропроцессорныхсистем (Intel 8080).

Цель работы: изучение элементов микропроцессорной техники и основ программирования на языке Ассемблера микропроцессора КР580ВМ80А (Intel 8080 или NTE 8080A фирмы NEC Micro).

Основные теоретические положения

В различных областях техники находят применение электрические сигналы разнообразной, изменяющейся в процессе работы формы. Формирование таких сигналов с помощью аналоговых и цифровых микросхем с жесткой логикой весьма сложно, так как определен- ная форма сигнала требует в этом случае соответствующего схемного решения. От этого недостатка свободны генераторы, построенные на основе микропроцессорных систем, обладающих высокой гибкостью и универсальностью.

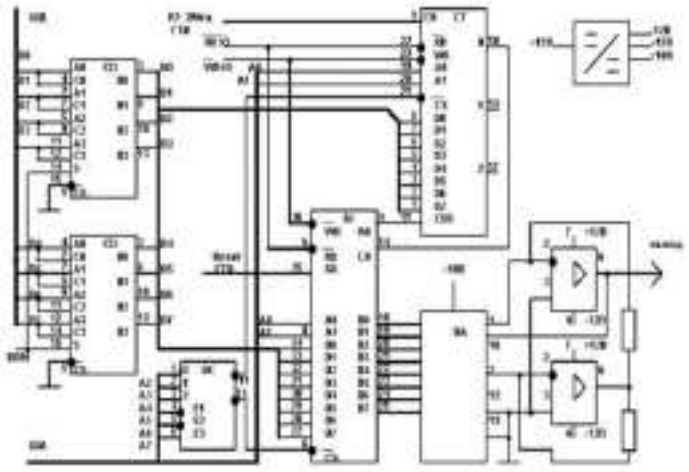
Генератор на основе микропроцессорной системы должен состоять из цифровой и аналоговой части. Цифровая часть системы генерирует цифровые коды, а аналоговая – преобразует коды в сигнал требуемой формы. Получение периодического сигнала в цифровой форме сводится к вычислению его значений в нескольких точках периода Т. На рисунке показан пример формирования периодического сигнала.Период сигнала разбивается на интервалы t.



Для каждого интервала вычисляется цифровой код, соответствующий мгновенному значению сигнала. Для повышения быстродействия системы, полученные коды целесообразно предварительно записать в память микропроцессорной системы в виде таблицы. При этом работа микропроцессорной системы сводится к поиску соответствующих кодов в таблице и выдачи их на вход преобразователя цифрового сигнала в аналоговый. Такой генератор работает по принципу кусочно-ступенчатой аппроксимации заданного сигнала.

Описание лабораторного макета

Принципиальная схема лабораторного макета показана на рисунке, он выполнен на основе микропроцессорного комплекта (УМК) и платы, содержащей шинные формирователи (CD) К589АП16, программируемый таймер (СТ) КР580ВИ53, параллельный программируемый адаптер (IO) КР580ВВ55А, цифро-аналоговый преобразователь (ЦАП), состоящий из микросхемы (DA) К572ПА1 и операционных усилителей К140УД6. Программируемый таймер может задавать длительность шага дискретизации t. Нулевой счетчик таймера должен работать в третьем режиме как делитель частоты им- пульсов, поступающих на его счетный вход.



Адаптер (IO) обеспечивает связь системы с ЦАП. Через порт «В» коды, соответствующие мгновенным значениям напряжения форми- руемого сигнала, поступают на вход ЦАП. Внешняя характеристика ЦАП показана на рис. 1.1.3., из которой видно, что для получения на выходе ЦАП максимального положи- тельного напряжения (+5 В), на его вход необходимо подать цифровой код 00Н.

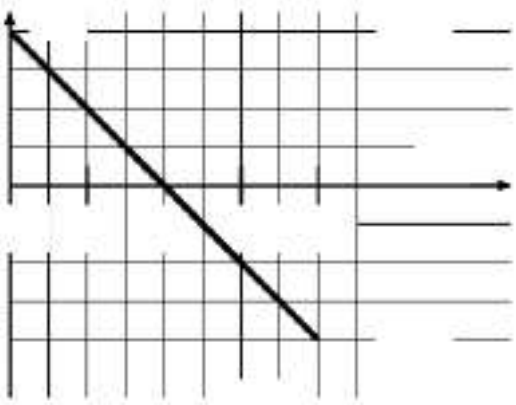


Рис. 1.1.3

Дешифратор (DC) К555ИД7 выполняет функции селектора микросхем, при этом регистр управляющего слова таймера имеет адрес 93Н, а регистр управляющего слова ППА-83Н.

Практическая часть.

Разработать программу на языке Ассемблера и в кодах микропроцессора К580.

Ввести программу в память УМК и отладить.

Измерить временные характеристики полученного сигнала.

Составить отчет. Отчет должен содержать схему макета и текст разработанной программы.

Контрольные вопросы

1. Устройство и принцип функционирования таймера и программируемого адаптера.
2. Команды программы и их назначение.
3. Байт состояния и сигналы управления системой.

Практическая работа №54. Интерфейсы встраиваемых микропроцессорных систем (NTE 8080A).

Цель работы: изучение принципов построения микропроцессорных систем отображения информации, принципов построения интерфейса и приобретения навыков программирования на языке Ассемблера микропроцессора КР580ВМ80А (Intel 8080), (NTE 8080A фирмы NEC Micro).

Основные теоретические положения

Повышение возможностей средств отображения информации (СОИ) связано с появлением новых индикаторных устройств, а также с использованием микропроцессоров в качестве элементов управления. Широкое применение находят матричные индикаторы. Однако они требуют сложных схем для адресации и коммутации большого количества индикаторных ячеек, входящих в индикаторное поле. Из существующих систем адресации наиболее часто используется схема двухкоординатной адресации, так как она хорошо стыкуется с конст-рукцией матричной панели. В такой системе возбуждение ячейки индикации происходит в узлах пересечения горизонтальных и верти- кальных шин матрицы. При этом наиболее часто применяется динамическая индикация.

Схема лабораторного макета показана на рис. 1.2.1.

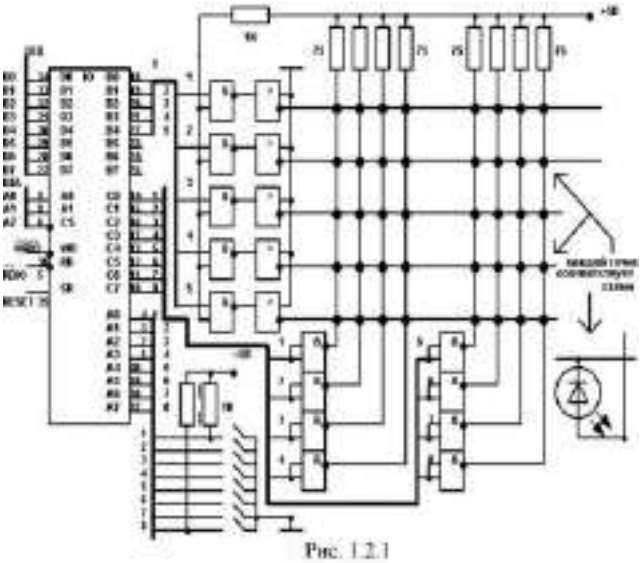


Рис. 1.2.1

Схема подключается к учебному микропроцессорному комплекту (УМК). В состав схемы входит параллельный программируемый адаптер IO, светодиодная матрица (5 х 8), собранная на светодиодных индикаторах АЛ307БМ, узел управления индикаторами на микросхемах К155ЛА13 и К155ЛП7, а также клавиатура из восьми кнопок, подключенных к порту «А» микросхемы ППА.

Практическая часть.

Составить программу на языке Ассемблера и в кодах микропроцессора К580, реализующую заданный алгоритм.

Ввести программу в «память» УМК и отладить.

Составить отчет. Отчет должен содержать схему лабораторного макета и текст разработанной программы.

Контрольные вопросы

1. Назначение, устройство, принцип работы параллельного программируемого адаптера.
2. Команды программы.
3. Байт состояния, циклы, такты, стек.

Практическая работа №55. Интерфейсы встраиваемых микропроцессорных систем (шины VME, VXI, PCI).

Цель работы: работа предназначена для изучения элементов микропроцессорной техники и основ программирования на языке Ассемблера КР580ВМ80А (Intel 8080, NTE 8080A).

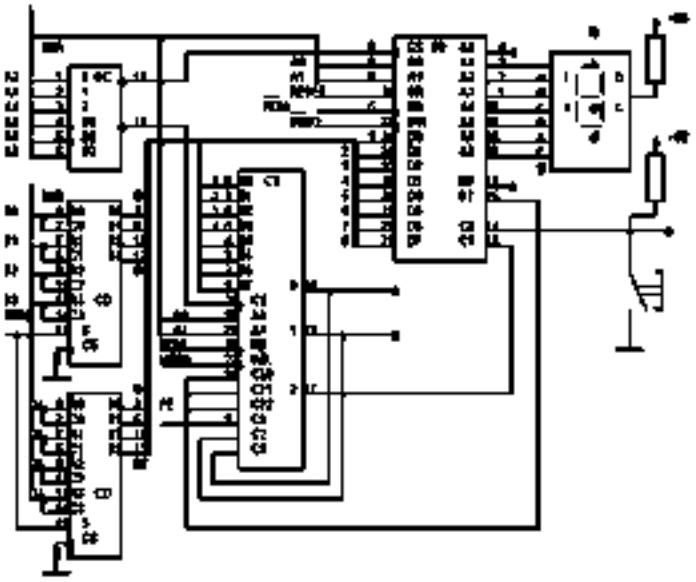
Основные теоретические положения

В состав микропроцессорной системы входят периферийные устройства (клавиатуры, дисплеи, различные исполнительные механизмы, датчики), которые требуют определенного набора управляющих сигналов, поэтому шины микропроцессора подключаются к периферийным устройствам через интерфейс. Под интерфейсом понимают совокупность программных и аппаратных средств, с помощью которых компоненты микропроцессорной системы объединяются для решения поставленной задачи.

Для построения интерфейса используются такие компоненты общего назначения, как буферные регистры, дешифраторы, шинные формирователи, таймеры, параллельные и последовательные адаптеры. Широкое применение находит программируемый периферийный (параллельный) адаптер (ППА) КР580ВВ55А. Он позволяет организовать три восьмиразрядных порта ввода-вывода параллельной информации, функции портов определяются кодом управляющего слова, загружаемого из микропроцессора в регистр управляющего слова ППА. Для организации работы микропроцессорной системы в режиме реального времени находит применение программируемый таймер (ПТ) КР580ВИ53. Таймер позволяет делить частоту

тактового генера- тора, вырабатывать одиночные, зависящие от тактовой частоты сиг- налы, вести подсчет числа внешних импульсов, измерять длительность промежутка времени. Таймер имеет три шестнадцатиразрядных счетчика с независимым управлением.

Схема лабораторного макета показана на рисунке.



Основу схемы составляет программируемый адаптер IO и таймер СТ. Дешифратор ДС (К555ИД7) служит для выбора этих микросхем, при этом регистр управляющего слова (РУС) ППА имеет адрес 83Н, а РУС таймера – адрес 93Н. Шинные формирователи СД (К589АП16) повышают нагрузочную способность линий шины данных микропро- цессора. Лабораторный макет позволяет реализовать различные алго- ритмы взаимодействия микросхем ППА и таймера и разнообразные алгоритмы работы индикатора, подключенного к порту «А» адаптера. Контроль состояний выходов ППА и таймера можно проводить с помощью осциллографа.

Практическая часть.

Для реализации заданного алгоритма разработать программу на языке Ассемблера и в кодах микропроцессора К580.

Выполнить программу.

Составить отчет. Отчет должен содержать схему лабораторного макета и текст разработанной программы.

Контрольные вопросы

1. Устройство и принцип работы ППА и таймера.
2. Порядок работы алгоритма и команд программы.

Практическая работа №56. Интерфейсы встраиваемых микропроцессорных систем (шина USB).

Цель работы: работа предназначена для изучения основных принципов построения схем и разработки программного обеспечения для микропроцессорных систем управления технологическим оборудованием.

Основные теоретические положения

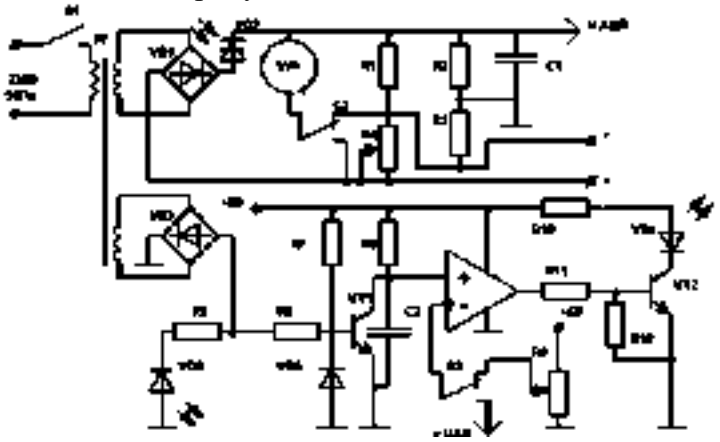
В настоящее время микропроцессорные системы находят широкое применение для управления разнообразными технологическими процессами, особенно они незаменимы в условиях, когда в течение технологического цикла требуется многократное изменение режимов работы оборудования. В качестве исполнительных устройств широкое применение находят вентильные преобразователи. Для поддержания на заданном уровне того или иного технологического параметра (температура, ток, напряжение и т. д.) применяют системы с обратной связью по контролируемому параметру. Так, система с обратной связью по току должна

содержать датчик тока (ДТ), аналого-цифровой преобразователь (АЦП), микропроцессорную систему (МС), цифро-аналоговый преобразователь (ЦАП) и систему фазового управления вентильным преобразователем (СУ).

Сигнал с ДТ преобразуется АЦП в цифровую форму и обрабатывается МС. Микропроцессорная система с помощью ЦАП формирует уровень сигнала управления, поступающего на вход СУ и определяющего величину угла управления тиристорами преобразователя.

Практическая часть.

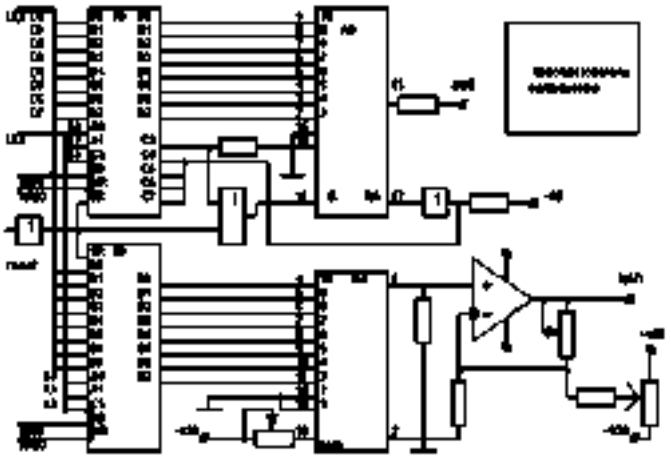
Лабораторный макет состоит из учебного микропроцессорного комплекта (УМК), интерфейса АЦА и вентильного преобразователя (ВП). Интерфейс АЦА состоит из АЦП и ЦАП. Принципиальная схема ВП показана на рисунке.



ВП выполнен по схеме управ- ляемого выпрямителя и состоит из трансформатора ТР, выпрямительного моста VD1 и оптотиристора VD2. Нагрузкой схемы служит переменный резистор R4, напряжение на котором можно контролировать с помощью осциллографа. Резистор R1 выполняет функции датчика тока. Элементы R2, R3, C1 образуют делитель и фильтр для сигнала обратной связи, поступающего с R1 на АЦП интерфейса АЦА.

Величина тока или напряжения на нагрузке (в зависимости от положения переключателя S2) измеряется с помощью прибора V/A. Для управления оптотиристором служит система управления, состоящая из схемы синхронизации (обмотка трансформатора и мост VD3), генератора пилообразного напряжения (VT1, C2), компаратора (операционный усилитель) и формирователя импульсов управления (VT2). ВП может работать в автономном режиме (регулирование напряжения на нагрузке осуществляется с помощью потенциометра R9) или в авто- матическом режиме. Выбор режима зависит от положения переключателя S3.

На следующем рисунке показана схема интерфейса АЦА, состоящая из АЦП (К1113ПВ1А), ЦАП (К572ПА1А и КР574УД1А), элементов формирования сигналов управления и преобразователя напряжения, который служит для получения напряжения 15 В.



Обмен информацией между ЦАП, АЦП и микропроцессором осуществляется с помощью микросхем ППА. Для подготовки ЦАП и АЦП к работе необходимо выполнить последовательность команд инициализации микросхем ППА: – MVI A,9AH; – OUT 83H; – MVI A,80H; – OUT 43H.

Ввод аналогового сигнала по каналу АЦП (сигнал обратной связи) осуществляется программой ввода:

– MVI A,FFH;
– OUT 82H;
– MVI A,00H;
– OUT 82H (сброс АЦП);
– M1:IN 82H;
– ANI FFH;– JNZ M1 (ожидание готовности);
– IN 81H (чтение АЦП).

Вывод аналогового сигнала по каналу ЦАП (сигнал управления вентильным преобразователем) осуществляется программой:

– MVI A, data;
– OUT 41H.

Для стабилизации величины тока нагрузки вентильного преобразователя на заданном уровне возможно применение следующего алго ритма управления:

1. Инициализация микросхем ППА.
2. Программа ввода. Считать АЦП и запомнить результат чтения в регистре В.
3. Временная пауза.
4. Программа ввода. Считать АЦП. Сравнить содержимое регистров А и В. Если содержимое регистра А меньше чем В, то необхо- димо в регистр А передать содержимое регистра С и содержимое ре- гистра А уменьшить на 1. В противном случае необходимо в регистр А передать содержимое регистра С и содержимое регистра А увеличить на 1.
5. Содержимое регистра А запомнить в регистре С.
6. Выполнить команду OUT 41H (вывод по каналуЦАП).
7. Передать управление в блок 3 этого алгоритма (временная пауза).

Работа алгоритма состоит в следующем: в автономном режиме работы ВП необходимо установить требуемую величину тока нагрузки и «запустить программу». При выполнении программы происходит считывание и запоминание в регистре В величины сигнала обратной связи, соответствующего установленной величине тока нагрузки. Если переключить ВП в режим автоматического управления, то осуществляется стабилизация величины тока нагрузки на заданном уровне (при изменении сопротивления нагрузки), так как микропроцессор постоянно считывает сигнал обратной связи и, если он оказывается больше или меньше заданной величины, то происходит коррекция сигнала управления вентильным преобразователем. Временная пауза (блок 3) задает длительность переходного процесса в цепи обратной связи (подбирается при настройке системы).

Практическая часть.

На основе приведенного выше алгоритма составить программу для микропроцессорной системы.

Порядок выполнения работы

1. Включить УМК.
2. Включить ВП.
3. Ввести программу в память системы.
4. Переключить ВП в автономный режим и установить величину тока нагрузки.
5. «Запустить» программу.
6. Переключить ВП в автоматический режим и, меняя величину нагрузки, контролировать ток. При отсутствии в программе ошибок, ток нагрузки стабилизируется на заданном уровне. С помощью осциллографа контролировать форму напряжения на нагрузке.

7. Оформить отчет. Отчет должен содержать схему лабораторного макета и текст программы.

Контрольные вопросы

1. Назначение и взаимодействие АЦП, ЦАП, ППА.
2. Назначение блоков алгоритма и их взаимодействие.

Практическая работа №57. Интерфейс с переферийными устройствами

Цель работы: работа предназначена для изучения основных принципов разработки программного обеспечения и построения микропроцессорных систем на базе однокристального микроконтроллера серии КР1816, а также принципов построения многопроцессорных систем

Основные теоретические положения

Микропроцессорный комплект серии КР1816 состоит из микросхем КР1816ВЕ35, КР1816ВЕ39, КМ1816ВЕ48, КР1816ВЕ49. Существуют многочисленные зарубежные аналоги этих микросхем, в частности, БИС фирмы Intel: 8035, 8039, 8048, 8049. Эти БИС представляют собой однокристальную 8-разрядную микро-ЭВМ, содержащую центральный процессор, ОЗУ данных, интерфейс ввода-вывода, 8-разрядный таймер, тактовый генератор, программную память, векторную систему прерываний. Перечисленные микросхемы имеют одинаковую структуру и отличаются быстродействием, объемом ОЗУ, наличием или отсутствием памяти программ. Например, БИС КР1816ВЕ48 содержит электрически перепрограммируемое ППЗУ емкостью 1К с ультрафиолетовым стиранием информации. На следующем рисунке показано графическое обозначение микросхем серии.

На рисунке приняты следующие обозначения выводов БИС:

T0 – используется как вход при выполнении команд JT0, JNT0 и как выход тактовых сигналов после выполнения команды ENT0 CLK;

BQ1, BQ2 – используются для подключения кварцевого резонатора, LC – переключение цепи или внешнего генератора;

SR – сигнал инициализации;

SS – используется для организации пошагового режима;

INT – сигнал прерывания;

EMA – сигнал обращения к внешней памяти программ;

RD – сигнал чтения из внешней памяти данных;

WR – сигнал записи во внешнюю память данных;

PME – сигнал чте- ния из внешней памяти программ;

ALE – стробирующий сигнал адреса;

P0 – восьмиразрядный двунаправленный трехстабильный порт;

P1 – восьмиразрядный квазидвунаправленный порт;

P2 – восьмираз- рядный квазидвунаправленный порт;

GND – общий;

PR – используется как выход для расширения каналов ввода/вывода и как вход для программирования ППЗУ КР1816ВЕ48;

Udd – напряжение питания +5 В (при программировании КР1816ВЕ48 +25В);

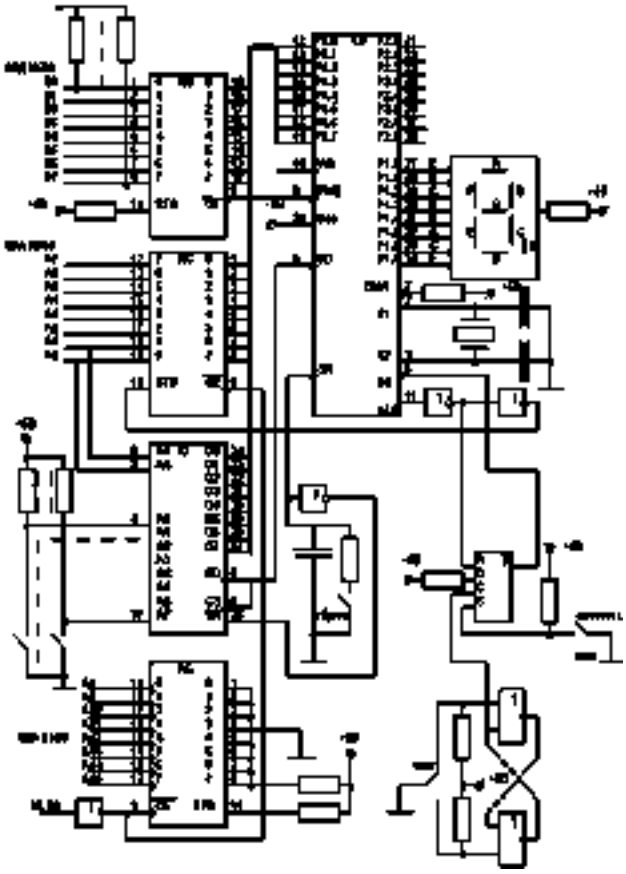
T1 – вход для счет- чика после выполнения команды STRT CNT, а также сигнал проверяемый командами JT1, JNT1;

Ucc – напряжение питания +5 В.

Микросхемы серии КР1816 имеют в своем составе все атрибуты микро-ЭВМ: арифметико-логическое устройство, устройство управления, ОЗУ данных и интерфейсные схемы. Структура, система команд и средства ввода-вывода микросхем серии КР1816 лучше всего приспособлены для решения задач управления различными технологическими процессами.

Описание лабораторного макета

Лабораторный макет выполнен на основе БИС КР1816ВЕ39, которая не имеет в своем составе памяти программ, поэтому ее функции выполняет ОЗУ учебного микропроцессорного комплекта, что позволяет проводить оперативную отладку программ. Схема лабораторного макета показана на следущем рисунке.



Связь микро-ЭВМ КР1816 с шиной данных D0 – D7 и шиной адреса A0 – A15 учебного комплекта осуществляется через регистры КР580ИР82 и КР580ИР83.

Программа функционирования макета должна располагаться в ОЗУ УМК, начиная с адреса 0800H, и первой обязательно должна быть команда HLT (код 76H). HLT – команда микропроцессора КР580. При ее выполнении буферные схемы шины данных и шины адреса микропроцессора КР580 переводятся в состояние высокого со- противления, а в байте состояния появляется сигнал HLTA (подтверждение остановки), который используется в схеме для перевода регистров, осуществляющих связь микро-ЭВМ КР1816 с шиной адреса УМК, в рабочее состояние. При этом открывается доступ КР1816 к ОЗУ УМК объемом 256 байт.

В состав схемы макета входит ППА, к порту «А» которого подключено восемь ключей. Сигнал «сброс» микро-ЭВМ и ППА формируется при включении питания или после нажатия кнопки «сброс». При этом порты ППА настраиваются на ввод информации в нулевом режиме. К порту Р1 микро-ЭВМ подключен индикатор.

В схеме предусмотрена возможность выполнения программы в автоматическом режиме или по шагам (по командам). Для этого переключателем АВТ./ ШАГ выбирается режим работы, а кнопка ШАГ используется для вы- полнения команд по шагам.

Практическая часть.

Задания предусматривают реализацию различных алгоритмов управления индикатором. Задания также можно строить на основе приведенной ниже программы, которая в зависимости от дво- ичного кода, набранного с помощью ключей, высвечивает на индика- торе цифры: 0, 1, 2,

Текст программы на языке ассемблера и в кодах микро-ЭВМ КР1816

Метка	Мнемокод	Адрес	Код

M1	HLT
	MOV A, #FFH
	OUTL P1, A
	MOV R0,
	MOVX A,
	MOVP A, @A
	JMP M1
		0800H
		0801H, 0802H
		0803H
		0804H, 0805H
		0806H
		0807H
		0808H, 0809H
			76H
			23H, FFH
			39H
			B8H, 00H
			80H
			A3H
			04H, 03H

Код цифры 3		08FCH	89H
Код цифры 2		08FDH	8CH
Код цифры 1		08FEH	EBH
Код цифры 0		08FFH	48H

Порядок выполнения практической работы

1. Разработать, ввести и отладить программу функционирования лабораторного макета.
2. Оформить отчет. Отчет должен содержать схему лабораторного макета и текст разработанной программы.

Контрольные вопросы

1. Принципы работы микро-ЭВМ КР1816.
2. Принципы взаимодействия узлов макета.
3. Назначение команд программы.

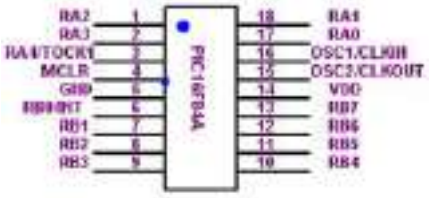
Практическая работа №58. Интерфейс с периферийными устройствами (JTAG – интерфейс).

Цель работы: работа предназначена для изучения принципов построения микропроцессорных систем управления технологическими процессами на основе микроконтроллеров серии PIC.

Основные теоретические положения

Производимое фирмой Microchip Technology семейство интегральных микросхем PIC16FXX, представляет собой восьмиразрядные микроконтроллеры с RISC архитектурой. Это семейство микроконтроллеров отличается низким энергопотреблением, высокой скоростью выполнения команд, низкой ценой. Микроконтроллеры имеют встроенное ПЗУ для хранения программ и энергонезависимое ОЗУ данных. Одним из простейших представителей этого семейства явля- ется микроконтроллер PIC16F84. Контроллер выполнен по КМОП технологии, имеет внутреннее 1 К х 14 бит электрически перепрограммируемое ПЗУ (EEPROM), предназначенное для хранения выполняемой программы и 64 байта энергонезависимого ОЗУ данных. Команды, выполняемые контроллером, состоят из одного слова шириной 14 бит и выполняются в течение одного цикла (400 нс. при частоте синхронизации 10 Мгц), команды передачи управления выполняются за два цикла. Длительность цикла составляет четыре периода тактовой частоты. Контроллер сохраняет работоспособность в диапазоне частот от 0 до 10 Мгц. Контроллер обеспечивает прерывания от 4-х источников и имеет для их обслуживания восьмиуровневый аппаратный стек. Содержит 8-битный таймер/счетчик с 8-битным программируемым предварительным делителем. Для подключения периферийных устройств (датчики, клавиатуры, исполнительные механизмы) контроллер имеет 13 линий, программно-настраиваемых на ввод или вывод информации. Высокая нагрузочная способность этих линий (20 мА) упрощает схемную реализацию драйверов ввода/вывода. Разработки устройств на базе этих контроллеров поддерживаются ассемблером, программаторами и программным симулятором (PIC Simulator IDE), который позволяет проводить отладку программ и моделировать

поведение внешних устройств, подключенных к портам контроллера. На следующем рисунке показана функциональная схема контроллера.



На схеме приняты следующие обозначения: выводы OSC1 и OSC2 предназначены для подключения кварцевого резонатора или RC-цепочки, или генератора тактовых импульсов. Выводы RA0 – RA4 образуют разряды порта A (PORTA). TOCK1 может также использоваться как вход таймера/счетчика. Выводы RB0 – RB7 являются разрядами порта B (PORTB). INT – вход внешнего прерывания. RB4, RB5 могут использоваться как входы прерывания по изменению сиг- нала. RB6 – вход прерывания по изменению сигнала или вход тактового сигнала при программировании контроллера. RB7 – вход прерывания по изменению сигнала или сигнал данных при программировании. MCLR – вход сигнала сброса или напряжения при программировании. VDD – положительный вывод питания. GND – общий вывод.

Применение однокристальных микро-ЭВМ позволяет значительно упростить процесс разработки сложных систем управления различными технологическими процессами. Такие системы отличаются высокими технико-экономическими и эксплуатационными характеристиками. Основное достоинство этих схем состоит в их гибкости и универсальности, что позволяет, с минимальными затратами труда и времени, перестраивать их для решения различных задач управления. При разработке таких устройств основное внимание уделяется разра- ботке программного обеспечения, реализующего заданный алгоритм управления.

Описание лабораторного макета

Схема лабораторного макета показана на следущем рисунке:

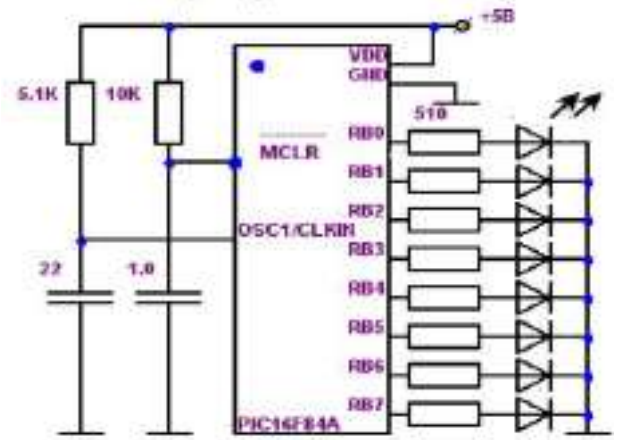


Схема содержит RC-цепочки для синхронизации и сброса. Параметры время задающей цепочки (5,1 Ком и 22 Пф) обеспечивают рабочую частоту контроллера (10 Мгц). К выводам порта «В» контроллера (RB0 – RB7), через токоограничивающие резисторы (510 Ом), подключены светодиоды. Питание схемы осуществляется от источника постоянного напряжения +5 В. Применение светодиодов, позволяет наглядно контролировать прохождение сигналов в схеме.

Ниже приведен текст, демонстрационной программы написанной на языке Ассемблера, реализующей алгоритм «бегущий огонь».

```
        bsf status,5h movlw 00h
        movwf trisb     ; настройка порта B на вывод
        bcf status,5h
        movlw 01h       ; в рабочий регистр w заносим число
01h movwf 0ch           ; в регистр 0ch заносим w
        m0:     movf 0ch,0      ; в регистр w заносим
содержимое
        ; регистра 0ch
        movwf portb ; содержимое регистра w выводим
        ; в порт B
        rlf 0ch,1       ; сдвиг содержимого регистра 0ch влево
        goto m0
```

Практическая часть.

На основе приведенного выше текста демонстрационной программы разработать программу, реализующую следующий алгоритм управления светодиодами:

а) «бегущий огонь»;

б) «мигают» крайние светодиоды;

в) «мигает» один светодиод и т. д.

Порядок выполнения работы

1. Используя программатор или симулятор, ввести и отладить программу, реализующую заданный преподавателем алгоритм.

2. Оформить отчет. Отчет должен содержать схему лабораторного макета и текст разработанной программы.

Контрольные вопросы

1. Назначение выводов контроллера.
2. Назначение и порядок работы элементов схемы.
3. Назначение и порядок работы команд программы.

Практическая работа №59. Интерфейс с периферийными устройствами (Передача типа Управление (Control)).

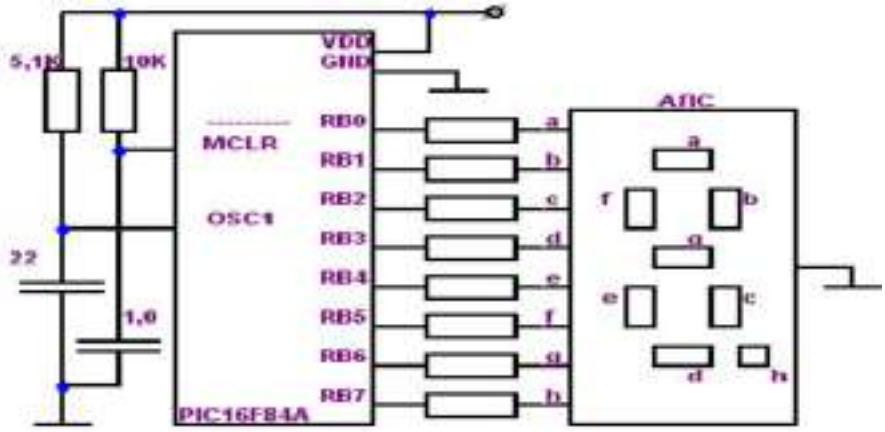
Цель работы: работа предназначена для изучения принципов построения схем и разработки программного обеспечения микропроцессорных устройств, выполненных на основе контроллеров семейства PIC.

Основные теоретические положения

В устройствах, предназначенных для измерения различных физических величин и для управления технологическими процессами, одним из наиболее важных узлов является узел отображения информации. Наиболее целесообразно в таких схемах использовать современную элементную базу, так как применение микроконтроллеров позволяет значительно расширить круг решаемых задач, повысить надежность и уменьшить затраты на разработку и изготовление устройств отображения информации. Наиболее простыми и широко распространенными современными индикаторами являются семисегментные полупроводниковые индикаторы. Существует большое разнообразие цифровых индикаторов, например, индикаторы типа: АЛС 320, АЛС 324 и т. д.

Описание лабораторного макета

Схема лабораторного макета приведена на следующем рисунке:



Неиспользованные выводы контроллера на схеме не показаны.

Схема содержит цифровой семисегментный индикатор, подключенный к выводам (RB0 – RB7) порта «B» контроллера. RC-цепочка (5,1 К, 22 Пф) служит для задания рабочей частоты (10 Мгц). RC-цепочка (10 К, 1,0 мкф) обеспечивает необходимую длительность сигнала «сброс».

Ниже приводится текст программы, выводящей на индикатор последовательность символов:

```
      bsf status,5h
      movlw 00h
      movwf trisb  ; настройка порта B на вывод
      bcf status,5h
m0:   movlw 3fh  ; запись в регистр w числа 3fh(код нуля)
      movwf portb ;  вывод w в порт B
      movlw 06h ; запись в w числа 06h(код единицы)
      movwf portb
      goto m0
```

Практическая часть.

1. Используя программатор или симулятор, ввести и отладить программу, реализующую заданный алгоритм.

Оформить отчет. Отчет должен содержать схему лабораторного макета и текст разработанной программы.

Контрольные вопросы

1. Назначение элементов схемы и принципы ее работы.
2. Назначение команд программы.

Практическая работа №60. Интерфейс с периферийными устройствами (применение программируемой логики).

Цель работы: работа предназначена для изучения схемотехники подключения клавиатуры к портам контроллера и принципов разработки программного обеспечения для управления клавиатурой.

Основные теоретические положения

Применение клавиатуры необходимо для обслуживаемых микропроцессорных систем, когда есть необходимость в оперативном изменении режимов работы системы или вводе необходимой для работы информации. Сложность клавиатуры зависит от сложности решаемых задач.

Широкое применение находят матричные клавиатуры, так как в этом случае клавиатура имеет минимальное количество клавиш при максимальном количестве решаемых функций.

Функциональная схема лабораторного макета показана на следующем рисунке:

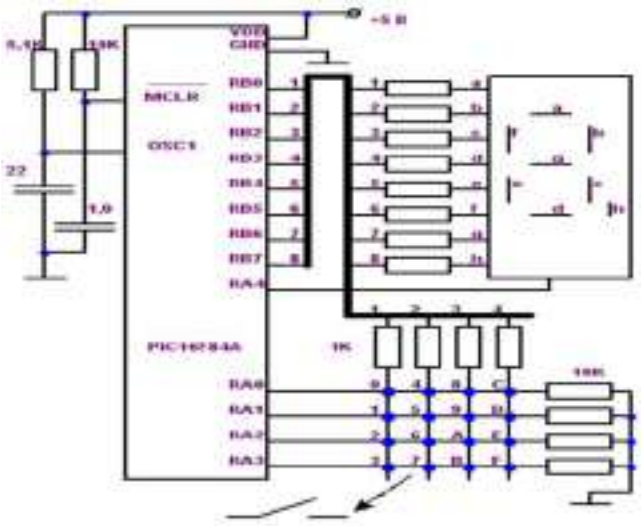


Схема содержит цифровой семисегментный индикатор, подключенный к разрядам порта «B» и матричную клавиатуру (4 x 4). Клавиши клавиатуры пронумерованы символами от 0 до F. Опрос клавиатуры осуществляется методом сканирования, при этом разряды порта « A» (RA0 – RA3) контроллера должны быть настроены на ввод, а разряды порта «B» – на вывод информации. Разряд RA4 порта «A» должен быть настроен на вывод: этот разряд позволяет разрешать (выводится лог.1) или запрещать (выводится лог.0) работу индикатора.

Ниже приведен текст демонстрационной программы, отображающей на индикаторе цифры: 0, 1, 2, 3, что зависит от номера нажатой клавиши клавиатуры (0, 1, 2, 3).

```
        bsf status,5h
        movlw 0fh
        movwf trisa ;  разряды 0–3 порта A на ввод
        movlw 00h
        movwf trisb ;  порт B на вывод
        bcf status,5h
m0:     clrf porta ;  очистка порта A (запрет индикатора)
        movlw 01h
        movwf portb ;  выбор кнопок 0, 1, 2, 3 (вывод  лог.1  в
RB0)    btfss porta,0h ; проверка RA0
        goto m1
        movlw 3fh
        movwf portb ; вывод символа «0»
        goto m10
m1:     btfss porta,1h ;  проверка RA1
        goto m2
        movlw 06h
        movwf portb ; вывод символа «1»
        goto m10
m2:     btfss porta,2h ;  проверка RA2
        goto m3
        movlw 5bh
        movwf portb ; вывод символа «2»
        goto m10
```

```
m3:     btfss porta,3h ; проверка RA3
        goto m0
        movlw 4fh
        movwf portb ; вывод символа «3»
m10:    movlw 10h
        movwf porta ; разрешить работу индикатора
        goto m0
```

Практическая часть.

1. Разработать ассемблер программу, реализующую заданный алгоритм.
2. Используя программатор или симулятор для PIC-контроллера ввести и отладить программу.
3. Оформить отчет. Отчет должен содержать схему лабораторного макета и текст разработанной ассемблер программы.

Контрольные вопросы

1. Назначение выводов PIC-контроллера.
2. Порядок настройки портов.
3. Назначение команд программы.

Практическая работа №61. Элементарные приемы программирования.

Цель работы: работа предназначена для изучения принципов построения схем и разработки программного обеспечения микропроцессорных устройств, выполненных на основе контроллеров семейства AVR.

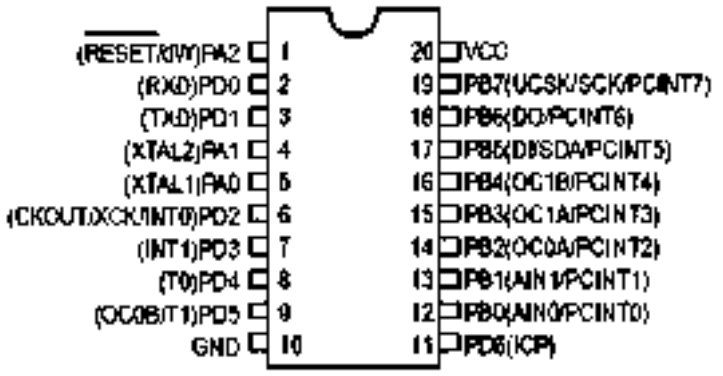
Основные теоретические положения

Компания Atmel выпускает обширную линейку восьмиразрядных микроконтроллеров на базе AVR-ядра, разбитых на несколько подсемейств, различающихся по техническим характеристикам, области применения и цене.

- ATtiny – семейство AVR-микроконтроллеров, предназначенных для использования в устройствах, требующих относительно большой производительности (до 1,0 MIPS, способны работать на частотах до 20,0 МГц), низкого энергопотребления (ATtiny единственное семейство способное работать от 0,7 В напряжения питания!) и компактности (есть микроконтроллеры в SOT23-6 корпусе – всего 6 выводов, и при этом каждый вывод обладает несколькими функциями.
- ATmega – семейство AVR-микроконтроллеров, предназначенных для использования в самых разнообразных областях, благодаря очень большому набору периферийных устройств, большому объему памяти программ и портов ввода/вывода.
- ATxmega – новое семейство AVR-микроконтроллеров с еще большим набором периферийных устройств, чем у ATmega (добавилось устройство прямого доступа к памяти, ЦАП, полноценный USB-интерфейс, более быстрый АЦП и др.), с рабочими частотами до 32,0 МГц.

Все вышеперечисленные семейства имеют единую архитектуру (что позволяет с легкостью переносить код с одного микроконтроллера на другой), выпускаются как в DIP, так и SMD корпусах.

Контроллеры семейства AVR в настоящее время находят широкое применение в различных устройствах. На следующем рисунке показана схема расположения выводов для контроллеров AT90S2313 и ATtiny2313. Отличие между этими контроллерами состоит в том, что AT90S2313 имеет два порта PD и PB, а контроллер ATtine2313 содержит три порта PA, PB, PD.



Выводы XTAL2, XTAL1 используются для подключения кварцевого резонатора.

Характеристика контроллера AT90S2313. AT90S2313 является 8-разрядным CMOS-микроконтроллером с низким энергопотреблением, основанным на усовершенствованной AVR RISC-архитектуре. Благодаря выполнению высокопроизводительных инструкций за один период тактового сигнала, AT90S2313 достигает производительности,приближающейся к уровню 1 MIPS на МГц, обеспечивая разработчику возможность оптимизировать уровень энергопотребления в соответствии с необходимой вычислительной производительностью. Ядро AVR содержит мощный набор инструкций и 32 рабочих регистра общего назначения. Все 32 регистра напрямую подключены к арифметико-логическому устройству (АЛУ), что обеспечивает доступ к двум независимым регистрам при выполнении одной инструкции за один такт. В результате, данная архитектура имеет более высокую эффективность кода, при повышении пропускной способности, вплоть до 10 раз, по сравнению со стандартными микроконтроллерами CISC.

AT90S2313 имеет: 2 Кбайт Flash-памяти с поддержкой внутрисистем- ного программирования, 128 байт EEPROM, 15 линий I/O общего назначения, 32 рабочих регистра общего назначения, универсальные таймеры/счетчики с режимами сравнения, внутренние и внешние прерывания, программируемый UART последовательного типа, програм- мируемый следящий таймер с встроенным тактовым генератором и программируемый последовательный порт SPI для загрузки программ в Flash-память, а также, два программно-выбираемых режима экономии энергопотребления. Режим ожидания «Idle Mode» останавливает CPU, но позволяет функционировать SRAM, таймеру/счетчикам, SPI порту и системе прерываний. Режим экономии энергопотребления «Power Down» сохраняет значения регистров, но останавливает тактовый генератор, отключая все остальные функции микроконтроллера, вплоть до следующего внешнего прерывания или до аппаратной инициализации. Контроллер производится с применением технологии энергонезависимой памяти с высокой плотностью размещения, разработанной в корпорации Atmel. Встроенная Flash-память с поддержкой внутрисистемного программирования обеспечивает возможность перепрограммирования программного кода в составе системы, посредством SPI последовательного интерфейса, или с помощью стандартного программатора энергонезависимой памяти. Благодаря совмещению усовершенствованного 8-разрядного RISC CPU с Flash-памятью с поддержкой внутрисистемного программирования на одном кристалле получился высокопроизводительный микроконтроллер AT90S2313, обеспечивающий гибкое и экономически-высокоэффективное решение для многих приложений встраиваемых систем управления. AVR AT90S2313 поддерживается полным набором программ и пакетов для разработки, включая: компиляторы С, макроассемблеры, от- ладчики/симуляторы программ, внутрисхемные эмуляторы и наборы для макетирования.

Характеристика контроллера ATtiny2313. ATtiny2313 – низко-потребляющий 8-битный КМОП микроконтроллер с AVR-RISC архитектурой. Выполняя команды за один цикл, ATtiny2313 достигает производительности 1 MIPS при частоте задающего генератора 1 МГц, что позволяет разработчику оптимизировать отношение потребления к производительности. AVR-ядро объединяет богатую систему команд и 32 рабочих регистра общего назначения. Все 32 регистра непосредственно связаны с арифметико-логическим устройством (АЛУ), что позволяет получить доступ к двум независимым регистрам при выполнении одной команды. В результате

эта архитектура позволяет обеспечить в десятки раз большую производительность, чем стандартная CISC-архитектура.

ATtiny2313 имеет следующие характеристики: 2 КБ программируемой в системе Flash-память программы, 128-байтную EEPROM память данных, 128-байтное SRAM (статическое ОЗУ), 18 линий ввода-вывода общего применения, 32 рабочих регистра общего назначения, однопроводный интерфейс для встроенного отладчика, два гибких таймера/счетчика со схемами сравнения, внутренние и внешние источники прерывания, последовательный программируемый USART, универсальный последовательный интерфейс с детектором стартового условия, программируемый сторожевой таймер со встроенным генератором и три программно-инициализируемых режима пониженного потребления. В режиме Idle останавливается ядро, но ОЗУ, таймеры/счетчики и система прерываний продолжают функционировать. В режиме Power-down регистры сохраняют свое значение, но генератор останавливается, блокируя все функции прибора до следующего пре- рывания или аппаратного сброса. В Standby режиме задающий генера-тор работает, в то время как остальная часть прибора бездействует. Это позволяет очень быстро запустить микропроцессор, сохраняя при этом в режиме бездействия мощность. Контроллер изготовлен по высокоплотной энергонезависимой технологии изготовления памяти компании Atmel. Встроенная ISP Flash позволяет перепрограммировать память программы в системе через последовательный SPI- интерфейс или обычным программатором энергонезависимой памяти. Объединив в одном кристалле 8-битное RISC-ядро с самопрограммирующейся в системе Flash-памятью, ATtiny2313 стал мощным микроконтроллером, который дает большую гибкость разработчика микропроцессорных систем. ATtiny2313 поддерживается различными программными средствами и интегрированными средствами разработки такими, как компиляторы С, макроассемблеры, программные отладчики/симуляторы, внутрисхемные эмуляторы и ознакомитель- ные наборы.

Функциональная схема макета показана на следующем рисунке:

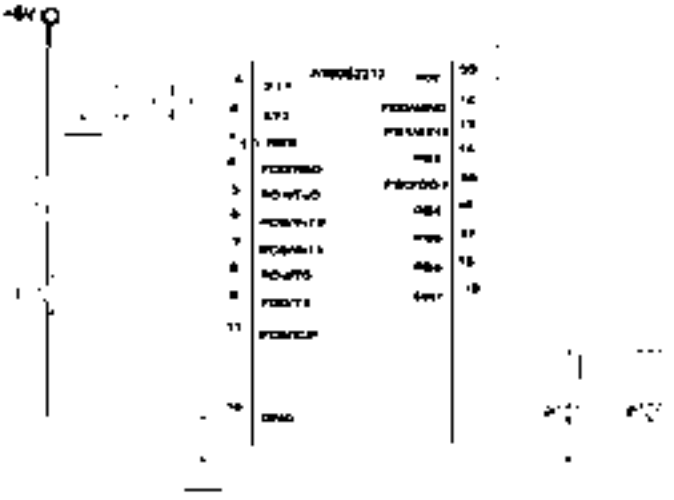


Схема выполнена на основе микроконтроллера AT90S2313, к портам которого подключены светодиоды и кнопка. Резисторы, включенные последовательно со светодиодами, служат для ограничения токов, протекающих через светодиоды и выводы контроллера. Макет может быть реализован в виртуальном виде на основе программного продукта PROTEUS или AVR-Simulator.

Ниже приведена программа управления светодиодами.

```
.def temp=r16
ldi temp,0x00
out ddrd,temp
ldi r21,0x01
m4:
 in r20,PIND
```

```
andi r20,0x04
brne m4
ldi temp,$0ff
out ddrb,temp
out PORTB,r21
ldi r17,0x0ff
m3:
ldi r18,0x0ff
m2:
dec r18
brne m2
dec r17
brne m3
ROL r21
rjmp m4
```

Практическая часть.

1. Ввести в память контроллера и выполнить выше приведенную программу, понять назначение команд программы.

2. Используя текст приведенной выше программы, разработать программу на языке Ассемблера, реализующую алгоритмы управления светодиодами:

а) «мигает» один светодиод;

б) «мигают» два светодиода.

3. Используя макет или симулятор схемы, ввести и отладить программу, реализующую заданный алгоритм.

4. Оформить отчет. Отчет должен содержать схему лабораторного макета и текст разработанной программы.

Контрольные вопросы

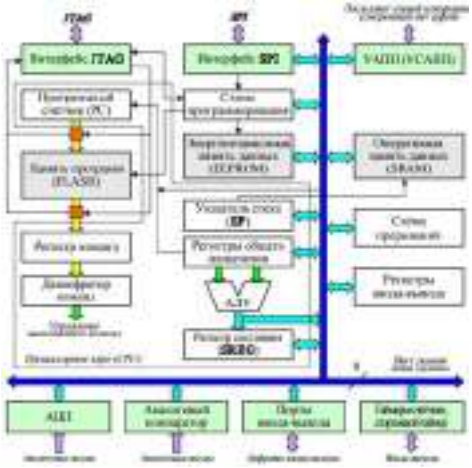
1. Назначение элементов схемы и принципы ее работы.
2. Назначение команд программы.

Практическая работа №62. Программирование микропроцессорных систем (архитектура и программная модель микроконтроллеров семейства AVR).

Цель работы: изучить архитектуру, программную модель микроконтроллеров семейства AVR.

Основные теоретические сведения.

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.



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.

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 последователь- ность 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 64K байт.

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

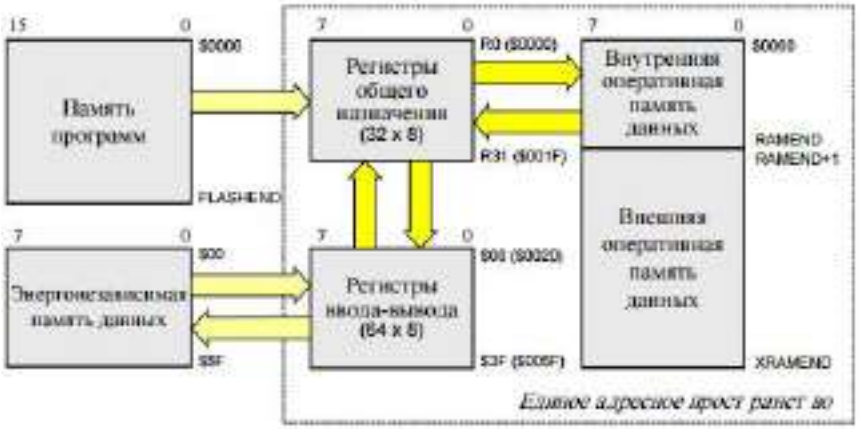
Схема прерываний обеспечивает возможность асинхронного прерывания процесса выполнения программы при определённых условиях.

К периферийным устройствам AVR-микроконтроллера относятся порты ввода-вывода, таймеры, счётчики, сторожевой таймер, аналоговый компаратор, аналого-цифровой преобразователь, универсальный асинхронный (синхронно- асинхроный) приёмопередатчик –

УАПП (УСАПП), последовательный периферийный интерфейс SPI, интерфейс JTAG и др. Набором периферийных устройств определяются функциональные возможности микроконтроллера.

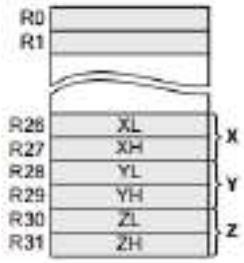
Обмен информацией между устройствами AVR-микроконтроллера осуществляется посредством внутренней 8-разрядной шины данных.

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.



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]. 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 записыва- ются 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 опера- тивная 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 микропроцес- сорной 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 микропро- цессорной системы (калибровочные коэффициенты, конфигурационные параметры и т. п.). Энергонезависимая память данных имеет отдельное адресное пространство и может быть считана и записана программным путём.

Практическая работа №63. Программирование микропроцессорных систем (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 би- тами. 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 учета флага переноса	Rd ← Rd + Rr	Z, C, N, V, H, S	1
ADC	Rd, Rr	Сложить с учетом 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 учета 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 учетом флага переноса	Rd ← Rd - Rr - C	Z, C, N, V, H, S	1
SBCI	Rd, K8	Вычесть константу с учетом 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 single (порядковая инверсия)	Rd ← $FF - Rd	Z, C, N, V, S	1
NEG	Rd	Вычислить дополнение до двух (изменить знак)	Rd ← $00 - Rd	Z, C, N, V, H, S	1
SBR	Rd, K8	Установить разряд (разряды) в POH	Rd ← Rd v K8	Z, C, N, V, S	1
CBR	Rd, K8	Сбросить разряды в POH	Rd ← Rd · ($FF - K8)	Z, C, N, V, S	1
INC	Rd	Инкрементировать содержимое POH	Rd ← Rd + 1	Z, N, V, S	1
DEC	Rd	Декрементировать содержимое POH	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 POH	Rd ← Rd ⊕ Rd	Z, C, N, V, S	1
SER	Rd	Установить все разряды POH	Rd ← $FF	Нет	1
...	Rd, Rr	Сравнить	Rd - Rr	Z, C, N, V, H, S	1
CPC	Rd, Rr	Сравнить с учетом флага переноса	Rd - Rr - C	Z, C, N, V, H, S	1
CPI	Rd, K8	Сравнить с константой	Rd - K8	Z, C, N, V, H, S	1
MUL	Rd, Rr	Перемножить содержимое двух POH (без знака)	R1:R0 ← Rd · Rr	Z, C	2
MULS	Rd, Rr	Перемножить содержимое двух POH (со знаком)	R1:R0 ← Rd · Rr	Z, C	2
MULSU	Rd, Rr	Перемножить содержимое двух POH (Rd – со знаком; Rr – без знака)	R1:R0 ← Rd · Rr	Z, C	2
FMUL	Rd, Rr	Перемножить содержимое двух POH (без знака) со сдвигом влево на 1 разряд	R1:R0 ← Rd · Rr << 1	Z, C	2
FMULS	Rd, Rr	Перемножить содержимое двух POH (со знаком) со сдвигом влево на 1 разряд	R1:R0 ← Rd · Rr << 1	Z, C	2
FMULSU	Rd, Rr	Перемножить содержимое двух POH (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 POH сброшен	если Rr(b) = 0, PC ← PC + 2 (или 3)	Нет	1/2/3
SBRS	Rr, b	Пропустить, если бит в POH 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 POH	**Rd** ← **Rr**	Нет	1
MOVW	Rd, Rr	Копирование пары POH	**Rd+1:Rd** ← **Rr+1:Rr**	Нет	1
LDI	Rd, kB	Загрузка константы в POH	**Rd** ← **kB**	Нет	1
LDS	Rd, k	Прямая загрузка из ОЗУ в POH	**Rd** ← (k)	Нет	2
LD	Rd, X	Косвенная загрузка из ОЗУ в POH	**Rd** ← (X)	Нет	2

1	2	3	4	5	6
LD	Rd, Y	Косвенная загрузка из ОЗУ в POH	**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 POH	**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
STD	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
XCH	Нет	Запись в память программ	(Z) ← **R1**:**R0**	Нет	–
IN	Rd, DO	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 POH в стек	**STACK** ← **Rr**	Нет	2
POP	Rd	Извлечение из стека в POH	**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 тайблы 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 POH во флаге T	T ← **Rr**(b)	T	1
BLD	Rd, b	Загрузить разряд из флага T в POH	**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 флаг 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 перехода	H ← 1	H	1
CLH	Нет	Сбросить флаг внутреннего перехода	H ← 0	H	1
NOP	Нет	Нет операции	Нет	Нет	1
RESET	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 (исходный) в исходный РОН;
Rr – исходный (второй) РОН;
IO – 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 объему памяти данных (при размере памяти данных более 64К байт) и доступ к расширенным 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 объему памяти программ для микроконтроллеров с размером памяти программ более 64К байт;
k8 – константа (8 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 полуперенос (перенос между третьим и четвертым разрядами байта);
T – флаг для команд пересылки;
I – флаг глобального разрешения (запрещения) прерываний.

Практическая работа №64. Программирование микропроцессорных систем (кодирование).

Цель работы: изучить процесс разработки прикладного ПО устройств на примере языка ассемблера AVR-микроконтроллеров.

Теоретические сведения.

Процесс разработки прикладного ПО устройств на основе однокристальных микроконтроллеров включает следующие этапы:

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

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 часть.

Составить программу вычисления произведения и суммы двух чисел A и B, находящихся в РОН. Из суммы A и B вычесть число C. За основу взять програм му, приведённую выше. Числа изменить в соответствии с заданным вариантом (табл. 1).

Таблица 1

№ варианта	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
B	100	99	98	97	96	95	94	93	92	91	90	89	88	87	86
C	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64

Продолжение табл. 1

№ варианта	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
A	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
B	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71
C	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79

В начало программы поместить директиву .device для микроконтроллера ATmega8535 (здесь и далее предполагается использование микроконтроллера ATmega8535). В комментариях указать фамилию и номер группы.

Контрольные вопросы

1. Назначение однокристальных микроконтроллеров.
2. Особенности архитектуры однокристальных микроконтроллеров.
3. Архитектура и программная модель AVR-микроконтроллеров.

Практическая работа №65. Программирование микропроцессорных систем (декодирование).

Цель работы: изучить процесс разработки прикладного ПО устройств на примере языка ассемблера AVR-микроконтроллеров.

Теоретические сведения.

Работа в среде AVR Studio. В состав среды AVR Studio входит редактор исходных текстов, транслятор с языка ассемблера, отладчик и симулятор. Транслятор работает с исходными программами на языке ассемблера, содержащими метки, директивы, команды и комментарии.

Метка представляет собой символическое обозначение адреса (последовательность символов, заканчивающаяся двоеточием). Метки используются для указания места в программе, в которое передаётся управление при переходах, а также для задания имён переменных.

Директивы являются инструкциями для транслятора и не заносятся в исполняемый код программы. Список директив приведён ниже.

Директивы ассемблера микроконтроллеров семейства AVR

Директива	Описание
.BYTE	Зарезервировать байты в оперативной памяти данных
.CSEG	Сегмент программы
.DB	Определить (задать) байты в памяти программ или энергонезависимой памяти данных
.DEF	Назначить регистру общего назначения символическое имя
.DEVICE	Определить устройство, для которого транслируется программа
.DSEG	Сегмент данных
.DW	Определить (задать) слова в памяти программ или энергонезависимой памяти данных
.ENDM, .ENDMACRO	Конец макроса
.EQU	Установить постоянное выражение
.ESEG	Сегмент энергонезависимой памяти данных
.EXIT	Выйти из файла
.INCLUDE	Вложить другой файл
.LIST	Включить генерацию листинга
.LISTMAC	Включить разворачивание макросов в листинге
.MACRO	Начало макроса
.NOLIST	Выключить генерацию листинга
.ORG	Установить положение в сегменте
.SET	Установить переменный символический эквивалент выражения

Директивы могут иметь один или несколько параметров. Команды записываются в программе в виде мнемонического обозначения выполняемой операции и могут иметь один или несколько операндов, т. е. аргументов, с которыми они вызываются.

Транслятор позволяет указывать операнды в различных системах счисления: десятичной (по умолчанию, например, 15, 154), шестнадцатеричной (префикс 0x или $, например, 0x0f, $0f, 0x9a, $9a), восьмеричной (префикс – нуль, например, 017, 0232) и двоичной (префикс 0b, например, 0b00001111, 0b10011010).

Строка программы должна быть не длиннее 120 символов и может иметь одну из четырёх форм: [метка:] .директива [параметры] [;Комментарий] [метка:] команда [операнды] [;Комментарий] [;Комментарий] [Пустая строка] Позиции в квадратных скобках необязательны.

Текст после точки с запятой и до конца строки является комментарием и транслятором игнорируется. Включение в текст программы комментариев является признаком хорошего стиля программирования и облегчает её сопровождение. Кроме того, улучшению читаемости также способствует форматирование текста программы. При программировании на ассемблере выполнение этих правил особенно важно, т. к. программы на языке ассемблера отличаются существенной неудобочитаемостью. Указать тип микроконтроллера, для которого транслируется программа, позволяет директива .device, например: .device ATmega8535 ; программа для микроконтроллера ATmega8535

При наличии в программе команд, не поддерживаемых указанным в директиве микроконтроллером, транслятор выдаёт соответствующее предупреждение. Входным для транслятора является файл .asm с текстом программы на языке ассемблера. Транслятор создаёт четыре новых файла: файл листинга (.lst), объектный файл (.obj), файл-прошивку памяти программ (.hex) и файл-прошивку энергонезависимой памяти данных (.eep). Файл листинга – это отчёт транслятора о своей работе.

На рисунке ниже приведена часть листинга трансляции программы, в которой числа 2, 5 и 19 заносятся соответственно в регистры R17, R18 и R19; вычисляется произведение и сумма содержимого регистров R17 и R18; из суммы содержимого регистров R17 и R18 вычитается содержимое регистра R19.

```
000000 e012    ldi R17, 2      ; загрузка числа 2 в регистр R17
000001 e053    ldi R18, 5      ; загрузка числа 5 в регистр R18
000002 e133    ldi R19, 19     ; загрузка числа 19 в регистр R19
000003 9f12    mul R17, R18    ; умножение R17 на R18, результат в R1:R0
000004 0f13    add R17, R18    ; сложение R17 и R18, результат в R17
000005 1b31    sub R19, R17    ; вычитание R17 из R19, результат в R19
000006 cfff    met: rjmp met   ; бесконечный цикл (для отладки)
```

Листинг содержит исходный текст транслируемой программы, каждой команде которой поставлены в соответствие машинные коды (правый столбец чисел) и адреса ячеек памяти программ, в которых они будут размещены (левый столбец чисел). Машинные коды и адреса приводятся в шестнадцатеричной системе счисления.

Объектный файл имеет специальный формат и используется для отладки программы с помощью симулятора-отладчика среды AVR Studio. Файл прошивки памяти программ служит для занесения отлаженной программы в память программ микроконтроллера. Файл прошивки EEPROM-памяти данных предназначен для загрузки информации в энергонезависимую память данных. Операции загрузки памяти программ и энергонезависимой памяти данных выполняются с помощью специальных аппаратных средств (программаторов).

Практическая часть.

Выполнить разработку программы в среде AVR Studio*, проделав следующие операции.

1. Создать новый проект, воспользовавшись командой New Project меню Project. В появившемся диалоговом окне в поле Project Name ввести имя создавае- мого проекта без расширения (информация о проекте сохраняется в файле с расширением .aps). В поле Location указать место размещения файлов проекта на диске (путь); в поле Project type выбрать пункт AVR Assembler (исходные тексты программ разрабатываются на языке ассемблера). Для создания файла исходной программы установить флажок Create initial File. Задать имя файла программы, отличающееся от имени проекта, можно в поле Initial File (расши- рение .asm файлов программ на ассемблере устанавливается автоматически). Создание каталога для хранения файлов проекта обеспечивается установкой флажка Create Folder. Рекомендуется каждый проект размещать в отдельном каталоге. Нажать кнопку «Next».

2. В группе Select debug platform and device в поле Debug Platform указать способ отладки создаваемого проекта – AVR Simulator (симулятор- отладчик), в поле Device выбрать тип микроконтроллера, для которого создаётся программа (ATmega8535). Нажать кнопку «Finish». На экране появится дерево иерархии проекта (окно Workspace, закладка Project) и окно редактора исходных текстов программ. Если при создании проекта не был установлен флажок Create initial file, создать файл исходного текста программы можно командой New File меню File.

3. Ввести и отредактировать текст программы. Сохранить файл, воспользовавшись командой Save меню File. Если файл с исходным текстом программы уже существует, его можно включить в проект командой Add existing File меню Project или контекстного меню окна иерархии проекта при выделенной группе Assembler.

4. Провести трансляцию созданной программы, воспользовавшись командой Build and run меню Project (или сочетанием клавиш Ctrl+F7 на клавиатуре). Перед трансляцией убедиться, что установлен флажок List file в диалоговом окне AVR Assembler, вызов которого осуществляется командой AVR Assembler Setup меню Project (это необходимо для создания листинга трансляции). По окончании трансляции в окне Output на закладке Build появится информация о результатах трансляции. Открыть файл листинга можно из дерева иерархии проекта (окно Workspace, закладка Project). После этого с помощью команды Save Project меню Project сохранить изменения в файле проекта.

5. Оформить отчет. Отчёт должен содержать: титульный лист с указанием номера и названия работы, номера группы и фамилий выполнивших работу; цель работы; схему программной модели AVR-микроконтроллера; перечень этапов разработки прикладного ПО для встраиваемых МП (МК); распечатку листинга трансляции созданной программы с расшифровкой одной из строк.

Контрольные вопросы

1. Этапы разработки ПО для встраиваемых микропроцессоров.
2. Формат строки программы на ассемблере для AVR-микроконтроллеров.
3. Состав листинга трансляции.

Практическая работа №66. Программирование микропроцессорных систем (логические основы).

Цель работы: ознакомление с аппаратными и программными средствами отладки ПО; изучение команд отладчика среды AVR Studio; приобретение навыков отладки программ под управлением отладчика.

Основные сведения.

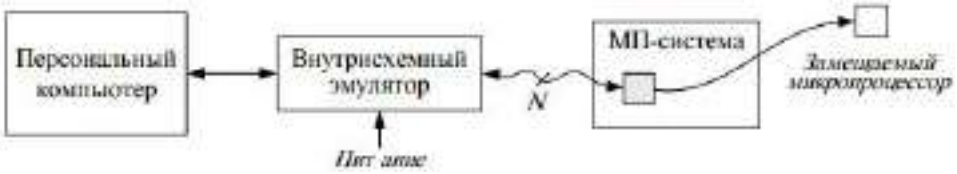
Особенность отладки ПО устройств на базе встраиваемых МП (в том числе однокристальных микроконтроллеров) состоит в отсутствии в их составе развитых средств для реализации пользовательского интерфейса и ограниченных возможностях системного ПО. В то же время, именно для встраиваемых микропроцессорных систем этап отладки является чрезвычайно ответственным, так как для них характерна тесная взаимосвязь работы ПО и аппаратных средств. Взаимодействие микропроцессора (микроконтроллера) с датчиками и исполнительными устройствами происходит путём передачи данных через регистры периферийных устройств (регистры ввода-вывода). Отдельные разряды таких регистров задают режимы работы периферийных устройств, имеют смысл готовности к обмену, завершения передачи данных и т. п. Состояние этих разрядов может устанавливаться как программно, так и аппаратно. При отладке ПО часто приходится переходить на уровень межрегистровых передач и проверять правильность установки отдельных разрядов. Кроме того, на этапе отладки может производиться оптимизация алгоритма, нахождение критиче- ских участков кода и проверка надёжности разработанного ПО. Для решения указанных задач применяются аппаратные и программные средства отладки ПО.

К аппаратным средствам отладки относятся аппаратные эмуляторы и проверочные модули. Аппаратные эмуляторы предназначены для отладки программного и аппаратного обеспечения микропроцессорных систем в режиме реального времени. Они работают под управлением «ведущего» компьютера, оснащённого специальным ПО – программами-отладчиками. Основными видами аппаратных эмуляторов являются:

− внутрисхемные эмуляторы или эмуляторы-приставки, замещающие микропроцессор в отлаживаемой системе;

− внутрикристальные эмуляторы, представляющие собой одно из внутренних устройств микропроцессора.

Внутрисхемный эмулятор (In-Circuit Emulator, ICE) – это устройство, содержащее аппаратный имитатор процессора и схемы управления имитатором. При отладке с помощью эмулятора микропроцессор извлекается из отлаживае- мой системы, на его место подключается контактная колодка, количество и на- значение контактов которой идентично выводам замещаемого микропроцессора.



С помощью гибкого кабеля контактная колодка соединятся с эмулятором. Управление процессом отладки осуществляется с персонального компьютера. Эмуляторам-приставкам присущи следующие недостатки: высокая стоимость, недостаточная надёжность, высокое энергопотребление, влияние на электрические характеристики цепей, к которым подключается эмулятор. Внутрикристальные эмуляторы (On-Chip Emulator) позволяют проводить отладку программ без извлечения микропроцессора из системы. При этом осуществляется непосредственный контроль за выполнением программы, так как средства внутрикристальной отладки обеспечивают прямой доступ к регистрам, памяти и периферии микропроцессора. Наиболее распространённым средством внутрикристальной отладки является последовательный интерфейс IEEE 1149.1, известный как JTAG (Joint Test Action Group – Объединённая рабочая группа по автоматизации тестирования). Последовательный отладочный порт JTAG микропроцессора с помощью специального устройства сопряжения подключается к компьютеру, чем обеспечивается доступ к отладочным средствам процессора.

Такой способ отладки также называют сканирующей эмуляцией. Достоинствами этого способа является возможность выполнения различных действий на процессоре без его изъятия из системы, использование малого числа выводов процессора и поддержка его максимальной производительности без изменения электрических характеристик системы.

Проверочные модули предназначены для быстрой отладки программного обеспечения в реальном масштабе времени. Проверочные модули бывают двух видов: стартовые наборы и отладочные платы. Стартовые наборы (Starter Kit) предназначены для обучения работе с конкретным микропроцессором. Стартовый набор позволяет изучить характеристики микропроцессора, отладить не слишком сложные программы, выполнить несложное макетирование, проверить возможность применения микропроцессора для решения конкретной задачи. В состав стартового набора входит плата, ПО и комплект документации. На плате устанавливается микропроцессор, устройство загрузки программ, последовательные или параллельные порты, разъёмы для связи с внешними устройствами и другие элементы. Плата подключается к компьютеру через параллельный или последовательный порт. Стартовые наборы удобны на начальном этапе работы с микропроцессором. Отладочные платы (Evaluation Board) предназначены для проверки разработанного алгоритма в реальных условиях. Они позволяют проводить отладку и оптимизацию алгоритма с использованием установленной на плате периферии, а также изготовить на базе платы законченное устройство. Обычно на плате размещается микропроцессор, схемы синхронизации, интерфейсы расширения памяти и периферии, схема электропитания и др. Плата подключается к компьютеру через параллельный или последовательный порт или непосредственно устанавливается в слот PCI.

Основными программными средствами отладки являются симуляторы и отладчики. Симуляторы (simulator) или симуляторы системы команд представляют собой программы, имитирующие работу того или иного процессора на уровне его команд. Симуляторы обычно используются для проверки программ или её отдельных частей перед испытанием на аппаратных средствах. Отладчики (debugger) представляют собой программы, предназначенные для анализа работы созданного программного обеспечения.

Можно указать следующие возможности отладчиков.

1. Пошаговое выполнение. Программа выполняется последовательно команда за командой с возвратом управления отладчику после каждого шага.

2. Прогон. Выполнение программы начинается с указанной команды и осуществляется без остановки до конца программы.

3. Прогон с контрольными точками. При выполнении программы происходит останов и передача управления отладчику после выполнения команд с адресами, указанными в списке контрольных точек.

4. Просмотр и изменение содержимого регистров и ячеек памяти. Пользователь имеет возможность выводить на экран и изменять (модифицировать) содержимое регистров и ячеек памяти. Отладчики ПО встраиваемых микропроцессоров обычно используются совместно с внутрисхемными или внутрикристальными эмуляторами, а также могут работать в режиме симулятора. Некоторые отладчики позволяют также выполнять профилирование, т. е. определять действительное время выполнения некоторого участка программы. Иногда функцию профилирования выполняет специальная программа – профилировщик (profiler).

Средства отладки ПО AVR-микроконтроллеров. Аппаратные средства отладки программного обеспечения AVR-микроконтроллеров представлены внутрисхемным эмулятором ICE50, внутрикристальным эмулятором JTAG ICE, а также стартовым набором STK500.

К программным средствам отладки ПО AVR-микроконтроллеров относятся отладчик и симулятор, входящие в состав среды AVR Studio. Отладчик среды AVR Studio позволяет проводить отладку программ как в исходных кодах (например, ассемблера), так и в кодах дизассемблера (оттранслированной или скомпилированной программы, записанной с помощью мнемоник ассемблера). Вызов окна с кодом дизассемблера производится командой Disassembler меню View или командой Goto Disassembly контекстного меню редактора исходного текста. Обратное переключение в окно исходного текста осуществляется командой Goto Source контекстного меню окна Disassembler.

Отладчик среды AVR Studio может использоваться с внутрисхемным эмулятором ICE50, внутрикристальным эмулятором JTAG ICE, отладочной платой STK500 или симулятором. Указание способа отладки производится при создании проекта. Симулятор среды AVR Studio предназначен для предварительной отладки программ без применения аппаратных средств. В дальнейшем в настоящем практикуме для отладки создаваемых программ предполагается применение отладчика среды AVR Studio в режиме симулятора.

Отладка ПО в среде AVR Studio. Команды отладчика в программе AVR Studio находятся в меню Debug.

Переход в режим отладчика в среде AVR Studio осуществляется автоматически при использовании для трансляции программы команды Build and Run или командой Start Debugging меню Debug при использовании для трансляции команды Build.

Выход из режима отладчика производится командой Stop Debugging меню Debug.

Пошаговое выполнение программы задаётся командами Step Into, Step Over меню Debug. Команда Step Into позволяет выполнить одну команду программы (в т. ч. команду вызова подпрограммы). Для завершения выполнения подпрограммы может использоваться команда Step Out. Команда Step Over также выполняет одну команду программы, но если это команда вызова подпрограммы, последняя полностью выполняется за один шаг.

Следующая выполняемая команда (команда, адрес которой содержится в программном счётчике) обозначается символом  в окне исходного текста программы.

Сброс выполнения программы осуществляется с помощью команды Reset.

Прогон (запуск или продолжение выполнения) программы осуществляется командой Run.

Для остановки выполнения программы служит команда Break.

Контрольные точки представляют собой специальные маркеры для программы-отладчика и могут быть трёх типов: точки останова, точки трассировки и точки наблюдения. Точки останова задаются командой Toggle Breakpoint меню Debug или контекстного меню редактора исходного текста программы. Точка останова обозначается в редакторе исходного текста символом слева от помечаемой строки. Просмотреть заданные точки останова можно на закладке Breakpoints окна Output; там же точки останова могут быть запрещены (путём сброса флажка напротив точки останова) и разрешены (путём установки флажка). При достижении точки останова во время прогона программы её выполнение приостанавливается. Повторный вызов команды установки точки останова на той же строке программы приводит к удалению точки останова. Удалить все заданные точки останова позволяет команда Remove Breakpoints меню Debug или команда Remove all Breakpoints контекстного меню закладки Breakpoints окна Output. Параметры точки останова задаются в диалоговом окне Breakpoint Condition, вызов которого осуществляется командой Breakpoints Properties контекстного меню редактора исходного текста программы. Установка флажка Iterations позволяет задать количество итераций (повторных выполнений) команды до останова прогона программы. При установке флажка Watchpoint по достижению точки останова производится только обновление значений регистров и ячеек памяти в окнах просмотра. Флажки Iterations и Watchpoint не должны устанавливаться одновременно. Установка флажка Show message обеспечивает отображение сообщений о достижении точки останова на закладке Breakpoints окна Output. Вызов диалогового окна задания свойств и удаление точки останова может быть произведено из контекстного меню закладки Breakpoints окна Output.

Точки трассировки предназначены для контроля выполнения программы в режиме реального времени. Трассировка позволяет отслеживать так называемую трассу программы – изменение содержимого регистров и ячеек памяти при выполнении определённых команд (команд, по адресам которых заданы точки трассировки). В среде AVR Studio функция трассировки может использоваться только при отладке программы с применением внутрисхемного эмулятора; при работе в режиме симулятора функция трассировки недоступна. Точки наблюдения задаются командой Add to Watch контекстного меню ре- дактора исходного текста программы. Точки наблюдения представляют собой символические имена регистров или ячеек памяти, содержимое которых необходимо отслеживать. При выполнении команды Add Watch на экране появляется окно Watches, разделённое на четыре столбца: Name (символическое

имя точки наблюдения), Value (значение), Type (тип), Location (местонахождение). Новая точка наблюдения может быть также задана в выделенной ячейке столбца Name окна Watches или командой Quickwatch в окне редактора исходного текста программы (при этом курсор должен находиться на имени регистра или ячейки памяти). Значения, отображаемые в столбце Value, обновляются при изменении содержимого соответствующего регистра или ячейки памяти. Удалить задан- ные точки наблюдения можно их окна Watches. Отладчик среды AVR Studio также обеспечивает следующие функции: выполнение до курсора (команда Run to Cursor меню Debug) и последовательное выполнение команд с паузами между ними (команда Auto Step меню Debug). Для удобства использования в процессе отладки ряд команд отладчика доступен с клавиатуры (табл. 2).

Таблица 2

Команда отладчика	Клавиша	Команда отладчика	Клавиша
Run	F5	Step Into	F11
Break	Ctrl+F5	Step Out	Shift+F11
Reset	Shift+F5	Step Over	F10
Run to Cursor	Ctrl+F10	Toggle Breakpoint	F9

Для просмотра и изменения содержимого регистров и ячеек памяти служат команды Registers, Memory, Memory 1, Memory 2, Memory 3 меню View.

По команде Registers на экране отображается окно Registers, в котором приводятся шестнадцатеричные представления содержимого РОН. Изменение (модификация) содержимого регистров производится путём двойного щелчка мышью. Наблюдение за содержимым РОН может быть также произведено с помощью дерева устройств микроконтроллера, находящегося на закладке I/O окна Workspace. Для этого необходимо раскрыть объекты Register 0-15 и Register 16-31 щелчком мыши по знаку «+».

Команды Memory, Memory 1, Memory 2, Memory 3 обеспечивают вызов окон Memory, служащих для отображения содержимого ячеек оперативной и энергонезависимой памяти данных, памяти программ, регистров ввода-вывода и РОН. Выбор типа памяти, отображаемой в окне Memory, производится с помощью списка, расположенного в панели управления окна (Data – оперативная память данных, Eeprom – энергонезависимая память данных, I/O – регистры ввода-вывода, Program – память программ, Register – РОН).

Для наблюдения за состоянием процессора необходимо раскрыть объект Processor закладки I/O окна Workspace. При этом будет отображена следующая информация: содержимое программного счётчика (Program Counter); содержимое указателя стека (Stack Pointer), количество тактов, прошедших с начала выполнения (Cycle Counter); содержимое 16-разрядных регистров- указателей X, Y и Z; тактовая частота (Frequency); затраченное на выполнение время (Stop Watch).

Для наблюдения содержимого регистров ввода-вывода необходимо раскрыть объект I/O закладки I/O окна Workspace, где – тип микроконтроллера. Регистры ввода-вывода, входящие в объект I/O, сгруппированы по типам периферийных устройств.

Модифицированные значения содержимого регистров и ячеек памяти действуют только во время текущего сеанса отладки, в исходный текст программы изменения не заносятся.

Практическая часть.

Провести отладку созданной в предыдущей практической работе программы с помощью симулятора-отладчика среды AVR Studio, проделав следующие операции.

1. Выполнить программу в пошаговом режиме, отслеживая изменение содержимого используемых в программе регистров. Обратить внимание на изменение содержимого программного счётчика. Сравнить содержимое программного счётчика при выполнении команд с их адресами в памяти программ, приведёнными в листинге трансляции и окне памяти программ.

2. Выполнить прогон программы. Проверить правильность результата работы программы.

3. Задать точку останова на команде загрузки в РОН числа B. Включить режим отображения сообщений о достижении точки останова. Выполнить прогон программы с

контрольными точками. Задать точку останова на команде умножения. Выполнить прогон программы с контрольными точками. Удалить заданные точки останова.

4. Задать точки наблюдения в используемых РОН. Выполнить программу в пошаговом режиме, отслеживая изменение их содержимого.

5. Оформить отчет. Отчёт должен содержать: титульный лист с указанием номера и названия работы, номера группы и фамилий выполнивших работу; цель работы; краткие теоретические сведения (классификацию средств отладки ПО, перечень основных функций программ-отладчиков); список использованных команд отладчика AVR Studio с указанием их назначения.

Контрольные вопросы.

1. Классификация средств отладки прикладного ПО встраиваемых МП.
2. Виды и особенности аппаратных средств отладки ПО.
3. Основные функции программных средств отладки ПО.
4. Пошаговое выполнение программы и его возможности.
5. Особенности прогона программы с контрольными точками.
6. Контрольные точки: типы, назначение, использование.

Практическая работа №67. Программирование микропроцессорных систем (изучение способов адресации операндов в AVR-микроконтроллерах).

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

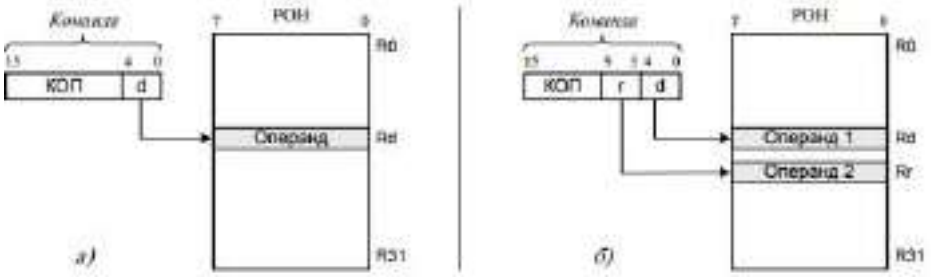
Основные сведения.

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 ис- пользовать в качестве первого операнда только регистры из старшей половины РОН (R16…R31).

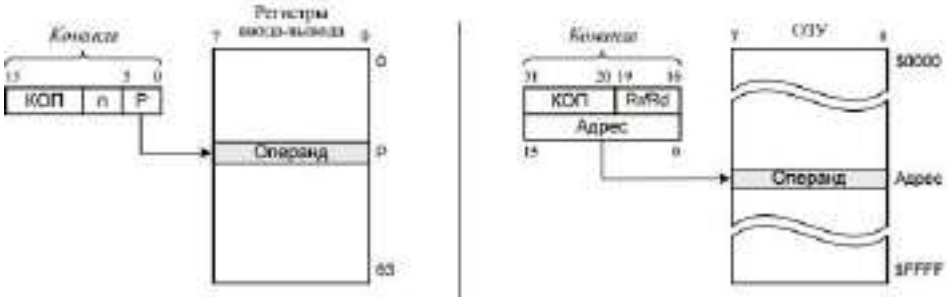


При прямой регистровой адресации регистра ввода-вывода операнд содержится в регистре ввода-вывода, указанном в команде. Адрес регистра ввода- вывода хранится в шести разрядах слова команды, где n определяет адрес регистра-источника или регистра-приёмника в РОН). Прямая регистровая адресация регистров 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

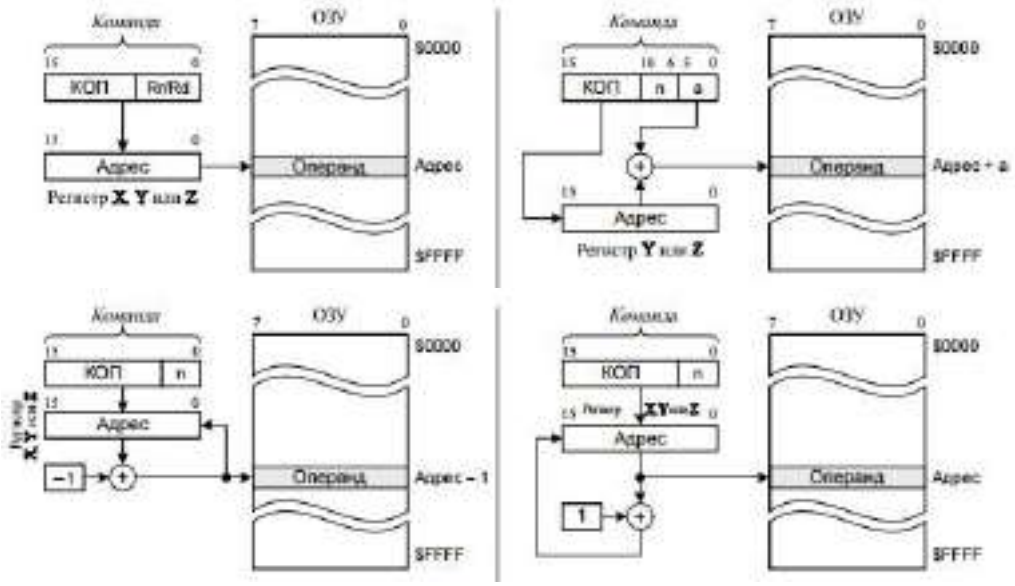
Для адресации оперативной памяти данных используются пять способов адресации: непосредственная, косвенная, косвенная со смещением, косвенная с предекрементом и косвенная с постинкрементом.

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 требуемое 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 продолжается 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 размещается 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 №68. Программирование микропроцессорных систем (контроль на четность/нечетность).

Цель работы: изучение методов контроля на четность/нечетность.

Основные сведения.

Наиболее часто для обнаружения ошибок используется контроль на четность/нечетность передаваемой информации. При таком контроле блок информации, передаваемый от источника к приемнику, должен содержать соответственно четное/нечетное число единиц.

Источник формирует передаваемый блок информации так, чтобы это выполнялось. Приемник проверяет выполнение этого условия. Если четность/нечетность нарушена, то считается, что произошла ошибка при передаче информации. Приемник сообщает об этом источнику, и источник заново посылает этот блок информации. Передаваемый блок информации состоит из информационных битов и одного контрольного:

ПЕРЕДАВАЕМЫЙ БЛОК ИНФОРМАЦИИ	
ИНФОРМАЦИОННЫЕ БИТЫ	КОНТРОЛЬНЫЙ БИТ
$X_1,X_2,\ldots,X_n$	K

При контроле на четность в бит K записывается сумма по модулю 2 информационных разрядов. При контроле на нечетность записывается инверсное значение этой суммы.

Рассмотрим пример такого контроля. Пусть от источника к приемнику надо переслать 8 бит информации с контролем на четность.

Тогда пересылаемый блок информации должен содержать 9 бит.

Пример. Переслать код 01110101.

Побитное сложение по модулю для этого кода дает значение, равное 1. Тогда значение контрольного бита должно равняться 1, чтобы передаваемый код содержал четное число единиц.

ПЕРЕДАТЧИК										ПРИЕМНИК										комментарий
информационные биты									контрольный бит	принятые биты									сумма по mod 2	
0	1	1	1	0	1	0	1	1	0	1	1	1	0	1	0	1	1	0		нет ошибки
									0	1	1	1	**1**	1	0	1	1	1		есть ошибка

Код Джонсона

Последовательность чисел в этом коде моделируется односторонним последовательным заполнением его разрядов вначале единицами, а затем нулями (таблица). Код Джонсона легко формируется с помощью регистров сдвига и легко дешифруется.

Код «1 из m»

Для случая m=8 представлен в таблице. Этот код характерен тем, что в любой кодовой комбинации присутствует только одна единица, что позволяет легко находить ошибки в случае искажения кода, и не требуется его дешифрация. Данный код, как и код Джонсона, является избыточным, требующим для своего изображения больше разрядов, чем соответствующие неизбыточные коды.

S	КОД ДЖОНСОНА	КОД «1 из 8»
0	0 0 0 0	0 0 0 0 0 0 0 1
1	0 0 0 1	0 0 0 0 0 0 1 0
2	0 0 1 1	0 0 0 0 0 1 0 0
3	0 1 1 1	0 0 0 0 1 0 0 0

4	1 1 1 1	0 0 0 1 0 0 0 0
5	1 1 1 0	0 0 1 0 0 0 0 0
6	1 1 0 0	0 1 0 0 0 0 0 0
7	1 0 0 0	1 0 0 0 0 0 0 0

Код хэмминга для обнаружения и исправления ошибки

Исправлять ошибку при приеме информации труднее, чем ее обнаруживать. Исправление ошибки предполагает два совмещенных процесса: обнаружение факта, что есть ошибка, и определение ее места. После решения этих двух задач, исправление тривиально – надо инвертировать значение ошибочного бита. В наземных каналах связи, где вероятность ошибки невелика, обычно используется только метод обнаружения ошибки и повторной пересылки блока информации, содержавшего ошибку. Для спутниковых каналов с типичными для них большими задержками коррекция ошибки становится необходимой. Здесь используется код Хэмминга.

Код Хэмминга представляет собой блочный код, который позволяет выявить и исправить ошибочно переданный бит в пределах переданного блока. Блочными называются коды, в которых информационный поток символов разбивается на отрезки, и каждый из них преобразуется в определенную последовательность (блок) кодовых символов. В блочных кодах кодирование при передаче (формирование проверочных элементов) и декодирование при приеме (обнаружение и исправление ошибок) выполняются в пределах каждой кодовой комбинации (блока) в отдельности по соответствующим алгоритмам.

Обычно код Хэмминга характеризуется двумя целыми числами, например, код (11,7), используемый при передаче семибитных ASCII-кодов. Такая запись говорит, что при передаче семибитного кода используется дополнительно четыре контрольных бита (7+4=11). При этом предполагается, что может иметь место ошибка в одном бите и что ошибка в двух или более битах существенно менее вероятна. С учетом этого исправление ошибки осуществляется с определенной вероятностью.

При рассмотрении кода Хэмминга требуется знать, что такое кодовое расстояние. Кодовое расстояние между двумя двоичными кодами одинаковой длины определяется количеством битов, в которых эти коды отличаются.

Пример 1. Кодовое расстояние между «кодом 1» и «кодом 2» равно 1.

КОД 1 0 1 0

КОД 2 0 1 1

Пример 2. Кодовое расстояние между «кодом 1» и «кодом 2» равно 2.

КОД 1	0	0	1
КОД 2	1	1	1

Правило вычисления кодового расстояния — комбинации кодов суммируются по модулю 2, после чего подсчитывается количество единиц в полученной сумме.

Например,

КОД 1	1	0	0	1	0
КОД 2	1	0	1	1	1
КОД1⊕КОД2	0	0	1	0	1

Кодовое расстояние равно 2, т.к. число единиц в сумме – 2.

Можно обнаружить ошибку только, если между используемыми кодовыми комбинациями есть необходимое для этого кодовое расстояние, т.е. между соседними используемыми

кодовыми комбинациями есть другие комбинации. Эти кодовые комбинации не используются для передачи информации, и их получение на приеме свидетельствует об ошибке передачи информации в канале связи. Для заданного кода минимальное кодовое расстояние – это минимальное из всех расстояний всех пар кодовых слов.

В обычном равномерном непомехоустойчивом коде число разрядов n в кодовых комбинациях определяется числом сообщений и основанием кода. Коды, у которых все кодовые комбинации разрешены к передаче, называются простыми или равнодоступными и являются полностью безызбыточными. Безызбыточные первичные коды обладают большой «чувствительностью» к помехам. Внесение избыточности при использовании помехоустойчивых кодов обязательно связано с увеличением числа разрядов (длины) кодовой комбинации.

В этом случае все множество $N=2^n$ комбинаций можно разбить на два подмножества: подмножество разрешенных комбинаций, т.е. обладающих определенными признаками, и подмножество запрещенных комбинаций, этими признаками не обладающих. Помехоустойчивый код отличается от обычного тем, что в канал передаются не все кодовые комбинации N, которые можно сформировать из имеющегося числа разрядов n, а только их часть N_K, которая составляет подмножество разрешенных комбинаций.

Если при приеме выясняется, что кодовая комбинация принадлежит к запрещенным, то это свидетельствует о наличии ошибки, т.е. таким образом решается задача обнаружения ошибок. При этом принятая комбинация не декодируется (не принимается решение о приеме сообщения). Помехоустойчивые коды называют корректирующими кодами. Корректирующие свойства избыточных кодов зависят от правила их построения, определяющего структуру кода, и параметров кода.

Например, при $n=3$ можно составить 8 кодов (000, 001, 010, 011, 100, 101, 110, 111).

Пусть все восемь кодовых комбинаций являются разрешенными (допустимыми), тогда кодовое расстояние между соседними комбинациями равно 1. Т.к. в данном случае отсутствует какой-либо признак, позволяющий судить о появлении ошибки в кодовой комбинации, то такой код не является помехозащищенным. Допустим, что лишь четыре из восьми кодовых комбинаций считаются разрешенными, например это комбинации 001, 010, 100 и 111. Тогда кодовое расстояние равно 2, причем искажение символа в одном из разрядов приводит к получению запрещенной кодовой комбинации (000, 011, 101 или 110), что легко выявляется при проверке. Полученный таким образом двоичный код называют кодом с обнаружением одиночной ошибки. Для кодового расстояния равного 3 в качестве разрешенных кодовых комбинаций можно принять, например, 010 и 101. При этом обеспечивается возможность не только обнаружения, но и исправления одиночной ошибки. Действительно, получение запрещенной кодовой комбинации 110 указывает на наличие ошибки, для исправления которой необходимо перейти к ближайшей из разрешенных кодовых комбинаций (в данном случае – 010). Данный код позволяет обнаруживать и двойные ошибки, так как при одновременном искажении символов в двух разрядах кодовой комбинации последняя также попадает в число запрещенных.

Первые работы по корректирующим кодам принадлежат Хэммингу, который ввел понятие минимального кодового расстояния и предложил код, позволяющий однозначно указать ту позицию в кодовой комбинации, где произошла ошибка. К «*M*» (*message*) информационным битам в коде Хэмминга добавляется «*C*» (*control*) проверочных битов для определения местоположения ошибочного бита.

Таким образом, код Хэмминга состоит из информационных и контрольных битов. Информационные биты будем обозначать через M, контрольные биты – через C:

$M=\{M_1,M_2,\ldots,M_n\}$;

$C=\{C_1, C_2,\ldots C_k\}$;

$N=M+C$ – общее количество передаваемых битов в канал связи (передаваемый блок).

Сущность кода Хэмминга заключается в том, что местоположение контрольных и информационных битов определяется по правилу, которое устанавливает зависимость, позволяющую в случае ошибки определить ее местоположение, т.е. номер бита.

Основные положения для получения кода Хэмминга состоят в следующем:

1. Число, составленное из контрольных разрядов, должно указывать номер бита, в котором произошла ошибка, или 0, если нет ошибки. Исходя из этого, если произошла ошибка, проверочное C-битовое число – должно быть в диапазоне от 1 до (2^C-1). Из этого следует, что $2^C-1 \geq N$; тогда $2^C \geq N+1$; умножим левую и правую части на 2^M, получим $2^M \times 2^C \geq 2^M \times (N+1)$, откуда $2^{(M+C)} \geq 2^M \times (N+1)$, тогда $2^N \geq 2^M \times (N+1)$; делим левую и правую части на $(N+1)$, получаем условие

$$\frac{2^N}{N+1} \geq 2^M.$$

С использованием этого условия и определяется минимально необходимое число контрольных битов для передачи заданного количества информационных.

2. Контрольные биты в коде Хэмминга занимают позиции с номерами, равными 2^n, где n=0,1,3,… и т.д. Причем позиции кода номеруются справа налево, начиная с 1. Информационные биты располагаются в остающихся позициях справа налево по возрастанию весов разрядов.

3. Значение каждого контрольного бита получается сложением по модулю 2 тех информационных битов, для которых в номере кодовой позиции информационных битов, записанном через контрольные разряды, значение бита равно 1. Эта формулировка будет более понятна, если посмотреть ее использование по табл. 1 или 2.

Практическая часть.

Вначале построим код Хэмминга для передачи четырехбитового кода.

Условие 1. Это условие будет выполняться при N=7:

$$\left\{\frac{27}{7+1}=\frac{128}{8}=16\right\} \geq \left\{2^4=16\right\}.$$

Таким образом, для четырех информационных битов (M=4) требуются три контрольных бита (C=3).

Условие 2. Определяем местоположение информационных и контрольных битов в коде.

Положение информационных и контрольных битов							
Номер позиции кода	4	3	2	3	1	2	1

Условие 3. Определяем логические выражения для вычисления контрольных битов. Для этого построим таблицу.

Таблица 1

ПОЗИЦИЯ КОДА	КОНТРОЛЬНЫЕ БИТЫ		
	C_3	C_2	C_1
3 бит (M_1)	0	1	1
5 бит (M_2)	1	0	1
6 бит (M_3)	1	1	0
7 бит (M_4)	1	1	1
СУММИРУЕМЫЕ ПО МОДУЛЮ 2 БИТЫ	M_2 M_3 M_4	M_1 M_3 M_4	M_1 M_2 M_4

Таким образом, передатчик вычисляет значения контрольных разрядов по следующим логическим выражениям:

$C_1=M_1 \oplus M_2 \oplus M_4$;

$C_2=M_1 \oplus M_3 \oplus M_4$;

$C_3=M_2 \oplus M_3 \oplus M_4$.

Приемник проверяет правильность принятого кода, вычисляя следующие выражения:

$C_{11}=C_1 \oplus M_1 \oplus M_2 \oplus M_4$;

$C_{12}=C_2 \oplus M_1 \oplus M_3 \oplus M_4$;

$C_{13}=C_3 \oplus M_2 \oplus M_3 \oplus M_4$.

Построим код Хэмминга для передачи семибитового кода.

Условие 1. Это условие будет выполняться при N=11:

$$\left\{\frac{2^{11}}{11+1}=\frac{2048}{12}\approx 170\right\} \geq \{2^7=128\}.$$

Таким образом, для семи информационных битов (M=7), требуются четыре контрольных бита (C=4).

Условие 2. Определяем местоположение информационных и контрольных битов в коде.

Положение информационных и контрольных битов: 7 6 5 4 4 3 2 3 1 2 1

Номер позиции кода: 1 0

Условие 3. Определяем логические выражения для вычисления контрольных битов. Для этого построим таблицу.

Таблица 2

ПОЗИЦИЯ КОДА	КОНТРОЛЬНЫЕ БИТЫ			
	C_4	C_3	C_2	C_1
3 бит (M_1)	0	0	1	1
5 бит (M_2)	0	1	0	1
6 бит (M_3)	0	1	1	0

7 бит (M_4)	0	1	1	1
9 бит (M_5)	1	0	0	1
10 бит (M_6)	1	0	1	0
11 бит (M_7)	1	0	1	1
СУММИРУЕМЫЕ ПО МОДУЛЮ 2 БИТЫ	M_5 M_6 M_7	M_2 M_3 M_4	M_1 M_3 M_4 M_6 M_7	M_1 M_2 M_4 M_5 M_7

Таким образом, передатчик вычисляет значения контрольных разрядов по следующим логическим выражениям:

$C_1=M_1 \oplus M_2 \oplus M_4 \oplus M_5 \oplus M_7$;

$C_2=M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7$;

$C_3=M_2 \oplus M_3 \oplus M_4$;

$C_4=M_5 \oplus M_6 \oplus M_7$.

Приемник проверяет правильность передачи кода, вычисляя следующие выражения:

$C_{11}=C_1 \oplus M_1 \oplus M_2 \oplus M_4 \oplus M_5 \oplus M_7$;

$C_{12}=C_2 \oplus M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7$;

$C_{13}=C_3 \oplus M_2 \oplus M_3 \oplus M_4$;

$C_{14}=C_4 \oplus M_5 \oplus M_6 \oplus M_7$.

Рассмотрим примеры передачи информации с использованием кода Хэмминга для случаев, когда ошибки при передаче нет, и когда есть.

Начнем с кода (7,4).

Пример. В канал связи нужно передать следующий блок информации: 1101_2.

Передатчик формирует код Хэмминга:

M_4	M_3	M_2	C_3	M_1	C_2	C_1
1	1	0	*	1	*	*

$C_1=M_1 \oplus M_2 \oplus M_4=1 \oplus 0 \oplus 1=0$;

$C_2=M_1 \oplus M_3 \oplus M_4=1 \oplus 1 \oplus 1=1$;

$C_3=M_2 \oplus M_3 \oplus M_4=0 \oplus 1 \oplus 1=0$,

тогда код, передаваемый в канал, будет:

M_4	M_3	M_2	C_3	M_1	C_2	C_1
1	1	0	0	1	1	0

Рассмотрим случай, когда не было ошибки при передаче кода.

Тогда точно такой же код принял приемник. Теперь приемник проверяет правильность передачи кода, вычисляя следующие выражения:

$C_{11}=C_1 \oplus M_1 \oplus M_2 \oplus M_4=0 \oplus 1 \oplus 0 \oplus 1=0$;

$C_{12}=C_2 \oplus M_1 \oplus M_3 \oplus M_4=1 \oplus 1 \oplus 1 \oplus 1=0$;

$C_{13}=C_3 \oplus M_2 \oplus M_3 \oplus M_4=0 \oplus 0 \oplus 1 \oplus 1=0$.

Поскольку значение, составленное из контрольных битов, равно нулю, ошибки нет, и код передается от приемника далее.

Рассмотрим случай, когда была ошибка при передаче кода.

Пусть ошибка произошла в третьем бите, т.е. принят код:

M_4	M_3	M_2	C_3	M_1	C_2	C_1

4	3	2	3	1	2	1
1	1	0	0	**0**	1	0

Теперь приемник проверяет правильность передачи кода, вычисляя следующие выражения:

$C_{11}=C_1 \oplus M_1 \oplus M_2 \oplus M_4=0 \oplus 0 \oplus 0 \oplus 1=1$;

$C_{12}=C_2 \oplus M_1 \oplus M_3 \oplus M_4=1 \oplus 0 \oplus 1 \oplus 1=1$;

$C_{13}=C_3 \oplus M_2 \oplus M_3 \oplus M_4=0 \oplus 0 \oplus 1 \oplus 1=0$.

Поскольку, значение, составленное из контрольных разрядов, равно 011_2, ошибка в бите № 3. Приемник меняет значение этого бита на противоположное.

Теперь рассмотрим код (11,7).

Пример. В канал связи нужно передать следующий блок информации: 1101010_2.

Передатчик формирует код Хэмминга:

M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1
1	1	0	*	1	0	1	*	0	*	*

$C_1=M_1 \oplus M_2 \oplus M_4 \oplus M_5 \oplus M_7=0 \oplus 1 \oplus 1 \oplus 0 \oplus 1=1$;

$C_2=M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7=0 \oplus 0 \oplus 1 \oplus 1 \oplus 1=1$;

$C_3=M_2 \oplus M_3 \oplus M_4=1 \oplus 0 \oplus 1=0$;

$C_4=M_5 \oplus M_6 \oplus M_7=0 \oplus 1 \oplus 1=0$,

тогда код, передаваемый в канал, будет:

M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1
1	1	0	0	1	0	1	0	0	1	1

Рассмотрим случай, когда не было ошибки при передаче кода.

Тогда точно такой же код принял приемник. Теперь приемник проверяет правильность передачи кода, вычисляя следующие выражения:

$C_{11}=C_1 \oplus M_1 \oplus M_2 \oplus M_4 \oplus M_5 \oplus M_7=1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1=0$;

$C_{12}=C_2 \oplus M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7=1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1=0$;

$C_{13}=C_3 \oplus M_2 \oplus M_3 \oplus M_4=0 \oplus 1 \oplus 0 \oplus 1=0$;

$C_{14}=C_4 \oplus M_5 \oplus M_6 \oplus M_7=0 \oplus 0 \oplus 1 \oplus 1=0$.

Поскольку значение, составленное из контрольных битов, равно нулю, ошибки нет, и код передается от приемника далее.

Рассмотрим случай, когда была ошибка при передаче кода.

Пусть ошибка произошла во втором бите, т.е. принят код:

M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1
1	1	0	0	1	0	1	0	0	**0**	1

Теперь приемник проверяет правильность передачи кода, вычисляя следующие выражения:

$C_{11}=C_1 \oplus M_1 \oplus M_2 \oplus M_4 \oplus M_5 \oplus M_7=1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1=0$;

$C_{12}=C_2 \oplus M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7=0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1=1$;

$C_{13}=C_3 \oplus M_2 \oplus M_3 \oplus M_4=0 \oplus 1 \oplus 0 \oplus 1=0$;

$C_{14}=C_4 \oplus M_5 \oplus M_6 \oplus M_7=0 \oplus 0 \oplus 1 \oplus 1=0$.

Поскольку значение, составленное из контрольных битов, равно 0010_2, ошибка во втором бите. Приемник меняет значение этого бита на противоположное.

Код Хэмминга является равномерно защищенным, исправляет ошибки как в информационных, так и в контрольных битах. С его помощью возможно исправление одиночной и обнаружение двойной ошибки. Для того чтобы обнаружить двойную ошибку, в код Хэмминга

вводят дополнительный контрольный бит E_0. Значение E_0 вычисляется как сумма по модулю 2 всех разрядов кода Хэмминга, т.е. он дополняет код Хэмминга по четности. На стороне приемника, кроме получения контрольных разрядов, которые определяют позицию с ошибкой, вычисляется контрольный разряд E_{10}:

$E_{10}=E_0 \oplus$ <сложенные по модулю 2 все остальные биты принятого кода>.

Если $E_{10}=0$, то ошибок нет. Если значение, полученное из контрольных разрядов, не равно 0 и $E_{10}=1$, то имеет место одиночная ошибка, позиция которой определяется значением контрольных разрядов. Если значение $C \neq 0$ и $E_{10}=0$, то имеет место двойная ошибка.

Рассмотрим пример для кода, исправляющего одиночную ошибку и обнаруживающего двойную.

Для кода (11,7) формат блока информации:

M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1	E_0

Пример. В канал связи нужно передать следующий блок информации: 1101010_2.

Передатчик формирует код Хэмминга:

M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1	E_0
1	1	0	*	1	0	1	*	0	*	*	*

$C_1=M_1 \oplus M_2 \oplus M_4 \oplus M_5 \oplus M_7=0 \oplus 1 \oplus 1 \oplus 0 \oplus 1=1;$

$C_2=M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7=0 \oplus 0 \oplus 1 \oplus 1 \oplus 1=1;$

$C_3=M_2 \oplus M_3 \oplus M_4=1 \oplus 0 \oplus 1=0;$

$C_4=M_5 \oplus M_6 \oplus M_7=0 \oplus 1 \oplus 1=0,$

$E_0=C_1 \oplus C_2 \oplus M_1 \oplus C_3 \oplus M_2 \oplus M_3 \oplus M_4 \oplus C_4 \oplus M_5 \oplus M_6 \oplus M_7=1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1=0,$

тогда код, передаваемый в канал, будет:

M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1	E_0
1	1	0	0	1	0	1	0	0	1	1	0

Рассмотрим случай, когда не было ошибки при передаче кода.

Тогда точно такой же код принял приемник. Теперь приемник проверяет правильность передачи кода, вычисляя следующие выражения:

$C_{11}=C_1 \oplus M_1 \oplus M_2 \oplus M_4 \oplus M_5 \oplus M_7=1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1=0;$

$C_{12}=C_2 \oplus M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7=1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1=0;$

$C_{13}=C_3 \oplus M_2 \oplus M_3 \oplus M_4=0 \oplus 1 \oplus 0 \oplus 1=0;$

$C_{14}=C_4 \oplus M_5 \oplus M_6 \oplus M_7=0 \oplus 0 \oplus 1 \oplus 1=0,$

$E_{10}= E_0 \oplus C_1 \oplus C_2 \oplus M_1 \oplus C_3 \oplus M_2 \oplus M_3 \oplus M_4 \oplus C_4 \oplus M_5 \oplus M_6 \oplus M_7=$

$=0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1=0.$

Поскольку значение, составленное из контрольных битов, равно 0, и $E_{10}=0$, ошибки нет, и код передается от приемника далее.

Рассмотрим случай, когда была одиночная ошибка при передаче кода.

Пусть ошибка произошла во втором бите (не считая E_0, т.е. в C_2) и принят код:

M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1	E_0
1	1	0	0	1	0	1	0	0	**0**	1	0

Теперь приемник проверяет правильность передачи кода, вычисляя следующие выражения:

$C_{11}=C_1 \oplus M_1 \oplus M_2 \oplus M_4 \oplus M_5 \oplus M_7=1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1=0;$

$C_{12}=C_2 \oplus M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7=0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1=1;$

$C_{13}=C_3 \oplus M_2 \oplus M_3 \oplus M_4=0 \oplus 1 \oplus 0 \oplus 1=0;$

$C_{14}=C_4 \oplus M_5 \oplus M_6 \oplus M_7=0 \oplus 0 \oplus 1 \oplus 1=0,$

$E_{10}= E_0 \oplus C_1 \oplus C_2 \oplus M_1 \oplus C_3 \oplus M_2 \oplus M_3 \oplus M_4 \oplus C_4 \oplus M_5 \oplus M_6 \oplus M_7=$

=0⊕1⊕0⊕0⊕0⊕1⊕0⊕1⊕0⊕0⊕1⊕1=1.

Поскольку значение E_{10}=1 и значение, составленное из контрольных разрядов, равно 0010_2, то ошибка во втором бите. Приемник меняет значение этого бита на противоположное.

Рассмотрим случай, когда была двойная ошибка при передаче кода.

Пусть ошибка произошла во втором (т.е. в C_2) и в пятом битах (т.е. M_2) и принят код:

M_7	M_6	M_5	C_4	M_4	M_3	M_2	C_3	M_1	C_2	C_1	E_0
1	1	0	0	1	0	**0**	0	0	**0**	1	0

Теперь приемник проверяет правильность передачи кода, вычисляя следующие выражения:

$C_{11}=C_1⊕M_1⊕M_2⊕M_4⊕M_5⊕M_7$=1⊕0⊕0⊕1⊕0⊕1=1;

$C_{12}=C_2⊕M_1⊕M_3⊕M_4⊕M_6⊕M_7$=0⊕0⊕0⊕1⊕1⊕1=1;

$C_{13}=C_3⊕M_2⊕M_3⊕M_4$=0⊕0⊕0⊕1=1;

$C_{14}=C_4⊕M_5⊕M_6⊕M_7$=0⊕0⊕1⊕1=0,

$E_{10}= E_0⊕C_1⊕C_2⊕M_1⊕C_3⊕M_2⊕M_3⊕M_4⊕C_4⊕M_5⊕M_6⊕M_7$=

=0⊕1⊕0⊕0⊕0⊕0⊕0⊕1⊕0⊕0⊕1⊕1=0.

Поскольку значение E_{10}=0 и значение, составленное из контрольных разрядов, равно $0111_2≠0$, то произошла двойная ошибка. Приемник не передает код далее, а запрашивает от источника повторную передачу этого блока информации.

Практическая работа №69. 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 данных 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 №70. 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:

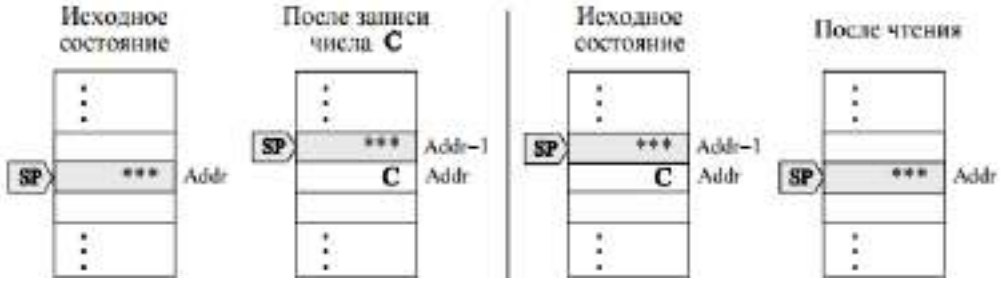
− вызывающей программе должно быть известно положение подпрограммы в общей структуре программы;

− 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 программным 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 про- изводится 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 под- программу 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 подпро- граммой.

Практическая работа №71. Элементарные приемы программирования (обработка однобайтных данных).

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 прерывания, 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 прерывания 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. запрещение 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 флагов прерываний производится записью единиц в соответствующие разряды ре- гистра 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 пре- рывание 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 №72. Элементарные приемы программирования (обработка многобайтных данных).

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 асин- хронные. 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 вы- водов 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 не- который РОН загружается число, которое затем на каждом проходе цикла уменьшается на единицу. Так продолжается до тех пор, пока содержимое реги- стра 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 за- держки 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, сиг- нализаторов, контактов реле, концевых выключателей и т. п. Такой опрос про- изводится путём чтения содержимого регистра PINx с определённой периодич- ностью или по сигналу прерывания при изменении состояния датчика. Если используемый источник двоичного сигнала имеет механические или электромеханические контакты, то при их замыкании возникает сложный пере- ходной 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 неудач- ный, 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 про- цесса. 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 сигна- лов внешних прерываний и др.), а также реализуют функции специализирован- ных 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 №73. Элементарные приемы программирования (особенности системы команд).

Цель работы: изучение функций системного и прикладного ПО в составе программных комплексов микропроцессорных систем; ознакомление с возможностями использования языков высокого уровня при разработке прикладного ПО микроконтроллеров и основными методами обеспечения надёжности ПО.

Основные сведения

Программное обеспечение встраиваемых микропроцессорных систем представляет собой сложный комплекс программ и данных, отличающихся по назначению, возможности модификации в процессе работы, частоте и кратности выполнения и т. п. Согласно наиболее общей классификации в структуре комплекса программ выделяют системное и прикладное ПО. Системное ПО служит для распределения ресурсов микропроцессорной системы и управления выполнением программ и заданий. Прикладное ПО предназначено для решения различных пользовательских задач (обработки информации, управления и др.).

Системное ПО. К системному программному обеспечению встраиваемых микропроцессорных систем относятся операционные системы (ОС), начальные загрузчики и другие программные средства. Операционная система (operating system, OS) – это совокупность про- грамм, обеспечивающих управление аппаратными ресурсами микропроцессор- ной системы, а также взаимодействие программных процессов с аппаратурой и другими процессами. Во встраиваемых микропроцессорных системах, в том числе построенных на базе RISC-микроконтроллеров, используются встраи- ваемые операционные системы. Встраиваемая операционная система выполня- ет следующие основные функции:

− управление памятью (memory management);
− управление процессами (tasks management);
− интерфейс с периферийными устройствами;
− поддержка различных коммуникационных протоколов.

Первые две функции обычно выполняет ядро (kernel) операционной системы. Интерфейс с периферийными устройствами и поддержку различных протоколов обеспечивают отдельные специализированные модули (аналоги драйверов), что повышает гибкость использования операционной системы. Одной из основных функций ОС является управление процессами. Это обусловлено тем, что микропроцессор, встроенный в систему управления некоторым объектом, должен обслуживать большое количество источников и потребителей информации. При этом,

как правило, возникает необходимость одновременного выполнения различных действий (процессов): обработки поступающей информации, формирования и выдачи управляющих воздействий, об- мена данными и др. Под процессом (process) понимают функционально и логически законченную последовательность команд микропроцессора. В литературе процесс иногда называют задачей (task). Так как процессорные ядра большинства современных микроконтроллеров представляют собой вычислители с одним потоком команд и одним потоком данных (SISD: Single Instruction – Single Data), в один момент времени обеспечивается выполнение только одной задачи. Совмещение во времени выполнения нескольких программных потоков достигается разделением процессорного времени между различными задачами. В этом случае говорят о квазипараллельном (лат. quasi – как будто, почти) выполнении нескольких программных потоков (задач), или многозадачности.

Термин «многозадачность» применительно к микроконтроллерам должен пониматься с некоторой долей условности, т. к. существенно ограниченные ресурсы (и в большинстве случаев отсутствие аппаратных средств поддержки многозадачного режима) не позволяют использовать в них развитые механизмы многозадачности. Общая идея обеспечения многозадачности в микроконтроллерах состоит в следующем. Каждому процессу выделяется своя область памяти для хранения данных и программного контекста (содержимого регистров процессора). Переключение между процессами производится согласно приоритетам с восстанов- лением соответствующего контекста процессора.

Наиболее совершенными средствами реализации многозадачности обладают операционные системы реального времени. Операционная система реального времени (ОС РВ – real-time OS, RTOS) – ОС, обеспечивающая взаимодействие микропроцессорной системы с внешними процессами в темпе, соизмеримом со скоростью протекания этих процессов, т. е. с гарантированным временем реакции (отклика). Типичная ОС реального времени выполняет задачи в соответствии с приоритетами, осуществляет динамическое распределение памяти, использует различные средства межзадачного взаимодействия (семафоры, сигналы, события, алармы, программные каналы, разделяемые модули данных, стеки сообщений и др.), обеспечивает быстрые процедуры обслуживания прерываний, гибкое взаимодействие между программными модулями и параллельную работу процессов, обслуживающих разные внешние устройства. В силу специфических условий применения основным требованием, предъявляемым к встраиваемым ОС, является компактность. Чаще всего ядро встраиваемой операционной системы имеет объём порядка нескольких килобайт, и ещё несколько килобайт занимают различные драйверы. В том случае, если по каким-либо причинам использование встраиваемой ОС в конкретной разработке невозможно (ограниченные ресурсы, отсутствие подходящей ОС и др.), функции ОС реализуются программистом при разработке прикладного ПО.

Начальный загрузчик (boot loader) – это специализированная программа, предназначенная для загрузки исполняемого кода программы (или операционной системы) в определённое место памяти программ при включении питания. Функции и характеристики начальных загрузчиков сильно различаются в зависимости от типа микропроцессора и структуры его памяти. Например, начальный загрузчик может выполнять функции программирования внутренней или внешней FLASH-памяти микроконтроллера с помощью интерфейсов RS232, SPI и др. или же (в том случае, если программа исполняется из ОЗУ) загружать код программы (операционной системы) из ПЗУ в оперативную память. 55В AVR-микроконтроллерах начальный загрузчик имеется в старших моделях и служит для загрузки памяти программ в режиме самопрограммирования.

Прикладное ПО. В настоящее время при разработке прикладного ПО для микроконтроллеров и других типов встраиваемых микропроцессоров всё шире применяются языки программирования высокого уровня (С, Паскаль, Бейсик и др.). Управляющие конструкции и структуры данных языков высокого уровня отражают понятия, естественные для некоторой предметной области (обработки информации, управления и т. п.), а не архитектуру микропроцессорной системы. По сравнению с языками ассемблера языки высокого уровня обеспечивают следующие основные преимущества:

− большую гибкость в описании алгоритмических конструкций и структур данных благодаря наличию различных операторов, типов данных и других развитых языковых средств;

− наглядность и удобство использования за счёт поддержки методов структурного программирования и существования обширных библиотек функций;

− переносимость программного обеспечения (программы, написанные на стандартизованных языках высокого уровня, могут быть перенесены на другие микропроцессоры без внесения изменений или с незначительными изменениями);

− ускорение разработки и упрощение сопровождения разработанного ПО.

Программа, написанная на языке высокого уровня, как правило, имеет больший объём и медленнее выполняется, чем аналогичная программа на языке ассемблера. Тем не менее, современные компиляторы с языков высокого уровня имеют эффективные средства оптимизации и позволяют получать программы, сравнимые по эффективности с программами на языке ассемблера. В том случае, если характеристики разработанной программы не удовлетворяют заданным требованиям (по быстродействию или размеру кода), используется следующий приём: выявляются критические с точки зрения производительности участки программы и записываются на языке ассемблера. Например, на языке ассемблера зачастую составляются процедуры обработки прерываний, которые, как следует из принципов организации режима работы по прерываниям, долж- ны выполняться максимально быстро.

Одним из наиболее популярных языков высокого уровня является язык С, характеризующийся следующими достоинствами:

− небольшим количеством элементов языка;

− высокой скоростью выполнения написанных на нём программ;

− поддержкой модульного программирования; − возможностью работы на аппаратном уровне.

Благодаря сочетанию управляющих структур, характерных для языков высокого уровня, с возможностью манипулирования битами, байтами и указателями (адресами) язык С занимает промежуточное положение между языками высокого уровня и языком ассемблера, в связи с чем язык С иногда называют языком среднего уровня. Язык С является стандартизованным, что значительно упрощает сопровождение и расширение написанного на нём программного обеспечения. Наибольшее распространение получила версия языка С, соответствующая стандарту, принятому в 1989 г. Американским национальным институтом стандартизации ANSI (American National Standards Institute). В 1999 г. был утверждён новый ANSI/ISO-стандарт языка С (ISO – International Standards Organization – Международная организация по стандартизации). Этот стандарт включает ряд усовершенствований и несколько новых средств, часть из кото- рых позаимствована из языка С++.

Архитектура AVR-микроконтроллеров специально оптимизирована для использования компиляторов с языков высокого уровня: AVR-микроконтроллеры имеют память программ большого объёма, содержат большое количество РОН, обладают высоким быстродействием. Одним из средств разра- ботки ПО для AVR-микроконтроллеров с использованием языков высокого уровня является оптимизирующий С-компилятор ICC AVR фирмы ImageCraft Creations. Пользовательский интерфейс компилятора ICC AVR выполнен в виде интегрированной среды разработки, включающей в себя текстовый редактор и менеджер проекта. Компилятор формирует несколько файлов, среди которых стандартный hex-файл для загрузки в память программ микроконтроллера и файл с листингом в текстовом формате. Компилятор совместим со стандартом ANSI. Исходный текст может содержать вставки на ассемблере (строки или ассемблерные модули). Библиотеки включают распределение памяти, строковые и математические функции, а также специальные функции, обеспечивающие доступ к энергонезависимой памяти данных и периферийным устройствам микроконтроллера.

В качестве иллюстрации возможности использования языка С в микроконтроллерных разработках на рисунке ниже приведён фрагмент программы на языке С. В программе использованы комментарии в стиле С++, поддерживаемые компилятором ICC AVR.

Надёжность ПО. Для сложных программных комплексов (особенно программных комплексов специализированных микропроцессорных систем, например, бортовых вычислителей) весьма актуальной является задача обеспечения высокой надёжности. Под надёжностью программного обеспечения понимается вероятность его работы без отказов в

течение определённого периода времени, рассчитанного с учётом стоимости каждого отказа. Отказ программного обеспечения – это проявление имеющейся в нём ошибки, заключающееся в том, что ПО не выполняет необходимых действий (операций). В зависимости от причины, порождающей ошибку, источники ошибок ПО можно разделить на внутренние и внешние. Внутренние – это ошибки проектирования, ошибки алгоритмизации, ошибки программирования, недостаточное качество средств защиты, ошибки в документации и т. п. Внешние ошибки связаны со сбоями и отказами аппаратуры, искажением информации в каналах связи, изменением конфигурации системы, ошибками пользователя.

```
// Подключение заголовочного файла
#include <iom8535.h>

// Векторы прерываний
#pragma interrupt_handler  EXT_INT0_interrupt:2     // прерывание INT0
#pragma interrupt_handler  EXT_INT1_interrupt:3     // прерывание INT1

// Подпрограмма задержки на ~10 мс при f = 8 МГц
void delay(void)
{
    unsigned int a;

    for (a = 0; a < 3000; a++) {
        }
}

// Подпрограмма инициализации
void init(void)
{
    // инициализация порта B для вывода данных
    DDRB = 0xff;

    // инициализация прерываний INT0 и INT1 (по положительному фронту)
    MCUCR = (1<<ISC01)|(1<<ISC00)|(1<<ISC10)|(1<<ISC11);

    // очистка флагов внешних прерываний
    GIFR = (1<<INTF1)|(1<<INTF0);

    // разрешение внешних прерываний INT0 и INT1

    GICR = (1<<INT0)|(1<<INT1);

    // глобальное разрешение прерываний
    SREG |= 0x80;
}

// Обработчик внешнего прерывания INT0
void EXT_INT0_interrupt(void)
{
    delay();                    // задержка
    if (PIND & 0x04 == 4)       // если нажата "+" кнопка,
        PORTB++;                // инкремент
}

// Обработчик внешнего прерывания INT1
void EXT_INT1_interrupt(void)
{
    // аналогично...
}

// Основная программа
void main(void)
{
    init();                     // инициализация
    while(1);                   // бесконечный цикл
}
```

Число ошибок в ПО, допущенных на стадии разработки, зависит от используемой технологии проектирования, организации работ и квалификации исполнителей и в принципе не является функцией времени. Однако интенсивность отказов аппаратуры из-за ошибок в ПО является функцией времени и может с течением времени как снижаться, так и повышаться. При этом отказы аппаратных средств являются функцией времени и обусловлены физическим старением, износом или разрушением отдельных элементов в процессе эксплуатации, хранения, транспортировки.

Выделяют следующие методы проектирования надёжного ПО:

− методы предупреждения ошибок, позволяющие минимизировать или исключить появление ошибок;

− методы обнаружения ошибок, направленные на выявление ошибок в программном обеспечении;

− методы обеспечения устойчивости к ошибкам, позволяющие системе функционировать при наличии ошибок и устранять их последствия.

Методы предупреждения ошибок включают в себя комплекс мер, направленных на уменьшение количества ошибок, допускаемых в процессе проектирования программного обеспечения. Основными методами предупреждения ошибок является уменьшение сложности

ПО, совершенствование способов обмена информацией, обнаружение и устранение ошибок на каждом этапе разработки ПО (до этапа тестирования).

Сложность является одной из главных причин низкой надежности программного обеспечения. В общем случае сложность объекта является функцией количества связей между его компонентами. Для уменьшения сложности ПО используются два основных приёма – иерархическое построение ПО и минимизация связей («сцепления») между модулями. Иерархическое построение ПО состоит в разделении программного комплекса по уровням понимания (абстракции), что позволяет анализировать систему, скрывая несущественные для данного уровня детали реализации других уровней. Минимизация связей между модулями заключается в усилении независимости отдельных программных модулей. Использование такого приёма должно предотвращать ситуацию, когда изменение одного модуля влечёт за собой необходимость внесения изменений в другие программные модули, связанные с ним. Методы обнаружения ошибок базируются на введении в программное обеспечение различных видов избыточности – временно́й, информационной и программной. Временна́я избыточность заключается в использовании части производительности процессора для контроля исполнения и восстановления работоспособности ПО после сбоя. Информационная избыточность состоит в дублировании части данных информационной системы для обеспечения надёжности и контроля достоверности данных. Программная избыточность подразумевает, что компоненты программно- го комплекса проектируются исходя из предположения, что другие компоненты и исходные данные содержат ошибки, и должны их обнаруживать. Программная избыточность обеспечивает нахождение и регистрацию ошибок, выполнение одинаковых функций разными модулями системы и сопоставление результатов обработки; контроль и восстановление данных с использованием других видов избыточности.

Методы обеспечения устойчивости к ошибкам направлены на минимизацию ущерба, вызванного наличием ошибок, и реализуются путём обработки сбоев аппаратуры, повторного выполнения операций, динамического изменения конфигурации, выполнения специальных операций в случае отказа отдельных функций системы, копирования и восстановления данных и ряда других действий. Основным способом обнаружения ошибок в программном обеспечении является тестирование. Из-за бесконечного числа возможных сочетаний исход- ных данных исчерпывающее тестирование всех ветвей алгоритма любой слож- ной программы во всех режимах практически неосуществимо. Поэтому считается, что сложные программные комплексы неизбежно содержит ошибки, которые выявляются в течение длительного периода при эксплуатации (иногда на протяжении всего периода эксплуатации). Выделяют следующие этапы тестирования программных комплексов:

− автономное тестирование – контроль отдельного программного модуля отдельно от других модулей программного комплекса;

− тестирование сопряжений – контроль связей между частями программного комплекса (модулями, компонентами, подсистемами);

− тестирование функций – контроль выполнения программным комплексом необходимых функций;

− комплексное тестирование – проверка соответствия программного комплекса заданным требованиям;

− тестирование конфигураций – проверка каждого конкретного варианта поставки (установки) программного комплекса.

Ошибки в программах и данных могут проявиться на любой стадии тестирования, а также в период эксплуатации системы. Зарегистрированные и обработанные сведения должны использоваться для выявления отклонений от требований заказчика или технического задания. Хотя при наличии неисправленной ошибки в ПО, приводящей к отказу при определённом сочетании входных данных, этот отказ проявится в случае последующего выполнения программы при том же сочетании входных данных, тем не менее можно считать, что отказы ПО носят вероятностный характер, так как, во-первых, ошибки в ПО распределены случайным образом, а во-вторых, случайными являются и конкретные наборы входных данных. Поэтому

интервал времени безотказной работы программного обеспечения также является случайной величиной.

Практическая часть.

Дополнить программу недостающими операторами в подпрограмме обработки внешнего прерывания INT1. Откомпилировать программу с помощью компилятора ICC AVR, проделав следующие операции. Запуск программы ICC AVR производится из группы ImageCraft Development Tools меню Пуск → Программы.

1. Создать новый проект с помощью команды New меню Project. В появившемся диалоговом окне Save New Project As… ввести имя проекта. После нажатия кнопки «ОК» в правой части окна программы ICC AVR в области представления иерархии проекта появится схематичное изображение иерархии соз- данного проекта.

2. Создать файл с исходным текстом программы. Для этого воспользоваться командой New меню File, после чего на экране появится окно редактора исходных текстов программ. Ввести и отредактировать текст программы; сохранить файл (с расширением .c), воспользовавшись командой Save As меню File.

3. Включить созданный файл с исходным текстом программы в проект командой Add File(s) меню Project или контекстного меню области представления иерархии проекта.

4. Провести компиляцию созданной программы, воспользовавшись командой Make project меню Project. По сообщениям, выводимым в процессе компиляции в нижней части окна программы ICC AVR (в области сообщений) убедиться в отсутствии ошибок. По окончании компиляции в рабочем каталоге проекта будет создано несколько новых файлов, среди которых файл листинга (с расширением .lst) с и файл-прошивка памяти программ (с расширением .hex).

5.Проанализировать листинг трансляции. По приведённым в листинге трансляции данным определить объём, занятый программой в памяти, и сравнить это значение с одноимённым параметром для аналогичной ассемблерной программы.

6. Занести эти сведения в отчёт. Отчёт должен содержать: титульный лист с указанием номера и названия работы, номера группы и фамилий выполнивших работу; цель работы; исходный текст и листинг трансляции созданной программы, а также сведения, указанные в задании.

Контрольные вопросы.

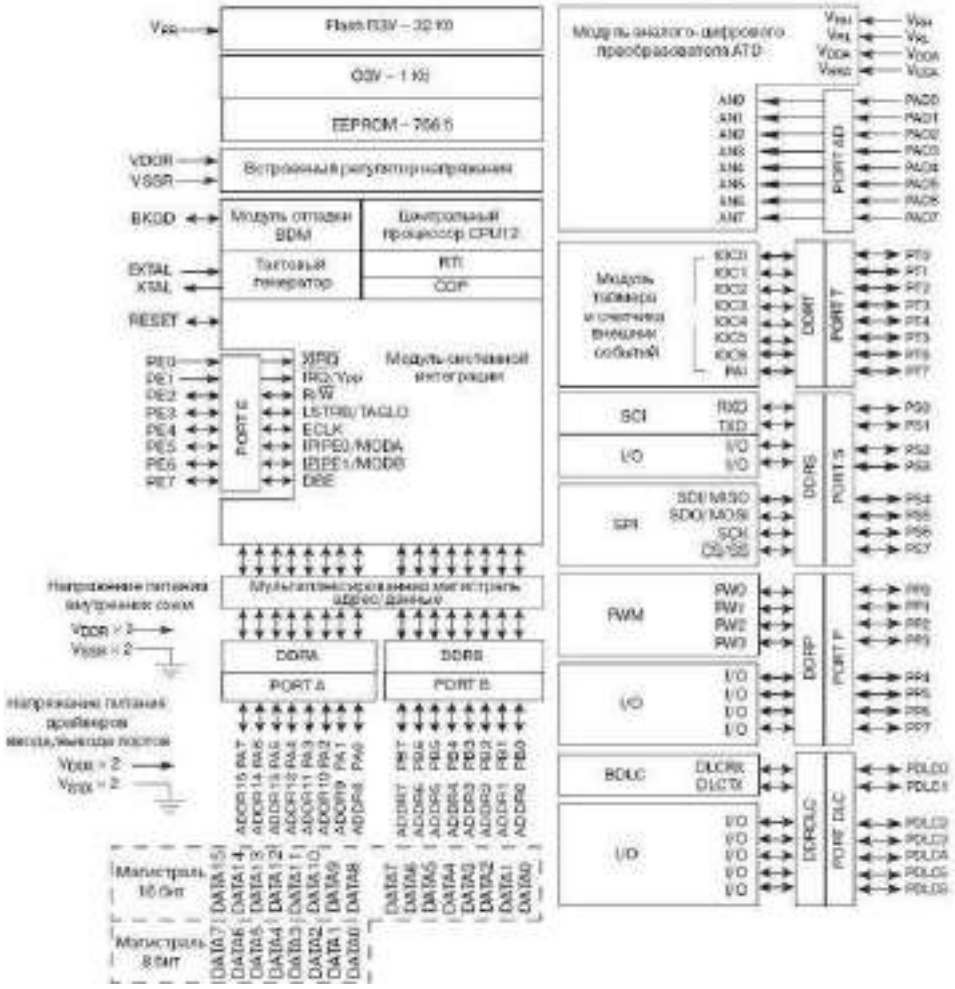
1. Назначение системного программного обеспечения.
2. Основные функции встраиваемых операционных систем.
3. Особенности операционных систем реального времени.
4. Преимущества и недостатки языков высокого уровня по сравнению с языком ассемблера.
5. Достоинства языка программирования С.
6. Характеристики надёжности программного обеспечения.
7. Способы повышения надёжности программного обеспечения.

Практическая работа №74. 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 В32 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 = 0x0f; /*установить 2 такта для времени выборки*/
/*и коэффициент деления, равный 4*/
printf( "ready\n" );
ADC_convert(); /*реализовать цифровой вольтметр*/
}
/*_____*/
```

Контрольные вопросы.

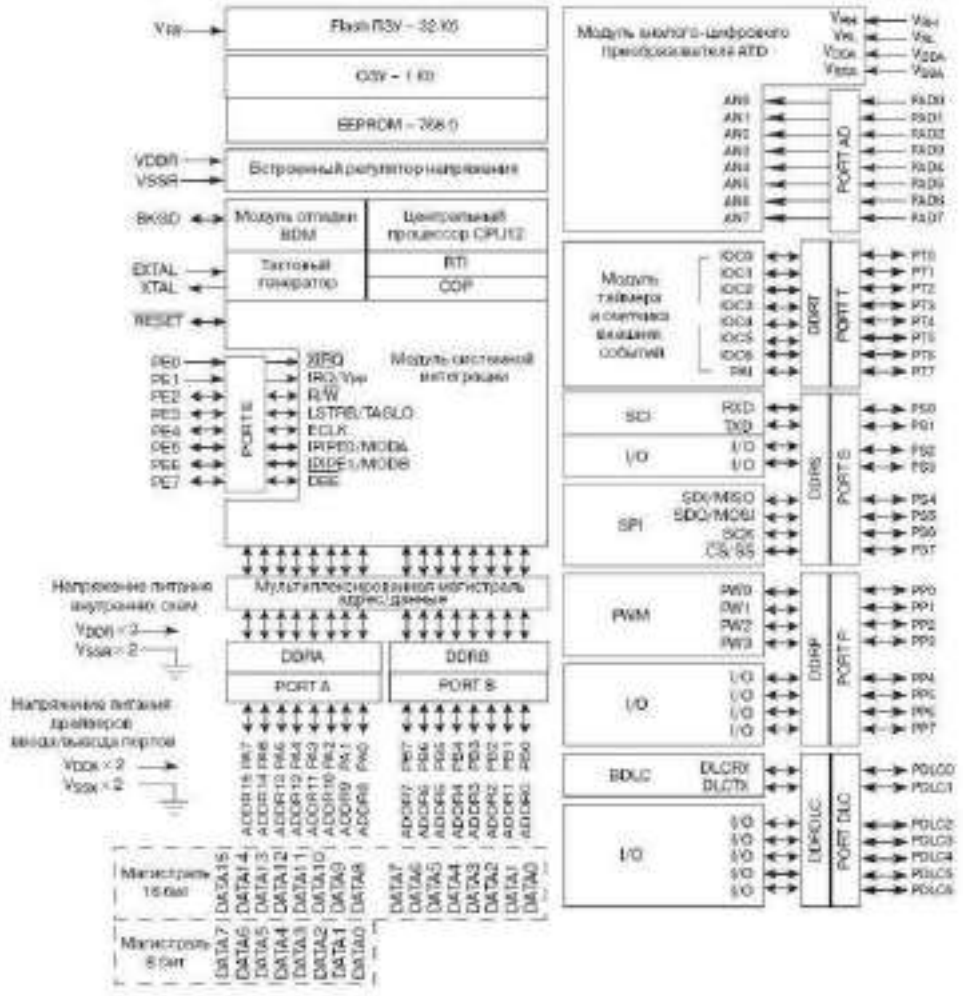
1) Назначение процессорного ядра микроконтроллеров.
2) Основные аппаратные средства современных МК.
3) Дополнительные аппаратные средства современных МК.
4) Особенности микроконтроллеров на примере МК семейства Motorola.
5) Многофункциональный таймер.
6) Модуль контроллера CAN интерфейса msCAN12 (Motorola 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 №75. Программные модели аппаратных средств микропроцессорных систем (способы построения программ).

Цель работы: ознакомление с внутренней структурой современных микроконтроллеров. Изучение основных функциональных блоков на примере МК 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 рисунке ниже.



Отличия современных микроконтроллеров на примере МК Motorola.

Низкое энергопотребление. Микроконтроллеры семейства 68HC12 производятся на основе CMOS технологии (CMOS – Complementary Metaloxide semiconductor). Эта технология позволяет создать транзисторные структуры с относительно низкими потерями энергии при работе в ключевых режимах. Поэтому МК семейства 68HC12 характеризуются малым потреблением энергии, что позволяет рекомендовать их для использования в изделиях с автономным питанием (от аккумуляторов или батареек). Однако не следует забывать, что потребляемая энергия для полупроводниковых CMOS структур прямопропорциональна частоте переключения. Поэтому для достижения оптимальных энергетических характеристик следует выбирать частоту тактирования центрального процессора микроконтроллеров 68HC12 минимально возможной для конкретного применения. Микроконтроллеры 68HC12 имеют специальные режимы пониженного энергопотребления, которые также позволяют оптимизировать энергетические характеристики проектируемого изделия.

Высокопроизводительное 16 разрядное процессорное ядро. Центральный процессор семейства 68HC12 выполняет действия над 16 разрядными операндами, поэтому время выполнения алгоритмов над переменными, разрядность которых превышает байт, существенно сокращается по сравнению с 8 разрядными МК. Максимальная частота тактирования процессорного ядра fBUS для микроконтроллеров семейства 68HC12 составляет 8 МГц. Однако частота внешнего генератора или кварцевого резонатора должна составлять 16 МГц, поскольку внутренние цепи микроконтроллера делят эту частоту на два для получения fBUS. Новое семейство МК HCS12 унаследовало архитектуру процессорного ядра и большинства периферийных модулей от своего предшественника, семейства 68HC12.

Отметим основные отличия:

1) Напряжение питания большинства моделей МК семейства HCS12 равно 5,0 В, что позволяет обеспечить электромагнитную совместимость в авто мобильных и общепромышленных применениях;

2) Повышенна производительность процессорного ядра. Частота тактирования центрального процессора и межмодульных магистралей МК семейства HCS12 составляет 25 МГц против 8 МГц у HC12;

3) Увеличен объем резидентной памяти. Объем встроенного в МК семейства HCS12 ОЗУ достигает 12 Кб, объем Flash ПЗУ – 512 Кб. Кроме того, в составе большинства моделей МК имеется значительная область EEPROM (до 4 Кб) для хранения перепрограммируемых констант пользователя;

4) Большое число интегрированных на кристалл разнообразных контроллеров последовательных интерфейсов, т.к. МК семейства предназначены для работы в качестве интеллектуальных узлов распределенных систем управления.

Функциональные блоки микроконтроллеров Семейство HCS12 объединяет ряд моделей МК с одинаковым процессорным ядром CPU HCS12. Отдельные представители семейства различаются объемом встроенной памяти и количеством и типом интегрированных на кристалл МК периферийных модулей. Однако каждый МК из семейства HCS12 имеет в своем составе следующие функциональные модули:

1) Память трех типов: FLASH память программ, энергонезависимая память EEPROM для хранения изменяемых констант пользователя и статическое ОЗУ для размещения промежуточных переменных прикладной программы управления;

2) Многофункциональный 16 разрядный таймер с 8 каланами IC/OC;

3) Многоканальный аналого-цифровой преобразователь; 12

4) Контроллеры последовательного обмена нескольких стандартов;

5) Модуль ШИМ общего назначения, ряд моделей оснащен специализированным модулем ШИМ для управления автономными вентильными преобразователями.

Режимы работы микроконтроллеров Микроконтроллеры семейства 68HC12/HCS12 функционируют в одном из восьми режимов, которые делят на две группы: рабочие режимы и специальные режимы. Рабочие режимы позволяют создать различную аппаратную реализацию встраиваемого контроллера, в то время как специальные режимы работы предназначены для проведения тестовых испытаний и диагностики МК в процессе производства. Поэтому инженерам по применению микроконтроллеров важно изучить лишь группу рабочих режимов.Каждый режим из группы рабочих задает собственное распределение адресного пространства МК и конфигурацию магистралей для подключения внешней памяти. Режим работы МК назначается посредством комбинации логических сигналов на входах BKGD, MODB, MODA микроконтроллера в состоянии начального запуска МК. Состояние начального запуска именуют также состоянием сброса (Reset). Сразу после выхода из состояния сброса МК запоминает кодовую комбинацию на перечисленных входах и переходит в соответствующий режим работы. Там же указаны альтернативные функции линий портов PORT A и PORT B, которые они приобретают в каждом из режимов работы. В большинстве проектируемых устройств Вы будете использовать МК 68HC12/HCS12 в одном из трех рабочих режимов:

• Однокристальном или автономном режиме;

• Расширенном режиме с 16 разрядной внешней шиной;

• Расширенном режиме с 8 разрядной внешней шиной.

Расширенные режимы работы предоставляют возможность подключения к МК внешней памяти и внешних периферийных ИС с использованием параллельных магистралей адреса и данных. Сигналы магистралей формируются на линиях портов PORTA и PORTB, поэтому использование соответствующих выводов МК в качестве линий ввода/вывода общего назначения в расширенных режимах работы становится невозможным. Краткое описание рабочих режимов Однокристальный режим работы (BKGD: 1, MODB: 0, MODA: 0) обеспечивает функционирование МК с использованием только внутренней памяти. Поэтому коды прикладной программы управления и ее переменные должны размещаться только во внутреннем ПЗУ и ОЗУ МК. Порты PORTA и PORTB используются в качестве обычных двунаправленных портов

ввода/вывода. Подключение внешних периферийных ИС должно производиться с использованием последовательных интерфейсов или с программной поддержкой временной диаграммы обмена на линиях портов ввода/вывода. Расширенный режим с 16 разрядной системной шиной (BKGD: 1, MODB: 1, MODA: 1) обеспечивает функционирование МК с использованием как внутренней, так и внешней памяти. Для подключения внешней памяти предназначена 16_разрядная мультиплексированная магистраль адрес/данные ADDR15_0/DATA15_0. При этом старший байт мультиплексированной во времени магистрали ADDR15_8/DATA15_8 формируется на линиях PORTA, младший байт ADDR7_0/DATA7_0 – на линиях PORTB. Расширенный режим с 8_разрядной системной шиной (BKGD: 1, MODB: 0, MODA:1) также реализует работу МК с использованием внутренней и внешней памяти. Но для13 подключения внешней памяти предназначены 16 разрядная магистраль адреса ADDR15_0 и 8 разрядная магистраль данных DATA7_0. Старший байт магистрали адреса ADDR15_8 выводится на PORTA, младший байт ADDR7_0 – на PORTB. Двунаправленная 8 разрядная магистраль данных DATA7_0 использует линии порта PORTA в мультиплексированном со старшими разрядами магистрали адреса режиме. В обоих расширенных режимах некоторые линии порта PORTE используются для передачи сигналов управления обменом по шине.

Практическая часть.

Перед началом выполнения практической части необходимо ознакомиться с лабораторной установкой на базе платы Freescale и программным обеспечением CodeWarrior. Рассмотрим последовательность действий, которую необходимо проделать для инициализации модуля PWM:

1) Для конкретного приложения следует определить разрешающую способность генерируемого ШИМ сигнала, т.е. число дискретных отсчетов частоты тактирования канала в периоде и длительности импульса выходного сигнала канала. На основании полученных данных следует определить, в каком режиме, 8_разрядном или 16_разрядном, Вы будете использовать каналы модуля PWM;

2) Для конкретного приложения следует определить требуемую частоту генерируемого ШИМ сигнала. На основе полученных данных определить структуру подсистемы тактирования каналов модуля PWM;

3)Установить биты CON23 и CON01 в регистре PWCLK для выбора 8 раз рядного или 16 разрядного режима работы;

4) Определить, будете ли Вы использовать режим центрированной или режим фронтовой ШИМ. В соответствие с выбранным режимом установить бит CENTR в регистре PWCTL;

5) Определить активный уровень ШИМ сигнала, в соответствии с выбором установить биты PPOL0…PPOL3 в регистре PWPOL;

6) Назначить источники тактирования для каналов, для чего установить биты PCLK0…PCLK3 в регистре PWPOL;

7) Установить коэффициенты деления для импульсных последовательностей CLOCK A и CLOCK_B, используя для этого биты PCKA2… PCKA0 и PCKB2… PCKB0 регистра PWCLK;

8)Установить для используемых каналов значения регистров периода и регистров коэффициента заполнения;

9) Разрешить работу выбранных каналов модуля PWM, используя для этого биты PWEN0…PWEN3 регистра EPWM. Определим параметры настройки модуля PWM для генерации ШИМ сигнала с частотой 976 Гц и коэффициентом заполнения 66,7%. Частоту тактирования канала ШИМ выберем равной 8МГц/32 = 250 кГц. Этой частотой будем тактировать 8_разрядный счетчик канала. Для формирования частоты 976 Гц потребуется 256 отсчетов частоты 250 кГц, что соответствует максимально возможному коэффициенту счета 8_разрядного счетчика периода канала. Для формирования сигнала с коэффициентом заполнения 66,7% следует установить код периода, равный 256 отсчетам, а код коэффициента заполнения – 171 отсчету. Для формирования ШИМ сигнала будем использовать канал 0 модуля PWM. Программный фрагмент init_pwm.c производит начальную установку регистров специальных функций модуля PWM для генерации на выходе PP0 ШИМ сигнала с частотой 976 Гц и коэффициентом заполнения 66,7%:

```
/*_____*/
/* Функция init_pwm задает начальные установки модуля PWM */
/*_____*/
void ini_pwm(void)
{
PWTST = 0x00; /*выбрать нормальный режим работы модуля PWM*/
PWCTL = 0x00; /*выбрать режим фронтовой ШИМ*/
PWCLK = 0x28 /*канал 0 в 8_разрядном режиме, коэфф. деления*/
/* частоты E_CLOCK равен 32*/
PWPOL = 0x01; /*установить высокий активный уровень сигнала*/
DDRP = 0xFF; /*настроить порт P на вывод*/
PWEN = 0x01; /*разрешить работу канала 0 модуля PWM */
PWPER0 = 255; /*установить код периода*/
PWDTY0 = 171 /*установить код коэффициента заполнения*/
}
/*_____*/
```

Контрольные вопросы

1) Внутренняя структура микроконтроллеров.
2) Отличия современных МК.
3) Функциональные блоки МК.
4) Режимы работы МК.
5) Однокристальный режим работы.
6) Расширенный режим с 16 разрядной системной шиной.
7) Расширенный режим с 8 разрядной системной шиной.
8) Назначение модуля широтно-импульсного модулятора PWM (Pulse Width Modulation).

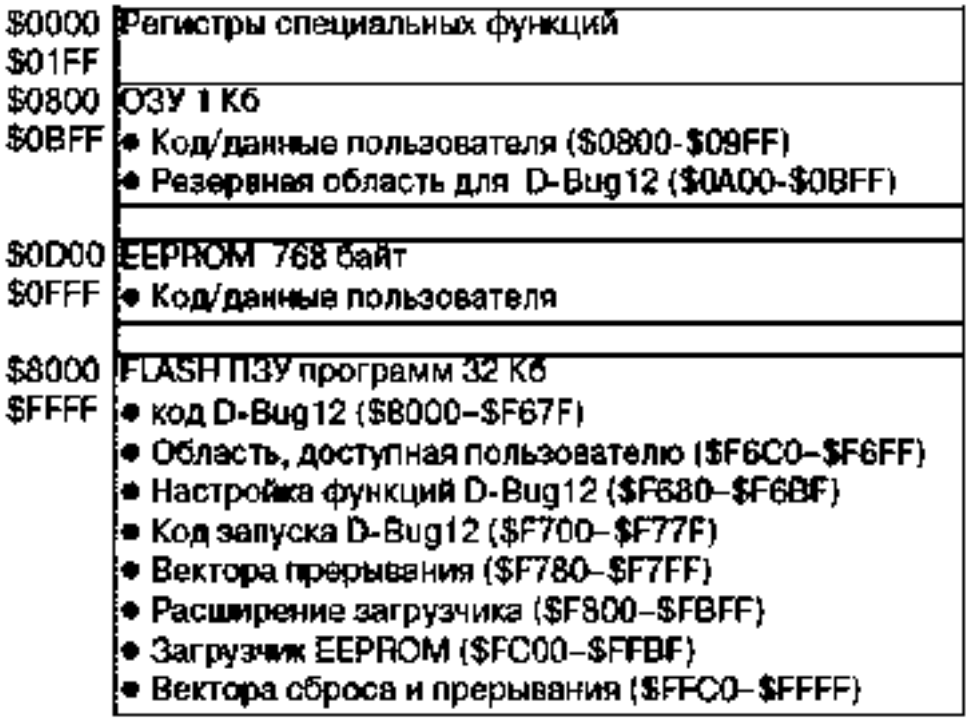
Практическая работа №76. Программные модели аппаратных средств микропроцессорных систем (моделирование функций).

Цель работы: ознакомление с видами памяти микроконтроллеров: память данных, память программ. Изучение регистров и стековой памяти МК.

Общие сведения

Подсистема памяти МК семейства 68HC12/HCS12 включает четыре различных модуля памяти: энергонезависимая Flash память программ, энергонезависимая EEPROM память данных, статическое ОЗУ и блок регистров специальных функций для управления режимами работы периферийных модулей. Расположение различных модулей памяти в адресном пространстве МК принято отражать на так называемой карте памяти. Карта памяти МК B32 представлена на рис. ниже. Указанные на ней адреса будут действительны при выходе МК из состояния сброса. В ходе исполнения прикладной программы адресное пространство для каждого модуля памяти может быть изменено. Тогда для обращения к ячейкам памяти будут использоваться не указанные на рис. физические адреса, а измененные виртуальные адреса. Память программ и память данных микроконтроллера на примере МК семейства Motorola Микроконтроллер B32 предназначен для использования преимущественно в однокристальном режиме работы. Он содержит в себе 32Кб ПЗУ программ, 768 байт памяти типа EEPROM, 1Кб статического ОЗУ и 512 регистров управления. Строго говоря, аббревиатура EEPROM (Electrically Erasable Programmable ROM) обозначает энергонезависимую память с электрическим программированием и электрическим стиранием. Поэтому и резидентная память программ, выполненная на основе технологии FLASH, и энергонезависимая память данных должны быть характеризованы как EEPROM. Однако энергонезависимая память программ и энергонезависимая память данных отличаются по своим свойствам не только на уровне технологии изготовления, но и на уровне разработчика встраиваемых систем. Память типа Flash допускает выполнение операции стирания только над некоторым множеством ячеек, что неудобно при необходимости замены только одного байта информации. Энергонезависимая память данных позволяет стереть и потом запрограммировать один байт информации. Однако ячейки памяти с подобным свойством занимают значительную площадь полупроводникового

кристалла МК, поэтому на их основе не может быть выполнена память программ большого объема. Во избежание путаницы у российских разработчиков принято использовать аббревиатуру EEPROM только для памяти с побайтным стиранием и побайтным программированием. А память со стиранием блоками обозначают как Flash, хотя и эта память по своим свойствам относится к EEPROM. Используя МК от Freescale Semiconductor, в частности семейство 68HC12/HCS12 следует знать, что гарантированное число циклов стирания/программирования для МК HCS12 составляет 10000, а для МК 68HC12 – всего 100. Именно поэтому во многих отладочных средствах на основе МК 68HC12 рекомендуется промежуточные версии программы записывать и исполнять из ОЗУ. Поскольку резидентное ОЗУ у микроконтроллеров обладает недостаточным объемом для размещения программы, то многие отладочные платформы используют расширенный режим работы МК с подключением внешнего ОЗУ. В МК семейства HCS12 для целей отладки обычно используется внутреннее Flash ПЗУ программ, т.к. 10000 циклов перезаписи обычно остаточно для внесения всех исправлений в процессе отладки. Пример применения16 В процессе эксплуатации память типа EEPROM часто используют для создания счетчиков аварийных ситуаций на объекте. Возможные аварии предварительно классифицируются, в системе устанавливаются датчики, которые позволяют микроконтроллеру отнести возникшую аварийную ситуацию к тому или иному типу. Если тип аварии диагностирован, то МК увеличивает соответствующий счетчик и запоминает его новое состояние в энергонезависимой памяти данных типа EEPROM. Такое решение позволяет сохранить информацию об авариях даже при отключении системы питания.



Карта памяти микроконтроллера

Регистры специальных функций Блок регистров специальных функций микроконтроллера по реализуемым функциям очень похож на пульт управления. Каждый регистр осуществляет управление или отображает состояние того периферийного модуля МК, к которому он отнесен в техническом описании. Часть битов регистров специальных функций может быть сопоставлена с переключателями: установка бита в 1 разрешает реализацию, какой либо функции периферийного модуля, сброс в 0 – прекращает исполнение этой функции. Другая часть битов регистров аналогична индикаторам состояния: установка 1 свидетельствует о завершении выполнения назначенной функции, пребывание в 0 свидетельствует о том, что процесс еще не завершился. Регистры специальных функций МК семейства 68HC12/HCS12 объединены в блок, который размещается в адресном пространстве памяти МК. Доступ к каждому из 512 регистров блока осуществляется теми же командами центрального процессора, что и к ячейкам памяти. Никакого различия с точки зрения программного обслуживания между регистрами специальных

функций и ячейками ОЗУ не существует. Исключение составляют лишь регистры конфигурации и другие, так называемые защищенные регистры, состояние которых невозможно изменить в произвольный момент исполнения прикладной программы. В момент начального запуска, когда МК находится в состоянии сброса, в каждый регистр специальных функций записывается определенное в техническом описании значение. Начальные состояния регистров специальных функций обычно таковы, что функции соответствующего периферийного модуля отключены. Это очень удобно как с точки зрения потребления энергии микроконтроллером, так и с точки зрения программиста, составляющего прикладную программу управления. Если при запуске большинство периферийных модулей не работает, то МК потребляет минимальную энергию. Далее будут активизированы только те модули, которые потребуются для выполнения управляющего алгоритма. Остальные модули останутся не задействованными, что обеспечит минимизацию энергии потребления в процессе функционировании конечного устройства. При составлении программы управления МК сначала предстает перед разработчиком в минимальной конфигурации, когда большинство периферийных модулей как бы не существует. Таким МК легко управлять, поэтому ошибок в прикладной программе будет меньше. По мере реализации программы управления разработчик активизирует работу только тех модулей, которые ему потребуются. Поэтому ему не придется думать о программном обслуживании не задействованных при реализации алгоритма периферийных модулей. Каждый регистр специальных функций имеет свой собственный адрес в карте памяти МК. При программировании на ассемблере изменение значений регистров специальных функций и считывание их состояния может быть произведено командами загрузки и пересылки данных, такими как LDAA, STAA, MOV, а также командами битовых операций BSET, BCLR. Для того чтобы избавить программиста от запоминания физических адресов регистров специальных функций, следует использовать псевдокоманду операции присваивания EQU языка Ассемблер. Эта псевдокоманда назначит символьному имени регистра, которое одинаково для всех моделей МК семейства 68HC12/HCS12, абсолютный адрес. И при записи исходного текста программы можно будет использовать только символьные имена регистров специальных функций. Использование символьных имен также полезно при переносе прикладной программы с одной модели МК на другую. В этом случае следует заменить только абсолютные адреса в псевдокомандах, в то время, как исходный текст программы останется неизменным.

Практическая часть.

Перед началом выполнения практической части необходимо ознакомиться с лабораторной установкой на базе платы Freescale и программным обеспечением CodeWarrior. В практической части данной работы рассматривается программирование регистров модуля АЦП. Регистр ATDCTL2, расположенный в памяти МК по адресу 0x0062, управляет работой модуля аналого_цифрового преобразователя ATD. Бит 7 этого регистра разрешает (1) или запрещает (0) работу модуля. Этот бит, обозначенный в техническом описании как ATPU (ATD Powerup Bit), устанавливается в 0 в состоянии сброса МК. Для включения модуля ATD в работу необходимо программно установить этот бит в 1. На языке Ассемблера это действие может быть произведено следующим фрагментом программы:

```
;_____________________________________________________
;MAIN PROGRAMM
;_____________________________________________________
ATDCTL2 EQU $0062 ;определить адрес регистра в МК
ATD_INI EQU $80 ;определить значение слова
;инициализации для включения АЦП
LDAA #ATD_INI ;загрузить слово инициализации в ACC
STAA ATDCTL2 ;записать значение инициализации в
;регистр управления АЦП
;_____________________________________________________
```

Первые две строки программы содержат директивы присвоения EQU языка Ассемблер, которые информируют программу о том, что вместо имени ATDCTL2 в кодах программы следует использовать значение $0062, а вместо ATD_INI – $80 или 10000000 в двоичном коде. Команда LDAA использует непосредственную адресацию, в результате ее выполнения число $80 загрузится в аккумулятор ACC. Команда STAA с прямой адресацией пересылает содержимое ACC в ячейку с адресом ATDCTL2. В результате, код в регистре управления ATDCTL2 будет

равен 10000000, т.е. установленный в 1 бит 7 вызовет включение модуля АЦП. На языке Си аналогичное действие выполняет следующий программный фрагмент:

```
/*_____________________________________________________*/
/*MAIN PROGRAMM
/*_____________________________________________________*/
#include<912b32.h>
void main(void)
{
unsigned char ATD_INI = 0x80;
ATDCTL2 = ATD_INI;
}
/*_____________________________________________________*/
```

Сравнить преимущества написания программ на С и на Ассемблер.

Контрольные вопросы

1) Структура памяти микроконтроллеров.
2) Память программ.
3) Память данных.
4) Энергозависимая память.
5) Энергонезависимая память.
6) Регистры специальных функций МК.
7) Регистры управления модулем АЦП.
8) Стековая память.
9) Преимущества программирования на языке Си и Ассемблер.

Практическая работа №77. Программные модели аппаратных средств микропроцессорных систем (алгебра логики).

Цель работы: ознакомление с устройством ввода/вывода микроконтроллеров. Изучение спецификации портов на примере МК Motorola.

Общие сведения

Все МК семейства 68HC12 имеют некоторое количество линий ввода/вывода данных. Линии объединены в 8 разрядные параллельные порты данных: Port A, Port B, Port E, PORT AD и т.д. За редким исключением, все линии ввода/вывода двунаправленные. Направление передачи линий ввода/вывода настраивается программно путем записи управляющего слова в регистр направления передачи соответствующего порта. Возможно изменение направления передачи в ходе выполнения программы посредством перепрограммирования этих регистров. Сигнал сброса устанавливает все двунаправленные линии в режим ввода.

Следует особо подчеркнуть, что направление передачи каждой линии может быть выбрано разработчиком произвольно, независимо от ругих линий, принадлежащих к одному и тому же порту ввода/вывода. Исключение составляют лишь линии однонаправленной передачи, которые изначально специализированы на ввод или на вывод. Часть линий ввода/вывода имеют так называемую альтернативную функцию, т.е. обеспечивают связь встроенных периферийных модулей МК с «внешним миром». Так линии порта PORT AD используются для подключения к встроенному АЦП измеряемых напряжений, линии порта PORT S служат входами и выходами контроллеров последовательного обмена. Если соответствующий периферийный модуль МК не используется, то его выводы можно задействовать как обычные линии ввода/вывода. Если линии порта двунаправленные, то для его обслуживания такого порта предусмотрены два типа регистров: PORTx – регистр данных порта x, где x – имя порта ввода/вывода; DDRx – регистр направления передачи порта x. Например, порт PORT A обслуживается регистрами PORTA и DDRA, а порт PORT B – регистрами PORTB и DDRB.

Если порт имеет схемотехнику с программно подключаемым «подтягивающим» резистором (R pull_up), то для обслуживания такого порта предусмотрен дополнительный регистр входного сопротивления порта. Ниже приведен фрагмент текста программы, которая

конфигурирует PORT T для вывода данных, а затем записывает в порт число S62. Для того, чтобы все линии порта PORT T стали линиями вывода, необходимо записать в регистр направления передачи DDRT код $FF. Спецификация портов ввода/вывода Подсистема параллельного ввода/вывода МК B32 состоит из 8 портов, причем линии многих портов обладают альтернативной функцией. Мы рассмотрим эти порты в порядке их расположения на рисунке по часовой стрелке.

1)PORT A. В однокристальном режиме работы – 8 разрядный порт ввода/вывода общего назначения. Направление передачи каждой линии порта определяется соответствующим битом регистра DDRA. В расширенном режиме работы на линиях порта формируются сигналы старшего байта мультиплексированной магистрали адрес/данные ADDR15_8/DATA15_8. В расширенном режиме с 8 разрядной шиной линии порта представляют собой мультиплексированную магистраль ADDR15_8/DATA7_0. 20

2) PORT B. В однокристальном режиме работы – 8 разрядный порт ввода/вывода общего назначения. Направление передачи каждой линии порта определяется соответствующим битом регистра DDRB. В расширенном режиме работы на линиях порта формируются сигналы младшего байта мультиплексированной магистрали адрес/данные ADDR7_0/DATA7_0. В расширенном режиме с 8 разрядной шиной линии порта представляют собой немультиплексированную магистраль ADDR7_0.

3) PORT E. 8_разрядный порт ввода/вывода общего назначения. Две линии порта PE0 и PE1 – однонаправленные и работают только на ввод. Остальные линии порта – двунаправленные, направление передачи линий PE2 и PE7 определяется соответствующими битами регистра DDRE. Все линии порта имеют альтернативную функцию. Линии PE1 и PE0 могут использоваться как входы внешних прерываний IRQ и XIRQ. Остальные линии служат для задания режимов работы МК (MOD A и MOD B) и в качестве сигналов управления внешней шиной при работе МК в расширенно режиме.

4) PORT AD. Однонаправленный 8_разрядный порт ввода. Все линии имеют альтернативную функцию. Если работа модуля аналого-цифрового преобразователя ATD разрешена, то линии порта используются для подключения измеряемых аналоговых сигналов.

5) PORT T. Двунаправленный 8 разрядный порт ввода/вывода общего назначения. Направление передачи каждой линии порта определяется соответствующим битом регистра DDRT. Альтернативная функция линий порта PORT T – обслуживание модуля таймера. Если работа таймера разрешена, то линии используются в качестве входов входного захвата IC или выходов выходного сравнения OC.

6) PORT S. Двунаправленный 8 разрядный порт ввода/вывода общего назначения. Направление передачи каждой линии порта определяется соответствующим битом регистра DDRS. Альтернативная функция линий порта PORT S – обслуживание модулей последовательного обмена SCI и SPI.

7) PORT P. Двунаправленный 8 разрядный порт ввода/вывода общего назначения. Направление передачи каждой линии порта определяется соответствующим битом регистра DDRP. Четыре линии порта PORT P могут использоваться в качестве выходов модуля генератора ШИМ сигнала (модуль PWM), если работа последнего программно разрешена.

8) PORT DLC. Двунаправленный 8_разрядный порт ввода/вывода общего назначения. Направление передачи каждой линии порта определяется соответствующим битом регистра DDRDLC. Альтернативная функция двух линий порта PORT DLC – обслуживание модуля последовательного обмена в стандарте BDLC.

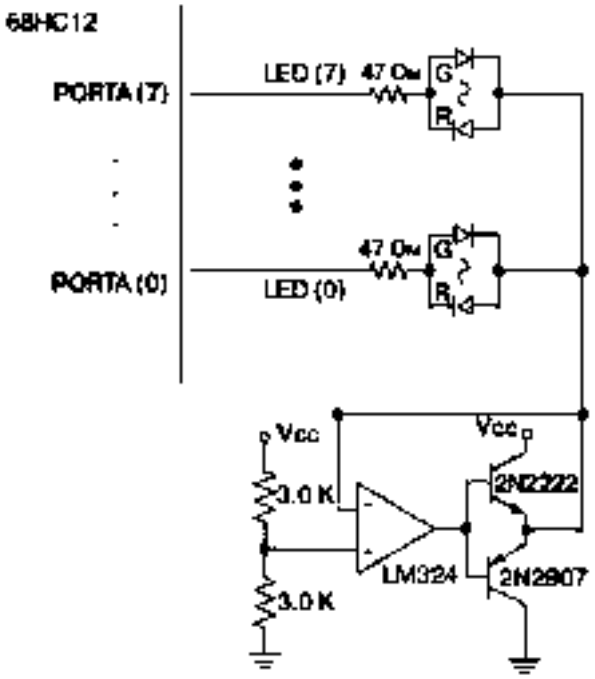
Регистры управления портами В МК семейства 68HC12/HCS12 каждый двунаправленный порт ввода/вывода общего назначения обслуживается двумя регистрами специальных функций. Это регистр данных порта и регистр направления передачи DDRx (вместо буквы «x» следует подставить буквенное обозначение порта). Если линии порта общего назначения настроены на ввод, то операция чтения регистра данных возвращает состояние выводов корпуса МК, с которыми связан порт. Если порт настроен на вывод, то операция записи в регистр данных устанавливает на выводах корпуса МК, связанных с портом, соответствующие логические уровни. Регистр DDRx определяет направления передачи каждой линии порта независимо от

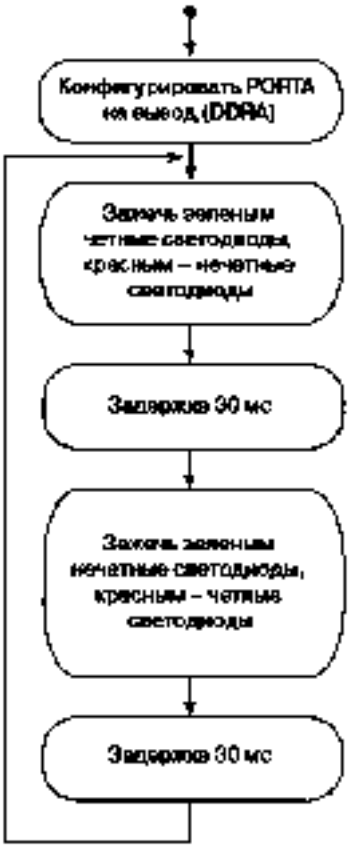
других линий этого же порта. Если какой либо бит регистра DDRx равен 0, то соответствующая линия настраивается на ввод, если 1 – то на вывод. Возможны решения, при которых часть линий одного и того же порта настроена на ввод, а часть на вывод. Например, при значении DDRx=10110010 линии D6, D3, D2 и D0 развернуты на ввод, а21 линии D7, D5, D4 и D1 – на вывод. В состоянии сброса МК все биты регистров направления передачи DDRx сбрасываются, поэтому сразу после включения питания все линии портов МК конфигурированы как входы с высоким входным сопротивлением.

Практическая часть

Перед началом выполнения практической части необходимо ознакомиться с лабораторной установкой на базе платы Freescale и программным обеспечением CodeWarrior. В практической части мы рассмотрим работу порта PORTA. Для этого подключим группу зеленых и красных светодиодов к выводам этого порта. Схема подключения показана на рисунке ниже.

Договоримся, что если на линии порта установлена 1, то будет гореть зеленый светодиод, если логический 0 – то красный светодиод. А если линия порта переведена в состояние ввода, т.е. она представляется для цепи светодиодов нагрузкой с высоким входным сопротивлением, то оба светодиода окажутся погашенными. Ниже приведена блок-схема алгоритма программы, которая зажигает на 30 мс зеленым цветом светодиоды с четными номерами и одновременно красным цветом светодиоды с нечетными номерами. Следующие 30 мс светодиоды «меняются цветами», далее этот процесс продолжается до бесконечности. Ниже приведен текст программы на языке Си, который реализует этот алгоритм.





```
/*_____*/
/* MAIN PROGRAM: Эта программа «зажигает» на выходах порта PORTA */
/*30 мс зеленым цветом горят светодиоды на выходах порта с четными */
/*номерами, красным цветом – светодиоды на выходах порта с нечетными */
/* номерами. Следующие 30 мс на месте зеленых горят красные, и наоборот */
/*_____*/
/*подключаемые файлы*/
#include<912b32.h>
/*используемые функции*/
void delay_100us(void)
void delay_30ms(void)
void main(void)
{
DDRA = 0xFF //установить порт PORTA на вывод
while(1)
{
PORTA = 0x55;
delay_30ms();
PORTA = 0xAA;
delay_30ms();
}
}
/*_____*/
/* Функция delay_30ms формирует задержку в 30мс мкс, частота тактирования*/
/* межмодульных магистралей МК составляет 8 МГц */
/*_____*/
void delay_30ms(void)
{
int i;
for (i=0; i<=299; i++)
delay_100us();
}
/*_____*/
/* Функция delay_100us формирует задержку в 100 мкс, частота тактирования */
/* межмодульных магистралей МК составляет 8 МГц */
/*_____*/
void delay_100us(void)
{
int j;
for (j=0; j<50; j++)
{
asm(«nop»\n) ;
}
}
/*_____*/
```

Обратите внимание, что функция задержки на 30 мс использует вложенную функцию задержки на 100 мкс.

В приведенном тексте программы для формирования задержки на 100 мкс используются 50 циклов повторения операторов функции delay_100us. Выбор числа повторений производился из предположения, что данная программа будет исполняться микроконтроллером, частота внутренней шины которого составляет 8 МГц. По результатам рассмотрения файла в формате *.lst было установлено, что команды ассемблера, соответствующие одному повторению цикла функции delay_100us , реализуются за 16 машинных циклов. При частоте шины в 8 МГц для формирования временного интервала в 100 мкс потребуется 800 машинных циклов. Поэтому число повторений цикла функции delay_100us должно составлять 800/16 = 50. Если бы эта программа исполнялась бы микроконтроллером DP256, частота внутренней шины которого составляет 25 МГц, то число циклов функции delay_100us должно было бы быть увеличено до 156.

Контрольные вопросы

1) Назначение устройств ввода/вывода информации.
2) Типы портов и алгоритмы обмена информацией.

3) Направление работы портов.
4) Спецификация портов.
5) Специальные регистры управления портами.
6) Дополнительные регистры управления портами.
7) Схема для реализации проверки состояния линий портов.
8) Программирование портов ввода/вывода.
9) Программная реализация временной задержки на языке Си и Ассемблер.

Практическая работа №78. Программные модели аппаратных средств микропроцессорных систем (минимизирование).

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 №79. Программные модели аппаратных средств микропроцессорных систем (логические схемы).

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 №80. Программные модели аппаратных средств микропроцессорных систем (формирователи импульсов).

Цель работы: изучение способов обращения к специализированным внешним устройствам микропроцессорных систем (МПС), определение и анализ со держимого внешнего запоминающего устройства (ВЗУ) ПК – NVRAM (NonVolatile RAM) или CMOS памяти.

Теоретическая часть.

Термин CMOS описывает технологию производства микросхемы часов реального времени (RTC – Real-Time Clock) и некоторого количества ячеек памяти в её составе. Микросхема часов реального времени изготавливается по технологии КМДП (CMOS), когда элементной базой БИС являются экономичные по потребляемой от источника электропитания мощности комплимен- тарные логические элементы структуры «металл-диэлектрик-

полупроподник». Необходимость включения CMOS памяти в состав аппаратуры ПК крайне сомнительна и её наличие можно отнести к странностям архитектуры ПК типа IBM PC. При включении электропитания ПК сначала выполняется программа самотестирования POST, по результатам работы которой заполняется область данных BIOS в оперативной памяти ЭВМ.

Внешнее ЗУ (NVRAM) также хранит некоторые данные, описывающие текущую конфигурацию ПК. Обычно эти данные дублируют содержимое области данных BIOS, но иногда эти данные вносятся в NVRAM оператором вручную. Размещение в памяти и внешнем ЗУ разнообразных данных, описывающих аппаратуру и программное обеспечение ЭВМ, выполняется стандартным образом для всех поколений ПК.

Практическая часть.

Программа, текст которой содержит Листинг 2, копирует в оперативную память содержимое первых 64 ячеек внешнего ЗУ и интерпретирует их в форме, доступной для восприятия человеком-оператором. Програм ма обращается к ячейкам внешнего ЗУ, используя метод регистровой косвенной адресации. В регистр внешнего устройства с адресом 0x70 заносится адрес реги- стра (ячейки памяти ВЗУ), к которому нужно обратиться, а затем после некоторой задержки считывается содержимое этого регистра по адресу 0x71. По странной случайности из ВЗУ можно таким способом считать закодированный пароль входа в программу BIOS Setup, задавая адреса восьми ячеек выше адреса 0x77. Кроме того, программа записывает содержимое первых 64 регистров ВЗУ в текстовый файл на магнитном диске.

Листинг 2

```
#include  <stdio.h>
#include  <dos.h>
#include  <bios.h>
#include  <conio.h>

FILE *uk;
unsigned char i,pam[64],password[8];
union basepam
      {
       unsigned int bapa;
       unsigned char osm[2];
      }bapam;
union expamsize
      {
       unsigned int expa;
       unsigned char expam[2];
      }epam;
union expamsi
      {
       unsigned int ex;
       unsigned char exm[2];
      }epamx;
union dis
      {
       unsigned char kol;
       struct disket
           {
            unsigned int secdis:4;
            unsigned int firdis:4;
           }n;
      }tydis;
union hdis
      {
       unsigned char hpre;
       struct hdisker
           {
            unsigned int fihadi:4;
            unsigned int sehadi:4;
           }m;
      }tyhdi;
union equip
      {
       unsigned char eqbyte;
       struct byeq
```

```
            {
             unsigned int askdis:1;
             unsigned int copro:1;
             unsigned int pusto:2;
             unsigned int disp:2;
             unsigned int koldri:2;
            }q;
      }ment;
union PamCMOS
      {
       unsigned int sum;
       unsigned char chsum[2];
      }chksum;
union mi
      {
       unsigned char bb;
       struct
          {
           unsigned int mus:6;
           unsigned int setup:1;
           unsigned int pal28:1;
          }sib;
      }misc;

void main(void)
{
 clrscr();
 for(i=0;i<=63;i++)
    {
     outportb(0x70,i);
     delay(1);
     pam[i]=inportb(0x71);
    }
 uk=fopen("XPAMCMOS.txt","wb");
 for(i=0;i<=63;i++)
    fprintf(uk,"%c",pam[i]);
 fclose(uk);
 ment.eqbyte=pam[0x14];
 epam.expam[0]=pam[0x30];
 epam.expam[1]=pam[0x31];
 tydis.kol=pam[0x10];
 tyhdi.hpre=pam[0x12];
 bapam.osm[0]=pam[0x15];
 bapam.osm[1]=pam[0x16];
 epamx.exm[0]=pam[0x17];
 epamx.exm[1]=pam[0x18];
 chksum.chsum[0]=pam[0x2f];
 chksum.chsum[1]=pam[0x2e];
 misc.bb=pam[0x33];
 printf("\n\r Содержимое CMOS - памяти:\n\r"
      "Hex Адрес\n\r"
      "32     Столетие - %02X\n\r"
      "9...6  Год - %02X месяц - %02X день - %02X"
      " день недели - %02X\n\r"
      "4,2,0  Час - %02X минута - %02X секунда - %02X\n\r"
```

```
"A,B    Регистры A и B состояния ЧРВ: A - %02X"
" Hex B - %02X Hex\n\r"
"C      Регистр ROM C состояния ЧРВ - %02X Hex\n\r"
"D      Регистр D состояния батареи Часов Реального"
" Времени - %02X Hex\n\r"
"       (D=80 Hex - батарея исправна, ну а если D=00"
" Hex, то - разряжена)\n\r"
"E      Регистр диагностики POST - %02X Hex\n\r"
"F      Регистр состояния отключения - %02X Hex\n\r"
"10     Дискетник 1 - %X    Дискетник 2 - %X\n\r"
"12     Винт C? - %X Hex     Винт D? - %X Hex\n\r"
"14     Дискетников - %X Тип дисплея - %X "
"Копроцессор - %X Дискетники? - %X\n\r"
"16,15  Основная память - %u килобайт\n\r"
"18,17  Размер расширенной памяти первый раз - %u"
" килобайт\n\r"
"19,1A  Тип драйва C: - %u  Тип драйва D: - %u\n\r"
"2F,2E  Контрольная сумма CMOS с адр.10 Hex по адр.20
    " Hex - %u\n\r"
    "31-30  Размер расширенной памяти второй раз - %u"
    " килобайт\n\r"
    "33     Вснкота: Опция 128 K - %u Используется"
    " Setup`ом - %u\n\r"
    "                          ...Жми любую!...",
    pam[0x32],pam[0x9],pam[0x8],pam[0x7],pam[0x6],pam[0x4],
    pam[0x2],pam[0],pam[0xa],pam[0xb],pam[0xc],pam[0xd],
    pam[0xe],pam[0xf],tydis.n.firdis,tydis.n.secdis,
    tyhdi.m.fihadi,tyhdi.m.sehadi,ment.q.koldri,ment.q.disp,
    ment.q.copro,ment.q.askdis,bapam.bapa,epamx.ex,pam[0x19],
    pam[0x1a],chksum.sum,epam.expa,misc.sib.pa128,misc.sib.setup);
bioskey(0);
clrscr();
gotoxy(30,12);
for(i=0;i<=11;i++)
    printf("%02X ",pam[0x34+i]);
gotoxy(30,14);
printf("Вот");
for(i=0;i<=7;i++)
   {
    outportb(0x70,0x78+i);
    delay(1);
    password[i]=inportb(0x71);
    printf("%c-%02X ",password[i],password[i]);
   }
bioskey(0);
clrscr();
)//Нажмите ENTER !!!
```

1. В интегрированной среде программирования (IDE) BorlandC++ вер.3.1 набрать текст программы, который содержит Листинг 2. 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 анализа энергонезависимой внешней памяти ПК, описание результатов работы программы и выводы по результатам исследования.

Практическая работа №81. Программные модели аппаратных средств микропроцессорных систем (схемы с памятью).

Цель работы: изучение способов обращения и структуры специализированных внешних устройств микропроцессорных систем (МПС).

Теоретическая часть.

Любой ПК типа IBM PC содержит устройство, называемое системным таймером. Адреса всех регистров таймера 8254 (К1810ВИ54) размещены в адресном пространстве внешних устройств, а его выходные сигналы используются в качестве управляющих в некоторых узлах системы. Микросхема таймера содержит три однотипных узла – каналы 0, 1 и 2. Выходной сигнал канала 0 подаётся на линию номер 0 запроса прерываний системного контроллера прерываний.

Системный контроллер прерываний состоит из двух каскадно включенных микросхем 8259 (К1810ВН59) и на вход запроса на прерывания IRQ0 и подаётся выходной сигнал канала 0 системного таймера. Системный контроллер преры- ваний инициализируется при включении ПК программой POST и вырабатывает прерывание INT 8h приблизительно 18,2 раза в секунду (точное значение - 1193180/65536 раз в секунду), что определяется инициализацией микросхемы системного таймера. При инициализации системы ПК BIOS устанавливает свой обработчик для прерывания таймера. Этот обработчик каждый раз увеличивает на 1 текущее значение четырехбайтовой переменной, располагающейся в области данных BIOS по адресу 0000:046Ch – счетчик тиков таймера. Если этот счетчик пере- полняется (прошло более 24 часов с момента запуска таймера), в ячейку 0000:0470h заносится 1.

Другое действие, выполняемое стандартным обработчиком прерывания таймера – контроль за работой двигателей накопителя на гибком магнитном диске (НГМД). Если после последнего обращения к НГМД прошло более 2 секунд, обработчик прерывания выключает двигатель. Ячейка с адресом 0000:0440h содержит время, оставшееся до выключения двигателя. Это время постоянно уменьшается обработчиком прерывания таймера. Когда оно становится равно 0, обработчик выключает двигатель НГМД. Последнее действие, которое выполняет обработчик прерывания таймера – вызов прерывания INT 1Ch. После инициализации системы вектор INT 1Ch указывает на команду IRET, т.е. ничего не выполняется. Программа пользователя может установить собственный обработчик этого прерывания для того чтобы выполнять какие-либо периодические действия. Необходимо отметить, что прерывание INT 1Ch вызывается обработчиком прерывания INT 8h до сброса контроллера прерывания, поэтому во время вы- полнения прерывания INT 1Ch все аппаратные прерывания запрещены. В частности, запрещены прерывания от клавиатуры. Обработчик прерывания INT 1Ch должен заканчиваться командой IRET. Если же вы подготавливаете собственный обработчик для прерывания INT 8h, пе- ред завершением его работы необходимо сбросить контроллер прерываний. Это можно сделать, например, так: 16 mov al, 20h out 20h, al Таймеры 8253 (КР1810ВИ53) и 8254 (КР1810ВИ54) состоят из трёх независимых каналов, или счётчиков. Каждый канал содержит регистры:

- ♦состояния канала RS (8 разрядов);
- ♦управляющего слова RSW (8 разрядов);
- ♦буферный регистр OL (16 разрядов);
- ♦регистр счетчика CE (16 разрядов);
- ♦регистр констант пересчета CR (16 разрядов).

Каналы таймера подключаются к внешним устройствам при помощи трёх линий:

- ♦GATE – управляющий вход;
- ♦CLOCK – вход тактовой частоты;
- ♦OUT – выход таймера.

Регистр счётчика CE работает в режиме вычитания. Его содержимое уменьшается по заднему фронту сигнала CLOCK при условии, что на входе GATE установлен уровень лог. 1. В зависимости от режима работы таймера при достижении счётчиком CE нуля тем или иным образом изменяется выходной сигнал OUT. Буферный регистр OL предназначен для запоминания текущего содержимого регистра счётчика CE без остановки процесса счёта. После

запоминания буфер- ный регистр доступен программе для чтения. Регистр констант пересчёта CR может загружаться в регистр счётчика, если это требуется в текущем режиме работы таймера. Упрощенная схема взаимодействия регистров канала приведена на рис. 3.1.

Возможны шесть режимов работы таймера. Они разделяются на три типа:

♦ режимы 0, 4 – однократное выполнение функций;

♦ режимы 1, 5 – работа с перезапуском;

♦ режимы 2, 3 – работа с автозагрузкой.

В режиме однократного выполнения функций перед началом счёта содержимое регистра констант пересчёта CR переписывается в регистр счётчика CE по сигналу CLOCK, если сигнал GATE установлен в лог. 1. В дальнейшем содержимое регистра CE уменьшается по мере прихода импульсов CLOCK. Процесс счёта можно приостановить, если подать на вход GATE уровень логического 0. Если затем на вход GATE подать лог. 1, счёт будет продолжен дальше.

Для CR CE OL Управляющая логика канала GATE CLOCK OUT повторения выполнения функции необходима новая загрузка регистра CR, т.е. повторное программирование таймера. При работе с перезапуском не требуется повторного программирования таймера для выполнения той же функции. По фронту сигнала GATE значение константы из регистра CR вновь переписывается в регистр CE, даже если текущая операция не была завершена. В режиме автозагрузки регистр CR автоматически переписывается в регистр CE после завершения счёта. Сигнал на выходе OUT появляется только при на- личии на входе GATE уровня лог. 1. Этот режим используется для создания программируемых импульсных генераторов и генераторов прямоугольных им- пульсов, например, со скважностью 2 (меандра).

В ПК типа IBM PC задействованы все три канала таймера и при включении ПК программа POST инициализирует их. Канал 0 используется в системных часах времени суток (не следует путать с часами реального времени, реализованными на другой микросхеме). Этот канал работает в режиме 3 и используется как генератор импульсов с частотой при- мерно 18,2 Гц. Именно эти импульсы (тики) вызывают аппаратное прерывание с вектором 8h (INT 8h).

Канал 1 используется для регенерации содержимого динамической памяти компьютера. Выход канала OUT используется для запроса к каналу прямого доступа контроллера DMA, который и выполняет обновление содержимого памяти. Пользователю не следует перепрограммировать этот канал, так как это может привести к нарушениям в работе основной оперативной памяти персо- нального компьютера.

Канал 2 подключен к громкоговорителю компьютера и может быть использован для генерации различных звуков или музыки, либо как генератор случайных чисел. Канал использует режим 3 таймера 8253/8254. Регистрам таймера соответствуют четыре порта ввода/вывода со следующими адресами в адресном пространстве внешних устройств ПК:

♦ 40h – канал 0;

♦ 41h – канал 1;

♦ 42h – канал 2;

♦ 43h – управляющий регистр.

Формат управляющего регистра приведён на рисунке ниже. Поле BCD определяет формат константы, использующейся для счёта – двоичный или двоично-десятичный. В двоично-десятичном режиме константа задаётся в диапазоне 1–9999. Поле M определяет режимы работы микросхемы 8254:

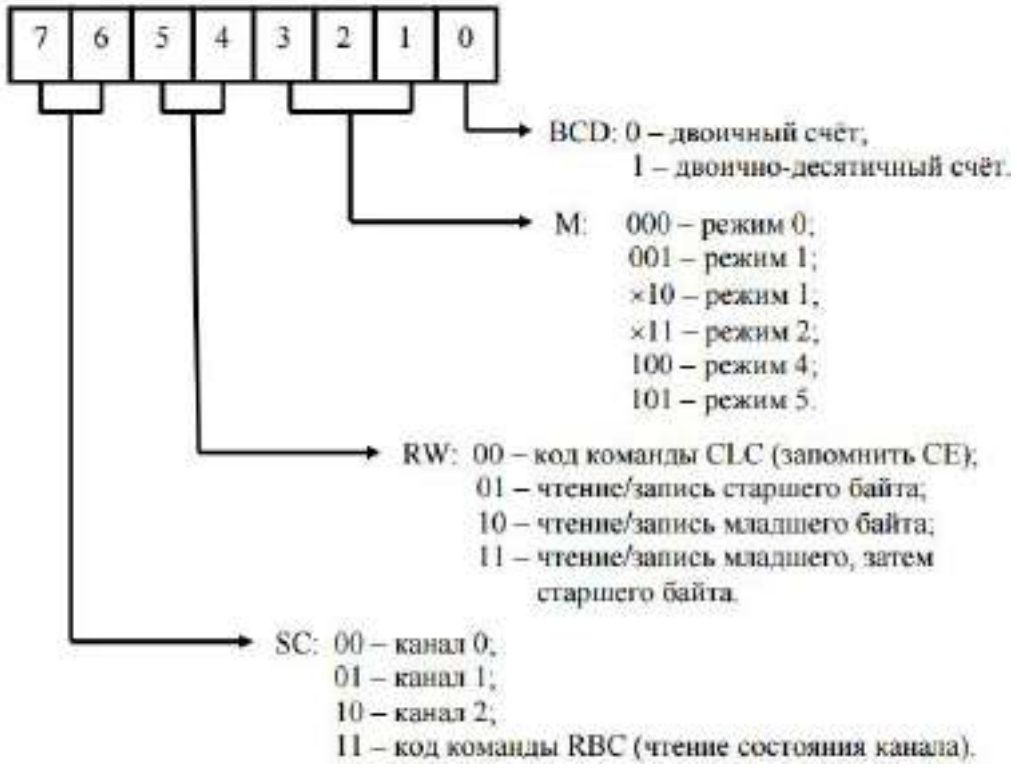
♦ 0 – прерывание от таймера;

♦ 1 – программируемый ждущий мультивибратор;

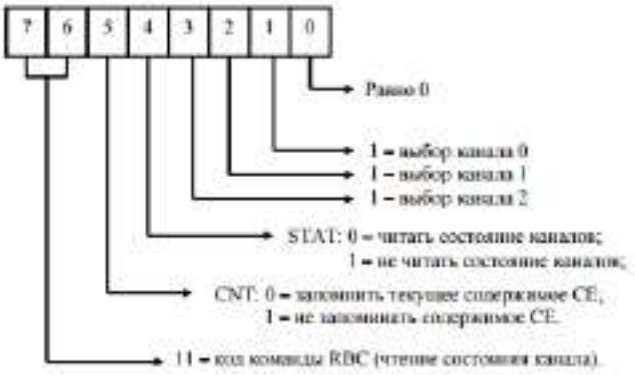
♦ 2 – программируемый генератор импульсов;

♦ 3 – генератор меандра;

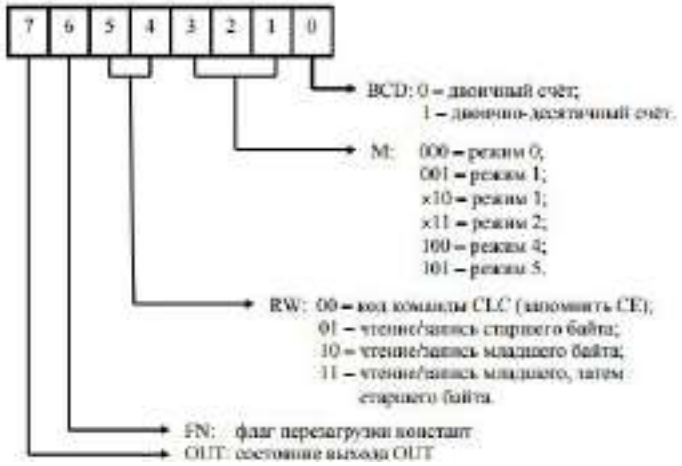
♦ 4 – программно-запускаемый одновибратор;
♦ 5 – аппаратно-запускаемый одновибратор.



В ПК в каналах 0 и 2 используется режим 3. Поле RW определяет способ загрузки констант через однобайтовый порт. Если в этом поле задано значение 00, это управляющее слово будет использоваться для фиксации текущего содержимого регистров счетчика CE в буферном регистре OL с целью чтения программой. Это код команды CLC – фиксация регистров. Код канала, для которого будет выполняться фиксация, должен быть указан в поле SC. Поля M и BCD при этом не используются. Поле SC определяет номер канала, для которого предназначено управляю- щее слово. Если в этом поле задано значение 11, будет выполняться чтение состояния канала (команда RBC). Формат управляющей команды RBC (чтение слова состояния канала) представлен на следующем рисунке:



С помощью этой команды выполняется операция чтения состояния каналов и запоминание регистра счётчика CE каналов. Можно выполнять эти операции, как для отдельных каналов, так и для всех каналов одновременно, если устано- вить соответствующие биты (1, 2, 3) в 1. Формат слова состояния канала напоминает формат регистра управляющего слова, за исключением двух старших разрядов 7 и 6:



Разряд FN используется, в основном, в режимах 1 и 5 для определения, произошла ли загрузка константы из регистра CR в регистр счётчика CE. Разряд OUT позволяет определить состояние выходной линии канала OUT в момент выполнения команды RBC. Для программирования канала таймера необходимо выполнить следующую последовательность действий:

♦ вывести в порт управляющего регистра с адресом 43h управляющее слово;

♦ требуемое значение счетчика посылается в порт канала (адреса 40h...42h), причём вначале выводится младший, а затем старший байты значения счётчика. Сразу после этого канал таймера начнёт выполнять требуемую функцию.

Для чтения текущего содержимого счётчика CE необходимо выполнить следующее:

♦ вывести в порт управляющего регистра код команды CLC (команда запо- минания содержимого регистра CE);

♦ вывести в порт управляющего регистра код команды запроса на чте- ние/запись в регистры канала (поле RW должно содержать 11);

♦ двумя последовательными командами ввода из порта нужного канала вве- сти младший и старший байты текущего состояния счетчика CE.

Перепрограммирование каналов таймера выполняется тогда, когда требуется повысить, например, точность измерения времени, выполняемого с помощью канала 0 таймера. При инициализации таймера частота этих импульсов устанавливается равной 18,2 Гц. По окончании измерений режим работы канала необходимо восстановить для правильного функционирования всей микропроцессорной системы.

Канал 2, подключенный к громкоговорителю, может использоваться для генерации различных звуков или даже для воспроизведения музыки. Этот же канал может быть использован для генерации случайных чисел. Описание работы. Таймер является неотъемлемой частью вычислительной микропроцессорной системы – персонального компьютера типа IBM PC и для управления им и его использования для управления вычислительной системой в составе системного программного обеспечения предусмотрены специальные программы. В Листинге 3 приведен текст пользовательской программы, с по- мощью которой выполняется управление каналами таймера с целью задания временных интервалов и частот для генерируемого каналом 2 звукового сигна- ла.

Листинг 3

```
#include  <dos.h>
#include  <math.h>
#include  <bios.h>
#include  <conio.h>

int i,j,f;
char oldport,newport;

union{
      int  Wdel;
```

```
      char Bdel[2];
     }del;
void main(void)
{
 oldport=inportb(0x61);
 newport=oldport|0x03;
 outportb(0x61,newport);
 outportb(0x43,0xb6);
          //Управляющее слово 0xb6=10110110
          // 10110110
          // |/|/|//|
          // | | |/ |
          // | | |  |_двоичное число делитель
          // | | |____один из пяти режимов (M3)
          // | |______считывание/запись сначала младшего,
          // |                    затем старшего байта
          // |________канал таймера N2

 for(i=0;i<=0xff;i++)
     {
      del.Wdel=(int)(1193.180-1100.0*sin(2.0*M_PI*i/0xff));
      cprintf("Целый делитель %04X hex=%5u Dec, "
              "два байта = %02X %02X\n\r"
             "   Частота равна %u Hz\n\r",
             del.Wdel,del.Wdel,
             del.Bdel[1]&0x00ff,del.Bdel[0]&0x00ff,
             (1193180/del.Wdel));
     outportb(0x42,del.Bdel[0]);
     outportb(0x42,del.Bdel[1]);
     delay(150);
     }
outportb(0x61,oldport);
f=1000;
sound(f);
outportb(0x43,0x86);
outportb(0x43,0xce);
outportb(0x43,0x80);

del.Bdel[0]=inportb(0x42);
del.Bdel[1]=inportb(0x42);
cprintf("\n\rДля частоты канала 2 %u Hz\n\r"
        "текущее содержимое двух байт счётчика = "
        "%X hex %X hex\n\r"
        "или = %u Dec\n\r",
              f,del.Bdel[1]&0x00ff,del.Bdel[0]&0xff,del.Wdel);
cprintf("\n\n\n\rPress Anny...");
bioskey(0);
nosound();
outportb(0x43,0x40);
j=0;
for(i=0;i<=0x45;i++)
  {
   del.Bdel[0]=inportb(0x40);
   del.Bdel[1]=inportb(0x40);
   j++;
   if(j>=15)
     {
      cprintf("\n\n\rPress Anny...");
      bioskey(0);
      j=0;
     }
   cprintf("\n\rТекущее содержимое счётчика канала N0 = "
           "%X hex %X hex\n\r"
           " или = %u Dec\n\r",
                 del.Bdel[1]&0x00ff,del.Bdel[0]&0x00ff,del.Wdel);
  }
 cprintf("\n\n\rPress Anny...");
 bioskey(0);
 clrscr();
}//press ENTER
```

Практическая часть.

1. В интегрированной среде программирования (IDE) BorlandC++ вер.3.1 набрать текст программы, который содержит Листинг 3. 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 работе. Обратить внимание на случайный характер последовательности чисел, считанных как текущее содержимое регистров счётчиков.

Содержание отчёта

Отчёт должен содержать подробно прокомментированный листинг програм- мы управления системным таймером ПК, описание результатов работы про- граммы и выводы по результатам исследования.

УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Рекомендуемая литература

Основные источники:

1. Микропроцессорные системы [Электронный ресурс] : учебное пособие для вузов / Е.К. Александров [и др.]. — Электрон. текстовые данные. — СПб. : Политехника, 2016. — 936 с. — 978-5-7325-1098-0. — Режим доступа: http://www.iprbookshop.ru/59491.html

2. Алиев, М.Т. Микропроцессоры и микропроцессорные системы управления. 8-разрядные процессоры семейства AVR: лабораторный практикум : учебное пособие / М.Т. Алиев, Т.С. Буканова. — Йошкар-Ола : ПГТУ, 2016. — 64 с. — ISBN 978-5-8158-1775-3. — Текст : электронный // Электронно-библиотечная система «Лань» : [сайт]. — URL: https://e.lanbook.com/book/92576

3. Богданов А.В. Микропроцессорные устройства релейной защиты и автоматизации в электроэнергетических системах [Электронный ресурс] : учебное пособие / А.В. Богданов, А.В. Бондарев. — ЭБСЭлектрон. текстовые данные. — Оренбург: Оренбургский государственный университет, ЭБС АСВ, 2016. — 82 с. — 8-987-903550-43-2. — Режим доступа: http://www.iprbookshop.ru/69913.html

Дополнительные источники:

1. Шишов, О.В. Аналого-цифровые каналы микропроцессорных систем управления : учебное пособие / О.В. Шишов. – М. ; Берлин : Директ-Медиа, 2015. – 211 с. : ил., схем., табл. – ISBN 978-5-4475-5273-2 ; То же [Электронный ресурс]. – URL: http://biblioclub.ru/index.php?page=book&id=363927 (11.01.2016).