ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Государственное образовательное учреждение высшего профессионального образования
«**ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**»

Ю.В. Алхимов

ЦИФРОВЫЕ И МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА

Лабораторный практикум

Издательство
Томского политехнического университета
2009

УДК 681.325.5-181.48(076.5)
ББК 32.973.26-04я73
А54

Алхимов Ю.В.

А54 Цифровые и микропроцессорные устройства: лабораторный практикум / Ю.В. Алхимов; Томский политехнический университет. – Томск: Изд-во Томского политехнического университета, 2009. – 168 с.

В учебном пособии представлены методические и справочные материалы для выполнения лабораторных работ, приведены описания лабораторных макетов и установок. Рассмотрены вопросы проектирования программного обеспечения для микропроцессоров и микроконтроллеров и применения современных отладочных средств и программ.

Разработано в рамках реализации Инновационной образовательной программы ТПУ по направлению «Неразрушающий контроль» и предназначено для студентов, аспирантов и специалистов в области НК.

УДК 681.325.5-181.48(076.5)
ББК 32.973.26-04я73

Рецензенты

Доктор технических наук, профессор ТУСУРа
Б.А. Люкшин

Кандидат физико-математических наук, доцент ТУСУРа
С.А. Бочкарева

ПРЕДИСЛОВИЕ

Эта книга является дополнением к учебному пособию «Микропроцессоры и цифровые системы в неразрушающем контроле» и призвана служить руководством для практических занятий, а также для подготовки к лабораторным работам по учебным курсам: «Микропроцессоры и цифровые системы в приборостроении», «Микропроцессоры в аппаратуре контроля и диагностики» и «Проектирование микропроцессорных средств измерений». Практические занятия, наряду с лабораторными занятиями, являются важной составной частью курсов, так как изучаемые предметы – прикладные инженерные дисциплины.

Лабораторный практикум содержит обширный материал, необходимый для подготовки и выполнения лабораторных работ по курсам, однако выполнение многих заданий требует изучения дополнительной литературы и привлечения справочных материалов. Список литературы, которой Вы можете пользоваться, приведен в конце учебника. Кроме того, Вы можете пользоваться фирменными справочными материалами и другими доступными материалами через сеть INTERNET. Посоветуйтесь с преподавателем, как это можно сделать лучше всего.

При подготовке к лабораторным работам сначала нужно внимательно изучить теоретический материал. При необходимости, для более детального изучения, нужно обратиться к дополнительной литературе или другим источникам. Затем нужно определить порядок выполнения работы, сделать необходимые таблицы, схемы и выполнить задания к работе. После выполнения работы сделать отчет. Требования к отчету приведены в описании каждой работы.

Лабораторная работа № 1
ОЗНАКОМЛЕНИЕ С РАБОТОЙ УЧЕБНОЙ МИКРОЭВМ

Цели работы: изучение режимов работы микроЭВМ, функционального назначения клавиш пульта оператора, директив программы «Монитор»; получение навыков записи программ и данных в память микроЭВМ и запуска программ на выполнение.

1. Методические указания к выполнению работы

1.1. Описание учебного микропроцессорного комплекта

Учебный микропроцессорный комплект (УМК) представляет собой учебную микроЭВМ, предназначенную для изучения программирования, проектирования и настройки микропроцессорных устройств и систем, выполненных на МП КР580ВМ80А (Intel 8080).

Структура УМК

Учебная микроЭВМ (рис. 1) состоит из модулей: центрального процессора, памяти, пошагового выполнения программы, пульта оператора и блока питания.

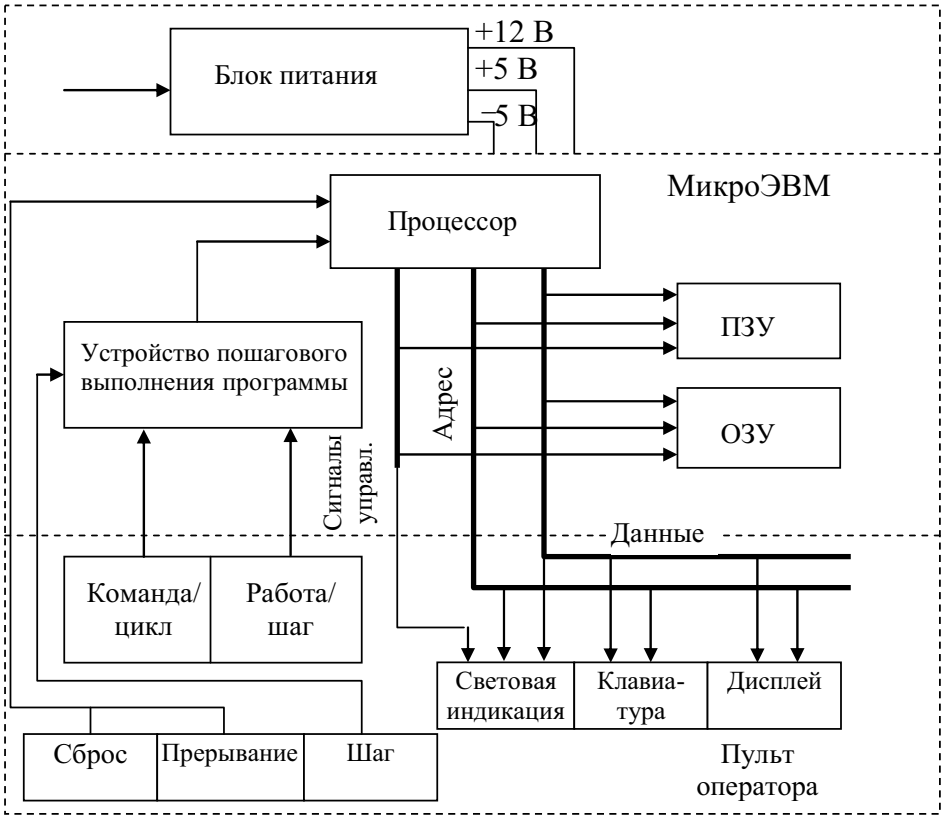

Рис. 1. Структурная схема УМК

Модуль центрального процессора выполнен на базе МП КР580ВМ80А и осуществляет все операции по обработке данных.

Пульт оператора (рис. 2) содержит клавиши:

- включения электропитания (11);
- сброса «СБ» (3);
- переключения режимов работы «РБ/ШГ» (работа/шаг) (6);
- выполнения шага «ШГ» (8);
- инициализации прерываний «ПР» (4);
- директив (10) – 8 клавиш;
- информационные (9) – 16 клавиш.

Рис. 2. Передняя панель УМК

На передней панели микроЭВМ расположены:

- светодиодные индикаторы адреса (1), данных (2) и регистра состояния МП (5);
- шестиразрядный цифровой дисплей для индикации адреса и данных в шестнадцатиразрядном коде (14);
- разъем для подключения макетных плат (13);
- индикаторы сбоя блока питания (12).

Модуль памяти содержит оперативное запоминающее устройство (ОЗУ) емкостью 2 Кбайт и постоянное запоминающее устройство (ПЗУ) емкостью 2 Кбайт. В ПЗУ записана системная программа «Монитор», обслуживающая ввод информации с пульта оператора и вывод ее на дисплей. Программа «Монитор» занимает 1 Кбайт ПЗУ и использует последние 54 ячейки ОЗУ (табл. 1).

Таблица 1

Распределение памяти микроЭВМ

Диапазон адресов	Тип памяти и ее назначение
0000H – 03FFH	Постоянное запоминающее устройство (ПЗУ). Здесь расположена программа «Монитор»
0400H – 07FFH	ПЗУ. Частично используется для вспомогательных программ
0800H – 0FC9H	Оперативное запоминающее устройство (ОЗУ). Хранение программ и данных пользователя
0FCAH – 0FFFH	ОЗУ. Хранение данных программы «Монитор»

Модуль интерфейса ввода/вывода подключается к УМК через разъем. На плате модуля расположена микросхема КР580ВВ55 – программируемый параллельный адаптер (ППА). Обращение к регистрам микросхемы ППА осуществляется по адресам 80H, 81H, 82H, 83H.

1.2. Режимы работы микроЭВМ

Переход в сходное состояние реализуется микропроцессором при нажатии кнопки «СБ». После этого МП переходит к чтению информации по нулевому адресу ПЗУ, а управление передается системной программе «Монитор», которая обеспечивает диалоговый режим работы с микроЭВМ.

Вызов директив «Монитора» осуществляется директивными клавишами, параметры директив задаются информационными клавишами.

Диалоговый режим работы позволяет выполнять следующие операции:

- чтение и модификация содержимого ячеек памяти (клавиша «П»);
- чтение и модификация содержимого регистров МП («РГ»);
- вычисление контрольной суммы массива памяти («КС»);
- перемещение заданного массива в памяти в адресном пространстве («ПМ»);
- заполнение массива памяти константой («ЗК»);
- выполнение программ пользователя с возможностью установки точки останова («СТ»).

Клавиша пробел «_» служит для разделения нескольких переменных при вводе. Клавиша «ВП» – выполнить – означает конец директивы. Введенная в микроЭВМ информация отображается в шестнадцатеричном коде на шестиразрядном дисплее, в четырех старших разрядах – адрес ячейки памяти или идентификатор регистра, в двух младших – содержимое ячейки или регистра.

Клавиши с 4/РН до F служат для ввода идентификаторов регистров МП. При неправильной работе с клавиатурой в крайней правой позиции дисплея индицируется знак «?».

Выполнение программ пользователя возможно как в автоматическом режиме, так и в пошаговом (задается переключателем «РБ/ШАГ»). В пошаговом режиме клавиша должна быть нажата.

В пошаговом режиме состояние шины адреса/данных и регистра состояний отражается в двоичных кодах светодиодами. Этот режим работы обеспечивается устройством пошагового выполнения программ, которое переводит МП в состояние «ОЖИДАНИЕ» после выполнения очередного шага. Переключателем «КМ/ЦК» (команда/цикл) выбирается размер шага. Для исполнения шага необходимо однократно нажать клавишу «ШГ». Выполнение программы может быть остановлено нажатием управляющей клавиши «ПР» (прерывание). При этом управление передается подпрограмме обработки прерываний командой RST7. Подпрограмма сохраняет состояние всех регистров процессора и производит передачу управления монитору. На дисплее индицируется содержимое счетчика команд, которое на единицу больше адреса выполненной команды. После этого можно вызвать выполнение любой директивы программы «Монитор». Выполнение прерванной программы возможно, начиная с адреса останова или любого другого адреса.

2. Подготовка УМК к работе

ВНИМАНИЕ! Запрещается:

- *соединять и разъединять разъемы УМК при включенном питании;*
- *устанавливать в вилку разъема для подключения макетной платы посторонние предметы;*
- *закрывать вентиляционные щели корпуса микроЭВМ;*
- *закрывать крышку при включенном в розетку сетевом кабеле.*

Для включения УМК необходимо:

1. Установить кнопку «~» в отжатое состояние.
2. Подключить сетевой кабель к сети 220 В, f = 50 Гц.
3. Переключатель «РБ/ШГ» установить в положение «РБ» (отжатое).
4. Нажать кнопку «~» (до ее фиксации в нажатом положении), подключив тем самым УМК к сети. Повторное включение возможно не ранее чем через 30 с.
5. Нажать управляющую кнопку «СБ» (сброс).

При этом происходит чтение информации по адресу 0000 ПЗУ, в котором находится первая команда системной программы «Монитор». Управление передается этой программе, о чем свидетельствует появление символа «–» в крайней левой позиции дисплея.

В режиме останова МП по команде «HLT» вызов программы «Монитор» можно осуществить нажатием кнопки «ПР».

3. Директивы программы «Монитор»

В общем случае команды программы «Монитор» имеют следующий вид:

КОП[ПАР1 _ ПАР2 _ ПАР3] ВП,

где КОП – идентификатор команды, соответствует одной из функциональных клавиш; ПАР1, ПАР2, ПАР3 – параметры команд (возможно от одного до трех параметров в зависимости от команды); ВП – клавиша, инициализирующая выполнение команды; [ ] – означает, что параметр может быть опущен.

Вся вводимая и выводимая информация представлена в шестнадцатеричном виде. Вводимые параметры «Данные» и «Адрес» отображаются по мере ввода в соответствующей части дисплея.

Если при вводе директивы допущена ошибка, то на дисплее выводится символ «?» и система возвращается в исходное состояние приема команд.

3.1. Команды программы «Монитор»

3.1.1. Чтение и модификация содержимого ОЗУ

Формат команды: П XXXX ВП,

где XXXX – шестнадцатеричный адрес ячейки памяти. После ввода команды в информационной части дисплея будет выведено содержимое указанной ячейки. Если оператор введет символ «_», то содержимое этой ячейки не изменится, а на дисплей будет выведен адрес и содержимое следующей ячейки.

Для модификации ячейки после ввода команды необходимо ввести новые данные, которые будут отображаться по мере ввода на дисплее. Для фиксации введенного значения необходимо нажать клавишу «_» (тогда система перейдет к следующей ячейке памяти) либо клавишу «ВП» – для завершения выполнения команды.

3.1.2. Чтение и модификация содержимого регистров микропроцессора

Формат команды: РГ X ВП,

где X – идентификатор регистра микропроцессора, соответствующий обозначениям, размещенным на числовой клавиатуре (выгравированы под цифрами, табл. 2).

После ввода команд на дисплее отображается идентификатор регистра и его текущее содержимое. Чтобы изменить содержимое, надо ввести новые данные, затем через разделитель «_» может быть введен идентификатор следующего регистра и т. д. Для завершения выполнения команды надо нажать клавишу «ВП».

3.1.3. Вычисление контрольной суммы массива памяти

Формат команды: КС A1 _ A2 ВП,

где A1 – начальный адрес массива; A2 – конечный адрес массива.

Контрольная сумма заданной области памяти подсчитывается как сумма значений всех ячеек памяти без учета переполнения. Команда служит для контроля сохранности информации в области памяти. После выполнения команды значение контрольной суммы индицируется на дисплее.

Таблица 2

Идентификаторы регистров микропроцессора

Идентификатор	Регистр микропроцессора
PCH	Старший байт счетчика команд
PCL	Младший байт счетчика команд
SPH	Старший байт указателя стека
SPL	Младший байт указателя стека
H	Регистр общего назначения H
L	Регистр общего назначения L
D	Регистр общего назначения D
E	Регистр общего назначения E
B	Регистр общего назначения B
C	Регистр общего назначения C
A	Аккумулятор
F	Регистр флагов

3.1.4. Выполнение программы пользователя

Формат команды: СТ [A1 _ A2] ВП,

где A1 – стартовый адрес выполнения программы; A2 – адрес точки останова.

Если задан только параметр A1, то управление передается программе пользователя, начиная с этого адреса, без возможности остановки программы в какой-либо точке. Если параметр A1 опущен (вместо него вводится «_»), то выполнение программы начинается с текущего значения счетчика команд и будет прервано в точке останова, если таковая задана. Если не указан ни один из параметров, то управление будет передано в точку текущего значения счетчика команд. Данный вариант команды без параметров может служить для продолжения выполнения программы пользователя после останова.

В момент останова на дисплее выводится адрес точки останова и запоминается значение всех регистров, которые можно прочесть, используя команду «РГ».

3.1.5. Заполнение массива ОЗУ константой

Формат команды:

ЗК А1 _ А2 _ К ВП,

где А1 – начальный адрес массива ОЗУ; А2 – конечный адрес массива ОЗУ; К – шестнадцатеричная константа.

В результате выполнения этой команды массив ОЗУ в заданных пределах будет заполнен значением константы.

3.1.6. Копирование областей памяти

Формат команды:

ПМ А1 _ А2 _ А3 ВП,

где А1 – начальный адрес копируемого массива; А2 – конечный адрес копируемого массива; А3 – начальный адрес записываемого массива.

В результате выполнения этой команды данные из одной области памяти переписываются в другую. Перекрытие областей не допускается.

4. Порядок выполнения работы

4.1. Включить УМК

Включить шнур питания УМК в сеть. Нажать клавишу «~». Нажать клавишу «СБ». Управление передается системной программе «Монитор», о чем свидетельствует появление символа «–» в крайней левой позиции дисплея.

При выполнении программ вызов системной программы «Монитор» можно осуществить по команде безусловного перехода JMP (код команды С3) по адресу 0000 – начальному адресу программы «Монитор».

4.2. Чтение содержимого ПЗУ и ОЗУ

Порядок выполнения задания:

1. Нажать клавишу «П», при этом на дисплее (в крайней левой позиции) гаснет символ «–».

2. Последовательно нажать клавиши «0», «3», «F», «F», «ВП». Убедиться при этом, что каждая цифра записывается в младший разряд адресного дисплея с последующим сдвигом влево при вводе следующей цифры. С нажатием клавиши «ВП» УМК выдает на дисплей число, записанное по этому адресу. На дисплее данных (две крайние правые позиции) появится число 91. Эта ячейка принадлежит программе «Монитор», и в ней записано число 91. В случае ошибки при введении кода адреса необходимо нажать последовательно клавиши «СБ» и «П» и повторить ввод. Можно два раза нажать клавишу «П» и повторить ввод.

3. Если ввести символ «_», то содержимое ячейки памяти не изменится, а на дисплей будет выведен адрес следующей ячейки и ее содержимое. Прочитайте содержимое ячеек ПЗУ 0100...010 А. Результат оформите в виде таблицы.

4. Вызовите начальный адрес ОЗУ пользователя 0800 и убедитесь еще раз, что можно проверить содержимое всех ячеек памяти УМК.

4.3. Запись информации в ОЗУ

Порядок выполнения задания:

1. Нажать клавишу «СБ».
2. Нажать клавишу «П». После этого ввести адрес 0800 и нажать клавишу «_». На дисплее будет выведен адрес и содержимое ячейки по этому адресу.
3. Набрать новые данные (две цифры), которые высветятся на дисплее данных.
4. Нажать клавишу «_» и перейти к следующей ячейке, или нажать клавишу «ВП», если конец записи.
5. Записать в ячейки 0800...080АН свои данные.
6. Прочитать записанную информацию из этих ячеек, результат оформить в таблицу.
7. Записать в ячейки ПЗУ 0100...010АН свои данные, например 0FFH.
8. Прочитать информацию из записанных ранее ячеек ПЗУ и убедиться, что информация в ПЗУ не сохранена и соответствует таблице, составленной в пункте 4.2.

4.4. Чтение содержимого программно-доступных регистров

Порядок выполнения задания:

1. Нажать клавишу «СБ».
2. Нажать клавишу «РГ», на дисплее погаснет символ «–».
3. Нажать клавишу «А» – идентификатор регистра А (аккумулятор) микропроцессора.
4. Считать данные с дисплея и нажать клавишу «ВП», если конец чтения, или нажать клавишу «_» и идентификатор следующего регистра.
5. Считать таким образом числа из всех программно-доступных регистров. Составить таблицу считанных данных.

4.5. Запись числа в программно-доступные регистры

Порядок выполнения задания:

1. Нажать клавишу «СБ».
2. Нажать клавишу «РГ».
3. Нажать клавишу с идентификатором программно-доступного регистра.
4. Ввести свои данные в вызванный регистр.
5. Повторить процедуру записи для всех программно-доступных регистров.
6. Прочитать содержимое регистров. Убедиться в правильности записанной в них информации.
7. Нажать клавишу «СБ».

8. Прочитать информацию из программно-доступных регистров еще раз, чтобы убедиться, что информация в регистрах при нажатии клавиши «СБ» возвращается к исходной.

4.6. Вычисление контрольной суммы массива памяти

Порядок выполнения задания:

1. Нажать клавишу «СБ».
2. Нажать клавишу «КС».
3. Ввести начальный адрес массива, например начальный адрес программы «Монитор» – 0000H.
4. Нажать клавишу «_».
5. Ввести конечный адрес программы «Монитор» 03FFH. Нажать клавишу «ВП». На дисплее данных отобразится контрольная сумма массива ПЗУ, в котором записана программа «Монитор».

4.7. Заполнение массива ОЗУ константой

Порядок выполнения задания:

1. Нажать клавишу «СБ».
2. Нажать клавишу «ЗК».
3. Ввести начальный адрес массива ОЗУ, например 0800H.
4. Нажать клавишу «_».
5. Ввести конечный адрес массива ОЗУ, например 0900H.
6. Нажать клавишу «_».
7. Ввести шестнадцатеричную константу, например 0AAH.
8. Нажать клавишу «ВП».
9. Прочитать содержимое нескольких ячеек из заполненного массива.

4.8. Копирование областей массива

Порядок выполнения задания:

1. Нажать клавишу «СБ».
2. Нажать клавишу «ПМ».
3. Ввести начальный адрес копируемого массива, например 0800H.
4. Нажать клавишу «_».
5. Ввести конечный адрес копируемого массива, например 08FFH.
6. Нажать клавишу «_».
7. Набрать начальный адрес записываемого массива, например 0900H.
8. Нажать клавишу «ВП». В результате этой команды данные копируются.
9. Проведите сравнение нескольких ячеек копируемого массива с соответствующими ячейками вновь записанного массива.

5. Содержание отчета

Отчет должен включать:

1) структурную схему микроЭВМ УМК;
2) таблицы результатов;
3) ответы на контрольные вопросы.

Лабораторная работа № 2
ИССЛЕДОВАНИЕ СТРУКТУРЫ МАШИННОГО ЦИКЛА МИКРОПРОЦЕССОРА INTEL 8080

Цели работы: исследование выполнения отдельных команд и простых программ; исследование процесса выполнения команд по машинным циклам; запись программ в машинных кодах и на языке ассемблер.

Оборудование: учебный микропроцессорный комплект (УМК); дополнительная макетная плата (плата ТЭЗ М1 из комплекта УМК); цифровой осциллограф.

1. Краткие сведения из теории

Микропроцессор Intel 8080 представляет собой однокристальный 8-разрядный микропроцессор. Структурная схема микропроцессора представлена на рис. 1. Микропроцессор имеет раздельные 8-разрядную шину данных и 16-разрядную шину адреса, которая обеспечивает адресное пространство памяти 64 Кбайт и раздельные адресные пространства ввода и вывода по 256 байт.

Микропроцессор состоит из устройства управления (УУ), блока регистров общего назначения (РОН), арифметико-логического устройства (АЛУ), схемы управления обменом (СУО), буферных схем данных и адреса (БУ и БД). Блоки микропроцессора объединены через 8-разрядную внутреннюю шину данных. УУ выполняет функции выборки команды, ее декодирования и выполнения, прием и выдачу управляющих сигналов для составных частей микропроцессора и системы.

Из системы УУ получает сигналы:

С1 и С2 – две не перекрывающиеся (т. е. не совпадающие во времени) периодические последовательности импульсов синхронизации;

INT – сигнал требования прерывания;

HLD – сигнал требования прямого доступа в память;

RESET – сигнал сброса (установка микропроцессора в исходное состояние);

RDY – сигнал готовности от составных частей МП-системы.

Устройство управления выдает в систему сигналы:

INTE – разрешение прерывания;

HLDA – разрешение прямого доступа в память;

WAIT – ожидание (микропроцессор находится в состоянии ожидания);

SYN – сигнал начала машинного цикла (выдается в начале каждого цикла).

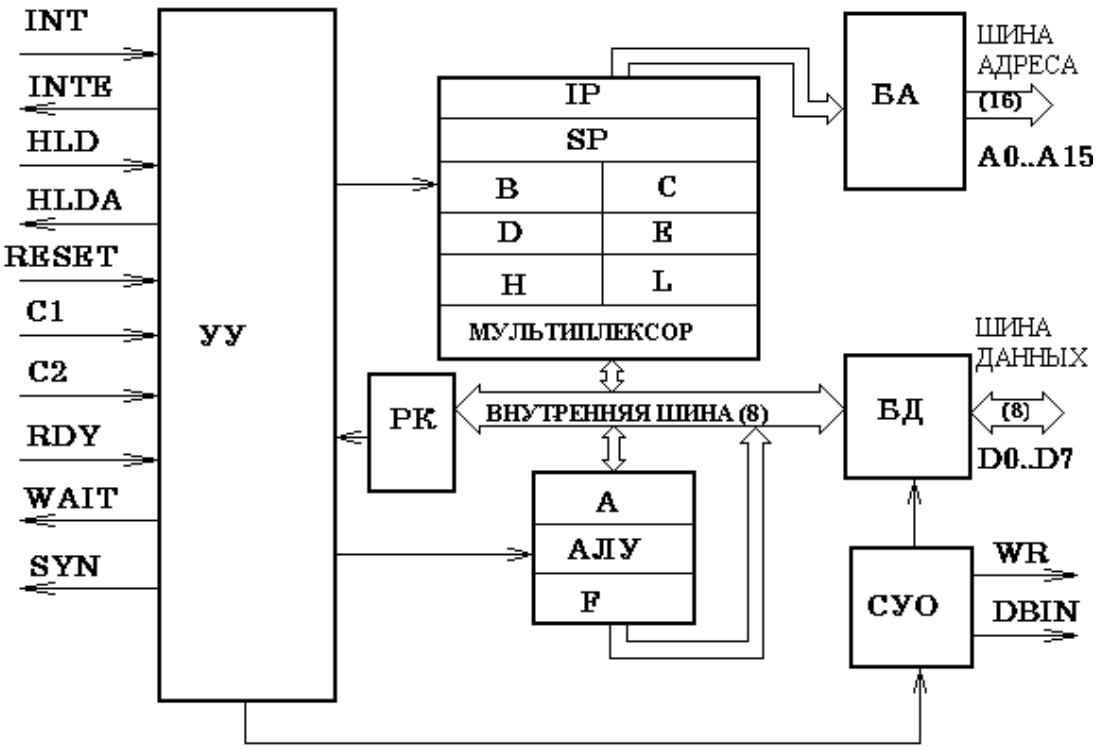

Рис. 1. Функциональная схема микропроцессора Intel 8080

Выполняемая команда хранится в специальном регистре, который называется регистром команд (РК).

Арифметико-логическое устройство совместно с аккумулятором (A) и регистром признаков (F) выполняет арифметические и логические операции с двоичными числами (они носят название операндов). В аккумуляторе хранится один из операндов перед выполнением операции и результат после выполнения операции. В регистр признаков помещаются признаки событий, происходящих при выполнении операций (например перенос из старшего значащего разряда, нулевой результат операции, отрицательный результат и т. д.).

Блок РОН включает шесть 8-разрядных регистров (B, C, D, E, H, L), которые могут использоваться попарно, как три 16-разрядных (BC, DE, HL), 16-разрядный регистр-счетчик команд (IP) и 16-разрядный регистр-указатель стека (SP). Схема управления обменом выдает сигналы, синхронизирующие обмен данными между микропроцессором и МП-системой. Сигнал WR сопровождает данные, выводимые из МП, а сигнал DBIN – синхронизирует прием информации. Этих сигналов явно недостаточно для управления всей системой, поэтому в начале каждого машинного цикла по шине данных передается служебная информация о типе операции, которая будет выполнена в текущем цикле. Эта информация сопровождается сигналом SYN, принимается специальной до-

полнительной схемой (она называется системным контроллером), которая, используя эту информацию, формирует все необходимые сигналы шины управления:

MRDC – чтение данных из памяти;

MWTC – запись данных в память;

IORC – чтение данных из порта ввода;

IOWC – запись данных в порт вывода.

Буферные схемы БА и БД обеспечивают необходимую нагрузочную способность шин адреса и данных, а также возможность отключения шин от внешней, по отношению к МП, системы (путем перевода выходов в третье состояние).

Микропроцессор имеет фиксированный набор из 244 команд. Выполнение команд происходит по машинным циклам. Каждый цикл – это выполнение элементарной операции по управлению шинами МП-системы. Время выполнения команды определяется процессом получения, декодирования и выполнения команды. В зависимости от вида команды время ее выполнения может составлять 1–5 машинных циклов. Машинный цикл состоит из 3–5 машинных тактов. Такт – это один период синхронизирующих импульсов. При типовой частоте импульсов синхронизации, равной 2 МГц, длительность одного такта равна 500 нс.

Для КР580ВМ80А имеется 10 различных типов машинного цикла: выборка кода команды (цикл М1), чтение данных из памяти, запись данных в память, чтение данных из стека, запись данных в стек, ввод данных из внешнего устройства, запись данных во внешнее устройство, цикл обслуживания прерываний, останов, обслуживание прерывания в режиме останова. Первым машинным циклом при выполнении любой команды является цикл М1. Тип выполняемого цикла, как указывалось выше, определяется информацией, выдаваемой на шину данных в начале каждого цикла. Соответствие типа цикла и сигналов в каждом разряде шины данных приведено в табл. 2 и 3 книги «УМК. Эксплуатационная документация» (лист 11). На шине УМК (плата ТЭЗ М1) разряду D0 соответствует сигнал ST0, разряду D1 – ST1 и т. д.

В каждом машинном цикле МП проверяет состояние сигнала «Готов» (RDY) на своем входе. При нулевом сигнале на этом входе работа МП приостанавливается. В УМК это используется для выполнения программы в пошаговом режиме. Одно нажатие кнопки «ШГ» приводит к выполнению одной команды или одного машинного цикла программы.

2. Исследование выполнения команд по машинным циклам

2.1. Присоединить к УМК дополнительную плату ТЭЗ М1. Общий провод платы М1 соединить с входом заземления, а контакт ST5 – с входом внешнего запуска осциллографа. Переключить осциллограф в режим внешней синхронизации. Пригласить преподавателя для проверки схемы.

ВНИМАНИЕ! НЕЛЬЗЯ ВКЛЮЧАТЬ СОЕДИНЕННУЮ СХЕМУ БЕЗ РАЗРЕШЕНИЯ ПРЕПОДАВАТЕЛЯ! ЭТО МОЖЕТ ПРИВЕСТИ К ВЫХОДУ ИЗ СТРОЯ УМК.

2.2. Включить УМК и осциллограф. Поставить переключатели разверток: по X – 1 В/дел., по Y – 1 мкс/дел. Добиться появления на экране линии развертки.

2.3.1. С адреса 0800H ввести в УМК последовательность шестнадцатеричных чисел 3A 00 09 C3 00 08. Эта последовательность представляет собой две команды. Первая – (3A 00 09) – команда извлечения числа из памяти и передача его в аккумулятор микропроцессора. Число 3A – код команды, 00 09 – адрес в памяти пересылаемого числа (этот адрес равен 0900H – в МП действует соглашение, что для двухбайтовых чисел младший байт расположен по меньшему адресу, старший – по большему). Вторая команда (C3 00 08) передает управление по адресу 0800H, т. е. приведенная программа – это бесконечное выполнение команды 3A 00 09.

2.3.2. Переключить УМК в пошаговый режим работы (кнопки «РБ/ШГ» и «КМ/ЦК» нажаты). Запустить программу, подав команду СТ 0800 ВП. Она остановится на первом шаге, причем на индикаторах будет высвечиваться информация. Нажимая кнопку «ШАГ», несколько раз выполним программу, информацию занесем в табл. 1.

Таблица 1

Шаг	Адрес	Данные	Состояние	Тип цикла	Комментарий
1	0000100000000000	00111010	10100010	M1	Загрузка команды
2	0000100000000001	00000000	10000010	Чтение	Чтение мл. байта адреса
3	0000100000000010	00001001	10000010	Чтение	Чтение ст. байта адреса
4	0000100100000000	10111111	10000010	Чтение	Чтение из памяти по адресу 0900H
5	0000100000000011	11000011	10100010	M1	Загрузка команды
6	0000100000000100	00000000	10000010	Чтение	Чтение мл. байта адреса
7	0000100000000101	00001000	10000010	Чтение	Чтение ст. байта адреса
8	0000100000000000	00111010	10100010	M1	Повтор шага 1

Примечание. На шаге 4 информация на индикаторах «Данные» может отличаться от приведенной в таблице.

Из анализа табл. 1 видим, что команда 3A 00 09 выполняется за 4 цикла (цикл M1 и 3 цикла чтения памяти). В первом цикле происходит загрузка кода команды 3A (находится в памяти по адресу 0800H) в РК микропроцессора. Во втором цикле читается младший байт адреса 00 (в памяти по адресу 0801H). В третьем цикле читается старший байт адреса 09 (в памяти по адресу 0802H). В четвертом цикле число, находящееся по адресу 0900 (в нашем случае оно равно 0BFH), считывается в аккумулятор микропроцессора.

2.3.3. Снять временную диаграмму выполнения команды 3A0009.

2.3.3.1. Переключить УМК в непрерывный режим работы (отжать кнопки «КМ/ЦК» и «РБ/ШГ»).

2.3.3.2. Снять и зарисовать временные диаграммы сигналов на контактах ST5, F2, IORC, IOWC, MRDC, MWTC, A0, A1, A8, D0. Все сигналы по времени на осциллографе привязаны к началу цикла M1 и должны быть зарисованы в одном временном масштабе. Сигнал F2 совпадает с сигналом синхронизации C2, и его период равен одному такту. Пример временной диаграммы приведен на рис. 2.

2.3.3.3. Проанализируем нашу временную диаграмму. Она показывает, что команда выполняется за 13 периодов сигнала F2, т. е. за 13 тактов, причем цикл M1 выполняется за 4 такта, а циклы чтения – за 3. Из сигналов синхронизации обмена информацией активным (активный уровень этих сигналов – низкий) бывает только сигнал MRDC (сопровождение чтения из памяти), что и следовало ожидать. Активным уровнем сигналов на шине адреса является низкий уровень, а на шине данных – высокий.

2.3.4. Записать в УМК с адреса 0800H 22 00 09 C3 00 08. Выполнить программу в пошаговом режиме, и результат занести в таблицу, аналогичную табл. 1. Снять временную диаграмму выполнения команды 22 00 09. Ответить на вопросы:

1. За сколько машинных циклов выполняется команда? За сколько тактов?

2. Какие циклы используются при выполнении команды?

3. Какие сигналы синхронизации обмена информацией бывают активны при выполнении команды?

4. Какие действия, по-Вашему, выполняет команда?

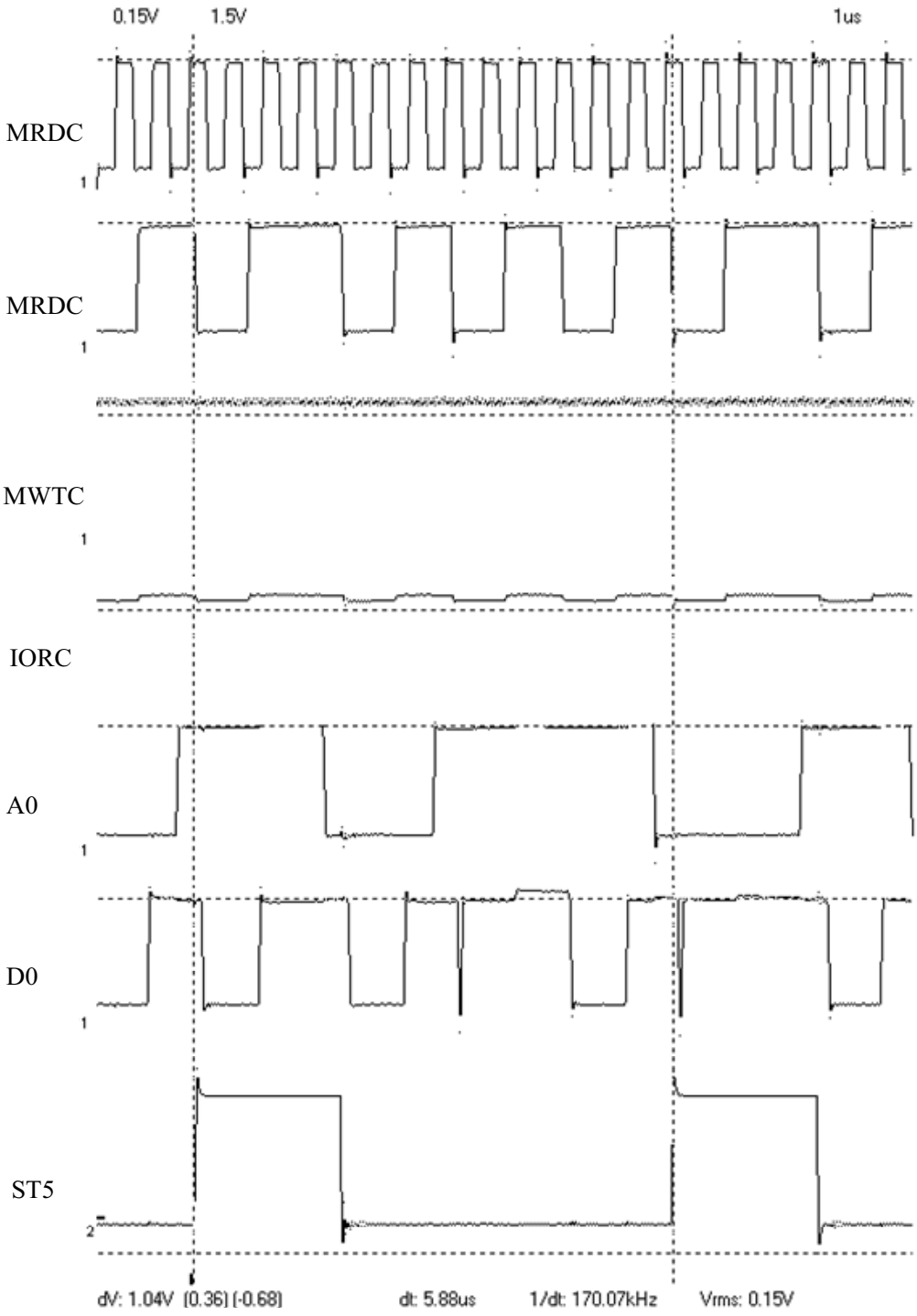

Рис. 2. Пример временной диаграммы

3. Выполнение простейших программ

Рассмотрим программу, извлекающую число из адреса памяти 0900, инвертирующую его и записывающую в адрес 0901. Для удобства программистов двоичные числа, которые являются командами, при написании программ заменяются буквенно-цифровыми кодами (мнемокодами), где буквенный код указывает, какую операцию нужно выпол-

нить, а цифры являются операндами команды. Мнемокод отражает суть производимого командой действия и является сокращением от соответствующих английских слов, например MOV – Move (двигать), передача данных между регистрами или регистрами и памятью микропроцессора.

Программа 1 (в мнемокодах)

```
Мнемокод                Комментарий
LDA 0900              ;Прочитать  число  из  памяти  по
                      ;адресу 0900H
CMA                   ;Инвертировать число
STA 0901              ;Записать результат по адресу 0901
HLT                   ;Останов
```

При записи программ все числа представляются в шестнадцатеричной системе счисления. Для записи программ в ОЗУ микроЭВМ необходимо перевести мнемокоды команд в машинные коды.

Команды в программе могут быть одно-, двух- или трехбайтовые и должны в памяти занимать, соответственно, одну, две или три ячейки.

Программа 1 (размещение по адресам памяти)

```
  Адрес        Число                  Комментарий
   0800          3A        ;   код команды
   0801          00        ;   младший байт адреса
   0802          09        ;   старший байт адреса
   0803          2F        ;   код команды CMA
   0804          32        ;   код команды STA
   0805          01        ;   младший байт адреса
   0806          09        ;   старший байт адреса
   0807          76        ;   код команды HLT
```

Предварительную запись программ удобно проводить в более компактной форме. В программе указывается начальный адрес каждой команды и при этом понимается, что в зависимости от длины команды в памяти будут занимать от одной до трех последовательных ячеек. При такой записи в левом столбце указываются лишь адреса команд в программе. Это позволяет сократить объем при описании программ и сделать более простым их анализ.

Программа 1 (общий вид записи)

```
Адрес     Машинный код    Мнемокод       Комментарий
0800      3A 00 09        LDA 0900 ;читать число из памяти
                               ;по адресу 0900H
0803      2F              CMA      ; инвертировать число
0804      32 01 09        STA 0901  ; записать число в память
                                   ;по адресу 0901H
0807      76              HLT      ; останов
```

Порядок выполнения задания:

1. Ввести в УМК программу 1.
2. Записать по адресу 0900 исследуемое число.

3. Осуществить пуск программы. Последовательно нажать клавиши «СБ», «СТ», ввести адрес 0800 и нажать клавишу «ВП». Выждать 1–2 с и нажать клавишу «ПР», при этом на адресном дисплее отобразится адрес ячейки, следующий за командой HLT.

4. Проверить результат выполнения программы путем считывания числа из ячейки 0901.

5. Исследовать процесс выполнения программы по командам.

Клавишу «РБ/ШГ» установить в нажатое положение. Осуществить пуск программы и, последовательно нажимая клавишу «ШГ», выполнить программу по командам. После выполнения каждой команды проанализировать показания индикаторов «ША», «ШД» и регистров состояния.

6. Исследовать процесс выполнения команд в программе 1 по машинным циклам. Обратить внимание на последовательность передачи и преобразования информации в микроЭВМ при выполнении каждой команды.

Программа 2

```
Адрес   Машинный код  Мнемокод         Комментарий
0800      21 00 09  LXI H, 0900 ;записать в регистры H,L
                                ;число 0900H
0803      7E        MOV A,M     ;получить число из памяти,
                                ;по адресу, указанному в
                                ;регистровой паре H,L
0804      2F        CMA         ;инвертировать число в
                                ;аккумуляторе
0805      2З        INX  H      ;увеличить на 1 число в
                                ;регистрах  H,L
0806      77        MOV  M,A    ;записать число из аккуму-
                                ;лятора по адресу, указан-
                                ;ному в регистрах  H, L
0807      76        HLT         ; останов
```

Порядок выполнения:

1. Ввести в УМК программу 2.

2. Записать по адресу 0900 исследуемое число.

3. Осуществить пуск программы с адреса 0800. Проверить результат выполнения программы по числу, записанному по адресу 0901.

4. Исследовать процесс выполнения команды MOV A,M по машинным циклам.

4. Содержание отчета

1. Цель работы.
2. Таблицы пошагового выполнения программ.
3. Временные диаграммы выполнения команд.
4. Ассемблерные записи 2-х программ с комментариями.
5. Ответы на вопросы.
6. Выводы.

Лабораторная работа № 3
СИСТЕМА КОМАНД МИКРОПРОЦЕССОРА INTEL 8080. РАЗРАБОТКА ПРОСТЫХ ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ КОМАНД ПЕРЕДАЧИ ДАННЫХ

Цели работы: исследование выполнения отдельных команд передачи данных и простых программ; исследование процесса выполнения команд по машинным циклам; самостоятельная разработка программ на языке ассемблер.

1. Краткие сведения из теории

Для разработки программ необходимо знание архитектуры микропроцессора и микропроцессорной системы. Архитектура отражает возможности прикладного использования микропроцессора и содержит описание программной модели и системы команд, обеспечивающих доступ к элементам модели. Программная модель системы на основе МП Intel 8080 (рис. 1) содержит непосредственно модель МП, модель памяти и модель портов ввода/вывода.

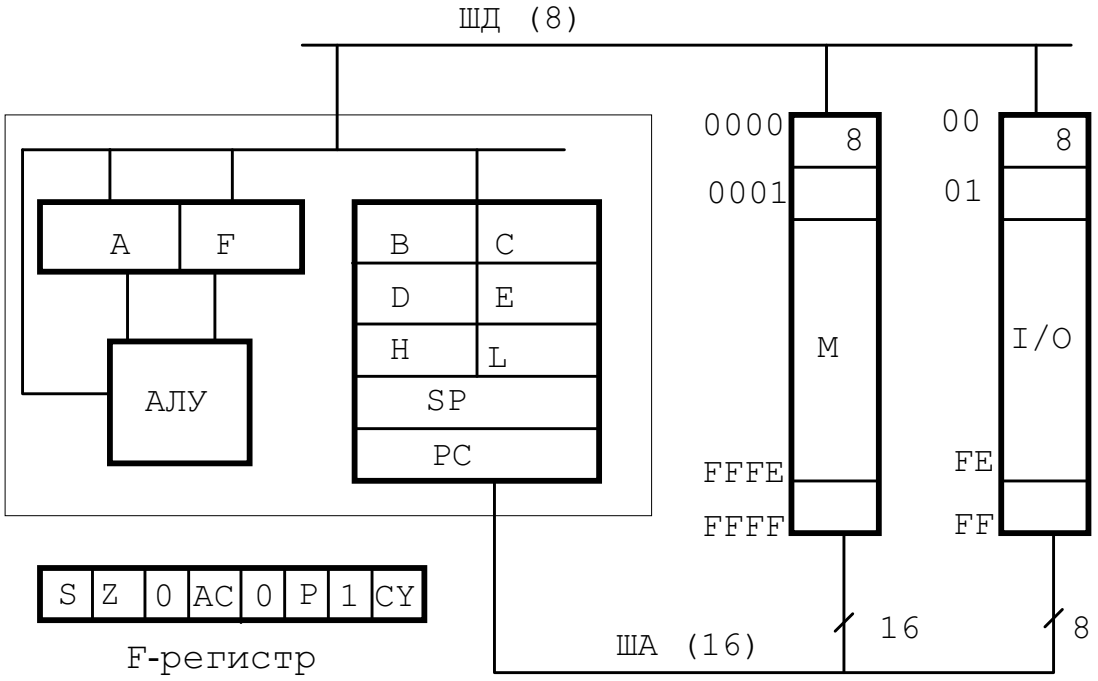

Рис. 1. Программная модель микропроцессорной системы на основе Intel 8080

Модель МП содержит следующие элементы:

1) шесть 8-разрядных регистров (B, C, D, E, H, L), которые могут использоваться автономно и попарно, как три 16-разрядных регистра (BC,

DE, HL). Регистры и регистровые пары используются для временного хранения данных, адресов памяти и косвенной адресации основной памяти (в качестве указателя памяти). Регистровая пара HL преимущественно используется как указатель памяти в коротких однобайтных командах, причем ячейка памяти, адрес которой содержится в HL-паре, называется М-регистром. В функциональном отношении этот регистр эквивалентен регистрам МП;

2) 16-разрядный регистр-указатель стека (SP). Используется для указания адреса вершины стека, который размещен в памяти;

3) 8-разрядный аккумулятор (А-регистр) – основной рабочий регистр МП, который используется АЛУ для размещения одного из операндов (данное участвующее в операции) и результата выполнения операции. Совместно с F-регистром образует регистровую пару, которая носит название «слово состояния» программы (PSW);

4) 8-разрядный F-регистр (регистр признаков, или флагов) – отражающий некоторые особенности выполнения операций АЛУ. F-регистр фиксирует 5 различных признаков (3 разряда не используются):

S – бит знака, равен 1, если число отрицательное (совпадает по значению со старшим (седьмым) битом аккумулятора);

Z – бит нуля, равен 1, если результат операции равен нулю;

P – бит паритета, равен 1, если число единиц результата четное;

CY – бит переноса, равен 1, если при выполнении операции произошел перенос (или заем) из старшего разряда АЛУ. Признак очень важен при выполнении операций с данными увеличенной разрядности;

AC – бит вспомогательного переноса, равен 1, если при выполнении операции произошел перенос из младшей тетрады (4 двоичных разряда) в старшую. Используется командой десятичной коррекции содержимого аккумулятора.

Модель памяти представляет собой упорядоченную и пронумерованную последовательность 8-разрядных структурных элементов – ячеек памяти. Номер ячейки является ее адресом. Шина адреса МП является 16-разрядной, поэтому максимальное число адресуемых ячеек равно 2^{16} (65536 = 64 К). Адрес любой ячейки памяти представляется 4-разрядным шестнадцатеричным числом (от 0000H до 0FFFFH), а ее содержимое – это 2-разрядное шестнадцатеричное число. Над каждой ячейкой памяти могут быть выполнены операции записи байта в ячейку и чтения содержимого ячейки. Память такого типа называют ОЗУ. Эта память используется для хранения программ, исходных, промежуточных и результирующих данных.

В реальных МП-системах фактическое число ячеек памяти может быть меньше адресного пространства и, кроме того, разделено на части

по конструктивным признакам. В нашем лабораторном макете, например, используются только первые 4 К адресного пространства, и первые 2 К – это постоянное ЗУ.

Модель портов ввода/вывода представляет собой также упорядоченную пронумерованную последовательность 8-разрядных регистров. Эти регистры адресуются по 8 младшим линиям шины адреса, поэтому максимальное число портов – 256. Внешние устройства обмениваются информацией с МП через соответствующий порт ввода или вывода.

Система команд – набор элементарных операций, выполняемых МП. Набор команд МП Intel 8080 содержит 78 базовых команд, имеющих 244 модификации. Функциональные возможности системы команд определяются форматом обрабатываемых данных, форматом команд и способами адресации операндов в командах. Малая разрядность регистров МП приводит к необходимости использования различных способов непрямой адресации памяти, сокращающих длину команды и размеры программ. В МП Intel 8080 используются следующие способы адресации операндов.

Прямая адресация – во втором и третьем байтах команды содержится адрес памяти, по которому находится операнд.

Регистровая адресация – операнд в одном из регистров МП и команда указывает на этот регистр.

Косвенная регистровая адресация – адрес операнда находится в регистровой паре, на которую указывает команда.

Непосредственная адресация – операнд размещен непосредственно в команде (во втором или во втором и третьем байте команды).

Стековая адресация – адрес памяти определяется регистром-указателем стека.

Формат команды – одно-, двух- или трехбайтовое число. Первый байт команды представляет собой код операции, второй и третий – данные или адрес. Команды с регистровым, косвенным регистровым и стековым методом адресации – однобайтовые; с прямым – трехбайтовые; с непосредственным – двух- или трехбайтовые. В трехбайтовых командах действует принцип – младший байт по младшему адресу, т. е. в коде команды сначала идет младший байт адреса, потом старший.

Все команды МП Intel 8080 подразделяются на 5 функционально специализированных групп:

команды передачи данных;

команды арифметических операций;

команды логических операций;

команды передачи управления;

команды стека, ввода/вывода и управления.

1.1. Команды передачи данных

Такие команды обеспечивают выполнение операций размещения, обмена, загрузки и перемещения данных.

1. MOV R1, R2. Передача данных из регистра R2 в R1. Однобайтовая команда с регистровым способом адресации. В качестве R1(R2) могут выступать регистры общего назначения (B, C, D, E, H, L), аккумулятор (A) и ячейка памяти (M). В последнем случае адрес ячейки должен находиться в регистровой паре HL.

Формат команды:

01DDDSSS ,

где **DDD** – код регистра приемника; **SSS** – код регистра источника (табл. 1).

Примеры: MOV B,D (копирование содержимого регистра D в регистр B); MOV A,M (копирование содержимого ячейки памяти в аккумулятор, адрес ячейки находится в регистровой паре HL).

2. MVI R, data. Передача в регистр непосредственных данных. Двухбайтовая команда с непосредственным способом адресации. Регистр R определяется, как и в предыдущем случае.

Формат команды:

00DDD110 КОП (байт кода операции);

XXXXXXXX Данные (1 байт).

Примеры: MVI H, 0A7H (загрузка непосредственного числа 0A7H в регистр H); MVI M, 0FH (загрузка непосредственного числа 0FH в ячейку памяти, адрес ячейки в регистровой паре HL).

Таблица 1

Регистр	DDD(SSS)
A	111
B	000
C	001
D	010
E	011
H	100
L	101
M	110

3. LXI RP, data16. Непосредственная загрузка регистровой пары. Трехбайтовая команда с непосредственным способом адресации. Загружает два байта в регистровую пару BC, DE, HL или SP.

Формат команды:

00RP0001 КОП;

XXXXXXXX младший байт данных;

XXXXXXXX старший байт данных.

Таблица 2

Регистровая пара	RP
BC	00
DE	01
HL	10
SP	11

4. LDA addr. Прямая загрузка аккумулятора.

STA addr. Прямое запоминание аккумулятора.

Команда загружает из памяти данные в аккумулятор (из аккумулятора в память), причем адрес ячейки памяти берется прямо из команды.

Форматы команд:

LDA	STA	
00111010	**00110010**	КОП;
xxxxxxxx	**xxxxxxxx**	младший байт адреса;
xxxxxxxx	**xxxxxxxx**	старший байт адреса.

5. LHLD addr. Прямая загрузка регистровой пары HL.

SHLD addr. Прямое запоминание регистровой пары HL.

Команда загружает из памяти данные в регистровую пару HL (из регистровой пары HL в память), причем адрес ячейки памяти берется прямо из команды.

Форматы команд:

LHLD	SHLD	
00101010	**00100010**	КОП;
xxxxxxxx	**xxxxxxxx**	младший байт адреса;
xxxxxxxx	**xxxxxxxx**	старший байт адреса.

6. LDAX RP. Косвенная загрузка аккумулятора.

STAX RP. Косвенное запоминание аккумулятора.

Команда загружает из памяти данные в аккумулятор (из аккумулятора в память), причем адрес ячейки памяти берется из регистровой пары BC или DE.

Форматы команд:

LDAX	STAX
00RP1010	**00RP0010**

7. XCHG. Обмен содержимого регистровых пар DE и HL.

Формат команды:

11101011

1.2. Команды передачи управления

Команды используются для создания разветвляющихся и циклических программ. В качестве проверяемого условия выступают значения битов флагового регистра.

1. JMP *adress*. Передача управления команде, находящейся по адресу *adress*.

Формат команды:

11000011 КОП;

XXXXXXXX младший байт адреса;

XXXXXXXX старший байт адреса.

2. Jусл *adress*. Передача управления команде, находящейся по адресу *address*, если выполнено условие, в противном случае переход к следующей по порядку команде.

Формат команды:

11ccc010 КОП (ссс – код условия из табл. 3);

XXXXXXXX младший байт адреса;

XXXXXXXX старший байт адреса.

Таблица 3

Код условия	Значение ссс	Комментарий
NZ	000	переход, если не нуль
Z	001	переход, если нуль
NC	010	переход, если не установлен перенос
C	011	переход, если установлен перенос
PO	100	переход, если нечетно
PE	101	переход, если четно
P	110	переход, если знак плюс
M	111	переход, если знак минус

3. CALL *adress*. Передача управления подпрограмме, находящейся по адресу *adress*. Содержимое программного счетчика записывается в стек, а *adress* записывается в программный счетчик.

Формат команды:

11001101 КОП;

XXXXXXXX младший байт адреса;

XXXXXXXX старший байт адреса.

4. Сусл *adress*. Передача управления подпрограмме, находящейся по адресу *adress* (содержимое программного счетчика записывается в стек, а *adress* записывается в программный счетчик), если выполнено условие, в противном случае – переход к следующей по порядку команде.

Формат команды:

11ccc100 КОП (ссс – код условия из табл. 3);

XXXXXXXX младший байт адреса;

XXXXXXXX старший байт адреса.

5. RET. Возврат из подпрограммы. Из стека в программный счетчик записывается значение адреса основной программы.

Формат команды:

11001001 КОП;

XXXXXXXX младший байт адреса;

XXXXXXXX старший байт адреса.

6. Rусл *adress*. Возврат из подпрограммы, если выполнено условие, в противном случае переход к следующей по порядку команде.

Формат команды:

11ccc000 КОП (ссс – код условия);

XXXXXXXX младший байт адреса;

XXXXXXXX старший байт адреса.

В данной работе мы будем использовать 4 команды из группы арифметических команд:

1) INR R. Инкремент (увеличение на 1) содержимого регистра R.

DCR R. Декремент (уменьшение на 1) содержимого регистра R. Однобайтовые команды с регистровым способом адресации. В качестве R могут выступать регистры общего назначения (B, C, D, E, H, L), аккумулятор (A) и ячейка памяти (M). В последнем случае адрес ячейки должен находиться в регистровой паре HL. Содержимое регистра R увеличивается (уменьшается) на 1 и результат помещается в тот же регистр. Команды влияют на все биты флагового регистра микропроцессора за исключением бита переноса, который не изменяется.

Формат команды:

00SSS10D,

где SSS – код регистра; D = 0 – инкремент; D = 1 – декремент.

2) INX RR. Инкремент содержимого регистровой пары.

DCX RR. Декремент содержимого регистровой пары. Однобайтовые команды с регистровым способом адресации. В качестве операнда используются регистровые пары BC, DE, HL и SP. На состояние битов флагового регистра команды не влияют.

Формат команды:

00RRD011,

где RR – код регистровой пары; D = 0 – инкремент; D = 1 – декремент.

2. Исследование выполнения программ

Рассмотрим программу, записывающую число 0A0H в память по адресу 0900H.

Программа 1 (общий вид записи)

```
Адрес  Машинный код  Мнемокод      Комментарий
0800     3E A0       MVI A,0A0; загрузить число в аккумулятор
0802     32 00 09    STA 0900 ; записать число по адресу 0900
0805     76          HLT      ; останов
```

Порядок выполнения:

1. Ввести в УМК программу 1.

2. Записать по адресу 0900 любое число.

3. Осуществить пуск программы. Последовательно нажать клавиши «СБ», «СТ», ввести адрес 0800 и нажать клавишу «ВП». Выждать 1–2 с и нажать клавишу «ПР», при этом на адресном дисплее отобразится адрес ячейки, следующий за командой HLT.

4. Проверить результат выполнения программы путем считывания числа из ячейки 0900.

В приведенной выше программе 1 используется прямой способ адресации.

Программа 2 (с использованием косвенного способа адресации).

```
Адрес   Машинный код  Мнемокод     Комментарий
0800     21 00 09    LXI H,0900; записать в регистры H,L
                          ; число 0900
0803     3E A0       MVI A,0A0 ; записать непосредственное
                          ; число 0A0 в аккумулятор
0805      77         MOV  M,A  ; записать число из аккуму-
                          ; лятора по адресу, указан-
                          ; ному в регистрах, H и L
0806      76         HLT       ; останов
```

Порядок выполнения:

1. Ввести в УМК программу 2.

2. Записать по адресу 0900 любое число.

3. Осуществить пуск программы с адреса 0800. Проверить результат выполнения программы по числу, записанному по адресу 0900.

4. Исследовать процесс выполнения программы по машинным циклам.

5. В точках останова исследовать содержимое аккумулятора, регистров H, L и ячейки памяти 0900.

Программа 3. Программа формирования в памяти, начиная с адреса 0900H, массива из 10 последовательных чисел.

```
Адрес    Машинный код  Мнемокод     Комментарий
0800      0E 14        MVI C,0AH  ;загрузка в регистр С коли-
                                  ;чества чисел
0802      3E 01        MVI A,01H  ;загрузка в А первого числа
0804      21 00 09     LXI H,0900H;в пару HL – адрес памяти
0807      7E           MOV A,M    ;содержимое А копируется в
                                  ;память
0808      3C           INR A      ;инкремент аккумулятора
0809      23           INX H      ;инкремент указателя памяти
080A      0D           DCR C      ;декремент счетчика
080B      C2 07 08     JNZ0807    ;возврат к шагу 0807
080E      76           HLT        ;останов
```

Порядок выполнения:

1. Ввести в УМК программу 3.

2. Осуществить пуск программы с адреса 0800. Проверить результат выполнения программы по числам, записанным по адресу 0900 и далее.

3. Исследовать процесс выполнения программы по машинным циклам.

4. В некоторых точках исследовать содержимое аккумулятора, регистров H, L и ячеек памяти 0900.

3. Задание

Используя команды передачи данных, составить программу пересылки данных из ячеек 0900 и 0901 по адресам 0910 и 0911, занести ее в УМК и проверить ее выполнение. Составить программу пересылки массива из 10 байтов, расположенных в памяти, последовательно начиная с адреса 0900H, в 10 ячеек, начиная с адреса 09A0H, причем данное в исходном массиве, находящееся по младшему адресу в полученном массиве, должно располагаться по старшему адресу.

4. Содержание отчета

1. Программная модель МП Intel 8080.
2. Команды передачи данных.
3. Ассемблерные записи программ с комментариями.
4. Пошаговый разбор работы программ.

Лабораторная работа № 4
АРИФМЕТИЧЕСКИЕ КОМАНДЫ МИКРОПРОЦЕССОРА *INTEL* 8080. ВЫПОЛНЕНИЕ ПРОГРАММ АРИФМЕТИЧЕСКИХ ВЫЧИСЛЕНИЙ

Цели работы: исследование выполнения отдельных команд и простых программ арифметических вычислений; самостоятельная запись и выполнение несложных программ.

1. Краткие сведения из теории

1.1. Команды арифметических операций

Команды арифметических операций обеспечивают выполнение двоичных операций сложения, вычитания, инкрементирования и декрементирования содержимого регистра (регистровой пары) или ячейки памяти, а также для сложения чисел в двоично-десятичном формате. Все команды сложения и вычитания используют в качестве одного из операндов аккумулятор и для выполнения действий используют арифметико-логическое устройство (АЛУ) микропроцессора. Поэтому все арифметические команды влияют на состояние разрядов флагового регистра микропроцессора. Исключение составляют команды инкремента регистровых пар и сложения регистровых пар, которые не используют АЛУ и на состояние флагов не влияют.

а) ADD R. Сложение данных из аккумулятора и регистра R.

ADC R. Сложение содержимого аккумулятора и регистра R с учетом бита переноса (флага переноса).

Однобайтовые команды с регистровым способом адресации. В качестве R могут выступать регистры общего назначения (B, C, D, E, H, L), аккумулятор (A) и ячейка памяти (M). В последнем случае адрес ячейки должен находиться в регистровой паре HL. В качестве второго операнда используется аккумулятор, туда же пересылается и результат операции.

Формат команды:

`1000CSSS`,

где SSS – код регистра источника (см. табл. 1 в работе 3); C = 0 – перенос не используется; C = 1 – перенос используется.

б) ADI data. Сложение содержимого аккумулятора и непосредственных данных.

ACI data. Сложение непосредственных данных с учетом бита переноса.

Двухбайтовые команды с непосредственным способом адресации. В качестве второго операнда и приемника результата также используется аккумулятор.

Формат команды:

1100C110 КОП;

XXXXXXXX Данные (1 байт).

Значение бита C определяется, как в предыдущей команде.

в) SUB R. Вычитание из аккумулятора содержимого регистра R.

SBB R. Вычитание из аккумулятора содержимого регистра R с учетом бита переноса (вычитание с заемом).

Однобайтовые команды с регистровым способом адресации. В качестве R могут выступать регистры общего назначения (B, C, D, E, H, L), аккумулятор (A) и ячейка памяти (M). В последнем случае адрес ячейки должен находиться в регистровой паре HL. В качестве второго операнда используется аккумулятор, туда же пересылается и результат операции.

Формат команды:

1001CSSS,

где SSS – код регистра источника; C = 0 – перенос не используется; C = 1 – перенос используется.

г) SUI data. Вычитание из содержимого аккумулятора непосредственных данных.

SBI data. Вычитание непосредственных данных с учетом бита переноса (заема).

Двухбайтовые команды с непосредственным способом адресации. В качестве второго операнда и приемника результата также используется аккумулятор.

Формат команды:

1101C110 КОП;

XXXXXXXX Данные (1 байт). Значение бита C определяется, как и в предыдущем случае.

д) INR R. Инкремент (увеличение на 1) содержимого регистра R.

DCR R. Декремент (уменьшение на 1) содержимого регистра R.

Однобайтовые команды с регистровым способом адресации. В качестве R могут выступать регистры общего назначения (B, C, D, E, H, L), аккумулятор (A) и ячейка памяти (M). Содержимое регистра R увеличивается (уменьшается) на 1, и результат помещается в тот же регистр. Команды влияют на все биты флагового регистра микропроцессора за исключением бита переноса, который не изменяется.

Формат команды:

00SSS10D,

где SSS – код регистра; D = 0 – инкремент; D = 1 – декремент.

е) INX RR. Инкремент содержимого регистровой пары.

DCX RR. Декремент содержимого регистровой пары.

Однобайтовые команды с регистровым способом адресации. В качестве операнда используются регистровые пары BC, DE, HL и SP. На состояние битов флагового регистра команды не влияют.

Формат команды:

00RRD011,

где RR – код регистровой пары; D = 0 – инкремент; D = 1 – декремент.

ж) DAD RR. Сложение содержимого регистровой пары HL и пары RR.

Однобайтовая команда с регистровым способом адресации. В качестве RR могут выступать регистровые пары BC,DE,HL,SP. Вторым операндом и приемником результата является регистровая пара HL. Команда не влияет на биты флагового регистра за исключением бита переноса, который может устанавливаться по результату операции.

Формат команды:

00RR1001,

где RR – код регистровой пары.

з) DAA. Десятичная коррекция содержимого аккумулятора.

Команда осуществляет коррекцию результата сложения двух чисел в двоично-десятичном коде таким образом, что в аккумуляторе тоже образуется число в двоично-десятичном коде. Команда использует биты переноса и дополнительного переноса и влияет на все флаги.

Формат команды:

00100111.

1.2. Команды сдвигов

Это команды, сдвигающие содержимое аккумулятора вправо или влево на один разряд. При этом сдвиг может происходить через бит переноса регистра флагов. Команды не влияют на другие флаги регистра признаков.

а) RLC. Циклический сдвиг влево содержимого аккумулятора.

RRC. Циклический сдвиг вправо содержимого аккумулятора.

Однобайтовые одноопернадные команды с неявным способом адресации. В качестве операнда и приемника результата используется аккумулятор.

Формат команды:

0000C111,

где C – направление сдвига (0 – влево, 1 – вправо).

б) RAL. Циклический сдвиг влево содержимого аккумулятора через перенос.

RAR. Циклический сдвиг вправо содержимого аккумулятора через перенос.

Однобайтовые одноoперандные команды с неявным способом адресации. В качестве операнда и приемника результата используется аккумулятор. В сдвиге как дополнительный бит участвует флаг переноса.

Формат команды:

000C1111,

где С – направление сдвига (0 – влево, 1 – вправо).

2. Порядок работы

2.1. Ввести в УМК с адреса 0800 программу, складывающую три числа, расположенных по адресам 0900, 0901, 0902, прибавляющую к результату число 10 (десятичное) и помещающую результат в ячейку 0903.

```
LXI H,0900 ;загрузка в пару HL адреса
MOV A,M    ;первое число в аккумулятор
INX H      ;инкремент указателя адреса
ADD M      ;сложить первое и второе число
INX H      ;инкремент указателя адреса
ADD M      ;прибавить третье число
ADI 0AH    ;прибавить число 10
INX H      ;инкремент указателя
MOV M,A    ;полученную сумму переслать в память
```

Занести по адресам 0900, 0901 и 0902 числа и выполнить программу. Проверить полученный результат.

2.2. Изменить программу, чтобы из суммы первых двух чисел вычиталось третье и десятичное число 20. Выполнить эту программу и проверить результат.

2.3. В предыдущих программах, если полученные результаты больше 256, т. е. не могут быть представлены однобайтовым числом, старшие биты результата теряются. В этом случае необходимо результат представлять двухбайтовым числом и использовать команды, учитывающие перенос или заем.

Введите в УМК следующую программу-пример и выполните ее. Сравните результат с программой 1.

Программа сложения трех чисел и числа 10 с учетом переноса

```
     LXI H,0900 ;загрузка в пару HL адреса
     MVI C,00H  ;обнулить регистр C
     MOV A,M   ;первое число в аккумулятор
     INX H     ;инкремент указателя адреса
     ADD M     ;сложить первое и второе число
     JNC L1    ;переход к метке L1, если нет переноса
     INR C     ;если перенос увеличить на 1 регистр C
L1:      INX H          ;инкремент указателя адреса
     ADD M     ;прибавить третье число
     JNC L2    ;переход к метке L1, если нет переноса
     INR C     ;если перенос увеличить на 1 регистр C
```

```
L2:  ADI 0AH    ;прибавить число 10
     JNC L3     ;переход к метке L1, если нет переноса
     INR C      ;если перенос увеличить на 1 регистр C
L3:  INX H      ;инкремент указателя
     MOV M,A    ;полученную сумму переслать в память
     INX H
     MOV M,C
```

2.4. Составить программу суммирования 10 чисел, расположенных в памяти, начиная с адреса 0900. Использовать циклическую процедуру суммирования, в качестве счетчика взять регистр C и зацикливание программы производить по ненулевому результату в регистре C. Результат поместить в ячейках 0910 (младший байт) и 0911 (старший байт). Выполнить программу и проверить результат.

2.5. Составить программу суммирования 10 двухбайтовых чисел, расположенных в памяти с адреса 0900. Выполнить ее и проверить результат.

2.6. Составить и выполнить программу сложения двух четырехразрядных десятичных чисел, представленных в двоично-десятичном коде. Исходные числа и результат разместить в памяти.

2.7. Составить и выполнить программу умножения двух однобайтовых чисел.

3. Содержание отчета

1. Ассемблерные записи программ с комментариями.
2. Пошаговый разбор работы программы 1.

Лабораторная работа № 5
ВЫВОД ИНФОРМАЦИИ ИЗ МИКРОПРОЦЕССОРА INTEL 8080 НА ВНЕШНЕЕ УСТРОЙСТВО

Цели работы: исследование выполнения программ вывода информации на светодиодный дисплей с использованием микросхемы контроллера параллельного интерфейса; самостоятельная разработка и выполнение программ.

1. Краткие сведения из теории

В работе используется светодиодный дисплей и средства управления им из комплекта УМК. Схема управления дисплеем и клавиатурой УМК приведена на рис. 1. Основа схемы – микросхема контроллера параллельного интерфейса Intel 8255 (российский аналог – 580ВВ55).

Микросхема программируемого периферийного параллельного адаптера (ППА) 8255 (рис. 2) предназначена для применения в МП-системе в качестве универсального элемента ввода/вывода, обеспечивающего обмен данными в параллельном формате между МП и системными, в частности периферийными, устройствами.

Микросхема подключается к МП-системе посредством двунаправленной 8-разрядной шины данных с тремя состояниями выхода ШД; 2-разрядной шины адреса ША – A0,A1; 4-разрядной шины управления ШУ, включающей сигналы: WR – запись, RD – чтение, CS – выбор микросхемы и SR – сброс. Адаптер включает три программно-доступных 8-разрядных порта ввода/вывода (PA, PB, PC) и 7-разрядный регистр управляющих слов РУС, содержимое которого определяет направление передачи и функциональное назначение 24 двунаправленных линий ввода/вывода, т. е. конфигурацию и режимы работы портов. Порты PA и PB предназначены для обмена байтами данных с системными устройствами, порт PC, как правило, – для обмена интерфейсными сигналами управления. Порт PC в отличие от портов PA и PB программно-доступен при операциях записи данных не только как элемент в целом, но и поразрядно, т. е. с независимой адресацией каждого отдельного разряда PC_i (i = 0,1...7), а в операциях выбора режима – как два полупорта: старший – PCH, младший – PCL или их части. Обмен данными между элементами ППА- и МП-системой происходит через внутреннюю шину, связанную с ШД, и через устройство управления (УУ) с шинами ША и ШУ.

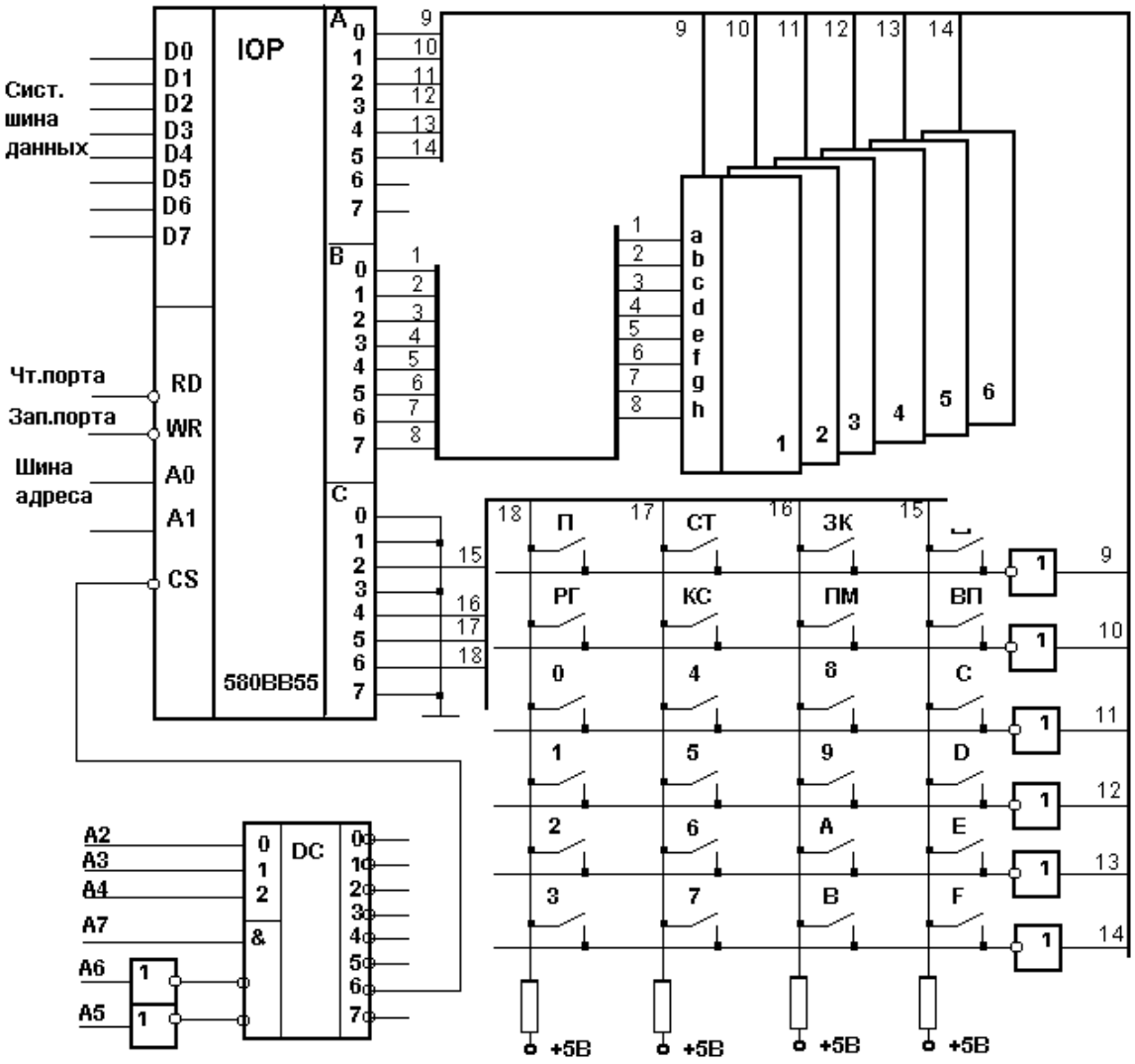

Рис. 1. Дисплей и клавиатура УМК

В МП-системе, содержащей ППА, возможны два типа операций над его элементами: чтение (ввод) в МП-содержимого адресуемого элемента и запись (вывод) из МП-байта данных в адресуемый элемент ППА. Эти операции выполняются программно с помощью двух команд МП: IN port и OUT port, где port – системный адрес конкретного порта ППА. В процессе выполнения указанных команд в МП-системе формируются сигналы управления, комбинация которых определяет ту или иную операцию над элементами ППА (табл. 1). Два разряда адреса (они не обязательно должны совпадать с одноименными разрядами шины адреса МП) определяют выбор одного из трех портов или регистра. Заметим, что если для каждого порта существует пара операций чтение–запись, то для регистра РУС отсутствует операция чтения.

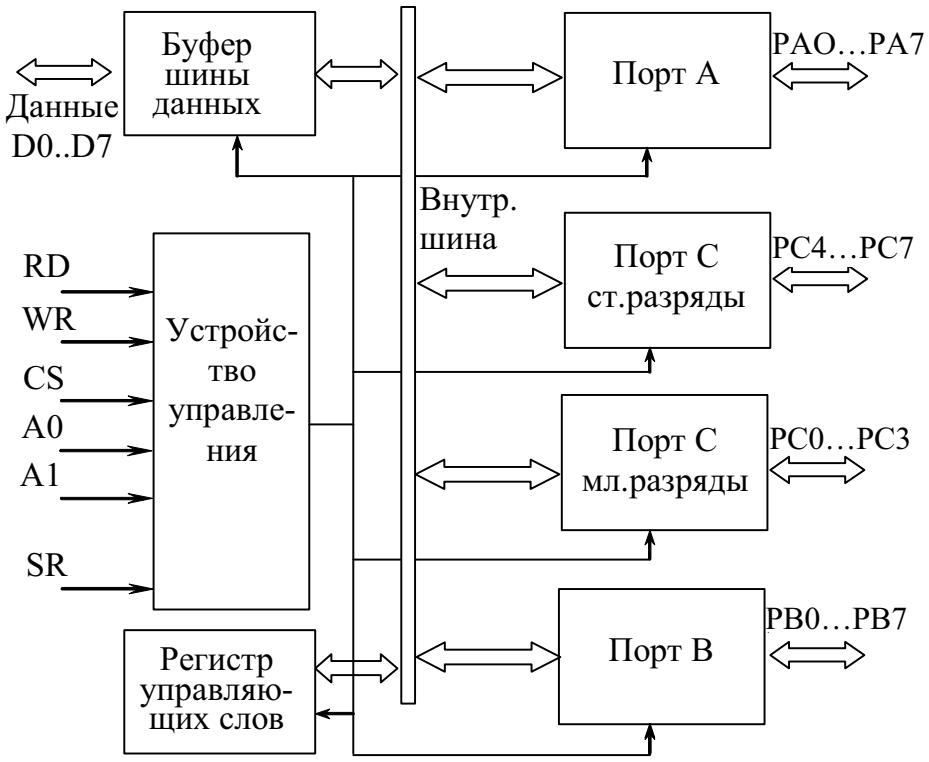

Рис. 2. Блок-схема БИС 8255

Программирование ППА, или его настройка, осуществляются с помощью операции записи управляющего слова (УС) в регистр РУС. Возможны два типа настройки.

1. Настройка разрядов – поразрядное программирование порта ПС. УС для этого типа программирования имеет разряд Д7 = 0.

2. Настройка режимов – программирование портов РА, РВ, РС на один из трех возможных режимов или их комбинацию: 0 – режим простого однонаправленного обмена; 1 – режим стробируемого однонаправленного обмена; 2 – режим стробируемого двунаправленного обмена. Для настройки режимов используется УС, в котором разряд Д7 = 1. Значение остальных разрядов приведено в табл. 2.

Когда на микросхему подается сигнал сброса, все порты устанавливаются в режим ввода данных и все 24 выхода портов переходят в состояние 1. После сигнала сброса все порты остаются в режиме простого ввода. Если микросхема должна использоваться именно в этих режимах, то дополнительного программирования не требуется. Если требуются другие режимы, то необходимо записать в регистр управляющих слов соответствующее слово инициализации. Перепрограммирование режимов работы микросхемы может быть сделано в любой момент работы программы.

Режим работы портов можно изменять как до, так и в процессе работы системных программ, что позволяет в определенном порядке обслуживать различные ПУ.

Таблица 1

Сигналы управления и операции ППА

Тип операции	Направление передачи	A1	A0	RD	WR	CS
Чтение	ШД ← РА	0	0	0	1	0
	ШД ← РВ	0	1	0	1	0
	ШД ← РС	1	0	0	1	0
Запись	ШД → РА	0	0	1	0	0
	ШД → РВ	0	1	1	0	0
	ШД → РС	1	0	1	0	0
	ШД → РУС	1	1	1	0	0
Отключено от ШД		×	×	×	×	1
Нет действия		1	1	0	1	0

Таблица 2

Назначение разрядов управляющего слова ППА

Разряд	Назначение разряда
D0	Режим работы линий PC0 – PC3 (1 – ввод; 0 – вывод)
D1	1 – ввод; 0 – вывод линий порта PB
D2	
D3	Режим работы линий PC4 – PC7 (1 – ввод; 0 – вывод)
D4	1 – ввод; 0 – вывод линий порта PA
D5, D6	Режим работы порта PB (0 0 – режим 0, 0 1 – режим 1, 1 × – режим 2)
D7	Признак управляющего слова

2. Порядок работы

2.1. Ввести в УМК с адреса 0800 программу начального программирования ППА и вывода на один из светодиодных индикаторов знака, представленного двоичным однобайтовым числом.

```
      MVI   A,89  ; Начальное программирование ППА (89 –
      OUT   RUS   ; управляющее слово, RUS – адрес порта РУС)
      MVI   A,1   ; В A заносится номер индикатора
      OUT   PORTA ; и передается в порт A ППА
      MVI   A,SIGN; В A заносится отображаемый знак
      OUT   PORTB ; и передается в порт B
      MVI   E,FF  ; Начало программы создания
L2:   MVI   D,FF  ; временной задержки
L1:   NOP
```

```
        DCR   D     ; Задержка нужна, чтобы мы успели
        JNZ   L1    ; увидеть действие основной
        DCR   E     ; программы
        JNZ   L2    ;
        RST   7     ; Конец программы
```

При вводе программы в УМК символическим именам нужно придать определенные значения (PORTA, PORTB, RUS – адреса портов ППА в УМК; SIGN – код символа, который должен быть отображен). Для определения адресов портов нужно воспользоваться рис. 1, а для определения кода символа рис. 3.

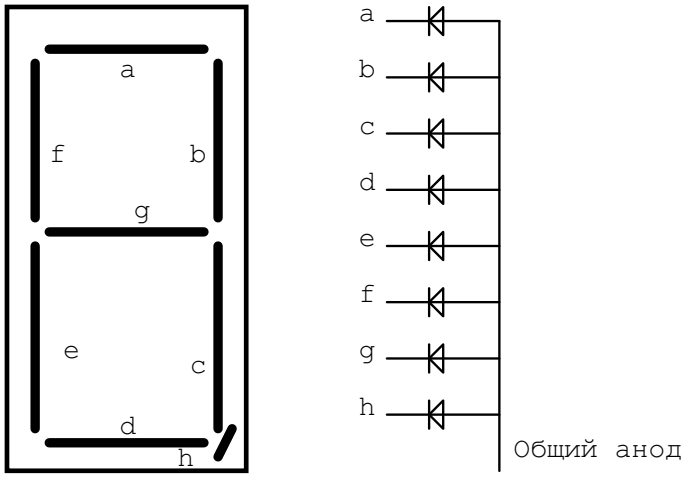

Рис. 3. Семисегментный светодиодный индикатор

Выполнить программу. Проверить полученный результат.

2.2. Изменить программу так, чтобы на другом индикаторе загоралось другое число. Выполнить эту программу.

2.3. Ввести программу динамической индикации (все 6 индикаторов горят попеременно, причем в каждый момент времени горит только один и одноименные сегменты индикаторов объединены; так как переключение происходит с большой частотой, внешне кажется, что все индикаторы горят одновременно и отображают разную информацию) блока данных из 6 байт, расположенных начиная с адреса 0900.

```
        MVI A,89
        OUT RUS
        LXI D,8FFFH  ; DE пара используется для создания
                     ; временной задержки
L1:     LXI H,0900H  ; загрузка в пару HL начального адреса
                     ; блока отображаемых чисел
L2:     MVI B,20H    ; В регистр В начальный номер индикатора
        MOV A,B      ; Номер индикатора – в аккумулятор
        OUT PORTA    ; и в порт А ППА
        MOV A,M      ; В А данное из памяти (HL)
        OUT PORTB    ; Вывод в порт В
```

```
NOP
MVI A,00H     ; Гашение горящего
OUT PORTB     ; индикатора
INX H         ; Переход к следующему данному
MOV A,B       ; Сдвиг
RRC           ; номера индикатора
MOV B,A       ; вправо
JNC L2        ; К началу индикации следующего данного
DCX D         ; Временная задержка
MOV A,D
ORA E
JNZ L1
RST7
```

Используя программу, получить на индикаторах УМК числа 0, 1, 2, 3, 4, 5.

2.4. Сделать отчет.

3. Содержание отчета

1. Структурная схема управления светодиодным дисплеем и адреса портов микросхемы ППА.
2. Ассемблерные записи программ с комментариями.

Лабораторная работа № 6
ПОДГОТОВКА ПРОГРАММ В СИСТЕМЕ ПРОГРАММИРОВАНИЯ MASM

Цели работы: рассмотреть процесс подготовки программ для персонального компьютера на основе микропроцессора семейства 8086 на языке ассемблера MASM под управлением операционной системы MS DOS; показать взаимодействие программы с операционной системой через прерывания DOS.

1. Система программирования MASM

Система программирования представляет собой пакет программ для подготовки и исполнения программ на языке ассемблера на компьютерах с системой команд микропроцессора 8086.

Поясним структуру программы, рассмотрев процесс ассемблирования.

Как видно из рис. 1, входом для ассемблера является исходный модуль (по умолчанию, должен иметь расширение ASM), подготовленный любым редактором текста. Выходом являются объектный модуль (OBJ), листинг (LST) и файл перекрестных ссылок (CRF).

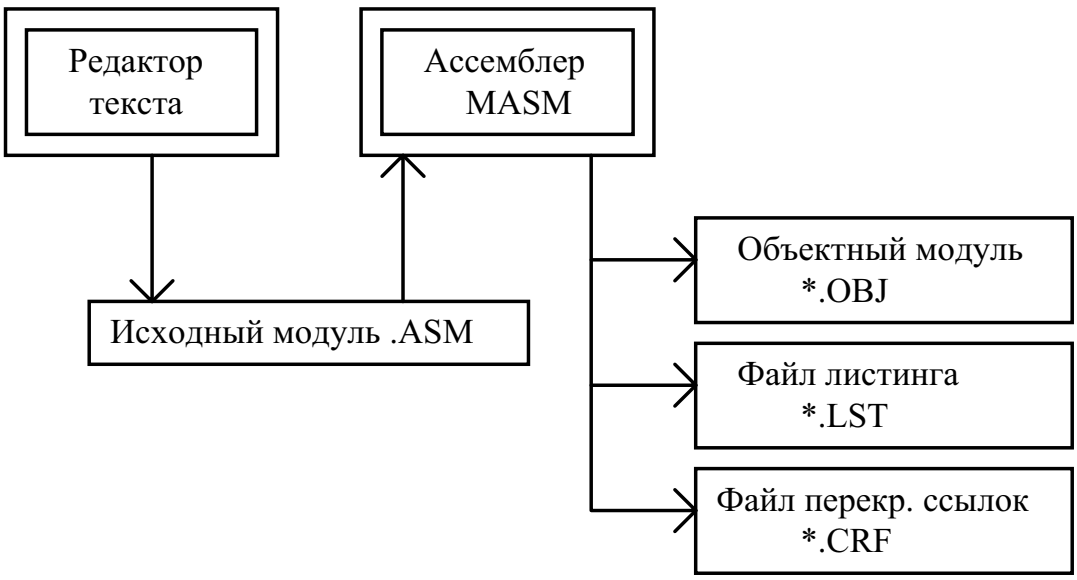

Рис. 1. Схема процесса ассемблирования

Ассемблер MASM является двухпроходным. Цель первого прохода – получение информации о местоположении идентификаторов, второго – генерирование машинного кода. Для локализации идентификаторов в

ассемблере предусмотрена переменная, называемая счетчиком ячеек (адресов). Ее можно считать указателем, который в ходе ассемблирования динамически фиксирует относительные позиции (т. е. смещения) внутри каждого сегмента. Таким образом, при сканировании программы производится инкремент счетчика ячеек на число байтов, занимаемых операторами, и при переходе из одного сегмента в другой счетчик ячеек сбрасывается в нуль. В ассемблере MASM именем, которое позволяет программисту непосредственно обращаться к счетчику ячеек, является символ $.

На первом проходе ассемблер с помощью счетчика ячеек строит таблицу идентификаторов. Элемент этой таблицы содержит тип и имя сегмента, в котором определяется соответствующий идентификатор, а также тип и имя самого идентификатора. На втором проходе эта информация используется при генерировании команд, операнды которых содержат идентификаторы. Ассемблер имеет две таблицы – машинных команд и директив, в которых находятся мнемоника, коды операций и т. п.

Рассмотрим функции первого и второго проходов ассемблера. Когда при первом проходе в самом левом поле оператора встречается метка или переменная, говорят, что она определяется. В этот момент метка или переменная связывается со смещением, для чего в таблицу имен помещается текущее содержимое счетчика ячеек, символическое представление метки или переменной и другие ее атрибуты. При достижении директивы END первый проход заканчивается и начинается второй. Кроме ассемблирования машинных команд, при втором проходе вводятся константы инициализации из директив определения данных, и вычисляются выражения, содержащиеся в операндах команд и директив. Когда при втором проходе встречается директива END, выходные файлы ассемблера выводятся на соответствующие периферийные устройства, и процесс ассемблирования заканчивается.

Если в момент появления в операнде переменной или метки она уже определена, имеет место обращение назад; в противном случае – вперед. Поскольку длина команды зависит от атрибутов операндов, которые могут быть не указаны явно, при обращении вперед ассемблеру приходится предполагать, на какое значение выполнить инкремент счетчика ячеек. Неправильное предположение либо вызовет ошибку ассемблирования, либо заставит второй проход компенсировать ошибки с получением неоптимального кода. Желательно явное указание атрибутов с помощью соответствующих операторов или префиксов замены сегмента, например SHORT, PTR, ES и др.

Рассмотрим команду JMP EXIT. Если EXIT является обращением вперед, в момент появления команды ассемблер не знает тип перехода:

короткий, близкий или далекий – и поэтому резервирует для команды 3 байта (по умолчанию все метки близкие). С другой стороны, команда JMP FAR PTR EXIT, без сомнения, имеет длину 5 байт. Аналогичная ситуация имеет место в команде MOV AX, OPERAND , где OPERAND является обращением вперед к сегменту, отличающемуся от сегмента DS, например SS. При первом проходе ассемблер предполагает, что должен использоваться сегментный регистр DS, и не учитывает необходимого префикса замены сегмента. Это вызовет ошибку при втором проходе.

Исходя из указанных особенностей, можно сделать следующие выводы:

1) необходимо применять атрибутные операторы для явного определения типов операндов во всех командах JMP;

2) следует помещать все директивы определения данных перед командами (т. е. сегменты данных перед сегментами кода);

3) если выражение в операторе EQU обращается к имени переменной или метки (или к имени, определенному в другом операторе EQU), оно должно быть направлено назад;

4) поскольку ассемблер просматривает исходный текст только до директивы END, допускается хранить неиспользуемые фрагменты текста программы после END.

1.1. Исходный модуль программы

Входом для ассемблера является файл, содержащий исходный текст программы. Более полную информацию несет ее листинг, однако он всегда на 41 позицию шире исходного текста, что необходимо учитывать при документировании программы. Макроассемблер MASM имеет группу директив управления листингом и форматом, позволяющих программисту производить постраничную разбивку текста, управлять заголовками страниц и подавлять печать всего объектного файла или его частей. Для управления страницами листинга используется директива PAGE [(длина)], [(ширина)], где значением параметра (длина), если он указан, является длина новой страницы в диапазоне от 10 до 255 строк (по умолчанию 50). Значение второго параметра – это ширина новой страницы в диапазоне от 60 до 132 символов (по умолчанию 80). Если директива задана без аргументов или с необязательным аргументом [+], ассемблер начинает новую страницу.

Выше приведен исходный текст программы вывода на дисплей слова «ФМиПКК». Директива PAGE 60, 80 устанавливает формат листинга 60 строк по 80 символов в строке на страницу. NAME

First_program определяет имя модуля, которое затем передается компоновщику. Директива TITLE *Первая программа* определяет заголовок для каждой страницы листинга. Так как программа использует операции MS DOS, она должна резервировать стек размером не менее 128 слов, причем тип STACK указывает на объединение стека программы с системным стеком. В сегменте *data* содержится сообщение 'ФМиПКК', завершающееся ограничителем '$' (для вывода строки будет использована функция 9 прерывания INT 33). Программа будет запущена загрузчиком операционной системы с метки *start* в сегменте code (так как указано END start) и начнет свою работу с того, что установит регистры SS, и DS на стек и данные, соответственно. Программа завершается обращением к функции 4ch MS DOS (прерывание INT 33), по которой управление будет передано в вызывающую программу.

```
PAGE      60,80           ;Листинг 60 строк 80 симв.
NAME      First_programm;Имя модуля для компоновщика
TITLE     Первая программа;Заголовок для листинга
stack     SEGMEMT STACK   ;Определение сегмента
          DW  256  DUP  (?);Резервирование 256 слов
top       LABEL WORD      ;Вершина стека
stack     ENDS            ;Конец сегмента
data      SEGMENT         ;
mess      DB 'ФМиПКК$'    ;Сообщение
data      ENDS            ;
code      SEGMENT         ;Программный сегмент
ASSUME  cs:code           ;Установка регистра CS
start:                    ;Точка входа в программу
          cli             ;
          mov ax,stack    ;
          mov ss,ax       ;Установка стековых
          mov sp,OFFSET top; регистров SS и SP
          sti             ;
ASSUME ss:stack           ;
          mov ax,data     ; Установка сегментного
          mov ds,ax       ; регистра DS
ASSUME ds:data            ;
          lea ds,mess     ;
          mov ah,9        ;Вызов прерывания DOS для
          int 21h         ;вывода сообщения
          mov ah,4ch      ;Вызов прерывания для
          int21h          ; окончания программы
code      ENDS            ;Конец программного сегмента
END  start                ;Конец программы
```

1.2. Запуск макроассемблера MASM и формат листинга

После создания исходного файла можно вызывать макроассемблер MASM, который предназначен для преобразования исходной программы в перемещаемый объектный файл. Впоследствии он может быть связан с другими файлами программой LINK, в результате чего образуется единый выполняемый модуль. Кроме того, макроассемблер может создавать файл листинга и файл перекрестных ссылок (с расширениями по умолчанию LST и CRF, соответственно).

Существуют два способа вызова транслятора. В первом случае необходимо ввести MASM. Макроассемблер загружается в память, после чего последовательно выдаются четыре подсказки, на которые требуется ответить. В конце каждой строки можно указать один или более ключей. Подсказки приводятся в табл. 1 (в квадратных скобках заданы значения, принятые по умолчанию).

Чтобы вызвать транслятор вторым способом, необходимо ввести следующую команду:

MASM <исх. ф.>, <об. ф.>, <листинг>, <перекр. ссылки> [/(ключи)]

Элементы этой команды являются ответами на четыре подсказки ассемблера. Ключи (табл. 2) управляют выбором тех или иных функций при работе транслятора и вводятся в конце строки ответа.

Существуют также два символа, обеспечивающих выполнение некоторых функций при вводе команд вызова транслятора. Символ «точка с запятой» (;), за которым следует возврат каретки, может быть задан в любой момент при ответе на подсказки, но только после ответа на вопрос об имени исходного файла (первая подсказка).

При его вводе на все остальные подсказки принимаются ответы по умолчанию. Этот символ нельзя использовать для отказа от ответа на отдельную подсказку (для этого используется возврат каретки).

Предположим, что исходный текст программы program.asm и masm.exe находится на диске C. Тогда трансляцию программы можно выполнить следующим образом:

C:> MASM PROGRAM,,PROGRAM,PROGRAM<ENTER>

В таком случае на диске C появятся три новых файла: program.obj, program.lst, program.crf. При возникновении ошибок в процессе трансляции ассемблер выдает соответствующие сообщения на экран и фиксирует их в выходном листинге program.lst (файлы program.obj и program.crf не создаются).

Листинг, выдаваемый ассемблером, делится на две части. В первой из них содержатся: номер для каждой строки исходного текста (в том случае, если создается файл перекрестных ссылок); смещение для каждой исходной строки, генерирующей код; генерируемый код; знак плюс (+), если код возник в процессе макрогенерации, или символ C, если код возник в результате ассемблирования файла, включенного по INCLUDE; исходный оператор. Вторая часть листинга включает: макросы – имя и длину в байтах; структуры и записи – имя, ширину поля; сегменты и группы – имя, размер, тип выравнивания, тип связывания и имя класса; символы – имя, тип, значение, атрибуты; количество предупреждающих сообщений и ошибок.

Таблица 1

Подсказка	Ответ
Sourse filename [.ASM]:	Ввести имя ассемблируемого файла. Имя файла ввести обязательно (умолчание не допускается); расширение по умолчанию – ASM; в противном случае указать расширение в явном виде.
Object filename [source.OBJ]:	Ввести имя объектного файла. По умолчанию оно совпадает с именем исходного файла, расширение по умолчанию – OBJ. Чтобы было принято значение по умолчанию, следует ввести только символ <Enter>.
Sourse listing [NUL.LST]:	Ввести имя файла листинга; по умолчанию листинг не выводится. Если ввести имя без расширения, по умолчанию принимается LST.
Cross reference [NUL.CRF]:	Ввести имя файла перекрестных ссылок; по умолчанию файл перекрестных ссылок не создается. Если имя без расширения, то расширение CRF

Таблица 2

Ключ	Выполняемая функция
/D	Указывает, что будет выдан листинг обеих фаз ассемблирования. Сравнение этих листингов позволяет обнаружить ошибки фазы. Ключ не оказывает никакого действия, если имя файла листинга не указано (листинг не создается).
/O	Вывод листинга производится в восьмеричной системе (коды и смещения выводятся в восьмеричной системе).
/X	Подавляет вывод в файл листинга блоков исходного текста в том случае, если выражение, заданное как условие ассемблирования, ложно.
/MX	Переводит ассемблер в режим распознавания верхнего и нижнего регистров

Таблица 3

Подсказка	Ответ
[.OBJ]	Ввести список объектных модулей, отделяемых друг от друга пробелами или знаками плюс (+). Расширение по умолчанию – OBJ; иначе указать в явном виде.
EXE file [.EXE]	По умолчанию принимается имя первого в списке объектных модулей с расширением EXE. Другое расширение недопустимо.
MAP file [Nul.MAP]	Файл перекрестных ссылок по умолчанию не создается, требуется указать имя. Расширение по умолчанию – MAP.
Libraries [LIB]	Ввести имена библиотек, в которых должен осуществляться поиск; имена отделяются пробелами или знаками плюс (+). Расширение по умолчанию – LIB

В листинге используются следующие обозначения: R – компоновщик перемещает значение, расположенное слева от R; E – внешний символ; = – в операторе задана директива EQU или = ; пп/ – оператор с префиксом REP или LOCK; [x] – выражение в DUP; xx – значение в скобках, следующее за DUP, и т. п.

1.3. Компоновщик LINK

Компоновщик предназначен для объединения отдельно оттранслированных объектных модулей в один перемещаемый загрузочный модуль с присоединением библиотечных процедур.

Программа LINK может вызываться несколькими способами. При одном из них требуется ввести LINK. Программа загружается в память, после чего последовательно выдаются четыре подсказки, на которые нужно ответить. В конце каждой строки можно указать один или более ключей. Подсказки приводятся в табл. 3, а ключи – в табл. 4.

Таблица 4

Ключ	Выполняемая функция
/HIGH	Выполняемый файл должен располагаться в памяти по старшим адресам; по умолчанию он расположен по младшим адресам.
/MAP	Должны выводиться все глобальные символы, определенные во входных модулях; по умолчанию выводятся только ошибки.
STACK:<число>	Определяет размер стека; по умолчанию он вычисляется автоматически

Обработаем программой LINK имеющийся объектный модуль program.obj.

Загрузив полученный выполняемый файл (он всегда будет иметь расширение EXE), получим на дисплее сообщение: ФМиПКК.

2. Задание

Создать программный файл на языке ассемблер.

Получить исполняемый файл и файлы листинга и перекрестных ссылок.

Выполнить полученную исполняемую программу.

Лабораторная работа № 7
ПРОГРАММА-СИМУЛЯТОР ДЛЯ ОТЛАДКИ ПРОГРАММ ДЛЯ МИКРОПРОЦЕССОРА 8085 НА ПЕРСОНАЛЬНОМ КОМПЬЮТЕРЕ

Цели работы: смоделировать на компьютере конкретную микропроцессорную систему, используя программу-симулятор AVSIM85; самостоятельно разработать и просимулировать программы для этой системы.

1. Краткие сведения из теории

1.1. Особенности разработки прикладного программного обеспечения микропроцессорных систем

Автоматизированные измерительные системы, к которым относятся приборы неразрушающего контроля, представляют собой сложный объект проектирования. Для проектирования таких систем общепринят блочно-иерархический подход, при котором представления об объекте разбиваются на уровни, начиная с наименее детализированных и кончая наиболее детализированными. Процесс проектирования носит итеративный характер и заключается в движении от решения более общих вопросов к решению частных с возможными возвратами на более общие уровни для коррекции принятых решений.

К числу основных инструментальных средств отладки программ, разрабатываемых для микропроцессорной системы, относятся:

- внутрисхемные эмуляторы;
- программные симуляторы;
- платы развития (оценочные платы);
- мониторы отладки;
- эмуляторы ПЗУ.

Внутрисхемный эмулятор – программно-аппаратное средство, способное замещать собой эмулируемый процессор в реальной схеме. Внутрисхемный эмулятор – это наиболее мощное и универсальное отладочное средство.

По сути дела, «хороший» внутрисхемный эмулятор делает процесс функционирования отлаживаемого контроллера прозрачным, т. е. легко контролируемым, произвольно управляемым и модифицируемым по воле разработчика.

Функционально внутрисхемные эмуляторы делятся на стыкуемые с внешней вычислительной машиной (обычно это бывает IBM PC) и функционирующие автономно. Автономные внутрисхемные эмуляторы имеют индивидуальные вычислительные ресурсы, средства ввода-вывода, не требуют для своей нормальной работы стыковки с какими-либо внешними вычислительными средствами, но за это пользователю приходится расплачиваться либо существенно более высокой ценой, либо пониженными функциональными и сервисными возможностями по сравнению с аналогичными моделями, стыкуемыми с IBM PC.

Обычно, стыковка внутрисхемного эмулятора с отлаживаемой системой производится при помощи эмуляционного кабеля со специальной эмуляционной головкой. Эмуляционная головка вставляется вместо микроконтроллера в отлаживаемую систему. Если микроконтроллер невозможно удалить из отлаживаемой системы, то использование эмулятора возможно, только если этот микроконтроллер имеет отладочный режим, при котором все его выводы находятся в третьем состоянии. В этом случае для подключения эмулятора используют специальный адаптер-клипсу, который подключается непосредственно к выводам эмулируемого микроконтроллера.

Как минимум, эмулятор содержит следующие функциональные блоки:

- отладчик;
- узел эмуляции микроконтроллера;
- эмуляционная память;
- подсистема точек останова.

Более продвинутые модели могут содержать дополнительно:

- процессор точек останова;
- трассировщик;
- профилировщик (анализатор эффективности программного кода);
- таймер реального времени;
- программно-аппаратные средства, обеспечивающие возможность чтения и модификации ресурсов эмулируемого процессора «на лету», т. е. в процессе выполнения программы пользователя в реальном времени;
- программно-аппаратные средства, обеспечивающие синхронное управление, необходимое для эмуляции в мультипроцессорных системах;
- интегрированную среду разработки.

Отладчик является своеобразным мостом между разработчиком и отладочным средством. Состав и объем информации, проходящей через

средства ввода-вывода, доступность ее для восприятия, контроля, и, при необходимости, для коррекции и модификации напрямую зависят от свойств и качества отладчика.

Хороший отладчик позволяет осуществлять:

- загрузку отлаживаемой программы в память системы;
- вывод на монитор состояния и содержимого всех регистров и памяти, а также, при необходимости, их модификацию;
- управление процессом эмуляции.

Более мощные отладчики, обычно их называют высокоуровневыми (High-Level Debuggers), помимо этого, позволяют:

- вести символьную отладку, благодаря тому, что отладчик «знает» адреса всех символьных переменных, массивов и структур (за счет использования специальной информации, поставляемой компилятором). При этом пользователь может оперировать более приемлемыми для человека символьными именами, не утруждая себя запоминанием их адресов;
- контролировать и анализировать не только дисассемблированный текст, но и исходный текст программы, написанной на языке высокого уровня, и даже с собственными комментариями.

Такой отладчик позволяет пользователю одновременно контролировать ход выполнения программы и видеть соответствие между исходным текстом, образом программы в машинных кодах и состоянием всех ресурсов эмулируемого микроконтроллера.

Следует отметить, что высокоуровневый отладчик обеспечивает выполнение всех своих функций только в том случае, если используется кросс-компилятор, поставляющий полную и правильную отладочную информацию (не все компиляторы, особенно их пиратские версии, поставляют такую информацию), и при этом формат ее представления должен быть «знаком» отладчику.

Наличие эмуляционной памяти дает возможность использовать ее в процессе отладки вместо ПЗУ в отлаживаемой системе и, более того, отлаживать программу без использования реальной системы или ее макета. При необходимости внесения изменений в отлаживаемую программу достаточно загрузить новую или модифицированную программу в память эмулятора, вместо того чтобы заниматься перепрограммированием ПЗУ.

Существуют модели эмуляторов, которые позволяют пользователю «подставлять» вместо ПЗУ эмуляционную память не только целиком, но и поблочно (в некоторых моделях минимальный размер блока может достигать одного байта), в порядке, определенном пользователем. Для этого пользователю достаточно задать распределение памяти данных и

памяти программ, в соответствии с которым процессор будет получать доступ и к содержимому ПЗУ в отлаживаемой системе, и к содержимому эмуляционной памяти внутрисхемного эмулятора. Такая память обычно называется памятью с возможностью мэппинга.

Трассировщик представляет собой логический анализатор, работающий синхронно с процессором и фиксирующий поток выполняемых инструкций и состояния выбранных внешних сигналов. Существуют модели внутрисхемных эмуляторов, которые позволяют трассировать не только внешние сигналы, но и состояния внутренних ресурсов микроконтроллера, например регистров. Такие эмуляторы используют специальные версии микроконтроллеров (эмуляционные кристаллы).

Процессор точек останова позволяет останавливать выполнение программы или выполнять иные действия, например запускать или останавливать трассировщик при выполнении заданных пользователем условий. В отличие от механизма обычных точек останова, процессор точек останова позволяет формировать и отслеживать условия практически любой степени сложности, и при этом эмулируемый процесс не выводится из масштаба реального времени.

Профилировщик (иначе анализатор эффективности программного кода) позволяет получить по результатам прогона отлаживаемой программы следующую информацию:

- количество обращений к различным участкам программы;
- время, затраченное на выполнение различных участков программы;
- анализ статистической информации, поставляемой профилировщиком, позволяет легко выявлять «мертвые» или перенапряженные участки программ, и в результате оптимизировать структуру отлаживаемой программы.

Совокупность программных средств, поддерживающая все этапы разработки программного обеспечения от написания исходного текста программы до ее компиляции и отладки и обеспечивающая простое и быстрое взаимодействие с другими инструментальными средствами (программным отладчиком-симулятором и программатором), носит название *интегрированной среды разработки*.

Наличие в программной оболочке эмулятора встроенного редактора, встроенного менеджера проектов и системы управления позволяют существенно облегчить работу разработчика, избавив его от множества рутинных действий. Для разработчика стирается грань между написанием программы, ее редактированием и отладкой. Переход от редактирования исходного текста к отладке и обратно происходит «прозрачно» и синхронно с активизацией соответствующих окон, менеджер проектов автоматически запускает компиляцию по мере необходимости и активизирует соответствующие окна программного интерфейса.

Столь же просто можно осуществить и переход к отладке проекта с помощью имеющегося отладчика-симулятора или приступить к «прошивке» ПЗУ отлаженной программой.

Симулятор – программное средство, способное имитировать работу микроконтроллера и его памяти. Как правило, симулятор содержит в своем составе:

- отладчик;
- модель ЦПУ и памяти.

Более продвинутые симуляторы содержат в своем составе модели встроенных периферийных устройств, таких как таймеры, порты, АЦП и системы прерываний.

Симулятор должен уметь загружать файлы программ во всех популярных форматах, максимально полно отображать информацию о состоянии ресурсов симулируемого микроконтроллера, а также предоставлять возможности по симуляции выполнения загруженной программы в различных режимах. В процессе отладки модель «выполняет» программу, и на экране компьютера отображается текущее состояние модели.

Загрузив программу в симулятор, пользователь имеет возможность запускать ее в пошаговом или непрерывном режимах, задавать условные и безусловные точки останова, контролировать и свободно модифицировать содержимое ячеек памяти и регистров симулируемого микропроцессора. С помощью симулятора можно быстро проверить логику выполнения программы и правильность выполнения арифметических операций.

В зависимости от класса используемого отладчика, различные симуляторы могут поддерживать высокоуровневую символьную отладку программ.

Некоторые модели симуляторов могут содержать ряд дополнительных программных средств, таких, например, как интерфейс внешней среды, встроенную интегрированную среду разработки.

В реальной системе микроконтроллер обычно занимается считыванием информации с подключенных внешних устройств (датчиков), обработкой этой информации и выдачей управляющих воздействий на исполнительные устройства. Чтобы в симуляторе, не обладающем интерфейсом внешней среды, смоделировать работу датчика, нужно вручную изменять текущее состояние модели периферийного устройства, к которому в реальной системе подключен датчик. Если, например, при приеме байта через последовательный порт устанавливается некоторый флажок, а сам байт попадает в определенный регистр, то оба эти действия нужно производить в таком симуляторе вручную. Наличие же

интерфейса внешней среды позволяет пользователю создавать и гибко использовать модель внешней среды микроконтроллера, функционирующую и взаимодействующую с отлаживаемой программой по заданному алгоритму.

Очевидной особенностью программных симуляторов является то обстоятельство, что исполнение программ, загруженных в симулятор, происходит в масштабе времени, отличном от реального. Однако низкая цена, возможность ведения отладки, даже в условиях отсутствия макета отлаживаемого устройства, делают программные симуляторы весьма эффективным средством отладки. Нужно подчеркнуть, что существует целый класс ошибок, которые могут быть обнаружены только при помощи симулятора.

1.2. Программа-симулятор AVSIM

Назначение программы – создание и отладка программ для микропроцессоров 8080/8085 с использованием персонального компьютера. Программа для микропроцессора может быть загружена непосредственно в ассемблерном виде при работе симулятора или в виде объектного файла, подготовленного с использованием кросс-ассемблера (например, AVMAC85). Затем симулятор имитирует работу микропроцессорной системы и отображает на экране результаты работы.

Наиболее удобный порядок работы следующий:

1. В текстовом редакторе создается исходный текстовый файл программы для микропроцессора на языке ассемблера.

2. Программный файл компилируется кросс-ассемблером. Исправляются все возникшие ошибки. При вызове кросс-ассемблера дополнительные ключи использовать не нужно.

3. Если требуется, подготовить текстовые файлы с необходимыми данными. Данные, которые должны быть размещены в памяти программ, определяются директивами DB, DW в основной программе.

4. Запускается симулятор. В работающем симуляторе загружаются объектный файл программы для микропроцессора, другие необходимые данные. Выполняются необходимые установки симулятора (например, установка и сброс счетчика циклов, установка типа и диапазона доступных адресов внешней памяти данных, установка точек останова для отладки и т. д.). Подключаются внешние файлы для имитации ввода/вывода на внешние по отношению к микропроцессору устройства.

5. Запускается в работу загруженная программа (т. е. собственно симуляция). Программа может работать в непрерывном или пошаговом режиме. Результаты работы отражаются на экране и за ними можно следить. По результатам можно судить о правильности работы программы.

6. Если в ходе симуляции выявились ошибки, процесс следует повторить с первого шага, изменив исходную программу.

7. После того, как программа заработала правильно, ее исходный текст перекомпилируют со всеми необходимыми ключами и полученную программу в машинных кодах заносят в ПЗУ программ.

Вызов симулятора выполняется из среды MS DOS командной строкой:

AVSIM <-c1> <letter> <file_name>/

-c1 – необязательный ключ, включает режим цветного отображения информации (при его отсутствии изображение черно-белое).

letter – необязательный элемент – буква (для нашего случая можно использовать букву A), если буква не указана, симулятор в процессе загрузки потребует сделать выбор из указанных вариантов.

file_name – имя симулируемого файла (можно не указывать).

После загрузки на экране появляется окно симулятора. Его условно можно разделить на зоны (рис. 1).

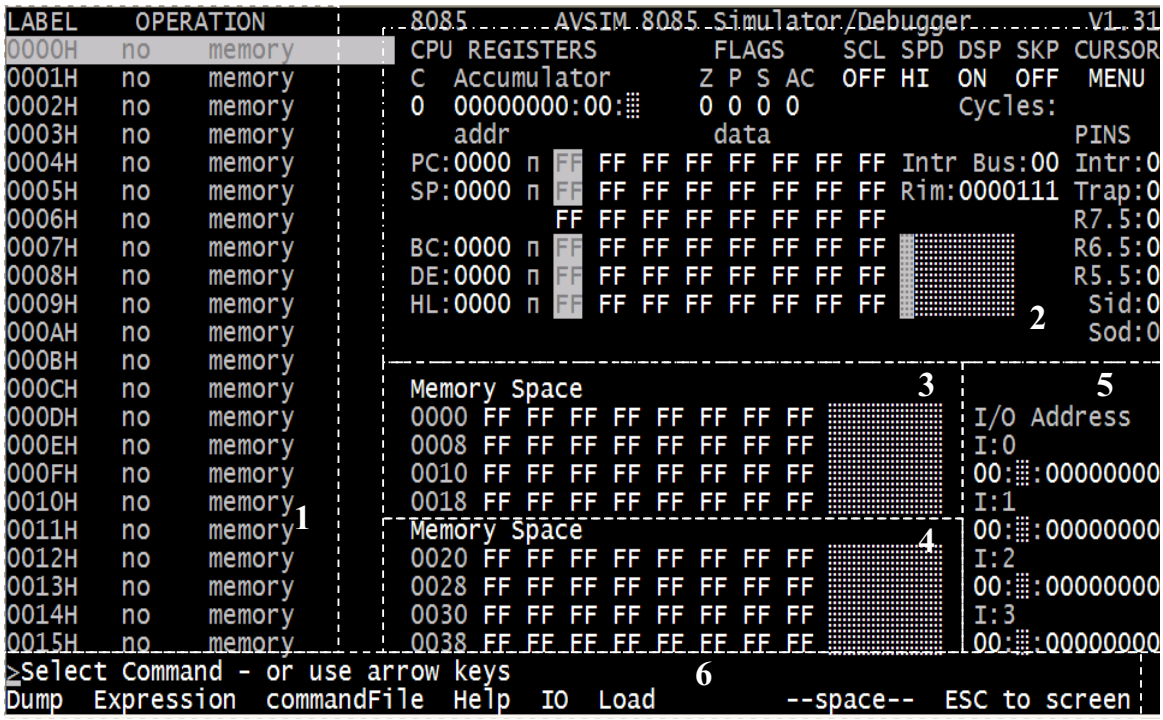

Рис. 1. Окно программы-симулятора

Зона 1 отображает содержимое памяти программ. Зона 2 отображает содержимое общих регистров и регистров специальных функций. Зона 3 – первый дамп памяти (можно настроить на отображение любой области памяти, по умолчанию отображает начало памяти данных). Зона 4 – второй дамп памяти. Зона 5 – адреса ввода/вывода. Зона 6 – главное меню программы.

Краткое описание симулятора приведено в приложениях 1 и 2.

1.3. Микропроцессорная система и ее компоненты

Предлагаемая для моделирования микропроцессорная система построена на основе однокристального микропроцессора 8085 (отечественный аналог – микросхема 1821ВМ85) и содержит также постоянное запоминающее устройство на микросхеме 573РФ2, дешифратор адреса 555ИД7, регистр хранения 555ИР22 и многоцелевую микросхему 8155 (1821РУ55). Принципиальная схема системы приведена на рис. 2.

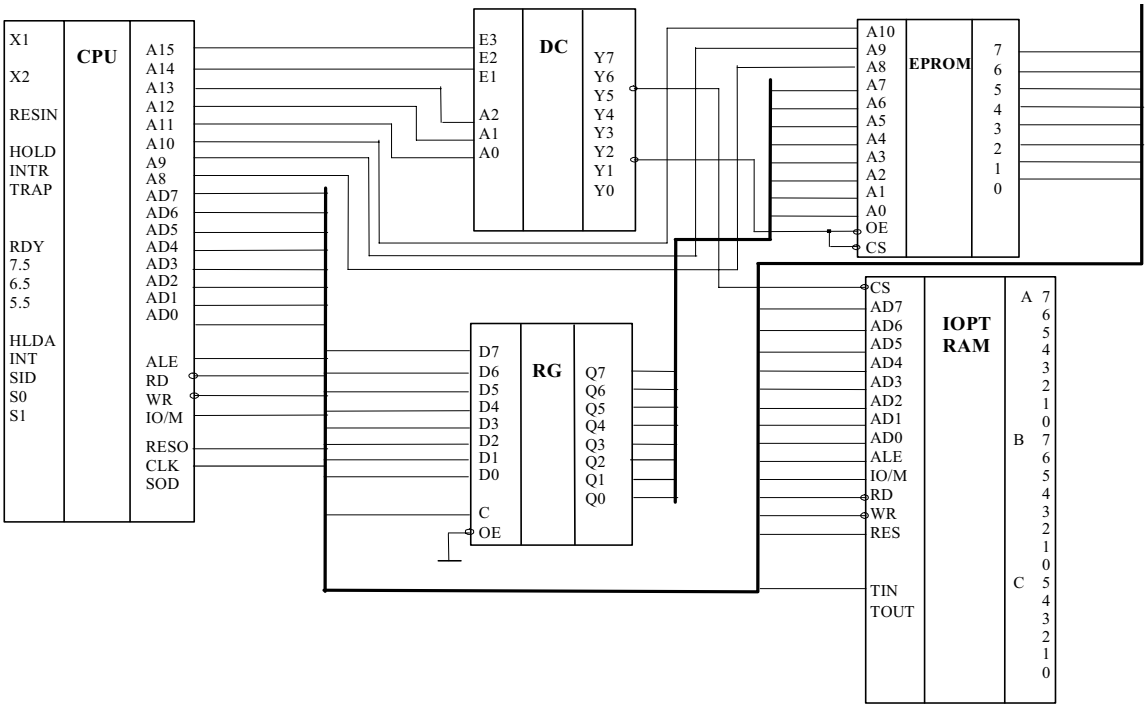

Рис. 2. Принципиальная схема моделируемой системы

Микропроцессор 8085(1821ВМ85) является модернизированным вариантом микропроцессора 8080(580ВМ80). По сравнению с прототипом 8085 обладает повышенным быстродействием и имеет встроенный тактовый генератор, а также несколько особенностей работы. Во-первых, это мультиплексированная шина адреса/данных. При передаче данных из микропроцессора сначала передается адрес, причем младший байт адреса должен быть зафиксирован в дополнительном регистре (в нашей системе это 555ИР22), а затем данные по линиям AD0 – AD7 (линии A8 – A15 содержат при этом старший байт адреса). Во-вторых, при обращении к внешним устройствам адрес внешнего устройства, который является однобайтовым, передается одновременно и по старшему и по младшему байту шины адреса. Это учтено в нашей системе тем, что адресный дешифратор подключен к старшему байту шины адреса и используется одновременно и для выбора банков памяти (573РФ2), и для выбора внешних устройств (1821РУ55). Сделать то же самое для микропроцессора 8080 (580ВМ80) нельзя.

Специализированная БИС 8155(1821РУ55) – микросхема, которая содержит на одном кристалле ОЗУ объемом 256 байт, параллельный интерфейс ввода/вывода (2 порта по 8 линий и 1 порт по 6 линий) и 14-разрядный программируемый таймер. Адресное пространство микросхемы показано на рис. 3. Микросхема может выполнять роль статической памяти, если на вход CS подан сигнал нулевого уровня и сигнал IO/M = 0. Сигналы RD и WR являются сигналами управления чтения/записи в ОЗУ. При активном (равном нулю) сигнале CS и IO/M = 1 происходит обращение к портам ввода/вывода микросхемы. В микросхеме их имеется 6 (назначение и адреса приведены на рис. 3).

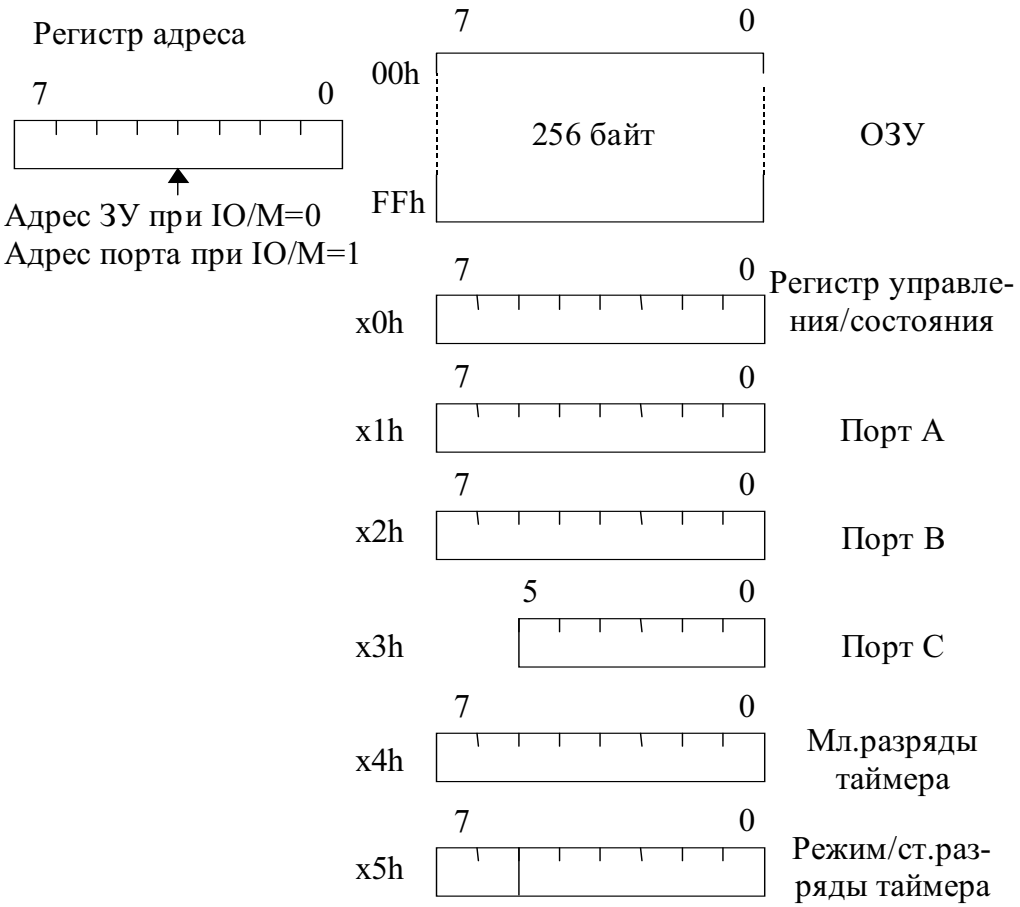

Рис. 3. Адресное пространство микросхемы 8155

Регистр управления содержит управляющее слово для программирования режима работы и направления передачи информации через порты A, B, C и управления таймером. Формат регистра управления показан на рис. 4.

Работа таймера программируется двумя регистрами, формат которых показан на рис. 5.

Слово состояния, формат которого приведен на рис. 6, определяет состояние таймера и портов параллельного обмена.

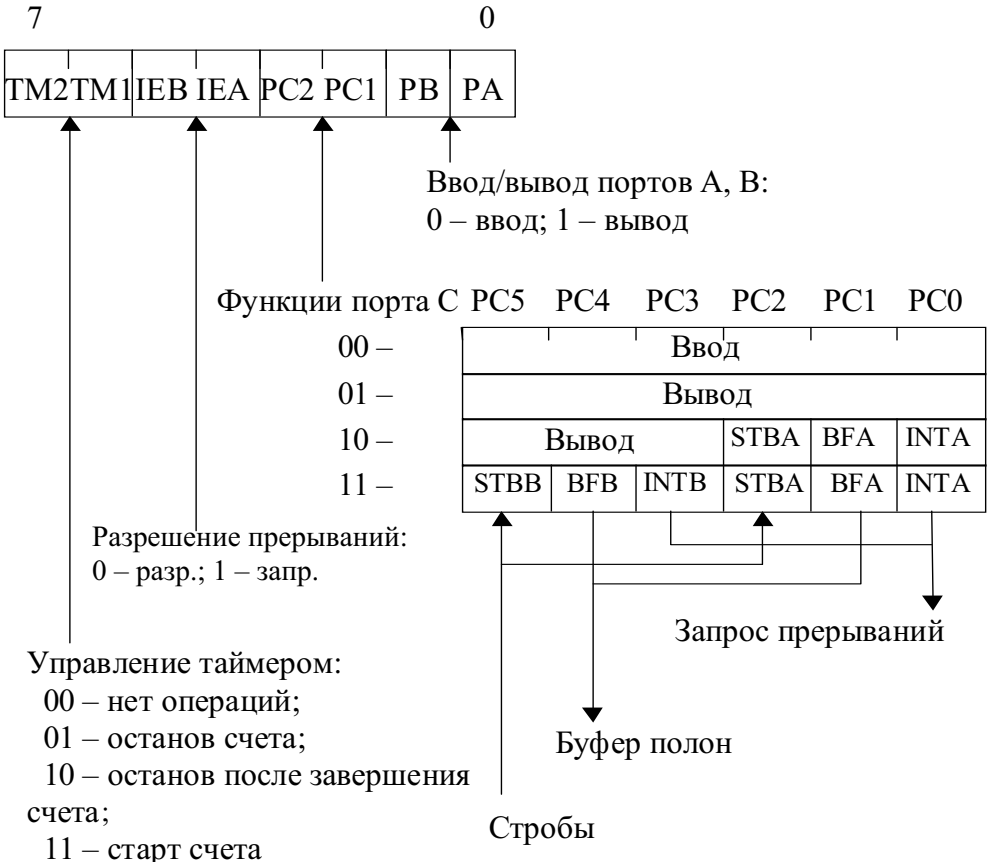

Рис. 4. Формат регистра управления

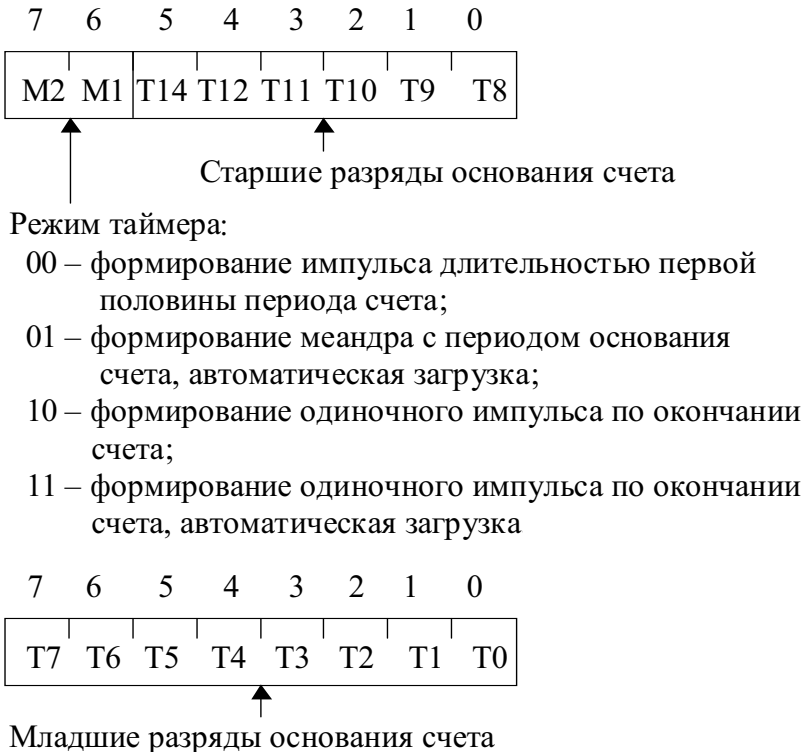

Рис. 5. Формат регистров управления таймером

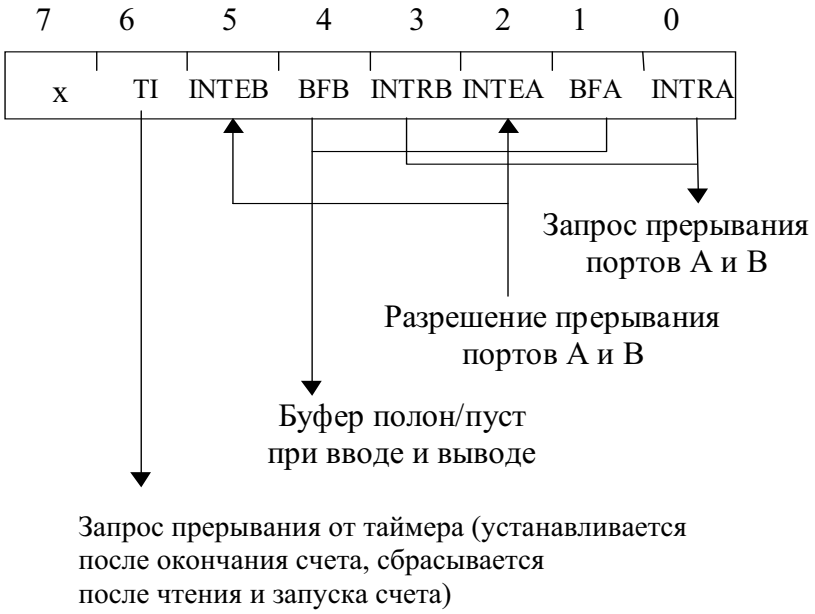

Рис. 6. Формат слова состояния

2. Задание

Используя программу-симулятор, промоделировать работу системы. Для этого выполнить следующее:

1) определить по принципиальной схеме адреса регистров микросхемы 1821РУ55 и диапазон адресов ОЗУ;

2) составить программу, которая должна находиться в ПЗУ (в ней должно быть начальное программирование микросхемы 1821РУ55, установка указателя стека и другие действия, необходимые при начальном включении системы);

3) подготовить программу к работе с симулятором;

4) включить программу-симулятор в режиме B;

5) командой SET / MEMORY установить определенные в п.1 верхние и нижние адреса ПЗУ и ОЗУ;

6) загрузить составленную программу и выполнить ее в пошаговом режиме;

7) составить отчет (в отчет включить программную модель системы и разработанную программу).

Лабораторная работа № 8
СТАНДАРТНЫЕ ИНТЕРФЕЙСЫ ПОСЛЕДОВАТЕЛЬНОГО ОБМЕНА ДАННЫМИ

Цель работы: познакомиться со стандартами, принятыми для обеспечения последовательного обмена в вычислительных системах, физической реализацией каналов последовательного обмена и их программированием.

1. Методические указания по подготовке к лабораторной работе

Микропроцессорная система без внешних устройств (клавиатуры, дисплея, накопителей информации на магнитных носителях, принтера) оказывается бесполезной. Поэтому при разработке вычислительной системы большое внимание уделяется организации взаимодействия системы с внешними устройствами. Основное требование при этом – надежная, без потерь, передача информации. Для обеспечения этого требования разработаны и стандартизованы различные интерфейсы, предназначенные для обмена информацией между вычислительной системой и внешними устройствами.

Интерфейсом называется набор сигналов, которыми обмениваются ЭВМ и внешнее устройство при приеме и передаче информации, а также аппаратура, обеспечивающая этот обмен. Последовательность передачи сигналов во времени называется *протоколом обмена*.

Интерфейсы принято делить на магистральные и радиальные. Магистральным называется такой интерфейс, к которому подключаются одновременно несколько устройств. Такой интерфейс обычно используется для связи блоков самой вычислительной системы. Для связи ЭВМ и внешнего устройства используется обычно радиальный интерфейс (для каждого внешнего устройства используется свой канал связи). По протоколу обмена интерфейсы делят на параллельные и последовательные. Все интерфейсы общего применения ориентированы на байтовый (восьмибитовый) обмен информацией. В параллельном интерфейсе байт данных передается по восьми линиям одновременно, а в последовательном – по одной линии бит за битом, последовательно. В настоящей работе рассматриваются радиальные последовательные интерфейсы общего применения. Наиболее употребительные из них RS-232C (международный стандарт), стык С2 (его отечественный аналог) и ИРПС.

1.1. Интерфейс RS-232C

Интерфейс RS-232C – один из наиболее часто использующихся стандартных последовательных интерфейсов. Основы функционирования этого интерфейса подробно изложены в учебнике, поэтому мы сосредоточим внимание на его физической реализации.

Стандартный последовательный порт интерфейса RS-232C представляет собой 25-контактный разъем. Терминальное оборудование обычно оснащено разъемом со штырьками, а связное – разъемом с отверстиями (но могут быть и исключения).

Разводка контактов разъема RS-232C приведена в табл. 1.

Таблица 1

Номер контакта	Сигнал	Направление передачи	Полное название сигнала
1	FG	–	Основная или защитная земля
2	TD(TxD)	к ООД	Передаваемые данные
3	RD(RxD)	к АПД	Принимаемые данные
4	RTS	к ООД	Запрос передачи
5	CTS	к АПД	Сброс передачи
6	DSR	к АПД	Готовность модема
7	SG	–	Сигнальная земля
8	DCD	к АПД	Обнаружение несущей данных
9	–	к АПД	Положительное контр. направление
10	–	к АПД	Отрицательное контр. направление
11	QM	к АПД	Режим выравнивания
12	SDCD	к АПД	Обнар. несущей втор. данных
13	SCTS	к АПД	Вторичный сброс передачи
14	STD	к ООД	Втор. передаваемые данные
15	TS	к АПД	Синхронизация передатчика
16	SRD	к АПД	Втор. принимаемые данные
17	RC	к АПД	Синхронизация приемника
18	DCR	к ООД	Разделенная синхр. приемника
19	SRTS	к ООД	Вторичный запрос передачи
20	DTR	к ООД	Готовность терминала
21	SQ	к АПД	Качество сигнала
22	RI	к АПД	Индикатор звонка
23	–	к ООД	Селектор скорости данных
24	TC	к ООД	Внешн. синхр. передатчика
25	–	к ООД	Занятость

На практике вспомогательный канал RS-232C применяется редко, и в асинхронном режиме из 25 линий используется обычно только первые 8. Назначение линий в этом случае приведено в табл. 2.

1.2. Интерфейс ИРПС

Стандарт ИРПС (интерфейс радиальный, последовательный) описывает вариант последовательного интерфейса, обеспечивающего асинхронную передачу постоянного тока (токовая петля) по четырехпроводной линии. В некоторых случаях допустима цепь, указывающая на готовность внешнего устройства. Линии сигналов интерфейса ИРПС приведены в табл. 4. Каждая линия представляет собой двухпроводную петлю. Наличие или отсутствие тока в петле является *логической единицей* или *нулем*. Стандартизованное значение лог. 1 – 15...25 мА, лог. 0 – 0...3 мА («20 мА токовая петля»). Цепи взаимосвязи выполняются витыми парами проводов. Типы применяемого разъема и кабеля стандартом не регламентируются.

1.3. Последовательный интерфейс IBM-совместимых компьютеров

В IBM-совместимых компьютерах для последовательной передачи данных применяется стандарт RS-232C, однако используются 9- и 25-контактные разъемы с нестандартным расположением линий интерфейса. В табл. 5 приведена разводка 9-контактного разъема.

Таблица 2

Контакт	Сигнал	Выполняемая функция
1	FG	Подключение защитной земли или экрана
2	TxD	Последовательные данные от АПД к ООД
3	RxD	Последовательные данные от ООД к АПД
4	RTS	Активным уровнем АПД показывает желание переслать данные к ООД
5	CTS	Активным уровнем ООД указывает на готовность принимать данные
6	DSR	Активным уровнем ООД сообщает, что есть связь
		Сигнальная земля
7	SG	Активным уровнем АПД указывает, что ООД может подклю-
8	DCD	читься к каналу связи

Таблица 3

Контакт	Номер цепи	Наименование сигнала	Группа
1	101	Экран	Заземление
7	102	Сигнальная земля	
2	103	Передаваемые данные	Данные
3	104	Принимаемые данные	
4	105	Запрос передачи	Управление
5	106	Готов к передаче	
6	107	Передатчик готов	
20	108.2	Приемник готов	
8	109	Детектор принимаемого линейного сигнала	
22	125	Индикатор вызова	

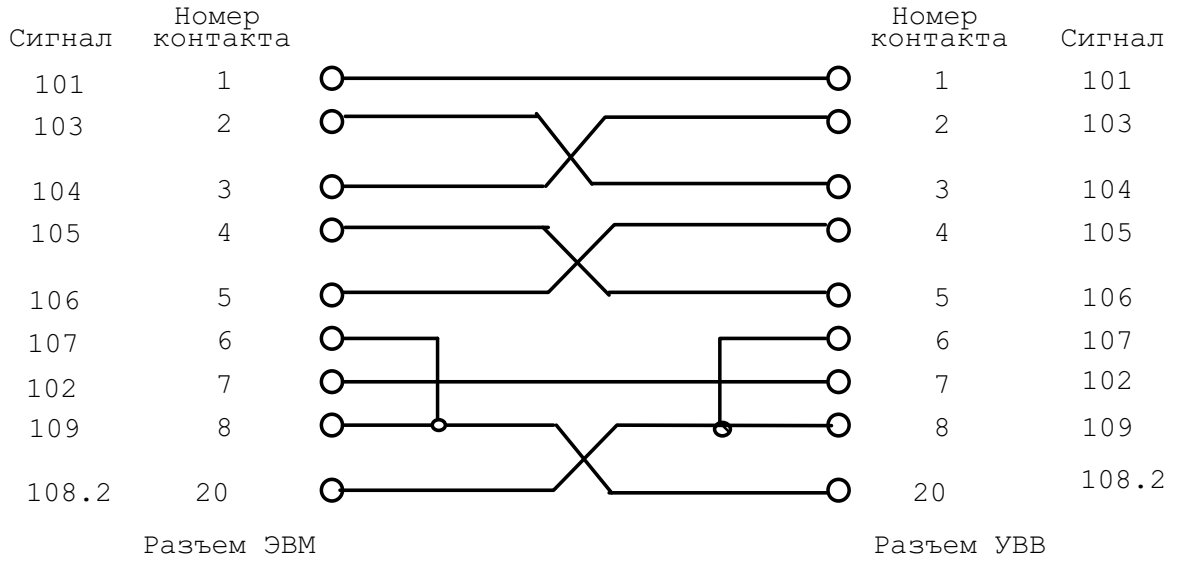

Рис. 1. Соединение линий последовательного интерфейса

Таблица 4

Номер	Наименование	Обозначение
1	Передаваемые данные (прямой и обратный провод)	ПД+/ПД-
2	Принимаемые данные	ПрД+/ПрД-
3	Готовность приемника (необязательная цепь)	ГП+/ГП-

Таблица 5

Номер контакта	Наименование сигнала	Назначение сигнала
1	DCD	Обнаружение несущей данных
2	RxD	Принимаемые данные
3	TxD	Передаваемые данные
4	DTR	Готовность терминала
5	–	Общий
6	DSR	Готовность модема
7	RTS	Запрос передачи
8	CTS	Сброс передачи
9	RI	Индикатор звонка

1.4. Контроллер последовательного интерфейса

Интерфейсы последовательных каналов ввода/вывода в микропроцессорных системах реализуются с использованием БИС КР580ВВ51. Микросхема представляет собой универсальный синхронно-асинхронный приемопередатчик последовательной связи, выполняющий функции приема и преобразования параллельных форматов слов в последовательные для их передачи по каналам связи, и последовательных форматов, принимаемых из каналов связи, в параллельный формат

для ввода в процессор. Микросхема может быть запрограммирована для работы в пяти режимах: асинхронная передача, асинхронный прием, синхронная передача, синхронный прием с внутренней синхронизацией, синхронный прием с внешней синхронизацией.

На рис. 2 приведена структурная схема БИС, а в табл. 6 – назначение выводов.

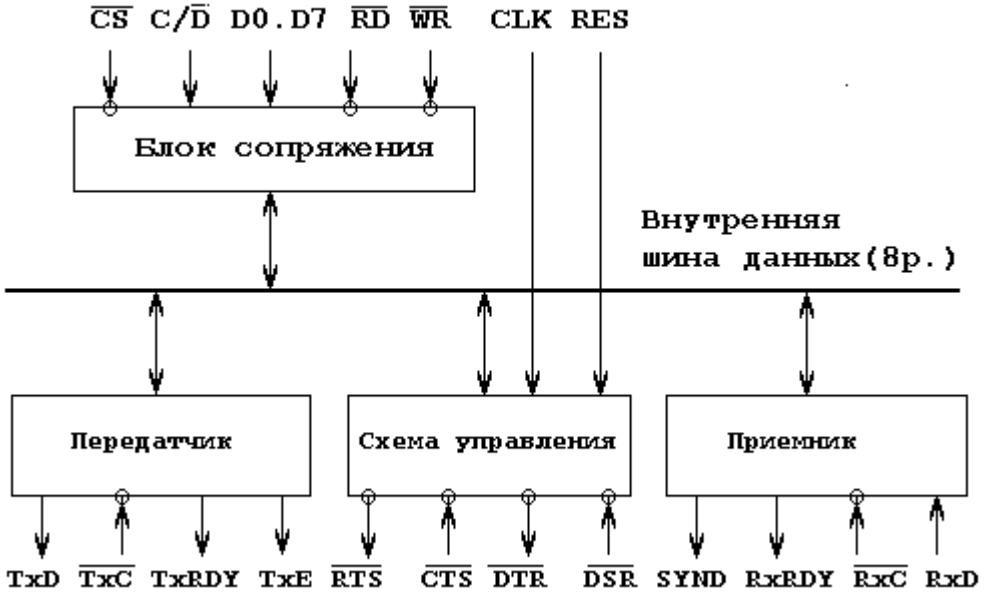

Рис. 2. Функциональная схема контроллера 580ВВ51

Передатчик принимает данные с шины данных, преобразует их в последовательный код, добавляет служебные разряды и выдает их на выход передатчика TxD под управлением сигналов синхронизации со входа TxC. Приемник принимает данные со входа RxD, преобразует их в параллельный код, исключает служебные символы и посылает на шину данных. Прием синхронизируется сигналами на входе RxC. В режиме асинхронной передачи/приема скорость передачи или приема кратна частоте сигналов на входе TxC/RxC. Коэффициент кратности устанавливается программно и равен 1; 16 или 64. Сигналы TxRDY и RxRDY используются для связи с процессором. Сигнал TxRDY указывает, что передатчик готов принять новое слово данных от процессора. В единичное состояние сигнал устанавливается после программного запуска передачи и после завершения передачи очередного слова данных, а сбрасывается в нулевое состояние после записи байта данных в регистр данных передатчика. Сигнал RxRDY показывает, что данные в приемнике готовы для ввода в процессор. Он устанавливается в единичное состояние после приема слова данных и сбрасывается после считывания

данных процессором. Оба сигнала могут быть использованы как сигналы требования прерываний, в случае организации ввода/вывода методом опроса, сигналы не используются.

Таблица 6

Обозн.	Наименование и назначение
D0..D7	Двунаправленные трехстабильные линии данных
CS	Вход выбора микросхемы
WR	Вход сигнала записи с шины данных в адресуемый регистр
RD	Вход сигнала чтения из регистра на шину данных
C/D	Управл./данные, 0 – на шине данные, 1 – управляющее слово
CLK	Вход сигнала от тактового генератора
RES	Вход сигнала сброса
SYND	Выход установления внутренней синхронизации
TxC	Вход синхронизации передатчика
RxC	Вход синхронизации приемника
CTS	Вход готовности приемника терминала
DSR	Вход готовности передатчика терминала
TxD	Выход передатчика
RxD	Вход приемника
RTS	Выход запроса приемнику терминала
DTR	Выход запроса передатчику терминала
TxE	Выход регистр передатчика пуст
RxRDY	Выход готовности приемника
TxRDY	Выход готовности передатчика

Схема управления содержит регистры управляющих слов, регистр состояния, схему управления модемом. Синхронизируется БИС сигналами, подаваемыми на вход CLK (обычно используют вторую фазу сигналов синхронизации микропроцессора). Сигнал SR длительностью не менее 6 периодов синхронизации используется для установки БИС в исходное состояние. Выход сигнала запроса приемнику терминала RTS устанавливается в 0 программно и используется как требование внешнего устройства к передаче данных. Сигнал на входе готовности приемника терминала CTS указывает (при CTS = 0), что передача данных внешним устройством разрешена. Выход сигнала запроса готовности передатчику терминала DTR можно использовать для синхронизации работы передатчика и управления скоростью выборки. Он устанавливается в 0 программно. Вход сигнала готовности передатчика терминала DSR указывает на готовность внешнего устройства к передаче, фиксируется в слове состояния и может быть проанализирован программой.

С процессором БИС сопрягается посредством шины данных DO...D7 и управляющих сигналов CS, C/D, RD и WR. На вход выбора кристалла CS подается сигнал логического 0 с селектора адреса, определяющего адрес, по которому обращаются к БИС при программировании. Вход C/D (управление/данные) обычно управляется линией AO шины адреса. Входы, управляющие чтением и записью информации RD и WR, соединяются с линиями IORC и IOWR процессора (линии, управляющие чтением/записью во внешние устройства). Возможные варианты сочетания управляющих сигналов и направления передачи информации в системе приведены в табл. 7.

Микросхема содержит 7 программно-доступных регистров: 8-разрядные регистры данных (РД), состояния (РС), режима (РР), команд (РК), первого синхросимвола (РСС1), второго синхросимвола (РСС2) и регистр передатчика. В асинхронном режиме работы регистры РСС1 и РСС2 не используются.

Перед использованием контроллер должен быть запрограммирован. Программирование осуществляется записью нужной информации в регистры БИС. Порядок записи управляющих слов и данных определяется рядом правил. После сигнала сброса по линии SR производится запись управляющего слова режима в РР. Формат управляющего слова режима приведен в табл. 8. Следующие управляющие слова, в зависимости от содержимого РР, интерпретируются либо как первый синхросимвол и записываются в РСС1 (для синхронного режима), либо как команда (в асинхронном режиме). В синхронном режиме третье управляющее слово воспринимается как второй синхросимвол и записывается в РСС2, а четвертое слово также воспринимается как команда.

Таблица 7

Тип операции	Операция	Сигналы управления			
		C/D	RD	WR	CS
Чтение	ШД←РД	0	0	1	0
(Ввод)	ШД←РС	1	0	1	0
Запись	ЩД→РПД	0	1	0	0
(Вывод)	ШД→РУ	1	1	0	0
Отключение от ШД		–	–	–	1
Запрещено		–	0	0	0

Таблица 8

Разряды рег. режима		Функции, выполняемые микросхемой в каждом режиме	
D1	D0	Режим, отношение частот синхронизации :	
0	0	синхронный режим работы;	
0	1	асинхронный режим работы, отношение частот 1:1;	
1	0	асинхронный режим работы, отношение частот 1:16;	
1	1	асинхронный режим работы, отношение частот 1:64	
D3	D2	Число информационных бит в пакете (без стоп-битов, бита паритета и старт-бита):	
0	0	5-разрядный символ;	
0	1	6-разрядный символ;	
1	0	7-разрядный символ;	
1	1	8-разрядный символ	
D5	D4	Контроль паритета:	
x	0	нет контроля;	
0	1	нечетный паритет;	
1	1	четный паритет	
D7	D6	Длина стоп-бита (асинхронный режим):	Вид синхронизации (синхронный режим):
0	0	запрещено;	внутренняя, 2 синхросимвола;
0	1	1 стоп-бит;	внешняя, 2 синхросимвола;
1	0	1,5 стоп-бита;	внутренняя, 1 синхросимвола;
1	1	2 стоп-бита	внешняя, 1 синхросимвол

Формат регистра команд приведен в табл. 9. На этом начальное программирование заканчивается. В дальнейшем на микросхему могут поступать данные и команды в произвольном порядке (команды от данных микросхема отличает по сопровождающему их сигналу C/D, лог. 1 – команды, лог. 0 – данные). Если необходимо сменить режим, нужно подать команду программного сброса или сигнал SR, после чего повторить процедуру начального программирования.

Программный контроль за состоянием приемопередатчика возможен посредством слова состояния, формат которого приведен в табл. 10.

В настоящей работе БИС используется в режиме асинхронного обмена.

Таблица 9

Разряд	Обозначение	Функции
D0	РПД	Разрешение передачи при ЗПД = 1
D1	ЗПД	Запрос передачи, DTR = 0 при ЗПД = 1
D2	РПМ	Разрешение приема при РПМ = 1
D3	РП	Разрыв (конец) передачи при РП = 1 (TxD = 0)
D4	СФО	Сброс флагов ошибок при СФО = 1
D5	ЗПМ	Запрос приема, RTS = 0 при ЗПМ = 1
D6	СБ	Программный сброс при СБ = 1
D7	РС	Разрешение поиска синхросимволов при РС = 1

Таблица 10

Разряд	Обозначение	Функции
D0	TxRDY	Готовность передатчика при TxRDY = 1
D1	RxRDY	Готовность приемника при RxRDY = 1
D2	TxE	Регистр передатчика пуст при TxE = 1
D3	ОК	Ошибка контроля четности при ОК = 1
D4	ОН	Ошибка переполнения приемника (процессор не считал данные до получения новых)
D5	ОФ	Ошибка формата при ОФ = 1 (не найдено битов останова в конце посылки)
D6	SYND	Синхронизация: 1 – есть, 0 – нет
D7	DSR	Готовность передатчика терминала при DSR = 1

2. Методические указания к работе

Для выполнения лабораторной работы используется следующее оборудование:

1) учебный микропроцессорный комплект УМК;
2) лабораторный макет;
3) персональный компьютер.

2.1. Описание лабораторного макета

Лабораторный макет представляет собой плату со смонтированными на ней контроллером последовательного интерфейса и обслуживающими схемами, которая вставляется в гнездо расширения УМК. Основа макета – плата ТЭЗ М1 из комплекта поставки УМК, которая представляет собой макетную плату со схемами буферизации системной шины микропроцессорной системы, контроллером параллельного интерфейса и дешифратором адреса. Схема ТЭЗ М1 приведена в книге «УМК. Эксплуатационная документация». Схема дополнительной части лабораторного макета приведена на рис. 3. Схема состоит из контроллера последовательного обмена на основе микросхемы К580ВВ51А, таймера на

основе микросхемы К580ВИ53 и схемы формирования сигналов по стандарту ИРПС (20 мА токовая петля). Выводы контроллера, предназначенные для обмена информацией с процессором, присоединены к соответствующим контактам ТЭЗ М1, вход CS контроллера соединен с выходом 15 дешифратора адреса (микросхема D9 ТЭЗ М1). Таким образом, в адресном пространстве портов ввода/вывода МП-системы регистры контроллера имеют адреса, начиная с адреса 80Н. Схема формирования сигналов токовой петли содержит транзисторные ключи для согласования с контроллером и оптронные диодные пары для гальванической развязки цепей контроллера и линий связи. Таймер необходим для формирования сигнала синхронизации передатчика и приемника контроллера последовательного интерфейса с частотой, близкой к 153600 Гц, чтобы получить скорость обмена информации 9600 бит/с, из системного сигнала синхронизации CLK (2 МГц).

Персональный компьютер предназначен для ввода символьной информации с клавиатуры и вывода символьной информации на экран видеомонитора. С микропроцессорной системой компьютер может связываться по стандартному последовательному интерфейсу через порт COM1 или COM2. Передача информации осуществляется в дуплексном режиме. Это означает, что символ после нажатия на клавишу клавиатуры компьютера появится на экране видеомонитора только после того, как микропроцессор вернет в него код нажатой клавиши.

2.2. Порядок работы

2.2.1. Соберите лабораторный макет, соедините макет и персональный компьютер проводами (витыми парами). Включите УМК и компьютер, прогрейте их в течение нескольких минут.

2.2.2. Введите в УМК программу начального программирования таймера и контроллера последовательного интерфейса. Запрограммируйте интерфейс в режим асинхронного обмена (отношение частот 1:16, 8 бит данных, 2 стоп-бита, контроля паритета нет).

2.2.3. Введите в УМК программу вывода одного символа на экран дисплея и выполните её. Убедитесь в правильной работе программы. Выполните программу в пошаговом режиме УМК.

2.2.4. Введите программу приема одного символа и выполните её. Убедитесь, что после нажатия на клавишу клавиатуры компьютера в ОЗУ УМК появляется код нажатой клавиши.

2.2.5. На основе рассмотренных программ составьте программу приема последовательности кодов нажатых клавиш с записью этой информации в ОЗУ УМК и отображением на экране дисплея. Занесите программу в память УМК и выполните её.

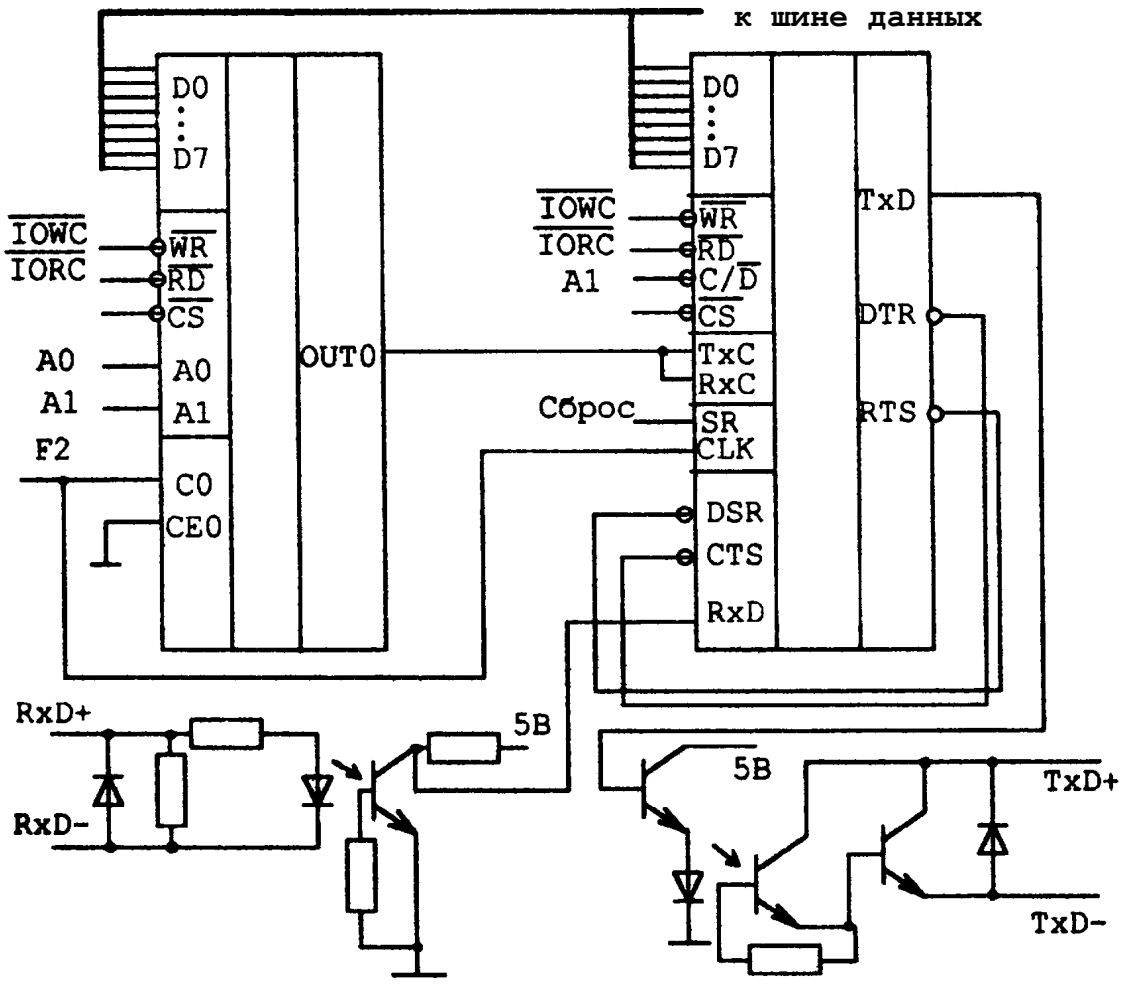

Рис. 3. Схема лабораторного макета

2.2.6. Составьте отчет о проделанной работе.

Лабораторная работа № 9
ИЗУЧЕНИЕ АРХИТЕКТУРЫ ЛАБОРАТОРНОГО СТЕНДА SDK 1.1

Цели работы: изучить архитектуру лабораторного стенда SDK-1.1 на основе микроконтроллера с системой команд MCS51; изучить инструментальное программное обеспечение стенда; загрузить и выполнить простые учебные программы.

1. Методические указания к работе

1.1. Учебный лабораторный комплекс SDK-1.1

Учебный лабораторный комплекс SDK-1.1 (рис. 1) предназначен для освоения студентами архитектуры и методов проектирования:

- систем на базе микропроцессоров и однокристальных микроЭВМ;
- встраиваемых контроллеров и систем сбора данных;
- периферийных блоков вычислительных систем;
- подсистем ввода/вывода встраиваемых систем.

Рис. 1. Комплекс SDK-1.1

В состав комплекса входит:

1. Лабораторный стенд SDK-1.1 (рис. 2), включающий:
   - микроЭВМ ADuC812 с архитектурой MCS-51. К внешнему разъему стенда подключены встроенные адаптеры ввода/вывода:
     - 8-канальный 12-разрядный АЦП;
     - 2-канальный 12-разрядный ЦАП;

- 4-разрядный порт ввода/вывода, поддерживающий функции запроса прерывания (2 канала), счетных входов (2 канала), входа синхронизации АЦП, интерфейса microLAN (Dallas).

- Внешнее ОЗУ 64Кб, используемое как память программ или данных.
- Оптически развязанный приемопередатчик инструментального канала RS232C.
- ИМС периферийных устройств:
  - E^2PROM AT24C01 (128 байт);
  - интегральные часы со встроенным ОЗУ PCF8583 (Philips);
  - модуль символьного ЖКИ 2*16;
  - матричная клавиатура 4*2;
  - звуковой излучатель;
  - 8 управляемых светодиодов;
  - 16-разрядный параллельный порт ввода/вывода.
- Ручные переключатели тестовых сигналов для аналоговых и дискретных портов ввода: коммутатор аналоговых каналов и стимулятор дискретных портов.
- Разъем интерфейса JTAG для контроля периферийной шины и портов, реализованных в ПЛИС MAX3064 (Altera).

Стенд выпускается в виде печатной платы с двусторонним поверхностным монтажом элементов, установленной в корпусе с выведенными клавиатурой, индикатором, переключателями коммутатора и стимулятора портов ввода-вывода, разъемами внешних интерфейсов и электропитания. Параметры портов ввода/вывода соответствуют спецификации TTL, динамический диапазон аналоговых сигналов ЦАП и АЦП – 0...5 В.

2. Внешний адаптер электропитания.
3. Коммуникационный кабель.
4. IBM PC-совместимый персональный компьютер.

Инструментальная система программирования включает:

- Транслятор с языка ассемблера или С для ядра MCS51 и симулятор ADuC812 (пакет *uVision51* фирмы *Keil Software*).
- Резидентный монитор-загрузчик HEX202.
- Программируемую инструментальную систему (загрузчик, терминал) T167B (IBM PC/ DOS) и T2 (IBM PC/Win32).
- Встроенное программное обеспечение и драйвер для ОС Windows 9x/NT для использования контроллера SDK-1.1 в качестве внешнего адаптера ввода/вывода в составе управляющего вычислительного комплекса на базе персонального компьютера.

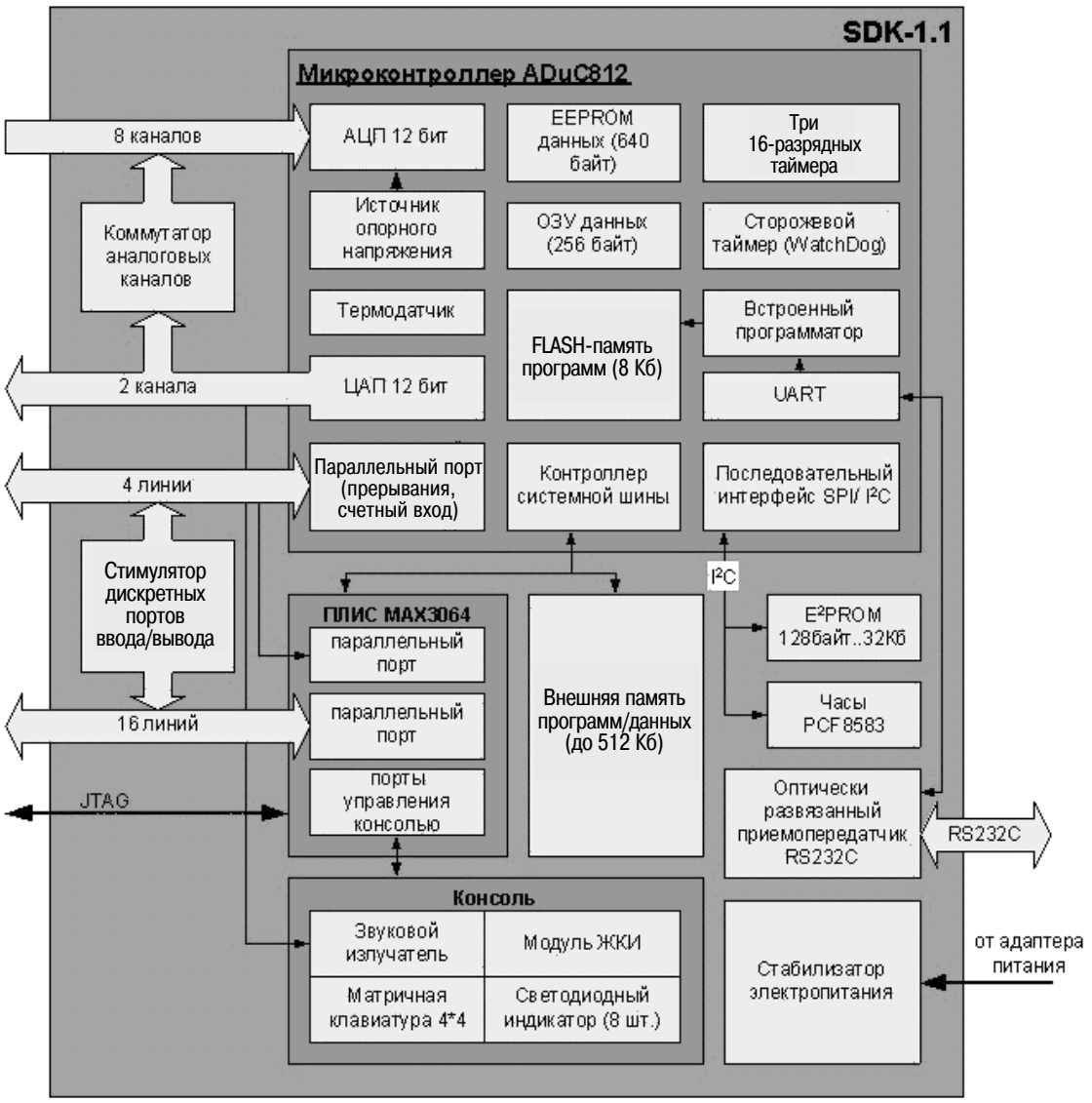

Рис. 2. Лабораторный стенд SDK-1.1

Рис. 3. Взаимодействие учебного стенда с ПК

Подготовка программ для микроконтроллера ADuC812 осуществляется на языке программирования Си на ПК в обычном текстовом редакторе (или средах программирования, IDE, предназначенных для разработки программ под ядро MCS-51), далее программа компилируется в исполняемый модуль, доставляемый в стенд с помощью разработанного программного обеспечения.

Программы для стенда располагаются в ОЗУ объемом 128 Кб. Из этих 128 Кб как память программ (особенности MCS-51) могут использоваться лишь 56 Кб (в стенде первые 8 Кб памяти программ заняты ПЗУ, в котором располагается системное программное обеспечение, отсюда 64 Кб – 8 Кб = 56 Кб). Однако, как показывает практика, программы такого размера для стенда подготавливать не требуется. В процессе обучения с использованием SDK-1.1 студенты могут на практике ознакомиться с управлением периферийными устройствами, взаимодействующими с вычислителем посредством различных интерфейсов, освоить некоторые специфические аспекты программирования встраиваемых вычислительных систем, эффективного управления ресурсами.

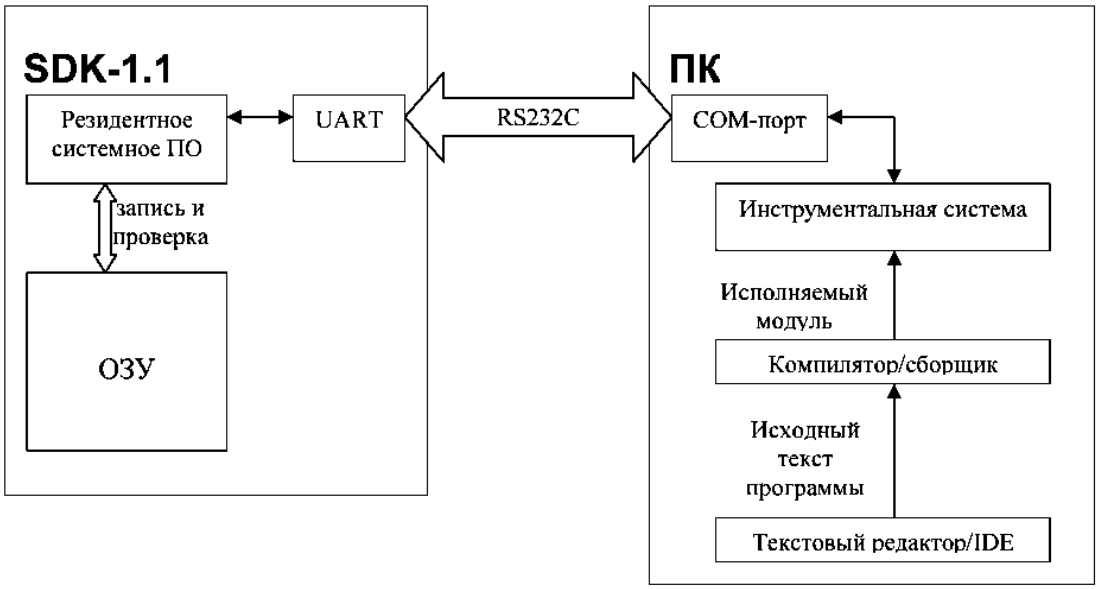

Рис. 4. Процесс написания программ для SDK-1.1

В стенде предусмотрена возможность симулировать некоторые внешние сигналы без использования дополнительного оборудования: сигналы внешних прерываний, счетные импульсы таймеров, аналоговые сигналы на входах АЦП. Интересно отметить возможность программного инициирования прерываний, не предусмотренную в MCS-51, однако реализованную в стенде через механизм программного управления состоянием входа внешнего прерывания INTO ADuC812. ПЗУ с резидентным программным обеспечением реализовано на кристалле

ОКЭВМ ADuC812 по технологии FLASH/EE и может быть перепрограммировано через интерфейс RS232C с обычного ПК. Новые версии резидентного программного обеспечения могут доставляться в стенд без использования специальных программаторов, а тем более новых микросхем ПЗУ – достаточно иметь лишь образ доставляемой программы в виде файла и специальную утилиту на ПК.

Некоторые устройства стенда подключены к вычислителю через периферийный расширитель, реализованный на базе ПЛИС небольшой емкости, перепрограммируемой через специальный JTAG-порт, что дает возможность при желании изменять механизмы работы с этими устройствами.

Иногда при программировании SDK-1.1 возникает необходимость сохранять программу и после выключения питания. Так как стенд создавался для массового использования студентами, такая возможность в базовой его конфигурации имеется лишь за счет замены содержимого FLASH-памяти вычислителя. Однако ее размер составляет всего 8 Кб, что не всегда достаточно для более-менее серьезных программ. К тому же, во FLASH-память ADuC812 может быть записан только один образ, т. е. либо одна программа, либо несколько программ, но скомпонованных в один файл. Это затрудняет обновление отдельной программы в наборе, так как, во-первых, для этого необходимо заново компоновать все программы из набора, во-вторых, невозможно перезаписать часть FLASH-памяти ADuC812 без стирания всего ее содержимого. Для решения этой проблемы в модификации стенда SDK-1.1/E может быть использована внешняя EEPROM емкостью 32 Кб. Во FLASH-памяти вычислителя можно поместить утилиту приема, распаковки и сохранения в EEPROM, а также извлечения и запуска исполняемых файлов. Все пользовательские программы, таким образом, можно размещать в EEPROM и загружать в ОЗУ только на время исполнения. При старте стенда на жидкокристаллическом индикаторе будет отображаться список содержащихся в EEPROM программ, и пользователь сможет с помощью клавиатуры стенда выбрать нужную. В EEPROM можно сохранять не только исполняемые файлы, но и требуемые пользовательскими программами файлы с данными, к которым они могут обращаться, пользуясь сервисами системного ПО, размещаемого во FLASH-памяти вычислителя. Иными словами, на имеющейся в SDK-1.1/E EEPROM можно организовать простейшую файловую систему в составе программного окружения. Пример такой системы показан на рис. 5.

На схеме также изображено взаимодействие с периферийными устройствами через сервисы системного ПО. Это может быть удобным, когда основной задачей не является научиться взаимодействовать с периферией, но требуется управлять ею в процессе решения иной задачи.

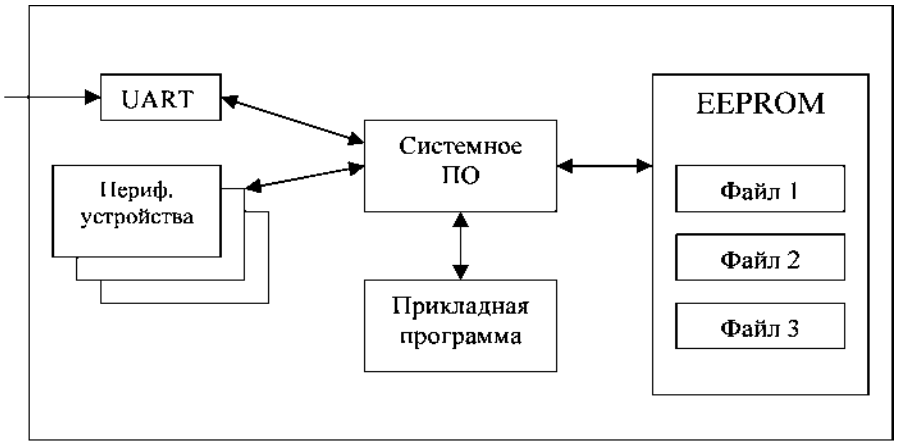

Рис. 5. Размещение программ в EEPROM стенда SDK-1.1

Стенды SDK-1.1 используются для решения широкого круга задач. В частности, можно использовать стенд в качестве контроллера приборов для автоматизации экспериментов. После запуска эксперимента может проводится автоматическое управление прибором и снятие измерений. Для проведения экспериментов не требуется наличия ПК. При этом имеется возможность переслать результаты на ПК при его подключении к стенду для дальнейшего их исследования.

1.2. Составные части комплекса SDK-1.1

1.2.1. Микроконтроллер ADuC812BS

Микроконтроллер ADuC812 (рис. 6) выпускается фирмой Analog Devices – мировым лидером в области аналоговой схемотехники. ADuC812 является сигнальным процессором и содержит в себе 12-битный АЦП со встроенным микропроцессором. Процессорное ядро ADuC812 является клоном ядра Intel MCS51.

Основные характеристики микроконтроллера:

- Рабочая частота 11.0592 МГц.
- 8-канальный 12-битный АЦП со скоростью выборок 200 К/с.
- Два 12-битных ЦАП (код-напряжение).
- Внутренний температурный сенсор.
- 640 байт программируемого E^2PROM со страничной организацией (256 страниц по 4 байта).
- 256 байт внутренней памяти данных.
- Адресное пространство 16 Мб.
- Режим управления питанием.
- Асинхронный последовательный ввод/вывод.
- Интерфейс I^2C.
- Три 16-битных таймера/счетчика и таймер WatchDog.

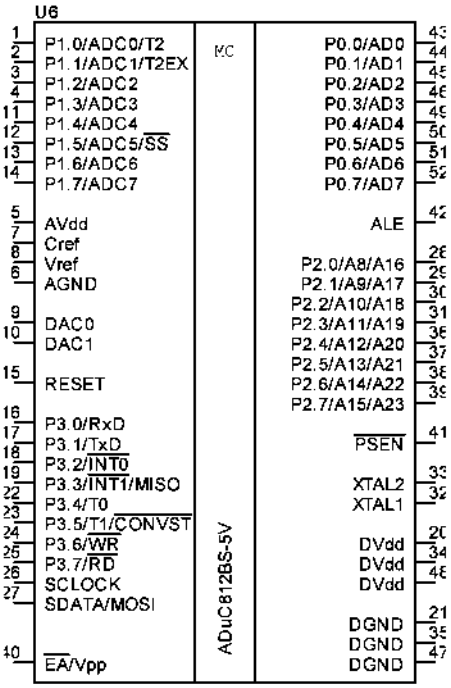

Рис. 6. Микроконтроллер ADuC812

1.2.2. ПЛИС MAX3064

В SDK-1.1 используется программируемая логическая интегральная схема (ПЛИС) семейства MAX3000A фирмы *Altera* (рис. 7). В очень упрощенном виде ПЛИС представляет собой набор макроячеек и механизм для организации связи между ними. Микросхема EPM3064A содержит 64 макроячейки. Информация о связях между макроячейками хранится в энергонезависимой памяти, находящейся внутри самой микросхемы. Для программирования EPM3064A использовался специальный САПР Max+PlusII. Электрическая принципиальная схема расширителя портов ввода/вывода была нарисована в этом САПР и преобразована в базис макроячеек ПЛИС и, далее, в конфигурационный файл, необходимый для соединения нужных логических ячеек ПЛИС. Конфигурационный файл доставляется в память ПЛИС через интерфейс JTAG (IEEE 1149.1).

В стенде SDK-1.1 MAX3064A используется как расширитель портов ввода-вывода. Микросхема MAX3064A подключена к внешней шине ADuC812. Адресная линия A19 используется как сигнал CS (chip select) для MAX3064A. ПЛИС выбирается, когда на линии A19 логическая единица. Физический адрес ПЛИС, таким образом, равен 0x80000, что соответствует восьмой странице памяти.

К ПЛИС подключены:

- клавиатура;
- ЖКИ;

- линейка светодиодов;
- звуковой излучатель;
- 16 дискретных портов ввода/вывода.

Для программиста расширитель портов представлен в виде нескольких однобайтовых регистров, находящихся в начале восьмой страницы памяти данных.

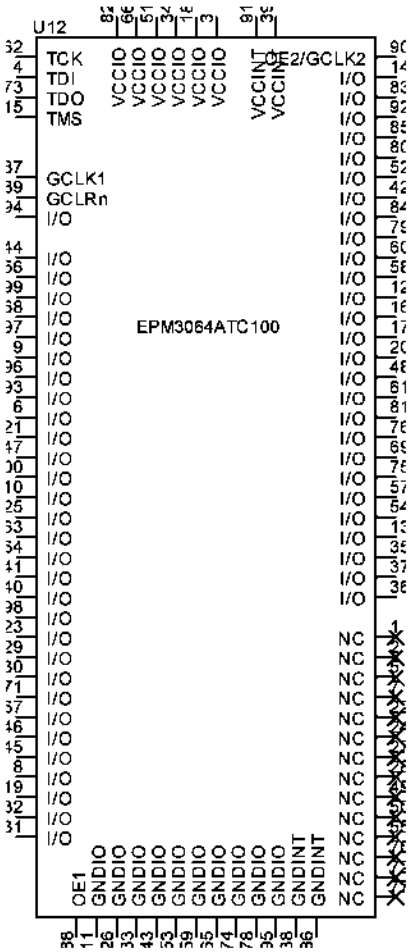

Рис. 7. ПЛИС MAX 3064A

1.2.3. Схема сброса

Схема сброса предназначена для формирования качественного сигнала RESET после включения питания, после нажатия кнопки RESET или после выключения питания. Проблема состоит в том, что при старте контроллера после включения питания или при выключении питания возможны различные переходные процессы, могущие привести к некорректному исполнению программ или порче содержимого ОЗУ. Супервизор питания (Ul) DS1813 обеспечивает формирование сигнала RESET на 150 мс, т. е. на время, достаточное для окончания всех переходных процессов.

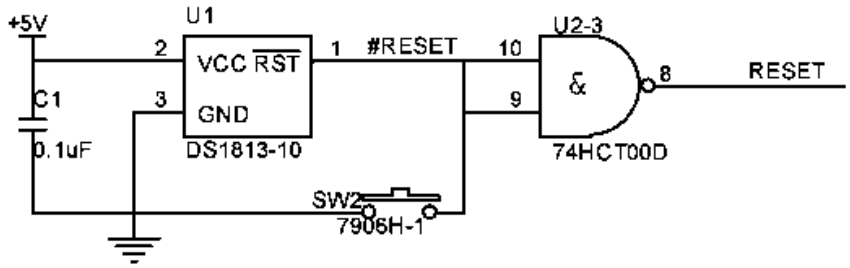

Рис. 8. Схема сброса

1.2.4. Источник питания

Схема встроенного стабилизатора питания лабораторного макета приведена на рис. 9. Переменное (15…16 В) или постоянное (9…10 В) напряжение от внешнего источника питания попадает на диодный мост U15 через разъем J4. Сердцем встроенного в SDK 1.1 источника питания является микросхема LM7805C. Эта микросхема является интегральным стабилизатором с защитой от перегрева и короткого замыкания. Выходное напряжение – 5В ± 2 %, выходной ток – до 1 А.

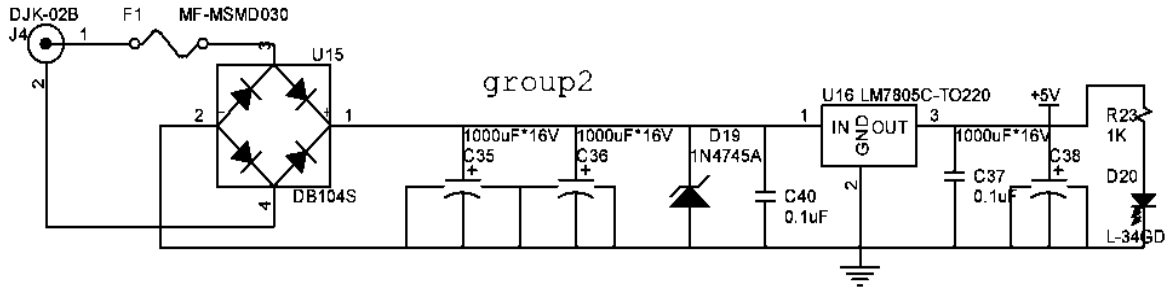

Рис. 9. Источник питания

Стабилитрон D19 (1N4745A) предназначен для защиты LM7805C и электролитических емкостей от превышения входного напряжения (напряжения пробоя стабилитрона – 16 В). Электролитические конденсаторы C35 и C36 необходимы для сглаживания пульсаций входного напряжения. Электролитический конденсатор C38 необходим для поддержки работоспособности SDK-1.1 при кратковременных пропаданиях напряжения питания. Емкости C40 и C37 необходимы для фильтрации высокочастотных помех, их использование определяется штатной схемой включения LM7805C.

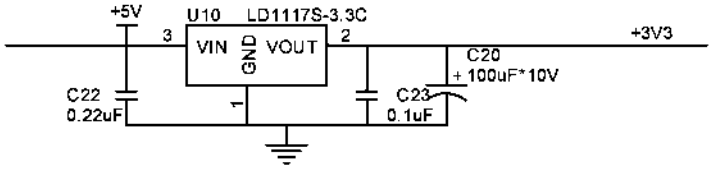

Рис. 10. Стабилизатор питания ПЛИС

Напряжение 3.3 В для питания ПЛИС формируется с помощью стабилизатора U10 (LD1117S) (рис. 10).

Фильтрующие емкости равномерно распределены по всей поверхности печатной платы. Каждый конденсатор соединяет плюс питания с корпусом. Фильтрующие емкости шунтируют высокочастотные помехи, возникающие в цепях питания 3,3 и 5 В (рис. 11).

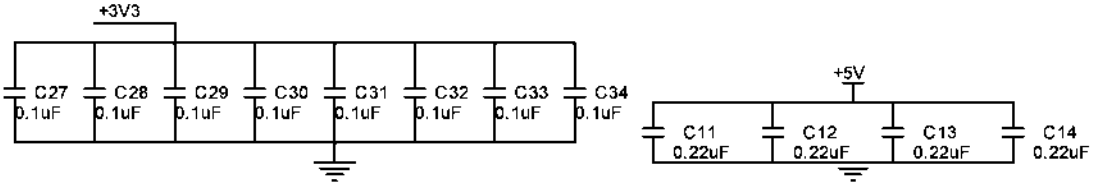

Рис. 11. Шунтирующие емкости

Шунтирование происходит из-за того, что активное сопротивление емкости тем меньше, чем выше частота сигнала.

$Xc = \frac{1}{2\pi fC}$, где Xc – активное сопротивление конденсатора; f – частота; C – емкость.

Для постоянного напряжения сопротивление конденсатора близко к бесконечности, а для переменного напряжения высокой частоты – конденсатор является резистором с низким сопротивлением.

1.2.5. Кварцевые резонаторы

Кварцевые резонаторы – устройства, использующие пьезоэлектрический эффект для возбуждения электрических колебаний заданной частоты. При совпадении частоты приложенного напряжения с одной из собственных механических частот кварцевого вибратора в приборе возникает явление резонанса, приводящее к резкому увеличению проводимости. Обладая среди резонаторов самой высокой добротностью $Q = 10^5 \ldots 10^7$ (добротность колебательного LC-контура не превышает 10^2, пьезокерамики – 10^3), кварцевые резонаторы имеют также высокую температурную стабильность и низкую долговременную нестабильность частоты.

Кварцевые резонаторы применяются в генераторах опорных частот, в управляемых по частоте генераторах, селективных устройствах: фильтрах, частотных дискриминаторах и т. д. В SDK-1.1 два кварцевых резонатора (рис. 12). Y1 служит для тактирования ADuC812 (12 МГц), а Y2 – для тактирования часов реального времени (32.768 КГц).

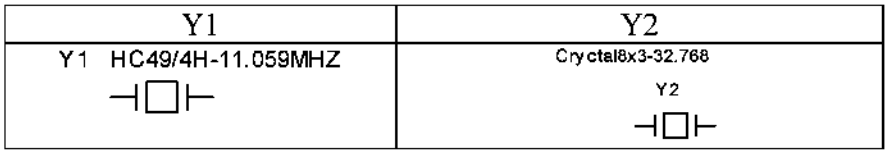

Рис. 12. Кварцевые резонаторы

1.2.6. Дискретные входы/выходы

Дискретные входы/выходы предназначены для ввода и вывода информации, представленной в двоичном виде. Сигнал на входе или выходе дискретного порта может принимать значение логического нуля или единицы. В SDK-1.1 дискретные порты выведены на разъем J3. Эти порты можно использовать для подключения модулей SDX или каких-либо других внешних устройств. Кроме этого, к дискретным входам/выходам подключены DIP-переключатели, позволяющие задавать фиксированные значения сигналов на входах. По умолчанию все входы притянуты к логической единице (через резисторы на +5 В). При замыкании переключателя SW3 на выбранном входе появляется логический нуль.

Дискретные входы/выходы не имеют гальванической изоляции. Логическому нулю соответствует 0 В, а логической единице – +5 В (уровни TTL). Нагрузочная способность дискретных портов ввода/вывода, подключенных к разъему J3, невелика, так как на разъем выведены порты ADuC812 без каких-либо дополнительных усилителей.

ADuC812 может обращаться к внешней памяти программ, внешней памяти данных или к каким-либо периферийным устройствам. При доступе к внешней памяти программ используется сигнал #PSEN (*Program Store Enable*) для чтения команд. При доступе к внешней памяти данных используются сигналы: #RD и #WR. Порт 0 и порт 2 используются в качестве шины адреса/данных при доступе к внешней памяти. В SDK-1.1 сигналы PSEN и RD объединены вместе с помощью логического элемента «И» (U2-1), что позволяет использовать единое пространство памяти программ и данных.

Так как для работы с 24-битной шиной адреса и 8-разрядной шиной данных используется только 16 выводов, в схеме поставлены регистры-защелки (latch) U3 и U7. В качестве восьмиразрядного регистра-защелки используется микросхема 74HCT73W. Положительный перепад на входе CLK приводит к запоминанию состояния входов D1...D8 и выдаче информации на выходы Q1...Q8. Активный уровень на входе CLR (логический «0») приводит к обнулению содержимого регистра-защелки. Регистры-защелки получают данные по сигналу ALE.

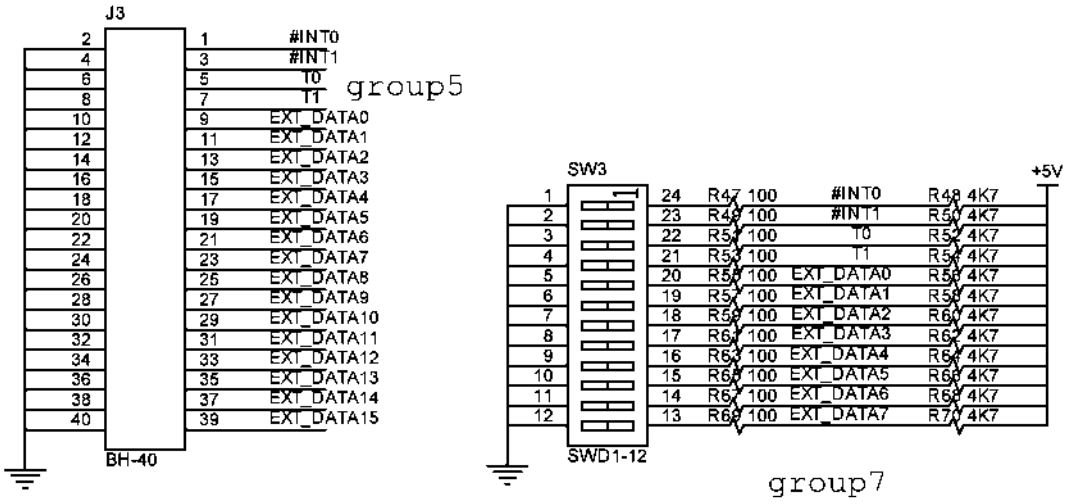

Рис. 13. Организация дискретных входов/выходов

В U3 попадает младшая часть адреса (A0...A7), в U7 – старшие 4 бита адреса (A16...A19) (рис. 15). Бит A19 используется как сигнал CS (Chip Select) для выбора микросхемы ОЗУ. ОЗУ выбирается, если на A19 логический нуль. При наличии логической единицы на A19 выбирается ПЛИС. При замыкании JP11 сигнал PSEN замыкается на корпус через резистор R22. Это переводит ADuC812 в режим загрузки ПО после аппаратного сброса.

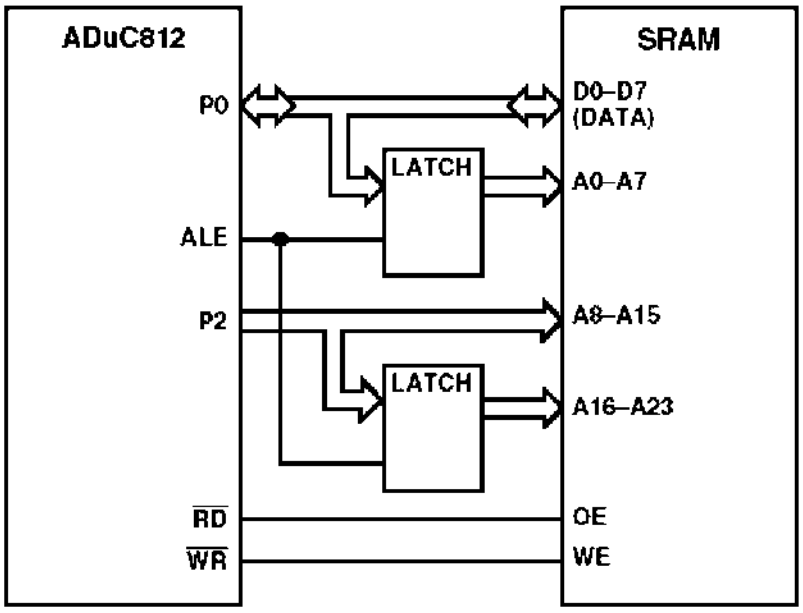

Рис. 14. Подключение ADuC812 к внешнему ОЗУ

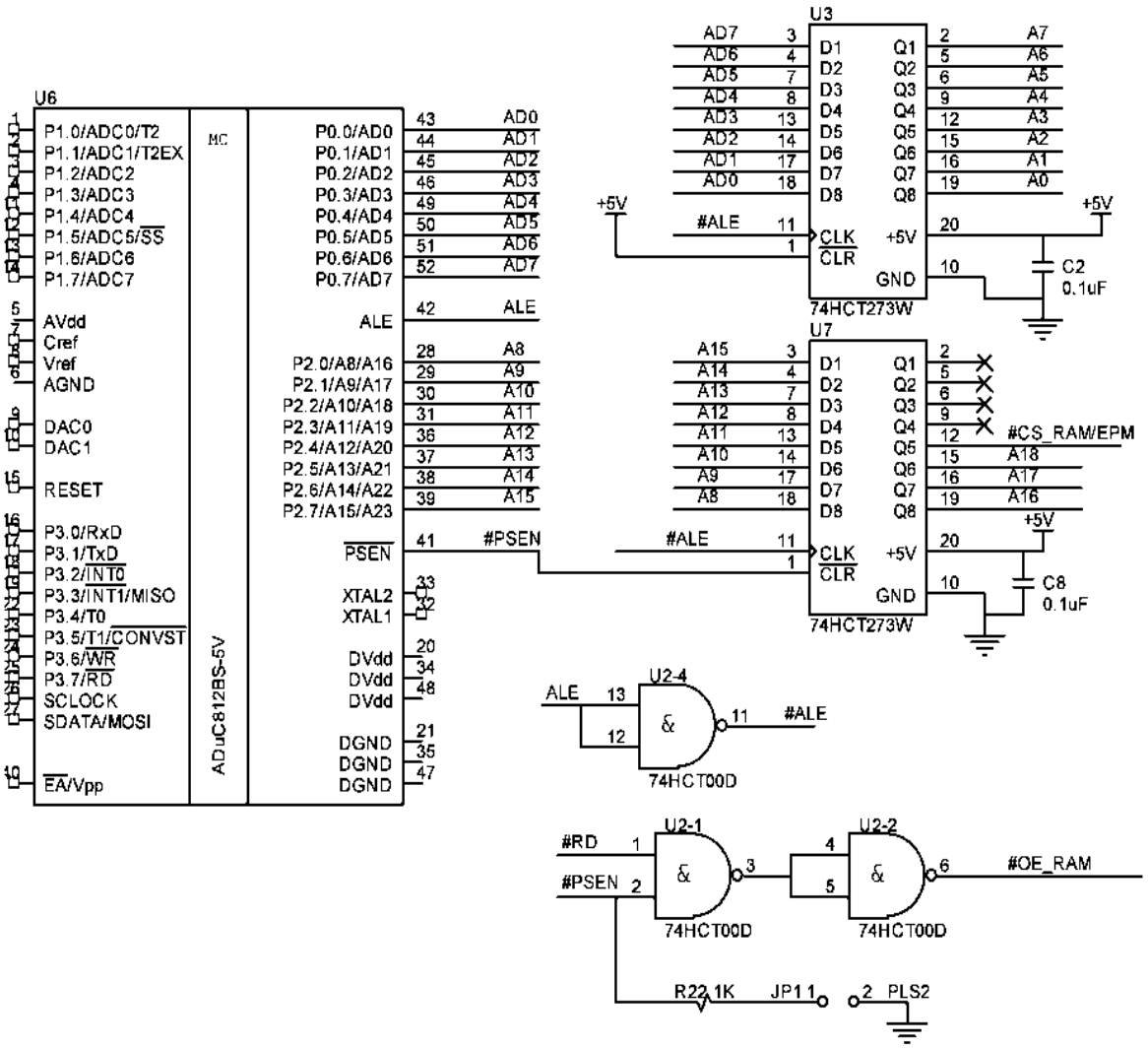

Рис. 15. Организация шин адреса/данных

SDK-1.1 может иметь две микросхемы статической памяти: КМ684000 и AS7C4096. Обе микросхемы являются микросхемами статической памяти с организацией 512Кх8 (512 Кбайт). В стенд SDK-1.1 впаивается только одна из микросхем, хотя на печатной плате есть место для установки обеих.

1.2.7. Аналоговые входы/выходы

ADuC812 имеет в своем составе 8 быстродействующих 12-разрядных АЦП и 2 12-разрядных ЦАП (выход напряжения) (рис. 16). Для коррекции зависимости параметров ЦАП и АЦП от температуры в ADuC812 встроен термодатчик. Все входы ЦАП и выходы АЦП выведены на разъем J1. Кроме того, выходы DAC0 и DAC1 можно замкнуть на входы ADC0 и ADC1 с помощью переключателя SW1.

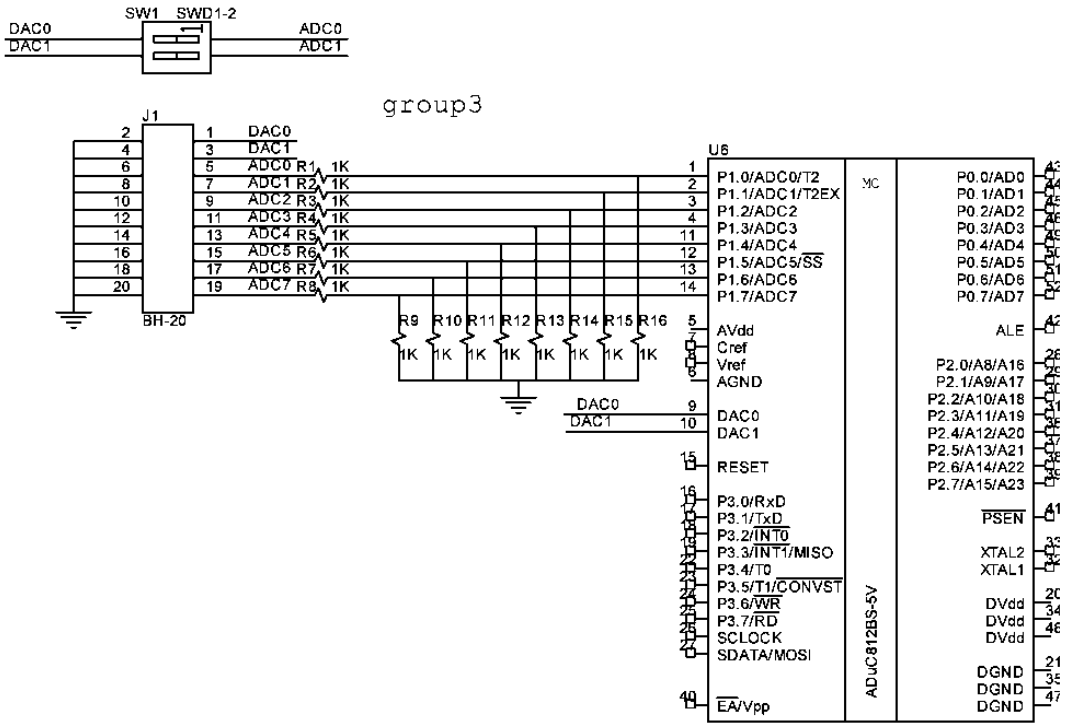

Рис. 16. Выходы АЦП и ЦАП

1.2.8. Светодиодные индикаторы

Светодиодные индикаторы подключены к расширителю портов ввода/вывода. Так как все катоды светодиодов подключены к корпусу, для зажигания светодиодов необходимо подать напряжение +5 В (лог. «1») на соответствующий анод. Резисторы R32…R43 ограничивают ток, текущий через порт ввода/вывода и светодиод. В данном случае приблизительный ток можно вычислить по закону Ома: I = U/R = 3,3/1000 = 3,3 мА. От силы тока зависит яркость горения светодиода. Если ток сделать очень большим, то порт ввода/вывода или светодиод могут выйти из строя.

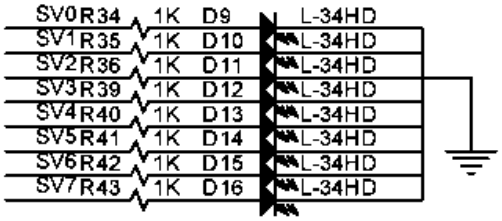

Рис. 17. Светодиодный индикатор

1.2.9. Устройства I^2C

В стенде SDK-1.1 два устройства, подключенных к шине I^2C: часы реального времени PCF8583 (U11) и EEPROM AT24C01A (U14) (рис. 18).

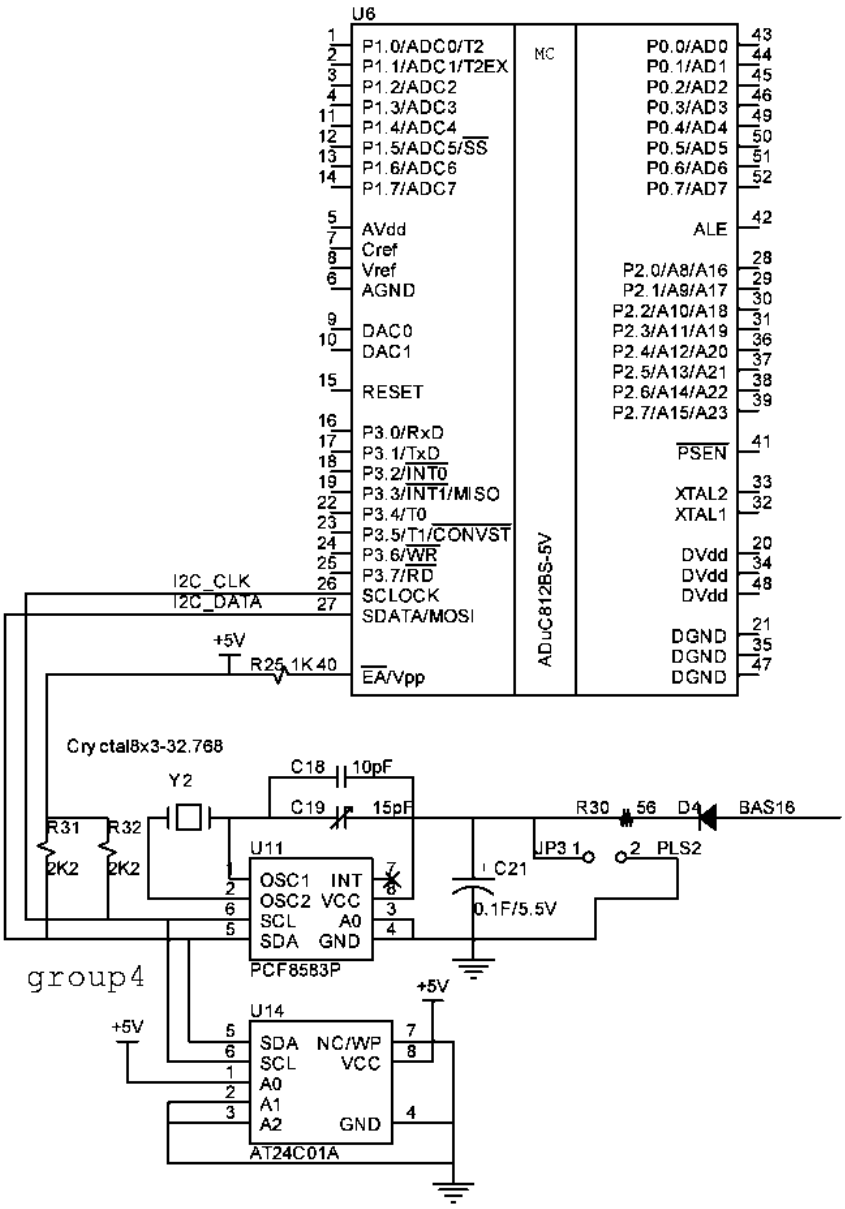

Рис. 18. Шина I^2C

Внешняя E^2PROM – перепрограммируемое электрически стираемое постоянное запоминающее устройство. Объем памяти E^2PROM, установленной в стенде SDK-1.1, составляет 128 байт (возможна установка E^2PROM большего объема, до 32 Кб). Микросхема E^2PROM взаимодействует с процессором посредством интерфейса I^2C.

Основные характеристики E^2PROM:

- возможность перезаписи до 1 млн раз;
- возможность побайтной и постраничной записи (в текущей конфигурации размер страницы составляет 8 байт).

Часы реального времени PCF8583 – часы/календарь с памятью объемом 256 байт, работающие от кварцевого резонатора с частотой 32,768 кГц.

Питание осуществляется ионистором (0.1 ф). Из 256 байт памяти собственно часами используются только первые 16 (8 постоянно обновляемых регистров-защелок на установку/чтение даты/времени и 8 – на будильник), остальные 240 байт доступны для хранения данных пользователя. Точность измерения времени – до сотых долей секунды. Взаимодействие с процессором осуществляется через интерфейс I^2C.

1.2.10. Матричная клавиатура AK1604A-WWB

Клавиатура организована в виде матрицы 4 × 4. Доступ к колонкам и рядам организован как чтение/запись определенного байта внешней памяти (4 бита соответствуют 4 колонкам, другие 4 бита – рядам). Ряды ROW1...ROW4 подключены к плюсу питания через резисторы. Это обеспечивает наличие логической единицы при отсутствии нажатия. На столбцы клавиатуры подают логический нуль. При нажатии на кнопку, происходит изменение значения сигнала на входе соответствующего ряда с единицы на нуль. Клавиатура подключена через расширитель портов на ПЛИС.

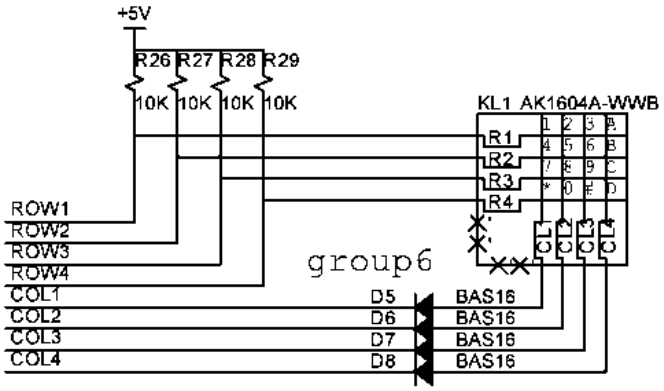

Рис. 19. Подключение клавиатуры

1.2.10. Последовательный канал

В SDK-1.1 последовательный канал гальванически развязан (рис. 20). Гальваническая изоляция или гальваническая развязка – разделение электрических цепей посредством не проводящего ток материала. Гальваническая изоляция позволяет защитить SDK-1.1 от высоких напряжений, различных наводок и подключать его к ПК во время работы.

Реализована гальваническая изоляция на базе двух оптронов U8 и U9 (TLP 181). Оптрон TLP181 состоит из светодиода (выводы 1, 3) и фототранзистора (выводы 6, 4). Если через светодиод пустить ток, то он начинает излучать свет. Свет падает на PN-переход фототранзистора и открывает его. Когда свет гаснет, фототранзистор закрывается. Гальваническая изоляция достигается как раз за счет того, что между двумя элементами оптрона нет никакой связи, кроме оптической.

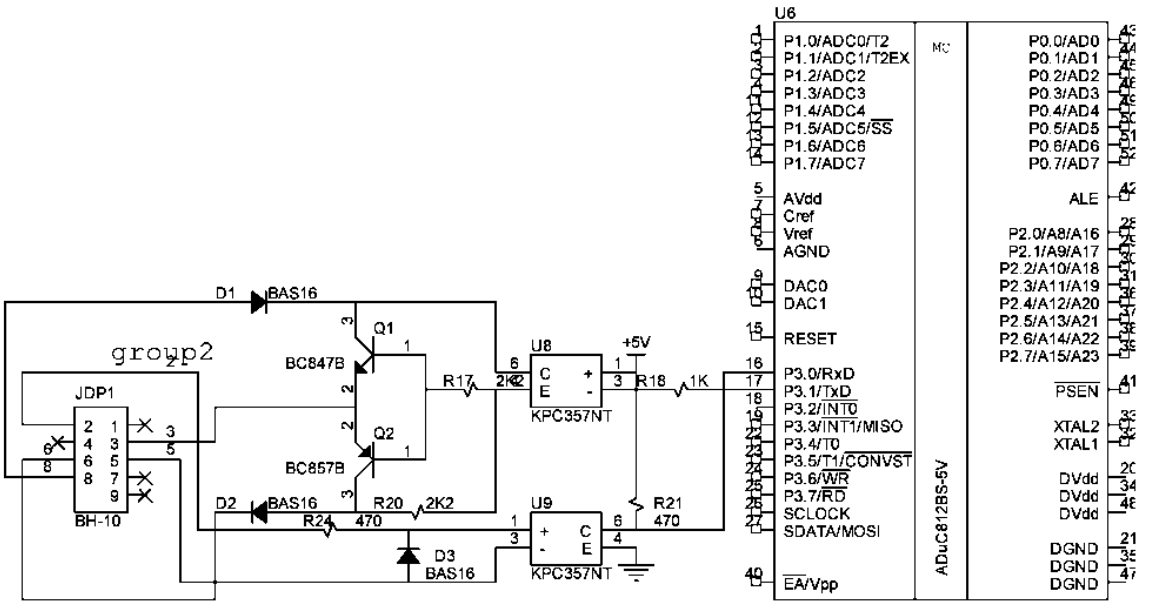

Рис. 20. Гальванически изолированный последовательный интерфейс SDK-1.1

Выход передатчика последовательного канала TxD попадает на катод оптрона U8. Далее сигнал попадает на двухтактный эмиттерный повторитель, сделанный на транзисторах Q1 и Q2. Сигнал TD снимается с эмиттеров транзисторов и далее попадает на ПК. Эмиттерный повторитель питается от двух сигналов RS232, поступающих от ПК: +Un получается от CTS, –Un получается от сигнала DSR. Для того чтобы схема заработала, необходимо подать –12 В на выход DTR ПК и +12 В – на выход RTS ПК.

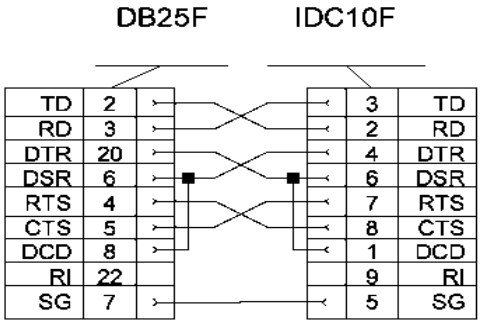

Рис. 21. Подключение сигналов последовательного порта

Схема работает так: когда на базы транзисторов попадает положительный сигнал с оптрона, открывается транзистор Q1 и на его эмиттере появляется положительный сигнал. Когда на выходе оптрона отрицательный сигнал (оптрон закрыт), то открывается транзистор Q2. Так как выход TxD подключен к катоду оптрона, сигнал в оптроне инвертируется, преобразуя TTL-сигнал в сигнал RS232. Входной сигнал RD попадает через защитный диод D2 прямо на оптрон. В оптроне сигнал инвертируется и подается на TTL-вход приемника RxD.

При программировании последовательного канала под *Windows*, нужного состояния сигналов RTS и DTR можно добиться с помощью двух строк кода на языке С:

```
EscapeCommFunction( Port, SETRTS);
EscapeCommFunction( Port, CLRDTR).
```

1.2.12. Жидкокристаллический дисплей

Жидкокристаллический индикатор (ЖКИ) работает в текстовом режиме (2 строки по 16 символов), имеет подсветку (цвет желто-зеленый). Основные характеристики ЖКИ:

- габариты: 80 × 36×13,2 мм;
- активная область 56,21 × 11,5 мм;
- размеры точки 0,56 × 0,66 мм; размеры символа 2,96 × 5,56 мм;
- встроенный набор 256 символов (ASCII + кириллица);
- генератор символов с энергозависимой памятью на 8 пользовательских символов.

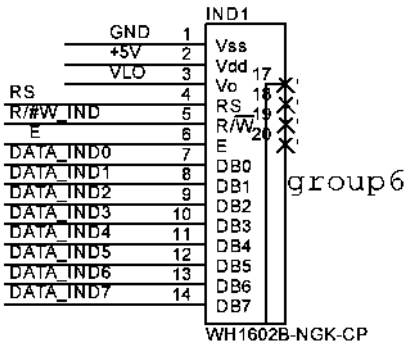

Рис. 22. ЖКИ WH1602B-YGK-CP

1.2.13. Звукоизлучатель

В SDK-1.1 используется пьезоэлектрический звукоизлучатель HPA17A (Z1). Выходы SND0...SND2 подключены к расширителю портов на базе ПЛИС.

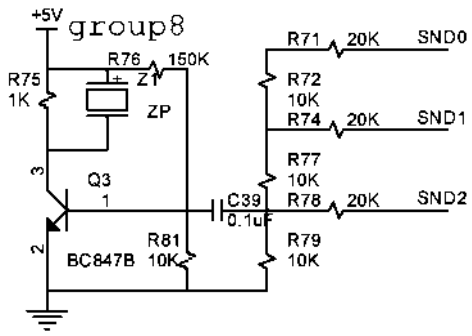

Рис. 23. Звукоизлучатель

2. Порядок выполнения работы

1. Изучите структурную и электрическую принципиальную схемы стенда SDK-1.1. Воспользуйтесь комплектом документации к SDK-1.1 и данным пособием.
2. Изучите описание инструментального программного обеспечения. Воспользуйтесь руководством пользователя SDK-1.1 и справочной системой *Keil Software*.
3. Изучите структуру STARTUP.A51 и простейшего примера программы на языке С.

3.1. Файл STARTUP.A51

```
;STARTUP.A51: Этот код исполняется после рестарта процессора
;Для трансляции этого файла используйте ассемблер A51
;со следующей командной строкой: A51 STARTUP.A51
;Для сборки вашего проекта и модифицированного вами файла
;STARTUP.OBJ используйте линкер BL51 со следующей командной
;строкой.
; BL51 <ваши объектные файлы>, STARTUP.OBJ <опции>
;Определяемая пользователем инициализация памяти после
;рестарта.
;С этими константами будет производиться инициализация памя-
ти:
;Абсолютный адрес начала памяти IDATA всегда 0
IDATALEN        EQU 8OH;Длина памяти IDATA в байтах.
XDATASTART      EQU OH ;Абсолютный адрес начала памяти XDATA
XDATALEN        EQU OH ; Размер памяти XDATA
PDATASTART      EQU OH ; Абсолютный адрес памяти PDTATA
PDATALEN        EQU OH ; Размер памяти PDATA
;Инициализация стека
;Следующие константы определяют положение стека, необходимого
;для работы реентерабельных функций
;Модель SMALL
IBPSTACK        EQU 0 ;  = 1, если используется стек для SMALL .
IBPSTACKTOP EQU OFFH+1; установка стека в старшую позицию +1.
;Модель LARGE
XBPSTACK        EQU 0;  = 1, если используется стек для LARGE
XBPSTACKTOP     EQU OFFFFH+1; установка стека в старшую
;позицию +1.
;Модель COMPACT
PBPSTACK        EQU 0;  = 1, если используется стек для COMPACT
PBPSTACKTOP     EQU OFFFFH+1, ; установка стека в старшую
;позицию +1.
;Определение PPAGE для модели Compact с 64 Кбайт xdata RAM
;Эти константы определяют положение переменных типа pdata.
PPAGEENABLE     EQU 0 ;  = 1, если pdata используется.
PPAGE           EQU 0 ;  определяет количество страниц PPAGE.
NAME                ?C_STARTUP
```

```
?C_C51STARTUP  SEGMENT   CODE
?STACK         SEGMENT   IDATA
               RSEG      ?STACK
               DS        1
               EXTRN CODE (?C_START)
               PUBLIC  ?C_STARTUP
; Установите 0, если вы компилируете программу для симулятора
               CSEG    AT  O
; Установите 2100H, если вы компилируете программу для
SDK1.1
               CSEG    AT  2100H
?C_STARTUP:    LJMP    STARTUP1
               RSEG    ?C_C51STARTUP
STARTUP1:
; Очистка IDATA
IF IDATALEN <>0
               MOV R0,#IDATALEN-1
               CLR A
IDATALOOP:     MOV @R0,A
               DJNZ    RO,IDATALOOP
ENDIF
; Очистка XDATA
IF XDATALEN <>0
               MOV DPTR,#XDATASTART
               MOV R7,#LOW(XDATALEN)
    IF (LOW(XDATALEN)) <>0
               MOV R6,#(HIGH XDATALEN)+1
    ELSE
               MOV R6,#HIGH(XDATALEN)
    ENDIF
               CLR A
XDATALOOP:     MOVX  @DPTR,A
               INC DPTR
               DJNZ    R7,XDATALOOP
               DJNZ    R6,XDATALOOP
ENDIF
IF PPAGEENABLE <>0
               MOV P2,#PPAGE
ENDIF
; Очистка  памяти  pdata
IF  PDATALEN <>0
               MOV  RO,#PDATASTART
               MOV  R7,#LOW(PDATALEN)
               CLR  A
PDATALOOP:     MOVX @R0,A
               INC  RO
               DJNZ    R7,PDATALOOP
```

```
ENDIF
; стек для модели SMALL
IF IBPSTACK <>0
EXTRN          DATA (?C_IBP)
               MOV ?C_IBP,#LOW IBPSTACKTOP
ENDIF
; стек для модели LARGE
IF XBPSTACK <>0
EXTRN          DATA (?C_XBP)
               MOV ?C_XBP,#HIGH XBPSTACKTOP
               MOV ?C_XBP+1,#LOW XBPSTACKTOP
ENDIF
; стек для модели COMPACT
IF PBPSTACK <>0
EXTRN          DATA (?C_PBP)
               MOV ?C_PBP,#LOW PBPSTACKTOP
ENDIF
; Установка указателя стека
               MOV SP,#?STACK-1
; Переход к программе инициализации памяти и далее к main()
               LJMP    ?C_START
END
```

3. Простейшая программа на языке Си

Следующая программа на языке Си для стенда SDK-1.1 инициализирует последовательный канал на скорости 9600 бод и выдает в него строчку «SDK 1.1». Текст программы:

```
#include «ADuC812.h»    // Включение в текст описания регистров
// специального назначения ADuC812
void main(void)
{
TH1  = OxFD;        // Скорость 9600
TMOD = 0x2 0;       // Таймер 1 в режиме autoreload
TCON = 0x4 0;       // Запуск таймера 1
SCON = 0x5 0;       //8 bit UART, разрешение приема
PCON& =  0x7F;      // Отключение дублирования скорости, ус-
                    //тановленной в TH1
EA    = 0;          // Запрещение прерываний
TI   = 0;           // Обнуление флага завершения посылки
SBUF = 'S';         // Инициация посылки символа «S»
while(!T1);         // Ожидание завершения посылки
TI   = 0; SBUF = 'D' ; while( !TI );
TI   = 0; SBUF = 'K'; while( !TI );
TI   = 0; SBUF = ' 1' ; while(!TI);
TI   = 0;
```

```
SBUF
while( !TI );
TI   = 0; SBUF = ' 1' ; while( !TI );
while ( 1 );        // Завершение программы
}
```

Необходимо обратить внимание на последний оператор в телефункции *main*(). Бесконечный цикл *while* (1) играет роль оператора завершения программы. Так как SDK-1.1 не находится под управлением операционной системы, то простой выход из пользовательской программы приведет к неконтролируемой выборке команд микроконтроллером из памяти, что может вызвать нежелательные последствия и даже привести к выходу стенда из строя. Поэтому рекомендуется все программы либо «завершать» бесконечным циклом, либо строить их таким образом, чтобы они работали по бесконечному алгоритму.

Для трансляции программы необходим компилятор C51 фирмы *Keil Software*.

1. Изучите структуру и назначение объектных файлов, OMF, HEX.

2. Изучите переменные окружения, необходимые для нормальной работы компилятора.

3. Разберитесь в командных файлах для трансляции программ и для загрузки программ в SDK-1.1.

Для трансляции программы используется пакет *Keil Software*. *Keil Software* поддерживает все стадии разработки приложения: создание исходного файла на С или Ассемблере, трансляцию, исправление ошибок, линкование объектных файлов, тестирование приложения.

В пакете *Keil Software* содержатся следующие средства разработки для микроконтроллера 8051:

- C51 – компилятор С;
- макроассемблер A51;
- динамический загрузчик/компоновщик BL51;
- конвертер объектных файлов OC51;
- конвертер объектных и HEX-файлов OH51;
- менеджер библиотек LIB 51;
- симулятор dScope-51 (для *Windows*);
- отладчик/компилятор *μ Vision*/51 (для *Windows*);
- операционная система реального времени (*Real-Time Operating System* – RTX).

4. Оттранслируйте пример простейшей программы и загрузите ее в SDK-1.1. Изучите назначение каждой строки исходного текста в программе. Измените программу и попробуйте ее оттранслировать и загрузить в SDK-1.1.

Для трансляции своей программы запустите командный файл make.bat.

Пример командного файла:

```
@echo off
rem Стираем загрузочный модуль
del test.hex
rem Транслируем стартовый модуль
A51 startup.a51
rem Если ассемблер вернул код ошибки, то завершаем командный файл
if errorlevel 1 goto ERROR
rem Транслируем нашу программу на языке C
C51 test.c   CODE LARGE WL(2)
rem Если ассемблер вернул код ошибки, то завершаем командный файл
if errorlevel 1 goto ERROR
rem Формирование файла в формате OMF из объектных модулей.
rem Сегмент кода начинается с адреса 0x2100, сегмент данных с адреса 0x7000
BL51 startup.obj, test.obj to test CODE(02100H) XDATA(07000H)
rem Преобразование файла в формате OMF в формат HEX
OH51 test
rem Добавляем стартовый адрес (0x2100) в HEX-файл
if exist test.hex tl67b 0x2100 0x0 addhexstart exhibit.hex bye
goto OK
:ERROR
echo------ ERROR-------------
:OK
```

5. Скомпилируйте и загрузите свою программу в симулятор *dScope Debugger*. Изучите процесс отладки программ в симуляторе.

Загрузка программы в SDK-1.1.

Подключите SDK-1.1 к ПК с помощью кабеля RS-232. Включите питание стенда.

Загрузочный модуль программы в формате HEX передается в SDK-1.1 по последовательному каналу RS-232 с помощью программы T167b или T2. Для запуска процесса загрузки запустите командный файл load.bat. Далее, перед каждым запуском загрузчика, нажимайте кнопку RESET на стенде, чтобы стенд переходил из режима выполнения вашей программы в режим загрузчика.

Пример командного файла:

```
tl67b 2 12 openchannelrts loadhex + test.hex bye
```

4. Содержание отчета

Для написания отчета необходимо изучить литературу, предложенную в конце раздела. Отчет должен содержать:

- описание архитектуры стенда;
- аппаратные блоки стенда;
- распределение памяти;
- описание инструментария;
- описание опций компилятора;
- описание команд Т167Ь и Т2;
- описание возможностей симулятора;
- исходный текст программы с комментариями.

5. Контрольные вопросы

1. Нарисуйте структуру стенда SDK-1.1.
2. Каковы функции ПЛИС в SDK-1.1?
3. Как изменить адрес загрузки исполняемой программы?
4. Зачем ставить while(l); в конце программы?
5. Каково назначение STARTUP.A51?
6. Что такое XDATA?
7. Что такое SBUF?
8. Зачем нужны флаги TI и RI?
9. Зачем нужно запрещать прерывания?
10. Какова разрядность ADuC812?
11. Сколько ОЗУ доступно программе в нулевой странице памяти?
12. Какие функции исполняет программа Т167Ь?
13. Каково назначение загрузчика/компоновщика BL51?
14. Зачем нужна программа OH51?
15. Что такое HEX?
16. Что такое RTC?
17. Зачем нужен EEPROM?
18. Что такое I^2C?
19. Какие переменные окружения нужно установить для обеспечения нормальной работы пакета *Keil Software*?
20. Какие функции есть у симулятора *dScopeDebugger*?

Лабораторная работа № 10
УПРАВЛЕНИЕ СВЕТОДИОДАМИ И ПОСЛЕДОВАТЕЛЬНЫМ ИНТЕРФЕЙСОМ В ЛАБОРАТОРНОМ СТЕНДЕ SDK 1.1

Цели работы: изучить архитектуру блока последовательного канала лабораторного стенда SDK-1.1, схему включения светодиодов; разработать программы управления последовательным интерфейсом и светодиодами; просимулировать программы в отладчике-симуляторе; загрузить и выполнить программы на лабораторном стенде.

1. Методические указания к работе

1.1. Управление светодиодными индикаторами

Управление светодиодным индикатором осуществляется с использованием программируемой логической интегральной схемы (ПЛИС) MAX3064A, входящей в состав лабораторного макета. Управление осуществляется путем помещения нужной информации в регистры ПЛИС. За светодиоды отвечает регистр SV ПЛИС. Он находится по адресу 080007H, значение регистра после сброса 00000000B. Каждый бит регистра управляет соответствующим светодиодом (0 разряд светодиодом 0, 7 разряд светодиодом 7).

Для доступа к регистрам ПЛИС нужно переключить страничный регистр DPP на 8-ю страницу памяти. Адреса регистров внутри страницы находятся в диапазоне от 0 до 7. Доступ к регистрам возможен через указатель: `unsigned char xdata *regnum.`

Ниже приведен пример функций для доступа к регистрам ПЛИС.

```
#define MAXBASE   8   // Страница памяти, в которую
/отображаются регистры ПЛИС
/************************************************
WriteMAX     - Запись байта в регистр ПЛИС
regnum       - адрес регистра ПЛИС
val          - записываемое значение
результат    - нет
************************************************/
void WriteMax(unsigned char xdata *regnum, unsigned char val)
  {
     unsigned char oldDPP = DPP;

     DPP     = MAXBASE;
     *regnum = val;
     DPP     = oldDPP;
  }
```

```
/************************************************
ReadMAX      - Чтение байта из регистра ПЛИС
regnum       - адрес регистра ПЛИС
результат    - прочитанное значение
************************************************/
unsigned char ReadMax(unsigned char xdata *regnum)
  {
     unsigned char oldDPP = DPP;
     unsigned char val    = 0;
     DPP = MAXBASE;
     val = *regnum;
     DPP = oldDPP;
     return val;
  }
```

Необходимо помнить, что при переключении страниц становятся недоступными все данные, размещенные в странице 0.

Для того чтобы избежать проблем со страничным регистром DPP, нужно использовать специальные функции для доступа к ПЛИС, которые перед началом работы с регистрами ПЛИС будут запоминать старое значение страничного регистра, а по окончании работы возвращать его обратно.

Нужно следить, чтобы передаваемые в регистры ПЛИС значения хранились во внутренней памяти микроконтроллера (DATA, IDATA). Убедиться, что передаваемая информация не содержится во внешней памяти контроллера (XDATA), достаточно легко, так как для доступа к внешней памяти в микроконтроллерах семейства MCS51 используется регистр DPTR, нужно просто просмотреть листинг программы и убедиться в том, что для доступа к переменным компилятор не использует DPTR.

1.2. Управление последовательным интерфейсом

Последовательный интерфейс предполагает для передачи данных в одном направлении единственную сигнальную линию, по которой информационные биты передаются друг за другом последовательно. При этом скорость изменения передатчиком состояния линии должна равняться скорости распознавания состояний приемником. Эта скорость измеряется в *бодах* (*baud*) – количестве изменений состояния линии за одну секунду. В простейшем случае в линии имеется всего два состояния сигнала, т. е. одним состоянием кодируется один бит, и тогда скорость изменения состояния в бодах совпадает со скоростью передачи двоичной информации, определяемой количеством передаваемых за секунду бит информации, *bps* (*bits per second*, бит/с). Однако, при использовании других методов модуляции возможны несколько состояний сигнала, что позволяет одним состоянием кодировать сразу несколько

передаваемых бит, и здесь скорость передачи данных *bps* превышает скорость изменения сигнала *baud*.

Обмен данными может быть:

1) дуплексным – предполагает прием и передачу данных одновременно;
2) полудуплексным – данные передаются в одном направлении с возможностью смены направления;
3) симплексным – данные передаются только в одном направлении.

Передача может осуществляться в синхронном и асинхронном режимах. Синхронный режим предполагает наличие средств синхронизации передатчика и приемника. Как правило, для синхронизации используют специальную линию для передачи тактовых импульсов. Информация в канале данных считывается приемником только в те моменты, когда на линии синхронизации сигнал активный.

В асинхронном режиме посылке очередного байта информации предшествует специальный старт-бит, сигнализирующий о начале передачи (обычно логический «0»). Затем следуют биты данных (их обычно 8), за которыми может следовать дополнительный бит (его наличие зависит от режима передачи, обычно этот бит выполняет функцию контроля четности). Завершается посылка стоп-битом (логическая «1»), длина которого (длительность единичного состояния линии) может соответствовать длительности передачи 1, 1.5 («полтора стоп-бита») или 2 бит. Стоп-бит гарантирует некоторую выдержку между соседними посылками, при этом пауза между ними может быть сколь угодно долгой (без учета понятия «тайм-аут»). Для асинхронного режима предусмотрен ряд стандартных скоростей обмена: 50, 75, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600 и 115200 bps.

В большинстве схем, содержащих интерфейс RS-232C, данные передаются асинхронно, т. е. в виде последовательности пакета данных. Каждый пакет содержит один символ кода ASCII, причем информация в пакете достаточна для его декодирования без отдельного сигнала синхронизации.

Символы кода ASCII представляются семью битами, например буква A имеет код 1000001. Чтобы передать букву A по интерфейсу RS-232C, необходимо ввести дополнительные биты, обозначающие начало и конец пакета. Кроме того, желательно добавить лишний бит для простого контроля ошибок по паритету (четности).

Наиболее широко распространен формат, включающий в себя один стартовый бит, один бит паритета и два стоповых бита. Начало пакета данных всегда отмечает низкий уровень стартового бита. После него следует 7 бит данных символа кода ASCII. Бит четности содержит «1» или «0» так, чтобы общее число единиц в 8-битной группе было нечет-

ным. Последним передаются два стоповых бита, представленных высоким уровнем напряжения. Эквивалентный ТТЛ-сигнал при передаче буквы А показан на рис. 1.

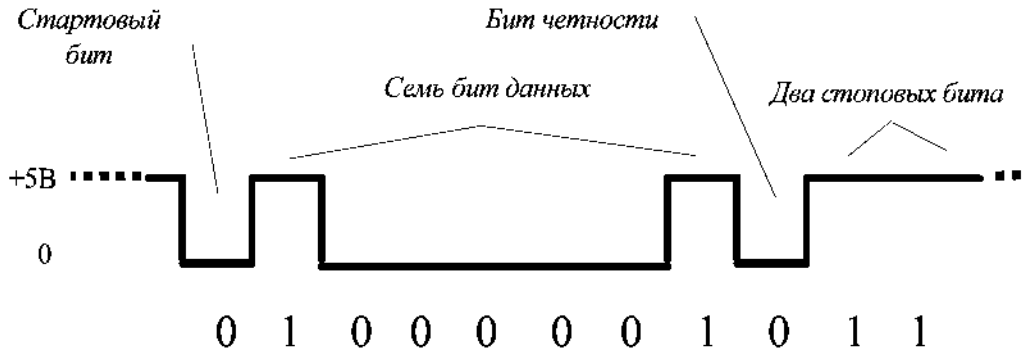

Рис. 1. Представление кода буквы А сигнальными уровнями ТТЛ

Таким образом, полное асинхронно передаваемое слово состоит из 11 бит (фактически данные содержат только 7 бит) и записывается в виде 01000001011.

Используемые в интерфейсе RS-232C уровни сигналов отличаются от уровней сигналов, действующих в компьютере. Логический «0» (SPACE) представляется положительным напряжением в диапазоне от +3 до +15 В, логическая «1» (MARK) – отрицательным напряжением в диапазоне от –3 до –15 В. На рис. 1 показан сигнал в том виде, в каком он существует на линиях TXD и RXD интерфейса RS-232C.

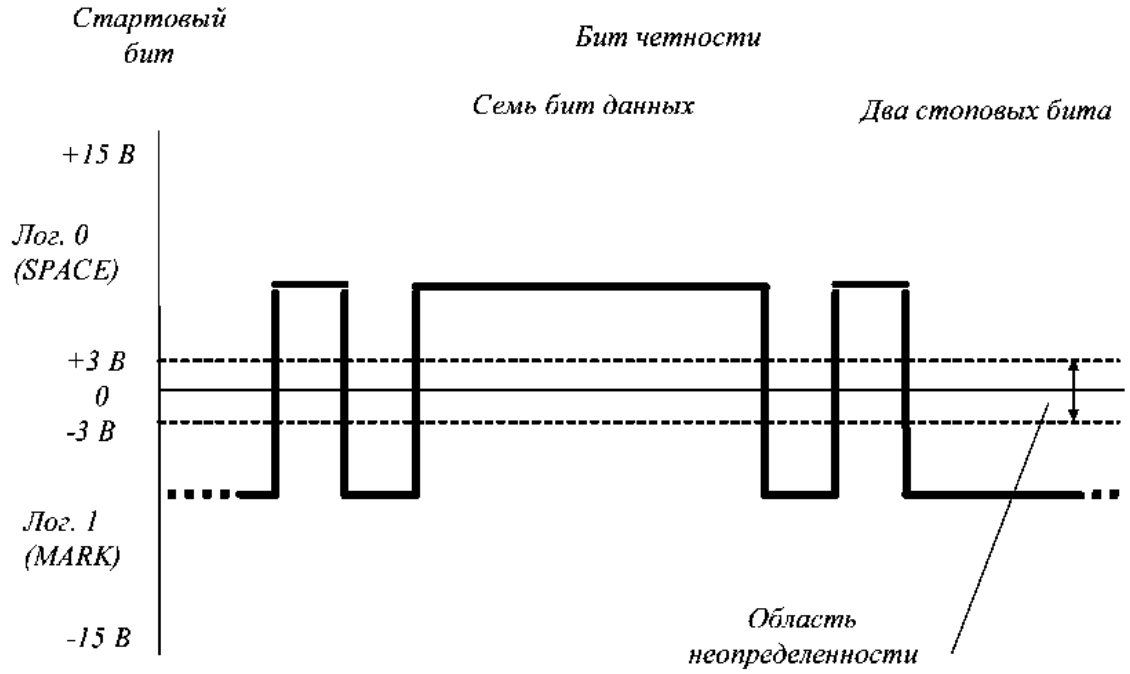

Рис. 2. Вид кода буквы А на сигнальных линиях TXD и RXD

Сдвиг уровня, т. е. преобразование ТТЛ-уровней в уровни интерфейса RS-232C и наоборот, производится специальными микросхемами драйвера линии и приемника линии. На рис. 3 представлен интерфейс RS-232C, реализованный в лабораторном стенде SDK1.1. Микросхема U17 осуществляет сдвиг уровней TTL – RS232C для сигналов TXD и RXD. Микросхемы U8 и U9 обеспечивают гальваническую развязку сигналов, поступающих на внешний разъем.

1.3. Особенности последовательного интерфейса микроконтроллеров семейства MCS51

Последовательный порт в контроллерах MCS51 позволяет осуществлять последовательный дуплексный ввод/вывод в синхронном и асинхронном режимах с разными скоростями обмена. Помимо обычного ввода/вывода, в нем предусмотрена аппаратная поддержка взаимодействия нескольких микроконтроллеров.

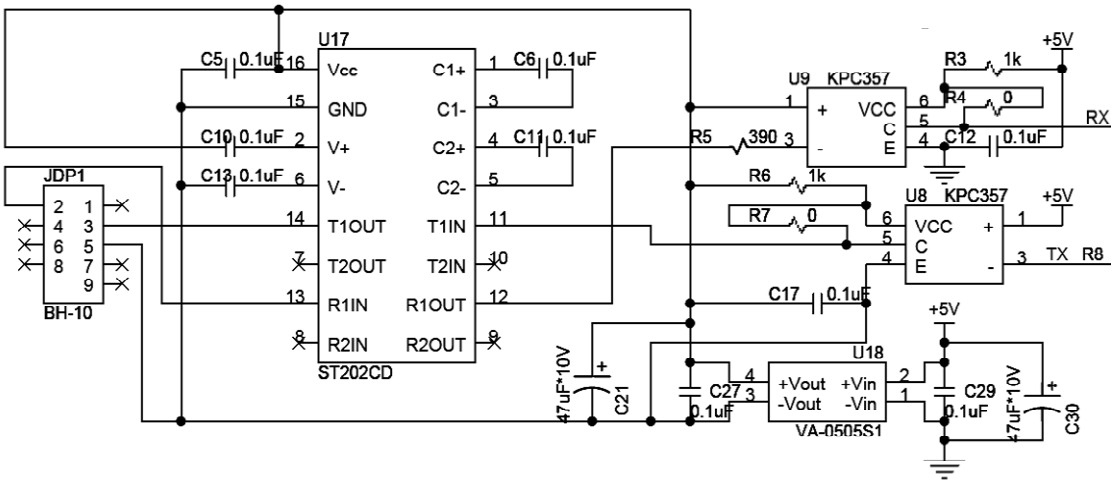

Рис. 3. Схема интерфейса RS-232C

Схематически контроллер порта представлен на рис. 4. Как видно из рисунка, регистр SBUF, доступный пользователю для чтения и записи, представляет собой физически два регистра, в один из которых можно только записывать, а из другого – читать. Контроллер позволяет дуплексный обмен данными, т. е. одновременно может передавать и принимать информацию по линиям TxD (P3.1) и RxD (P3.0), соответственно.

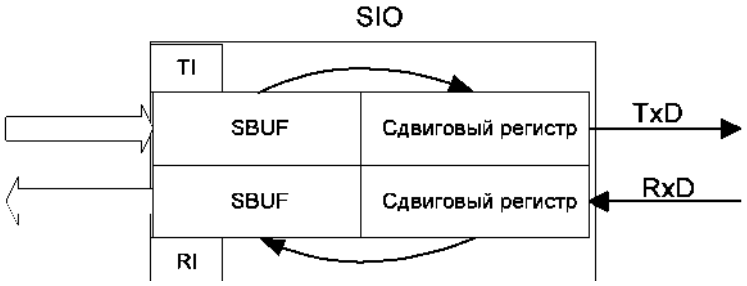

Рис. 4. Контроллер последовательного порта

Передача инициируется записью в SBUF байта данных. Этот байт переписывается в сдвиговый регистр, из которого пересылается бит за битом. Как только байт будет переслан целиком, устанавливается флаг TI, сигнализирующий о том, что контроллер готов к передаче очередного байта.

Прием осуществляется в обратном порядке: после начала процесса приема принятые биты данных последовательно сдвигаются в сдвиговом регистре, пока не будет принято их установленное количество, затем содержимое сдвигового регистра переписывается в SBUF и устанавливается флаг RI. После этого возможен прием следующей последовательности бит. Необходимо отметить, что запись принятого байта из сдвигового регистра в SBUF происходит только при условии, что RI = 0, поэтому при считывании пользовательской программой очередного байта из SBUF ей необходимо сбрасывать этот флаг, иначе принятый в сдвиговый регистр, но не записанный в SBUF, следующий байт будет безвозвратно утерян.

Контроллер последовательного порта может работать в одном из четырех режимов (один синхронный и три асинхронных), различающихся скоростями обмена и количеством передаваемых/принимаемых бит:

- **Режим 0** (синхронный): данные передаются и принимаются через RxD, TxD является синхронизирующим (выдает импульсы сдвига). Скорость в этом режиме фиксирована (1/12 частоты тактового генератора).
- **Режим 1** (8 бит данных, асинхронный, переменная скорость). Передаются 10 бит: старт-бит, 8 бит данных (SBUF) и стоп-бит.
- **Режим 2** (9 бит данных, асинхронный, фиксированная скорость). Передаются 11 бит: старт-бит, 8 бит (SBUF), бит TB8/RB8 (посылка/прием, соответственно) и 1 стоп-бит. Биты TB8 и RB8 содержатся в регистре SCON (биты 3 и 2, соответственно), первый устанавливается программно, а второй содержит 9-й бит принятой комбинации. С их помощью можно организовать, например, контроль четности или обмен с двумя стоп-битами.
- **Режим 3** (9 бит данных, асинхронный, переменная скорость). То же, что и режим 2, только скорость обмена переменная.

Режим контроллера, а также другие параметры его работы задаются в регистре SCON (098h) (рис. 5). Описание битов регистра приведено в табл. 1.

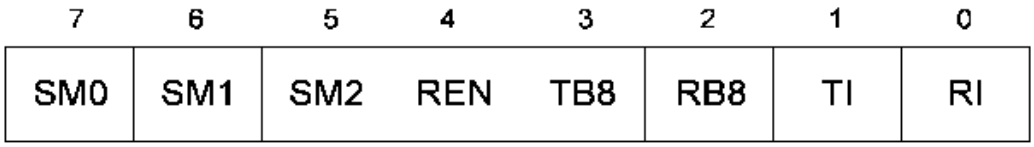

Рис. 5. Регистр SCON микроконтроллера

Таблица 1

Назначение битов регистра SCON

Бит		Описание
SM0	SM1	Режим последовательного обмена (Serial Mode)
0	0	Режим 0
0	1	Режим 1
1	0	Режим 2
1	1	Режим 3
SM2		Включает поддержку взаимодействия нескольких микроконтроллеров, объединенных в сеть. Если SM2 = 1, то при приеме в режимах 2 и 3 RI не устанавливается в том случае, если принятый 9-й бит (RB8) равен 0. При приеме в режиме 1 RI не устанавливается, если не был принят правильный стоп-бит. В режиме 0 SM2 должен быть равен 0
REN		(Receiver Enable) Если установлен, то разрешен прием данных. Устанавливается и сбрасывается программно
TB8		(Transmitter Bit 8) 9-й бит посылаемых данных в режимах 2 и 3. Устанавливается и сбрасывается программно
RB8		(Receiver Bit 8) 9-й бит принимаемых данных в режимах 2 и 3. В режиме 1, если SM2 = 0, в RB8 записывается принятый стоп-бит (точнее то, что было принято в момент приема стоп-бита). В режиме 0 не используется
TI		(Transmitter Interrupt) Флаг завершения посылки. Устанавливается аппаратно по завершении посылки 8-го бита данных в режиме 0 или стоп-бита в других режимах. При ES = 1 (бит 4 регистра IEN0(0A8h)) происходит прерывание (вектор 023h), когда TI устанавливается контроллером в 1. Должен быть сброшен программно
RI		(Receiver Interrupt) Флаг завершения приема. Устанавливается аппаратно по завершении приема 8-го бита данных в режиме 0 или стоп-бита в других режимах. При ES = 1 (бит 4-го регистра IEN0(0A8h)) происходит прерывание (вектор 023h), когда RI устанавливается контроллером в 1

Скорости обмена в разных режимах различаются. В двух из них скорости фиксированы, а в двух других – переменные. Рассмотрим подробнее способы задания скорости обмена в каждом из режимов.

- **Режим 0**. Фиксированная скорость обмена. В этом режиме скорость вычисляется следующим образом: $baudrate = \frac{Core\_Clk}{12}$, где *Core_Clk* – внутренняя тактовая частота микроконтроллера.

- **Режим 2**. Скорость обмена зависит от значения бита SMOD (старший бит в регистре PCON(087h)). Если SMOD = 0 (по умолчанию), то скорость определяется как 1/64 тактовой частоты. SMOD = l удваивает это значение. В общем случае: $baudrate = \frac{2^{SMOD} \times f_{osc}}{64}$.
- **Режимы 1 и 3**. В этих режимах порт можно синхронизировать от двух источников: от встроенного генератора импульсов и от таймера 1. Источник синхронизации определяется значением бита BD (старший бит регистра ADCON(0D8h)): если BD = 1, то используется встроенный генератор импульсов, в противном случае используется таймер 1.

Встроенный генератор импульсов делит тактовую частоту МП на 2500. Если SMOD = l, то полученное значение удваивается. Например, при тактовой частоте МП 12MHz можно получить стандартные скорости 12MHz/2500 = 4800 бод и (12MHz/2500)*2 = 9600 бод. В общем случае:

$$baudrate = \frac{2^{SMOD} \times f_{osc}}{2500}.$$

При использовании таймера 1 скорость обмена определяется временем переполнения таймера и значением бита SMOD:

$$baudrate = \frac{2^{SMOD} \times \mathrm{TM1OVrate}}{32},$$

где TMlOVrate – время переполнения таймера 1, зависящее от режима его работы. Таймер 1 может быть установлен как «таймер» или как «счетчик», в любом режиме работы. Прерывание от таймера при этом обычно запрещается. Чаще всего таймер 1 устанавливают как «таймер» в режиме 2: для этого старшая тетрада регистра TMOD(089h) должна равняться 0010b. При таком использовании после каждого переполнения регистра таймера TL1 в него загружается значение из регистра TH1. Скорость обмена в этом случае вычисляется по формуле:

$$baudrate = \frac{2^{SMOD}}{32} \times \frac{f_{osc}}{12 \times (256 - \mathrm{TH1})}.$$

В табл. 2 приведены способы задания некоторых стандартных скоростей обмена.

Таблица 2

Способы задания стандартных скоростей обмена

Скорость	f_{osc}, MHz	SMOD	Таймер 1		
			C/T	Режим	TH1
62.5 Кбод	12.0	1	0	2	FF
19200 бод	11.059	1	0	2	FD
9600 бод	11.059	0	0	2	FD
4800 бод	11.059	0	0	2	FA
2400 бод	11.059	0	0	2	F4
1200 бод	11.059	0	0	2	E8
11О бод	6.0	0	0	2	72
11О бод	12.0	0	0	1	FEEB

1.4. Работа с последовательным каналом по опросу

Простейшим из способов организации последовательного обмена является управление им из прикладной программы. То есть, если требуется переслать байт, то: 1) сбрасывается TI; 2) в SBUF записывается нужный байт данных; и 3) ожидается, пока TI не будет установлен контроллером. С приемом в этом случае дело обстоит сложнее: нужно постоянно проверять флаг RI и, если он установлен, то читать принятый байт из SBUF и сбрасывать RI. Такой способ удобен, когда разработчику четко известно, когда произойдет прием данных и когда его завершать. Неудобен он тем, что время выполнения самой программы напрямую зависит от скорости обмена. Чем медленнее скорость, тем дольше по времени цикл ожидания готовности к приему/посылке следующего байта.

При программировании последовательного канала под *Windows*, нужного состояния сигналов RTS и DTR можно добиться с помощью двух строк кода на языке С:

```
EscapeCommFunction( Port, SETRTS);
EscapeCommFunction( Port, CLRDTR).
```

2. Порядок работы

В процессе работы необходимо изучить:

1. Аппаратный блок последовательного канала:

- структурную схему контроллера последовательного канала;
- схему включения UART;
- страничную организацию памяти.

2. ПЛИС MAX:

- регистры ПЛИС.

3. Схему подключения светодиодов.

В процессе работы необходимо:

1. Разработать архитектуру драйверов и системы тестирования.
2. Написать тесты драйвера светодиодов.
3. Написать тест драйвера последовательного канала.
4. Написать драйвер светодиодов.
5. Написать драйвер UART (функции чтения и записи байтов, опрос готовности приемника) без использования прерываний.
6. Отладить и протестировать драйверы.

3. Содержание отчета

Отчет должен содержать:

1. Описание архитектуры:
   - используемой аппаратной части;
   - драйверов;
   - системы тестирования.
2. Исходные тексты драйверов.
3. Исходные тексты тестов.
4. Результаты тестирования.

4. Контрольные вопросы

1. Какие проблемы могут возникнуть при обращении к ПЛИС?
2. Какой режим работы последовательного канала используется в SDK-1.1 при связи с ПК?
3. Нарисуйте на бумаге формат посылки для байтов 0×14, 0×25, 0×A0.
4. Какой режим работы последовательного канала ADuC812 Вы использовали?
5. Что такое гальваническая изоляция?
6. Что такое оптрон?
7. Как работает схема гальванической развязки последовательного канала в SDK-1.1?
8. Какой бит устанавливается в логическую «1» при приеме байта по последовательному каналу?
9. Когда устанавливается бит TI, после записи байта в SBUF или после его передачи?
10. Нарисуйте регистровую модель контроллера последовательного канала ADuC812.

Лабораторная работа № 11
ТАЙМЕР, ИСПОЛЬЗОВАНИЕ ПРЕРЫВАНИЙ В ЛАБОРАТОРНОМ СТЕНДЕ SDK 1.1

Цели работы: изучить архитектуру таймеров и систему прерываний лабораторного стенда SDK-1.1, принципы обработки прерываний; разработать программы для управления системным таймером; просимулировать программы в отладчике-симуляторе; загрузить и выполнить программы на лабораторном стенде.

1. Методические указания к работе

1.1. Таймеры

ADUC812 имеет три 16-битных таймера/счетчика общего назначения: таймер 0, таймер 1, таймер 2. Каждый таймер состоит из двух 8-битных регистров THx и TLx (x обозначает 0, 1 или 2 для таймеров 0,1 и 2). Все три таймера могут быть настроены на работу в режимах «таймер» или «счетчик». В режиме «таймер» регистр инкрементируется каждый машинный цикл, то есть можно рассматривать это как подсчет машинных циклов. Так как машинный цикл состоит из 12 перепадов напряжения на тактовом входе контроллера, частота инкрементирования таймера в 12 раз меньше тактовой частоты контроллера.

В режиме «счетчик» регистр инкрементируется по перепаду 1→0 на соответствующем входе T0, T1 или T2. В этом режиме внешний вывод опрашивается в S5P2 каждого машинного цикла. Когда опрос показывает высокий уровень в одном цикле и низкий уровень в следующем, счетчик увеличивается на 1. Новое значение появляется в счетчике в S3P1 машинного цикла, следующего за тем, в котором был обнаружен перепад. Так как требуется два машинных цикла на обнаружение перепада 1→0, максимальная частота инкремента счетчика равна 1/24 тактовой частоты контроллера. Нет никаких ограничений на поведение сигнала на входе, но для обеспечения того, чтобы он был хотя бы раз опрошен, необходимо продержать его значение не менее одного машинного цикла. Для конфигурации и контроля таймеров используются 3 SFR-регистра: TMOD и TCON – для управления таймерами 1 и 0, T2CON – для управления таймером 2.

1.1.1. Таймеры 0 и 1

Каждый таймер состоит из двух 8-битных регистров. В зависимости от режима работы они могут использоваться как независимые регистры или как один объединенный 16-битный регистр.

TH0 и TH0 (SFR-адреса 8Ch и 8Ah) – старший и младший байты таймера 0.

TH1 и TH1 (SFR-адреса 8Dh и 8Bh) – старший и младший байты таймера 1.

Формат регистра режимов таймеров 0 и 1 TMOD (SFR-адрес 89h) приведен в табл. 1.

Таблица 1

Регистр режима таймеров TMOD

Номер бита	Обозначение	Описание
7	GATE	Управление стробированием таймера 1. Устанавливается программно и разрешает работу таймера-счетчика 1 только тогда, когда сигнал INT1 сохраняет высокий уровень и установлен бит TR1 в регистре TCON. Сбрасывается программно и разрешает работу таймера 1 только при установленном бите TR1
6	C/T	Выбор режима таймер/счетчик для таймера 1: 0 – работа в режиме «таймер»; 1 работа в режиме «счетчик»
5 4	MI MO	Устанавливают режим работы таймера 1 MI M0 0 0 – 13-битный таймер/счетчик 0 1 – 16-битный таймер/счетчик 1 0 – 8-битный режим с автоперезагрузкой 1 1 – таймер/счетчик 1 остановлен
3	GATE	Управление стробированием таймера 0. Устанавливается программно и разрешает работу таймера-счетчика 0 только тогда, когда сигнал INT0 сохраняет высокий уровень и установлен бит TR0 в регистре TCON. Сбрасывается программно и разрешает работу таймера 0 только при установленном бите TR0
2	C/T	Выбор режима таймер/счетчик для таймера 0: 0 – работа в режиме «таймер»; 1 – работа в режиме «сетчик»
1 0	MI M0	Устанавливают режим работы таймера 0 MI M0 0 0 – 13-битный таймер/счетчик 0 1 – 16-битный таймер/счетчик 1 0 – 8-битный режим с автоперезагрузкой 1 1 – таймер/счетчик 1 остановлен

Формат регистра управления таймерами 0 и 1 TCON (SFR-адрес 88h) приведен в табл. 2. В таблице описаны только 4 старших бита, младшие в управлении таймером не участвуют.

Таблица 2

Регистр управления таймерами TCON

Номер бита	Обозначение	Описание
7	TF1	Бит переполнения таймера 1. Устанавливается аппаратно при переполнении таймера/счетчика 1. Сбрасывается также аппаратно при переходе процессора на процедуру обработки прерывания
6	TR1	Бит запуска таймера 1. Устанавливается программно для запуска таймера 1. Программно сбрасывается для остановки таймера 1
7	TF0	Бит переполнения таймера 0. Устанавливается аппаратно при переполнении таймера/счетчика 0. Сбрасывается также аппаратно при переходе процессора на процедуру обработки прерывания
4	TR0	Бит запуска таймера 0. Устанавливается программно для запуска таймера 0. Программно сбрасывается для остановки таймера 0

1.1.2. Режимы работы таймеров/счетчиков 0 и 1

Режим 0 (13-битный таймер/счетчик) образует 8-битный таймер/счетчик с предварительным делителем на 32 на входе. Регистр таймера конфигурируется как 13-битный регистр. Когда регистр таймера переполняется (переходит из состояния «все 1» в состояние «все 0»), то выставляется флаг переполнения таймера TFx, который может использоваться для организации запроса на прерывание. Счет разрешается при TRx = 1 и либо GATE = 0, либо INTx = 1. Установка GATE = 1 приводит к тому, что работа таймера контролируется внешним входом INTx, что позволяет измерять длительность импульса.

13-битный регистр состоит из всех восьми бит THx и младших пяти регистра TLx. Старшие три бита TLx не определены и должны быть игнорированы. Установка бита запуска TRx не очищает регистр.

Работа таймера в этом режиме 1 (16-битный таймер/счетчик) аналогична работе в режиме 1, за исключением регистра таймера, который теперь конфигурируется как 16-разрядный регистр.

В режиме 2 таймер представляет собой 8-битный регистр с автоматической перезагрузкой. Когда TLx переполняется, не только устанавливается флаг TFx, но в TLx загружается значение THx, записанное туда ранее программно. Значение THx при перезагрузке сохраняется.

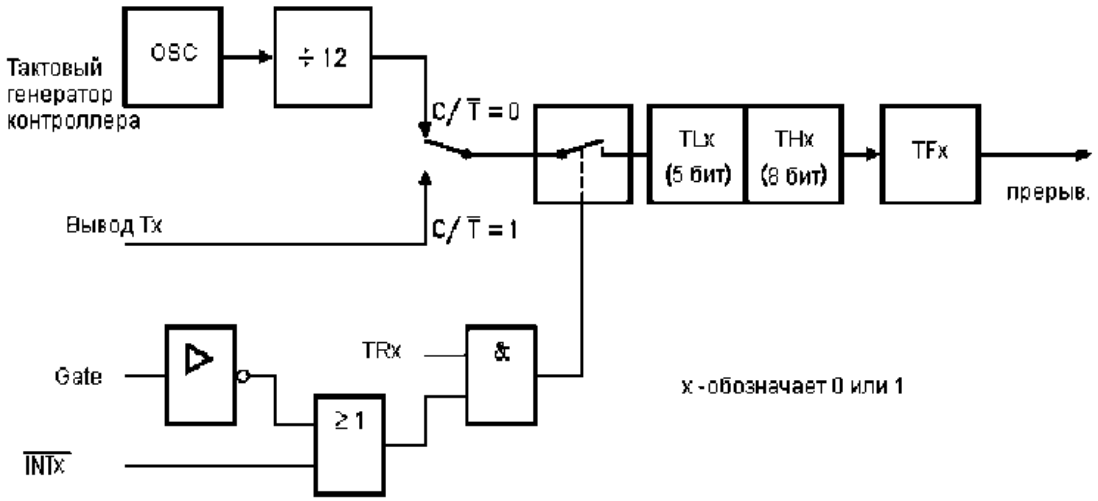

Рис. 1. Работа таймера в режиме 0

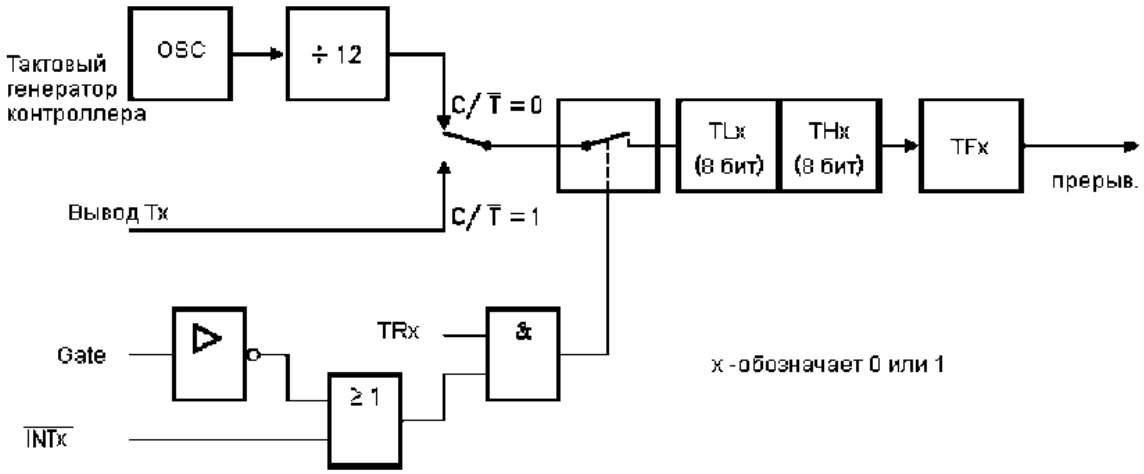

Рис. 2. Работа таймера в режиме 1

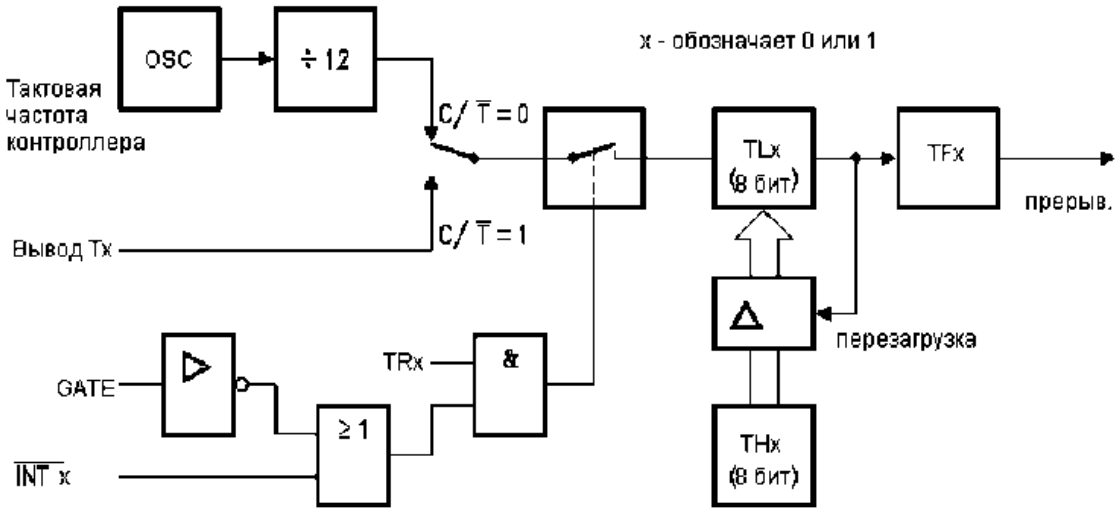

Рис. 3. Работа таймера в режиме 2

Работа в режиме 3 (два независимых 8-битных таймера/счетчика) имеет отличия для таймеров 0 и 1. Таймер 1 в этом режиме просто останавливает свой счет. Тот же эффект даст установка TR1 = 0. Таймер 0 в режиме 3 устанавливает TL0 и TH0 как два разных счетчика. Счетчик на базе TL0 использует биты управления таймера 0: C/T, GATE, TR0, INT0, TF0. TH0 зафиксирован в режиме таймера, считающего машинные циклы, и использует для управления биты TR1 и TF1 таймера 1. Таким образом, прерывание от переполнения регистра TH0 будет обозначено флагом TF1.

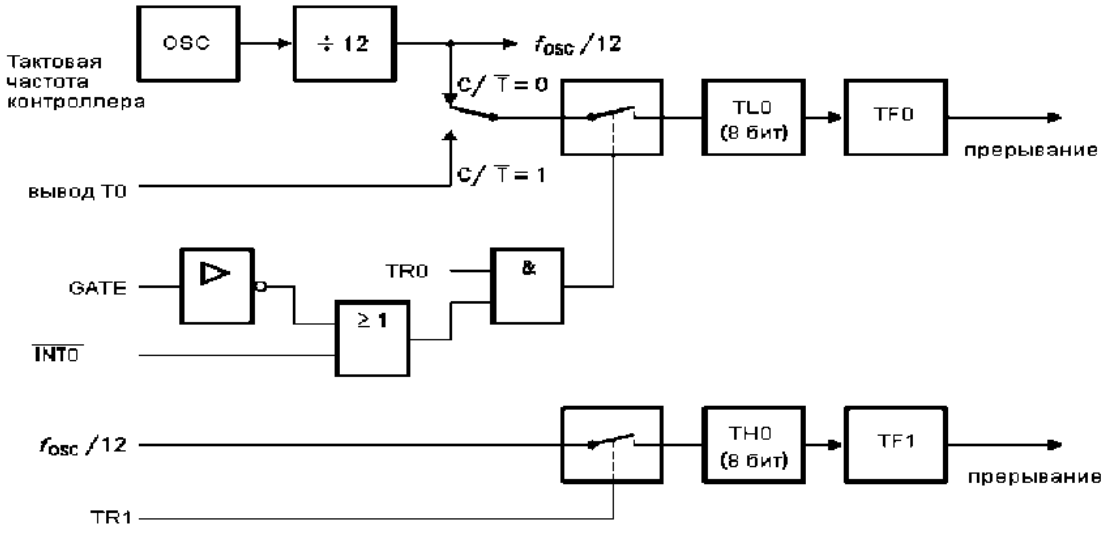

Рис. 4. Работа таймера в режиме 3

Режим 3 предназначен для приложений, которым нужен дополнительный 8-битный таймер или счетчик. Когда таймер 0 работает в режиме 3, таймер 1 может быть выключен установкой его в режим 3, или может быть оставлен включенным для использования в качестве генератора тактовых импульсов для последовательного интерфейса, или для любого приложения, которому не требуется прерывание именно от таймера 1.

1.1.3. Таймер 2

Регистры данных таймера 2 представлены регистрами:

TH2 и TL2 (SFR адреса CDh и CCh) – старший и младший бит данных. RCAP2H и RCAP2L (SFR адреса CBh и CAh).

1.1.4. Режимы работы таймера 2

Работа таймера 2 в 16-битном режиме с автоперезагрузкой зависит от значения бита EXEN2. Если EXEN2 = 0, таймер 2 при переполнении устанавливает флаг TF2 и перезагружает свои регистры TL2 и TH2 числами из регистров RCAP2L и RCAP2H, загруженными туда ранее про-

граммно. Если EXEN2 = 1, то таймер 2, кроме выше перечисленных функций, дополнительно производит автоперезагрузку и в случае появления отрицательного перехода 1→0 на внешнем входе T2EX.

Таблица 3

Регистр управления таймером 2 T2CON (SFR-адрес – C8h)

Номер бита	Обозначение	Описание
7	TF2	Флаг переполнения таймера 2 Устанавливается аппаратно в случае переполнения таймера 2. TF2 не устанавливается когда RCLK, либо TCLK равны 1. Сбрасывается программно
6	EXF2	Внешний флаг таймера 2. Устанавливается аппаратно когда происходит захват или перегрузка по отрицательному переходу на входе T2EX, при этом EXEN2 = 1
5	RCLK	Выбор таймера для тактирования приема по последовательному каналу: 1 – разрешает тактирование приема данных по последовательному каналу в режимах 1 и 3 таймером 2; 0 – разрешает тактирование приема данных по последовательному каналу в режимах 1 и 3 таймером 1
4	TCLK	Выбор таймера для тактирования передачи по последовательному каналу: 1 – разрешает тактирование передачи данных по последовательному каналу в режимах 1 и 3 таймером 2; 0 – разрешает тактирование передачи данных по последовательному каналу в режимах 1 и 3 таймером 1
3	EXEN2	Флаг разрешения внешнего управления. Устанавливается пользователем для разрешения захвата или перегрузки по отрицательному переходу на входе T2EX, если таймер 2 не используется для тактирования последовательного порта. Сбрасывается пользователем для игнорирования сигналов на T2EX
2	TR2	Бит запуска таймера 2. Устанавливается программно для запуска таймера 2. Программно сбрасывается для остановки таймера 2
1	CNT2	Выбор режима таймер/счетчик: 0 – работа в режиме «таймер»; 1 – работа в режиме «сетчик»
0	CAP2	Выбор режима захват/перезагрузка: 0 – режим перезагрузки по переполнению или по отрицательному перепаду на входе T2EX, если EXEN2 = 1. При тактировании последовательного порта бит игнорируется; 1 – режим захвата по отрицательному переходу на входе T2EX, если EXEN2 = 1

Таблица 4

Режимы работы таймера 2

RCLK или TCLK	CAP2	TR2	Режим
0	0	1	16 бит с автоперезагрузкой
0	1	1	16 бит с захватом
1	X	1	Тактирование последовательного канала
X	X	0	Выключен

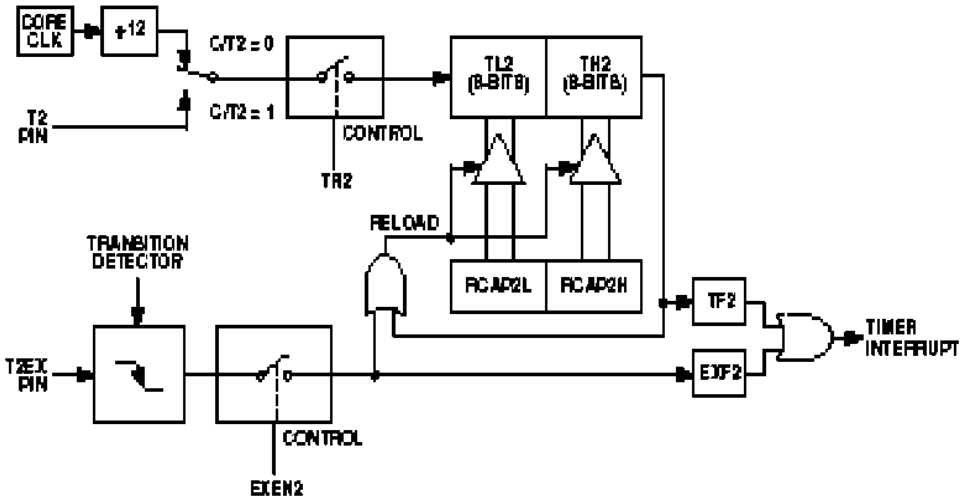

Рис. 5. Работа таймера 2 в режиме с автоперезагрузкой

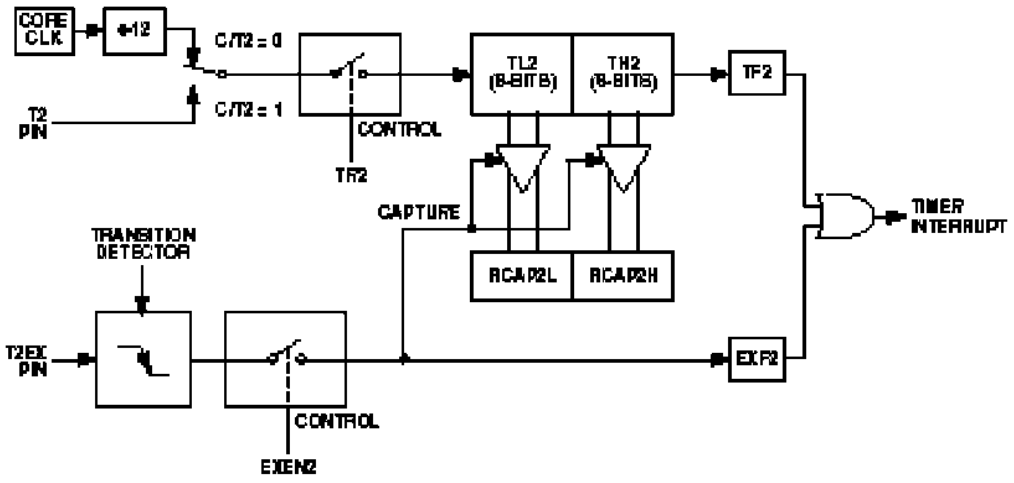

Рис. 6. Работа таймера 2 в режиме захвата

Работа таймера 2 в 16-битном режиме захвата также зависит от бита EXEN2 регистра TCON. Если EXEN2 = 0, то таймер 2 работает как 16-битный таймер/счетчик, при переполнении которого устанавливается флаг TF2, который можно использовать для организации запроса на прерывание. Если EXEN2 = 1, то таймер 2, кроме перечисленных функций, дополнительно в случае появления отрицательного перепада на

внешнем входе T2EX, произведет захват содержимого регистров TL2 и TH2 в регистры RACP2L и RACP2H, соответственно. Кроме этого установится флаг EXF2, который может сгенерировать прерывание.

В этом режиме тактирования последовательного порта таймер 2 используется для тактирования передачи последовательного порта, и флаг прерывания TF2 не устанавливается. Однако флаг EXF2 устанавливается, и может вызывать прерывания. В этом режиме его можно использовать в качестве еще одного источника внешних прерываний.

1.2. Система прерываний

Процессор AduC812 поддерживает 9 источников прерываний с 2 уровнями приоритета. Конфигурирование и контроль системы прерываний осуществляется через 3 регистра специальных функций.

Регистр IE (SFR адрес A8h) – регистр разрешения прерываний. Все биты регистра устанавливаются и сбрасываются пользователем.

Таблица 5

Регистр разрешения прерываний IE

Бит	Обозн.	Описание
7	EA	0 – запрещены; 1 – разрешены прерывания от всех источников
6	EADC	0 – запрещены; 1 – разрешены прерывания от АЦП
5	ET2	0 –запрещены; 1 – разрешены прерывания от таймера 2
4	ES	0 – запрещены; 1 – разрешены прерывания от UART
3	ET1	0 – запрещены; 1 – разрешены прерывания от таймера 1
2	EX1	0 – запрещено; 1 – разрешено внешнее прерывание 1
1	ET0	0 – запрещены; 1 – разрешены прерывания от таймера 0
0	EX0	0 – запрещено; 1 – разрешено внешнее прерывание 0

Регистр IP (SFR адрес B8h) – регистр приоритета прерываний. Все биты регистра устанавливаются и сбрасываются пользователем. После включения питания по умолчанию содержит 00h. Назначения битов регистра приведены в табл. 6.

Таблица 6

Регистр приоритета прерываний IP

Бит	Обозн.	Описание
7	PSI	0 – высокий; 1 – низкий приоритет прерывания от SPI/I²C
6	PADC	0 – высокий; 1 – низкий приоритет прерывания от АЦП
5	PT2	0 – высокий; 1 – низкий приоритет прерывания от таймера 2
4	PS	0 – высокий; 1 – низкий приоритет прерывания UART
3	PT1	0 – высокий; 1 – низкий приоритет прерывания от таймера 1
2	PX1	0 – высокий; 1 – низкий приоритет внешнего прерывания 1
1	PT0	0 – высокий; 1 – низкий приоритет прерывания от таймера 0
0	PX0	0 – высокий; 1 – низкий приоритет внешнего прерывания 0

Регистр IE2 (SFR адрес A9h) – регистр разрешения вторичных прерываний (табл. 7).

Таблица 7

Регистр разрешения вторичных прерываний IE2

Бит	Обозн.	Описание
7–2	–	Зарезервированы для использования в будущем
1	ET0	0 – запрещены; 1 – разрешены прерывания от монитора источника питания
0	EX0	0 – запрещено; 1 – разрешены прерывания от SPI/I²C

Регистры разрешения прерывания устанавливаются пользователем для разрешения прерываний от отдельных источников, в то время как регистры установки приоритета устанавливаются пользователем для выбора одного из двух уровней приоритета для каждого прерывания. Прерывание с высоким уровнем приоритета может прерывать обслуживание прерывания с низким уровнем приоритета, а если прерывания с разными уровнями придут одновременно, то прерывание с высоким приоритетом будет обслужено первым. Обслуживание прерывания не может быть прервано прерыванием с таким же уровнем приоритета. Если два прерывания с одинаковым уровнем приоритета пришли одновременно, то порядок их обслуживания определяется с помощью табл. 8.

Таблица 8

Приоритет источников прерываний

Источник	Приоритет	Описание
PSMI	1 (наивысший)	Монитор источника питания
IE0	2	Внешнее прерывание 0
ADCI	3	Прерывание от АЦП
TFO	4	Прерывание от таймера/счетчика 0
IE1	5	Внешнее прерывание 1
TF1	6	Прерывание от таймера/счетчика 1
I2CI+ISPI	7	Прерывание от I C/SPI
RI+TI	8	Прерывание от UART
TF2+EXF2	9 (низший)	Прерывание от таймера/счетчика 2

Когда происходит прерывание, значение программного счетчика помещается в стек, а в сам счетчик загружается адрес соответствующего вектора прерывания. Адреса векторов указаны в табл. 9.

Прерывания ADuC812 имеют вектора в диапазоне 0003h-0043h, которые попадают в область младших адресов памяти программ. Это пространство соответствует 8 Кб (0000h-2000h) Flash-памяти. Следовательно, пользователь, не имеющий возможности записи во Flash-память, не может подставить свои процедуры обработки прерываний (точнее, команды перехода к процедурам) по адресам, соответствующим векторам прерываний.

Таблица 9

Вектора прерываний ADuC812

Прерывание	Наименование	Адрес вектора	Приоритет
PSMI	Источник питания ADuC812	43H	1
IE0	Внешнее прерывание INT0	03H	2
ADCI	Конец преобразования АЦП	33H	3
TF0	Переполнение таймера 0	0BH	4
IE1	Внешнее прерывание INT1	13H	5
TF1	Переполнение таймера 1	1BH	6
I2CI/ISPI	Прерывание последовательного интерфейса (I^2C, SPI)	3BH	7
RI/TI	Прерывание UART	23H	8
TF2/EXF2	Переполнение таймера 2	2BH	9

Проблема использования прерываний в пользовательских программах решается следующим образом:

1. По адресам (0003h-0043h) векторов прерываний во Flash-памяти SDK-1.1 располагаются команды переходов на вектора пользовательской таблицы, размещенной в адресах 2003h-2043h.

2. По адресам векторов пользовательской таблицы пользователем указываются команды переходов на процедуры обработки прерываний.

Переадресацию векторов прерываний иллюстрирует рис. 7.

Следующая программа – пример помещения собственного вектора в пользовательскую таблицу. Пусть требуется осуществить обработку прерываний от таймера 0 (прерывание 0Bh). В программу на языке Си можно вставить следующий код:

```
void T0_ISR(void) interrupt 1 // Обработчик прерывания от
                              // таймера 0
{
   // Действия, выполняемые обработчиком
}
void SetVector(unsigned char xdata * Address, void * Vector)
// Функция, устанавливающая вектор прерывания Vector по
//адресу Address
{
   unsigned short xdata * TmpVector;// Временная переменная
   *Address = 0x02; //Первым байтом по указанному адресу за-
//писывается код команды передачи управления ljmp,равный 02h
   TmpVector = (unsigned short xdata *) (Address+1);
   *TmpVector = (unsigned short) Vector;
// Далее записывается адрес перехода Vector
// Таким образом, по адресу Address теперь располагается
// инструкция ljmp Vector
}
```

```
void main(void)
{
   /*…*/
SetVector(0x200B, (void *) TO_ISR); // Установка вектора в
// пользовательской таблице
   ET0 = 1; EA = 1; // Разрешение прерываний от таймера 0
}
```

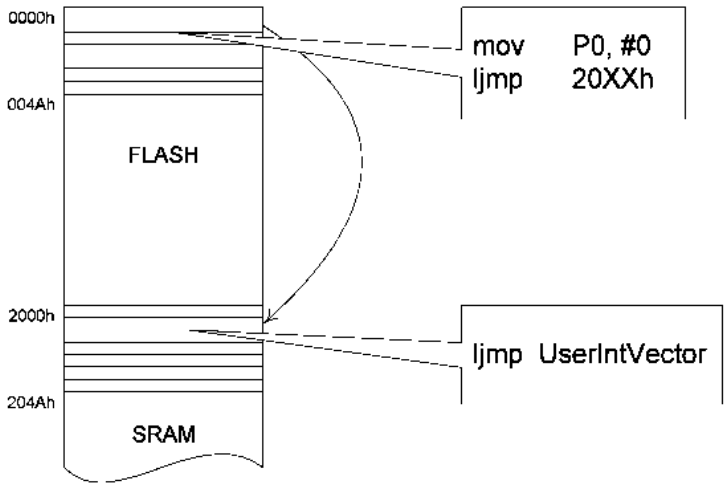

Рис. 7. Использование прерываний в SDK-1.1

Компилятор C51 предоставляет возможность вызова функций при возникновении прерываний. Это дает возможность написания на языке С собственных обработчиков прерываний. Однако следует соблюдать осторожность в выборе номера прерывания и банка регистров. Компилятор автоматически генерирует вектор прерывания, а также код входа в обработчик и выхода из него. Атрибут *interrup*t, включенный в объявление функции, указывает на то, что данная функция обрабатывает прерывание. Кроме того, можно указать банк регистров для данного прерывания с помощью атрибута *using*.

```
unsigned int interruptcnt;
unsigned char second;
void timerO (void) interrupt 1 using 2
{
     if (++interruptcnt = =  4000)
       {
          /* count to 4000        */
           second++;  /* second counter     */
          interruptcnt  = 0;  /* clear int counter */
       }
}
```

2. Порядок работы

В процессе работы необходимо изучить:

1. Архитектуру аппаратных блоков:
   - таймеры микроконтроллера;
   - систему прерываний микроконтроллера.
2. Принципы обработки прерываний.

В процессе работы необходимо:

1. Разработать архитектуру драйвера системного таймера и системы тестирования.
2. Написать тесты драйвера.
3. Написать драйвер системного таймера.
4. Отладить и протестировать драйвер.

3. Содержание отчета

Отчет должен содержать:

1. Описание архитектуры:
   - используемой аппаратной части;
   - драйверов;
   - системы тестирования.
2. Исходные тексты драйверов.
3. Исходные тексты тестов.
4. Результаты тестирования.

Лабораторная работа № 12
РАБОТА С КЛАВИАТУРОЙ ЛАБОРАТОРНОГО СТЕНДА SDK 1.1

Цель работы: Изучить архитектуру клавиатуры в составе лабораторного стенда SDK-1.1. Разработать и написать драйвер клавиатуры для учебно-лабораторного стенда SDK-1.1. Написать тестовую программу для разработанного драйвера, которая выполняет определенную задачу.

1. Методические указания к работе

Клавиатура SDK-1.1 организована в виде матрицы 4×4. Один вывод каждой кнопки соединяется с горизонтальной линией, второй вывод соединяется с вертикальной линией. Каждая горизонтальная линия одним концом соединяется с источником питания, а вторым подключается к входному порту. Вертикальная линия подключается только с одной стороны к отдельным разрядам выходного порта. Схематично клавиатура лабораторного стенда показана на рис. 1.

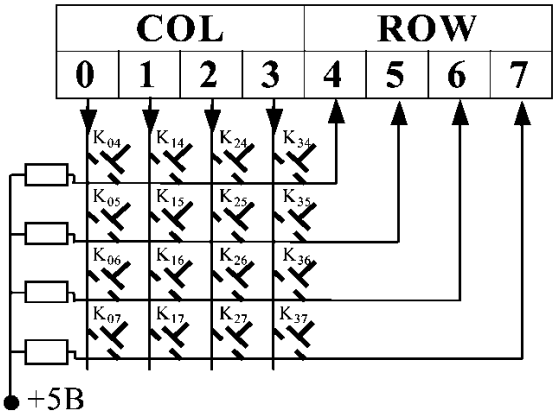

Рис. 1. Клавиатура

Если все ключи разомкнуты, то потенциал на выходных портах ROW остается высоким, т. е. соответствует логической единице. Однако если замкнуть один из ключей и на соответствующую вертикальную линию подать сигнал низкого уровня, то потенциал горизонтальной линии, с которой соединена эта кнопка, также станет низким. Иначе говоря, если записать в порт COL значение 1101, то это позволит сканировать состояние кнопок на этого столбца. Например, если при этом будет нажата кнопка $К_{26}$, то на горизонтальной линии будет низкий потенциал и соответствующий разряд выходного порта примет значение логического нуля, т. е. ROW = 1101.

Для доступа к портам клавиатуры используется регистр KB микросхемы ПЛИС (адрес 080000h).

Драйвер клавиатуры должен работать по прерыванию от таймера. Во время прерывания производится опрос регистра клавиатуры, и если нажата какая-либо клавиша, ее код заносится в буфер драйвера. Чтение символа из буфера драйвера производится с помощью API-функции чтения *ReadKeyBuffer*(), которая в случае удачного завершения возвращает 1 и передает считанный из буфера байт, а в случае, если буфер пуст, возвращает 0. Буфер клавиатуры организован аналогично буферам драйвера последовательного интерфейса. При инициализации необходимо указать задержку перед повтором символа (первый параметр) и скорость повтора символа (второй параметр).

Таблица 1

Регистр клавиатуры

COL				ROW			
0	1	2	3	4	5	6	7
W	W	W	W	R	R	R	R

Таким образом, полный опрос матричный клавиатуры включает последовательное обнуление младших четырех битов регистра KB с анализом на каждом шаге старших разрядов регистра.

При нажатии на кнопку напряжение не сразу устанавливается на уровне 0 В, а изменяется в течение некоторого времени (1–10 мс), пока цепь надежно не замкнется. После того, как клавиша будет отпущена, напряжение также изменяется, пока не установится на уровне логической «1». Это явление носит название «дребезг» контактов (рис. 2).

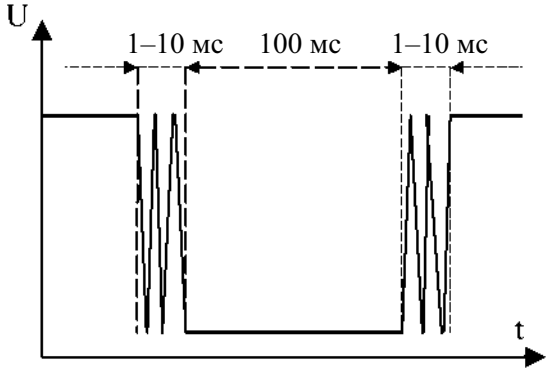

Рис. 2. «Дребезг» контактов клавиатуры

Поскольку процессор обладает высоким быстродействием, то он может воспринять эти скачки напряжения за несколько нажатий. Для программного устранения влияния «дребезга» используется задержка. После того как в результате сканирования обнаружится 0 в регистре

ROW, сканирование прекращается и производится задержка на некоторое время. После этого сканируется тот же столбец и, если на том же месте регистра ROW обнаружен 0, то фиксируется нажатие клавиши. После этого через некоторое время, достаточное для отпускания клавиши, еще раз проверяется тот же столбец. Если состояние линии изменилось, то фиксируется отпускание клавиши и продолжается сканирование клавиатуры. Если клавиша все еще нажата, то производится задержка на время перед повтором символа, и если состояние регистра не изменилось, то в буфер клавиатуры повторно заносится символ. После этого, пока клавиша не будет отпущена, в буфер заносится код клавиши через промежутки времени, определяемые скоростью повтора символа.

В связи с тем, что сканирование клавиатуры и обработка нажатия клавиши производится в теле обработчика прерывания, необходимо следить, чтобы время его выполнения не превышало времени между соседними прерываниями. Это приведет к повторному входу в обработчик прерывания, последствия этого могут быть непредсказуемы. Также увеличение времени выполнения обработчика приведет к уменьшению времени выполнения основной программы, т. е. может оказаться, что процессор большую часть времени будет занят обработкой прерываний.

Исходя из этого, все задержки в обработчике были реализованы не на прямую, а через счетчик прерываний. После фиксации нажатия устанавливается флаг нажатия *KeyPress*, и сканирование запрещается до тех пор, пока клавиша не будет отпущена. Теперь при каждом вызове обработчика увеличивается значение счетчика и проверяется его состояние. Если счетчик равен определенным значениям, то выполняются связанные с ними действия (например: count = l, прошло время достаточное для пропускания «дребезга»; count = 3, можно фиксировать отпускание клавиши; count = 3+*DelayBeforeRepeat*, пропущена задержка перед повтором символа). Перед сканированием клавиатуры производится проверка на нажатие, что позволяет не тратить время на бесполезное сканирование, если не одна клавиша не нажата. Если фиксируется нажатие, то производится проверка буфера на заполнение, и если буфер переполнен, то работу обработчика необходимо завершить. Дальше производится сканирование клавиатуры, пока не будет идентифицирована нажатая клавиша. После этого устанавливается флаг, и работа обработчика завершается. Когда приходит следующее прерывание, то поскольку установлен флаг нажатия, клавиатура не сканируется, а сразу осуществляется переход к обработке нажатия на клавишу. Поскольку между текущим прерыванием и прерыванием, во время которого было зафиксировано нажатие клавиши, прошло некоторое время, то «дребезг» был пропущен. Теперь можно повторно проверить состояние кла-

виши, отмеченной переменными COL и ROW. Если клавиша еще нажата, то соответствующий символ в буфер, в противном случае флаг нажатия клавиши *KeyPress* сбрасывается, обработчик завершается и продолжается сканирование клавиатуры.

После записи кода клавиши в буфер осуществляется задержка, достаточная для отпускания клавиши, и еще раз опрашивается состояние клавиши. Если состояние клавиши изменилось, то фиксируется отпускание клавиши (флаг нажатия клавиши *KeyPress* сбрасывается) и продолжается сканирование клавиатуры. Иначе осуществляется задержка перед повтором символа, и если клавиша еще не отпущена, повторно заносится код символа. После этого через промежутки, определяемые скоростью повтора, код клавиши заносится в буфер до тех пор, пока не будет зафиксировано отпускание клавиши.

2. Требования к выполнению работы

1. В тестовой программе должна быть продемонстрирована работа с клавиатурой и последовательным интерфейсом по прерыванию.

2. Переключение между двумя задачами в тестовой программе должно быть выполнено с использованием DIP-переключателей.

3. Должен быть предусмотрен контроль ввода корректных значений в рамках второй задачи программы.

3. Содержание отчета

1. Иллюстрация организации и функционирования разработанного программного обеспечения (драйверы, тестовая программа) в виде блок-схемы, диаграммы процессов, потоков данных, диаграммы состояний автоматов и других схем поясняющего характера.

2. Исходный текст программы с комментариями.

3. Основные результаты.

Лабораторная работа № 13
РЕАЛИЗАЦИЯ ИНТЕРФЕЙСА I²C В ЛАБОРАТОРНОМ СТЕНДЕ SDK 1.1

Цели работы: изучить архитектуру последовательного интерфейса I^2C лабораторного стенда SDK-1.1; разработать программы управления последовательным интерфейсом; просимулировать программы в отладчике-симуляторе; загрузить и выполнить программы на лабораторном стенде.

1. Методические указания к работе

1.1. Интерфейс I²C, общие сведения

Интерфейс I^2C (*Inter Integrated Circuit*) является двухпроводной беспроводной системой связи, позволяющей установить связь между несколькими ведущими и несколькими ведомыми устройствами. Связь осуществляется с помощью двух двунаправленных линий: SCLOCK управляет передачей данных, SDATA используется для обмена данными. Скорость передачи определяется частотой синхронизирующих импульсов на линии SCLOCK, которые генерирует ведущее устройство. Ведомое устройство этими импульсами управляется. Типичная последовательность передачи данных по интерфейсу I^2C показана на рис. 1.

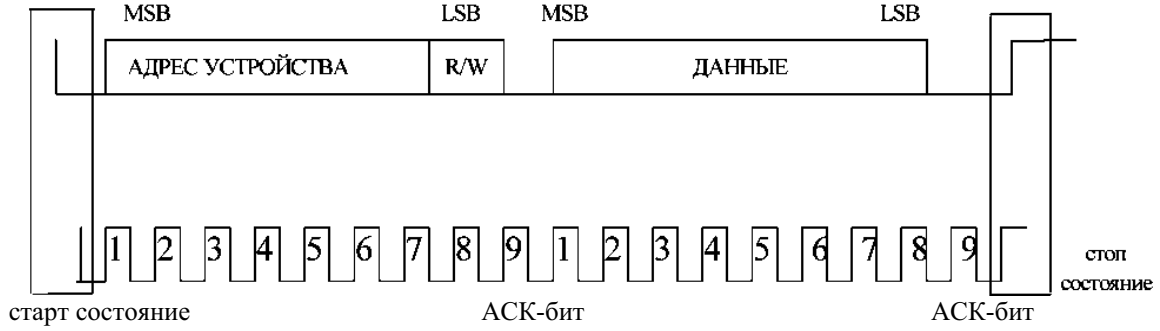

Рис. 1. Последовательность передачи данных по интерфейсу I^2C

Последовательность начинается стартовым состоянием. Стартовое состояние представляет собой изменение уровня сигнала на линии SDATA от низкого на высокий, в то время как уровень сигнала на SCLOK высокий.

После стартового бита ведущий посылает байт (начиная со старшего бита), который содержит адрес ведомого устройства и бит статуса R/W (чтение/запись). Первые семь бит содержат адрес устройства, а младший определяет направление сообщения: 0 – передача; 1 – прием.

Теперь каждое ведомое устройство в системе сравнивает принятый семибитный адрес со своим адресом, и, если он совпадает, выдает бит подтверждения (ACK). Подтверждение выглядит как низкий уровень на линии SDATA на девятом такте и должно выдаваться ведомым устройством после приема каждого байта. Если нет подтверждения или передача закончилась, ведущее устройство генерирует состояние завершения.

Состояние завершения представляет собой изменение уровня сигнала на линии SDATA от высокого к низкому, в то время как уровень сигнала на SCLOK высокий.

1.2. Реализация интерфейса I²C на Aduc812

Aduc812 поддерживает двухпроводный последовательный интерфейс, совместимый с I^2C. Этот интерфейс разделяет контакты внутреннего SPI-интерфейса, и поэтому в каждый момент времени пользователю доступен либо один, либо другой интерфейс (зависит от бита SPE-регистра SPICON). Интерфейс конфигурируется в два режима: «программный ведущий» и «аппаратный ведомый», и использует два контакта: SDATA (контакт 27) – ввод/вывод последовательных данных и SCLOCK (контакт 26) – последовательный синхросигнал. Для управления интерфейсом используется три SFR-регистра:

I2CCON (SFR адрес E8h) – регистр управления I^2C (табл. 1);

I2CADD (SFR адрес 55h) – регистр адреса интерфейса I^2C, содержит адрес периферийного устройства, возможна запись кодом пользователя;

I2CDAT (SFR адрес 9Ah) – регистр данных интерфейса I^2C, в регистр записываются данные для передачи по интерфейсу I2C или считываются только что пришедшие по интерфейсу данные.

Таблица 1

Регистр управления I^2C I2CCON

Бит	Название	Описание
7	MDO	Бит выходных данных (только в режиме «программный ведущий»). Используется для передачи данных в режиме «программный ведущий». Данные, записанные в этот бит, будут выдвинуты на контакт SDATA, если установлен бит разрешения передачи данных (MDE)
6	MDE	Бит разрешения передачи данных (только в режиме «программный ведущий»). Устанавливается пользователем для назначения контакта SDATA: 1 – передача данных; 0 – прием данных

Окончание табл. 1

Бит	Название	Описание
5	MCO	Бит выходного синхроимпульса (только в режиме программный ведущий). Используется в режиме передающего I²C интерфейса. Данные, записанные в этот бит, будут переданы на вывод SCLOCK
4	MDI	Бит входных данных (только в режиме «программный ведущий»). Используется для приема данных в режиме «программный ведущий». В этот бит считывается значение SDATA по сигналу SCLOCK, если сброшен бит разрешения передачи данных (MDE)
3	I2CM	Бит выбора режима «ведущий/ведомый». Устанавливается и сбрасывается пользователем. 0 – аппаратный ведомый; 1 – программный ведущий
2	I2CRS	Бит сброса I2C (только в режиме аппаратный ведомый). Устанавливается и сбрасывается пользователем. 0 – нормальная работа I²C; 1 – сброс интерфейс I²C
1	I2CTX	Бит направления передачи (только в режиме аппаратный ведомый). Устанавливается аппаратно. 0 – прием; 1 – передача
0	I2CI	Флаг прерывания (только в режиме аппаратный ведомый). Устанавливается аппаратно после приема или передачи байта. Сбрасывается программно

1.3. Устройства, подключенные к шине I²C в стенде SDK1.1

1.3.1. Электрически стираемое и перезаписываемое ПЗУ E²PROM

На стенде SDK1.1 установлена E²PROM AT24C01A, состоящая из 128 однобайтных страниц общим объемом 1 К. Адрес E²PROM на шине I²C равен 1010001, плюс младший байт, отвечающий за направление обмена.

Внутренний счетчик адреса содержит последний адрес, к которому производилось обращение для чтения или записи, увеличенный на единицу. После выключения питания счетчик не сохраняет свое значение. Во время операции чтения счетчик адреса автоматически переключается с последнего байта последней страницы на первый байт первой страницы. При записи он переходит с последнего байта страницы на первый байт той же самой страницы.

1.3.2. Часы реального времени

Микросхема PCF8583 содержит 8-битную оперативную память объемом 256 байт с 8-битным адресным регистром, осуществляющим автоматическое инкрементирование адреса, генератор частоты, шину I^2C и схему сброса при выключении питания. Первые 16 байт ОЗУ представляют собой 8-битные регистры специального назначения. Первый регистр по адресу 00 – регистр управления/состояния. Следующие 7 регистров – счетчики, последние восемь регистров могут быть запрограммированы как регистры сигнала или быть вообще отключены и использованы как обычные регистры памяти (когда сигналы отключены). Оставшиеся 240 байт используются как оперативная память. При подключении питания к шине I^2C сбрасывается регистр управления/состояния и все счетчики часов. Устройство начинает отсчет в режиме часов на частоте 32,768 кГц в 24-часовом формате времени и с датой и временем, установленными на 1 января в 0.00.00:00. Адрес устройства на шине I^2C – 1010000, плюс младший бит направления обмена.

При программировании регистра состояний может быть выбран один из следующих режимов работы:

- режим часов на частоте 32.768 кГц;
- режим часов на частоте 50 Гц;
- режим счетчика событий.

В случае если выбран режим часов, то сотые доли секунд, секунды, минуты, часы, дата, месяц (календарь на 4 года) и дни недели хранятся в двоично-десятичном формате.

Режим счетчика используется для подсчета импульсов, подаваемых на вход генератора.

Счетчики в этом режиме работают также в двоично-десятичном формате и могут хранить до 6 цифр данных. Таким образом, в счетчике может быть сохранено до миллиона событий.

При установке бита, разрешающего сигнал, активизируется регистр управления сигналом.

Таблица 2

Регистр управления состоянием (адрес 00)

7	Флаг остановки счета: 0 – импульсы счета; 1 – остановка счета, сброс делителя
6	Флаг сохранения последнего считывания: 0 – счет; 1 – сохранить и скопировать в регистры задвижки
5–4	Режим работы: 00 – часы на частоте 32,768 кГц; 01 – часы на частоте 50 Гц; 10 – счетчик событий; 11 – тестовый режим

Окончание табл. 2

3	Флаг маски: 0 – немаскируемое чтения адресов 05-06; 1 – непосредственное чтение даты и месяца
2	Бит разрешения сигнала. 0 – сигналы отключены; 1 – сигналы включены
1	Флаг сигнала (50%-й занятости флага секунд, если бит разрешения 0)
0	Флаг таймера (50%-й занятости флага минут, если бит разрешения 0)

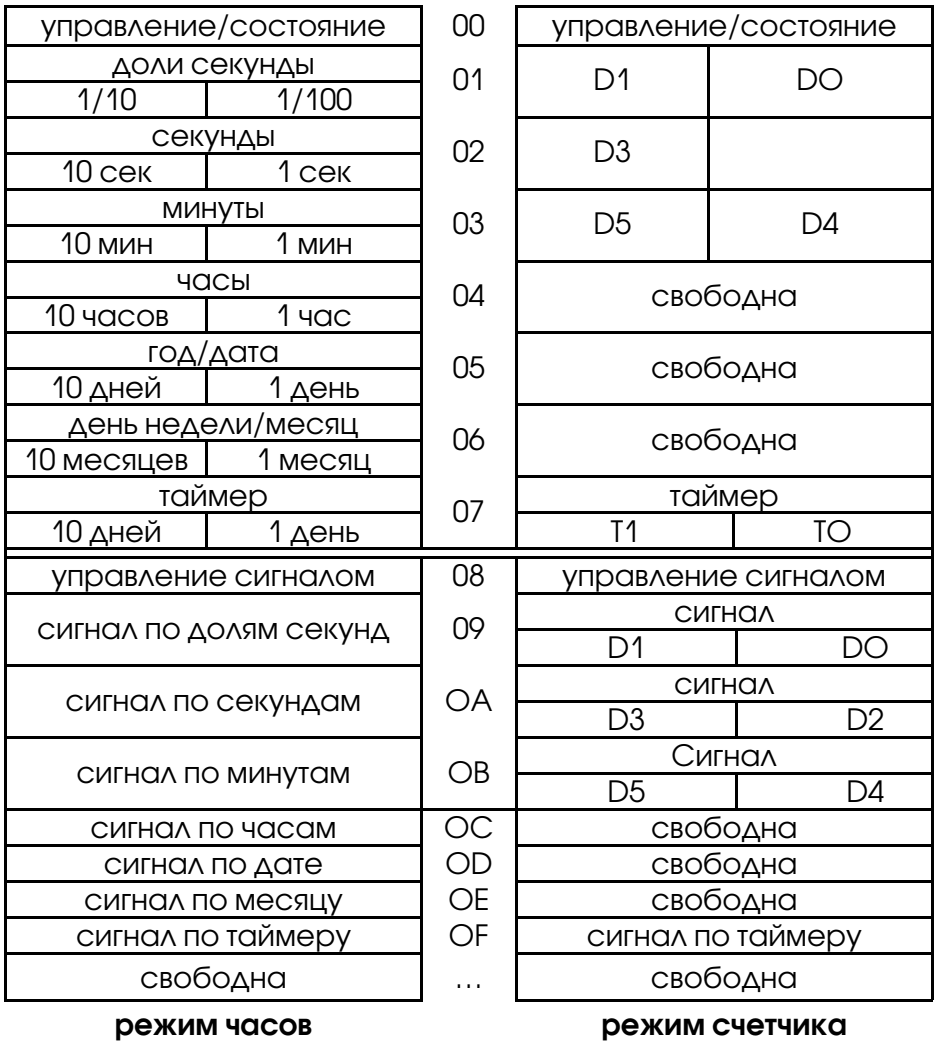

Рис. 2. Регистры-счетчики

Если бит разрешения сигнала выставлен, активизируется регистр управления сигналом, который управляет сигналом, таймером и выходами прерываний.

Таблица 3

Счетчик часов (адрес в памяти 04)

7	Формат времени: 0 – 24-часовой формат. Флаг AM/PM не изменяется; 1 – 12-часовой формат
6	Флаг AM/PM: 0 – AM; 1 – PM
5–4	10 часов (то 0 до 2) в двоичном формате
3–0	Часы в двоично-десятичном формате

Таблица 4

Счетчик года/даты (адрес в памяти 05)

7–6	Год (от 0 до 3) в двоичном формате
5–4	10 дней (то 0 до 3) в двоичном формате
3–0	Дни в двоично-десятичном формате

Таблица 5

Счетчик дни недели/месяц (адрес в памяти 06)

7–5	Дни недели (от 0 до 6 в двоичном формате)
4	10 месяцев
3–0	Месяцы в двоично-десятичном формате

Таблица 6

Регистр управления сигналом (адрес в памяти 08)

7	Разрешения прерывания по сигналу: 0 – установка флага не вызывает прерывание; 1 – установка флага вызывает прерывание
6	Разрешение сигнала по таймеру: 0 – нет сигнала по таймеру; 1 – есть сигнал по таймеру
5–4	Сигнал по часам: 00 – нет сигнала; 01 – ежедневный сигнал; 10 – сигнал по дню недели; 11 – сигнал по дате
3	Разрешение прерывания по таймеру: 0 – нет прерывания по таймеру; 1 – есть прерывание по таймеру
2–0	Функции таймера: 000 – нет таймера; 001 – сотые доли секунды; 010 – секунды; 011 – минуты; 100 – часы; 101 – дни; 110 – не используется; 111 – тестовый режим

Регистры управления сигналов (09-0F). Генерация сигналов происходит тогда, когда каждый бит регистра сигнала совпадает с аналогичным битом соответствующего регистра счетчика. Однако если речь идет о сигнале по дате, игнорируются дни недели. При генерации ежедневного сигнала игнорируются биты месяца и даты. Если выбран сигнал по дням недели, то из регистра сигнала дней недели/месяца будут выбраны соответствующие дни недели.

Таблица 7

Регистр сигнала по дням недели (адрес в памяти 0E)

7	Не используется
6	День недели 6, разрешен, если установлен
5	День недели 5, разрешен, если установлен
4	День недели 4, разрешен, если установлен
3	День недели 3, разрешен, если установлен
2	День недели 2, разрешен, если установлен
1	День недели 1, разрешен, если установлен
0	День недели 0, разрешен, если установлен

Таймер

Таймер ведет счет или от 0, или запрограммированного заранее значения до 99. При переполнении устанавливается флаг таймера, а таймер принимает значение 0. Флаг сбрасывается программно, инвертированное значение может быть передано внешнему прерыванию установкой флага разрешения прерывания по таймеру.

Кроме этого, сигнал по таймеру может быть запрограммирован установкой флага разрешения сигнала по таймеру. Тогда значение таймера станет равным числу в регистре сигнала по таймеру.

Счетчик

Сигнал счетчика выдается, когда содержимое регистра счетчика совпадает со значением, хранящимся в регистрах по адресам 09,0A,0B, при этом устанавливается флаг сигнала. Инвертированное значение флага может быть передано внешнему прерыванию установкой флага разрешения прерывания по сигналу.

Таймер в этом режиме инкрементируется по наступлению каждого первого, сотого, десятитысячного, миллионного события, в зависимости от значений, установленных в битах 0,1 и 2 регистра управления сигнала.

1.4. Работа с устройствами, подключенными к шине I²C

1.4.1. Запись

Запись байта

После того как устройство подтвердит возможность приема, ему передается байт адреса, который загружается во внутренний счетчик адреса устройства. После подтверждения приема адреса отправляется байт данных. После приема байта устройство выдает подтверждение приема, и адресующие устройство должно остановить процесс записи выдачей стоп-сигнала.

В этот момент E²PROM начинает внутренний цикл записи в постоянную память. До тех пор пока запись не будет завершена отключаются все входы и E²PROM не реагирует не на какие сигналы.

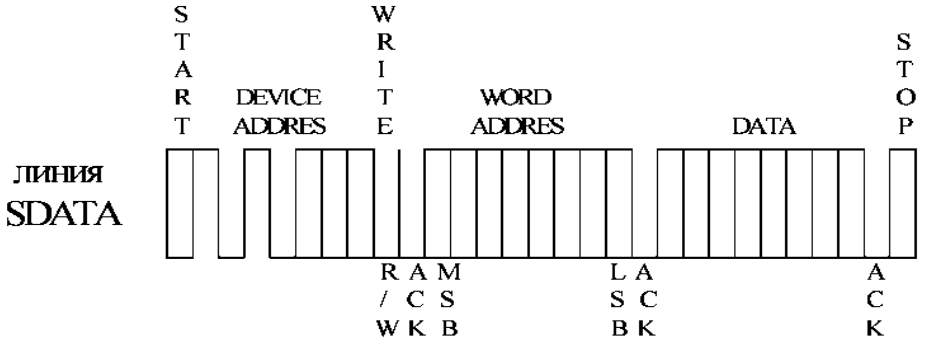

Рис. 3. Запись байта

Страничная запись

E²PROM может производить страничную запись по 8 байт. Процесс страничной записи инициируется так же, как и запись одного байта, отличие заключается лишь в том, что после записи первого байта ведущие устройство не выставляет стоп-сигнал. Вместо этого, как только E²PROM выдаст подтверждение приема первого байта, ведущее устройство может передать еще 7 байт данных, после чего выставляет стоп-бит. После приема каждого байта E²PROM выставляет сигнал подтверждения.

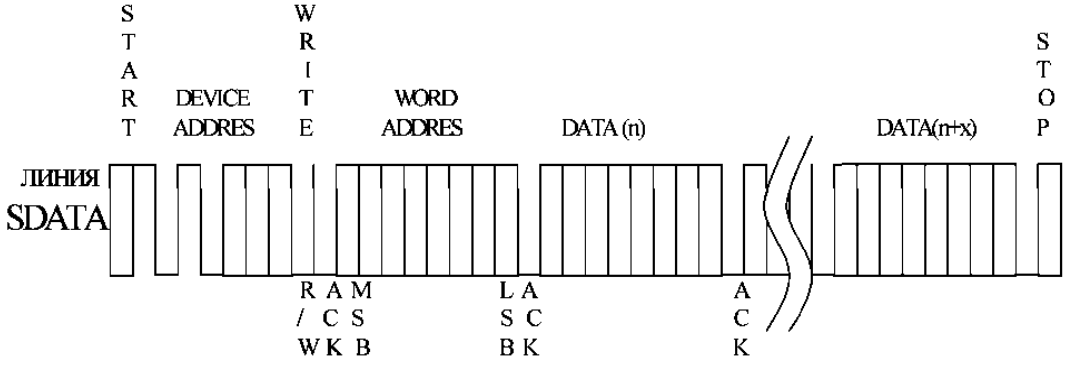

Рис. 4. Страничная запись

1.4.2. Чтение

Чтение по текущему адресу

Производится чтение по текущему адресу. Для этого, после того как устройство вышлет бит подтверждения, ведущие устройство принимает байт данных, но после этого вместо бита подтверждения высылает стоп-бит.

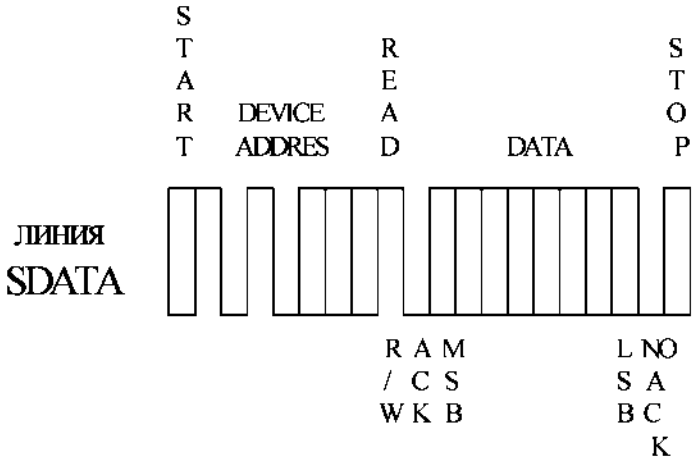

Рис. 5. Чтение по текущему адресу

Чтение в режиме произвольного доступа

В этом режиме сначала устройство передает бит внутреннего адреса. После этого еще раз передается стартовый бит и адрес устройства, только теперь в качестве операции указывается чтение. После прихода бита подтверждения читается байт данных.

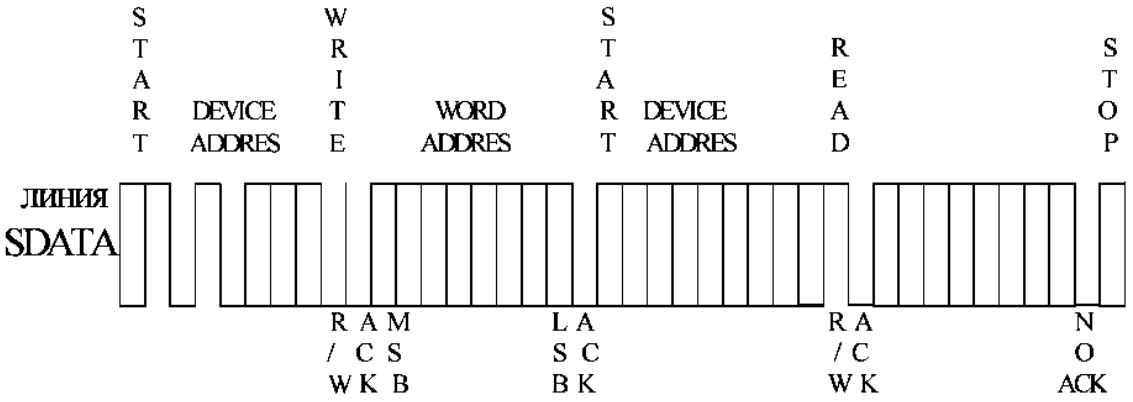

Рис. 6. Чтение в режиме произвольного доступа

Чтение в режиме последовательного доступа

Последовательное чтение может быть осуществлено как с текущего адреса, так и с произвольного адреса. Байты последовательно считываются из памяти после каждого полученного байта, ведущее устройство

выдает бит подтверждения. Операция чтения будет производиться до тех пор, пока ведущее устройство не выдаст на линию стоп-бит вместо бита подтверждения.

2. Требования к выполнению работы

1. В программе должна быть продемонстрирована работа с шиной I^2C по прерыванию.

2. Должен быть предусмотрен контроль ввода корректных значений в рамках выполнения прикладной задачи.

3. Каждый драйвер должен быть оформлен в отдельный модуль (файл).

4. Текст программы должен соответствовать правилам оформления программ на языке Си, приведенным в Приложении. Требования к оформлению программ на языке Си.

3. Содержание отчета

1. Титульный лист.

2. Номер варианта, задание.

3. Иллюстрация организации и функционирования разработанного программного обеспечения (драйверы, прикладная программа) в виде блок-схемы, диаграммы процессов, потоков данных, диаграммы состояний автоматов и других схем поясняющего характера (по выбору студента). Главное – это описание, что конкретно делает разработанная программа (какие функции она выполняет).

4. Разработанные протоколы, форматы данных и др.

5. Исходный текст программы с комментариями.

6. Основные результаты.

Лабораторная работа № 14
ИЗУЧЕНИЕ *ЦАП* И *АЦП* ЛАБОРАТОРНОГО СТЕНДА SDK 1.1

Цели работы: изучить архитектуру блоков ЦАП и АЦП лабораторного стенда SDK-1.1; разработать программы управления ЦАП и АЦП обработки аналоговых сигналов; просимулировать программы в отладчике-симуляторе; загрузить и выполнить программы на лабораторном стенде.

1. Методические указания к работе

1.1. Реализация АЦП в ADuC812

1.1.1. Передаточная функция АЦП

Диапазон входных напряжений АЦП составляет от 0 до +Vref. Для этого диапазона смена кодов происходит посередине соответствующего кванта (т. е. 1/2 LSB, 3/2 LSB, 5/2 LSB, ..., FS -3/2 LSB). Выходной код – прямая в двоичном коде с lLSB = FS/4096 или 2.5/4096 = 0.61 Мв при Vref = 2.5 В. Идеальная передаточная функция показана на рис. 1.

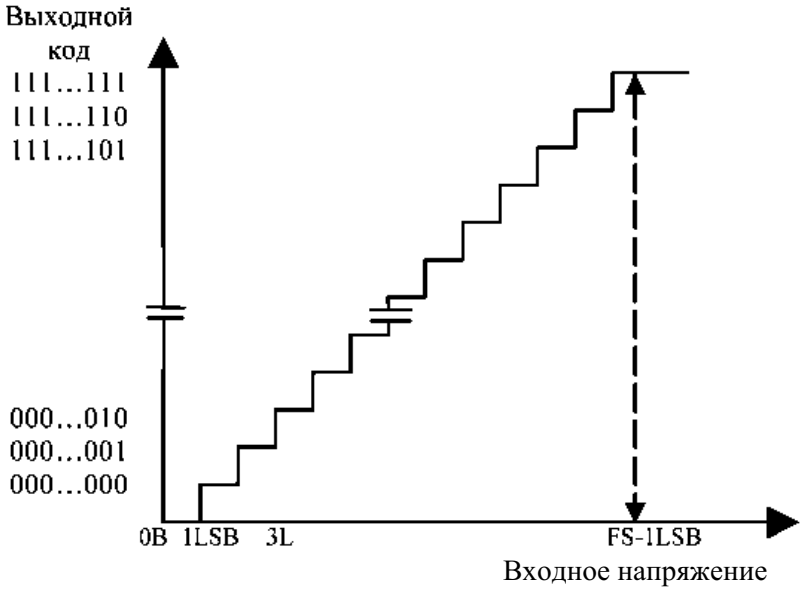

Рис. 1. Идеальная передаточная функция АЦП

Управление и настройка АЦП осуществляется при помощи 3 SFR регистров. Результат преобразования в 12-битном формате записывается в ADCDATAH/L. В первые четыре бита регистра ADCDATAH записывается биты выбора канала. Следующий формат слова результата представлен на рис. 2.

ADCCON1 (адрес SFR EFh) – регистр управляет преобразованием, временем переключения, режимом преобразования, токопотреблением устройства. Формат представлен в табл. 1.

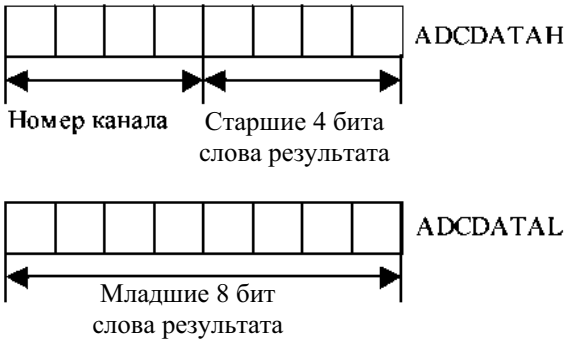

Рис. 2. Формат слова результата ADCDATAH/L

Таблица 1

Формат регистра ADCCON1

7–6	MD1 MD0	Два бита выбирают режима АЦП. MDl MD0 Режим 0 0 дежурный 0 1 нормальный 1 0 дежурный, если не выполняется цикл преобразования 1 1 холостой, если не выполняется цикл преобразования
5–4	CLK1 CLK0	Биты деления тактовой частоты. Устанавливают коэффициент деления основной частоты, используемой для генерирования частоты работы АЦП CLK1 CLK0 Делитель 0 0 1 0 1 2 1 0 4 1 1 8
3–2	AQ1 AQ0	Биты задержки переключения выбирают время, необходимое для перезарядки УВХ при переключении мультиплексора. AQ1 AQ0 Число тактов задержки 0 0 1 0 1 2 1 0 3 1 1 4
1	T2C	Бит запуска преобразования от таймера 2. Если бит установлен, то сигнал переполнения таймера 2 используется для АЦП
0	EXC	Бит разрешения внешнего запуска. Если бит установлен, то разрешен внешний запуск от низкого уровня на контакте 23 (CONVST). Активный низкий уровень должен поддерживаться на контакте не менее 100 нс

ADCCON2 (адрес SFR D8h) – регистр управляет режимом преобразования и выбором канала.

Таблица 2

Регистр ADCCON2

<table>
<tr><td>7</td><td>ADCI</td><td>Бит прерывания АЦП.
Устанавливается аппаратно в конце каждого цикла преобразования или в конце передачи блока данных в режиме прямого доступа к памяти. Сбрасывается аппаратно после перехода процессора на процедуру обработки прерывания</td></tr>
<tr><td>6</td><td>DMA</td><td>Бит разрешения прямого доступа к памяти.
Устанавливается пользователем для начала операции передачи в режиме прямого доступа к памяти</td></tr>
<tr><td>5</td><td>CCONV</td><td>Бит разрешения циклического преобразования.
В этом режиме АЦП, после того как преобразование закончилось, начинает следующее преобразование. Параметры работы АЦП должны быть заранее сконфигурированы</td></tr>
<tr><td>4</td><td>SCONV</td><td>Бит однократного преобразования.
Устанавливается пользователем для инициализации однократного цикла преобразования. После того, как цикл завершился, этот бит автоматически сбрасывается в 0</td></tr>
<tr><td>3–0</td><td>CS3
CS2
CS1
CS0</td><td>Биты выбора канала. Позволяют пользователю программно выбирать номер канала, для которого будет осуществляться преобразование. Для режима прямого доступа номер канала будет зависеть от идентификатора во внешней памяти
<table>
<tr><th>CS3</th><th>CS1</th><th>CS2</th><th>CS0</th><th>Номера входных каналов</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>2</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>3</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>4</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>5</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>6</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>1</td><td>7</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>Температурный сенсор</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>Остановка КПД (DMA)</td></tr>
</table></td></tr>
</table>

ADCCON3 (адрес SFR F5h) – регистр дает пользователю доступ к флагу занятости АЦП.

Таблица 3

Регистр ADCCON3

7	BUSY	Флаг занятости ЦАП (только для чтения). Устанавливается аппаратно на время цикла преобразования или калибровки. Автоматически сбрасывается ядром в конце преобразования или калибровки
6–0		Биты зарезервированы для дальнейшего использования

1.1.2. Частота тактирования

Для корректной работы АЦП частота тактирования должна находится между 400 кГц и 4 МГц, а оптимальными являются частоты от 400 кГц до 3 МГц. Время преобразования составляет 15 тактовых интервалов плюс 1 интервал для синхронизации, плюс задержка переключения. Например, если задержка переключения 4 тактовых интервала, то время одного преобразования будет составлять 20 тактовых интервалов. Таким образом, итоговая частота работы АЦП может быть вычислена по следующей формуле:

Частота = част.такт.АЦП/((16+задержка_перекл.)*делитель_частоты).

1.1.3. Режимы работы

Встроенный АЦП сконструирован таким образом, что может осуществлять выборки каждые 5 мс (частота 200 КГц). Таким образом, процессору за 5 мс необходимо прочитать значение выборки и записать его во внешнюю память. Если он не успеет этого сделать, то значение выборки будет потеряно. Если управление осуществляется прерыванием, то процессору еще надо будет перейти на процедуру обработки прерывания. Для приложений, в которых процессор не может поддерживать такую высокую скорость обработки прерываний, и предназначен режим прямого доступа к памяти (DMA – *Direct Memory Access*).

Для установки этого режима надо выставить 6 бит (DMA) в регистре ADCCON2. Это приведет к тому, что результаты преобразования будут записываться во внешнюю статическую память, минуя ядро процессора. Для работы в этом режиме необходимо произвести следующие настройки:

1. АЦП переводится в дежурный режим сбросом старших двух бит регистра ADCCON1.

2. Устанавливается адрес начала области памяти, куда будут записываться результаты работы АЦП. Для этого адрес начала области памяти записывается в регистры DMAL, DMAH, DMAP. При этом сначала заполняется DMAL, потом DMAH, а затем DMAP.

3. Размечается внешняя память. В старшие четыре бита каждого второго байта внешней памяти, начиная с адреса, записанного в регистры DMA, записывается идентификатор канала. Поскольку в этом режиме АЦП не зависит от ядра, необходимо завершить разметку памяти стоповой командой. Это делается дублированием идентификатора последнего преобразуемого канала, за которым в следующем слове следует стоповая последовательность 1111. Типичная последовательность разметки памяти приведена на рис. 3.

4. DMA инициализируется записью в SFR-регистры АЦП следующей последовательности:

a) в регистре ADCCON1 устанавливается бит разрешения режима DMA;
b) производятся настройки в регистре ADCCON1, устанавливается тактирование от таймера 2 или от внешнего триггера;
c) инициируется преобразование. Это делается началом единичного/циклического преобразования (АЦП переводится в рабочий режим) и запуском таймера 2 или подачей сигналов на внешний триггер.

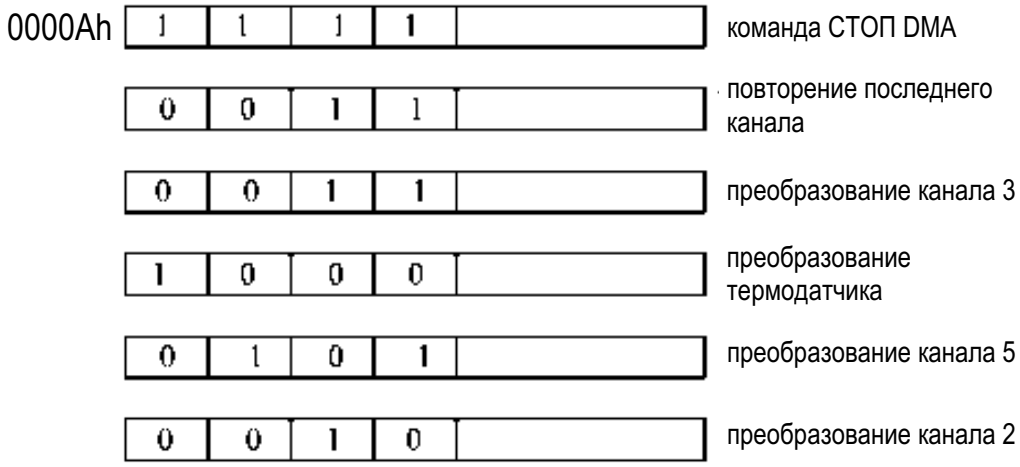

Рис. 3. Разметки памяти для режима DMA

Когда DMA-преобразование завершается, аппаратно устанавливается флаг прерывания, а внешняя SRAM содержит результаты преобразования. Необходимо отметить, что в последние два слова размеченной памяти (содержащие стоп-команду и дубликат последнего преобразуемого канала) не записывается никаких результатов.

1.2. Термодатчик

Напряжение на выходе термодатчика в 600 мВ соответствует 25 °С, а увеличение напряжения на 3 мВ соответствует уменьшению температуры на 1 градус. Исходя из этого, преобразование кода в температуру производится следующим образом:

- код преобразуется в напряжение 0,61 мВ* (значение кода);
- полученное напряжение вычитается из 675 мВ (соответствует 0 °С), а разность делится на 3. Результат этих операций представляет собой значение температуры со знаком.

2. Описание тестовой программы и некоторых функций драйвера

Обработчик прерывания вызывается после завершения очередной операции преобразования. Значения регистров данных АЦП (ADCDATAL и ADCDATAH) записываются в буфер драйвера. Сначала записывается старший байт, потом сдвигается на 8 разрядов и переписывается младший байт.

InitADC() – инициализация АЦП. Для простоты инициализации производится жесткая настройка АЦП без возможности изменения настроек пользователем. Устанавливается циклическое преобразование 9-го канала (термодатчик) – ADCCON2 = 0 × 28. Делитель частоты – 8, число тактов задержки – 4, тактирование от таймера 2 – ADCCONl = 0 × 3F. Устанавливается процедура обработки прерывания и АЦП переключается в нормальный рабочий режим.

ConvertToTemp() – преобразует передаваемое значение кода в температуру в строковом формате. Вначале передаваемое значение избавляется от старшей тетрады, которая содержит идентификатор канала. После этого код преобразуется в соответствующее ему значение напряжения (61*значение_кода = напряжение в 0.01 милливольтах). Напряжение вычитается из 675 мВ (напряжения 0 ºС) и делится на 300. Теперь переменная *Temperature* содержит значение температуры в градусах. Если знак переменной отрицательный, то температура положительна, поэтому значение умножается на –1 и в начало строки заносится знак температуры «+». В противном случае в начало строки заносится «–». В переменную *Grad* заносится 1/6 доли градуса. После всего этого полученные переменные преобразуются в строку.

Приложение 1
КРАТКОЕ ОПИСАНИЕ СИМУЛЯТОРА AVSIM85

```
+-------------------+
¦ Командное меню    ¦
+-------------------+
Большинство команд вызываются по нажатию первой буквы этой
команды
+------------+
¦ УПРАВЛЕНИЕ ¦
+------------+
Ctl-C прерывает выполнение команды и вызывает переход к
пункту меню более высокого уровня
SIMULATION Function (управление процессом симуляции) клавиши
и MODE toggle (Esc)(клавиша переключения режимов) продолжают
работать в любое время.
+------------------+
¦ Ввод выражений   ¦
+------------------+
Используемые операторы:                  Операнды определяют-
ся следующим образом:
     + - @                               CPU symbols - см.
файл AVSIM51.REG
     ()   до 4 уровней                   symbols  - таблица
символов пользователя
                                         $          Текущее
сост. прогр. счетчика
Указатели на систему счисления:          .          результат
предыдущего выраж.
     Binary:  %111   or  111B            'x' or «x» Символь-
ные константы
     Octal:   @377   or  377Q         digits     число в со-
ответствующей системе
     Hex:     $FF    or  FFH
+-----------------+
¦ ENTER MNEMONIC ¦
+-----------------+
Мнемоника вводится так же, как в ассемблере.
Операнды вводятся по правилам, описанным в предыдущем разде-
ле.
Формат строки:    {метка:}  операция  {операнды} {; коммен-
тарий}
+--------------+
¦ клав. редакт. ¦
+--------------+
```

```
←    курсор влево                  Esc       переключение
окон
→    курсор вправо                 ctl-C     выход в основ-
ное меню
<-   backspace                     INS       вставка
enter   ввод                       DEL       стереть символ
+-------------+
¦Точки останова¦
+-------------+
On Breakpoints, a delay can be specified by typing digits
BEFORE selecting
     the Breakpoint TYPE.
On sticky breakpoints, this delay is restored after each ac-
tivation.
+---------------------------+          +--------------------+
¦Клавиши управления режимами¦          ¦ Быстрые клавиши    ¦
+---------------------------+          +--------------------+
ESC:   реключатель режима.          ctl-A/B   Аккумулятор / B
Командное меню/окно отображения
F7: Режим курсора: Hex, ASCII,        ctl-P  Программный Bi-
nary счетчик
Ctl-PgUp:  Переключатель прокрутки             ctl-S     Ука-
затель стека
                                               ctl-R     Банк
регистров
   +---------------------+                     alt-0/7   Ре-
гистр:0/7
   ¦ управление курсором ¦                     ctl-D     DPTR
   +---------------------+                     ctl-CXFO  Фла-
ги: C/AC/F0/OV
←          влево                              ctl-I     Разре-
шение прерываний
→          вправо                             ctl-T     Таймер
0
↑          вверх                               alt-AB    DUMP
окно
↓          вниз                                alt-PQ    Пор-
ты: 0/2
HOME       к первому символу в окне            alt-SC    По-
следов.порт BUF/CON
END        к последнему символу в окне         alt-Y     Счет-
чик циклов
PgUp       на одно окно вверх                    +------------
-------------------+
PgDn       на одно окно вниз                     ¦ клав. Ре-
дактирования объектов ¦
```

```
                                                  +------------
------------------+
   -----между окнами-----                        +/-
inc/dec byte/word/flag
Return    Вернуться к предыдущему               INS   Переклю-
чатель byte/nibble/bit
ctl-→     на одно окно вправо                   ctl-END   очи-
стка до конца
ctl-←     на одно окно влево                    ctl-HOME  очи-
стка оставшейся части
+----------------------+
¦ Управление симуляцией ¦
+----------------------+
Запуск: -->  F1   Переключатель вкл./выкл. - включение симу-
ляции до конца программы (точки останова) или повторного на-
жатия
     -->  F10  Пошаговое выполнение  - выполнение одной ко-
манды
     -->  F9   Undo - Отмена одной команды
Точки останова: --> F2   Передвинуть курсор точек останова
вверх
      --> F4   Передвинуть курсор точек останова вниз
      --> F3   Установка точки останова - уст. точку остано-
ва в текущее положение курсора точек останова
F5   скорость - установка скорости симуляции
+----------------+
¦ TOGGLE CONTROL ¦
+----------------+
ALT-F5  LABEL Toggle - «LABEL»: Addresses & operands are
displayed symbolically
                       «ADDR» (top left of screen): No sym-
bols in disassembly
F6    DISPLAY Toggle - ON:  Screen is updated after each in-
struction during GO
                       OFF: Only TRACED windows are updated
until trap occurs
ALT-F6  TRACE Toggle - ON:  Window is updated even when dis-
play OFF
                       OFF: Window is updated only when dis-
play ON
F7       CURSOR TYPE - Hex / Ascii / Binary
                       Cursor will move to preferred screen
object
                       type, if displayed as several types
F8       SKIP Toggle - SKIP ON will Single Step across call
opcodes
                       (by setting a bkpt at the next in-
struction
```

and GOing with display OFF)

Команды меню

Dump
   Select DUMP Area: 1 2
   DUMP: Absolute Indirect
   Задает область резидентной памяти, отображаемой в 1-м и 2-м окнах
   Обозначения адресов: D:адрес – резидентная память данных (можно не указывать)
                         X:адрес – внешняя память данных
                         C:адрес – память программ
                         B:адрес – область прямо адресуемых битов
                  Absolute – абсолютный адрес
                  Indirect – косвенный адрес

Expression
   Enter Expression:
   Expression will be stored at __
   Задает выражение, которое будет вычислено и записано в указанный регистр
commandFile
   Load/Save keypressed sequences
   COMMAND FILE: Load SAVE: Open Close Restart
   Задает файл с последовательностью нажатия клавиш
Help
   Commands Display Simulation Avocet
   Вызов справки по указанным разделам
IO
   IO FILE: Open Close
   Задает файл ввода/вывода данных
Load
   LOAD: Avocet Data Program Symbol-table rOm
   Загрузка в память программы или данных из внешнего файла
Memory
   MEMORY: Clear Fill Move Search searchNext
      LOWER Address
      UPPER Address
   Установка или поиск данных в резидентной памяти данных
Patch
   PATCH: Patch code Open output file Close file
   Заполнение резидентной памяти программ
Quit
   QUIT: Exit
   Выход из программы

```
Reset
   RESET:  Cpu  Disptrace  cYcles
   Сброс процессора, трассировки или счетчика циклов
Set
   SET:  Memory-map  Passpoint  opTions  cYcles  V-drive
   Установка карты памяти, точек останова, счетчика циклов
   SET BREAKPOINT:  Conditional  Dynamic  Opcode  Sticky
   Установка точек останова
setUp
   SETUP:  Undo
   Задание числа шагов отмены
View
   VIEW:  Bkpts  IO-files  Memory-map  Opc-traps  Passpts
Symbols
   Просмотр параметров отладки
eXecute
   Execute Instruction
   Ввести и выполнить команду
```

Приложение 2
ЯЗЫК АССЕМБЛЕРА AVSIM85. ОСНОВНЫЕ СВЕДЕНИЯ

1. Языки ассемблера

Основные общие особенности языка ассемблера (чаще не совсем точно называемого просто ассемблером; строго ассемблером называется программа, которая переводит последовательность команд с языка ассемблера на язык машинных кодов процессора) микропроцессоров совпадают с особенностями всех языков подобного типа.

Языки ассемблера являются машинно-ориентированными языками и, следовательно, для каждого типа процессоров существует свой язык. Почти каждая команда ассемблера эквивалентна команде на машинном языке процессора. Однако программирование на ассемблере, по сравнению с программированием на машинном языке (на уровне машинных кодов), существенно облегчается за счет возможности использования символического обозначения всех элементов программы (кодов операций, адресов ячеек памяти, программ и данных, переменных и констант, операндов и т. д.). Используемые символические обозначения элементов обычно отражают их содержательный смысл. При программировании на языке ассемблера программист может не заботиться о распределении памяти, о назначении конкретных адресов операндам. В ассемблере допускается оформление повторяющейся последовательности команд как одной макрокоманды. Соответствующие версии языка, допускающие использование макрокоманд, называют макроассемблерами. Кроме того, ассемблеры позволяют в той или иной форме использовать при программировании стандартные структуры типа цикл, разветвление.

При программировании на ассемблере доступны все ресурсы системы и конкретного процессора (регистры, стек, память и т. д.). Это позволяет получать эффективные программы с точки зрения времени их выполнения и объема памяти, необходимого для размещения программы. Проблемы, связанные с конкретной аппаратурой и периферийными устройствами процессора, лучше и удобнее решать на языке ассемблера. Однако программирование на ассемблере предполагает знание архитектуры и свойств процессора, т. е. всего того, что входит в понятие «программная модель процессора».

Современные версии языков ассемблера предоставляют программисту ряд возможностей, характерных для языков высокого уровня, таких как условное ассемблирование, организация циклов арифметического и условного типа, т. е. позволяют использовать стандартные логические структуры, рекомендуемые методами структурного программирования.

Ниже рассмотрены основные сведения о языке ассемблера пакета AVSIM85 v2.02 для МП Intel 8080/8085 (КР580/КР1821).

2. Структура программы на языке ассемблера

Предложения (строки) ассемблера

Исходная программа на языке ассемблера состоит из последовательности утверждений, которые называют также ассемблерными строками или предложениями.

В ассемблерной строке могут быть записаны директивы ассемблера, команды или инструкции процессора, команды препроцессора, макрокоманды и комментарии. Запись строки производится в соответствии с некоторыми **формальными правилами**. Нарушение этих правил приводит к большому количеству ошибок, особенно на первом этапе освоения ассемблера.

Директивы ассемблера не порождают машинные команды и какие-либо действия в процессоре. Они задают структуру программы, сообщают транслятору и компоновщику информацию о том, что им надо делать с командами и данными. Примеры директив:

end – определяет логический конец программы, все записанное после нее не будет восприниматься транслятором;

seg – определяет начало новой секции;

db – размещает в памяти константы.

Команды или инструкции процессора порождают машинные команды и выполняются в заданной последовательности во время работы процессора. Примеры команд:

MOV A,B

ADD B

Команды препроцессора являются фактически разновидностью директив, которые выполнятся на первом шаге трансляции. Пример команды препроцессора:

% include f1.asm – помощью данной команды в исходный текст будет вставлен текст из файла **f1.asm.**

Макрокоманды и некоторые конкретные директивы будут рассмотрены ниже.

Комментарии не влияют на результат трансляции и служат для пояснения и описания программы. Комментарии без изменений переносятся в файл, получаемый после трансляции, – листинг трансляции. Вся строка ассемблера может являться комментарием. В этом случае она начинается специальным символом: [*] или [;].

Строка (предложение) программы в ассемблере делится на несколько полей, разделенных одним или более пробелами. В строке могут быть следующие поля:

- поле метки;
- поле мнемоники;
- поле операнда;
- поле комментария.

Таким образом, строка имеет следующий формат:

[метка[:]] <мнемоника> [операнд] [; комментарий]

Здесь, как и обычно, при описании системы команд и синтаксиса языка, для обозначения необязательного элемента конструкции использованы квадратные скобки [ ]. Таким образом, метка, операнд, комментарий являются необязательными элементами и могут отсутствовать.

Примеры ассемблерных строк:

```
MM2     ADD C       ; команда сложения, MM2 – метка
MM3:
        SUB D       команда вычитания, MM3 – метка
```

Поле метки

Метка в общем случае является необязательным элементом ассемблерной строки. Для некоторых директив наличие метки обязательно. Метка начинается в первой позиции строки и может содержать (как и любой идентификатор) алфавитно-цифровые знаки (A-Z, a-z, 0-9, _ и $), первым из которых должна являться буква.

Если в первой позиции строки стоит пробел или символ (;), то считается, что метка отсутствует. Метка может заканчиваться двоеточием, которое не входит в состав метки.

На линии может стоять одна метка, а последующие части строки ассемблера располагаться на следующих линиях без знака продолжения строки. В этом случае наличие двоеточия в конце метки обязательно.

Поле мнемоники

Поле мнемоники начинается после первого пробела в строке и заканчивается одним или более пробелами. Поле мнемоники содержит одно из следующих утверждений:

- мнемоническое обозначение команды процессора, например **ADD, MOVE**;
- мнемоническое обозначение макрокоманды;
- мнемонику директивы ассемблера;
- мнемонику команды препроцессора.

Ассемблер проверяет допустимость кодов мнемоник по своей внутренней таблице кодов команд и директив, а затем по таблице макрокоманд. В случае отсутствия используемых мнемоник выдается сообщение об ошибке.

Поле операнда

Поле операнда определяет информацию, над которой в соответствии с командами производятся действия. Поле операнда начинается сразу за пробелом (или пробелами), заканчивающим поле мнемоники, и, в свою очередь, заканчивается одним или более пробелами. В поле операнда могут быть записаны константы, символы и выражения, состоящие из символов и констант. Эти конструкции языка описаны ниже. Если команда или директива требует нескольких операндов, то отдельные операнды разделяются запятыми, но не пробелами. Интерпретация поля операнда зависит от мнемоники соответствующей команды или директивы. Операнд определяет объекты, над которыми производятся операции, в качестве таких объектов могут быть конкретные значения (константы, переменные) или в какой-либо форме адреса конкретных значений (регистры процессора, ячейки памяти).

Основные конструкции ассемблера

При записи операндов и меток используются различные конструкции языка. Рассмотрим основные конструкции.

Константы

Константа является величиной, которая не изменяется в течение всего времени выполнения программы. Константы могут быть числовые и строчные.

Числовые константы могут быть записаны в одной из трех систем счисления – двоичной, десятичной или 16-ричной. Отрицательные константы ассемблером записываются в дополнительном коде (для положительных чисел представление в дополнительном и прямом кодах совпадают). Длина внутреннего представления константы равна 1 байт (8 двоичных разрядов) или 2 байта в зависимости от команды, с которой она используется.

Система счисления обозначается спецификатором, который может либо предшествовать константе, либо стоять в ее конце. В качестве спецификаторов используются:

H, h, $ – 16-ричная система;

@ (отсутствие спецификатора) – десятичная система;

B, b, % – двоичная система.

Если число в 16-ричной системе начинается с буквы, слева от нее должен быть дописан 0 (при спецификаторе **H**).

Пример

Коды команд	Команды
`3E 0C`	`MVI A,12`
`3E 0A`	`MVI A,@10`

```
3E 05                    MVI A,5
3E FB                    MVI A, -5
3E FA                    MVI A,@ -6
3E A1                    MVI A,0A1H
3E A1                    MVI A,$A1
3E A1                    MVI A,$0A1
3E 09                    MVI A,1001B
3E 09                    MVI A, %1001
C3 5634                  JMP 03456h
```

В примере приведен фрагмент листинга, полученного в результате трансляции программы, команды которой приведены во второй колонке. В первой колонке приведены полученные коды. Вторым операндом во всех командах MVI являются константы, записанные в различных системах счисления. Первым байтом кода во всех этих случаях является код 3E, соответствующий MVI A. Константа, записываемая в регистр A, представлена в 16-ричной системе вторым байтом кода. Следует обратить внимание на запись чисел 5 и –5 в 3-й и 4-й командах. В команде JMP операндом является адрес перехода, который может иметь длину 2 байта.

Символы (символические имена)

Символы используются в качестве меток или идентификаторов. Можно использовать знаки A–Z, a–z, 0–9 и _(подчеркивание). Общее количество знаков – до 31. В качестве знака в строке не может использоваться пробел. Заглавные и строчные буквы рассматриваемый ассемблер воспринимает как одинаковые (например, символы ABC и abc).

Символы, которые используются как метки, становятся идентификаторами адресов ячеек памяти, в которых хранятся кодовые слова программы. Они могут применяться в поле метки только один раз.

Символы, используемые в поле операнда, становятся идентификаторами переменных величин, т. е. идентификаторами адресов ячеек памяти данных, где хранятся эти величины или идентификаторами констант. Эти символы должны быть определены либо директивами ассемблера непосредственно в программном файле, либо в командном файле компоновки.

Символы действуют только внутри некоторого программного модуля, т. е. для каждого символа имеется некоторая область видимости.

Некоторые символы (помимо мнемоник команд и директив ассемблера) являются стандартно определенными или предопределенными, т. е. они зарезервированы ассемблером и не могут использоваться в качестве идентификаторов. Таковыми являются, например, имена регистров РОН. Кроме того, не следует использовать символы: ***x, y*** и ***M.***

Выражения

Выражением является последовательность констант, символов, функций, объединенных арифметическими операторами и круглыми скобками. Выражение может являться операндом или частью операнда команды процессора или директивы.

Выражение вычисляется при трансляции, т. е. ассемблер может взять на себя задачу расчета фактического значения операнда. Смысл результата вычисления определяется местоположением выражения в программе. Например, результат может являться адресом или значением какой-либо величины (например, величины сдвига). Вычисление выражения подчиняется правилам алгебры и булевой алгебры.

Три главных фактора определяют порядок вычисления выражения:

- круглые скобки;
- ранги арифметических и логических операторов;
- направление вычислений.

Выражения обычно вычисляются слева направо, если другой порядок вычисления не определяется круглыми скобками или рангами операций. Внутри скобок направление вычислений – также слева направо при равенстве рангов операций.

Выражения могут быть абсолютными и относительными (перемещаемыми). Абсолютное выражение состоит из абсолютных величин, например констант.

Относительное – содержит относительные (перемещаемые) величины, которые могут изменяться, например метки, которые могут меняться при изменении размещения программы в памяти (при компоновке).

Операторы

Оператор является символом операции, которая должна быть выполнена. Среди операторов выделяют унарные, т. е. операторы, которые используются с одним операндом (например, знаки числа). Ранги операторов определяют порядок выполнения соответствующих операций в выражении. Некоторые используемые в языке операторы и примеры приведены в табл. 1.

Логические операции выполняются поразрядно над совпадающими разрядами двух чисел. Результатом операции сравнения является логическое значение ДА/НЕТ (True/False), которое обозначается двоичным 1/0. Они записываются во все двоичные разряды чисел. Примеры будут приведены в выражениях ниже. Операторы LOW и HIGH определяют байт, который будет использован из слова длиной в 2 байта.

Таблица 1

Оператор	Операция	Пример	Комментарий (результат)
Арифметические операции			
+	Унарный плюс (знак числа)	+5	
+	Сложение	Z+2	Сумма
–	Унарный минус (знак числа)	–10	
–	Вычитание	t – 7	Разность
*	Умножение	5*6	Произведение
/	Деление	8/3	Результатом будет целая часть частного: 2
MOD	Значение по модулю	31 MOD 6	31 по модулю 6 будет равно 1
Логические операции			
NOT	Поразрядное НЕ	NOT 21	21 = 15h (результат 0EA)
AND	Поразрядное И	31 AND 1	Результат 1
OR	Поразрядное ИЛИ	31 OR 1	Результат 1Fh
XOR	Поразрядное сложение по модулю 2	31 XOR 1	Результат 1Eh
Операции сравнения			
EQ	=	5 EQ 6	False – 0
NE	= / (не равно)	5 NE 6	True – 1
LT	<	5 LT 1	False – 0
LE	< =	5 LE 5	True – 1
GT	>	6 GT 5	True – 1
GE	> =	7 GE 8	False – 0
Определение типа операнда			
LOW	Младший байт слова	LOW 3456h	56h
HIGH	Старший байт слова	HIGH 3456h	34h

Для записи логических операторов, операторов сравнения и определения типа операнда можно использовать как строчные, так и прописные буквы.

Ниже приведен фрагмент листинга, полученного в результате трансляции программы, команды и директивы которой приведены в третьей колонке. В первой колонке приведены полученные коды.

Для директив **equ** эти коды являются результатом вычисления выражения, являющегося операндом директивы.

Вторым операндом во всех командах **MVI** являются константы, записанные в различных системах счисления. Первым байтом кода во всех этих случаях является код 3E, соответствующий команде **MVI A**, вторым байтом является результат вычисления выражения.

```
 = 0003 k        equ     1+2
 = 0005 z        equ     5
 = 0008 t        equ     Z+3
 = 0007 k        equ     +7
 = FFF9 l        equ     -7
 = 000A x1       equ     10
 = FFF7 x3       equ     1-X1
 = 0032 Y100     equ     x1*z
 = FFFE z4       equ     k-x1
 = 0002 u0       equ     4/2
 = 0002 n1       equ     7/3
 = 0002 n2       equ     8/3
 = FFEA n3       equ     not 21
 = 0001 n4       equ     31 mod 6
 = 0001 n6       equ     31 And 1
 = 001F n7       equ     31 or 1
 = 3456 n8       equ     3456h
0000' 3E 56              MVI     A, low n8
0002' 3E 34              MVI     A, high n8
 = 000D u2       equ     2*6+1
 = 000E u3       equ     2*(6+1)
 = 0000 n6       equ     5 EQ 6
 = FFFF N7       equ     6 le 10
 = 0000 u4       equ     2 LE 3 AND 2 EQ 5 ; результат FALSE
 = FFFF u5       equ     2 LE 3 AND 3 EQ 3 ; результат TRUE
 = 0000 u6       equ     2 LE 3 AND 3 EQ x1 ; результат FALSE
 = 0000 u10      equ     2 LE 3 AND 3 EQ y100 ; FALSE
 = 0000 u11      equ     2 LE 3 AND 3 EQ t ; результат FALSE
 = 0000 u12      equ     i LE z AND x1 EQ t
 = 0001 u15      equ     y2-two*(y2/two)
0006' 0E 0A              MVI     c,5*2
000A' 06 44              MVI     B,567+(6+7)
-----> test8.asm:62 ERROR 30: Operand out of range
```

```
-----> test8.asm:62 ERROR 22: Byte value not in the range -
128..+255
0015' 21 0B13          LXI     H,567*5
0018' D6 0D            SUI     u2
001E' D6 FC            SUI     n1+2
0026' E6 0A            ANI     x1
005C' FE 01            CPL     u/2
```

Результат вычисления операнда директивы может быть величиной в 2 байта. Операнд в команде **MVI** может быть величиной размером в 1 байт; поэтому для команды **MVI B,567+(6+7)** транслятор выдал сообщения об ошибке ERROR 30 и ERROR 22, которое состоит в том, что операнд находится вне допустимого диапазона (–128...+255). Для команды **LXI** , которая загружает величину в пару регистров, допускается операнд величиной в 2 байта.

Средства макроассемблера

Язык макроассемблера отличается от языка ассемблера наличием макросредств, облегчающих составление программ на языке ассемблера. К таким средствам относят макрокоманды, средства организации повторений (циклов) отдельных команд и блоков команд, организацию условного ассемблирования (трансляции) и т. д.

Следует отметить, что макросредства не готовят другой конечный продукт, но избавляют программиста от некоторой рутинной работы, например от многократной записи одной команды при необходимости ее повторения.

Рассмотрим здесь только макрокоманды.

Макрокоманды

При разработке программ часто возникает необходимость в повторении (иногда с модификациями, при других параметрах) некоторой группы команд. Такие группы повторяющихся команд можно оформить как процедуру или подпрограмму. Макроассемблер позволяет применить другой вариант краткой ссылки на используемую часто последовательность команд: такую последовательность можно определить один раз как большую команду – макрокоманду с уникальной мнемоникой, не совпадающей с мнемоникой команд процессора. Макрокоманду после определения ее в начале программы можно рассматривать как входящую в систему команд процессора.

Таким образом, использование подпрограмм (процедур) и макрокоманд позволяет сократить длину исходной программы и улучшить ее «читаемость». Но между этими конструкциями языка имеются существенные различия.

При использовании подпрограмм в текст исходной программы включается команда **CALL**, которая реально будет выполняться в процессоре. При ее выполнении происходит передача управления в другое место памяти программ.

При использовании макрокоманды ее имя (мнемоника) ассемблером заменяется при трансляции на последовательность команд, определяемых этой мнемоникой. Вышеуказанное приводит к тому, что время выполнения программы при использовании макрокоманд сокращается (по сравнению с использованием подпрограмм) за счет отсутствия команд переходов. При этом длина выполняемой программы (и место, занимаемое программой в памяти) увеличивается за счет многократного повторения команд процессора, входящих в макрокоманду.

Макрокомандами можно также объявить часто используемые программистом типовые последовательности действий (процедуры).

Из макрокоманд можно создать библиотеку. Макрокоманды могут быть вложенными друг в друга.

Макроопределение и макровызов

Макроопределением является набор исходных ассемблерных строк, включающий команды процессора, макрокоманды и директивы ассемблера, который должен выполняться процессором в соответствии с определяемой макрокомандой.

Макроопределение обычно располагается либо в начале исходной программы (во всяком случае, до вызова макрокоманды), либо в библиотеке макроопределений.

Макроопределение обычно имеет следующую структуру:

<имя макрокоманды> директива начала % MACRO [список параметров] [; комментарий]

……...

последовательности команд и директив – тело макрокоманды

……

директива конца макро % ENDM

Директива начала ***% MACRO*** и директива конца макро **% ENDM** являются макродирективами, т. е. директивами, которые открывают и завершают макроопределение, соответственно.

Имя макрокоманды размещается в поле метки, т. е. начинается с первой позиции ассемблерной строки. Последовательность параметров макрокоманды ***[список параметров]*** записывается через запятую в поле операнда ассемблерной строки. Параметры в макроопределении являются формальными и заменяются на фактические после макровызова. Макроопределение может не иметь параметров. Через параметры в макроопределении могут задаваться любые операнды и метки.

Макровызов имеет следующую структуру:

[метка] <имя макрокоманды> [список параметров] [; комментарий]

Имя макрокоманды, расположенное в поле мнемоники команды, должно совпадать с именем, расположенным в поле метки соответствующего макроопределения.

Последовательность величин (или символов), являющихся фактическими параметрами, записывается через запятую в поле операнда ассемблерной строки. Параметры по количеству и порядку следования должны соответствовать формальным параметрам в макроопределении. Каждый параметр может быть константой или выражением любого типа, распознаваемого ассемблером.

Пример 1. Макрокоманда без параметров. В соответствии с данной макрокомандой величина, находящаяся в аккумуляторе, умножается на три. Результат остается в A.

```
UMN3 %macro   MOV C,A
         RLC ; сдвиг влево на 1 разряд эквивален-
      тен умножению на 2
         ADD C
          %endm
```

Транслятор вместо использованных в тексте программы двух макровызовов этой макрокоманды

```
UMN3
UMN3
```

подставит макрорасширения (фрагмент взят из листинга)

```
;UMN3 ; исходная макрокоманда превращена в ком-
ментарий
        MOV C,A
        RLC
        ADD C
;UMN3 ; исходная макрокоманда превращена в ком-
ментарий
        MOV C,A
        RLC
        ADD C
```

Пример 2. Макрокоманда T1 с параметрами и директивой. Макрокоманда использует 4 формальных параметра: p1, p2, p3, p4. Используемому внутри макрокоманды символу S присваивается некоторое значение. Так как символ S зависит от параметра и используется только внутри данной макрокоманды, он может принимать при каждом вызове макрокоманды другое значение, поэтому директивой %**local** он объяв-

ляется локальным, т. е. используемым только внутри команд, соответствующих одной макрокоманде.

```
T1          %macro  p1,p2,p3,p4
      %local s
S           equ P1
     MVI   A,P2+S
     MVI   B,P3+S
     MVI   C,P4+S
      %endm
```

Транслятор вместо использованных в тексте программы двух макровызовов этой макрокоманды

```
T1  2,3,4,5
T1  6,7,8,9
```

подставит макрорасширения (фрагмент взят из листинга)

```
;T1 2,3,4,5 ; исходная макрокоманда превращена в комментарий
??0000 equ 2
MVI  A,3+??0000
MVI  B,4+??0000
MVI  C,5+??0000
;T1  6,7,8,9 ; исходная макрокоманда превращена в комментарий
??0001  equ  6
MVI  A,7+??0001
MVI  B,8+??0001
MVI  C,9+??0001
```

Транслятор вместо локального символа **S** для каждого макрорасширения подставляет свою перемещаемую величину ??0000 и ??0001 , абсолютное значение которой будет определено после компоновки. Вместо формальных параметров **p2, p3, p4** подставляются фактические параметры, задаваемые в макрорасширениях.

Пример 3. Макрокоманда с параметром и локальными метками. Внутри тела макрокоманды в зависимости от результата выполнения команды **SUI p** производятся условные и безусловные переходы. Так как при каждом вызове макрокоманды адреса переходов будут различны (соответствующие метки переходов должны стоять в разных местах), директивой %**local** метки, обозначающие адреса переходов, объявляются локальными.

```
DEL  %macro p
       %local   L1
L1    SUI p
       %local   L2
      JP L2
       %local   L3
      JMP  L3
```

```
L2     INR  C
       JMP L1
L3
        %endm
```

Транслятор вместо использованных в тексте программы двух макровызовов этой макрокоманды

```
DEL 3
DEL 4
```

подставит макрорасширения (фрагмент взят из листинга)

```
??0000      ;DEL        3
            SUI         3
            JP          ??0001
            JMP         ??0002
??0001      INR         C
            JMP         ??0000
??0002
            ;DEL        4
??0003      SUI         4
            JP          ??0004
            JMP         ??0005
??0004      INR         C
            JMP         ??0003
??0005
```

Транслятор вместо локальных меток L1, L2, L3 для каждого макрорасширения подставляет свои величины, абсолютное значение которых будет определено после компоновки. Вместо формального параметра **p** подставляются фактические параметры, задаваемые в макрорасширениях.

Основные директивы и команды препроцессора ассемблера

Директивы

Основные директивы уже упоминались в предыдущих разделах. Перечислим их еще раз.

Директива **end**

Директива не имеет операндов и используется для обозначения конца программы. При ее отсутствии транслятор выдает сообщение об ошибке. Пример ее использования имеется в приведенных текстах программ.

Директивы **equ, teq**

Директивы используются для присваивания значений символам. Они имеют следующий формат:

<символ>] equ <выражение> [; комментарий]

<символ>] teq <выражение> [; комментарий]

Пример

```
Z equ 5 ; символу z присваивается значение 5
t equ Z+3 ; символу t присваивается значение 8
t1 teq t+t/z ; символу t1 присваивается значение 9
```

В соответствии с данными директивами символу, стоящему в поле метки (в первой позиции), присваивается результат вычисления выражения, стоящего в поле операнда. Если в выражении используется символ, то он должен быть определен ранее (выше в программе). Разница между директивами **equ, teq** состоит в том, что символ, определенный директивой **teq**, далее в программе может быть переопределен с помощью этой же директивы.

Директивы **extern, public**

Директивы используются для определения одних и тех же символов в различных программных модулях при использовании нескольких исходных модулей для получения конечной выполняемой программы. Формат директив:

extern <список символов>

public <список символов>

Пример

```
extern z
public k,x1,I
```

Директива **extern z** означает, что символ **z** не определяется в данном программном модуле, а будет определен в другом программном модуле, с которым данный будет объединен при компоновке. Символ, объявленный директивой **extern,** может быть также определен в командном файле компоновки. Транслятор отложит его вычисление до компоновки.

Директива **public k,x1,i** означает, что символы **k,x1,i** , значение которых определяется в данном программном модуле, будут также использованы в других. Транслятор запомнит их значение в выходном объектном модуле для использования при компоновке (в противном случае после присваивания значений транслятор «забывает» символы).

Ниже приведена часть текста программы, в которой используются эти директивы.

```
    extern  z
    public  k,x1,i
    x     equ 1+2
    t     equ  Z+3
    k     equ +7
    I     equ  -7
    x1    equ  10
    x5    equ  t+1
    Y     equ x1*t
X2  equ  z*t
X3  equ  5/z
```

Как видно из примера программы, неопределенные символы могут использоваться в выражениях, однако последнее выражение **5/z** вызовет

сообщение об ошибке при трансляции (деление на 0), так как в общем случае **z** может быть равно 0. Поэтому подобные выражения недопустимы.

Директивы **db, dw, ds**

Директивы используются для резервирования ячеек памяти и размещения в памяти констант и начальных значений переменных. Формат директив:

[метка] db <список выражений>

[метка] dw <список выражений>

[метка] ds <список выражений>

Примеры

```
M1          db  3,4,5,6
M2          db  3*4,5/2
```

В соответствии с данными директивами при загрузке программы в память в отдельные ее ячейки (в соответствии с размещением директивы в тексте программы) будут записаны величины **3**, **4**, **5**, **6** и результаты вычисления выражений **3*4** и **5/2**. Метки в этих директивах являются необязательными элементами. Если они присутствуют, то они являются символическими адресами ячеек, в которых размещены константы. Так **M1** будет являться адресом ячейки, где расположена константа **3**, а M2 адресом ячейки с константой 3*4.

Директива **db** размещает в ячейках значения длиной в байт. Директива **dw** размещает в двух последовательных ячейках слова длиной в 2 байта.

Пример

```
M3          dw  0AC5Eh
```

В соответствии с данной директивой в ячейку с символическим адресом **M3** будет записано число **0Ach**, а в следующую – число **5Eh**.

Директива **ds** резервирует (оставляет пустыми) в памяти для последующего использования при выполнении программы заданное число ячеек.

```
M3          ds  6
```

В соответствии с данной директивой в памяти будет зарезервировано 6 ячеек памяти. Первой пустой ячейке будет присвоен символический адрес **M3**. Место резервирования ячеек определяется расположением директивы в программе.

Директива **include**

Директива используется для включения одного файла в другой. Формат директивы

include <имя файла>

Следует отметить следующее. Включаемый файл не должен содержать в конце директиву **end**, так как после выполнения директивы **include** в середине основного окажется директива **end**, а транслятор ничего после нее воспринимать не будет. Включаемый файл должен находиться в рабочей директории. Директиву **include** нельзя использовать для включения в файл макроопределений, для этого используется команда препроцессора %**include**.

Команды препроцессора

Команды препроцессора были рассмотрены в подразделе «Средства макроассемблера». Напомним их назначение.

Команда %**include** используется для включения в файл библиотеки макрорасширений.

Команды %**macro** и **%endm** определяют начало и конец макроопределения.

Построение простой программы

В качестве «простой» рассмотрим вариант построения программы без использования механизма сегментов и соответствующих директив. Ниже приведен пример подобной программы.

Основная программа test.asm

```
        extern  x1,two
         %include lib1.asm  ; включение библиотеки макро-
расширений
    z     equ 5
    t     equ x1+3
    k     equ +7
    i     equ - 7
          include init.asm;  включение  команд  инициализации
процессора
    Begin MVI A,x1
          UMN3
          MVI B,z
          MVI  C,k+i
          ADI t
          SUI two
    M1    T1  z,t,k,i
          MOV  D,A
          PUSH  B
          MVI  B,1
          POP B
          include stack.asm   ;  резервирование  памяти  для
стека
          end
```

Программа не имеет какого-либо содержательного смысла и носит чисто учебный характер для отражения различных возможностей ее по-

строения. В программе используются объявленные директивой **extern** символы **x1,two**, значение которых будет определено при компоновке. В программе используются макрокоманды **UMN3** и **T1 z,t,k,i**. Макроопределения этих команд записаны в библиотеке **lib1.asm** и включаются в основную программу командой препроцессора %**include**. Использованные макрокоманды описаны в подразделе «Средства макроассемблера».

Файл lib1.asm

```
* тексты макроопределений
UMN3  %macro
      MOV  C,A
      RLC
      ADD  C
       %endm
T1     %macro   p1,p2,p3,p4
       %local    s
s     equ       p1
      MVI  A,p2+s
      MVI  B,p3+s
      MVI  C,p4+s
       %endm
```

В начале программы обычно записываются команды инициализации процессора; они могут задавать различные режимы, а также в первую очередь они должны определять таблицу векторов прерываний, организацию стека и разрешение прерываний (если прерывания и стек используются в программе). Соответствующие команды можно писать прямо в тексте программы. Поскольку эти действия носят типовой характер и могут повторяться в других программах, они могут быть вынесены в отдельный файл и включены в основную программу директивой **include**. В приводимом примере команды инициализации процессора записаны в файле **init.asm**. Таблица векторов прерывания обычно занимает 64 ячейки, и основная программа начинается с 64 (**40h**). Для выполнения этого условия в приводимом файле с помощью директив **ds** пропускаются пустые ячейки.

Файл init.asm

```
        JMP  INIT       ; таблица векторов прерывания
        ds   5
        JMP  M1
        ds   53
INIT    LXI  SP,stack   ; начало программы, организация стека -
             ; запись в указатель символического адреса
                        ; вершины стека
        EI              ; разрешение прерываний
        JMP  BEGIN      ; переход к основным действиям
```

Стек размещается в основной памяти процессора. Его обычно размещают в памяти после основной программы. Так как заполнение стека идет вниз (в сторону уменьшения адресов), после конца программы директивой **ds** резервируется 20 ячеек для стека. Символический адрес вершины стека **stack** будет соответствовать концу зарезервированной области памяти.

Файл stack.asm

```
    ds   20
stack                ;вершина стека.
```

Трансляция программы

Трансляция производится программой – транслятором **avmac85.exe**. Формат команды вызова транслятора:

avmac85 <имя транслируемого файла> [xref] [sm] [si],

где ***<имя транслируемого файла> –*** обязательный элемент;

[xref] – необязательная опция, задающая формирование таблицы перекрестных ссылок на символы, используемые в программе, в таблице указываются номера строк, в которых используются символы;

[sm] – необязательная опция, которая определяет наличие в листинге трансляции макрорасширений, подставляемых вместо макровызовов;

[si] – необязательная опция, которая определяет наличие в листинге трансляции команд из файла, включаемого директивой **include.**

В результате трансляции формируется объектный файл с расширением **obj**, листинг трансляции – файл с расширением **prn**, а также файл с перекрестными ссылками (при использовании опции **xref**) – файл с расширением **xrf**. Имена всех этих файлов соответствуют имени транслируемого файла.

Сообщения об ошибках, обнаруженных при трансляции, выводятся на монитор и включаются в листинг. Некоторые ошибки не позволяют сформировать листинг трансляции.

Форма листинга

Ниже приведен текст листинга трансляции программы, рассмотренной в качестве примера в подразделе «Построение простой программы». Листинг получен при трансляции с использованием опций **si, sm**.

В первой колонке листинга приводится:

- для команд ее относительный (относительно начала программы) перемещаемый адрес;
- для символов, определяемых директивой **equ**, их значение. Если символ является внешне определенным (определенным директивой **extern**), он обозначается как **X**.

Во второй колонке записывается код команды. Если в команде присутствует перемещаемый адрес (например, в команде перехода), то код сопровождается знаком **'** (**C3 0040'**). Если в команду входит неопределенное значение символа, то код сопровождается знаком *****.

```
1       2          3   4            5      6       номера колонок
листинга
                   1                extern x1,two
                   2   $SETLN(LIB1.ASM)   ; %include lib1. asm
                   1   * тексты макроопределений
                   2   ;UMN3          %macro
                   3   ;             MOV    C,A
                   4   ;             RLC
                   5   ;             ADD    C
                   6   ;             %endm
                   7
                   8   ;T1           %macro p1,p2,p3,p4
                   9   ;             %local s
                   10  ;S            equ    P1
                   11  ;             MVI    A,P2+S
                   12  ;             MVI    B,P3+S
                   13  ;             MVI    C,P4+S
                   14  ;             %endm
                   15
                   16
                   17  ;RESETLN
 = 0005            3   z             equ    5
 = …X              4   t             equ    Z+3
 = 0007            5   k             equ    +7
 = FFF9            6                 equ    -7
                   7                 include init.asm
0000'   C3 0040'   1                 JMP    INIT
0003'   (0005)     2                 ds     5
0008'   C3 0054'   3                 JMP    M1
000B'   (0035)     4                 ds     53
0040'   31 006F'   5   INIT          LXI    SP?Stack
0043'   FB         6                 EI
0044'   C3 0047'   7                 JMP    BEGIN
0047'   3E 00*     8   Begin         MVI    A,x1
                   9                 ;UMN3
0049'   4F         9                 MOV    C,A
004A'   07         9                 RLC
004B'   81         9                 ADD    C
004C'   06 05      10                MVI    B,z
004E'   0E 00      11                MVI    C,k+i
0050'   C6 ..X     12                ADI    t
0052'   D6 00*     13                SUI    two
                   14  M1            ;T1    z,t,k,i
 = 0005            14  ??0000        equ     Z
```

```
0054'   3E //X   14              MVI    A,T+??0000
0056'   06 0C    14              MVI    B,k+??0000
0058'   0E FE    14              MVI    C,i+??0000
005A'   57        15              MOV    D,A
005B'   C5        16              PUSH   B
005C'   06 01     17              MVI    B,1
005E'   C1        18              POP    B
                  19              include stack.asm
005B'   (0014)    1               ds     20
                  2   stack               ; вершина стека
                  20              end
```

В третьей колонке помещается номер строки соответствующего исходного или вставляемого файла. Для вставляемых файлов используется своя нумерация строк.

В четвертой колонке и далее помещаются поля метки, мнемоники, операнда и комментария, причем для макрорасширений локальные значения сопровождаются знаком «??».

Компоновка программы

Компоновку программы можно производить разными способами. Наиболее простым представляется использование командного файла компоновки. При этом команда вызова компоновщика имеет вид

Avlink @<имя командного файла компоновки>

например, ***Avlink @1.lnk***.

Для компоновки программы test.asm можно использовать следующий командный файл:

```
test = test.OBJ      ; имя выходного файла (test) и имена компонуемых
                     ; объектных модулей (*.obj)
                     ; опции:
– SYMBOLS            ; сохранение в выходном файле символов программы
                     ; для удобства отладки
– START(M,0H)        ; задание начального адреса размещения программы
                     ; в памяти
– DS(x1,5)           ; задание значений символов, не определенных в
– DS(two,6)          ; исходном файле
```

Приведенный пример командного файла содержит все необходимые опции при компоновке простой программы (без использования сегментов). В этом случае программа представляет собой одну общую последовательность кодов, для которой задается начальный адрес размещения ее в памяти. При необходимости опциями **DS** задаются значения символов. Следует отметить следующее: в приведенном примере записанные после знака (;) комментарии даны для пояснения. В выполняемом командном файле никакие комментарии не допустимы (все, что

записано после (;), будет игнорироваться, в том числе продолжение текста на других строках), все опции могут быть записаны подряд (в одну строку); в качестве разделителя можно использовать пробел, запятую или перевод строки.

Можно соединить последовательно две (или больше) простые программы при компоновке. В этом случае коды программ будут последовательно объединяться. Порядок объединения кодов определяется порядком записи имен объектных модулей. Например, при записи

test = test.OBJ,vstavka.obj

коды файла **vstavka.obj** будут в выходном файле следовать за кодами файла **test.OBJ**.

Результатом компоновки являются выходные файлы со следующими расширениями: ***.hex, *.sym** и ***.map**. Файл ***.hex** содержит исполняемые коды в 16-ричной системе, файл ***.sym** содержит символьную информацию, необходимую для отладки. Весьма важным файлом, который обязательно необходимо проверить после компоновки, является файл ***.map** (карта компоновки), который содержит информацию о выполненном размещении программы в памяти. Полученное размещение может отличаться от «заказанного» без вывода каких-либо сообщений об ошибках.

Для программы **test** был получен следующий файл: **test.map**

AVLINK --- LOAD MAP

For: Avocet 8085/Z80 Assembler v2.02, #01235

RELOCATED SEGMENTS – CLASS 'M'

SEGMENT NAME	START	STOP	LENGTH	ovl/cat def/undef
CODE	0000	0072	0073	Concat Defined

ZERO LENGTH SEGMENTS

SEGMENT START

DATA 0000

IOSPACE 0000

No Transfer Address.

В этом файле следует обратить внимание на величины START = 000 STOP = 0072 LENGTH = 0073, которые в 16-ричной системе дают начальный, конечный адрес и длину программы.

Если в командном файле не определить символы **x1** и **two**, появится следующее предупреждение:

WARNING: Symbol TWO undefined

WARNING: Symbol X1 undefined

Использование программы, состоящей из отдельных сегментов

Обычно мало-мальски сложная прикладная программа представляет совокупность нескольких единиц – модулей. Это следует также из принципов структурного программирования. Физической единицей программы является программный модуль (файл). Логической единицей программы на ассемблере является секция или сегмент. Функциональный модуль программы (подпрограмма, процедура и т. д.) может совпадать с секцией или объединять несколько секций.

Использование нескольких программных модулей для получения общей программы обусловлено существованием больших и сложных программ, совместной работой нескольких программистов, удобством разработки программы.

Механизм секций поддерживается современными языками ассемблера в связи с использованием в процессорах памяти с достаточно разветвленной (и иногда сложной) структурой памяти, состоящей из блоков различного назначения и с различными характеристиками. Программу, состоящую из нескольких секций, можно загружать в разные блоки и места памяти, распределяя различные секции программы в зависимости от назначения. Например, часть программы, представляющую команды инициализации процессора, которые выполняются один раз при включении системы и время выполнения которых не критично, можно разместить в медленной внешней памяти, а критичные, с точки зрения времени выполнения, фрагменты – в быстрой внутренней памяти процессора.

Отдельный программный модуль не связан с определенным количеством секций – модуль может содержать одну или несколько секций, в свою очередь секция может быть расположена в нескольких программных модулях.

Каждая секция может являться отдельным функциональным элементом программы. При этом секция может быть независимо настроена на размещение в определенном месте (в соответствии с требуемыми типом и адресом памяти) используемого пространства памяти. Эта настройка на конкретные адреса, так же как и размещение секций в определенном порядке, осуществляется компоновщиком

Секции (сегменты) программы прежде всего разделяются по назначению. Как правило, во всех языках используются (иногда этим и ограничиваются) секции трех видов: секции кодов программы, секции инициализируемых данных программы (т. е. данных, которые загружаются в память) и секции неинициализируемых данных (т. е. данных, появляющихся в процессе выполнения программы). Секции могут иметь имя, задаваемое программистом (пользователем).

В языке AVSIM механизм сегментов поддерживается специальными директивами.

Организация сегментов

Для организации сегментов используются несколько директив.

Директива **defseg**

Формат директивы

defseg <имя сегмента>[, атрибуты сегмента]

Директива определяет используемый в программе сегмент. Обязательным элементом является имя сегмента. Атрибуты сегмента (о них чуть ниже) являются необязательным элементом.

Директива **seg**

Формат директивы

seg <имя сегмента>

Данная директива открывает сегмент. Директива определения сегмента должна предшествовать директиве открытия. Она может стоять в начале программы.

Сегмент закрывается либо директивой открытия другого сегмента, либо директивой конца программы **end**.

СПИСОК ЛИТЕРАТУРЫ

1. Проектирование микропроцессорной электронно-вычислительной аппаратуры: справочник / В.Г. Артюхов и др. – Киев: Техника, 1988.
2. Гуртовцев А.Л., Гудыменко С.В. Программы для микропроцессоров. – Минск: Высш. школа, 1989.
3. Токхайм Р. Микропроцессоры. – М.: Энергоатомиздат, 1989.
4. Григорьев В.Л. Программное обеспечение микропроцессорных систем. – М.: Энергоатомиздат, 1989.
5. Гольденберг Л.М. и др. Цифровые устройства и микропроцессорные системы. – М.: Радио и связь, 1992.
6. Хоровиц П., Хилл У. Искусство схемотехники. – Т. 2. – М.: Мир, 1983.
7. Микропроцессоры и микропроцессорные комплекты интегральных схем: в 2-х томах / под ред. В.А. Шахнова. – М.: Радио и связь, 1989.
8. Учебный стенд SDK-1.1. Руководство пользователя.
9. ADuC 812. Многоканальный АЦП со встроенным микропроцессором и FLASH.
10. ADuC 812. Multichannel 12-Bit ADC with Embedded FLASH MCU.
11. Учебный стенд SDK-1.1. Руководство пользователя (Версия 1.0.11).
12. Принципиальная схема учебного лабораторного стенда SDK-1.1 (Rev 4, 03/2005).
13. PCF8583: Clock/calendar with 240 x 8-bit RAM Datasheet.
14. AT24C02A/04A: Two-wire Automotive Temperature Serial EEPROM (Rev D, 4/2007).
15. Specification WH1602A-NYG-CT (Rev 0, 04/2004).

ОГЛАВЛЕНИЕ

Учебное издание

АЛХИМОВ Юрий Васильевич

ЦИФРОВЫЕ И МИКРОПРОЦЕССОРНЫЕ УСТРОЙСТВА

Лабораторный практикум

Научный редактор
доктор технических наук,
профессор Б.И. Капранов

Редактор *М.В. Пересторонина*

Компьютерная верстка *В.П. Аршинова*

Дизайн обложки *О.Ю. Аршинова, О.А. Дмитриев*

Подписано к печати ••.••.2009. Формат 60x84/16. Бумага «Снегурочка».
Печать XEROX. Усл. печ. л. 9,82. Уч.-изд. л. 8,89.
Заказ ••••-09. Тираж 200 экз.

Томский политехнический университет
Система менеджмента качества
Томского политехнического университета сертифицирована
NATIONAL QUALITY ASSURANCE по стандарту ISO 9001:2008

ИЗДАТЕЛЬСТВО ТПУ. 634050, г. Томск, пр. Ленина, 30
Тел/факс: +7 (3822) 56-35-35, www.tpu.ru