Программа данного курса лабораторных работ предполагает изучение системы команд микроконтроллеров семейства MCS-51, а также ознакомление и приобретение навыков работы с соответствующими средствами разработки и отладки программного обеспечения на языке ассемблера. При выполнении лабораторных работ используется демонстрационная версия программы-отладчика PICE-51, разработанная российской фирмой «Phyton» [1]. В качестве модели для изучения в рамках курса выбран базовый элемент семейства MCS-51 – микроконтроллер AT89C51 производства компании «Atmel» [2]. Информацию о его архитектуре можно найти в [3–5]. Для выполнения лабораторных работ необходимы знания форматов представления чисел в различных системах счисления [6, 7, 12].

1. Описание интерфейса пользователя программы-отладчика

Запуск и конфигурирование программы-отладчика. Используемая при выполнении лабораторных работ программа PICE-51 версии 3.01.71 предназначена для работы под управлением операционной системы Microsoft Windows. Запуск программы производится двойным щелчком мыши по файлу **PICE-51.exe** из папки «Phyton». В появившемся после этого окне (рис. 1) следует щелкнуть мышью по кнопке «Демо».

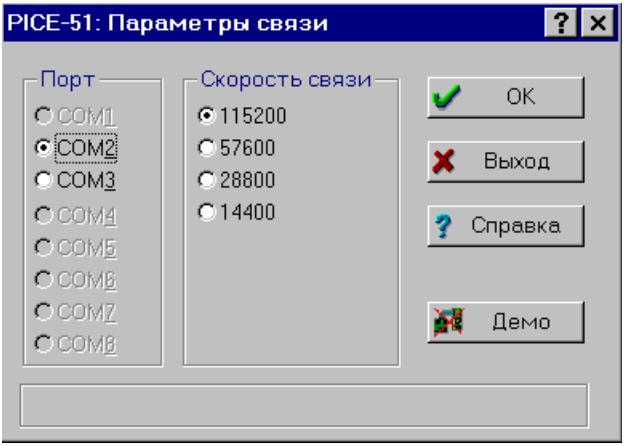


Рис. 1

Далее выбирается тип микроконтроллера: в списке «POD» появившегося окна (рис. 2) следует выбрать POD-51-31, в списке «Поддерживаемые микроконтроллеры» – Atmel 89C51, после чего щелкнуть мы-

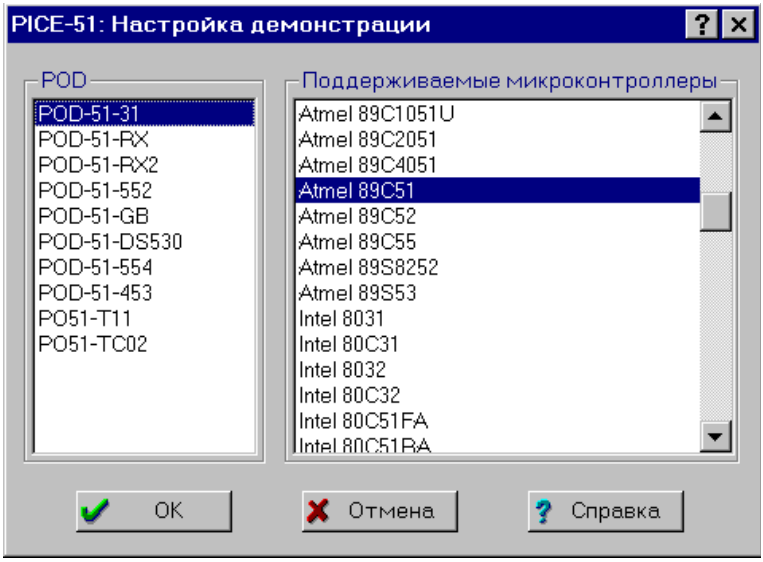


Рис. 2

шью по кнопке «OK». После этого происходит загрузка среды программы-отладчика PICE-51 (если в процессе загрузки будут появляться сообщения об ошибках или предупреждения, то во всех случаях ответ один – нажатие кнопки «Закрыть»).

Общий вид главного окна программы-отладчика приведен на рис. 3. Для выполнения лабораторных работ необходимо настроить вид и параметры инструментальной среды путем последовательного выбора в меню «Просмотр» следующих пунктов, каждый из которых активирует новое окно отображения информации при моделировании работы микроконтроллера:

1. Дизассемблер. В этом окне производится набор программы на языке ассемблера ASM-51 (система команд языка приведена в Прил. 2).

2. CPU registers. Окно отображает содержимое основных регистров микроконтроллера: PC, SP, DPTR, аккумулятора (ACC), регистра B (числа в ACC и B автоматически отображаются в десятичном и двоичном форматах).

3. Флаги PSW. Окно позволяет осуществлять детальный мониторинг битов состояния арифметико-логического устройства микроконтроллера.

4. Дамп Data. В отличие от предыдущих пунктов здесь вывод окна осуществляется в два этапа: сначала в меню выбирается пункт «Дамп

памяти...», после чего в окне настройки «Параметры дампа памяти» следует отметить пункт «DATA» и нажать кнопку «OK». В этом окне отображается содержимое всех ячеек внутреннего ОЗУ (IRAM) микроконтроллера, включая регистры специальных функций (всего 256 байтов, см. Прил. 1).

5. Дамп XData. Выполняются действия, аналогичные п. 4, только в окне «Параметры дампа памяти» выбирается пункт «XData». Это окно отображает содержимое внешнего ОЗУ (ERAM) объемом 64 килобайта (ячейки с адресами от 0000h до FFFFh), которое подключается к микроконтроллеру по стандартной схеме [2, 3].

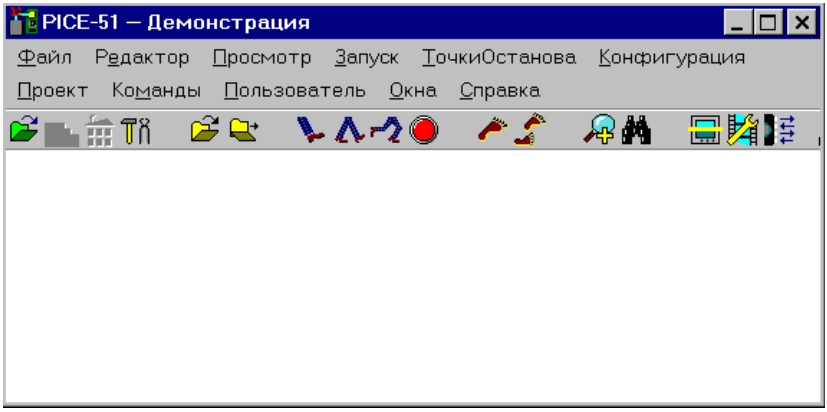


Рис. 3

Взаимное расположение этих пяти окон в главном окне программы настраивается при помощи стандартных средств Windows-приложений. Рекомендуемая конфигурация рабочего поля инструментальной среды разработки приведена на рис. 4. Помимо этих окон, при выполнении некоторых вариантов заданий могут потребоваться окна из подменю «Периферийные устройства», выбираемые из меню «Просмотр». *Если при загрузке программы активировались какие-нибудь «непонятные» окна, их следует закрыть, затем выбрать в меню «Проект» опцию «Новый», и только после этого можно приступать к настройке рабочей среды программы.*

Вновь созданную конфигурацию рабочего поля программы можно сохранить путем выбора одной из опций меню «Файл». Во-первых, можно создать специальный файл, содержащий параметры текущей настройки программы, с возможностью его загрузки в последующих сеансах

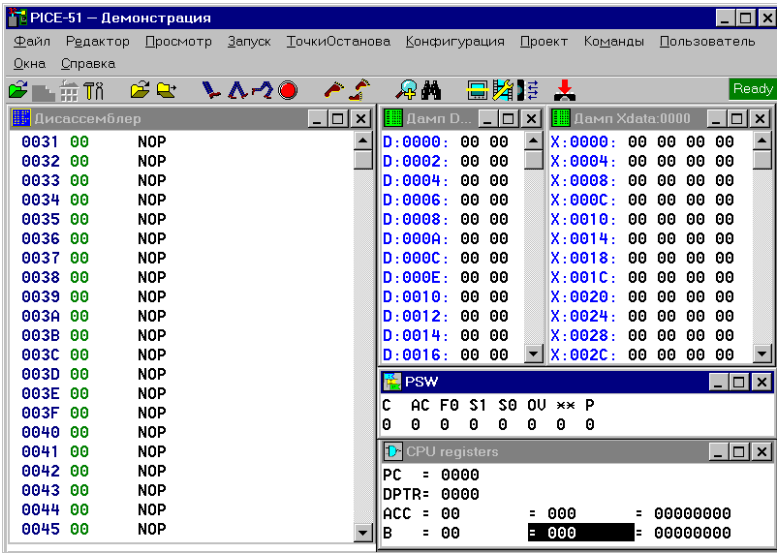


Рис. 4

работы (группа опций «Конфигурационные файлы...»). Во-вторых, можно активировать опцию «Автосохранение конфигурации при выходе», тогда текущая конфигурация рабочего поля автоматически загрузится в следующем сеансе работы. Для лабораторных занятий предпочтительнее первый способ.

В программе отладчика имеется развитая система контекстного меню, вызываемого нажатием правой кнопки мыши в любом месте активного окна. Помимо стандартного набора управляющих элементов Windows-приложений (как-то: возможности изменения вида и размера шрифта и т. д.), контекстное меню содержит ряд специфических опций.

Настройка форматов отображения информации. Содержимое ячеек памяти микроконтроллера может интерпретироваться по-разному, в зависимости от особенностей решаемой задачи. В большинстве случаев можно оставить настройку, принятую по умолчанию: каждая 8-разрядная ячейка памяти имеет свой порядковый номер (адрес), а ее содержимое отображается в шестнадцатеричном формате двумя символами. Однако при необходимости формат отображения чисел можно изменить на двоичный, десятичный, с плавающей точкой, как со знаком, так и без. В дополнение к этому имеется возможность группировки в одно число 2 последовательно расположенных байтов (формат данных типа

«слово») или даже 4 байтов (формат «двойное слово»). Группировка нескольких байтов в одно число бывает нужна, если в программе пользователя производятся сколько-нибудь сложные арифметические вычисления.

Вид контекстного меню окон дампа памяти показан на рис. 5.

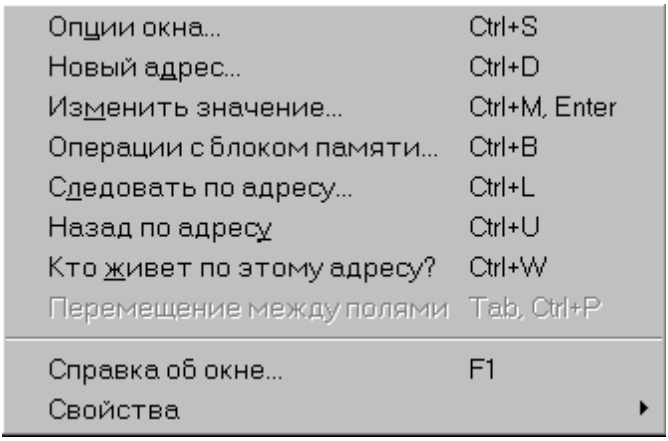


Рис. 5

Для настройки формата отображения чисел следует выбрать мышью пункт «Опции окна...», в результате чего появится окно «Параметры окна дампа памяти» (рис. 6).

Поле «Адресное пространство» позволяет выбирать область памяти для просмотра и ручной корректировки содержимого ее ячеек. Если, например, активировать кнопку «Code», то после нажатия кнопки «OK» окно дампа памяти отображает содержимое Flash-памяти микроконтроллера, содержащей коды команд программы пользователя. При этом можно даже осуществлять ввод и корректировку команд в окне дампа памяти непосредственно в машинных кодах в двоичном или шестнадцатеричном виде (см. столбцы «КОП» и «КОПh» в таблице Прил. 2), не обращаясь к окну «Дизассемблер». Очень удобно то, что в окне «Дизассемблер» при этом автоматически будут появляться ассемблерные инструкции (рис. 4), соответствующие вводимым машинным кодам. Для просмотра содержимого ячеек памяти данных предназначены соответственно кнопки: a) Idata (128 ячеек внутреннего ОЗУ данных, с адресами от 00h до 7Fh, т. е. без отображения содержимого регистров специальных фун-

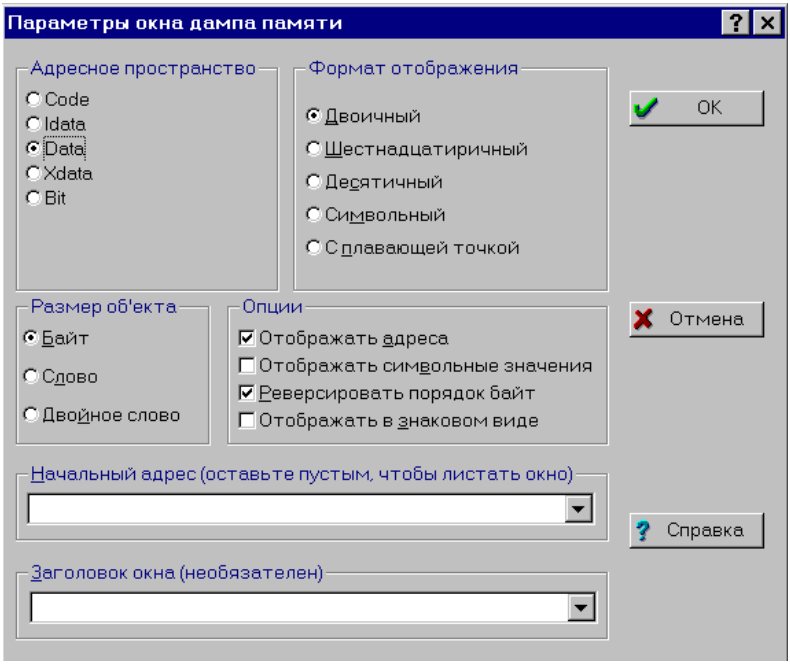


Рис. 6

кций); б) Data (все 256 ячеек внутреннего ОЗУ, с адресами от 00h до FFh); в) XData (ячейки внешнего ОЗУ объемом 64 кБайт).

Перед выполнением лабораторных работ рекомендуется самостоятельно проверить правильность своих представлений о различных вариантах форматов отображения чисел посредством изменения содержимого ячеек окна «Дамп памяти» при различных конфигурациях настроек окна «Параметры окна дампа памяти».

Техника написания программы пользователя. Команды вводятся в окне «Дизассемблер». Каждая строка окна содержит 3 последовательных элемента, которые в совокупности образуют 3 столбца (рис. 4): первый (всегда 4 символа) – адрес команды; второй (2, 4 или 6 символов, в зависимости от команды) – шестнадцатеричный код команды; третий – мнемоническая запись команды на языке ассемблера ASM-51 (см. Прил. 2).

Ввод возможен либо на языке ассемблера, либо непосредственно в шестнадцатеричном формате в поле кода команды (2-й столбец). Для

набора ассемблерной инструкции необходимо щелкнуть мышью на той инструкции в 3-м столбце, которая подлежит изменению (по умолчанию во всех ячейках записаны команды NOP), и начать ввод с клавиатуры. При этом автоматически появляется специальное окно «Ассемблировать инструкцию» (рис. 7), содержащее и предысторию ввода, из которой мышью можно при необходимости выбрать инструкцию s это ускоряет набор программы. После нажатия кнопки «OK», если мнемоника команды набрана правильно, окно закрывается и новая инструкция появляется в программе. Если же при наборе допущена какая-то ошибка (даже если пропущена запятая или поставлен лишний пробел между операндами!), то появится сообщение об ошибке (чаще всего с текстом «Неизвестная мнемоника инструкции») и ввод не состоится.

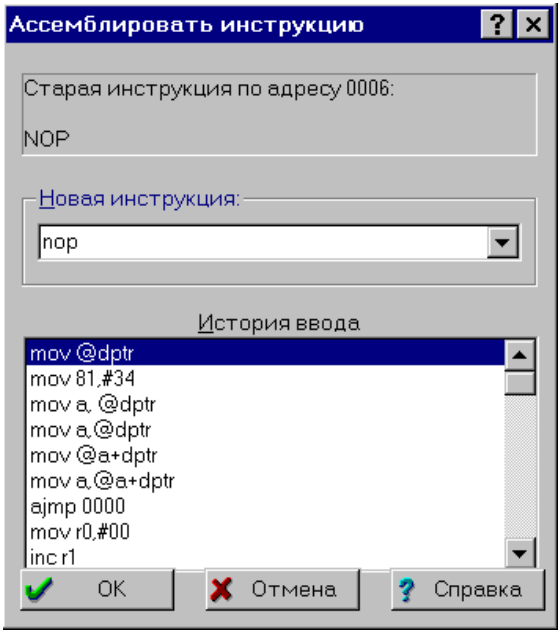


Рис. 7

К сожалению, демо-версия программы PICE-51 v. 3.01.71 по непонятным причинам неадекватно реагирует на ввод чисел в шестнадцатеричном формате записи, обычно выдавая сообщение об ошибке. Например, при попытке ввода инструкции MOV A,#FF появляется сообщение с текстом «Символьное имя не найдено» (здесь не исправляет ситуацию и вариант MOV A,#FFh). Бывает, что в подобной ситуации

после закрытия окна с сообщением об ошибке ввод все же производится, но константа воспринимается как введенная в десятичном формате (например, вместо введенной константы #80 в окне «Дизассемблер» появится число #50, так как десятичному числу 8010 соответствует 50h), а то и вовсе заменяется числом #00h. Есть 2 способа решения этой проблемы: либо следует вводить числа в двоичном формате, добавляя в конце без пробела символ «b» (например, так: MOV A,#11111111b); либо, после появления команды в окне «Дизассемблер», можно «перебивать» шестнадцатеричный код константы непосредственно в поле кода команды (второй столбец символов) на нужное значение. Этот способ удобнее, особенно при корректировке команд передачи управления, – можно быстро «подбирать» адрес перехода. Таким же образом можно вводить и сами шестнадцатеричные коды команд.

По мере накопления опыта работы с системой команд ассемблера метод «перебивки» шестнадцатеричных кодов непосредственно во втором столбце окна «Дизассемблер» зачастую даже значительно ускоряет ввод программы по сравнению с вводом более длинных ассемблерных инструкций.

Необходимо еще заметить, что в рассматриваемой программе (как во всех программах-отладчиках подобного типа) *невозможно осуществлять вставку команды в какое-либо место между двумя ассемблерными инструкциями – можно только заменять одну инструкцию на другую*. А если после тестирования написанной программы и выявления ошибок все же выяснится, что без такой вставки никак не обойтись, то обычно приходится переписывать и всю программу, расположенную ниже места вставки. Чтобы избежать этого утомительного процесса, при написании первых вариантов программ следует резервировать место под возможные вставки, «разбавляя» программу инструкциями типа NOP, которые никак не влияют на логику работы программы. Следует иметь в виду, что под вставку 3-байтовой команды понадобится три инструкции типа NOP, идущие подряд. И лишь после полной отладки логической структуры программы ее «ужимают»; при этом, конечно, приходится корректировать значения адресов переходов.

Еще одна опасность, подстерегающая при написании программы на ассемблере, состоит в том, что после замены длинной (2 или 3-байтовой) инструкции на инструкцию меньшей длины, остающийся «хвост» (константа или адрес) интерпретируется отладчиком уже как самостоятельная инструкция, ничего общего не имеющая ни со старой, ни с

новой введенной командой. Причем эти «фантомы» могут посчитать «своими» и коды нижерасположенных команд, – в результате программа превращается в полную бессмыслицу (и, увы, опция «Отмена» в меню программы не предусмотрена!). Чтобы этого избежать, следует *перед* заменой длинной команды на более короткую сначала «перебить» 2-й байт шестнадцатеричного кода заменяемой длинной команды (для двухбайтовой команды) или 2-й и 3-й байты (для трехбайтовой команды) на нули. Если это будет сделано, то после ввода новой «короткой» команды ниже нее дополнительно появится только одна или две «лишние» команды NOP, а это уже воспринимается нормально.

Сохранение результатов работы. Для сохранения программы пользователя на диске необходимо выбрать в меню «Файл» пункт «Записать файл из памяти процессора...» (рис. 8).

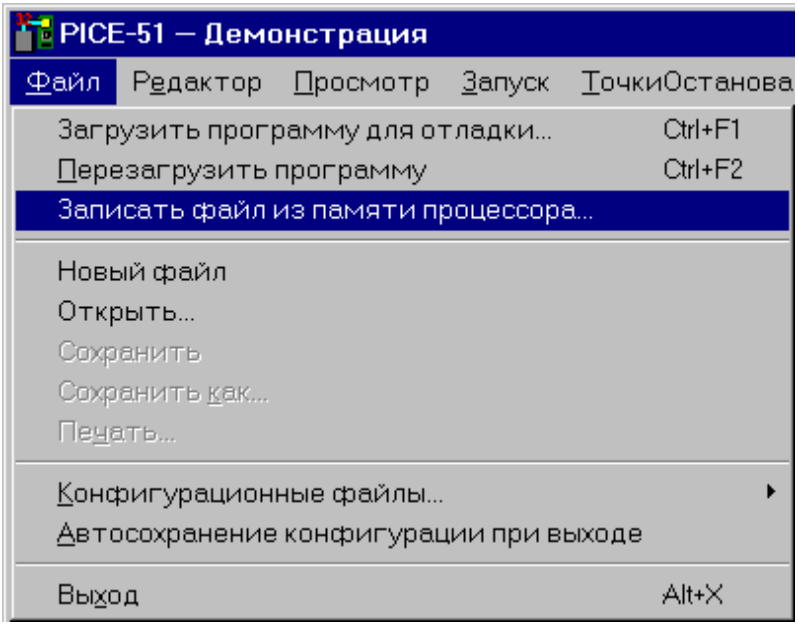


Рис. 8

В появившемся окне «Записать файл на диск» (рис. 9) выбирается область памяти микроконтроллера, содержимое которой требуется сохранить, и формат сохраняемого файла. Чтобы сохранить программу пользователя, следует в поле «Адресное пространство» активировать ключ «Code», а в поле «Формат сохраняемого файла» выбрать вариант

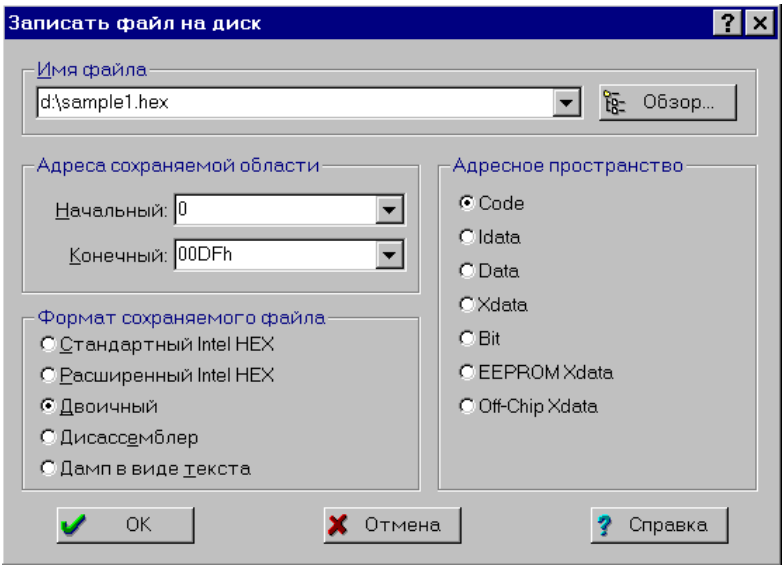


Рис. 9

«Двоичный» (это самый компактный формат сохранения файла с программой). Когда же нужен текст программы на ассемблере для распечатки отчета из текстового редактора, то выбирается формат «Дизассемблер» (но загрузить файл в данном формате для последующей отладки в программе PICE-51 будет невозможно).

Еще при сохранении программы обязательно нужно указать диапазон адресов сохраняемой области памяти: «Начальный» и «Конечный», соответствующие длине написанной программы в окне «Дизассемблер»: начальный адрес равен нулю, конечный – это адрес команды, которой заканчивается программа. *По умолчанию в PICE-51 v. 3.01.71 в поле «Конечный» записано нулевое значение, и если его не изменить, то написанная программа вообще сохранена не будет*!

Открытие файла с программой, сохраненной ранее на диске, производится посредством выбора в меню «Файл» пункта «Загрузить программу для отладки...» (см. рис. 10). В появляющемся затем окне (см. рис. 11) необходимо проследить за тем, чтобы *все настройки формата открываемого файла полностью соответствовали установкам, использованным ранее при его сохранении*. При необходимости можно сохранять и впоследствии загружать не только написанную программу, но и результаты ее работы, содержащиеся во внут-

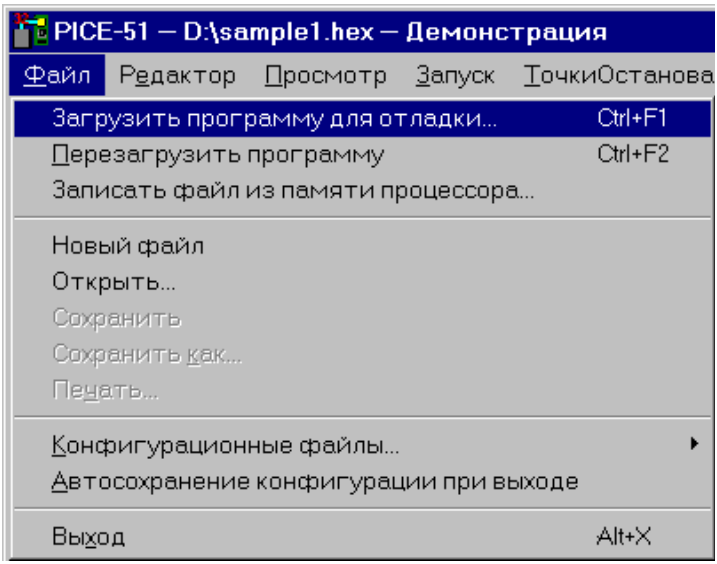

PICE-51 — D:\sample1.hex — Демонстрация
Файл
Редактор
Просмотр
Запуск
ТочкиОстанова
Загрузить программу для отладки... Ctrl+F1
Перезагрузить программу Ctrl+F2
Записать файл из памяти процессора...
Новый файл
Открыть...
Сохранить
Сохранить как...
Печать...
Конфигурационные файлы...
Автосохранение конфигурации при выходе
Выход Alt+X

Рис. 10

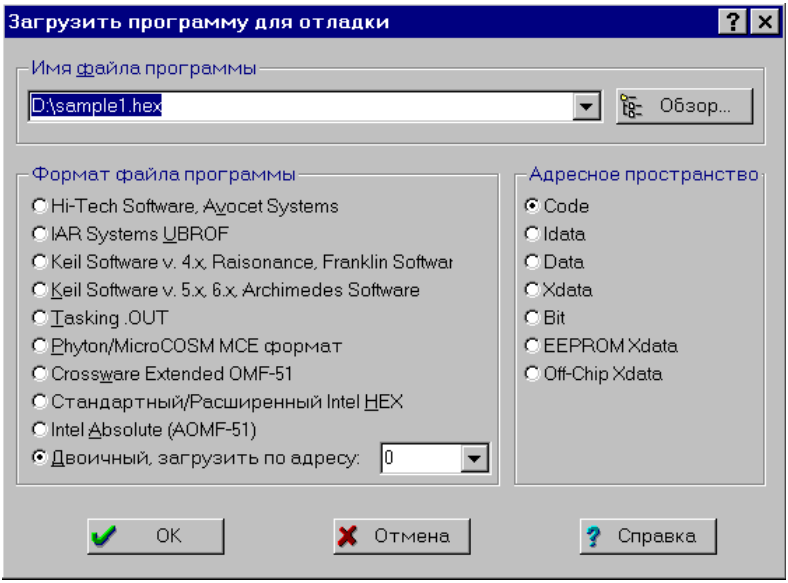

Загрузить программу для отладки
Имя файла программы
D:\sample1.hex
Обзор...
Формат файла программы
Hi-Tech Software, Avocet Systems
IAR Systems UBROF
Keil Software v. 4.x, Raisonance, Franklin Softwar
Keil Software v. 5.x, 6.x, Archimedes Software
Tasking .OUT
Phyton/MicroCOSM MCE формат
Crossware Extended OMF-51
Стандартный/Расширенный Intel HEX
Intel Absolute (AOMF-51)
Двоичный, загрузить по адресу: 0
Адресное пространство
Code
Idata
Data
Xdata
Bit
EEPROM Xdata
Off-Chip Xdata
OK
Отмена
Справка

Рис. 11

ренней и внешней памяти: достаточно в поле «Адресное пространство» выбирать соответствующие опции. Массив данных, сохраненных в двоичном формате, может быть легко прочитан в других приложениях, как файл данных типа «byte» или «shortint». Это может быть использовано, например, для графического представления временной диаграммы по последовательности отсчетов, сохраненных во внешнем ОЗУ.

Отладка программы. Для трассировки программы используются опции меню «Запуск» (рис. 12) или соответствующие кнопки панели инструментов, а также контекстное меню (рис. 13), вызываемое правой кнопкой мыши для любой из строк окна «Дизассемблер».

Для отладки написанной программы предусмотрено 2 режима: пошаговый режим и прогон всей программы. В пошаговом режиме в ответ на действие пользователя «Выполнить шаг» (нажатие клавиши **F7** или кнопки  на панели инструментов) выполняется одна мнемокоманда. Пошаговый режим удобен при изучении системы команд и при отладке программы: имеется возможность наблюдать влияние команды на содержимое различных ячеек памяти микроконтроллера. Если же программа отлажена и интерес представляет только конечный результат ее работы, то осуществляется прогон всей программы путем выбора подменю в меню «Запуск» опции «Запуск/Останов» (возможно нажатие клавиши **F9** или кнопки  на панели инструментов).

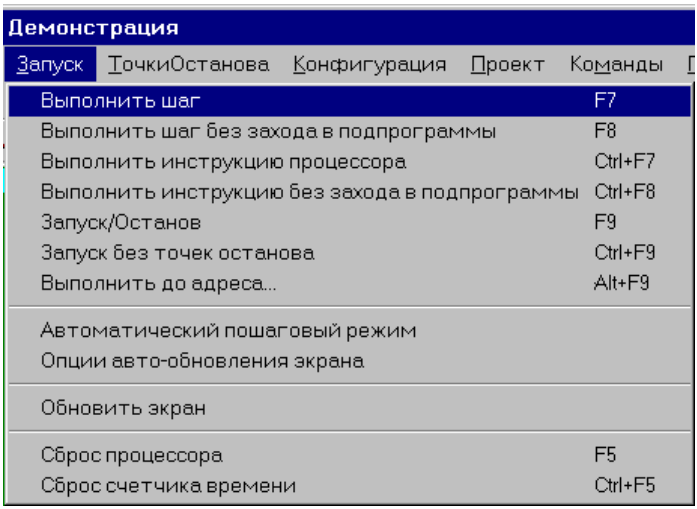


Рис. 12

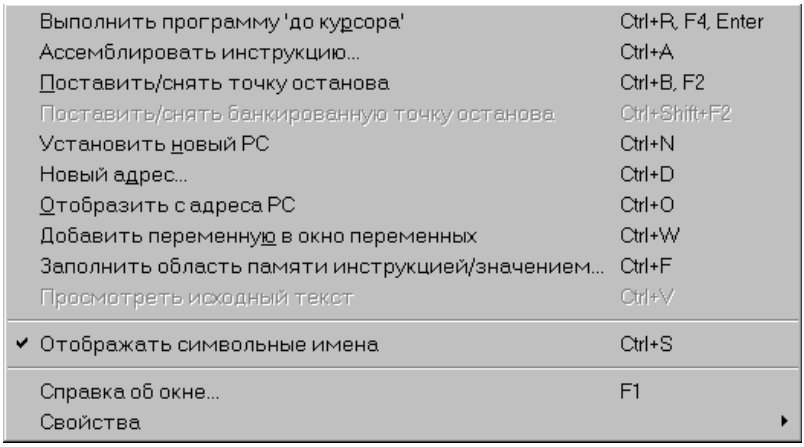


Рис. 13

При прогоне следует помнить, что режимы аппаратного останова микроконтроллера (режимы IDLE и Power Down) в демо-версии программы PICE-51 не поддерживаются, поэтому в конце программы или ее отлаживаемого фрагмента следует установить так называемую точку останова (breakpoint). Это осуществляется при помощи контекстного меню (рис. 13): на требуемой строке программы следует нажать на правую кнопку мыши и выбрать пункт «Поставить/снять точку останова». При этом строка, на которой должно остановиться выполнение программы, выделится красным цветом. Теми же действиями поставленная ранее точка останова снимается.

Если в процессе отладки программы необходим пошаговый режим только для какого-либо фрагмента программы, находящегося «далеко» от ее начала, то при помощи опции «Установить новый PC» контекстного меню (рис. 13) можно начинать выполнение программы с нужного места. Если к этому моменту содержимое некоторых ячеек памяти микроконтроллера уже должно иметь какое-либо конкретное значение, необходимые изменения легко производятся в соответствующих окнах посредством щелчка левой кнопки мыши в нужном месте и ввода необходимого кода с клавиатуры.

Чтобы после прогона программы восстановить исходное состояние микроконтроллера (которое бывает после аппаратного сброса), следует нажать клавишу **F5** или красную кнопку  на панели инструментов.

При этом установленные точки останова сохраняются. Следует иметь в виду, что *сигнал сброса не изменяет содержимого внутреннего ОЗУ*. Поэтому если по логике построения программы важно, чтобы начальное состояние каких-либо ячеек было, например, нулевым, то необходимо предусмотреть в программе (обычно где-то в самом ее начале) соответствующие ассемблерные инструкции.

2. Система команд микроконтроллеров семейства MCS-51

Сводная информация о системе команд микроконтроллеров семейства MCS-51 [3] приведена в Прил. 2. В ассемблерных инструкциях используются следующие мнемонические обозначения и сокращения:

Rn (n = 0...7) – n-й регистр общего назначения (РОН) из активного регистрового банка (по умолчанию после сброса выбран нулевой банк);

A – аккумулятор;

DPTR – регистр-указатель данных;

PC – указатель счетчика команд;

SP – указатель стека;

@Ri – адрес ячейки, используемый в командах с косвенной адресацией, задается содержимым РОН (либо R0, либо R1) из активного банка регистров;

ad – адрес прямоадресуемого байта в IRAM;

ads – адрес прямоадресуемого байта-источника в IRAM;

add – адрес прямоадресуемого байта-приемника в IRAM;

ad11 – 11-битный абсолютный адрес;

ad16 – 16-битный абсолютный адрес;

rel – 8-битный относительный адрес;

#d – непосредственный 8-битный операнд (константа типа «байт»);

#d16 – непосредственный 16-битный операнд (константа типа «слово»);

bit – 8-разрядный адрес прямоадресуемого бита;

rrr – эти 3 бита в двоичном коде команды соответствуют номеру РОН;

РПД – регистровая память данных (внутреннее ОЗУ, или IRAM);

ВПД – внешняя память данных (внешнее ОЗУ, или ERAM);

ПП – память программы (ПЗУ, как IROM, так и EROM, в зависимости от уровня напряжения на входе EA микроконтроллера и содержимого PC).

Команды языка ASM-51 разделены на несколько функциональных подгрупп, описание которых приводится далее.

Команды пересылок. Обмен данными является важнейшим элементом любой программы. В эту группу объединены команды, позволяющие переносить данные из одного места памяти микроконтроллера в другое. Команды этой группы отличаются друг от друга лишь способом определения источника и приемника данных. Помимо мнемонических обозначений регистров, как источник, так и приемник данных может специфицироваться непосредственным адресом, откуда записываются данные, а также указанием на адрес, где находятся данные (случай косвенной адресации). Необходимо помнить, что при пересылке данные источника *копируются* в приемник, а не меняются местами – т. е. после выполнения какой-либо команды пересылок в приемнике и в источнике будут находиться одинаковые числа.

Примеры (во всех примерах в скобках указан номер команды в таблице Прил. 2):

(10) **MOV 02,0A** – данные из ячейки внутреннего ОЗУ с адресом 0Ah копируются в ячейку внутреннего ОЗУ с адресом 02h.

(13) **MOV @R0,A** – результатом выполнения данной команды будет копирование данных из аккумулятора в ячейку внутреннего ОЗУ с адресом, равным содержимому регистра R0.

Помимо копирования данных, имеется и возможность взаимного обмена данными: т. е. данные из источника переносятся в приемник, и одновременно – из приемника в источник (команды 23–26). Некоторые функциональные ограничения этого способа обмена данными связаны с тем, что в качестве одного из операндов всегда выступает аккумулятор.

Несколько особняком в ряду команд пересылок стоят операции со стеком (команды 27 и 28). В качестве операнда используется один параметр: команда **PUSH ads** осуществляет пересылку данных из ячейки внутреннего ОЗУ с адресом ads в стек. При этом приемником данных выступает ячейка ОЗУ с адресом, указанным в SP. Необходимо отметить, что при выполнении команды PUSH указатель стека сначала увеличивается на единицу, а только потом в ячейку с полученным адресом записываются данные. Для извлечения данных из стека используется команда **POP add**, где add – это адрес ячейки-приемника данных. В случае извлечения данных из стека сначала выбираются данные, а потом указатель стека SP уменьшается на единицу. Ячейка стека способна играть роль некоторого «стакана» при необходимости взаимного обмена содержимым между ячейками ОЗУ, минуя аккумулятор. Нелишне заметить, что, согласно принятому при работе со стеком способу адре-

сации операндов, *число после его извлечения в стеке больше не хранится.*

Команды логических операций. Использование побитовых логических операций И, ИЛИ, НЕ и исключающего ИЛИ позволяет видоизменять отдельные биты операнда, не затрагивая остальных. Результат выполнения логической операции записывается на место первого операнда.

Пример

(38) **ORL A,#03** – после выполнения этой команды 2 младших бита аккумулятора примут единичные значения, остальные не изменятся. Если, например, в аккумуляторе хранилось число 45h, то после данной команды аккумулятор будет содержать число 47h.

Операторы сдвига (команды 49–52) играют важнейшую роль в программировании микроконтроллеров: с их помощью выполняются функции регистров сдвига. Команды сдвига перемещают биты содержимого аккумулятора влево (рис. 14) или вправо (для этого случая направление стрелок на рис. 14 должно быть противоположным).

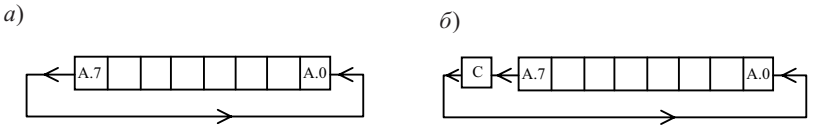


Рис. 14. Схемы действия команд циклического сдвига влево:
***а* – обычный циклический сдвиг; *б* – циклический сдвиг через перенос**

Команды **RL A** и **RLC A** отличаются тем, что при использовании команды **RLC A** в процесс вовлекается бит переноса C, расположенный в регистре PSW, как (рис. 14, *б*). В частном случае, операции сдвига используются для умножения на 2 (один сдвиг влево) или деления на 2 (один сдвиг вправо), которые выполняются в 4 раза быстрее универсальных команд **MUL** и **DIV** (см. группу команд арифметических операций).

Команды передачи управления необходимы для организации в программе ветвлений и циклов. В языке ASM-51 в эту группу объединены:

1) команды безусловного перехода (номера команд: 54, 55, 56);
2) команда перехода по базовому индексу со смещением (номер 57);
3) команды условных переходов (58–70);
4) команды работы с подпрограммами (71–74).

Команды безусловного перехода позволяют продолжить выполнение программы с нового адреса, однозначно определенного в команде. Три команды этой подгруппы – SJMP, AJMP, LJMP – отличаются только максимальной длиной перехода: например, инструкция **SJMP** позволяет осуществлять «короткие» переходы (не более 128 адресов вверх и не более 127 вниз относительно места данной команды в программе), а команда **AJMP** – уже в пределах объема программы в 2кБайта. Команда **LJMP** позволяет осуществлять «длинные» переходы, *но работает только с внешней памятью программ! Во избежание ошибок при расчете адреса перехода всегда нужно помнить, что количество адресов, занимаемое каким-либо фрагментом программы, в общем случае не совпадает с количеством ассемблерных инструкций в этом фрагменте*, поскольку есть 2 и 3-байтовые команды.

Следует учитывать, что при вводе команды *на языке ассемблера адрес перехода указывается в явном виде*: т. е. какой адрес записан в мнемокоде, на тот и будет осуществляться переход. Но при коррекции шестнадцатеричного кода операции (это суть второй байт команды) следует вводить число, равное количеству адресов, *на которое надо перейти вперед или назад относительно данного места*. Для перехода «назад» второй байт команды представляется в дополнительном коде. Эту особенность поясняет следующий пример.

Пример

Пусть команда **SJMP 00AB** находится по адресу **00A1**. Непосредственно из ассемблерной инструкции видно, что команда выполняет переход по адресу 00AB. А два байта кода данной операции в шестнадцатеричной записи суть **80 08**. Здесь первый байт, 80, – это код команды SJMP, а 08 показывает, что переход осуществится на 8 ячеек вперед (в MCS-51 отсчет всегда начинается с адреса команды, *непосредственно следующей* за командой перехода). Поскольку SJMP 00ABh является двухбайтовой командой, то отсчет и надо начинать с адреса 00A1 + 0002 = = 00A3. Тогда, с учетом равенства **00A3 + 08 = 00AB**, взаимосвязь адреса в ассемблерной инструкции и кода операции становится очевидной. Если же было бы нужно осуществить переход на 8 адресов назад, то число минус 8 в коде операции следовало бы представить в дополнительном коде: –08h=F8h, т. е. в данном случае ассемблерной инструкции SJMP 009B будет соответствовать машинный код **80 F8**.

Команды условных переходов выполняют переход только при справедливости некоторого условия. Если оно не выполнено, то переход не состоится и будет исполняться команда, непосредственно следующая за командой условного перехода. В MCS-51 предусмотрены следующие условия перехода: состояние бита переноса или любого доступного для прямой адресации бита; равенство или неравенство содержимого аккумулятора нулю; результат сравнения регистра с константой (в частном случае, с нулем) или значения аккумулятора с содержимым ячейки ОЗУ. Следует помнить, что *при сравнении чисел во всех без исключения командах ASM-51 предполагается, что эти числа положительны* (т. е. десятичный эквивалент числа F8h равен плюс 248, а не минус 8).

В этой группе имеются замечательные инструкции, совершающие сразу несколько действий в течение своего выполнения: команды **CJNE** и **DJNZ**. Команда **CJNE N1,N2,rel** сравнивает 2 числа и выполняет переход, если эти два числа не равны, попутно изменяя состояние бита переноса. В данном случае на месте параметра **N1** может оказаться аккумулятор, РОН или адрес ячейки ОЗУ, указанный в регистре Ri (косвенная адресация), а в качестве **N2** может фигурировать либо прямой адрес ячейки ОЗУ, либо константа. При любых сочетаниях типов параметров команда CJNE изменяет бит переноса C следующим образом: если *содержимое* элемента N1 меньше содержимого N2, то C=1, а если наоборот (**N1**)>(**N2**), то C=0. Команда DJNZ сначала уменьшает число, затем сравнивает его с нулем и совершает переход, если получившееся число не равно нулю. Имеются варианты этой команды с прямой и с регистровой адресацией: **DJNZ ad,rel** и **DJNZ Rn,rel**, соответственно. Эти команды исключительно удобны для построения циклов с заданным количеством повторений, с уменьшением переменной цикла.

Важно подчеркнуть, что переходы во всех командах условных переходов относятся к классу «коротких», что иногда вызывает неудобства при необходимости «перепрыгнуть» фрагмент программы длиной больше 127 адресов. Для решения этой задачи приходится осуществлять переход в два этапа: сначала короткий переход на команду «длинного» безусловного перехода, а уже она, в свою очередь, загружает указатель команд необходимым числом и передает управление в нужное место. Другой способ решения проблемы состоит в использовании команд вызова подпрограмм.

Команды вызова подпрограмм. В отличие от команд переходов, при переходе на новое место в программе автоматически сохраняют адреса

возврата (адрес команды, следующей за командой вызова подпрограмм) в стеке, оставляя возможность продолжения выполнения основной программы с этого места. Команды **ACALL** и **LCALL** работают одинаковым образом, их отличие заключается только в максимальной длине перехода. Для возврата в точку вызова служит команда **RET** или **RETI**, при выполнении любой из этих команд в счетчик команд PC из стека загружается сохраненный ранее при вызове адрес возврата.

При написании подпрограмм обработки прерываний часто используются команды работы со стеком: если в подпрограмме в принципе могут измениться ключевые данные, с которыми в момент прерывания, возможно, оперировала основная программа (чаще всего это аккумулятор и регистр PSW), то эти данные обычно сохраняются в стеке в самом начале подпрограммы, а потом извлекаются из стека обратно в самом ее конце. Очевидно, что при использовании команд работы со стеком внутри тела подпрограммы следует быть внимательным, – нужно строить программу так, чтобы к моменту выполнения команд RET или RETI указатель стека при любых коллизиях, которые могут возникать при выполнении подпрограммы, указывал на адрес возврата. Общее правило для корректного возврата из подпрограммы таково: *количество команд PUSH и POP, выполненных внутри подпрограммы до команды возврата, должно быть строго одинаковым*.

Команды операций с битами. Эти команды предназначены для проведения операций над отдельными битами регистров микроконтроллера: сброс или установку бита, логические операции с битами, пересылка бита в перенос и обратно. Операции над отдельными битами возможны не для всех ячеек внутреннего ОЗУ (см. Прил. 1).

Команды операций с битами наиболее часто используются для управления флагами регистра состояния процессора (PSW). Управление битом переноса C играет важную роль при реализации на базе микроконтроллера эквивалентов цифровых схем, построенных по структуре вида «регистр сдвига с обратной связью» (таких как, например, генераторы псевдослучайных чисел, распределители).

Пример

(76) **CLR C** обнуляет бит переноса.

Команды арифметических операций. В микроконтроллерах семейства MCS-51 предусмотрен полный набор элементарных арифметических действий над данными: сложение (как с учетом бита переноса, так

и без), вычитание с учетом значения бита переноса (заема), умножение, деление, а также и операция десятичной коррекции. В арифметических операциях одним из операндов является аккумулятор. Результаты сложения и вычитания всегда помещаются в аккумулятор. Имеются также операции инкремента (**INC**) и декремента (**DEC**), которые увеличивают и уменьшают число на единицу, соответственно.

Команды умножения (**MUL AB**) и деления с (**DIV AB**) используют аккумулятор и регистр-расширитель аккумулятора B (запятая между A и B в мнемокодах не ставится!). При умножении восьмиразрядных чисел результат будет 16-разрядным числом, для его размещения используется аккумулятор и регистр B: старший байт результата записывается в регистр B, младший – в A. При делении делимое записывается в аккумулятор, делитель – в B. Результат деления помещается в аккумулятор, остаток от деления – в B.

3. Типовые решения, применяемые при программировании микроконтроллеров семейства MCS-51*

Методика составления программ. Процесс написания программы на языке ассемблера можно существенно облегчить, если выполнить следующие предварительные этапы:

1) определить источники входных данных и место для размещения выходных данных (результата работы программы);

2) составить блок-схему алгоритма, используя любые символьные имена для переменных, используемых для решения задачи;

3) назначить ячейки памяти микроконтроллера для всех символьных имен переменных, обращая внимание на разрядность операндов (для размещения числа с разрядностью больше 8 следует отводить несколько ячеек памяти микроконтроллера).

Причем основные показатели качества программы – длина кода и ее быстродействие – в значительной степени определяются тщательностью выполнения последнего из перечисленных этапов.

Вообще, в зависимости от требований конкретного приложения и сложности решаемой задачи, иногда требуется максимально уменьшить размер кода программы, иногда – достичь предельного быстродействия. На первый взгляд, сокращение длины кода программы должно приво-

---

* В написании данного раздела принимали участие студенты гр. 4013k А. О. Ковалев и А. В. Старостин.

дить к уменьшению времени выполнения программы, но обычно это не так, поскольку для сокращения объема программы приходится прибегать к циклам, организации подпрограмм и другим методам, которые требуют больше времени на выполнение. С другой стороны, если отказаться от циклов и переходов к подпрограммам, то чтение кода программ будет затруднено. Поэтому часто приходиться искать компромиссы, такие как:

– сокращение потребных объемов памяти программы и памяти данных (для хранения промежуточных результатов) за счет снижения быстродействия;

– повышение быстродействия за счет ухудшения возможностей доработок и доступности текста программы для чтения;

– совместная оптимизация – достижение наибольшего быстродействия при наименьшей длине кода программы – плохо поддается формализации и существенно увеличивает время разработки программы.

Оптимизация программы по быстродействию. Первое, на что следует обратить внимание – это алгоритм действий, который используется для выполнения поставленной задачи. Если он не удовлетворяет требованиям по быстродействию, то стоит подыскать другой алгоритм, который выполнял бы те же задачи быстрее. Второй момент – это повторные вычисления. Если в программе вычислено какое-либо число и это значение потребуется в дальнейшем в другом месте программы, то его следует сохранить в какой-либо ячейке ОЗУ и извлекать по мере необходимости, а не вычислять заново. В третьих, следует по возможности учитывать особенности решаемой задачи с тем, чтобы вместо универсальных команд использовать команды, подходящие для конкретного случая. Например, умножение на 4 может быть реализовано двумя командами сдвига влево **RLC** вместо использования универсальной команды умножения **MUL**, которая выполняется вдвое дольше.

И, наконец, следует заметить, что если в программе используются условные переходы, всегда желательно преобразовывать логическую структуру программы таким образом, чтобы условие перехода в среднем оказывалось истинным реже, чем условие для его отсутствия. При оптимизации иногда стоит вообще отказаться от использования циклов: в случае, если расчетное время выполнения тела цикла будет меньше времени выполнения команд передачи управления, лучше последовательно разместить повторяющиеся команды. В ряде случаев целесообразно отказаться и от организации подпрограмм, что-

бы обладать более широкими возможностями использования стека для хранения данных.

Оптимизация программы по размеру кода. Если возможность практического использования программы ограничена не временем ее выполнения, а размером объема памяти, которое она занимает, то необходимо задуматься о мерах, в основном противоположным тем, которые используются при оптимизации по быстродействию. Следует выяснить, что обусловливает большой размер – собственно код или используемые исходные константы. Если проблема заключается в больших объемах констант, то стоит задуматься о возможной избыточности этого массива и попытаться устранить ее путем отыскания методов вычисления какой-либо группы констант через остальные, которые и составят базовый набор. Нельзя забывать и о различных вариантах адресации одной и той же ячейки памяти: например, команда **MOV R0, #ABh** займет меньше места, чем эквивалентная ей команда **MOV 00h,#ABh**.

Кроме того, нужно внимательно просмотреть код написанной программы на предмет повторяющихся фрагментов, и если таковые будут найдены, следует организовать их в подпрограммы. В идеале это приводит к программе модульного типа, состоящей из отдельных подпрограмм и вызовов к ним. И если удастся построить программу так, чтобы в каждый момент времени работал только один модуль, то для хранения промежуточных результатов вычислений можно использовать общую память данных.

Примеры* программных решений типовых задач

Организация циклов. Очень распространенной является ситуация, когда в программе необходимо выполнять последовательность однотипных действий определенное количество раз. Тогда в программе организуется цикл: для этого следует выбрать одну из переменных, которая будет определять количество повторений тела цикла, и задаться условием выхода из цикла. В ряде практических приложений требуется, чтобы некоторый цикл выполнялся бы в течение всего времени, пока микроконтроллер включен (например, при использовании микроконтроллера для генерирования сигналов в портах вводам–вывода), – это так называемый «бесконечный» цикл. Конечные же циклы сверяются с каким-

*Во всех примерах точка останова ставится *на команду*, следующую за последней командой программы.

либо особым условием: при его выполнении осуществляется выход из цикла. Условия выхода из цикла могут быть двух видов: количество повторений тела цикла определяется либо только начальным значением переменной цикла, либо каким-либо условием (или совокупностью условий), когда заранее определить количество повторений очень трудно. Возможна также и комбинация этих двух видов.

Пример 1

Необходимо сформировать бесконечную последовательность типа «бегущий огонь» (предполагается, что к выводам порта микроконтроллера подсоединены светоизлучающие индикаторы, зажигающиеся при наличии на выходе высокого логического уровня):

0000 0001
0000 0010
0000 0100
0000 1000
0001 0000
0010 0000
0100 0000
1000 0000
0000 0001
и т. д.

Достаточно очевидно, что в данном случае одним из ключевых элементов программы будет команда сдвига влево, а поскольку цикл бесконечный, не требуется организовывать проверку каких-либо условий. Тогда программа может иметь следующий вид:

Адрес	Команда	Комментарий
0000	**MOV A,#01**	Загружаются исходные данные
0002	**MOV 80, A**	Пересылка значения аккумулятора в порт P0
0004	**RL A**	Сдвиг содержимого аккумулятора без переноса
0005	**SJMP 0002**	Возврат к команде пересылки MOV 80, A

Пример 2

Можно потребовать, чтобы формируемая последовательность содержала бы, например, только 5 периодов (т. е. чтобы единица «пробежала» от начала до конца только 5 раз, а затем цикл завершился бы), то необходимо осуществить «обрамление» предыдущей программы элементами цикла. Например, это можно сделать так:

Адрес	Команда	Комментарий
0000	**MOV A,#01**	Загружаются исходные данные
0002	**MOV R0, #28**	Загрузка переменной цикла: количество повторений тела цикла заранее известно: 5×8 раз=40 (т. е. 28h)
0004	**MOV 80, A**	Пересылка значения аккумулятора в порт P0
0006	**RL A**	Сдвиг содержимого аккумулятора без переноса
0007	**DJNZ R0,0004**	Сначала R0:=R0–1, далее если R0≠0, то осуществляется переход по адресу 0004 к команде пересылки MOV 80, A; если же R0=0, то выполнение программы продолжится с команды, следующей за DJNZ (т. е. с записанной по адресу 0009)

Если проанализировать состав команд передачи управления стандарта MCS-51, то становится ясно, что рациональнее создавать циклы, в которых счетчик повторений уменьшается, а не увеличивается (так как команда DJNZ является многофункциональной, а аналогичной команды с инкрементом значения параметра в MCS-51, увы, не предусмотрено). Однако создать цикл со счетчиком на увеличение вполне возможно, и следующий пример показывает, как таким образом можно решить предыдущую задачу.

Пример 3

Адрес	Команда	Комментарий
0000	**MOV A,#01**	Загружаются исходные данные
0002	**MOV R0, #00**	Обнуление счетчика цикла
0004	**MOV 80, A**	Пересылка значения аккумулятора в порт P0
0006	**RL A**	Сдвиг содержимого аккумулятора без переноса.
0007	**INC R0**	Увеличение счетчика циклов на 1: R0:=R0+1
0008	**CJNE R0,#28,0004**	Если R0≠28h (т. е. R0≠40) то осуществляется переход по адресу 0004 к команде пересылки MOV 80, A; если же R0=28h, то выполнение программы продолжится с команды, следующей за CJNE (т. е. с записанной по адресу 000B)

Нелишне заметить, что код данной программы на 2 байта длиннее предыдущего варианта с командой DJNZ, и тело цикла выполняется дольше.

Вообще же, при организации циклических структур всегда следует придерживаться двух общих правил:

– не делать в цикле того, что можно сделать за его пределами (т. е. нужно выносить из тела цикла операции, не связанные с переменной цикла);

– по возможности не использовать в теле цикла команды передачи управления.

Эффективным способом оптимизации является метод сращивания циклов: если вдруг в программе найдутся два цикла, которые выполняются одинаковое количество раз, следует попытаться совместить их в один. Таким образом убирается одна команда перехода.

Использование подпрограмм. Использование подпрограмм позволяет структурировать программу при необходимости выполнения какого-то алгоритма с различными исходными данными. Подпрограммы можно размещать в любом месте памяти программ микроконтроллера.

Пример 4

Адрес	Команда	Комментарий
0000	**MOV A,#01**	Загружаются исходные данные
0002	**MOV R0, P2**	Загрузка переменной цикла R0
0004	**ACALL 07F0**	Вызов подпрограммы, формирующей «бегущий огонь», размещенной по адресу 07F0
0006	**SJMP 0002**	Возврат к команде загрузки переменной цикла
0008	**…**	Здесь могут быть любые команды, не связанные с выполнением рассматриваемой задачи, но *важно, чтобы до адреса 07EF была бы команда перехода, предотвращающая возможный вход в начало подпрограммы без команды CALL*
0009	**…**	
…	**…**	
07EE	**…**	
07EF	**…**	
07F0	**MOV 80, A**	*Здесь записана первая команда подпрограммы* : пересылка значения аккумулятора в порт P0
07F2	**RL A**	Сдвиг содержимого аккумулятора без переноса
07F3	**DJNZ R0,07F0**	R0:=R0–1, если R0≠0, то переход к команде MOV 80, A; если же R0=0, то выполнение подпрограммы завершится
0FF5	**RET**	Автоматический возврат из подпрограммы в точк у вызова по адресу 0006: к команде SJMP

Пусть требуется формировать конечную последовательность типа "бегущий огонь" в порту P0, так чтобы количество сдвигов логической единицы задавалось бы кодом на входах порта P2 (полных периодов последовательности, очевидно, будет в 8 раз меньше, чем заданное значение кода).

Приведенная далее программа использует передачу в подпрограмму одного параметра (переменной цикла), это осуществляется через глобальную переменную, размещенную в регистре R0.

Следует подчеркнуть, что *при использовании подпрограмм необходимо принимать специальные меры к тому, чтобы доступ к ним мог осуществиться только через команды вызова подпрограмм типа CALL и никак иначе.*

Использование косвенной адресации операндов. Косвенная адресация удобна при последовательном переборе элементов массива данных. В зависимости от местоположения массива используются различные команды для доступа к его элементам. Для примера рассмотрим задачу циклической перестановки элементов массива данных: т.е. выполнения последовательности операций вида $X[i] \leftarrow X[i+1]$ для $i=1,2,...N$ (с одновременной пересылкой исходного содержимого $X[1]$ на место $X[N]$) над последовательностью операндов, размещенных во внутреннем ОЗУ микроконтроллера. Данный пример также иллюстрирует применение ***команд работы со стеком***.

Пример 5

Решение поставленной задачи для $N=32$ (т. е. 20h в шестнадцатеричной записи) и при размещении исходного массива данных во внутреннем ОЗУ по адресам с 30h по 4Fh может иметь следующий вид:

Приведенный алгоритм циклической перестановки данных может быть использован как составная часть алгоритмов цифровой фильтрации (см. задания к лабораторной работе № 7), применяемых при реализации корректирующих звеньев цифровых систем автоматического управления [3, 8].

Организация арифметических вычислений. Одной из типовых задач, возлагаемых на микроконтроллеры в цифровых системах автоматического управления, является коррекция статических характеристик динамических звеньев [3]. Например, статическая характеристика углового дискриминатора вида $U(\vartheta) = \sin(\vartheta)$ может быть линеаризована посредством перевода напряжения U в цифровой код, вычисления функ-

Адрес	Команда	Комментарий
0000	**PUSH 30**	Сохранение в стеке элемента X[1]
0002	**MOV R0, #30**	Загрузка адреса, по которому размещается X[1]
0004	**MOV R1,#31**	Загрузка адреса, по которому размещается X[2]
0008	**MOV A,@R1**	Начало тела цикла, выполняющего $X[i] \leftarrow X[i+1]$ для $i=1,2,...N$. Здесь элемент $X[i+1]$ пересылается в аккумулятор
0009	**MOV @P0,A**	Запись содержимого аккумулятора по адресу в памяти, соответствующему расположению элемента $X[i]$, т. е. совместно с предыдущей командой произведена пересылка $X[i] \leftarrow X[i+1]$
000A	**INC R0**	Увеличение на единицу индексов элементов, участвующих в элементарной пересылке вида $X[i] \leftarrow X[i+1]$
000B	**INC R1**	
000C	**CJNE R0,#4F,0008**	Проверка на длину массива: закончен ли перебор элементов, и, если нет, то производится возврат к началу цикла по адресу 0008
000F	**POP 4F**	Пересылка ранее сохраненного в стеке элемента $X[1]$ на место в ОЗУ, отведенное под элемент $X[N]$

ции $Y(U) = \arcsin(U)$, и обратного преобразования в напряжение. Если микроконтроллер используется в качестве генератора гармонического сигнала [11], то необходимо вычислять отсчеты функций $\sin(kt)$ и $\cos(kt)$. Особенность данного круга задач заключается в том, что требования к точности вычислений невысоки: для большинства приложений допустима относительная ошибка порядка нескольких процентов, что достигается даже при 8-разрядном представлении. Кроме того, обычно входные и выходные величины предполагаются дробными числами (по модулю меньше единицы), что упрощает вычисления.

Основой организации арифметических вычислений с дробными числами на процессорах с фиксированной точкой являются следующие положения (для 8-разрядного представления):

– Входной и выходной аргумент представляются в формате 0xNNh (в этом случае целое число NNh просто-напросто интерпретируется как 0xNNh: например, для беззнаковых чисел $0x40h=0.25_{10}$, $0xC0h=0.75_{10}$ и т. д.).

– При умножении двух дробных 8-разрядных чисел, представленных в формате 0xNNh, можно использовать команду целочисленного умножения; а полученный 16-разрядный результат 0xNNNNh при необходи-

мости приводится к 8-разрядному формату 0xNNh путем простого отбрасывания младшего байта. Например, 8Ah×C0h=6780h, причем это соответствует как точному равенству для целых чисел $138_{10}\times192_{10}=26496_{10}$, так и приближенному, – для дробных $0.5390_{10}\times0.75_{10}=0.4043_{10}$. Отбрасывание младшего байта полученного 16-разрядного результата 6780h для целых чисел приводит к полной бессмыслице, а для дробных лишь незначительно изменит результат: $0x67h\cong0.4023_{10}$). Аналогично организуется умножение дробного 16-разрядного числа на дробное 8-разрядное с округлением результата до 16 разрядов.

– Деление дробного числа, представленного в формате 0xNNh, на целое число посредством команды целочисленного деления приводит к результату в формате 0xNNh. То же справедливо для 16-разрядного представления.

– При необходимости умножения дробного числа 0xNNh на правильную дробь (числитель и знаменатель являются целыми числами, знаменатель больше числителя) следует сначала выполнять операцию деления, а результат 0xNNNNh (при необходимости приведенный к 0xNNh) умножить на значение числителя, – при такой последовательности действий переполнения не будет.

Пример 6

Вычислить $Y(X)=\sum_{k=1}^{X}k$ для любого значения аргумента $X\in\left[1,2^8-1\right]$

на основе итерационного алгоритма.

Предварительный теоретический анализ выражения показывает, что максимальное значение результата составит 32640_{10}, т. е. для размещения *Y* следует отвести две 8-разрядные ячейки памяти микроконтроллера. Пусть для переменных в программе отведены следующие ячейки: 1) *R*2 – для *X*, а также и для *k*; 2) *R*3 – для младшего байта результата *Y*; 3) *R*4 – для старшего байта результата *Y*. Тогда программа может иметь следующий вид:

Адрес	Команда	Комментарий
0000	**MOV R2,#FA**	Загрузка входного аргумента X=250$_{10}$
0002	**CLR A**	Обнуление аккумулятора
0003	**MOV R3,A**	Обнуление ячеек, в которых размещается результат Y
0004	**MOV R4,A**	
0005	**CLR C**	Обнуление бита переноса
0006	**MOV A,R3**	Сложение 16-разрядной целочисленной переменной $R4$ $R3$ с 8-разрядной целочисленной переменной $R2$. (*Здесь следует обратить внимание на опе-рации с битом переноса, обычно применяемые при сложении многобайтовых чисел*)
0007	**ADDC A, R2**	
0008	**MOV R3, A**	
0009	**MOV A, R4**	
000A	**ADDC A, #00**	
000C	**MOV R4, A**	
000D	**DJNZ R2, 0005**	Декремент переменной k, размещенной в $R2$, и проверка на ноль. Если $k\neq0$, то производится переход к операции суммирования двухбайтового числа с однобайтовым; если же k=0, то вычисления завершаются.

Очевидно, что в данном случае вычисление суммы начинается «с конца» – за счет использования многофункциональной команды DJNZ получается более короткий и быстрый код.

Следующий пример показывает, каким образом следует строить программу вычисления бесконечного ряда с дробными числами.

Пример 7

Вычислить $Y(X)=\sum_{k=1}^{\infty}\frac{X^k}{k+1}$, $X\in\left[0,\frac{\pi}{4}\right]$. Вычисления следует производить с 16-разрядной точностью.

В данном случае известно, что $Y\in[0,1]$, т. е. результат Y является положительным дробным числом. Критерием для завершения итерационных вычислений является равенство машинному нулю очередного члена ряда.

Обобщенная блок-схема алгоритма вычислений имеет следующий вид:

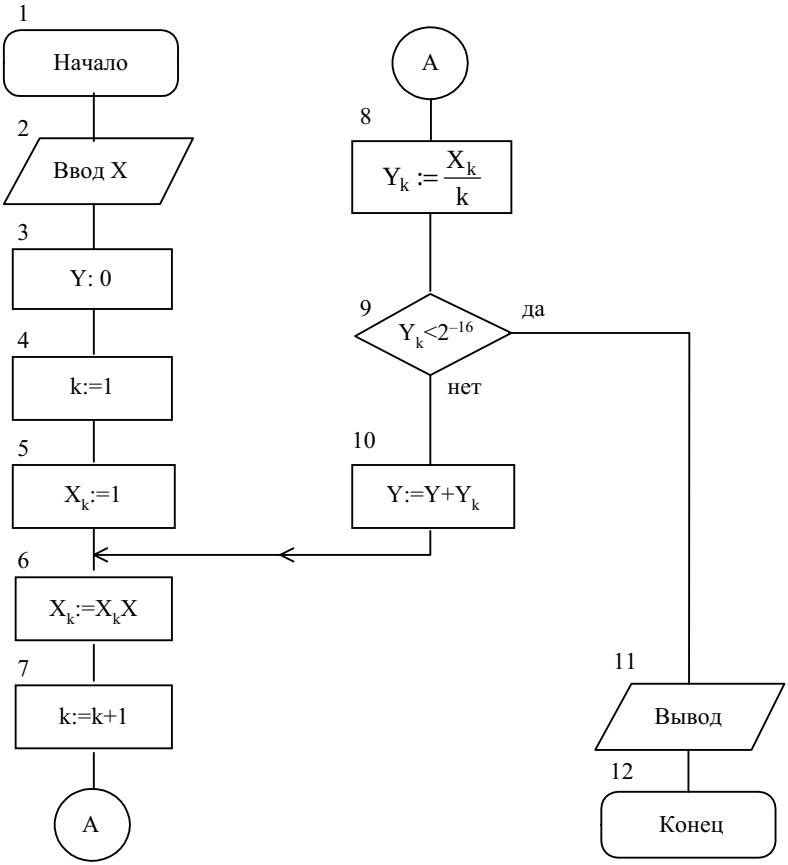


Рис. 15

Проверка максимального количества итераций не производится, так как при 16-разрядном представлении чисел при $k=2^{16}$ очередной член ряда Y_k окажется равен машинному нулю при любом допустимом значении X.

Для размещения каждой из переменных, используемых в программе, требуется зарезервировать по две байтовые ячейки памяти; так что для пяти переменных программы потребуется 10 ячеек. Это вызывает некоторые неудобства: при размещении операндов в РПД не удастся работать только с одним банком РОН. «Лишний» операнд Y_k (он вычисляется заново в каждой итерации) хранится в регистрах $R0$ и $R1$ 1-го банка РОН, а иногда – в стеке.

Пусть операнды размещены в нулевом банке РОН следующим образом: 1) для аргумента X отведены регистры $R0$ (для младшего байта X) и $R1$ (для старшего байта X), 2) для k – $R2$ и $R3$, 3) для X_k – $R4$ и $R5$, 4) для Y – $R6$ и $R7$. Программа имеет вид:

Адрес	Команда	Комментарий
0000	**MOV R0,#00**	Загрузка 16-разрядного аргумента C000h, т. е. X:=0.75_{10}
0002	**MOV R1,#C0**	
0004	**MOV R2,#01**	Загрузка 16-разрядного значения переменной цикла k:=1_{10} (или 0001h)
0006	**MOV R3,#00**	
0008	**MOV R4,#FF**	Присвоение *дробной* переменной X_k единичного значения: X_k:=1 (точнее 0xFFFFh=0.999984_{10})
000A	**MOV R5,#FF**	
000C	**MOV R6,#00**	Обнуление ячеек РОН, отведенных под размещение 16-разрядного результата Y
000E	**MOV R7,#00**	
0010	**MOV 08,R6**	Обнуление ячеек РОН 1-го банка, отведенных под размещение Y_k
0012	**MOV 09,R7**	
0014	**MOV 81,#1F**	Загрузка указателя стека SP числом #0F для предотвращения пересечения с адресами 1-го банк РОН
0017	**ACALL 0059**	Вызов процедуры вычисления X_k:=X_kX
0019	**INC R2**	Инкремент 16-разрядной переменной k (элемент № 7 в блок-схеме на рис. 15)
001A	**CJNE R2,#00, 001E**	
001D	**INC R3**	
001E	**CLR F0 (PSW.5)**	Обнуление бита, по значению которого будет контролироваться результат деления
0020	**MOV A, R4**	Сохранение в стеке значения 16-разрядной переменной X_k, поскольку используемый метод деления X_k на k методом последовательных вычитаний необратимо меняет значение X_k
0021	**PUSH A**	
0023	**MOV A, R5**	
0024	**PUSH A**	
0026	**CLR C**	Ядро используемой процедуры деления: здесь осуществляется вычитание 16-разрядных чисел: делителя из остающегося делимого. Если по окончании вычитания установится бит C, значит произошел заем, т. е. деление завершено. Этот факт проверяется следующей командой
0027	**MOV A, R4**	
0028	**SUBB A, R2**	
0029	**MOV R4,A**	
002A	**MOV A,R5**	
002B	**SUBB A, R3**	
002C	**MOV R5,A**	

Продолжение табл.

Адрес	Команда	Комментарий
002D	**JC 003C**	Переход в конец процедуры деления, если C=1
002F	**SETB F0 (PSW.5)**	Здесь производится инкремент 16-разрядного результата от деления с установлением флага F0, используемого в качестве индикатора ненулевого результата. Для сохранения результата производится переключение на 1-й банк РОН путем установки бита S0 (3-й бит регистра PSW, его адрес D3h). После сохранения результата в паре регистров *R*0–*R*1 1-го банка РОН производится переключение на 0-й банк
0031	**SETB S0 (PSW.3)**	
0033	**INC R0**	
0034	**CJNE R0,#00, 0038**	
0037	**INC R1**	
0038	**CLR S0 (PSW.3)**	
003A	**SJMP 0026**	Переход к новой итерации процедуры деления
003C	**POP ACC**	Возврат из стека сохраненного ранее значения *Xk*
003E	**MOV R5, A**	
003F	**POP ACC**	
0041	**MOV R4, A**	
0042	**SETB S0 (PSW.3)**	Переключение на 1-й банк РОН
0044	**MOV A, R1**	Загрузка в стек старшего байта 16-разрядного результата от деления
0045	**PUSH A**	
0047	**MOV A, R0**	Загрузка в стек младшего байта 16-разрядного результата от деления.
0048	**PUSH ACC**	
004A	**CLR S0 (PSW.3)**	Возврат к 0-му банку РОН
004C	**JNB F0, 00AC**	Переход к концу программы, если в процессе деления не был установлен флаг F0 (т. е. $Y_k<2^{16}$)
004F	**POP ACC**	Вычисление выражения $Y:=Y+Y_k$ путем последовательного извлечения из стека сохраненных ранее двух байтов 16-разрядного числа Y_k и суммирования с предыдущим значением результата Y
0051	**ADD A, R6**	
0052	**MOV R6, A**	
0053	**POP ACC**	
0055	**ADDC A, R7**	
0056	**MOV R7, A**	
0057	**SJMP 0017**	Переход к очередному итерационному циклу

Продолжение табл.

Адрес	Команда	Комментарий
0059	**MOV A, R4**	Начало подпрограммы умножения. Получение первого частного произведения *R*0·*R*4 и сохранение в стеке его старшего байта
005A	**MOV B,R0**	
005C	**MUL AB**	
005D	**PUSH B**	
005F	**MOV A, R4**	Получение второго частного произведения *R*1·*R*4 и сохранение в стеке сначала младшего байта, потом старшего
0060	**MOV B,R1**	
0062	**MUL AB**	
0063	**PUSH ACC**	
0065	**PUSH B**	
0067	**MOV A, R5**	Получение третьего частного произведения *R*0·*R*5 и сохранение в стеке сначала младшего байта, потом старшего
0068	**MOV B,R0**	
006A	**MUL AB**	
006B	**PUSH ACC**	
006D	**PUSH B**	
006F	**MOV A, R5**	Получение четвертого частного произведения *R*1·*R*5 и пересылка младшего байта в регистр *R*4, старшего – в регистр *R*5
0070	**MOV B,R1**	
0072	**MUL AB**	
0073	**MOV R5,B**	
0075	**MOV R4, A**	
0076	**POP B**	Сложение старшего 16-разрядного слова произведения со старшим байтом третьего частичного произведения
0078	**ADD A,B**	
007A	**MOV R4,A**	
007B	**MOV A,R5**	
007C	**ADDC A,#00**	
007E	**MOV R5,A**	
007F	**POP ACC**	«Перетасовка« отдельных элементов частных произведений в стеке
0081	**POP B**	
0083	**PUSH ACC**	
0085	**MOV A,R4**	Сложение старшего 16-разрядного слова произведения со старшим байтом второго частичного произведения
0086	**ADD A,B**	
0088	**MOV R4,A**	
0089	**MOV A,R5**	

Окончание табл.

Адрес	Команда	Комментарий
008A	**ADDC A,#00**	
008C	**MOV R5,A**	
008D	**POP ACC**	Сложение младших байтов второго и третьего частных произведений и сохранение результата в стеке
008F	**POP B**	
0091	**ADD A,B**	
0093	**PUSH ACC**	
0095	**MOV A,R4**	Суммирование старшего 16-разрядного слова произведения со значением бита переноса, образующегося в результате предыдущего сложения (т. е. младших байтов второго и третьего частных произведений)
0096	**ADDC A,#00**	
0098	**MOV R4,A**	
0099	**MOV A,R5**	
009A	**ADDC A,#00**	
009C	**MOV R5,A**	
009D	**POP ACC**	Сложение результата суммирования младших байтов второго и третьего частных произведений со старшим байтом первого частного произведения
009F	**POP B**	
00A1	**ADD A,B**	
00A3	**MOV A,R4**	Суммирование старшего 16-разрядного слова произведения со значением бита переноса, образующегося в результате предыдущего сложения
00A4	**ADDC A,#00**	
00A6	**MOV R4,A**	
00A7	**MOV A,R5**	
00A8	**ADDC A,#00**	
00AA	**MOV R5,A**	
00AB	**RET**	Возврат из подпрограммы умножения

Нельзя сказать, чтобы данная программа отличалась особым изяществом структуры и уж тем более была бы оптимизирована по какому-либо критерию, – это может быть упражнением для самостоятельной работы. Однако в приведенной программе имеется две процедуры, иллюстрирующие ***принципы программной реализации вычислений с 16-разрядной точностью***:

1. Процедура деления дробного беззнакового 16-разрядного числа на целое 16-разрядное положительное число (фрагмент основной программы в диапазоне адресов 001Ah–0044h) на основе метода целочисленного деления без учета остатка. Для деления используется простой алгоритм последовательных вычитаний (*при выполнении заданий лабора-*

торной работы № 3 данный алгоритм не следует брать за основу!). 16-разрядный результат деления загружается в стек последовательно: *сначала старший байт*, затем младший. Если результат от деления ненулевой, устанавливается флаг F0.

2. Процедура вычисления выражения $X_k := X_k \cdot X$, оформленная в виде подпрограммы (расположена в диапазоне адресов 0053h–007Fh). Умножение дробных беззнаковых 16-разрядных чисел производится на основе метода целочисленного умножения с использованием алгоритма секционированного умножения. Результат «автоматически» округляется до 16 разрядов и пересылается в регистры *R*4 и *R*5. В данной подпрограмме также наглядно демонстрируются преимущества работы со стеком. Изучение ее структуры будет полезным при выполнении заданий лабораторной работы № 6.

При выполнении лабораторных работ, связанных с арифметическими вычислениями, для проверки правильности получаемых результатов удобно использовать программу «Калькулятор», поставляемую вместе с операционной системой Windows. Например, получение десятичного эквивалента дробного числа, представленного в шестнадцатеричном формате, производится по следующему алгоритму: при нажатой кнопке «Hex» набирается шестнадцатеричный код, затем нажимается кнопка «Dec», после чего число нужно поделить на 256_{10} (при 8-разрядном представлении исходного числа: 0xNNh) или на 65536_{10} (при 16-разрядном представлении исходного числа: 0xNNNNh). Для получения шестнадцатеричного кода дробного числа, записанного в десятичном формате, производятся обратные действия.

Реально полученные результаты работы программы из примера 7 для четырех различных значений аргумента *X* приведены в табл. 1.

Таблица 1

Входной аргумент *X*		Точное значение *f*(*X*) для *X*, dec		Результат *Y*, полученный в программе		Погрешность	
hex	dec	hex	dec	hex	dec	абс., ед.	ε(X), dec
C000	0,7500	D930	0,84839	D91D	0,84810	19	0,00034
8000	0,5000	62E3	0,38629	62DC	0,38617	7	0,00031
4000	0,2500	2696	0,15073	2692	0,15066	4	0,00046
1999	0,1000	0DB8	0,05360	0DB7	0,05357	1	0,00056

Погрешность полученного результата оценивается следующим образом (пример для первой строки табл. 1):

– абсолютная: D930h–D91Dh=0013h=1910 (т. е. 19 единиц двоичного кода);

– относительная погрешность $\varepsilon(X)=\left|1-\frac{Y}{f(X)}\right|$ полученного результата Y для аргумента X=0.75$_{10}$ составляет: $\varepsilon(0,75)=\left|1-\frac{0,84810}{0,84839}\right|\cong 0,00034$.

Работа с массивами данных. Для адресации элементов массивов данных, размещенных во внешнем ОЗУ, используются команды (19)– (22).

Пример 8

Найти значение наибольшего элемента массива данных из 2^{16} элементов (целые положительные 8-разрядные числа), размещенного в расширенной ВПД, и вывести значение этого элемента и его порядковый номер в массиве.

Пусть для переменных программы отведены следующие ячейки: 1) для размещения максимального значения массива X_{max} – регистр $R2$, 2) для сохранения индекса элемента с максимальным значением – регистры $R3$ (для младшего байта) и $R4$ (для старшего байта). Программа имеет следующий вид:

Адрес	Команда	Комментарий
0000	**MOV DPTR, #0000**	Обнуление указателя данных в ВПД
0003	**MOV R2,#00**	Пусть X_{max}=0
0005	**MOV A,@DPTR**	Пересылка $X[i]$ из ВПД в аккумулятор
0006	**CLR C**	Вычитание $R2=X_{max}$ из аккумулятора: если не возникнет заем (т. е. если после вычитания по-прежнему будет C=0), то значит $X[i]>X_{max}$, и необходимо выполнить операцию $X_{max}=X[i]$
0007	**SUBB A, P2**	
0008	**JNC 0010**	Если C=1 (т. е. $X_{max}\geqslant X[i]$), то *обходится* фрагмент программы, размещенный по адресам 000A–000F, в котором сохраняется значение X_{max} и его индекс

Адрес	Команда	Комментарий
000A	**MOV A,@DPTR**	Пересылка элемента из ВПД в регистр *R*2, т. е. $X_{max}=X[i]$
000B	**MOV R2,A**	
000C	**MOV R3,DPL**	Сохранение индекса элемента массива с максимальным значением
000E	**MOV R4,DPH**	
0010	**INC DPTR**	Увеличение индекса очередного выбираемого элемента массива данных в ВПД
0011	**MOV A, DPL**	Проверка окончания длины массива: если после инкремента указатель данных обнулен (DPTR=0: т. е. если и DPL=0, и DPH=0), то выполнение программы завершится и выводится результат. Если же перебор массива не закончен, то производится переход на адрес 0005)
0013	**JNZ 0005**	
0015	**MOV A, DPH**	
0017	**JNZ 0005**	

4. Задания[#] на лабораторный практикум

Лабораторная работа № 1

ПРИМЕНЕНИЕ ЛОГИЧЕСКИХ ОПЕРАЦИЙ

1. ***Сформировать*** **«*бесконечную*» *последовательность*** в порту *P*1 в соответствии с вариантом из табл. 2 (в данном задании *использование в программе каких-либо заранее вычисленных констант не допускается*!).

Таблица 2

№ такта	Вариант задания							
	1	2	3	4	5	6	7	8
1	00000000	00000000	11111111	11111111	00000000	00000000	00000001	00000000
2	00000001	00000001	11111110	11111110	10000001	00000001	00000011	00000010
3	00000010	00000011	11111100	11111101	11000011	00000011	00000101	00000100
4	00000100	00000111	11111000	11111011	11100111	00000101	00000111	00000110
5	00001000	00001111	11110000	11110110	11111111	00001001	00001001	00001000
6	00010000	00011111	11100001	11101101	11100111	00010001	00001011	00001010
7	00100000	00111111	11000011	11011011	11000011	00100001	00001101	00001100
8	01000000	01111111	10000111	10110110	10000001	01000001	00001111	00001110
9	10000000	11111111	00001111	01101101	00000000	10000001	00010001	00010000
10	00000000	00000000	00011110	11011011	10000001	00000000	00010011	00010010

[#] Символом # отмечены задания повышенной сложности.

2. ***Организовать генератор псевдослучайных чисел*** по следующему принципу: в аккумуляторе задается исходное ненулевое число *N*, затем выполняется логическая операцию заданного вида с битами ACC.X и ACC.Y, результат которой записывается в бит переноса *C*, после чего производится операция циклического сдвига аккумулятора через бит переноса; полученное содержимое аккумулятора пересылается в порт вывода (например, в *P*1) и процедура вновь повторяется, начиная с логической операции с битами аккумулятора. Варианты заданий приведены в табл. 3.

Таблица 3

Параметры N, hex	Вариант задания							
	1	2	3	4	5	6	7	8
	A5	D7	48	06	99	40	13	FF
Логическая операция	И	И-НЕ	ИЛИ	ИЛИ-НЕ	Искл. ИЛИ	НЕ	И-НЕ	Искл. ИЛИ
X	6	5	7	5	4	3	1	2
Y	1	2	3	4	5	-	Бит C	Бит C
Сдвиг	влево	влево	вправо	вправо	влево	влево	вправо	вправо

Лабораторная работа № 2

ПРИМЕНЕНИЕ БИТОВЫХ ОПЕРАЦИЙ

1. *Выполнить операцию* «*логического умножения со сборкой*» (варианты заданий приведены в табл. 4). Операция состоит в том, что из исходного двоичного числа *N* извлекаются цифры тех разрядов, в которых в другом из исходных чисел *M* (так называемом «извлекателе») содержатся единицы; выбранные цифры затем располагаются одна за другой в старших разрядах результата, а младшие разряды забиваются нулями.

Таблица 4

Параметры N, hex	Вариант задания							
	1	2	3	4	5	6	7	8
	40	5F	48	90	B3	7B	B2	E7
M, hex	13	7A	AC	49	3A	A3	F1	C4

2. *Определить встречается ли в 16-разрядном числе заданная последовательность битов* (варианты заданий приведены в табл. 5).

Таблица 5

Параметр	Вариант задания							
	1	2	3	4	5	6	7	8
Последовательность	40	5F	48	90	B3	7B	B2	E7

3. *Подсчитать количество нулей в 16-разрядном числе.*

4. *Подсчитать количество единиц в 16-разрядном числе.*

5. *Подсчитать количество пар нулей, стоящих рядом в 16-разрядном числе.*

6. *Подсчитать количество пар единиц, стоящих рядом в 16-разрядном числе.*

7. *Подсчитать количество троек нулей, стоящих рядом в 16-разрядном числе.*

8. *Подсчитать количество троек единиц, стоящих рядом в 16-разрядном числе.*

9. *Осуществить бит-реверсивное преобразование 8-разрядного (или 16-разрядного) числа* (самый старший бит двоичного числа меняется местами с самым младшим, следующий – с предпоследним, и т. д.).

10. *Реализовать дешифратор на 16 выходов* (4-битный входной двоичный код находится в одном из портов, результат – в двух других).

11. *Реализовать дешифратор кода для управления семисегментным цифровым индикатором* (4-битный входной двоичный код находится в одном из портов, результат представляется в другом порту; предполагается, что к 7-битовым выходам порта подключены сегменты, светящиеся при наличии 1).

Лабораторная работа № 3

ОПЕРАЦИИ ЦЕЛОЧИСЛЕННОЙ АРИФМЕТИКИ

1. *Вычислить произведение 16-разрядного целого числа и 8-разрядного целого числа без учета их знаков.*

2. *Вычислить произведение 16-разрядного целого числа и 8-разрядного целого числа с учетом знаков обоих чисел.*

3. *Вычислить произведение двух 16-разрядных целых чисел без учета их знаков.*

4. *Вычислить произведение двух 16-разрядных целых чисел с учетом их знаков.*

5. *Разделить 16-разрядное положительное целое число на 8-разрядное целое положительное число*: получить частное и остаток от деления.

6. *Умножить знаковое 16-разрядное целое число на 8-разрядное число, равное* 2^m, при условии сохранения знака $\left(m \in [1, 7]\right)$.

7. *Разделить знаковое 16-разрядное целое число на 8-разрядное число, равное* 2^m, при условии сохранения знака $\left(m \in [1, 7]\right)$.

8. *Округлить 8-разрядное целое число, представленное в двоичном позиционном коде, до десятков* (округление по законам десятичной арифметики: т. е., например, в *десятичном* представлении 243≈240 (для данного случая в программе из F3h должно быть получено F0h), 247≈250.

9. *Преобразовать 16-разрядное целое число, представленное в двоичном позиционном коде, в код Грея* (указание: каждый *i*-й бит числа в коде Грея образуется как сумма по модулю два *i*-го и соседнего (*i*+1)-го (более старшего) разрядов двоичного позиционного кода того же числа).

10. *Преобразовать 16-разрядное число, представленное в коде Грея, в двоичный позиционный код* (указание: каждый *i*-й бит числа, представленного в двоичном позиционном коде, образуется как сумма по модулю два *i*-го и всех более старших разрядов кода Грея того же числа).

11. *Перевести 8-разрядное целое положительное число, представленное в двоичном коде, в двоично-десятичный формат* (под размещение результата понадобится 10 двоичных разрядов).

12. *Перевести 8-разрядное целое положительное число из двоично-десятичного формата в двоичный позиционный код.*

13. *Перевести значение температуры из градусов Цельсия в градусы Фаренгейта* (*указание*: $C^\circ = \frac{5}{9}\left(F^\circ - 32_{10}\right)$. Оба значения представляются *в двоично-десятичном формате* (так что *обе величины* не превышают 99_{10}); знак числа следует закодировать значением бита F0 в регистре PSW).

14. *Перевести значение температуры из градусов Фаренгейта в градусы Цельсия* (см. указания и требования к предыдущему заданию).

15. *Классифицировать 8-разрядное целое число*: простое или непростое, четное или нечетное, отрицательное или положительное – результат представить набором из 3 битов.

Лабораторная работа № 4

АРИФМЕТИЧЕСКИЕ ВЫЧИСЛЕНИЯ С ЦЕЛЫМИ ЧИСЛАМИ

1. *Найти все простые числа до заданного* $N \in [1, 255]$.

2. *Найти наибольший общий делитель* двух заданных чисел $N_1 \in [1,255]$, $N_2 \in [1, 255]$ (это суть максимальное целое число, на которое делятся без остатка и $N1$, и $N2$; можно использовать алгоритм Евклида [12]).

3. #*Найти наименьшее общее кратное* двух заданных чисел N_1 и N_2 $N_1, N_2 \in [1, 255]$ (это суть минимальное целое число, которое делится без остатка и на N_1, и на N_2; под результат нужно отвести 2 ячейки памяти).

4. *Разложить число* $N \in [1, 255]$ *на простые множители*, результат – совокупность множителей – разместить в ячейках внутреннего ОЗУ.

5. *Рассчитать числа Фибоначчи* (каждое новое число равно сумме двух предыдущих, в десятичной записи: 1, 1, 2, 3, 5, 8, 13, 21, 34 и т. д.), не превышающие заданного числа $N \in \left[2^8, 2^{16}-1\right]$. Результат разместить во внешнем ОЗУ (отображается в окне XData), отводя 2 ячейки на каждое число.

6. *Вычислить* $\sum_{k=1}^{N}(2k-1)$ для заданного $N \in [1, 255]$.

7. *Вычислить* $\sum_{k=1}^{N}\left(k+(-1)^k\right)$ для заданного $N \in [1, 255]$.

8. *Извлечь квадратный корень из 8-разрядного целого числа* методом вычитания нечетных чисел [8] (ответ – целая часть результата).

9. *Извлечь квадратный корень из 16-разрядного целого числа* методом вычитания нечетных чисел [8] (ответ – целая часть результата).

10. #*Извлечь квадратный корень из 8-разрядного целого числа* итерационным методом: $\sqrt{X} = \lim_{k\to\infty}(n_k)$, где $n_{k+1} = \frac{1}{2}\left(n_k + \frac{X}{n_k}\right)$. *Указание*: в итерациях использовать целую часть от деления.

11. #*Выполнить предыдущее задание для 16-разрядного целого числа.*

12. #*Извлечь квадратный корень из 8-разрядного целого числа*, используя бесконечную «цепную» дробь: $\sqrt{X} = 1 + \dfrac{X-1}{2 + \dfrac{X-1}{2 + \dfrac{X-1}{2 + \cdots}}}$.

Указание: в итерациях использовать целую часть от деления.

13. #*Выполнить предыдущее задание для 16-разрядного целого числа.*

14. #*Извлечь квадратный корень из 8-разрядного целого числа* итерационным методом, используя соотношения: $\sqrt{X} = \lim\limits_{k\to\infty}\left(\dfrac{n_k}{m_k}\right)$, где $n_{k+1} = n_k^2 - 2;\ \ m_{k+1} = n_k m_k$; начальные значения для итерации выбираются на основе уравнения $n_0^2 - Xm_0^2 = 4$. В отчете следует привести целую часть результата, а также предельное значение m_k и остаток от деления n_k на m_k.

Задания лабораторных работ № 3 и 4 считаются выполненными успешно, если целая часть результата вычислена абсолютно точно.

Лабораторная работа № 5

АРИФМЕТИЧЕСКИЕ ВЫЧИСЛЕНИЯ С ДРОБНЫМИ ЧИСЛАМИ

1. *Перевести 8-разрядное дробное число, представленное в двоичном позиционном коде, в двоично-десятичный формат.*

2. *Перевести 8-разрядное дробное положительное число, представленное в двоично-десятичном формате, в двоичный позиционный код.*

3. *Рассчитать значение элементарной функции Y = f(X) по приближенным формулам* [9], приведенным в табл. 6. Значение аргумента *X* задается из диапазона, где одновременно $X \in [0..1]$ и $f(X) \in [0..1]$, если в варианте задания не указаны иные пределы изменения аргумента. Аргумент тригонометрических функций везде предполагается заданным в радианах. *Все коэффициенты в формулах рассчитываются заранее и в программе загружаются как константы*. При выполнении данной группы заданий для представления как аргумента *X*, так и результата *Y* (во

всех вариантах это положительные дробные числа) следует использовать формат с фиксированной точкой вида 0xNN (вычисления с 8-разрядной точностью) или 0xNNNN (вычисления с 16-разрядной точностью – т. е. и для *X*, и для *Y* отводится по две 8-разрядные ячейки ОЗУ микроконтроллера).

Лабораторная работа № 6

ИТЕРАЦИОННЫЕ ВЫЧИСЛЕНИЯ

Рассчитать значение функции Y = f(X) или константы (варианты с 10 по 14), используя разложения в ряды [10], приведенные в табл. 7. Аргумент *X* задается из диапазона, где одновременно $X \in [0..1]$ и $f(X) \in [0..1]$, если в варианте задания не указаны иные пределы. Аргумент тригонометрических функций везде предполагается заданным в радианах. Завершение итераций – адаптивное, по результатам проверки в программном цикле значения очередного члена ряда. *Все используемые в формулах коэффициенты должны рассчитываться итерационно в программе.* При выполнении заданий данной подгруппы для представления как аргумента *X*, так и результата *Y* (во всех вариантах это положительные дробные числа) следует использовать формат с фиксированной точкой вида 0xNNNN (вычисления с 16-разрядной точностью).

Указание: организовать две процедуры: 1) умножения 16-разрядного дробного числа на 16-разрядное дробное число; 2) деления 16-разрядного дробного числа на 16-разрядное целое.

Задания лабораторных работ № 5 и 6 считаются выполненными успешно, если относительная погрешность полученного результата для любого заданного преподавателем значения аргумента (из области его допустимых значений) не превышает 0,05 для вычислений с 8-разрядной точностью и 0,005 для вычислений с 16-разрядной точностью. При малых значениях аргумента, где относительная погрешность может резко возрастать, *абсолютная* погрешность не должна превышать 4 единицы двоичного кода для вычислений с 8-разрядной точностью и 128 единиц двоичного кода для вычислений с 16-разрядной точностью.

Лабораторная работа № 7

ОБРАБОТКА МАССИВОВ ДАННЫХ

1. *Осуществить сортировку 8-разрядных знаковых чисел массива, размещенных во внутреннем ОЗУ* (длина массива составляет 2^N чисел, значение параметра $N \in [2, 7]$ задается преподавателем, непрерывная область адресов памяти для размещения исходного массива выбирается студентом).

2. *Осуществить сортировку 8-разрядных знаковых чисел массива, размещенных во внешнем ОЗУ* (длина массива составляет 2^N чисел, значение параметра $N \in [2, 16]$ задается преподавателем, непрерывная область адресов памяти для размещения исходного массива выбирается студентом).

3. *Рассчитать среднее арифметическое массива 8-разрядных знаковых чисел, размещенных во внутреннем ОЗУ* (длина массива составляет 2^N чисел, значение параметра $N \in [2, 7]$ задается преподавателем, непрерывная область адресов для размещения исходного массива выбирается студентом).

4. *Рассчитать среднее арифметическое массива 8-разрядных знаковых чисел, размещенных во внешнем ОЗУ* (длина массива составляет 2^N чисел, значение параметра $N \in [2, 16]$ задается преподавателем, непрерывная область адресов памяти для размещения массива выбирается студентом).

5. *Рассчитать среднее квадратическое массива 8-разрядных знаковых чисел, размещенных во внутреннем ОЗУ* (длина массива составляет 2^N чисел, значение параметра $N \in [2, 7]$ задается преподавателем, непрерывная область адресов памяти для размещения исходного массива выбирается студентом).

6. *Рассчитать среднее квадратическое массива 8-разрядных знаковых чисел, размещенных во внешнем ОЗУ* (длина массива составляет 2^N чисел, значение параметра задается преподавателем, непрерывная область адресов для размещения исходного массива выбирается студентом).

7. *Заполнить внешнее ОЗУ микроконтроллера объемом 64кБайт последовательностями кодов* (которые в реальных приложениях могут использоваться для генерации сигналов методом выборки из ОЗУ с последующей выдачей в порт ввода–вывода) в соответствии вариантами задания (табл. 8). На линейных участках графиков предполагается из-

Таблица 6

Номер варианта	Формулы для приближенного вычисления функций $f(X)$
1	$\exp(X) \approx 0,9946 + 0,9974X + 0,5430X^2 + 0,1771X^3$
2	$\exp(-X) \approx 1 + X\,(-0,9664 + 0,3536X);\ X \in [0,\ \ln(2)]$
3	$\ln(1+X) \approx X - 0,5\ X^2\ \ X \in [0,\ 0,5]$
4	$\ln(1+X) \approx X\,(1,25 - 0,5X)$
5	$\lg(X) \approx \frac{X-1}{X+1}\left(0,8630 + 0,36415\left(\frac{X-1}{X+1}\right)^3\right);\ X \in \left[10^{-0,5},\ \ 1\right]$
6	$\sin(X) \approx 0,9974\ X - 0,1562\ X^3$
7	$\sin(X) \approx X\left(1 + X^2\left(-0,16605 + 0,00761X^2\right)\right)$
8	$\sin(X) \approx X\left(1 - \frac{1}{3!}X^2\left(1 - \frac{3!}{5!}X^2\left(1 - \frac{5!}{7!}X^2\left(1 - \frac{7!}{9!}X^2\right)\right)\right)\right)$
9	$\cos(X) \approx 1 + X^2\left(-0,4967 + 0,03705X^2\right)$
10	$\cos(X) \approx \left(1 - \frac{1}{2!}X^2\left(1 - \frac{2!}{4!}X^2\left(1 - \frac{4!}{6!}X^2\left(1 - \frac{6!}{8!}X^2\right)\right)\right)\right)$
11	$\mathrm{tg}(X) \approx X\left(1 + X^2\left(0,31755 + 0,2033X^2\right)\right);\ X \in [0,\ 025\pi]$
12	$\mathrm{tg}(X) \approx X\left(1 + \frac{1}{3}X^2\left(1 + \frac{6}{10}X^2\left(1 + \frac{255}{630}X^2\right)\right)\right)$
13	$\arcsin(X) \approx X\left(1 + \frac{1}{6}X^2\left(1 + \frac{18}{40}X^2\left(1 + \frac{600}{1008}X^2\right)\right)\right)$
14	$\mathrm{arctg}(X) \approx X\left(1 - \frac{1}{3}X^2\left(1 - \frac{3}{5}X^2\left(1 - \frac{5}{7}X^2\right)\right)\right)$

Таблица 7

Номер варианта	Ряды для вычисления функций $f(X)$ и констант
1	$\exp(-X)=1-X+\frac{X^2}{2!}-\frac{X^3}{3!}+\frac{X^4}{4!}\mp\cdots$
2	$\ln(1+X)=X-\frac{X^2}{2}+\frac{X^3}{3}-\frac{X^4}{4}\pm\cdots$
3	$\sin(X)=X-\frac{X^3}{3!}+\frac{X^5}{5!}\mp\cdots$
4	$\sin(\pi X)=X\prod_{k=1}^{\infty}\left(1-\left(\frac{X}{k}\right)^2\right)$
5	$\cos(X)=X-\frac{X^2}{2!}+\frac{X^4}{4!}\mp\cdots$
6	$\cos(\pi X)=\prod_{k=0}^{\infty}\left(1-\left(\frac{2X}{2k+1}\right)^2\right)$
7	$\arcsin(X)=X+\frac{1}{2}\frac{X^3}{3}+\frac{1}{2}\frac{3}{4}\frac{X^5}{5}+\frac{1}{2}\frac{3}{4}\frac{5}{6}\frac{X^7}{7}+\cdots$
8	$\operatorname{arctg}(X)=X-\frac{X^3}{3}+\frac{X^5}{5}\mp\cdots$
9#	$\sqrt{1-X}=1-\frac{X}{2}-\frac{1}{1\cdot 2}\left(\frac{X}{2}\right)^2-\cdots-\frac{1\cdot 3\cdot 5\cdot\ldots\cdot(2\cdot n-3)}{n!}\left(\frac{X}{2}\right)^n-\cdots,\ X\in[0,\ 0{,}9]$
10	$\frac{\pi}{4}=\frac{2}{3}\frac{4}{3}\frac{4}{5}\frac{6}{5}\frac{6}{7}\frac{8}{7}\frac{8}{9}\cdots$
11	$\frac{\pi}{4}=\sum_{k=1}^{\infty}\frac{(-1)^{k-1}}{2\cdot k-1}$
12	$\frac{\pi^2}{6}-1=\sum_{k=2}^{\infty}\frac{1}{k^2}$
13	$\mathrm{e}-2=\sum_{k=2}^{\infty}\frac{1}{k!}$
14	$\ln(2)=\sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{k}$

менение кода на единицу. Для представления отрицательных чисел следует использовать дополнительный код. В варианте № 12 требуется получить периодическую последовательность кодов, убывающих по экспоненциальному закону: в аналитической записи для одного периода $\exp(-N_1 k)$, где в пределах каждого периода $k \in [0, N-1]$, 8-разрядная константа N_1 интерпретируется как положительное дробное число, а N – как положительное целое число; в варианте № 13 требуется получить последовательность дискретных значений функции $\sin\left(2\pi\frac{k}{N}\right)$, $k = 0, 1, \ldots 2^{16}-1$; в № 14 – $\cos\left(2\pi\frac{k}{N}\right)$, $k = 0, 1, \ldots 2^{16}-1$.

Во избежание разночтений все варианты заданий дополнены примерами последовательностей при следующих значениях входных параметров: $N = 8$, $N_1 = 8$, $N_2 = 4$. В каждом варианте подпоследовательность, выделенная жирным шрифтом, соответствует одному периоду сигнала и должна далее повторяться.

8. *Реализовать нерекурсивный фильтр* в соответствии с алгоритмом

$$Y[k] = \frac{1}{N}\sum_{n=0}^{N-1} X[k-n],\ k = N-1,\ N,\ N+1, \ldots 2^{16}-1,$$

где $N = 2m$, $m \in [1, 7]$ – параметр фильтра; $X[k]$ – исходный массив из 2^{16} чисел, содержащийся во внешнем ОЗУ микроконтроллера; $Y[k]$ – последовательность чисел на выходе фильтра.

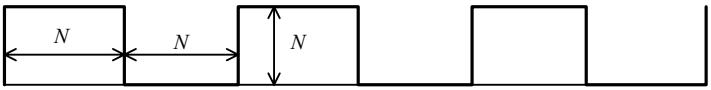
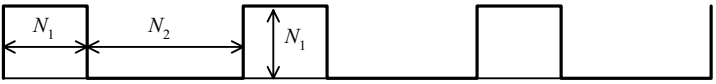
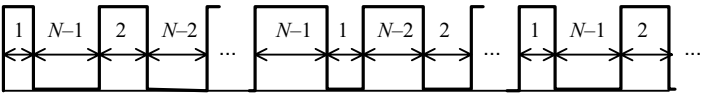
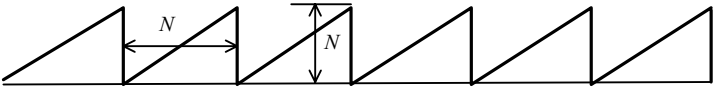
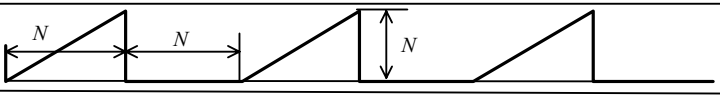
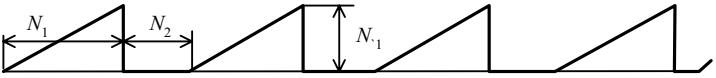
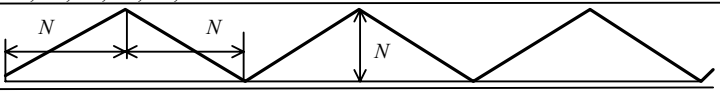
9. #*Реализовать рекурсивный фильтр 1-го порядка* в соответствии с рекуррентным алгоритмом

$$Y[k] = a\,Y[k-1] + b\,X[k],\ k = 1,\ 2 \ldots 2^{16}-1,$$

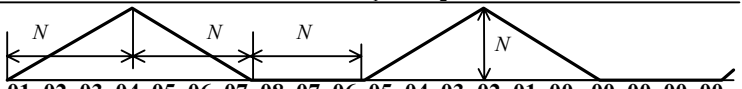
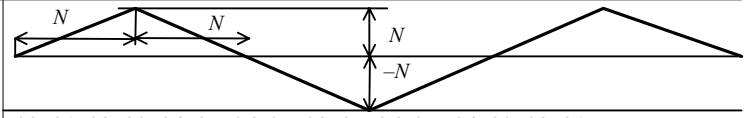
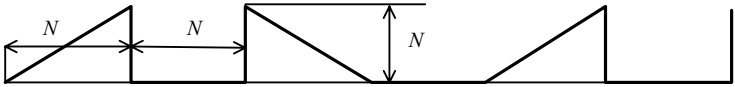
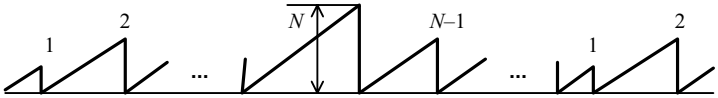
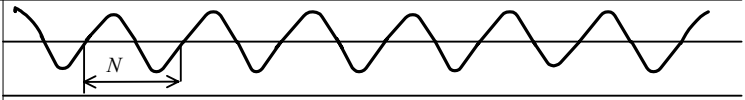
где $a \in (-1, 1)$ и $b = \sqrt{1-a^2}$ = – *заранее* рассчитанные коэффициенты фильтра, в программе они вводятся как константы. Начальное условие $Y[0] = 0$.

Поскольку в демонстрационной версии программы отладчика невозможно моделировать поток ввода–вывода, то при реализации всех фильтров выходную последовательность $Y[k]$ следует записывать в ОЗУ на место исходных данных. Формат представления отсчетов входной и выходной последовательности – 0xNN со знаком. При возникновении переполнения следует производить ограничение значений выходных отсчетов по модулю 1.

*Таблица 8**

Номер варианта	Схематичное изображение требуемой последовательности кодов в виде «эпюры напряжения». Числовые значения приведены для $N = 8$, $N_1 = 8$, $N_2 = 4$
1	 **08, 08, 08, 08, 08, 08, 08, 08, 00, 00, 00, 00, 00, 00, 00, 00,** 08, 08, 08, 08, 08, 08, 08, 08, 00, 00, 00, 00, 00, 00, 00, 00, 08, ...
2	 **08, 08, 08, 08, 08, 08, 08, 08, 00, 00, 00, 00,** 00, 00, 00, 00, 08, ...
3	 **01, 00, 00, 00, 00, 00, 00, 00, 00, 02, 02, 00, 00, 00, 00, 00, 00, 03, 03, 03, 00, ... 00, 07, 07, 07, 07, 07, 07, 07, 00, 06, 06, 06, 00, 06, 06, 06, 00, 00, 05, 05,** ... 02, 02, 00, 00, 00, 00, 00, 00, 01, 00, 00, 00, 00, 00, 00, 00, 02, 02, 00, 00, 00, 00, 00, ...
4	 **00, 01, 02, 03, 04, 05, 06, 07, 08,** 01, 02, 03, 04, 05, 06, 07, 08**,** 00, 01, 02, 03, 04, 05, 06, 07, 08, 00, 01, ...
5	 **01, 02, 03, 04, 05, 06, 07, 08, 00, 00, 00, 00, 00, 00, 00, 00,** 01, 02, 03, 04, 05, 06, 07, 08, 00, 00, 00, 00, 00, 00, 00, 00, 01, ...
6	 **01, 02, 03, 04, 05, 06, 07, 08, 00, 00, 00, 00,** 00, 00, 00, 00, 01, ...
7	 **00, 01, 02, 03, 04, 05, 06, 07, 08, 07, 06, 05, 04, 03, 02, 01,** 00, 01, 02, 03, 04, 05, 06, 07, 08, 07, ...

Окончание табл. 8

Номер варианта	Схематичное изображение требуемой последовательности кодов в виде «эпюры напряжения». Числовые значения приведены для $N = 8$, $N_1 = 8$, $N_2 = 4$
8	 **01, 02, 03, 04, 05, 06, 07, 08, 07, 06, 05, 04, 03, 02, 01, 00, 00, 00, 00, 00, 00, 00, 00,** 01, 02, 03, 04, 05, 06, 07, 08, 07, ...
9	 **00, 01, 02, 03, 04, 05, 06, 07, 08, 07, 06, 05, 04, 03, 02, 01, FF, FE, ED, FC, FB, FA, F9, F8, F9, FA, FB, FC, FD, FE, FF,** 00, 01, 02, 03, 04, 05, 06, 07, 08, 07, 06, 05, 04, 03, 02, 01, 00, FF, ...
10	 **01, 02, 03, 04, 05, 06, 07, 08, 00, 00, 00, 00, 00, 00, 00, 00, 08, 07, 06, 05, 04, 03, 02, 01, 00, 00, 00, 00, 00, 00, 00, 00, 00,** 00, 00, 00, 00, 00, 00, 00, 00, 08, 07, 06, 05, 04, 03, ...
11	 **00, 01, 00, 01, 02, 00, 01, 02, 03, 00, 01, ... 06, 07, 00, 06, 07, 08, 00, 01, 02, 03, 04, 05, 06, 07, 08, 00, 01, 02, 03, 04, 05, 06, 07, 00,01, 02, 03, 04, 05, 06, 00,01, ... 03, 00, 01, 02,** 00, 01, 00, 01, 02, 00, 01, 02, 03, 00, 01, 02, 03, 04, 00, 01, ...
12	 **7F, 7B, 77, 73, 6F, 6F, 6B, 67, 63,** 7F, 7B, 77, 73, 6F, 6B, 63, 7F, 7B, ...
13#	 **00, 5A, 7F, 5A, 00, A6, 80, A6,** 00, 5A, 7F, 5A, 00, 5A, 00, A6, 80, A6, 00, 5A, ...
14#	 **7F, 5A, 00, A6, 80, A6, 00, 5A, 7F**, 5A, 00, A6, 80, A6, 00, 5A, 7F, 5A, ...

* Здесь везде N, N_1, $N_2 \in [01h, 7\Phi h]$, для вариантов № 13 и 14 значение константы $N = 2^m$, $m \in [4, 16]$ задается преподавателем.

5. Требования к отчету о выполнении лабораторной работы

Обязательными разделами отчета о лабораторной работы являются:

1. Титульный лист.

2. Формулировка задания на лабораторную работу.

3. Используемые расчетные соотношения и пояснения к ним (при необходимости приводятся рисунки, временные диаграммы и т. п.)

4. Блок-схема алгоритма программы, оформленная в соответствии с требованиями ЕСПД.

5. Комментарии к блок-схеме: в пояснительном тексте обязательно раскрывается функциональное назначение каждой переменной программы и указывается, какие именно ячейки памяти микроконтроллера отведены под их размещение.

6. Текст программы на языке ASM-51, с указанием адресов команд, их шестнадцатеричных кодов, а также с комментариями, достаточными для пояснения связи с блок-схемой используемого алгоритма.

При выполнении *заданий лабораторных работ № 4 –6*, связанных с вычислением функций аргумента *X*, приводятся результаты работы программы (табл. 1) для двух-трех различных ненулевых значений аргумента из заданного диапазона его изменения. Привести следует как шестнадцатеричные коды (hex), так и соответствующие им десятичные значения чисел (dec). Для примера в табл. 9 приведены данные для функции $Y = \sin(X)$, вычисленной с 8-разрядной точностью. При защите работы необходимо быть готовым заполнить еще одну строку данной таблицы для любого значения аргумента (из области его допустимых значений), заданного преподавателем.

Таблица 9

Входной аргумент *X*		Точное значение *f(X)* для *X* , dec		Результат *Y*, полученный в программе		Погрешность	
hex	dec	hex	dec	hex	dec	абс., ед.	ε (X), dec
80	0,5	7A	0,4794	79	0,4727	1	0,0140
19	0,1	19	0,0998	16	0,0859	3	0,1393

Процедура защиты всех лабораторных работ предполагает обязательную демонстрацию работы представленной в отчете программы в среде отладчика PICE-51. Перед началом демонстрации заданий лабораторных работ № 1–6 все ячейки внутреннего и внешнего ОЗУ данных дол-

жны содержать нулевые значения (при необходимости производится выход из программы-отладчика). Значения элементов массива для заданий лабораторной работы № 7 задаются студентом так, чтобы наглядно проиллюстрировать корректность работы представленной программы.

Библиографический список

1. www.phyton.ru
2. www.atmel.ru
3. *Зиатдинов С. И., Изранцев В. В., Красильникова О. И*. Системы автоматического управления с микроЭВМ: Учеб. пособие / СПбГУАП. СПб., 1998.
4. *Гребнев В. В*. Однокристальные микроЭВМ семейства AT89 фирмы Atmel. СПб., 1998.
5. Однокристальные микроЭВМ: Справочник / *А. В. Боборыкин, Г. П. Липовецкий* и др. М.: МИКАП. 1994. 400 с.
6. *Карцев М. А.* Арифметика цифровых машин. М.: Наука, 1969. 575 с.
7. *Соловьев Г. М.* Арифметические устройства ЭВМ. М.: Энергия, 1978. 177 с.
8. Микропроцессорные системы автоматического управления / *В. А. Бесекерский* и др. Л.: Машиностроение, 1988. 365 с.
9. *Смит Д. М.* Математическое и цифровое моделирование для инженеров и исследователей: Пер. с англ. М.: Машиностроение, 1980. 271 с.
10. *Корн Г., Корн Т*. Справочник по математике для научных работников и инженеров: Пер. с англ. М.: Наука, 1974. 832 с.
11. *Каган Б. М., Сташин В. В.* Основы проектирования микропроцессорных устройств автоматики. М.: Энергоатомиздат, 1987. 304 с.
12. *Кнут Д.* Искусство программирования для ЭВМ: Пер. с англ. Т. 2. Получисленные алгоритмы. М.: Мир. 1977.

ПРИЛОЖЕНИЕ 1

Таблица

Карта памяти микроконтроллера AT89C51

Диапазон адресов	Условные обозначения	Комментарий
00–07	R0–R7	0-й банк РОН
08–0F	R0–R7	1-й банк РОН
10–17	R0–R7	2-й банк РОН
18–1F	R0–R7	3-й банк РОН
20–2F	–	Ячейки внутреннего ОЗУ с возможность ю побитовой адресации
30–7F	–	Ячейки памяти общего назначения
80	P0	Порт ввода–вывода P0
81	SP	Указатель стека
82	DPL	Младшие 8 разрядов адреса ячейки во внешней памяти данных (окно XData)
83	DPH	Старшие 8 разрядов адреса ячейки во внешней памяти данных (окно XData)
84–86	–	Свободные ячейки памяти
87	PCON	Регистр управления энергопотреблением
88	TCON	Регистр управления/состояния таймеров-счетчиков
89	TMOD	Регистр режимов таймеров-счетчиков
8A	TL0	Младший байт таймера-счетчика T/C0
8B	TL1	Младший байт таймера-счетчика T/C1
8C	TH0	Старший байт таймера-счетчика T/C0
8D	TH1	Старший байт таймера-счетчика T/C1
8E–8F	–	Свободные ячейки памяти
90	P1	Порт ввода–вывода P1
91–97	-	Свободные ячейки памяти
98	SCON	Регистр управления приемопередатчиком последовательного порта
99	SBUF	Буфер приемопередатчика последовательного порта
9A–9F	–	Свободные ячейки памяти
A0	P2	Порт ввода–вывода P2

Окончание таблицы

Диапазон адресов	Условные обозначения	Комментарий
A1–A7	–	Свободные ячейки памяти
A8	IE	Регистр разрешения (масок) прерываний
A9–AF	–	Свободные ячейки памяти
B0	P3	Порт ввода–вывода P3
B1–B7	–	Свободные ячейки памяти
B8	IP	Регистр приоритетов прерываний
B9–CF	–	Свободные ячейки памяти
D0	PSW	Регистр слова состояния (флагов) программы
D1–DF	–	Свободные ячейки памяти
E0	ACC или A	Аккумулятор
E1–EF	–	Свободные ячейки памяти
F0	B	Регистр-расширитель аккумулятора
F1–FF	–	Свободные ячейки памяти

* П р и м е ч а н и е. Ячейки, выделенные серым цветом, позволяют осуществлять побитовую адресацию.

ПРИЛОЖЕНИЕ 2

Система команд микроконтроллеров семейства MCS-51

Таблица

1. Группа команд передачи данных							
№	Название команды	Мнемокод	КОП, bin	КОПh	Б.	Ц.	Операция
1	Пересылка в аккумулятор из регистра (n=0–7)	**MOV A, Rn**	1110 1rrr	E8..EF	1	1	(A)←(Rn)
2	Пересылка в аккумулятор прямоадресуемого байта	**MOV A, ad**	1110 0101	E5	2	1	(A)←(ad)
3	Пересылка в аккумулятор байта из РПД* (i=0,1)	**MOV A, @Ri**	1110 011i	E6, E7	1	1	(A)←((Ri))
4	Загрузка аккумулятора константой	**MOV A, #d**	0111 0100	74	2	1	(A)←#d
5	Пересылка в регистр из аккумулятора	**MOV Rn, A**	1111 1rrr	F8..FF	1	1	(Rn)←(A)
6	Пересылка в регистр прямоадресуемого байта	**MOV Rn, ad**	1010 1rrr	A8..AF	2	2	(Rn)←(ad)
7	Загрузка регистра константой	**MOV Rn, #d**	0111 1rrr	78..7F	2	1	(Rn)←#d
8	Пересылка по прямому адресу из аккумулятора	**MOV ad,A**	1111 0101	F5	2	1	(ad)←(A)
9	Пересылка по прямому адресу из регистра	**MOV ad, Rn**	1000 1rrr	88..8F	2	2	(ad)←(Rn)
10	Пересылка прямоадресуемого байта по прямому адресу	**MOV add, ads**	1000 0101	85	3	2	(add)←(ads)
11	Пересылка байта из РПД по прямому адресу	**MOV ad, @Ri**	1000 011i	86, 87	2	2	(ad)←((Ri))
12	Пересылка по прямому адресу константы	**MOV ad, #d**	0111 0101	75	3	2	(ad)←#d
13	Пересылка в РПД из аккумулятора	**MOV @Ri, A**	1111 011i	F6, F7	1	1	((Ri))←(A)
14	Пересылка в РПД прямоадресуемого байта	**MOV @Ri, ad**	0110 011i	66, 67	2	2	((Ri))←(ad)
15	Пересылка в РПД константы	**MOV @Ri, #d**	0111 011i	76, 77	2	1	((Ri))←#d
16	Загрузка указателя данных	**MOV DPTR, #d16**	1001 0000	90	3	2	(DPTR)←#d16

1. Группа команд передачи данных							
№	Название команды	Мнемокод	КОП, bin	КОПh	Б.	Ц.	Операция
17	Пересылка в аккумулятор байта из ПП*	**MOVC A, @A+DPTR**	1001 0011	93	1	2	(A)←((A)+(DPTR))
18	Пересылка в аккумулятор байта из ПП	**MOVC A, @A+PC**	1000 0011	83	1	2	(PC)←(PC)+1; (A)←((A)+(PC))
19	Пересылка в аккумулятор байта из ВПД*	**MOVX A, @Ri**	1110 001i	E2, E3	1	2	(A)←((Ri))
20	Пересылка в аккумулятор байта из расширенной ВПД	**MOVX A, @DPTR**	1110 0000	E0	1	2	(A)←((DPTR))
21	Пересылка в ВПД из аккумулятора	**MOVX @Ri, A**	1111 001i	F2, F3	1	2	((Ri))←(A)
22	Пересылка в расширенную ВПД из аккумулятора	**MOVX @DPTR, A**	1111 0000	F0	1	2	((DPTR))←(A)
23	Обмен аккумулятора с регистром	**XCH A, Rn**	1100 1rrr	C8..CF	1	1	(A)↔(Rn)
24	Обмен аккумулятора с прямоадресуемым байтом	**XCH A, ad**	1100 0101	C5	2	1	(A)↔(ad)
25	Обмен аккумулятора с байтом из РПД	**XCH A, @Ri**	1100 011i	C6, C7	1	1	(A)↔((Ri))
26	Обмен младшей тетрады аккумулятора с младшей тетрадой байта из РПД	**XCHD A, @Ri**	1101 011i	D6, D7	1	1	$(A_{0-3})\leftrightarrow((Ri)_{0-3})$
27	Загрузка в стек	**PUSH ad**	1100 0000	C0	2	2	(SP)←(SP)+1; ((SP))←(ad);
28	Извлечение из стека	**POP ad**	1101 0000	D0	2	2	(ad)←(SP); (SP)←((SP)–1

Продолжение таблицы

2. Группа команд передачи данных							
№	Название команды	Мнемокод	КОП, bin	КОПh	Б.	Ц.	Операция
29	Логическое И аккумулятора и регистра	**ANL A, Rn**	0101 1rrr	58..5F	1	1	(A)←(A)∧(Rn)
30	Логическое И аккумулятора и прямоадресуемого байта	**ANL A, ad**	0101 0101	55	2	1	(A)←(A)∧(ad)
31	Логическое И аккумулятора и байта из РПД	**ANL A, @Ri**	0101 011i	56, 57	1	1	(A)←(A)∧((Ri))
32	Логическое И аккумулятора и константы	**ANL A, #d**	0101 0100	54	2	1	(A)←(A)∧#d
33	Логическое И прямоадресуемого байта и аккумулятора	**ANL ad, A**	0101 0010	52	2	1	(ad)←(ad)∧(A)
34	Логическое И прямоадресуемого байта и константы	**ANL ad, #d**	0101 0011	53	3	2	(ad)←(ad)∧#d
35	Логическое ИЛИ аккумулятора и регистра	**ORL A, Rn**	0100 1rrr	48..4F	1	1	(A)← (A)∨(Rn)
36	Логическое ИЛИ аккумулятора и прямоадресуемого байта	**ORL A, ad**	0100 0101	45	2	1	(A)← (A)∨(ad)
37	Логическое ИЛИ аккумулятора и байта из РПД	**ORL A, @Ri**	0100 011i	46, 47	1	1	(A)←(A)∨((Ri))
38	Логическое ИЛИ аккумулятора и константы	**ORL A, #d**	0100 0100	44	2	1	(A)←(A)∨#d
39	Логическое ИЛИ прямоадресуемого байта и аккумулятора	**ORL ad, A**	0100 0010	42	2	1	(ad)←(ad)∨(A)
40	Логическое ИЛИ прямоадресуемого байта и константы	**ORL ad, #d**	0100 0011	43	3	2	(ad)←(ad)∨#d
41	Исключающее ИЛИ аккумулятора и регистра	**XRL A, Rn**	0110 1rrr	68..6F	1	1	(A)←(A)⊕(Rn)
42	Исключающее ИЛИ аккумулятора и прямоадресуемого байта	**XRL A, ad**	0110 0101	65	2	1	(A)←(A)⊕(ad)

1. Группа команд передачи данных							
№	Название команды	Мнемокод	КОП, bin	КОПh	Б.	Ц.	Операция
43	Исключающее ИЛИ аккумулятора и байта из РПД	**XRL A, @Ri**	0110 011i	66, 67	1	1	(A)←(A)⊕((Ri))
44	Исключающее ИЛИ аккумулятора и константы	**XRL A, #d**	0110 0100	64	2	1	(A)←(A)⊕#d
45	Исключающее ИЛИ прямоадресуемого байта и аккумулятора	**XRL ad, A**	0110 0010	62	2	1	(ad)←(ad)⊕(A)
46	Исключающее ИЛИ прямоадресуемого байта и константы	**XRL ad, #d**	0110 0011	63	3	2	(ad)←(ad)⊕#d
47	Сброс аккумулятора	**CLR A**	1110 0100	E4	1	1	(A)←0
48	Инверсия аккумулятора	**CPL A**	1111 0100	F4	1	1	$(A) \leftarrow (\overline{A})$
49	Сдвиг аккумулятора влево циклический	**RL A**	0010 0011	23	1	1	(An+1)←(An), для n=0...6, (A0)←(A7)
50	Сдвиг аккумулятора влево через перенос	**RLC A**	0011 0011	33	1	1	(An+1)←(An), для n=0...6, (A0)←(C), (C)←(A7)
51	Сдвиг аккумулятора вправо циклический	**RR A**	0000 0011	03	1	1	(An)(An+1), для n=0...6, (A7)←(A0)
52	Сдвиг аккумулятора вправо через перенос	**RRC A**	0001 0011	13	1	1	(An)←(An+1), для n=0...6, (A7)←(C), (C)←(A0)

Продолжение таблицы

3. Группа команд передачи данных							
№	Название команды	Мнемокод	КОП, bin	КОПh	Б.	Ц.	Операция
53	Обмен местами тетрад в аккумуляторе	**SWAP A**	1100 0100	C4	1	1	(A0–3)←(A4–7)
3. Группа команд передачи данных							
54	Абсолютный переход в полном объеме ПП	**LJMP ad16**	0000 0010	02	3	2	(PC)←ad16
55	Абсолютный переход внутри текущей страницы ПП в 2 Кбайта	**AJMP ad11**	$a_{10}a_9a_8$0 0001	01,21.. ..C1,E1	2	2	(PC)←(PC)+2 (PC0–10)←ad11
56	Короткий относительный переход на rel ячеек (rel s от -128 до 127) ПП внутри страницы в 256 байт	**SJMP rel**	1000 0000	80	2	2	(PC)←(PC)+2 (PC)←(PC)+rel
57	Косвенный относительный переход	**JMP @A+DPTR**	0111 0011	73	1	2	(PC)←(A)+ +(DPTR)
58	Переход, если аккумулятор равен нулю	**JZ rel**	0110 0000	60	2	2	если (A)=0, то (PC)←(PC)+rel, а если (A)≠0, то (PC)←(PC)+2
59	Переход, если аккумулятор не равен нулю	**JNZ rel**	0111 0000	70	2	2	если (A)≠0, то (PC)←(PC)+rel, а если (A)=0, то (PC)←(PC)+2
60	Переход, если перенос равен единице	**JC rel**	0100 0000	40	2	2	если (C)=1, то (PC)←(PC)+rel, а если (C)=0, то (PC)←(PC)+2

3. Группа команд передачи данных							
№	Название команды	Мнемокод	КОП, bin	КОПh	Б.	Ц.	Операция
61	Переход, если перенос равен нулю	**JNC rel**	0101 0000	50	2	2	если (C)=0, то (PC)←(PC)+rel, а если (C)=1, то (PC)←(PC)+2
62	Переход, если бит равен единице	**JB bit, rel**	0010 0000	20	3	2	если (bit)=1, то (PC)←(PC)+rel, а если (bit)=0, то (PC)←(PC)+3
63	Переход, если бит равен нулю	**JNB bit, rel**	0011 0000	30	3	2	если (bit)=0, то (PC)←(PC)+rel, а если (bit)=1, то (PC)←(PC)+3
64	Переход, если бит установлен, с последующим сбросом бита	**JBC bit, rel**	0001 0000	10	3	2	если (bit)=1,то (PC)←(PC)+rel, а также (bit)←0, а если (bit)=0, то (PC)←(PC)+3
65	Декремент регистра и переход, если не нуль	**DJNZ Rn, rel**	1101 1rrr	D8..DF	2	2	(Rn)←(Rn)–1, если Rn≠0, то (PC)←(PC)+rel, а если Rn=0, то (PC)←(PC)+2

3. Группа команд передачи данных							
№	Название команды	Мнемокод	КОП, bin	КОПh	Б.	Ц.	Операция
66	Декремент прямоадресуемого байта и переход, если не нуль	**DJNZ ad, rel**	1101 0101	D5	3	2	(ad)←(ad)–1, если (ad)≠0, то (PC)←(PC)+rel, а если (ad)=0, то (PC)←(PC)+2
67	Сравнение аккумулятора с прямоадресуемым байтом и переход, если не равно	**CJNE A, ad, rel**	1011 0101	B5	3	2	если (A)≠(ad), то (PC)←(PC)+rel, и если при этом (A)<(ad), то (C)←1, иначе (C)←0; а если (A)=(ad), то (PC)←(PC)+3
68	Сравнение аккумулятора с константойи переход, если не равно	**CJNE A, #d, rel**	1011 0100	B5	3	2	если (A)≠#d, то (PC)←(PC)+rel, и если при этом (A)< #d, то (C)←1, иначе (C)←0; а если (A)=#d, то (PC)←(PC)+3

3. Группа команд передачи данных							
№	Название команды	Мнемокод	КОП, bin	КОПh	Б.	Ц.	Операция
69	Сравнение регистра с константой и переход, если не равно	**CJNE Rn, #d, rel**	1011 1rrr	B8..BF	3	2	если (Rn)≠#d, то (PC)←(PC)+rel, и если при этом (Rn)< #d, то (C)←1, иначе (C)←0; а если (Rn)=#d, то (PC)←(PC)+3
70	Сравнение байта в РПД с константой и переход, если не равно	**CJNE @Ri, #d, rel**	1011 011i	B6, B7	3	2	если ((Ri))≠#d, то (PC)←(PC)+rel, и если при этом ((Ri))< #d, то (C)←1, иначе (C)←0; если а, если ((Ri))=#d, то (PC)←(PC)+3
71	Абсолютный вызов подпрограммы в полном объеме ПП	**LCALL ad16**	0001 0010	12	3	2	(PC)←(PC)+3, (SP)←(SP)+1, ((SP))←(PC0–7), (SP)←(SP)+1, ((SP))←(PC8–15), (PC)←ad16

3. Группа команд передачи данных							
№	Название команды	Мнемокод	КОП, bin	КОПh	Б.	Ц.	Операция
72	Абсолютный вызов подпрограммы в пределах страницы в 2 Кбайта	**ACALL ad11**	$a_{10}a_9a_8$ 10001	11,31.... D1,F1	2	2	(PC)←(PC)+2, (SP)←(SP)+1, ((SP))←(PC0–7), (SP)←(SP)+1, ((SP))←(PC8–15), (PC0–10)←ad11
73	Возврат из подпрограммы	**RET**	0010 0010	22	1	2	(PC8–15)←((SP)), (SP)←(SP)–1, (PC0–7)←((SP)), (SP)←(SP)–1
74	Возврат из подпрограммы обработки прерывания	**RETI**	0011 0010	32	1	2	(PC8–15)←((SP)), (SP)←(SP)–1, (PC0–7)←((SP)), (SP)←(SP)–1
75	Холостая команда	**NOP**	0000 0000	00	1	1	(PC)←(PC)+1
4. Группа команд операций с битами							
76	Сброс переноса	**CRL C**	1100 0011	C3	1	1	(C)←0
77	Сброс бита	**CRL bit**	1100 0010	C2	2	1	(bit)←0

4. Группа команд операций с битами							
№	Название команды	Мнемокод	КОП, bin	КОПh	Б.	Ц.	Операция
78	Установка переноса	**SETB C**	1101 0011	D3	1	1	(C)←1
79	Установка бита	**SETB bit**	1101 0010	D2	2	1	(bit)←1
80	Инверсия переноса	**CPL C**	1011 0011	D3	1	1	(C)←($\overline{C}$)
81	Инверсия бита	**CPL bit**	1011 0010	D2	2	1	(bit)←($\overline{bit}$)
82	Логическое И бита и переноса	**ANL C, bit**	1000 0010	82	2	2	(C)←(C)∧(bit)
83	Логическое И инверсии бита и переноса	**ANL C, /bit**	1011 0000	B0	2	2	(C)←(C)∧($\overline{bit}$)
84	Логическое ИЛИ бита и переноса	**ORL C, bit**	0111 0010	72	2	2	(C)←(C)∨(bit)
85	Логическое ИЛИ инверсии бита и переноса	**ORL C, /bit**	1010 0000	A0	2	2	(C)←(C)∨($\overline{bit}$)
86	Пересылка бита в перенос	**MOV C, bit**	1010 0010	A2	2	1	(C)←(bit)
87	Пересылка переноса в бит	**MOV bit, C**	1001 0010	92	2	2	(bit)←(C)
5. Группа команд арифметических операций							
88	Сложение аккумулятора с регистром (n=0...7)	**ADD A, Rn**	0010 1rrr	28..2F	1	1	(A)←(A)+(Rn)
89	Сложение аккумулятора с прямоадресуемым байтом	**ADD A, ad**	0010 0101	25	2	1	(A)←(A)+(ad)
90	Сложение аккумулятора с байтом из РПД (i=0,1)	**ADD A, @Ri**	0010 011i	26, 27	1	1	(A)←(A)+((Ri))
91	Сложение аккумулятора с константой	**ADD A, #d**	0010 0100	24	2	1	(A)←(A)+#d
92	Сложение аккумулятора с регистром и переносом	**ADDC A, Rn**	0011 1rrr	38..3F	1	1	(A)←(A)+(Rn)+ +(C)
93	Сложение аккумулятора с прямоадресным байтом и переносом	**ADDC A, ad**	0011 0101	35	2	1	(A)←(A)+(ad)+ +(C)

4. Группа команд операций с битами							
№	Название команды	Мнемокод	КОП, bin	КОПh	Б.	Ц.	Операция
94	Сложение аккумулятора с байтом из РПД и переносом	**ADDC A, @Ri**	0011 011i	36, 37	1	1	(A)← (A)+((Ri))+ + (C)
95	Сложение аккумулятора с константой и переносом	**ADDC A, #d**	0011 0100	34	2	1	(A)←(A)+#d+ +(C)
96	Десятичная коррекция аккумулятора	**DA A**	1101 0100	D4	1	1	если (A0–3)> >9∨((AC)=1), то (A0–3)←(A0–3)+ +6, иначе (A0– –3)←(A0–3); затем, если (A4–7)> >9∨((C)=1), то (A4–7)←(A4–7)+ +6, иначе (A4–7) ←(A4–7)
97	Вычитание из аккумулятора регистра и заема	**SUBB A, Rn**	1001 1rrr	98..9F	1	1	(A)←(A)–(C)– –(Rn)
98	Вычитание из аккумулятора прямоадресного байта и заема	**SUBB A, ad**	1001 0101	95	2	1	(A)←(A)–(C)– –((ad))
99	Вычитание из аккумулятора байта РПД и заема	**SUBB A, @Ri**	1001 011i	96, 97	1	1	(A)←(A)–(C)– –(Ri)
100	Вычитание из аккумулятора константы и заема	**SUBB A, #d**	1001 0100	94	2	1	(A)←(A)–(C)–#d
101	Инкремент аккумулятора	**INC A**	0000 0100	04	1	1	(A)←(A)+1

4. Группа команд операций с битами							
№	Название команды	Мнемокод	КОП, bin	КОПh	Б.	Ц.	Операция
102	Инкремент регистра	**INC Rn**	0000 1rrr	08..0F	1	1	(Rn)←(Rn)+1
103	Инкремент прямоадресуемого байта	**INC ad**	0000 0101	05	2	1	(ad)←(ad)+1
104	Инкремент байта в РПД	**INC @Ri**	0000 011i	06, 07	1	1	((Ri))←((Ri))+1
105	Инкремент указателя данных	**INC DPTR**	1010 0011	A3	1	2	(DPTR)← ←(DPTR)+1
106	Декремент аккумулятора	**DEC A**	0001 0100	14	1	1	(A)←(A)–1
107	Декремент регистра	**DEC Rn**	0001 1rrr	18..1F	1	1	(Rn)←(Rn)–1
108	Декремент прямоадресуемого байта	**DEC ad**	0001 0101	15	2	1	(ad)←(ad)–1
109	Декремент байта в РПД	**DEC @Ri**	0001 011i	16, 17	1	1	((Ri))←((Ri))–1
110	Умножение аккумулятора на регистр B	**MUL AB**	1010 0100	A4	1	4	(B)(A)←(A)×(B)
111	Деление аккумулятора на регистр B	**DIV AB**	1000 0100	84	1	4	(A).(B)←(A)/(B)