Министерство образования и науки Российской Федерации

Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина

И. Н. Огородников

МИКРОПРОЦЕССОРНАЯ ТЕХНИКА: ПРАКТИЧЕСКИЙ КУРС

*Рекомендовано методическим советом УрФУ
в качестве учебного пособия для студентов
Физико-технологического института, обучающихся
по направлениям подготовки «Ядерные физика и
технологии», «Биомедицинская инженерия» и
«Биотехнические системы и технологии»*

Екатеринбург
УрФУ
2012

УДК 004.382.7+075.8
О39

Рецензенты:
Институт химии твердого тела УрО РАН
(главный научный сотрудник кандидат физико-математических наук, доктор химических наук М. В. Кузнецов);

А. Н. Сафонов, кандидат физико-математических наук, заведующий лабораторией испытаний ООО «Авитек-плюс»

Огородников, И. Н.

О39 Микропроцессорная техника : практический курс : [учеб. пособие] / И. Н. Огородников. – Екатеринбург : УрФУ, 2012. – 137 с.

ISBN 978-5-321-02171-2

Учебное пособие по курсам «Микропроцессорная техника» и «Проектирование импульсной и микропроцессорной техники» нацелено на формирование у студентов практических навыков разработки и программирования микропроцессорных устройств автоматики физических установок, приборов радиационной безопасности человека и окружающей среды, а также различных приборов биофизического и медицинского назначения.

Предназначено для студентов технических специальностей физико-технологического института всех уровней обучения.

Библиогр.: 8 назв. Табл. 13. Рис. 14. Прил. 3.

УДК 004.382.7+075.8

ISBN 978-5-321-02171-2

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Предлагаемое учебное пособие основано на двухсеместровом курсе по основам микропроцессорной техники, читаемом в Уральском федеральном университете для студентов физико-технологического института, специализирующихся в областях электроники и автоматики физических установок, приборов для применения в области радиационной безопасности человека и окружающей среды, защиты от излучений, радиационной экологии, биомедицинской инженерии. Учебное пособие может быть полезно и для студентов других родственных специальностей. Рассмотрены практические вопросы программирования микропроцессорных устройств. Обсуждаются также некоторые вопросы проектирования устройств на их основе. Отметим, что данное учебное пособие касается лишь практической части учебного курса по основам микропроцессорной техники. Оно содержит дополнительный теоретический материал и техническую информацию, которые необходимы студентам при выполнении практических занятий, лабораторного практикума, индивидуального домашнего задания и курсовой работы. Отсюда название – практический курс. Теоретические основы микропроцессорной техники, излагаемые в лекционном курсе, содержатся в учебнике по микропроцессорной технике [1].

Следует отметить, что микропроцессорная техника представляет собой весьма обширную, динамично развивающуюся область знаний. В данное пособие включен лишь ограниченный круг вопросов, выбор и глубина освещения которых продиктованы, в первую очередь, требованиями государственных образовательных стандартов высшего профессионального образования по направлениям «Ядерные физика и технологии», «Биомедицинская инженерия» и «Биотехнические системы и технологии».

В первой главе приведены необходимые сведения о программировании на языке ассемблера ASM-51, знание которого необходимо для практического освоения работы с платформой x51. Данные по основам программирования на языке ассемблера приведены в объеме, достаточном для их практического использования. Рассмотрены основы построения языка: синтаксис, операторы, директивы, ключевые зарезервированные слова. Практическое применение языка ассемблера продемонстрировано на двадцати четырех примерах, которые составляют основу для групповых практических занятий. Приведенные примеры охватывают все группы команд языка ассемблера, а так-

же некоторые типовые операции программирования периферийных устройств, опрос двоичных датчиков, основы применения микроконтроллеров в аналого-цифровом преобразовании.

Вторая глава посвящена индивидуальному домашнему заданию (ИДЗ), выполнением которого завершаются практические занятия по программированию на языке ассемблера. Приведены методические указания по выполнению ИДЗ, требования к структуре, содержанию и оформлению пояснительной записки. Приведен конкретный пример выполненного индивидуального домашнего задания.

Третья глава посвящена лабораторному практикуму по микропроцессорной технике. Приведены необходимые технические данные по лабораторному стенду, программному обеспечению, используемому при выполнении лабораторных работ, порядку выполнения лабораторных заданий и варианты индивидуальных заданий к лабораторному практикуму. Все данные приведены в объеме, достаточном для их практического использования.

Четвертая глава посвящена описанию организации стандартных библиотек Keil μVision, содержащих типовые процедуры для работы с арифметическими и элементарными функциями и с числами в формате с плавающей точкой. Приведено подробное описание технологии вызова стандартных функций из программ пользователя, написанных на языке ассемблера. Обсуждаются механизмы передачи параметров и возврата результата. Помимо технического описания основ работы с библиотечными функциями обсуждается организация стартового механизма запуска программ пользователя в случае использования стенда SDK-1, адаптация стандартных процедур к конкретному микропроцессорному устройству. Стандартные библиотеки будут далее использоваться при выполнении курсовой работы по проектированию импульсной и микропроцессорной техники.

В пятой главе приведены методические рекомендации по выполнению курсовой работы по проектированию импульсной и микропроцессорной техники. Приведены требования к структуре, содержанию и оформлению пояснительной записки. Обсуждается порядок и последовательность действий при выполнении курсовой работы.

Курсовая работа завершает учебный курс по основам микропроцессорной техники. В этой работе студент должен продемонстрировать все полученные в рамках курса знания, умения и практические навыки.

Отметим, что рассматриваемый двухсеместровый курс основ микропроцессорной техники является вводным курсом перед последующим изучением более сложного профессионально-ориентированного курса «Микропроцессорные системы».

1. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АССЕМБЛЕРА

Приведены необходимые данные по программированию на языке ассемблера микроконтроллеров семейства x51. Вначале рассмотрены общие вопросы программирования на языке ассемблера, а затем приведены примеры конкретных программ и разобраны типовые задачи, предлагаемые студентам при выполнении практических занятий.

1.1. Язык ассемблера ASM-51

1.1.1. От исходного текста к машинным кодам

Язык программирования ASM-51 поддерживает модульное написание программ. Графическое изображение процесса подготовки программы на языке программирования ASM-51 приведено на рис 1.1. Обсудим этапы подготовки программы более детально.

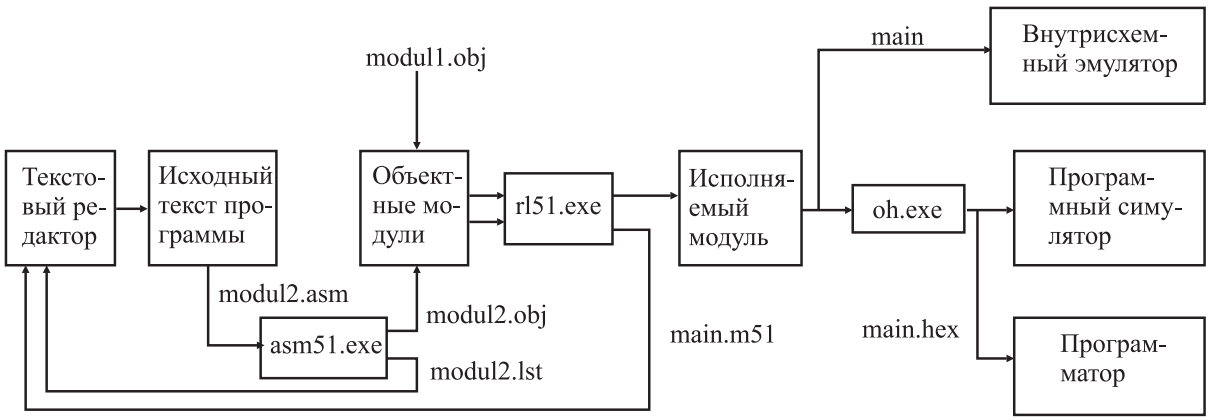

Рис. 1.1. Процесс подготовки программы на языке программирования ASM-51

Исходный модуль – это файл, в котором хранится исходный текст программы (или просто – программа), написанный на языке ассемблера. Для исходного текста программы принято использовать одно из следующих расширений имени файла: `asm`, `a51`, `srs` или `s51`. Исходный текст программы можно написать, используя любой текстовый редактор, сохраняющий результат редактирования в кодировке ASCII.

Исходный модуль подается на вход программы ассемблер, а на выходе при этом получают другой модуль, который называется *объектный модуль*.

Объектный модуль – это файл с промежуточным представлением отдельного модуля программы, получаемый в результате трансляции исходного модуля. Объектный файл содержит в себе бинарный код программы , который может быть объединен с другими объектными файлами при помощи редактора связей (компоновщика) для получения готового исполняемого модуля либо библиотеки. Иными словами, объектный модуль – это программа на машинном языке с неразрешенными внешними ссылками. Получить объектный модуль можно, указав имя исходного модуля программы в качестве параметра программы-транслятора в командной строке или строке командного файла:

```
asm51.exe modul.asm
```

Программа *редактор связей* (компоновщик) позволяет объединять несколько объектных файлов (модулей) в один. Для объединения нескольких модулей в исполняемую программу имена всех модулей передаются в редактор связей в качестве параметров при запуске этой программы. Пример вызова редактора связей из командной строки для объединения трёх модулей:

```
rl51.exe main.obj modul1.obj modul2.obj
```

Результатом работы редактора связей является исполняемый модуль. Имя исполняемого модуля программы по умолчанию совпадает с именем первого объектного файла в списке параметров строки запуска редактора связей. Исполняемый модуль программы записывается в файл без расширения. Формат файла представляет собой двоичное представление каждого байта данных без каких-либо адреса, контрольной суммы или описания формата. Этот формат используют, например, для передачи данных внутрисхемному эмулятору.

При компиляции исходного текста программы транслятор составляет таблицу ссылок на константы, переменные и команды. Если при втором просмотре исходного текста программы, во время которого формируется объектный модуль, транслятор не обнаружит имени переменной или метки в своей таблице, то будет сформировано сообщение об ошибке и объектный модуль не будет сформирован.

Для того чтобы транслятор вместо формирования сообщения об ошибке записал в объектный модуль информацию, необходимую для редактора связей, нужно использовать специальные директивы ссылок на внешние переменные или метки. Обычно эти директивы назы-

ваются PUBLIC (общие) и EXTRN (внешние). Для ссылки на переменную или метку используется директива EXTRN, в которой перечисляются через запятую метки и переменные. Редактор связей должен сначала получить их точное значение из другого модуля, а затем модифицировать все команды, в которых эти метки или переменные используются. Для того чтобы редактор связей мог осуществить связывание модулей в единую программу, переменные и метки, объявленные, по крайней мере, в одном из модулей как EXTRN, в другом модуле должны быть объявлены как доступные для всех модулей при помощи директивы PUBLIC.

Многие микропроцессорный устройства (программаторы, стенды и т.п.) не могут напрямую работать с объектным форматом исполняемого модуля программы, поэтому для загрузки машинного кода в них необходимо преобразовать объектный формат исполняемого модуля в общепринятый гексадецимальный (`hex`) код. При преобразовании форматов вся отладочная информация теряется. Машинный код процессора в гексадецимальном формате называется загрузочным модулем.

Загрузочный модуль программы получают при помощи программы-преобразователя `oh.exe`, передав ей в качестве параметра имя файла исполняемого модуля программы:

```
oh.exe main
```

Полученный файл main.hex загружается в разрабатываемое устройство или программный симулятор для последующей отладки.

1.1.2. Исходный текст программы

Исходный текст программы представляет собой последовательность операторов языка, сгруппированных в сегменты и оформленных в виде файла.

Оператор – это базовая конструкция языка программирования, определяющая действия в программе. В одной строке может быть записан только один оператор. Максимальный размер строки – 255 символов. Признаком конца оператора является символ *возврат каретки*.

Оператор состоит из четырех полей:

```
<метка>     <операция> <операнд>    ; <комментарий>
```

Любое из полей, в том числе и все поля, может отсутствовать. Оператор, в котором все поля отсутствуют, называется пустым опе-

ратором. Он используется для придания большей наглядности тексту программы.

Метка. В поле метки размещается символическое имя ячейки памяти, в которой хранится отмеченная команда или операнд. Метка представляет собой буквенно-цифровую комбинацию, начинающуюся с буквы. Используются только буквы латинского алфавита. Ассемблер ASM-51 допускает использование в метках символа подчеркивания (_). Длина метки не должна превышать 31 символ. Метка всегда завершается двоеточием (:).

Пример оператора, записанного на языке ASM-51:

```
SumDig:     ADD  A,#56    ; Сложить (A) + 56
            mov  R0,A     ;
            mov  A,@R0    ;
```

Если в операторе присутствует только метка, то она помечает ближайший следующий оператор, в котором присутствует инструкция процессора или директива ассемблера. Признаком конца поля метки является символ *двоеточие* (`:`). Однако язык программирования ASM-51 в виде исключения допускает использовать символы интервала как признак конца поля метки.

Пример использования оператора, содержащего только метку:

```
Podprog1:                 ; Помечен след. оператор
            mov  R0,A     ;
            mov  A,@R0    ;
```

Операция. В поле операции записывается мнемоническое обозначение команды MCS51 или директивы ассемблера. Используется строго определенный и ограниченный набор мнемонических кодов. Любой другой набор символов, размещенный в поле операции, воспринимается ассемблером как ошибочный.

Операнды. В этом поле определяются операнды (или операнд), участвующие в операции. Команды ассемблера могут быть без- , одно- или двухоперандными. Операнды разделяются запятой (,).

Операнд может быть задан непосредственно или в виде его адреса (прямого или косвенного). Непосредственный операнд представляется числом (`MOV A,#15`) или символическим именем (`ADDC A,#OPER2`) с обязательным указанием префикса непосредственного операнда (`#`).

Прямой адрес операнда может быть задан мнемоническим обозначением (`IN A, P1`), числом (`INC 40`), символическим именем

```
MOV A, MEMORY
```

Указанием на косвенную адресацию служит префикс (`@`). В командах передачи управления операндом может являться число (`LCALL 0135H`), метка (`JMP LABEL`), косвенный адрес (`JMPP @A`) или выражение (`JMP $-2`, где `$` – текущее содержимое счетчика команд).

Используемые в качестве операндов символические имена и метки должны быть определены, а числа представлены с указанием системы счисления.

Комментарий. Поле комментария может быть использовано программистом для текстового или символьного пояснения логической организации прикладной программы. Поле комментария полностью игнорируется ассемблером, а потому в нем допустимо использовать любые символы. По правилам языка ассемблера поле комментария начинается после точки с запятой (;).

Алфавит языка

Символы исходной программы представляют собой подмножество таблиц символов ASCII для DOS и ANSI для WINDOWS. В исходном тексте программы, написанном на языке программирования ASM-51, допустимо использование символов интервала, букв, знаков цифр.

Символы интервала определяют один или несколько пробелов в предложении исходного модуля. К этим символам относятся *пробел* и *табуляция*.

В качестве букв воспринимаются латинские буквы верхнего и нижнего регистра, цифры (0 1 2 3 4 5 6 7 8 9) и знаки (`# $ ' ( ) * + , - . / : ; < > = ? @`)

Знаки, комбинации знаков (<>, >=, <=), а также символы интервала являются разделителями конструкций языка. До и после знака *минус* разделителя в любой конструкции языка могут быть вставлены символы интервала.

ASCII-символы, не входящие в перечень основных символов алфавита языка, считаются дополнительными. Эти символы могут использоваться для пояснений в исходном тексте программы, а также для определения символьных констант. Из символов формируются идентификаторы и числа.

Идентификаторы

Идентификатор – это символическое обозначение объекта программы. В качестве идентификатора может быть использована любая последовательность букв и цифр. При этом в качестве буквы может быть использована любая буква латинского алфавита, а также вопросительный знак (?) и знак *нижнее подчеркивание* (_). Идентификатор может начинаться только с буквы! В идентификаторах язык программирования ASM-51 различает буквы верхнего и нижнего регистров.

Количество символов в идентификаторе ограничено длиной строки (255 символов). Транслятор различает идентификаторы по первым 31 символам.

Примеры идентификаторов:

`ADD5`, `FFFFH`, `?`, `ALFA_1`.

В языке программирования ASM-51 имеется три категории идентификаторов: *ключевые слова*, *встроенные имена* и *определяемые имена*.

Ключевые слова. Ключевое слово является определяющей частью оператора языка ассемблера. Значения ключевых слов языка ассемблера ASM-51 не могут быть изменены или переопределены в программном модуле каким-либо образом. Ключевому слову не может быть назначено имя – синоним. Ключевые слова могут быть написаны буквами как верхнего, так и нижнего регистров.

В языке ASM-51 имеются следующие категории ключевых слов:

– инструкции;

– директивы;

– вспомогательные слова;

– операции.

Инструкции по форме записи совпадают с мнемоническими обозначениями команд микроконтроллеров семейства MCS51 и совместно с операндами составляют команды микроконтроллера.

Директивы совместно со вспомогательными словами определяют действия в программе, которые должны быть выполнены ассемблером в процессе преобразования исходного текста программы в объектный код. В языке программирования ASM-51 используются следующие основные директивы:

```
BIT,    BSEG,     CODE,  CSEG,   DATA,   DB,
DBIT,   DS,       DSEG,  DW,     END,    EQU,
EXTRN,  IDATA,    ISEG,  NAME,   ORG,    PUBLIC,
RSEG,   SEGMENT,  SET,   USING,  XDATA,  XSEG.
```

Директивы ассемблера не преобразуются в двоичные коды, а потому не могут иметь меток. Исключение составляют директивы резер-

вирования памяти и определения данных (DS, DB, DW). У директив, осуществляющих определение символических имен, в поле метки записывается определяемое символическое имя, после которого двоеточие не ставится.

В качестве символических имен и меток не могут быть использованы мнемокоды команд, директивы и операторы ассемблера, а также мнемонические обозначения регистров и других внутренних блоков микроконтроллера.

Вспомогательные слова, используемые в ассемблере:

```
AT,      BIT,    BITADDRESSABL,  CODE,
DATA,    IDATA,  INBLOCK,        NPAGE,
NUMBER,  PAGE,   UNIT,           XDATA.
```

Ассемблер ASM-51 допускает использование выражений в поле операндов, значения которых вычисляются в процессе трансляции. Перечень операций, использующихся языком ASM-51:

```
AND,  EQ,  GE,   GT,  HIGH,  LE,   LOW,  LT,
MOD,  NE,  NOT,  OR,  SHL,   SHR,  XOR.
```

Выражение представляет собой совокупность символических имен и чисел, связанных операторами ассемблера. Операторы ассемблера обеспечивают выполнение арифметических (`'+'` – сложение, `'-'` – вычитание, `'*'` – умножение, `'/'` – целочисленное деление, MOD – деление по модулю) и логических (OR – ИЛИ, AND – И, XOR – исключающее ИЛИ, NOT – отрицание) операций в формате двухбайтных слов. Например, запись ADD A, #((NOT 13) + 1) эквивалентна записи ADD A,#0F3H и обеспечивает сложение содержимого аккумулятора с числом –13, представленным в дополнительном коде.

Широко используются также операторы LOW и HIGH, позволяющие выделить младший и старший байты двухбайтного операнда.

Встроенные имена. Встроенные имена присвоены адресам регистров специальных функций, адресам флагов специальных функций AR0-AR7, рабочим регистрам R0 – R7 текущего банка регистров, а также аккумулятору A и флагу переноса C.

```
A,   R0,  R1,    R2,  R3,  R4,  R5,
R6,  R7,  DPTR,  PC,  C,   AB,  SP.
```

Определяемые имена. Определяемые имена объявляются пользователем.В языке программирования ASM-51 имеются следующие категории определяемых идентификаторов:

- метки;
- внутренние и внешние переменные адресного типа;
- внутренние и внешние переменные числового типа;
- имена сегментов;
- названия программных модулей.

Представление чисел

В языке программирования ASM-51 используются целые беззнаковые числа, представленные в двоичной, восьмеричной, десятичной и шестнадцатеричной формах записи. Для определения основания системы счисления используется суффикс (буква, следующая за числом): `B` (двоичное число), `Q` или `O` (восьмеричное число), [`D`] (десятичное число) и `H` (шестнадцатеричное число). Для десятичного числа суффикс может отсутствовать. Количество символов в числе ограничено размером строки, однако значение числа определяется по модулю 2^{16} (т.е. диапазон значений числа находится в пределах от 0 до 65535).

Примеры записи чисел:

```
011101b, 1011100B, 735Q, 456o, 256, 0FAH, 0CBH.
```

Часто число используется для представления символов. В этом случае для определения числа можно воспользоваться литеральной константой. Литеральная константа заключается в апострофы (’a’, ’W’):

```
mov SBUF, #'Б'
```

Для записи фраз в памяти программ можно воспользоваться литеральными строками:

```
Nadp: DB 'Ошибка в блоке 5'
```

В этом случае каждый символ заменяется отдельным байтом и запоминается в ПЗУ памяти программ.

Сегменты памяти

Ассемблер поддерживает определенную логическую структуру памяти микроконтроллера для спецификации размещения отдельных частей программы. Основу логической структуры составляет понятие общего (родового) сегмента (generic segment).

Общий сегмент имеет имя, класс памяти (class) и другие атрибуты. Сегменты с одним и тем же именем, но расположенные в разных объектных модулях считаются частями одного и того же общего сегмента и называются частными (partial) сегментами. Объединение частных

сегментов осуществляет редактор связей. Общий сегмент создается директивой SEGMENT, специфицирующей имя и класс сегмента.

```
MYPROG      SEGMENT  CODE    ; Имя = MYPROG,
                             ; класс = CODE
```

Параметр class используется редактором связей для группировки сегментов с одним классом памяти. В ассемблере поддерживаются классы памяти BIT, CODE, DATA, IDATA, XDATA.

Необязательные параметры директивы SEGMENT позволяют задать параметры для настройки редактора связей, характеризующие тип перемещаемости и тип выделяемых ресурсов. Ассемблер поддерживает следующие типы перемещаемости (reloctype):

AT – абсолютный сегмент. Вспомогательной директивой AT задается абсолютный адрес начала сегмента;

BITADDRESSABLE – сегмент, размещаемый в области памяти прямоадресуемых битов. Эта область расположена в РПД от 20H до 2FH. Длина сегмента не может превышать 16 байтов. Такой тип сегмента может быть задан лишь для класса памяти DATA;

INBLOCK – сегмент, размещаемый в 2048-байтовом блоке. Может быть специфицирован только для класса памяти CODE;

INPAGE – сегмент, размещаемый на странице памяти в 256 битов;

OVERLAYABLE – оверлейный сегмент. Занимаемая область памяти может быть общей для нескольких оверлейных сегментов. Есть ряд ограничений на выбор имен для оверлейных сегментов.

Ассемблер ASM-51 поддерживает только один тип выделяемых (alloctype) ресурсов: PAGE предписывает редактору связей разместить стартовый адрес сегмента точно на границе 256-байтовой страницы памяти.

Для выбора одного из ранее созданных общих сегментов используется директива RSEG, которая делает указанный общий сегмент активным.

Пример назначения сегмента стека:

```
STACK       SEGMENT  IDATA    ; Создать сегмент
            RSEG     STACK    ; Активировать сегмент
            DS       10H      ; Под сегмент 16 байтов
```

В ассемблере ASM-51 предопределены пять общих сегментов, соответствующих пяти областям адресного пространства в архитектуре MCS51: сегмент программ (CSEG), сегмент данных в резидентной па-

мяти данных (РПД), доступных по прямому адресу (DSEG), сегмент данных в РПД, доступных по косвенному адресу (ISEG), сегмент внешних данных (XSEG) и сегмент битов (BSEG). Для каждого сегмента поддерживается свой собственный счетчик адреса. Счетчик становится активным, когда соответствующий сегмент активизируется. Когда сегмент активизируется в первый раз, его счетчик адреса равен нулю. Вспомогательная директива AT позволяет задать требуемое начальное значение счетчика сегмента. В начале работы компилятора активным считается по умолчанию сегмент программных адресов (CSEG). Это позволяет при необходимости работать с ассемблером без явного декларирования общего сегмента и его активизации.

Счетчиком адреса текущего сегмента можно управлять при помощи директив ORG, DS, DBIT. Если после переключения сегмента мы возвратимся к ранее используемому сегменту, то значение его счетчика адреса будет восстановлено таким, каким оно было, когда произошел переход к другому сегменту. Переменная $ содержит текущее значение счетчика адреса активного сегмента.

1.1.3. Описание директив ассемблера

Ассемблер ASM-51 имеет набор директив, которые позволяют установить значение меток, зарезервировать и инициализировать память, управлять размещением программы.

Директивы присваивания

EQU Присваивает числовое значение символическому имени. В процессе ассемблирования всюду, где встретится данное символическое имя, оно будет заменено числом или выражением. Символическое имя может быть определено директивой EQU только один раз.

```
<имя>       EQU  <выражение>    ;

PET         EQU  13             ; Пример: PET := 13
MAT         EQU  PET + 4        ; Будет MAT := 17
COUNTER     EQU  R0             ;
ASCII_D     EQU  'D'            ;
```

SET Присваивает новое числовое значение символическому имени. Действует аналогично директиве EQU, но символическое имя, изначально определенное директивой SET, может быть далее по тексту переопределено директивой SET неограниченное число раз.

```
<имя>       SET  <выражение>    ;

PET         SET  3              ; Пример: PET := 3
PET         SET  PET + 2        ; Будет PET := 5
```

BIT Присваивает указанному символическому имени адрес бита. Имя будет иметь тип сегмента BSEG и может быть использовано только в битовых операндах и операторе DB. Карта расположения адресуемых битов в резидентной памяти данных (РПД) и регистрах специальных функций приведена в [1]. Биты с адресами от 0 до 127D расположены в области РПД от 20H до 2FH, с адресами от 128 до 255D – в регистрах специальных функций. Численные значения битовых адресов, превышающие 255D, воспринимаются как ошибка.

```
<имя>       BIT  <выражение>    ;

Flag        BIT  17H.2          ;
Err         BIT  OV             ;
```

CODE Присваивает символическому имени адрес ячейки, расположенной в памяти программ. Имя будет иметь тип сегмента CSEG. Численное значение адреса не должно превосходить 65535D.

```
<имя>       CODE  <выражение>          ;

RESET       CODE  0                    ;
EXTI0       CODE  RESET + (1024/16)    ;
```

DATA Присваивает символическому имени адрес прямоадресуемой ячейки в РПД. Имя будет иметь тип сегмента DSEG и не может быть использовано в операциях с битами и переходах. Численное значение, превышающее 255D, воспринимается как ошибка.

```
<имя>       DATA  <выражение>    ;

CONIN       DATA  SBUF           ; CONIN := 99H
Table       DATA  70H            ; Table := 70H
```

IDATA Присваивает символическому имени адрес ячейки в РПД, адресуемой косвенно. Имя будет иметь тип сегмента ISEG и не может быть использовано в операциях с битами и переходах. Численное значение, превышающее 255D, воспринимается как ошибка.

```
<имя>       IDATA  <выражение>    ;

TOKEN       IDATA  60             ;
BYTE_CNT    IDATA  TOKEN + 1      ;
ADDR        IDATA  TOKEN + 2      ;
```

XDATA Присваивает указанному символическому имени адрес внешней памяти данных. Имя будет иметь тип сегмента XSEG и может быть использовано только в командах DW и загрузки DPTR. Численное значение, превышающее 65535, воспринимается как ошибка.

```
<имя>       XDATA  <выражение>    ;

Date        XDATA  777H           ;
Time        XDATA  Date + 3       ;
```

Директивы управления сегментами

SEGMENT Создает новый родовой сегмент. Обязательные параметры директивы: уникальное имя родового сегмента (Name) и класс (class) памяти сегмента. Необязательные параметры reloctype и alloctype служат для настройки редактора связей.

```
Name SEGMENT class  [reloctype] [alloctype]    ;
```

RSEG Назначает указанный (Name) родовой сегмент текущим. Действует до следующего применения директивы RSEG. Специфицируемый родовой сегмент должен быть ранее создан директивой SEGMENT.

```
            RSEG  Name    ;
```

BSEG Выбирает сегмент битовых адресов, восстанавливая последнее значение битового счетчика адреса. Счетчик адреса этого

сегмента может быть изменен директивами ORG и DBIT. Вспомогательная (необязательная) директива AT позволяет задать значение счетчика при выборе сегмента.

```
[метка:]    BSEG  [AT <выражение>]    ;

            BSEG                      ;
            BSEG  AT 32               ; BSEG := 32D
```

CSEG Выбирает сегмент программных адресов, восстанавливая последнее значение программного счетчика адреса. Этот сегмент устанавливается по умолчанию в начале программы. Счетчик адреса этого сегмента может быть изменен директивами ORG, DS, DB, DW, а также любой ассемблерной командой. Вспомогательная (необязательная) директива AT позволяет программисту задать конкретное значение счетчика при выборе сегмента.

```
[метка:]    CSEG  [AT <выражение>]    ;

            CSEG                      ;
            CSEG  AT 32               ; CSEG := 32D
```

DSEG Выбирает сегмент адресов внутрикристальных данных, восстанавливая последнее значение счетчика адреса внутрикристальных данных. Счетчик адреса этого сегмента может быть изменен директивами ORG и DS. Вспомогательная (необязательная) директива AT позволяет задать значение счетчика при выборе сегмента.

```
[метка:]    DSEG          ;

            DSEG          ;
            DSEG  AT 32   ; счетчик DSEG := 32D
```

ISEG Выбирает сегмент адресов РПД, восстанавливая последнее значение счетчика адреса данных в РПД. Счетчик адреса этого сегмента может быть изменен директивами ORG и DS. Вспомогательная (необязательная) директива AT позволяет задать значение счетчика при выборе сегмента.

```
[метка:]    ISEG          ;

            DSEG          ;
            ISEG  AT 32   ; счетчик ISEG := 32D
```

XSEG Выбирает сегмент адресов внешних данных, восстанавливая последнее значение счетчика адреса внешних данных. Счетчик адреса этого сегмента может быть изменен директивами ORG и DS. Вспомогательная (необязательная) директива AT позволяет задать значение счетчика при выборе сегмента.

```
[метка:]    XSEG          ;

            XSEG          ;
            XSEG  AT 32   ; счетчик XSEG := 32D
```

Директивы резервирования памяти

DS Резервирует память в указанном количестве байтов. Директиву DS можно использовать в сегментах DSEG, ISEG, или XSEG. Счетчик сегмента увеличивает свое значение на указанную величину. Численное значение указанной величины не должно приводить к выходу счетчика за границы сегмента.

```
[метка:]    DS  <выражение>    ;

            DS  10H            ; Резервирует
                               ; 16 битов памяти
STRING:     DS  10H            ; STRING := адрес
                               ; первого байта
```

DBIT Резервирует область в сегменте битов, он может использоваться только в битовом сегменте.

```
[метка:]    DBIT  <выражение>    ;
```

ORG Применяется в любых сегментах. Когда в программе встречается ORG, значение выражения присваивается счетчику адреса текущего сегмента.

```
            ORG  <выражение>    ;

            ORG  800H           ;
```

DB Инициализирует программную память байтовыми величинами и символьными строками. Байтовые величины должны быть разделены запятой.

```
[метка:]    DB  <список_выражений>    ;

            DB  0FFH                  ;
Lab:        DB  1,2,3,5H,7D           ;
Age:        DB  'MARY',27,'JOE',18    ;
```

DW Инициализирует программную память с помощью списка 16-битовых значений. Старший байт помещается по старшему адресу.

```
[метка:]    DW  <список_выражений>    ;

            DW  0EFFH                 ;
Lab:        DB  950,0FFFFH            ;
```

Прочие директивы

USING Назначает текущий банк регистров общего назначения. Директива упрощает выбор текущего банка регистров, но результат ее действия может быть переопределен командами переключения банков регистров. Численное значение номера банка регистров лежит в диапазоне от 0 до 3D.

```
            USING  <выражение>    ;

            USING  0              ; Выбран банк 0
            USING  1+1+1          ; Выбран банк 3
```

NAME Задает имя объектного модуля, генерируемого данной программой. Имя может иметь длину до 40 символов и записывается в объектный файл вместе с соответствующим модулем. В общем случае имя объектного модуля может не совпадать с именем объектного

файла. В исходном файле может быть только одна директива Name. Если эта директива отсутствует, то в качестве имени объектного модуля используется имя исходного файла без расширения.

```
NAME <имя объектного модуля>    ;
```

PUBLIC Перечисляет символьные имена, определенные в данном модуле, которые редактор связей должен сделать доступными для использования в других модулях. В качестве разделителя имен используется запятая. В список не входят имена регистров и сегментов.

```
PUBLIC <список символьных имен>       ;

PUBLIC PUT_CRLF,PUT_STRING,PUT_EOS    ;
PUBLIC ASCBIN, BINASC                 ;
PUBLIC GETTOKEN, GETNUMBER            ;
```

EXTRN Перечисляет символьные имена, определенные в других модулях и декларированные там директивой PUBLIC, в целях использования этих символьных имен в данном модуле. В директиве специфицируется класс памяти (class) декларируемых символов: BIT, CODE, DATA, IDATA, XDATA или NUMBER. Параметр NUMBER означает любой класс памяти. Внутри круглых скобок символьные имена разделяются запятыми.

```
EXTRN   class (<список символьных имен>)    ;

EXTRN   CODE (PUT_CRLF), DATA (BUFFER)      ;
EXTERN  CODE (BINASC, ASCBIN)               ;
EXTRN   NUMBER (TABLE_SIZE)                 ;
```

END Завершает ассемблерную программу.

```
END     ;
```

```
;-----------------------------------------------
;    Пример программы на языке ассемблера ASM-51
;-----------------------------------------------
NAME    SAMPLE

EXTRN   CODE    (PUT_CRLF, PUTSTRING)
PUBLIC  TXTBIT

PROG    SEGMENT CODE
CONST   SEGMENT CODE
VAR1    SEGMENT DATA
BITVAR  SEGMENT BIT
STACK   SEGMENT IDATA

    RSEG  STACK
    DS    10H  ; 16 байтов под стек

    CSEG  AT   0
    USING   0  ; Банк регистров 0
; При включении программа стартует с адреса 0.
    JMP   START

    RSEG  PROG
; Установит указатель стека
START:  MOV   SP,#STACK-1

; Инициализация последовательного интерфейса
; Использовать таймер 1 для задания скорости
; Частота резонатора = 11.059 MHz
    MOV   TMOD,#00100000B  ;C/T = 0, Mode = 2
    MOV   TH1,#0FDH
    SETB  TR1
    MOV   SCON,#01010010B

; Очистка TXTBIT для чтения CODE-памяти
    CLR   TXTBIT

; Это головная программа.
; В цикле выводит текст на консоль
REPEAT:
; печать сообщения
    MOV   DPTR,#TXT
```

```
    CALL  PUTSTRING
    CALL  PUT_CRLF
; повторение
    SJMP  REPEAT
;
    RSEG  CONST
TXT:    DB    'TEST PROGRAM',00H

; Исключительно для примера
    RSEG  VAR1
DUMMY:  DS    21H

; TXTBIT = 0 Чтение текста из CODE-памяти
; TXTBIT = 1 Чтение текста из XDATA-памяти
    RSEG  BITVAR
TXTBIT: DBIT  1
    END
;-----------------------------------------------
```

1.2. Типовые программы

В данном разделе на конкретных примерах рассмотрены вопросы программирования на языке ассемблера ASM-51 и разобраны типовые задачи, предлагаемые студентам при выполнении практических занятий по программированию на языке ассемблера для микроконтроллеров семейства x51. Предлагаемые задания сгруппированы по темам: группы команд, булевский процессор, ожидание внешних событий, обработка сигналов с двоичных датчиков, включая проблему программного подавления переходных процессов, программирование таймеров, а также технология использования микроконтроллеров в аналого-цифровых преобразователях.

1.2.1. Команды передачи данных

Пример 1. Передать содержимое буфера универсального асинхронного приемопередатчика (УАПП) микроконтроллера в резидентной памяти данных (РПД) по косвенному адресу из регистра R0. С точки зрения программирования буфер УАПП представляет собой обычный регистр с символическим именем SBUF, поэтому задача сводится к пересылке данных между регистром SBUF и ячейкой памяти, адрес которой был размещен ранее в регистре R0.

```
        ORG  30H        ;
Begin:  MOV  @R0,SBUF   ; Пересылка байта из
        END             ; УАПП в РПД
```

Пример 2. Загрузить в указатель данных (DPTR) начальный адрес 7F00H массива данных, расположенного во внешней памяти данных (ВПД) микроконтроллера. Особенностью примера является определение константы с символическим именем ARRAY. Далее константа ARRAY может быть использована в программе.

```
        ORG  30H           ;
ARRAY   EQU  7F00H         ; Определение симво-
                           ; лического имени
Begin:  MOV  DPTR,#ARRAY   ; Загрузка указателя
        END                ;
```

Пример 3. Загрузить управляющее слово в регистр TCON управления таймером микроконтроллера. Пример полностью аналогичен предыдущему случаю.

```
        ORG  30H           ;
WORDT   EQU  00000101B     ;
Begin:  MOV  TCON,#WORDT   ; Загрузка упр.слова
        END                ;
```

Пример 4. Сбросить все флаги пользователя, расположенные в области РПД с адресами от 20H до 2FH. Данную область памяти занимает битовый массив из 128 флагов, определяемых пользователем (класс памяти BIT), которые сгруппированы в 16 ячеек по одному байту каждая. Для доступа к флагам можно использовать байтовую адресацию к этим ячейкам, а их обнуление проводить в цикле. Для организации цикла использована команда DJNZ и счетчик цикла в регистре R1. Регистр R0 служит регистром косвенного адреса для доступа к ячейкам памяти.

```
        ORG   30H       ;
Begin:  MOV   R0,#20H   ; Задание нач. адреса
        MOV   R1,#16D   ; Кол-во байтов в области
LOOP:   MOV   @R0,#0    ; Сброс одного байта
        INC   R0        ; Переход к след. байту
        DJNZ  R1,LOOP   ; Организация цикла
        END             ;
```

Пример 5. Запомнить в ВПД содержимое регистров банка 0. Начальный адрес ВПД: 5000H. Переключатель банков расположен в регистре PSW и занимает два бита PSW.4 и PSW.3. Устанавливаем в качестве текущего банка банк 1. Копирование восьми регистров нулевого банка осуществляется в цикле. В качестве указателей данных используются регистры DPTR (указатель приемника) и R1 (указатель источника).

```
        ORG   30H              ;
ADDR    EQU   5000H            ;
Begin:  MOV   PSW,#00001000B   ; Выбор банка 1
        MOV   R0,#8            ; Счетчик цикла
        MOV   DPTR,#ADDR       ; Нач. адреса ВПД
        MOV   R1,#0            ; Нач. адреса банка 0
LOOP:   MOV   A,@R1            ; (A) <-- банк 0
        MOVX  @DPTR,A          ; ВПД <-- (A)
        INC   R1               ;
        INC   DPTR             ;
        DJNZ  R0,LOOP          ;
        END                    ;
```

Пример 6. Осуществить обращение к памяти программ с использованием таблицы готовых решений. Есть специальная команда для работы с таблицами констант, хранящимися в памяти программ – это команда MOVC. Пусть требуется составить подпрограмму вычисления синуса угла X (X меняется в пределах от 0 до 89° с дискретом 1°). Наиболее быстрое вычисление получают путем выборки готового значения синуса из таблицы. Такая таблица для диапазона 0–89° займет 90 байтов при погрешности 0.4%. Каждый байт таблицы будет содержать дробную часть двоичного представления синуса. Исходным параметром для подпрограммы служит значение угла X, находящегося в аккумуляторе.

Отметим, что данная программа обходится без использования регистра DPTR. Инкремент аккумулятора перед обращением к таблице необходим из-за наличия однобайтной команды RET между MOVC и началом таблицы значений синуса. Обращение к таблице осуществляется путем вызова подпрограммы командой CALL SINX. Аргумент *X* передается подпрограмме через аккумулятор. Результат работы подпрограммы возвращается также через аккумулятор. Расчет значений синуса для таблицы проводится заранее любым доступным методом.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Вычисление SIN(X) по таблице значений синуса
;Вход: (A) <-- X в пределах от 0 до 89 градусов
;Выход (A) <-- дробная часть значения синуса
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
       ORG   30H        ;
SINX:  INC   A          ; Инкремент аккумулятора
       MOVC  A,@A+PC    ; Загрузка из таблицы
       RET              ;
;Таблица значений синуса
SINUS: DB    0          ; Sin(0)=0
       DB    00000100B  ; Sin(1)=0.017
       DB    00001001B  ; Sin(2)=0.035
       DB    00001101B  ; Sin(3)=0.052
       DB    00010010B  ; Sin(4)=0.070
       ...              ;
       ...              ;
       DB    11111111B  ; Sin(89)=0.999
       END              ;
```

Пример 7. Осуществить операции со стеком. При записи в стек сначала происходит инкремент SP, а потом запись байта. При аппаратном сбросе указателю стека присваивается значение 07H. Для его переопределения используют команду `MOV SP,#d`, где d – требуемое значение указателя стека. Иными словами, стек может располагаться в любом месте РПД. Составим подпрограмму обработки прерывания. Подпрограмма должна сохранить в стеке содержимое тех регистров, которые сама использует, а перед возвратом – восстановить их значения. Пример подпрограммы обработки прерываний уровня 0. Обратить внимание на порядок записи в стек и извлечения из стека. Нарисовать размещение байтов в стеке в точке CONTR1.

```
        ORG   3             ; Вектор прерывания
Begin:  SJMP  SUBIN0        ; Переход к п/п
        ORG   30H           ; обработки прерываний
Bank1   EQU   08B           ; Банк 1
SUBIN0: PUSH  PSW           ; Сохранение в стеке
        PUSH  ACC           ;
        PUSH  B             ;
        PUSH  DPL           ;
        PUSH  DPH           ;
CONTR1: MOV   PSW,#Bank1    ; Выбор банка 1
        ...                 ; Обработка прерывания
        ...                 ;
        ...                 ;
        POP   DPH           ; Восстановление из стека
        POP   DPL           ;
        POP   B             ;
        POP   ACC           ;
        POP   PSW           ;
        RETI                ; Возврат и разрешение
        END                 ; прерываний
```

1.2.2. Команды арифметических операций

Пример 8. Сложить два двоичных многобайтных целых числа. Оба слагаемых находятся в РПД, начиная с младшего байта. Начальные адреса слагаемых заданы в R0 и R1. Формат слагаемых (количество байтов) задан в R2. Результат необходимо поместить на место первого слагаемого. Перенос между байтами учитывается при помощи команды ADDC. Перед началом цикла бит переноса сбрасывается командой CLR. При сложении беззнаковых чисел на переполнение укажет флаг C, а при сложении чисел со знаком – флаг OV.

```
        ORG   0H     ;
Begin:  CLR   C      ; Сброс бита переноса
LOOP:   MOV   A,@R0  ; Загрузка текущего
                     ; байта первого слагаемого
        ADDC  A,@R1  ; Сложение байтов с учетом C
        MOV   @R0,A  ; Сохранение результата
```

```
        INC   R0        ; Инкремент указателей
        INC   R1        ;
        DJNZ  R2,LOOP   ; Цикл
        END             ;
```

Пример 9. Осуществить умножение. Команда MUL находит произведение двух целых беззнаковых чисел, хранящихся в регистрах (B) и (A). Регистр (B) содержит старший байт. Если после умножения (B)=0, то флаг OV сбрасывается, иначе – устанавливается. Пусть требуется умножить целое двоичное число произвольного формата на константу 173D. Исходное целое число размещается в РПД, адрес младшего байта находится в регистре R0. Формат числа в байтах хранится в R1.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Пример программы умножения
; 01020304H * 173D = 0AE5C09B4H
; 173D=0ADH
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
       ORG   0H         ;
Begin: MOV   60H,#04H   ; Младший байт
       MOV   61H,#03H   ;
       MOV   62H,#02H   ;
       MOV   63H,#01H   ; Старший байт
       MOV   R0,#60H    ; Адрес мл. байта числа
       MOV   R1,#4      ; Кол-во байтов в числе
MULTY: CLR   A          ; Очистка аккумулятора
       XCH   A,@R0      ; Загрузка множимого обменом
LOOP:  MOV   B,#173D    ; Загрузка множителя
       MUL   AB         ; Умножение одного байта
       ADD   A,@R0      ; Частичная сумма
       MOV   @R0,A      ; Запись младшего байта
                        ; частичного произведения
       MOV   A,B        ; Пересылка ст.байта в (A)
       ADDC  A,#0H      ; Сложить ст.байт и бит C
       INC   R0         ; Инкремент указателя байтов
       XCH   A,@R0      ; Формирование байта произв.
                        ; и загрузка байта множимого
       DJNZ  R1,LOOP    ; Цикл
       END              ;
```

Полученное произведение размещается на месте исходного числа и занимает в РПД на один байт больше. Составить блок-схему алгоритма программы и объяснить организацию переноса между байтами при умножении.

Пример 10. Осуществить деление. Команда DIV производит деление содержимого аккумулятора на содержимое регистра B. После деления аккумулятор содержит целую часть частного, а регистр B – остаток. Флаги C и OV сбрасываются. При делении на нуль устанавливается флаг переполнения, а частное остается неопределенным. Команда деления может быть использована для быстрого преобразования двоичных чисел в двоично-десятичный код (ДДК). В качестве примера рассмотрим программу, переводящую беззнаковое целое двоичное число, находящееся в аккумуляторе, в трехразрядный ДДК. Сотни будут размещаться в регистре R0, а десятки и единицы – в аккумуляторе.

```
        ORG   0H      ;
Begin:  MOV   B,#100  ; Вычисление кол-ва сотен
        DIV   AB      ; (A) содержит кол-во сотен
        MOV   R0,A    ; Сохранение числа сотен
        XCH   A,B     ; Пересылка остатка в A
        MOV   B,#10   ; Вычисление кол-ва десятков
        DIV   AB      ; (A) содержит число десятков
                      ; регистр B -- число единиц
        SWAP  A       ; Размещение числа десятков
                      ; в старшей тетраде
        ADD   A,B     ; Подсуммирование числа единиц
        END           ;
```

1.2.3. Команды логических операций

Пример 11. Команды логических операций используют для выполнения операций над отдельными битами. Ниже приведено шесть примеров, иллюстрирующих различные битовые операции над содержимым регистров и портов.

```
        ORG  0H                ;
Begin:  ANL  P2,#10111010B     ; Сброс битов 0,2,6 P2
        ORL  P1,#00001111B     ; (P1.0-P1.3) <-- 1111
        ANL  PSW,#11100111B    ; Выбор банка 0
        XRL  P1,A              ; (P1) <-- (P1) XOR (A)
        XRL  A,#0FH            ; (A) <-- (A) XOR 0FH
        XRL  P0,#11100000B     ; (P0) <-- (P0) XOR #d
        END                    ;
```

Пример 12. Осуществить управление группой битов порта. В РПД находится массив распакованных ДДК. Требуется передать массив внешнему устройству в соответствии с протоколом: P1.4 – вывод строба «Данные готовы» (высокий уровень), P1.5 – ввод строба «Данные приняты» (высокий уровень). Выводимое ДДК (4 бита) число выводится в P1.0 – P1.3. Исходные параметры: начальный адрес массива (R0) и длина массива (R1). Неиспользуемые биты порта P1 сохранять в неизменном виде.

```
        ORG   0H               ;
Begin:  ORL   P1,#00100000B    ; Настройка P1.5 на ввод
LOOP:   ANL   P1,#11101111B    ; Сброс строба P1.4
        JB    P1.5,$           ; Ожидание ответа '0'
        MOV   A,@R0            ; Загрузка байта в (A)
        ANL   A,#00001111B     ; Маскиров. ст.тетрады
        ANL   P1,#11110000B    ; Сброс данных P1.0-P1.3
        ORL   P1,A             ; Выдача данных
        ORL   P1,#00010000B    ; Выдача строба P1.4
        JNB   P1.5,$           ; Ожидание ответа '1'
        INC   R0               ; Инкремент указателя
        DJNZ  R1,LOOP          ; Цикл
        END                    ;
```

1.2.4. Команды операций с битами

Использование булевского процессора дает возможность осуществлять битовые операции без использования логических команд. Это позволяет строить весьма компактные и эффективно работающие программы. Рассмотрим несколько примеров на использование команд булевского процессора.

Пример 13. Выдать содержимое аккумулятора в последовательном коде младшими битами вперед через нулевую линию порта 1, оставляя без изменения остальные линии порта. Время исполнения программы 41 мкс, время передачи бита – 5 мкс, скорость передачи 200 кбит/с.

```
        ORG   30H       ;
Begin:  MOV   R7,#8     ; Счетчик цикла
LOOP:   RRC   A         ; C <-- ACC.0
        MOV   P1.0,C    ; Передача бита
        DJNZ  R7,LOOP   ; Цикл
        END             ;
```

Пример 14. Организовать последовательную передачу данных из аккумулятора на P2.0. Передачу необходимо вести в манчестерском коде (каждый бит передается двумя интервалами: первый интервал содержит инверсию бита, второй интервал – прямое значение). Передача выполняется младшими битами вперед. Длительность одного интервала равна шести машинным циклам (6 мкс), время передачи бита равно 12 мкс, время передачи байта – 96 мкс, скорость 83 кбит/с или 10.4 кбайт/с.

```
        ORG   30H       ;
        MOV   R0,#8     ; Счетчик битов
LOOP:   RRC   A         ; C <-- ACC.0
        CPL   C         ; Инверсия бита
        MOV   P2.0,C    ; Передача инверсного бита
        CPL   C         ; Восстановление прямого бита
        NOP             ; Три команды NOP для выров-
        NOP             ; нивания интервалов
        NOP             ;
        MOV   P2.0,C    ; Передача прямого бита
        DJNZ  R0,LOOP   ; Цикл
        END             ;
```

Пример 15. Вычислить булеву функцию трех переменных X, Y, Z: F(X,Y,Z) = NOT(X & Y + X & NOT(Z)). Переменные X, Y и Z поступают по линиям порта P0.2, P0.1 и P0.0 соответственно. Результат F(X,Y,Z) необходимо вывести на линию P0.3. Сигналы квитирования: сигнал «Данные готовы» (уровень логической единицы) поступает на P0.3 от внешнего устройства. Сигнал «Данные приняты» (уровень ло-

гической единицы) выдает МК на линию P0.4. Флаг F0 используется для временного хранения промежуточного результата.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Тестовая программа реализации логических ф-й
; F(X,Y,Z)=NOT(X & Y + X & NOT(Z)).
; Разрешение -- High; подтверждение -- High
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ORG   0H    ;
Begin:  SJMP  FUNC  ; Переход на начало программы
        ORG   30H   ;


; Настройка P0.0...P0.3 на ввод, установка P0.4 = 0
FUNC:   MOV   P0,#00001111b  ;
WAIT0:  JNB   P0.3,WAIT0     ; ожидание разрешения
        MOV   C,P0.2         ; C = X
        ANL   C,P0.1         ; C = X&Y
; Запись промежуточного результата в битовую память
        MOV   F0,C           ;
        MOV   C,P0.2         ; C = X
        ANL   C,/P0.0        ; C = X& /Z
; Выдача сигнала подтверждения конца ввода
        SETB  P0.4           ;
;Ожидание снятия сигнала разрешения на P0.3
WAIT1:  JB    P0.3,WAIT1     ;
        CLR   P0.4           ;
        ORL   C,F0           ; C = X&Y+X& /Z
        CPL   C              ; C = /(X&Y+X&/Z)
        MOV   P0.5,C         ; выдача результата
        JMP   BEGIN          ; переход на начало
        END                  ;
```

1.2.5. Опрос двоичного датчика

Пример 16. Осуществить ожидание статического сигнала (уровень). Типовая процедура состоит из следующих действий: ввод сигнала с датчика, анализ значения сигнала и передача управления в зависимости от состояния датчика. Конструкция датчика может быть различной. На рис. 1.2, *а* представлен возможный вариант организации двоичного датчика. Он может быть подключен к любой линии

порта. Например, датчик подключен к линии P1.3 (рис. 1.2). Процедура опроса основана на использовании команды JNB (ожидание единичного уровня) или JB (ожидание нулевого уровня).

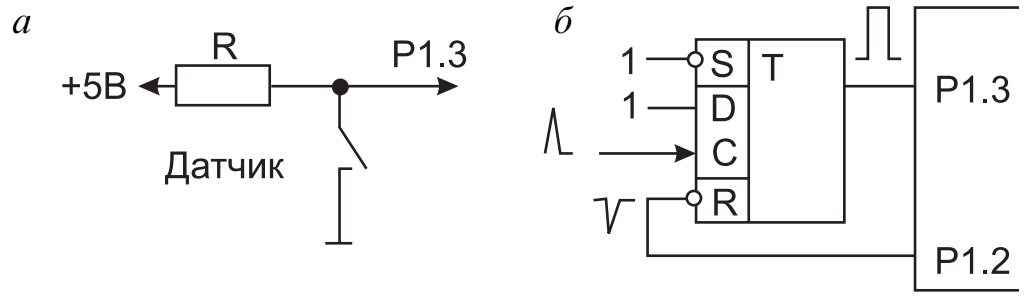

Рис. 1.2. Принципиальные схемы простого двоичного датчика (*а*) и двоичного датчика с флажковым триггером (*б*)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Процедура ожидания единичного уровня
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ORG   30H     ;
Begin:  SETB  P1.3    ; Программируем P1.3 на ввод
        JNB   P1.3,$  ; Ожидание ед. уровня с датчика
        END           ;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Процедура ожидания нулевого уровня
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ORG   30H     ;
Begin:  SETB  P1.3    ; Программируем P1.3 на ввод
        JB    P1.3,$  ; Ожидание нул. уровня с датчика
        END           ;
```

Пример 17. Осуществить ожидание импульсного сигнала (1–0–1), а не уровня. Микроконтроллер должен обнаружить не только факт появления, но и факт окончания сигнала. Процедура ожидания импульсного сигнала представляет собой последовательную комбинацию двух процедур ожидания нулевого и единичного уровней.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Импульсный сигнал 1--0--1
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ORG   30H      ;
Begin:  SETB  P1.3     ; Программируем на ввод
        JB    P1.3,$   ; Ожидание P1.3 <-- 0
        JNB   P1.3,$   ; Ожидание P1.3 <-- 1
        END            ;
```

Минимальная длительность импульса ограничена временем выполнения петли ожидания. Обычно это 5–10 мкс.

Более короткие импульсы надо фиксировать на внешнем флажковом *D*-триггере: короткий импульс поступает на вход C при D=1. На вход микроконтроллера поступает уровень с флажкового триггера. Сброс триггера осуществляют программным путем. Длительность импульса снизу будет ограничена только быстродействием триггера. Пример: триггер подключен к P1.3, а сброс триггера осуществим через линию P1.2 (рис. 1.2, *б*).

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Процедура ожидания единичного уровня на
; флажковом триггере
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ORG   30H      ;
Reset   BIT   P1.2     ; Определение битовых
Input   BIT   P1.3     ; констант
Begin:  SETB  Reset    ; Исходное положение
        SETB  Input    ; Программирование на ввод
        JNB   Input,$  ; Ожидание 1 с триггера
        CLR   Reset    ; Вывод 0 по P1.2
        SETB  Reset    ; Вывод 1 по P1.2
        END            ;
```

Пример 18. Организовать устранение дребезга контактов. Метод подсчета заданного числа совпадающих значений сигнала. Проверяют *N*-раз подряд, что контакт замкнут. Если хоть один раз при этом продетектировали, что он разомкнут, то начинают проверку снова.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Процедура определения, что контакт разомкнут
; (=1) методом многократной проверки
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ORG   30H          ;
Begin:                     ;
DBNC:   MOV   R3,#20       ; Инициализация счетчика
DBNC1:  JNB   P3.4,DBNC    ; Если =0, то начать заново
        DJNZ  R3,DBNC1     ; N раз проверить, что =1
        END                ;
```

В методе временной задержки программа, обнаружив размыкание контакта (=1), запрещает доступ к нему на время задержки. Время задержки заведомо больше длительности переходных процессов в датчике (1–10 мс). Задержку формирует подпрограмма DELAY.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Процедура определения, что контакт разомкнут
; (=1) методом задержки
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ORG   30H          ;
Begin:                     ;
DBNC:   JNB   P3.4,DBNC    ; Если =0, то начать заново
        CALL  DELAY        ; Вызов п/п задержки
        END                ;
```

Пример 19. Организовать подсчет числа импульсов. Пусть импульсы поступают на вход T1 таймера/счетчика T/C1 (режим 1) и их число не превышает 65535. Логика работы управляющих цепей счетчика представлена на рис. 1.3. По рисунку выбираем GATE=0, $\overline{\text{INT1}}$=1 (исходный уровень) и C/$\overline{\text{T}}$=1. В этом случае старт счетчика будет осуществляться сигналом TR1=1, а стоп – TR1=0. Сигналы GATE, C/$\overline{\text{T}}$ и выбор режима задаются состоянием битов регистра TMOD (рис. 1.3). Сигнал TR1 задается состоянием бита TCON.6.

Рассмотрим три варианта организации программы. Для настройки счетчика 1 выбрано управляющее слово 50H = 01010000B. Это соответствует значениям старшей тетрады TMOD: TMOD.7 (GATE) = 0; TMOD.6 (C/T) = 1 (счетчик внешних событий); TMOD.5=0, TMOD.4 = 1 (режим 1).

1. Подсчет числа импульсов между внешними событиями. В каче-

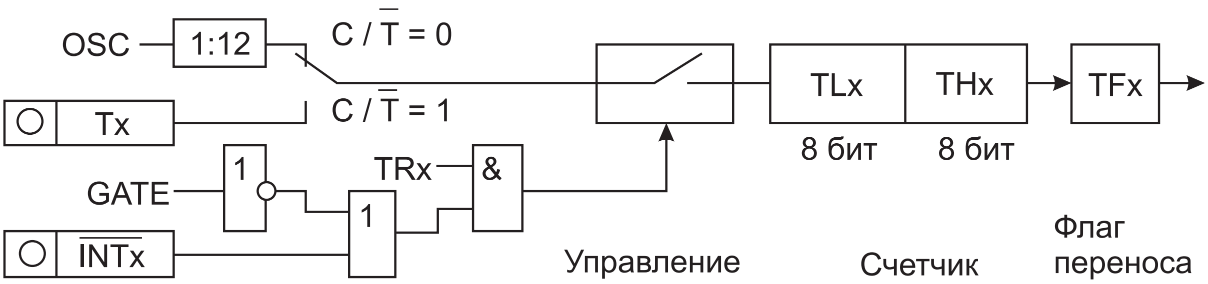

Рис. 1.3. Логика управления каналом таймера/счетчика T/Cx в первом режиме. Параметр $x = 1$ для T/C1 и $x = 0$ для T/C0

стве внешних событий используем сигнал на ножке P3.5 (T1), а сигнал управления «Вкл. конвейер» подается на ножку P3.4: 0 – пуск, 1 – стоп. Состояние ножки P3.4 будем опрашивать программно. Выбираем первый режим работы счетчика-таймера. Ниже приведен текст программы, реализующий поставленную задачу.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Импульсы поступают на ножку T1.
; 'Вкл. конвейер' (=0) поступает на ножку P3.4
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
       ORG   30H        ;
Begin: MOV   TMOD,#50H  ; Настройка счетчика 1
       CLR   A          ; Очистка (A)
       MOV   TH1,A      ; Сброс счетчика
       MOV   TL1,A      ;
       JB    P3.4,$     ; Ожидание вкл. конвейера
       SETB  TCON.6     ; Пуск счетчика 1
       JNB   P3.4,$     ; Ожидание выкл. конвейера
       CLR   TCON.6     ; Останов счетчика
       MOV   B,TH1      ; Сохранение результата
       MOV   A,TL1      ; подсчета
       END              ;
```

2. Подсчет числа импульсов за заданный промежуток времени. В этом случае пуск счетчика происходит при запуске программы. Особенностью программы является использование дополнительной подпрограммы временной задержки с именем DELAY. Отличие от предыдущей программы – лишь в организации старта и стопа счетчика. Останов счетчика определяется программной временной задержкой, которую формирует подпрограмма DELAY. Ниже приведены тексты основной программы и подпрограммы DELAY.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Импульсы поступают на ножку T1.
; Программная задержка
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ORG   30H          ;
Begin:  MOV   TMOD,#50H    ; Настройка счетчика 1
        CLR   A            ; Очистка (A)
        MOV   TH1,A        ; Сброс счетчика
        MOV   TL1,A        ;
        SETB  TCON.6       ; Пуск счетчика 1
        CALL  DELAY        ; Задание промежутка времени
        CLR   TCON.6       ; Останов счетчика
        MOV   B,TH1        ; Сохранение результата
        MOV   A,TL1        ; подсчета
        SJMP  $            ; Конец основной программы
;Подпрограмма задержки на заданное время
DELAY:  MOV   R1,#099H     ; Задержка около 150 мкс
        DJNZ  R1,$         ; Организация цикла
        RET                ; Возврат из подпрограммы
        END                ;
```

Принцип работы подпрограммы DELAY ясен из приведенного текста. Для формирования временной задержки используется оператор цикла DJNZ с однобайтным счетчиком цикла. Для увеличения времени задержки возможно использование как программных циклов с двухбайтным счетчиком цикла, так и нескольких вложенных программных циклов с однобайтными счетчиками цикла.

3. В данном случае используется полностью аппаратный метод подсчета. Временную задержку формируем таймером T/C0. Для таймера T/C0 выбираем режим 1, GATE=0, $\overline{\text{INT0}}$=1, C/$\overline{\text{T}}$=0.

При тактовой частоте 12 МГц изменение состояния таймера будет происходить каждую микросекунду. Для заданного времени измерения, например 10 мс, необходимо отсчитать 10000 импульсов. Поскольку счет ведется до переполнения таймера, то исходное число в таймере должно быть дополнением до состояния «все единицы» +1. Дополнительная единица обусловлена тем, что флаг переполнения таймера устанавливается не при переходе в состояние «все единицы», а при следующем импульсе, т.е. при переходе в состояние «все нули». Остальное – как в предыдущем примере.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Импульсы поступают на ножку T1.
; Интервал счета задается таймером T0.
; Тактовая частота F = 12 МГц.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
       ORG   30H              ;
;Константа TIME для отсчета интервала времени 10 мс
TIME   EQU   NOT(10000)+1     ;
;Маска для одновременного пуска счетчика и таймера
Start  EQU   01010000B        ;
;Маска для останова счетчика и таймера
Stop   EQU   NOT(Start)       ;
Begin: MOV   TMOD,#50H        ; Настройка счетчика
       CLR   A                ; Очистка (A)
       MOV   TH1,A            ; Сброс счетчика
       MOV   TL1,A            ;
       MOV   TH0,#HIGH(TIME)  ; Загрузка в TC0
       MOV   TL0,#LOW(TIME)   ; константы TIME
;Одновременный пуск счетчика и таймера
       ORL   TCON,#Start      ;
WAIT:  JBC   TCON.5,EXIT      ; Ожидание флага TF0,
                              ; Переход и сброс TF0
       SJMP  WAIT             ; Организация цикла
;Одновременный останов счетчика и таймера
EXIT:  ANL   TCON,#Stop       ;
       MOV   B,TH1            ; Результат счета
       MOV   A,TL1            ; в регистрах (BA)
       END                    ;
```

В данной программе использованы зарезервированные слова ассемблера ASM-51.

NOT – означает дополнение до единицы (инверсию) операнда, указанного в круглых скобках;

HIGH, LOW – означают операции выделения старшего или младшего байта операнда, указанного в круглых скобках.

Отметим, что операции NOT, HIGH и LOW выполняет программа ассемблер, а не микроконтроллер. Итоговый машинный код содержит только результаты выполнения этих операций, которые для микроконтроллера являются константами.

Пример 20. Произвести опрос группы двоичных датчиков. Сравнить входные сигналы порта P0 с эталонным кодом в аккумуляторе. Запрограммировать ожидание заданной комбинации сигналов группы датчиков. Решение поставленной задачи основано на использовании особенности команды CJNE. Данная команда проводит сравнение однобайтовых значений первого и второго операндов и осуществляет переход по указанному адресу при неравенстве этих операндов.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Ожидание заданного кода MYCODE в P0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ORG   30H          ;
MYCODE  EQU   11001110B    ; Эталонный код
Begin:                     ;
WTCODE: MOV   A,#MYCODE    ; Загрузка кода
        MOV   P0,#0FFH     ; Все линии P0 на ввод
        CJNE  A,P0,$       ; Ожидание кода
        END                ;
```

Рассмотрим другую программу. Передача управления одной из семи подпрограмм в зависимости от кода на P0.0 – P0.2.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Передача управления одной из семи программ
; в зависимости от кода на P0.0 -- P0.2
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ORG   0H                    ;
        SJMP  Begin                 ;
        ORG   30H                   ;
Begin:                              ;
        MOV   DPH,#HIGH(PROG0)      ; ст. байт адреса
        MOV   P0,#0FFH              ; прогр. на ввод
        MOV   A,P0                  ; чтение порта
        ANL   A,#07H                ; маскирование битов
        ADD   A,#5                  ; 5 доп. байтов
        MOVC  A,@A+PC               ;
        MOV   DPL,A                 ; 2Б
        MOV   A,#0                  ; 2Б
        JMP   @A+DPTR               ; 1Б
;-------------------------------------
```

```
;--------------------------------------
BASE:   DB   LOW PROG0  ;
        DB   LOW PROG1  ;
        DB   LOW PROG2  ;
        DB   LOW PROG3  ;
        DB   LOW PROG4  ;
        DB   LOW PROG5  ;
        DB   LOW PROG6  ;
        DB   LOW PROG7  ;
;--------------------------------------
        ORG  100H       ;
PROG0:  NOP             ;
;--------------------------------------
        ORG  110H       ;
PROG1:  NOP             ;
;--------------------------------------
        ORG  120H       ;
PROG2:  NOP             ;
;--------------------------------------
        ORG  130H       ;
PROG3:  NOP             ;
;--------------------------------------
        ORG  140H       ;
PROG4:  NOP             ;
;--------------------------------------
        ORG  150H       ;
PROG5:  NOP             ;
;--------------------------------------
        ORG  160H       ;
PROG6:  NOP             ;
;--------------------------------------
        ORG  170H       ;
PROG7:  NOP             ;
;--------------------------------------
        END             ;
```

В данной программе для организации передачи управления использована команда перехода по косвенному адресу JMP @A+DPTR. Отметим, что в системе команд микроконтроллера x51 это единственная команда, допускающая переменную в адресе перехода.

1.2.6. Формирование временных задержек

Пример 21. Осуществить формирование временной задержки малой длительности на основе таймера. Логика работы управляющих цепей таймера показана на рис. 1.3. Старт таймера происходит при запуске программы, а останов – в результате выполнения подпрограммы обслуживания прерывания по переполнению счетчика-таймера T/C0.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Формирование временной задержки на основе
; таймера T/C0. Рабочий диапазон: 1--65536 мкс
; В примере формируем задержку в 50 мс
; Тактовая частота F = 12 МГц.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ORG   0H          ;
TIME    EQU   50000       ; Величина задержки в мкс
        SJMP  Begin       ; Стартовый адрес программы
;.........................................
        ORG   0BH         ; Вектор прерываний T/C0
        CLR   TCON.4      ; Останов T/C0
        RETI              ;
;.........................................
        ORG   30H         ;
Begin:  MOV   TMOD,#01H   ; Режим 1 для T/C0
        MOV   TL0,#LOW(NOT(TIME) + 1)
        MOV   TH0,#HIGH(NOT(TIME) + 1)
        SETB  TCON.4      ; Старт T/C0
        SETB  IE.1        ; Разрешение прерываний T/C0
        SETB  PCON.0      ; Режим холостого хода
NEXT:   ...               ;
        END               ;
```

Пример 22. Измерить временной интервал импульса положительной полярности, подаваемого на вход $\overline{\text{INT0}}$. Метод измерения – заполнение временного интервала импульсами с известной частотой. Число импульсов в счетчике будет пропорционально длительности временного интервала. Верхний предел измерения 65536 мкс, а максимальная погрешность 1 мкс. Логика работы управляющих цепей таймера показана на рис. 1.3. Для выбранного режима нужно установить GATE=1.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Измерение временного интервала импульса
; положительной полярности, подаваемого
; на вход INT0.
; Результат -- количество импульсов в T/C0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ORG   0           ;
        SJMP  Begin       ;
;.....................................
        ORG   30H         ;
Begin:  MOV   TMOD,#09H   ; Настройка T/C0
                          ; Режим 1, GATE=1
        MOV   A,#0        ;
        MOV   TH0,A       ; Сброс T/C0
        MOV   TL0,A       ;
        SETB  TCON.4      ; Старт T/C0
        JNB   P3.2,$      ;
        JB    P3.2,$      ;
        CLR   TCON.4      ; Стоп T/C0
        MOV   B,TH0       ; Сохранение
        MOV   A,TL0       ; результата
        END               ;
```

1.2.7. Аналого-цифровое преобразование

Пример 23. Реализовать аппаратно-программный аналого-цифровой преобразователь (АЦП) с поразрядным взвешиванием (метод последовательных приближений, метод побитного уравновешивания). Данный метод преобразования характеризуется средним быстродействием (несколько медленнее чисто аппаратного преобразования), средней стоимостью (использует дополнительные компоненты, такие как цифро-аналоговый преобразователь (ЦАП) и компаратор). На рис. 1.4 приведена схема реализации АЦП с поразрядным взвешиванием на основе микроконтроллера x51. Алгоритм работы преобразователя заключается в следующем. Микроконтроллер готовит и выдает пробный код, по которому ЦАП формирует опорный уровень $U_{оп}$ и строб готовности к взвешиванию, подаваемый на P1.6. Компаратор сравнивает U_x и опорный уровень. Сравнение проводится для каждого бита кода по отдельности, начиная со старшего бита. Для этого в старший

бит выдается единица, если компаратор показал на выходе логический ноль ($U_{оп} < U_x$), то эта единица в данном разряде сохраняется, а если данный компаратор показал на выходе логическую единицу ($U_{оп} > U_x$), то в данном разряде записывают ноль. После этого переходят к взвешиванию следующего более младшего разряда. Этот процесс продолжают до тех пор, пока не достигнут самого младшего разряда. Сколько разрядов в коде, столько и взвешиваний необходимо произвести.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; АЦП с поразрядным взвешиванием с 8-битным ЦАП и
; компаратором в цепи обратной связи.
; R3 -- бегущая единица для взвешивания;
; R4 -- цифровой эквивалент изм.величины;
; R5 -- счетчик битов;
; ЦАП подключен к P0;
; Строб 'Готов' от ЦАП 0-1-0 на линию P1.6;
; Компаратор подключен к P1.7: 1 == U(ADC) > Ux;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ORG   0H          ;
        SJMP  Begin       ;
        ORG   30H         ;
Begin:                    ;
;Настройка P1.7 и P1.6 на ввод
        MOV   P1,#11000000B
;Бегущая единица для взвешивания
        MOV   R3,#1       ;
        MOV   R4,#0       ; Очистка регистра кода
        MOV   R5,#8D      ; Счетчик битов
LOOP:   MOV   A,R3        ; Формирование ед. во
        RR    A           ; взвешиваемом разряде
        MOV   R3,A        ; сохранение маски
;Пробный код = старый код + маска
        ORL   A,R4        ;
        MOV   P0,A        ; Вывод пробн. кода
        JNB   P1.6,$      ; Ожидание 1
        JB    P1.6,$      ; Ожидание 0
        JB    P1.7,OMIT   ; Взвешивание
        MOV   R4,A        ; ст. код <-- пробн.код
OMIT:   DJNZ  R5,LOOP     ;
        END               ;
```

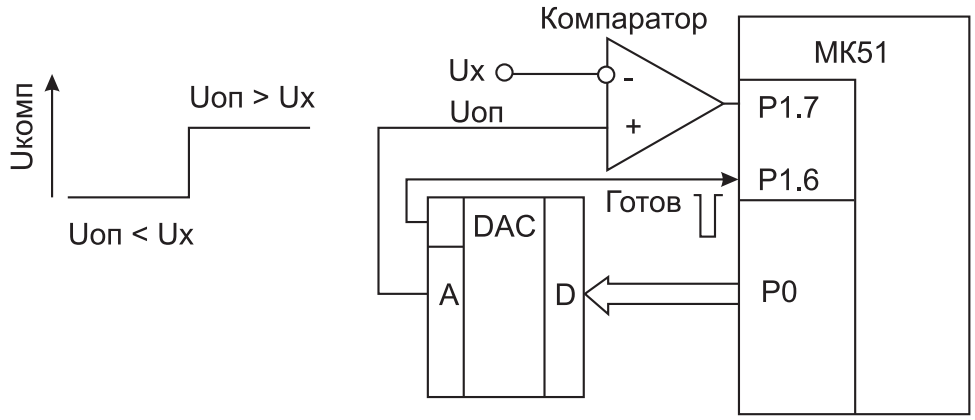

Рис. 1.4. Состояния компаратора и структурная схема АЦП

Пример 24. Программно-аппаратурный метод двойного интегрирования – это самый дешевый, но самый медленный способ преобразования, который может обеспечить очень высокую точность преобразования. Дополнительное оборудование: операционный усилитель, компаратор и аналоговый коммутатор (AMUX) на два входа. Первоначально на вход интегратора подают положительный уровень $E_{оп}$. На выходе интегратора имеем постоянный минус. Затем подаем неизвестный отрицательный уровень U_x – на выходе интегратора будет линейно-изменяющееся напряжение (ЛИН), растущее от отрицательного исходного уровня. Момент пересечения нулевого уровня ЛИНом считается t_0. В момент t_1 на вход интегратора подают снова положительное $E_{оп}$, тогда ЛИН уменьшается и в момент t_2 пересекает нулевой уровень. Временной интервал $T_1 = t_1 - t_0$ задается фиксированным и отсчитывается таймером, а интервал $T_2 = t_2 - t_0$ измеряют таймером. Измеряемое напряжение вычисляют по формуле $U_x = E_{оп} \cdot (T_2 / T_1)$.

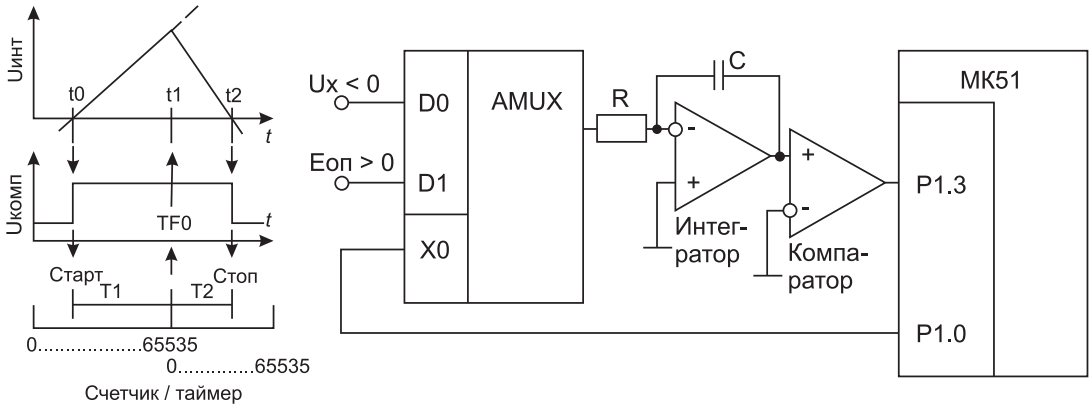

Рис. 1.5. Временная диаграмма работы и структурная схема АЦП

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; АЦП по методу двойного интегрирования
; Выход компаратора подключен к P1.3
; Управление AMUX -- к P1.0
; (1 -- выбор Eоп; 0 -- выбор Uх)
; Интервал T1 -- задает таймер T/C0
; Интервал T2 -- измеряет таймер T/C0
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ORG   0H          ;
        SJMP  Begin       ;
        ORG   30H         ;
Begin:  MOV   TMOD,#01H   ; T/C0 в режиме 1 (16 бит)
        MOV   TH0,#HIGH(NOT(T1) +1)
        MOV   TL0,#LOW(NOT(T1) +1)
        SETB  P1.1        ; P1.3 (выход комп.) на ввод
        SETB  P1.0        ; P1.0 (упр. MUX) Выбор Eоп
;Ожидание смены знака (0) компаратора
        JB    P1.3,$      ;
        CLR   P1.0        ; Выбор Ux на вх. MUX
;Ожидание смены знака (1) компаратора (момент t0)
        JNB   P1.3,$      ;
        SETB  TCON.4      ; Старт T/C0
;Ожидание переполнения T/C0 (момент t1)
        JNB   TCON.5,$    ; Ожидание момента t1
        SETB  P1.0        ; Выбор Eоп на входе MUX
;Ожидание смены знака (0) компаратора (момент t2)
        JB    P1.3,$      ;
        CLR   TCON.4      ; Стоп T/C0
        CLR   TCON.5      ; Сброс ϕлага TF0
        MOV   B,TH0       ; Ст. байт T2
        MOV   A,TL0       ; Мл. байт T2
        END               ;
```

Программа позволяет сформировать 16-битный код, эквивалентный входному сигналу в диапазоне от 0 до -10 В. Это очень высокая точность (около 0.002 % относит.погрешн.). Максимальное время преобразования составляет 2×65.535 мс.

2. ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

Для закрепления теоретического материала, изложенного в гл. 1, студентам предлагается выполнить индивидуальное домашнее задание. Выполнением этого задания завершается курс практических занятий по основам программирования на языке ассемблера ASM-51. В данном разделе приведены методические указания по выполнению индивидуального домашнего задания по дисциплине «Микропроцессорная техника», сформулированы требования к содержанию и оформлению пояснительной записки, приведен пример индивидуального домашнего задания. Выполнению индивидуального домашнего задания должно предшествовать аудиторное и самостоятельное изучение теоретического курса, а также выполнение аудиторных практических заданий, предусмотренных учебным планом для дисциплины «Микропроцессорная техника».

2.1. Общие положения

Индивидуальное домашнее задание (ИДЗ) – это учебный документ, выполняемый студентами по учебному плану на промежуточных этапах обучения. Процесс выполнения индивидуального домашнего задания является одним из видов самостоятельной работы студентов, выполняемой в течение времени, отводимом на изучение данной дисциплины.

Выполнение индивидуального домашнего задания имеет целью развить навыки самостоятельной работы, проверить способность студента самостоятельно творчески применять на практике знания по элементарному программированию микропроцессорных устройств, полученные при изучении базовой дисциплины «Микропроцессорная техника». В процессе выполнения задания студенты приобретают навыки по составлению алгоритма заданных операций, доведения поставленной задачи до конкретного программного решения и документированию программного продукта.

Индивидуальное домашнее задание каждому студенту выдается преподавателем. Варианты заданий при этом не повторяются. Задание предусматривает решение какой-либо типовой задачи по программированию микропроцессорных устройств для обработки информации.

Исходные данные для ИДЗ задаются в виде, стимулирующем поиск прогрессивных алгоритмических решений и получение высоких показателей при обработке потоков данных. При формулировке содержания пояснительной записки основной акцент делается на получение студентом навыков в выборе алгоритма решения задач, возникающих перед разработчиком микропроцессорных устройств.

2.2. Порядок выполнения ИДЗ

Приступить к выполнению индивидуального задания студент должен сразу же после получения задания. Для качественного и своевременного выполнения индивидуального домашнего задания рекомендуется придерживаться следующей последовательности:

1. Ознакомиться с выданным преподавателем заданием и провести его анализ. При необходимости следует сразу же уточнить у преподавателя формулировку задания.

2. Изучить поставленный вопрос по материалам теоретического курса [1, 2] и практических занятий (гл. 1), при необходимости – по специальной технической литературе.

3. Составить эскиз схемы обработки потоков данных и сигналов, дать ее краткое описание. Составить карту памяти – распределение переменных по регистрам и ячейкам памяти.

4. Разработать блок-схему алгоритма программы и привести ее подробное описание и обосновать выбранные алгоритмические решения.

5. В соответствии с блок-схемой алгоритма и принципами структурного программирования приступить к проектированию на языке ассемблера подпрограммы, реализующей задание. Текст подпрограммы следует снабжать подробными комментариями. Составить описание технологии передачи подпрограмме исходных данных и получения возвращаемых подпрограммой обработанных данных.

6. Провести расчет требуемых подпрограмме ресурсов (количество занимаемых ячеек в памяти программ (длина программы), количество ячеек в памяти данных, необходимых для хранения переменных, требования к стеку и т.п.).

7. Оценить время выполнения подпрограммы в машинных тактах и единицах измерения времени (секундах, миллисекундах и т.п.). Тактовую частоту при таких оценках принимать равной 12 МГц. Для подпрограмм, осуществляющих непрерывную обработку потоков данных, оценить характерное время какой-либо одной итерации (функционально законченной обработки какой-либо порции данных).

8. Разработать тестовый пример, содержащий набор тестовых данных и заранее рассчитанных ожидаемых результатов, а также текст тестовой программы, осуществляющей размещение этих тестовых данных, вызов разработанной подпрограммы и получение результата.

9. Приступить к оформлению пояснительной записки индивидуального домашнего задания.

2.3. Требования к пояснительной записке

Выполненное индивидуальное домашнее задание состоит из пояснительной записки. Студент несет ответственность за правильность вычислений, работоспособность алгоритма и программного кода, качество оформления пояснительной записки, за своевременное выполнение задания и представление его на проверку преподавателю.

Пояснительная записка является основным документом, представляемым к проверке. Качество ее оценивается полнотой, точностью и логикой изложения, а также аккуратностью и правильностью заполнения составляющих частей: текста, эскизов и других иллюстраций.

Объем записки составляет 5–7 страниц стандартного формата А4 (210×297 мм^2), включая иллюстрации. В отдельных случаях допускается использование большего объема записки. По краям листа оставляются поля: слева – 25 мм, справа – 10 мм, сверху и снизу – по 20 мм. Общие правила оформления текста приведены в [3].

Все страницы записки, в том числе с иллюстрациями, листингом и т.п., нумеруются и скрепляются (переплетаются) в единую папку.

Схемы, диаграммы, рисунки и другие иллюстративные материалы выполняются единообразно по всей работе, с указанием номера рисунка и подрисуночной подписью; все таблицы нумеруются и сопровождаются названием таблицы (подписью к таблице); все формулы следует пронумеровать.

Пояснительная записка в общем случае может состоять из следующих частей (в скобках указано рекомендуемое количество страниц).

1. Титульный лист, на котором указывается Министерство образования и науки Российской Федерации, полное наименование университета, факультета и кафедры, а также тема индивидуального домашнего задания, Ф.И.О. студента и номер группы, Ф.И.О. преподавателя, город и год (1 страница).

2. Бланк индивидуального домашнего задания, утвержденный заведующим кафедрой. Задание является нормативным документом, устанавливающим границы и глубину разработки темы, а также сроки представления работы на кафедру в завершенном виде. Бланк задания

содержит данные о студенте, название работы и краткие исходные данные. Здесь кратко приводится основное содержание пояснительной записки и графического материала. Задание необходимо поместить в начале пояснительной записки после титульного листа (1 страница).

3. Эскиз и описание схемы обработки потоков данных и сигналов, карта памяти. При необходимости рассматриваются также теоретические аспекты данного вопроса (0.5–1 страница).

4. Графическое представление и описание блок-схемы алгоритма программы (1 страница).

5. Листинг программы на языке ассемблера с подробными комментариями (1–2 страницы).

6. Результаты оценки требуемых ресурсов и времени выполнения программы (0.5–1 страница).

7. Тестовый пример и программа проверки задания (1 страница).

Полностью подготовленная и оформленная пояснительная записка сдается на проверку преподавателю. Преподаватель делает отметку на бланке о ее завершении, а после проверки проставляет оценку.

2.4. Пример выполнения ИДЗ

Пример выполнения упрощенного индивидуального задания, иллюстрирующий основные положения методических указаний к выполнению ИДЗ, которые были изложены выше, приведен в прил. 1.

Пример формулировки задания

В гипотетической микропроцессорной системе (платформа x51) четырехбитовые данные в двоичном коде поступают в асинхронном режиме из внешнего устройства в младшую тетраду порта P0 микроконтроллера по сигналам квитирования «Данные готовы» (единичный уровень от внешнего устройства на линию P0.5) и «Данные обработаны» (нулевой уровень от линии P0.4 микроконтроллера к внешнему устройству). Микроконтроллер переводит эти четырехбитовые данные в семисегментный код и размещает байты семисегментного кода во внешней памяти данных, начиная с адреса 5000H. Количество входных данных не превышает 256 тетрад и задано в регистре (A).

Перевод двоичных кодов в семисегментные осуществлять с использованием таблицы готовых решений. Составить подпрограмму, реализующую данный процесс. Обеспечить локализацию переменных в подпрограмме.

3. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

В данном разделе приведены необходимые данные для выполнения лабораторного практикума по программированию на языке ассемблера микроконтроллеров семейства x51. Рассмотрены особенности организации лабораторного стенда SDK-1, специфика подготовки программ в интегрированной среде μVision фирмы Keil Software, организация загрузочного монитора, программирование периферийных устройств, доступных через регистры программируемых логических интегральных схем (ПЛИС) и через интерфейс I^2C, а также общие вопросы практического программирования на языке ассемблера ASM-51 различных микроконтроллерных устройств.

Данное учебное пособие не заменяет оригинальных руководств и технических описаний. При выполнении лабораторных работ настоятельно рекомендуется пользоваться дополнительной литературой. Так, общие вопросы организации микропроцессорных систем, и в частности платформы x51, рассмотрены в [1], составление программ на языке ассемблера описано в [2], руководство пользователя лабораторного стенда SDK-1 в [4], а спецификация однокристального микроконтроллера ADuC812 приведена в [5].

3.1. Особенности организации лабораторного стенда

Лабораторный стенд SDK-1 построен на основе однокристального микроконтроллера ADuC812, который является дальнейшим развитием микроконтроллера i80c51, имеет архитектуру и систему команд x51, но содержит дополнительное встроенное оборудование и дополнительные регистры для работы с ним. В данном разделе мы рассмотрим лишь некоторые наиболее важные для лабораторного практикума отличия этих кристаллов и особенности построения лабораторного стенда SDK-1. Полное техническое писание лабораторного стенда приведено в руководстве пользователя [4], а полное техническое описание микроконтроллера ADuC812 дано в [5].

3.1.1. Распределение памяти в SDK-1

Стандартная для архитектуры x51 структура резидентной памяти данных представлена четырьмя банками по 8 регистров общего назначения (диапазоны адресов 00H–07H, 08H–0FH, 10H–17H и 18H–1FH), битовым сегментом (20H–2FH), свободным участком 30H–7FH, областью размещения регистров специальных функций 80H–0FFH, доступной при прямой адресации, и свободной областью 30H–7FH, доступной при косвенной адресации (IDATA).

Внешняя память SDK-1 разбита на следующие области с условными названиями ADuC812 Flash/EE, SRAM и MAX.

ADuC812 Flash/EE. Это область размером 8 Кбайт на которую отображается резидентная Flash-память программ микроконтроллера. Напомним, что в резидентной Flash-памяти располагается таблица векторов прерываний и резидентный загрузчик файлов формата HEX-80 в SRAM.

SRAM. Статическая оперативная память SRAM в SDK-1 имеет страничную организацию (максимум 8 страниц по 64K адресов) и условно разделяется на две области. Первая занимает младшие 64 Кбайт (страница 0) и доступна для выборки команд микроконтроллером. Программы могут располагаться только в этой области адресного пространства. Подчеркнем, что резидентная Flash-память программ отображается в самые младшие адреса 0-й страницы (0000H–1FFFH), поэтому реально для размещения программ пользователя на нулевой странице SRAM доступны только 56 Кбайт (2000H–0FFFFH). Остальные страницы внешней памяти доступны только для размещения данных. Для выбора нужной страницы внешней памяти служит регистр специального назначения – регистр-указатель страницы – DPP (адрес 84H).

MAX. В самых младших адресах 8-й страницы адресного пространства (080000H–080007H) располагается 8 ячеек-регистров ПЛИС MAX8064 (MAX8128). Эта область предназначена для взаимодействия с периферийными устройствами стенда.

3.1.2. Система прерываний

Микроконтроллер ADuC812 обеспечивает 9 источников и два уровня приоритета прерываний. Пять из них являются типовыми для платформы x51. Четыре дополнительных источника прерываний специфицированы следующим образом:

PSMI – источник питания, вектор 43H, порядок опроса 1;

ADCI – завершение преобразования аналогоцифрового преобразователя, вектор 33H, порядок опроса 3;

I2CI/SPII – последовательный интерфейс I^2C / SPI, вектор 3BH, порядок опроса 7;

TF2/EXF2 – переполнение разрядности таймера T2, вектор 2BH, порядок опроса 9.

ADuC812 имеет векторы прерываний в диапазоне 0003H–0043H, лежащем в области младших адресов резидентной Flash-памяти программ. В стенде SDK-1 по адресам векторов прерываний во Flash-памяти записаны команды переходов на векторы пользовательской таблицы, размещенной в SRAM в области адресов 2003H–2043H. По адресам векторов пользовательской таблицы 20xx (xx-вектор прерываний) пользователем указываются команды переходов на процедуры обработки прерываний. Например, вектору прерываний 03H будет соответствовать адрес пользовательской таблицы 2003H.

3.1.3. РСФ и битовые поля

В микроконтроллере ADuC812 есть ряд дополнительных (по отношению к базовому кристаллу i80c51) регистров специальных функций и битовых полей. Если дополнительные регистры и поля не используются, то на программном уровне нет никаких отличий от i80c51. В системах разработки имена всех регистров и битовых полей кристалла i80c51 обычно являются зарезервированными именами и определены по умолчанию.

Если в программе используются дополнительные (по отношению к кристаллу i80c51) возможности микроконтроллера, то дополнительные регистры специальных функций и битовые поля необходимо специально определить. Адреса всех регистров приведены в описании микроконтроллера [5]. Для примера определим регистр **dpp** – указатель страницы внешней памяти программ. Адрес регистра 84H.

```
dpp     data  84H
```

После этого данный регистр доступен по своему имени – **dpp**.

В тех случаях, когда требуется сделать много таких определений, удобно воспользоваться специально подготовленным файлом, в котором приведены ассемблерные определения всех байтовых регистров и

битовых полей микроконтроллера ADuC812 (**ADuC812.inc**). Для подключения этого файла необходимо в тексте программы на языке ассемблера сделать декларацию

```
#include    <ADuC812.inc>
```

Необходимо обратить внимание, что соответствующие файлы с определениями на языке C51 имеют другое расширение, а именно (***.h**). Угловые скобки вокруг имени файла означают, что он помещен в стандартную системную директорию интегрированной среды: **\Keil Software\C51\ASM**.

В стандартную поставку интегрированной среды Keil μVision файл **ADuC812.inc** не входит, и в конкретной системе его может не оказаться. В этом случае данный файл необходимо самостоятельно скопировать в системную директорию. При отсутствии прав на запись в системную директорию файл можно разместить в любом другом месте, например в своей рабочей директории **Мой проект Keil**. При этом имя файла следует указывать в двойных кавычках:

```
#include    "ADuC812.inc"
```

Это означает, что файл находится в директории текущего проекта, т.е. в **Мой проект Keil**. При необходимости можно использовать любую другую директорию и внутри кавычек указывать полное маршрутное имя файла.

Программа ассемблер запрещает повторное определение символических имен, определенных ранее. Поскольку в файле **ADuC812.inc** даются определения всех без исключения регистров и адресуемых битовых полей микроконтроллера ADuC812, в том числе имеющихся в кристалле i8051 и определенных по умолчанию, то при ассемблировании будет конфликт имен и сообщение об ошибке. Для устранения конфликта при использовании файла **ADuC812.inc** необходимо отключить определения регистров и битовых полей, используемые в системе по умолчанию (п. 3.2.1).

Предостережение. Особо подчеркнем, что если в разрабатываемой программе не содержится команда `#include <ADuC812.inc>` или `#include "ADuC812.inc"`, то ни в коем случае нельзя отключать определения регистров и битовых полей, используемые в системе по умолчанию (п. 3.2.1).

3.2. Подготовка и загрузка программы

3.2.1. Интегрированная среда Keil μVision

Keil μVision – интегрированная среда фирмы Keil Software для микроконтроллеров семейства x51. Она включает в себя стандартный интерфейс Windows, средства управления проектами, мощный текстовый редактор и многофункциональный отладчик в удобной программной оболочке, транслятор с языка C51, макроассемблер A51, редактор связей L51, симулятор-отладчик, операционную систему реального времени и встроенную справочную систему. Рассмотрим коротко порядок работы с интегрированной средой Keil μVision.

1. В своей личной студенческой директории на сервере необходимо создать директорию '`Мой проект Keil`'. Здесь будут располагаться многочисленные файлы, создаваемые интегрированной средой в процессе работы: исходные тексты, результаты ассемблирования и различные вспомогательные файлы.
2. Для запуска пакета μVision служит исполняемая программа

   `'c:\Program File\Keil Software\uv3\uv3.exe'`

   Для удобства запуска необходимо создать ярлык на нее на своем рабочем столе.
3. Для работы над конкретной задачей или программой в среде μVision необходимо открыть уже существующий проект или создать новый проект. Для этого в меню выбрать

   `Project\New Project...`

   В появившемся окне обзора выбрать свою директорию '`Мой проект Keil`' и ввести имя проекта без расширения. При выборе имени проекта желательно использовать свое имя или фамилию, а также номер лабораторной работы. Это облегчит в дальнейшем поиск нужных файлов. Интегрированная среда μVision корректно обрабатывает любые имена файлов, допустимые в операционной системе Windows. Однако другие вспомогательные программы, например загрузочный монитор T2, имеют ограничения на структуру имен файлов. В этой связи настоятельно рекомендуется имена файлов набирать только латиницей, не использовать в имени файла символ пробела и знаки препинания. Например, выберем имя проекта '`Ivanovlab1`'.
4. После открытия нового проекта необходимо настроить его характеристики и свойства. Для этого на вкладке

   `Project\Select device for Target ...`

выбрать тип микроконтроллера: `Analog Devices\ADuC812`. На вкладке

`Project\Options for Target..\Output`

поставить галочку в квадратике '`Create HEX File`', что приведет к формированию файла загрузочного модуля – т.н. hex-файла. Именно этот hex-файл будет в дальнейшем загружен в лабораторный стенд.

5. Открыть или создать новый текстовый файл для записи исходного текста программы на языке ассемблера. Для этого выбрать в меню `'File\New...'`. Появится окно текстового редактора. В меню выбрать `'File\Save...'`, а в появившемся окне обзора выбрать свою директорию '`Мой проект Keil`' и ввести имя файла с расширением «asm». При первом ознакомлении желательно (но не обязательно), чтобы имя файла совпадало с именем проекта. Например, '`Ivanovlab1.asm`'.
6. Включить созданный файл в проект. Для этого на вкладке

`Project\Components, Environment, Books.. ..\Project components`

выбрать команду **`Add Files`**, в появившемся окне обзора выбрать файл **`Ivanovlab1.asm`**, созданный в предыдущем пункте, и нажать кнопку **Add**. Если необходимо добавить в проект другие файлы, то нужно их по очереди выбирать и нажимать кнопку **Add**. Для завершения процедуры нажать кнопку **Close**.

Отметим, что в проект можно добавлять не только файлы с исходными текстами программы на языке ассемблера, но и файлы с программами, написанными на языке С51 (***.c**), а также объектные файлы (***.obj**) и объектные библиотеки (***.lib**). В зависимости от расширения имени файла интегрированная среда будет вызывать соответствующую обрабатывающую программу (макроассемблер, компилятор С51, редактор связей и т.д.). При выполнении некоторых лабораторных работ нам в дальнейшем потребуется подключение тех или иных объектных библиотек.
7. В текстовом окне набрать текст программы. При использовании в программе файла **ADuC812.inc** необходимо отменить определения регистров и битовых полей, используемые в системе по умолчанию. Для этого открыть вкладку

`Project\Options for Target\A51`

и убрать галочку в квадратике **`Define 8051 SFR Names`**. Если файл **ADuC812.inc** не используется в программе (нет команд

`#include <ADuC812.inc>` или `#include "ADuC812.inc"`), то галочку убирать не надо!

8. Сохранить программу **'File\Save...'**. Теперь можно провести компиляцию (т.е. трансляцию в машинные коды). Для этого выбираем в меню

   Project\Build target

   и следим за сообщениями в окне сообщений. Если есть сообщения об ошибках, то необходимо их проанализировать и исправить исходный текст программы, а затем повторить компиляцию.
9. В результате успешной компиляции в директории **Мой проект Keil** будет создан загрузочный файл с расширением (***.hex**) – т.н. hex-файл с машинными кодами для загрузки в лабораторный стенд. Его имя совпадает с именем проекта, но он имеет расширение (hex). Например, **Ivanovlab1.hex**. Теперь можно свернуть окно интегрированной среды μVision или даже выйти из этой программы. Операцию пересылки hex-файла в лабораторный стенд выполняет другая программа – загрузочный монитор.

3.2.2. Загрузочный монитор T2

Загрузочный монитор T2 работает на компьютере под управлением операционной системы Windows 9.x/NT/2k/XP/W7. Основные функции монитора T2:

- преобразование hex-файлов, заключающееся в добавлении стартового адреса программы в передаваемый файл;
- передача загрузочных модулей в лабораторный стенд;
- получение информации из лабораторного стенда и взаимодействие с резидентным загрузчиком HEX202, работающим в лабораторном стенде;
- обеспечение элементарных операций с последовательным каналом: прием, передача, настройка скорости и т.д.

Рассмотрим коротко порядок работы с загрузочным монитором T2.

1. Приступая первый раз к работе с монитором T2 скопируйте программу T2.exe в директорию 'Мой проект Keil'. В дальнейшем всегда запускайте T2.exe только из этой директории. Это избавит от проблем, связанных с полными маршрутными именами файлов. После запуска появится окно монитора с промптом (#). Все

команды монитора вводятся из командной строки после промпта. Отметим, что монитор T2 предназначен для работы с различными лабораторными стендами и поэтому имеет большой список команд. При работе со стендом SDK-1 нам потребуется лишь несколько команд. Полный список команд содержится в руководстве пользователя монитора.

2. Стенд SDK-1 должен быть включен в режиме ожидания загрузки. На жидкокристаллическом дисплее стенда должна появиться надпись «HEX202 Ожидание...».
3. В окне монитора T2 выполнить подготовительные команды:

   opencom1

   Эта команда готовит порт COM1 компьютера к работе. Если стенд подключен к порту COM2, то команда будет иметь вид **opencom2**. После этого подается команда, добавляющая стартовый адрес программы в конец hex-файла.

   0x02000 0x0 addhexstart <имя_файла>.hex

   Резидентный загрузчик HEX202 использует этот адрес для автоматического запуска программы, загруженной в лабораторный стенд. Параметры команды **addhexstart**: первая цифра (0x02000) – стартовый адрес программы для добавления в конец hex-файла; вторая цифра (0x0) должна присутствовать, но при работе со стендом SDK-1 не используется и может иметь любое значение; последний параметр – имя hex-файла. Команда **addhexstart** добавляет в конец hex-файла строчку, содержащую стартовый адрес загружаемой программы.
   При многократном выполнении команды **addhexstart** в конец hex-файла будет добавлено соответствующее количество идентичных строк, содержащих один и тот же стартовый адрес. Такой hex-файл не может быть корректно загружен в лабораторный стенд.
4. Загрузка программы в лабораторный стенд осуществляется командой

   loadhex <имя_файла>.hex

   В окне монитора появится информация об успешной (или безуспешной) передаче. В случае успешной передачи резидентный монитор HEX202 автоматически запустит программу в лабораторном стенде с указанного адреса.
5. Для штатного выхода из монитора T2 служит команда «bye». В случае «зависания» монитора T2 используют аварийный выход по CTRL–C или CTRL–Break.

Часто повторяющийся набор команд можно оформить в виде скриптового файла, т.е. файла, содержащего команды монитора T2. Скрипт-файл может иметь любое расширение или быть без расширения. Пример скрипт-файла с именем «a»» без расширения:

```
0x2000 0x0 addhexstart kb.hex loadhex kb.hex
```

Файл должен находиться в одной директории с T2.exe. Для вызова скрипт-файла существует специальная команда T2 монитора, которой необходимо сообщить имя скрипт-файла: `lfile a`.

В данном примере команда «`lfile`» загружает и выполняет скрипт-файл с именем «a». Это ускоряет процесс загрузки и запуска программы после компиляции.

Монитор T2 использует ресурсы последовательного порта монопольно. Иными словами, на компьютере можно открывать только одно окно с монитором T2. При открытии двух и более окон с мониторами T2 возникнет конфликт доступа к последовательному порту и мониторы T2 работать не будут!

3.2.3. Монитор T2 в режиме эмуляции терминала

Приведенные выше команды монитора T2 достаточны для взаимодействия с резидентным загрузчиком HEX202. Однако монитор T2 предоставляет и другие полезные возможности. Команда «`N term`» включает эмулятор терминала

`N term`

Т.е. переводит T2 в т.н. режим терминала: все байты, приходящие по последовательному каналу отображаются на экране монитора в двоичном (N = 0) или шестнадцатеричном (N = 1) коде, а все байты, набираемые на клавиатуре, не воспроизводятся на экране, а передаются по последовательному каналу в лабораторный стенд. Выход из этого режима производится по нажатии клавиши **ESC** на клавиатуре компьютера. Для записи получаемой информации в файл предусмотрена команда создания файла с заданным именем для вывода, а также команды включения и выключения копирования консольного вывода в созданный файл. Подчеркнем, что в файл копируются только те байты, которые поступают в компьютер через COM-порт.

`echo <имя_файла_для_вывода>`
`+echo`
`-echo`

Для настройки параметров COM-порта компьютера загрузочный терминал имеет команду вида

```
19200 openchannel com1
```

Первая цифра (19200) задает скорость порта в бодах. В данном примере открывается порт COM1 для работы на скорости 19200 бод. Отметим, что команда opencom1 является просто сокращением команды «9600 openchannel com1». Для закрытия канала существует команда

```
closechannel
```

С использованием данного режима монитора T2 возможно выводить из лабораторного стенда и записывать в файл на компьютере большие массивы информации, например создавать образы памяти или выводить отладочную информацию.

3.2.4. Инструментальная среда HEX202ldr

Штатное программное обеспечение стенда SDK-1 включает в себя резидентный загрузчик HEX202, находящийся в памяти микроконтроллера, и монитор T2, находящийся в памяти компьютера. Несмотря на достаточно развитую функциональность, монитор T2 имеет ряд ограничений, таких как консольный интерфейс и отсутствие возможностей для работы по локальной сети.

На кафедре экспериментальной физики разработана графическая инструментальная среда HEX202ldr [6]. Ее особенностью является использование технологии клиент-сервер. Связь между клиентом и сервером реализуется по стандартному TCP/IP протоколу с использованием локальных или глобальных сетевых телекоммуникаций. Сервер через последовательный канал передачи данных взаимодействует с лабораторным стендом, клиент взаимодействует с источником hex-образа программы. Исполняемый файл инструментальной системы HEX202ldr содержит оба компонента: клиент и сервер, первоначально связанные через локальный IP-адрес 127.0.0.1. К серверу данной системы по сети могут быть подключены удаленные клиенты, а клиент данной системы, в свою очередь, может быть подключен к удаленному серверу. Это дает возможность сетевой загрузки hex-образа программы на микроконтроллерный стенд, подключенный к удаленному компьютеру.

В инструментальной системе HEX202ldr реализован удобный оконный интерфейс, предоставлены дополнительные возможности при загрузке и эмуляции терминала. Из дополнительных сервисных возможностей системы отметим наличие встроенной системы мгновенных сообщений (чат), которая позволяет обмениваться сообщениями между

удаленными клиентами и серверами. В режиме эмуляции терминала организован ввод-вывод в различных системах счисления. Разработанная инструментальная система рассчитана на работу с различными типами операционных систем от Windows 9x до Vista. Возможна работа с операционной системой Linux при использовании бинарного эмулятора Wine.

Система HEX202ldr имеет графический интуитивно понятный интерфейс и встроенное руководство пользователя на русском языке. Данная система рекомендуется к использованию в лабораторном практикуме вместо загрузочного монитора T2.

3.3. Особенности программирования ADuC842

Лабораторные стенды SDK-1 начиная с апреля 2004 года поставляются не только с микроконтроллером ADuC812, но и с другими микроконтроллерами семейства MicroConverter фирмы Analog Devices Inc. Все микроконтроллеры этого семейства имеют одинаковую систему команд, базовый набор периферии, однако имеют и некоторые отличия [7, 8], которые следует учитывать при подготовке программ для стенда SDK-1 (табл. 3.1).

3.3.1. Организация памяти стенда с ADuC842

В стендах SDK-1 с микроконтроллером ADuC842 объем резидентной Flash-памяти программ составляет 62 Кбайт. Возможность исполнения команд из внешних по отношению к микроконтроллеру запоминающих устройств отсутствует. В то же время возможно перепрограммирование части внутреннего блока Flash-памяти во время исполнения команд из другой его части. Весь блок Flash-памяти программ (диапазон адресов 0000H–F7FFH) доступен для стирания/перезаписи в инструментальном режиме работы микроконтроллера, однако в штатном режиме для стирания/перезаписи доступны только ячейки по адресам 0000H–DFFFH (56 Кбайт). Для этого вся память в указанном диапазоне разбита на секторы по 64 байта каждый, которые можно стирать независимо друг от друга. Программировать можно как отдельную ячейку (байт), так и страницу из 256 байт Flash-памяти команд. При этом команды на стирание/программирование могут располагаться где угодно, поскольку на время выполнения операции с Flash-памятью ядро микроконтроллера приостанавливается (не выбирает команды и не отвечает на запросы на прерывание). Таким образом, возможность модификации содержимого памяти программ

Таблица 3.1. Ключевые отличия вычислителей [7]

Параметр	ADuC812	ADuC842
Резидентная Flash-память команд, Кбайт	8	62
Перепрограммирование во время исполнения	Нет	Да
Возможность исполнения команд из внешнего ОЗУ	Да	Нет
Flash-память данных, байт	640	4096
ОЗУ IDATA, байт	256	256
ОЗУ XDATA, байт	–	2048
Частота ядра, МГц	11.0592	Настраиваемая (PLL)
Длительность машинного цикла, машинных тактов	12	1
Скорость UART макс., бод	19200	115200
Синхронизация UART	T/C1, T/C2	T/C1, T/C2, T/C3
Доступная в SDK-1 дополнительная периферия (по сравнению с ADuC812)	Монитор питания	Два ШИМ, счетчик интервалов времени

в SDK-1.1 с вычислителем ADuC842 сохранена, но память программ стала энергонезависимой. Область адресов E000H–F7FFH отведена для хранения системных программ. Она недоступна пользователю в штатном режиме. Однако в инструментальном режиме возможна за-

грузка в эту область системных программ. Диапазон адресов F800H–FFFFH (2 Кбайт) в штатном режиме работы микроконтроллера отображает последовательность из инструкций «nop» (код 00H).

3.3.2. Обеспечение совместимости программ

Для сохранения совместимости программ в стенде с ADuC842 предусмотрена поддержка старой процедуры установки пользовательских векторов с помощью записи по адресу (2000H + *x*) команды передачи управления на код обработки прерываний. При этом предполагалось, что по адресу (2000H + *x*) пользователь записывает команду безусловной дальней передачи управления «ljmp address», где address – адрес обработчика прерываний пользователя. Именно такой способ обработки прерываний (дальняя передача управления на свой обработчик) предполагался наиболее вероятным в SDK-1 с ADuC812. Для сохранения совместимости в SDK-1 с ADuC842 при возникновении прерываний управление передается на код, который извлекает 16-разрядное число по адресу (2000H + *x* + 1), интерпретирует его как адрес обработчика и передает по нему управление.

В SDK-1 с ADuC842 код, обеспечивающий совместимость со старым механизмом установки векторов прерываний, располагается в диапазоне адресов 0000H–007FH. Эти адреса соответствуют области Flash-памяти, отведенной для хранения пользовательского кода. При загрузке программ, располагающихся в адресах, которые не пересекаются с диапазоном 0000H— 007FH, ячейки в этом диапазоне адресов не стираются. В случае, если не предполагается использовать описанную выше схему установки векторов, в этом диапазоне можно разместить собственный код/данные. При этом, очевидно, имевшийся там на момент поставки стенда код будет утерян. Если в дальнейшем потребуется его восстановить, то для этого достаточно загрузить в стенд образ демонстрационной программы, входящей в комплект поставки.

В стенде SDK-1 с ADuC842 тактовая частота настраивается с помощью встроенного блока PLL как частное от деления 16.777216 МГц на число из набора {1, 2, 4, 8, 16, 32, 64, 128}. Длительность машинного цикла составляет один такт синхронизации. Это повысило производительность микроконтроллера, но из-за кардинального снижения длительности машинного цикла в них пришлось несколько увеличить число циклов, необходимых для исполнения части инструкций (это в основном инструкции передачи и возврата управления). Данное обстоятельство может повлиять на совместимость программ, учитывающих длительность исполнения инструкций. Частота синхронизации микроконтроллера влияет на весь набор его периферии, поэтому дан-

ное обстоятельство следует учитывать при работе с разными стендами и написании совместимых программ. Это касается, в первую очередь, таймеров 0 и 1, UART (если он тактируется от таймера 1). Различия в производительности микроконтроллеров становятся особенно заметными при использовании программ временных задержек, основанных на времени исполнения инструкций или блоках таймера 0 или 1. Например, программа задержки, основанная на таймере 0, в стенде с ADuC842 на 16.777216 МГц будет в 18.2 раза короче, чем в стенде с ADuC812.

В ADuC842 имеется возможность синхронизации UART как от таймеров 1 и 2, так и от специального таймера 3, предназначенного только для тактирования UART. Максимальная стандартная скорость, развиваемая UART при использовании таймера 3, равняется 230400 бод. Однако в SDK-1 с ADuC842 порт RS232 может быть рассчитан только на максимальную скорость 115200 бод, что вполне достаточно для быстрого обмена данными через RS232.

3.3.3. Загрузка программ пользователя

В SDK-1 с микроконтроллером ADuC842 нет необходимости при каждом старте стенда ожидать от компьютера появления образа новой пользовательской программы. Может быть запущена программа, ранее размещенная во Flash-памяти программ. Для доставки с компьютера нового образа программы пользователя производится проверка канала связи сразу после старта стенда. Протокол взаимодействия с компьютера по доставке пользовательских программ остался прежним, поэтому для связи с компьютером резидентный загрузчик SDK-1 сначала должен продемонстрировать готовность к приему HEX-команд выдачей символа ‘.’ в канал связи. Для этого при старте SDK-1 в канал выдается последовательность из 10 символов ’.’, после каждого из которых производится ожидание реакции со стороны компьютера (приход HEX-команды). Временной интервал каждой проверки составляет примерно 20 мс. Таким образом, суммарное время установления связи с компьютером составляет примерно 200 мс, после чего управление передается пользовательской программе, которая была записана в память ранее. Кроме последовательности из 10 символов ’.’, в канал связи с компьютером также выдается надпись «HEX202-03» и, возможно, другие (в отладочных целях). Однако все они составляют не более 20 символов.

Запуск программ пользователя в стенде с ADuC842 осуществляется с адреса, запомненного в момент загрузки. Для хранения этого адреса (точки входа), а также другой информации зарезервированы

две страницы Flash-памяти данных по адресам 00H и 01H.

Для непосредственной загрузки программы пользователя во Flash-память программ стенда на компьютерах PC с операционной системой Windows используется графическая среда Windows Serial Downloader (WSD). Программа wsd.exe предоставляет пользователю удобный графический интерфейс, позволяющий настроить параметры последовательного порта, выбрать требуемый режим загрузки Flash-памяти программ и Flash-памяти данных и осуществить сам процесс загрузки и запуска программы пользователя на выполнение.

Помимо графического интерфейса программа wsd.exe предоставляет пользователю возможность работы из командной строки:

wsd.exe опции file.hex

В этом случае необходимые параметры могут быть переданы через опции командной строки (опции могут отсутствовать):

/C:n – выбор номера COM-порта (по умолчанию 1);

/D – произвести загрузку Flash-памяти программ без стирания Flash-памяти данных;

/R – запустить программу на выполнение с адреса 0000H;

/R:xxxx – запустить программу на выполнение с адреса xxxx;

/V – после загрузки провести верификацию кода программы;

/X:freq – установить частоту внешнего резонатора (для ADuC812);

/B – опция загрузчика, запускает с адреса E000H;

/M:n – режим загрузки:

0 – загрузить коды и данные (файл данных должен быть указан в командной строке после файла кодов);

1 – загрузить только коды программы пользователя;

2 – загрузить только данные (файл данных должен быть указан в командной строке после файла кодов, однако файл кодов не будет загружен);

/S:n – режим безопасности:

0 – режим блокирования;

1 – безопасный режим;

2 – режим безопасности последовательного канала;

3 – установка всех битов безопасности;

/T:n – программа остается открытой на *n* секунд, *n* = 0–9.

В комплект программного обеспечения, поставляемого со стендом SDK-1 с ADuC842, входит также файл loader.hex. В этом файле содержится образ программы резидентного загрузчика HEX202-03 для программирования резидентной Flash-памяти программ микроконтроллера ADuC842. Для установки резидентного загрузчика его образ при помощи программы `wsd` размещается в системной области Flash-памяти программ микроконтроллера ADuC842:

```
wsd.exe /B loader.hex
```

После этого становится возможным использовать для загрузки в резидентную память программ микроконтроллера ADuC842 любые программы, предназначенные для работы с резидентным загрузчиком HEX202 микроконтроллера ADuC812 (монитор T2, HEX202ldr и др.). При этом необходимо учитывать, что стенд с ADuC842 после сигнала сброса ожидает от компьютера отклик только в течение примерно 200 мс. Рекомендуемая последовательность действий при этом – сначала запустить загрузочную программу-монитор на компьютере, а затем осуществить аппаратный сброс лабораторного стенда SDK-1.

3.4. Доступ к периферийным устройствам через ПЛИС

Лабораторный стенд SDK-1 содержит ряд периферийных устройств, доступных через регистры ПЛИС (табл. 3.2). Это клавиатура, жидкокристаллический (ЖК) дисплей, линейка из восьми светодиодов, внешний параллельный порт и звуковой излучатель. Все регистры ПЛИС расположены на странице 8 внешней памяти данных. Для доступа к регистрам ПЛИС нужно сначала выбрать 8-ю страницу внешней памяти данных. Для этого служит специальный регистр DPP (адрес 84H) – указатель страницы данных. Необходимо помнить, что при переключении страниц становятся недоступными все данные, размещенные на других страницах памяти. Т.е. перед обращением к DPP все передаваемые данные необходимо разместить в резидентной памяти данных. Кроме того, перед записью в регистр DPP необходимо временно сохранить его предыдущее содержимое (номер предыдущей страницы). После окончания работы с регистрами ПЛИС необходимо восстановить предыдущее содержимое регистра DPP.

Подпрограмма `PUTBYTE` иллюстрирует технологию доступа к регистрам ПЛИС лабораторного стенда SDK-1. Программа подробно прокомментирована. Следует обратить особое внимание на приемы корректной работы с регистрами микроконтроллера и на технологию доступа к страницам внешней памяти данных.

Таблица 3.2. Спецификация регистров ПЛИС

Адрес	Регистр	Доступ	Назначение
080000H	KB	R/W	Регистр клавиатуры
080001H	DATA_IND	R/W	Регистр шины данных ЖК-дисплея
080002H	EXT_LO	R/W	Регистр данных внешн. порта (d7…d0)
080003H	EXT_HI	R/W	Регистр данных внешн. порта (d15…d8)
080004H	ENA	W	Регистр управления внешним портом ввода-вывода, звуком и сигналом $\overline{\text{INT0}}$
080006H	C_IND	W	Регистр управления ЖК-дисплеем
080007H	SV	W	Регистр управления светодиодами

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Подпрограмма PUTBYTE
; Вывод байта в регистр ПЛИС (0h...7h) стенда.
; (A) -- байт данных; (dptr) -- адрес рег. ПЛИС.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
putbyte:
       push dpp       ; Сохр. номер текущей стр.
       mov  dpp,#08d  ; Переход на 8-ю стр.
       movx @dptr,A   ; Запись в регистр ПЛИС.
       pop  dpp       ; Восст. номера стр.
       ret            ;
; Конец подпрограммы PUTBYTE
       end            ;
```

3.4.1. Светодиодные излучатели

Для работы с линейкой светодиодов служит регистр SV, каждый бит которого управляет свечением одного светодиода. Единичное значение бита соответствует свечению светодиода. Программирование светодиодных излучателей сводится к выводу байта данных в регистр SV (табл. 3.2). Для иллюстрации приведем программу, которая копирует байт данных из аккумулятора в регистр SV с использованием подпрограммы PUTBYTE.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Вывод байта на линейку светодиодов стенда SDK-1
; с использованием подпрограммы putbyte
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        org   02000h       ; Начало SRAM ВПП.
dpp     data  84h          ; Адрес регистра dpp.
        mov   a,#11000011b ; Байт данных для вывода.
        call  svdisp       ;
        sjmp  $            ; Конец программы.
svdisp: mov   dptr,#07h    ; Адрес рег. светодиодов.
        call  putbyte      ;
        ret                ;
        end                ;
```

3.4.2. Жидкокристаллический дисплей

Двухстрочный дисплей на основе жидкокристаллических индикаторов типа WH1602B-YGK-CP (Winstar Display) содержит 32 знакоместа (по 16 знакомест в каждой строке). После сброса вывод происходит в левую верхнюю позицию дисплея. При каждом выводе курсор автоматически сдвигается вправо на следующее знакоместо. Данный дисплей не переводит автоматически строку и не возвращает автоматически курсор в начало строки. Это необходимо осуществлять вручную при помощи специальных команд.

Дисплей имеет встроенный знакогенератор, содержащий различные символы, цифры и буквы (латиница и кириллица). Каждый символ кодируется одним байтом. Большинство символов с кодами 00H…7FH соответствуют таковым для ASCII кодировки. В частности, сюда входят знаки препинания, цифры, а также строчные и прописные латинские буквы. Наличие ASCII кодировки дает возможность использовать литералы для задания кода символов при программировании. Кириллица имеет нестандартную кодировку: знакогенератор содержит лишь кириллические символы, отличающиеся по начертанию от латиницы. Все прочие кириллические символы необходимо заменять латинскими символами, сходными по начертанию.

Для взаимодействия с жидкокристаллическим дисплеем служат два регистра ПЛИС (табл. 3.2). Регистр данных `DATA_IND` позволяет устанавливать данные на шине данных дисплея и считывать их оттуда. При этом необходимо также использовать регистр управления дисплеем `C_IND` (табл. 3.3). В этом регистре программист может использовать лишь три младших бита.

В табл. 3.4 приведены некоторые наиболее употребительные команды для работы с дисплеем. Приведенный набор команд достато-

Таблица 3.3. Спецификация битов регистра C_IND

Бит	Поле	Описание
0	E	Бит управления входом «E» ЖК-дисплея. Наличие положительного импульса 0–1–0 на входе «E» фиксирует информацию на шине ЖК-дисплея. Данные и сигналы RS, RW к этому моменту должны быть уже установлены. Операцию чтения следует проводить при установленном значении «E» = 1
1	RW	Переключение шины ЖК-дисплея на чтение (1), запись (0)
2	RS	Переключение режимов команды (0) или данные (1)

Таблица 3.4. Примеры команд управления дисплеем

Команда	RS	RW	Регистр DATA_IND							
Очистка дисплея	0	0	0	0	0	0	0	0	0	1
Возврат в начало строки	0	0	0	0	0	0	0	0	1	×
Установка текущей позиции курсора	0	0	1	a6	a5	a4	a3	a2	a1	a0
Чтение флага занятости (BF) и текущей позиции курсора	0	1	BF	a6	a5	a4	a3	a2	a1	a0

Примечание. (a6...a0) – код позиции курсора, отсчитанный слева направо: 00H...0FH для верхней строки и 40H...4FH для нижней строки дисплея.

чен для выполнения лабораторных работ. Он включает в себя команды очистки дисплея, позиционирования курсора, перевода строки, возврата курсора в начальное положение строки и т.д. Кодировка координат курсора приведена в примечании к табл. 3.4. Полный список команд управления жидкокристаллическим дисплеем содержится в руководстве пользователя стенда SDK-1 [4].

Дисплей имеет два режима работы, переключаемые битом RS: режим команд и режим данных. В первом случае контроллер дисплея воспринимает коды на шине данных дисплея в качестве команд управ-

ления дисплеем (табл. 3.4). В режиме данных происходит вывод кодов на шине дисплея на экран в соответствие с таблицей встроенного знакогенератора. Особую роль в обмене данными между микроконтроллером и дисплеем играет бит E (табл. 3.3). При любом виде обмена данными на этот бит должен быть подан строб 0–1–0. Непосредственный обмен происходит при значении E = 1. Для режима записи данных (информация передается от микроконтроллера к дисплею) необходимо вначале выставить данные на шине данных ЖК-дисплея, а затем подать строб E. Менее очевидна последовательность действий при чтении данных (информация пересылается от дисплея к микроконтроллеру). В этом случае необходимо сначала подать единичный уровень на вход E, выполнить чтение шины данных дисплея, а затем снять единичный уровень со входа E. Иными словами, чтение необходимо провести на середине строба 0–1–0 при его единичном значении. До и после строба E информация не может быть прочитана. При чтении от контроллера дисплея может быть получена информация о текущем положении курсора и о состоянии флага занятости дисплея. Установленный флаг занятости BF означает, что контроллер дисплея в это время занят выполнением внутренних операций и дисплей является программно недоступным. Время, в течение которого дисплей недоступен, может быть достаточно продолжительным – до миллисекунд. Это определяется временем операций в ЖК-дисплее. При выводе на дисплей потока данных опрос флага занятости является обязательным. В противном случае некоторые данные могут быть утеряны и не выведены на экран.

Подпрограмма PUTCHAR иллюстрирует технологию работы с дисплеем. Она позволяет передавать байт данных или команды контроллеру дисплея. При анализе программы следует обратить внимание на описание битовых полей и формирование управляющего строба бита E.

Значение бита RS, определяющего режим команд или данных, задается перед вызовом подпрограммы в бите F0. Байт информации для вывода задается в регистре (A). Строб E формируется после выставления информации на шину данных дисплея.

Подпрограмму PUTCHAR можно легко модифицировать для работы в режиме чтения данных с дисплея, т.е. превратить в подпрограмму GETCHAR. При модификации подпрограммы недостаточно просто изменить направление пересылки данных. Чтение данных необходимо выполнить в момент единичного значения строба E.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Подпрограмма PUTCHAR
; Вывод одного байта на ЖК-дисплей стенда SDK-1.
; Байт для вывода -- в аккумуляторе.
; Значение бита RS -- в F0 (0--команды, 1--данные).
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dpp      data   84h              ; Адрес рег. dpp.
DATA_IND xdata  01h              ; Рег. данных ЖКД.
C_IND    xdata  06h              ; Рег. управ. ЖКД.
; Образы битовых полей регистра C_IND.
E        BIT    acc.0            ;
RW       BIT    acc.1            ;
RS       BIT    F0               ;
putchar: push   psw              ;
         push   dph              ;
         push   dpl              ;
         push   dpp              ;
; Переключение на 8-ю стр. внешней памяти данных.
       mov      dpp,#08d         ; Выбор 8-й стр.
; Вывод данных в рег. данных ЖК-дисплея.
       mov      dptr,#DATA_IND ; Адр. рег. данных.
       movx     @dptr,A          ; Вывод данных.
; Запись кодовых полей в рег. управления ЖКД
       clr      RW               ; RW=0.
       mov      c,RS             ; Чтение RS.
       mov      acc.2,c          ; RS=1 или 0.
; Формирование строба 0--1--0 на входе 'E' ЖКД.
       mov      dptr,#C_IND      ; Адр. рег. команд.
       clr      E                ; E=0.
       movx     @dptr,A          ; Запись в C_IND.
       nop                       ;
       setb     E                ; E=1.
       movx     @dptr,A          ; Запись в C_IND.
       nop                       ;
       clr      E                ; E=0.
       movx     @dptr,A          ; Запись в C_IND.
       pop      dpp              ;
       pop      dpl              ;
       pop      dph              ;
       pop      psw              ;
       ret                       ;
; Конец подпрограммы PUTCHAR
```

3.4.3. Матричная клавиатура

Клавиатура типа AK1604A-WWB (ACCORD) реализована в виде квадратной матрицы 4×4. Столбцы и строки сведены в регистр клавиатуры KB (табл. 3.5).

Программа GETKEY иллюстрирует технологию сканирования и опроса клавиатуры. Она не содержит схемы подавления дребезга контактов, а срабатывает по первому замыканию контакта. На основе кодов COL и ROW формируется код нажатой клавиши (00h…0fh – слева направо по строкам, начиная сверху) и выдается в аккумулятор. Программа однократно сканирует клавиатуру и выдает код нажатой клавиши или код 0ffh,если ни одна клавиша не была нажата.

Таблица 3.5. Спецификация регистра клавиатуры

Биты	Поле	Описание
0…3	COL	Поле предназначено для сканирования клавиатуры (колонки матрицы). Сканирование производится (0) в одном из разрядов поля
4…7	ROW	Поле предназначено для считывания данных с клавиатуры (строки матрицы). Если ни одна из кнопок не нажата, то все биты содержат (1). Если кнопка нажата и на ее колонку подан логический нуль при сканировании, то в поле ROW появится логический нуль

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Подпрограмма GETKEY.
; Опрос клавиатуры стенда SDK-1.
; Код нажатой клавиши (00h...0fh) -- в (A).
; Если ни одна клавиша не нажата, то код 0ffh.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
dpp     data   84h      ; Адрес регистра dpp.
KB      xdata  00h      ; Адрес регистра клавиатуры.
; Сохранение используемых регистров в стеке.
getkey:
        push   psw      ;
        mov    psw,#0H  ; Выбрали нулевой банк.
        push   dph      ;
        push   dpl      ;
        push   1h       ; Сохранение r1
```

```
        push  b              ;
        push  dpp            ; Сохранение номера стр.
; Переключение на 8-ю страницу ВПД.
        mov   dpp,#08        ;
; Сканирование клавиатуры.
        mov   dptr,#KB       ; адрес регистра KB.
        mov   r1,#0          ; счетчик колонок = 0
        mov   b,#01111111b   ; сканирующий (0)
KB_lp:  mov   a,b            ;
        rl    a              ; подготовка скан.(0)
        mov   b,a            ; сохранение скан.(0)
        movx  @dptr,a        ; вывод COL в KB скан.(0)
        movx  a,@dptr        ; чтение KB ROW
        cpl   a              ; инверсия ROW
        anl   a,#0f0h        ; ROW в ст.тетраде (A)
        jnz   kb_cod         ; переход, если нажата кл.
        inc   r1             ; счетчик колонок (r1)
        cjne  r1,#4,KB_lp    ; продолжить сканирован.
        mov   a,#0ffh        ; нет нажатых клавиш
        sjmp  kb_end         ;
; Формирование кода нажатой клавиши (00h...0fh)
; (A)=(rrrr0000), где 'rrrr' = NOT(ROW)
; COL -- не сохраняется, вместо него -- сч. колонок
; R1 =(000000cc), где 'cc' = счетчик колонок (0...3)
kb_cod: mov   b,#0           ; очистка рег. (B)
        mov   c,acc.7        ;
        orl   c,acc.5        ;
        mov   b.2,c          ;
        mov   c,acc.7        ;
        orl   c,acc.6        ;
        mov   b.3,c          ;
        mov   a,b            ;
        add   a,r1           ;
; Восстановление использованных регистров
kb_end: pop   dpp            ;
        pop   b              ;
        pop   1h             ;
        pop   dpl            ;
        pop   dph            ;
        pop   psw            ;
        ret                  ;
; Конец подпрограммы GETKEY
```

3.4.4. Звуковой излучатель

Лабораторный стенд SDK-1 содержит встроенный пьезоэлектрический звуковой излучатель типа HPA17 (JL World Corp. Ltd). Для работы со звуковым излучателем служат три бита регистра ENA ПЛИС (табл. 3.2): ENA.2, ENA.3 и ENA.4. На любой из этих битов необходимо подать меандр из логических уровней 0–1–0–1–… с заданной частотой. Биты различаются лишь громкостью звука: от тихого (ENA.2) до чуть более громкого (ENA.4). При выводе данных в регистр ENA необходимо иметь в виду, что бит ENA.5 соединен со входом $\overline{\text{INT0}}$ микроконтроллера и на нем должна поддерживаться логическая единица (отсутствие сигнала прерывания).

Приведенная ниже программа иллюстрирует формирование непрерывного звука заданного тона. Подпрограмма задержки реализована методом программного цикла и на кристалле ADuC812 позволяет получить задержку 155 мкс (частота колебаний около 3.2 кГц).

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Формирование непрерывного звука заданного тона
; с использованием подпрограммы PUTBYTE.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        org   2000h      ;
dpp     data  84h        ; Указатель страницы ВПД.
ena     xdata 04h        ; Адрес регистра ENA.
buz     bit   acc.3      ; Бит для вывода звука.
SOUND:  mov   dptr,#ena  ;
        mov   a,#20h     ; INT0 <- 1.
loop:   setb  buz        ; (1) для вывода.
        call  putbyte    ;
        call  delay      ; Задержка.
        clr   buz        ; (0) для вывода.
        call  putbyte    ;
        call  delay      ;
        sjmp  loop       ; Конец программы.
; Подпрограмма временной задержки для ADuC812.
delay:  mov   r1,#099h   ; Задержка около 150 мкс.
        djnz  r1,$       ; Цикл.
        ret              ; Возврат из п/п.
; Конец примера программы генерации звука.
        end              ;
```

3.4.5. Внешний параллельный порт ввода-вывода

Регистры ПЛИС EXT_LO и EXT_HI предоставляют доступ к младшему (биты 0…7) и старшему (биты 8…15) регистрам внешнего параллельного порта ввода-вывода. Для управления регистрами этого порта используют биты EN_LO (ENA.0) и EN_HI (ENA.1) регистра ENA. При выводе данных необходимо установить EN_LO или EN_HI в состояние логической единицы, а для ввода – в состояние логического нуля. Старший и младший регистры порта программируются независимо друг от друга.

16 линий внешнего параллельного порта ПЛИС и 4 линии порта P3 микроконтроллера (линии T0/1, INT0/1) выведены на внешний разъем J3, расположенный справа от клавиатуры. Отметим, что все остальные линии портов микроконтроллера (P0, P1, P2) уже задействованы в стенде для служебных целей (см. руководство пользователя).

Набор переключателей SW3 лабораторного стенда позволяет принудительно подавать логический нуль или единицу на соответствующие выводы разъема J3. Набор переключателей содержит две перемычки и панель из 10 DIP-микропереключателей (цветная панель справа от ЖК-дисплея). Логическому нулю соответствует замыкание выводов перемычки или положение ON микропереключателя. Следует обратить внимание, что номер переключателя в описании и цифра на панели DIP-микропереключателей не совпадают. В табл. 3.6 приведено соответствие номеров. Для стенда с вычислителем ADuC812 состояние бита 7 порта EXT_LO проверяется резидентной программой начальной инициализации, находящейся в Flash-памяти и автоматически запускаемой после аппаратного сброса или включения питания. При нулевом значении бита 7 (ON-положение микропереключателя 10 – движок в левом положении) происходит запуск встроенных демонстрационных и тестовых программ лабораторного стенда. В положении OFF (движок в правом положении) управление сразу передается резидентному загрузчику HEX202.

3.5. Программирование УАПП

Наличие в микроконтроллере ADuC8xx встроенного программируемого универсального асинхронного приемопередатчика (УАПП) позволяет организовать эффективное взаимодействие лабораторного стенда SDK-1 с эмулятором терминала T2, функционирующего на компьютере. При этом все набираемые на клавиатуре коды будут поступать на вход УАПП, а все байты, передаваемые УАПП, будут отображаться на терминале в бинарном или шестнадцатеричном коде. Та-

Таблица 3.6. Спецификация переключателей SW3

Номер переключателя	Цифра на панели DIP	Местоположение	Коммутируемая линия
1	–	Перемычка	Вход $\overline{\text{INT0}}$ (P3.2)
2	–	Перемычка	Вход $\overline{\text{INT1}}$ (P3.3)
3	1	DIP	Вход T0 (P3.4)
4	2	DIP	Вход T1 (P3.5)
5	3	DIP	Линия 0 порта `EXT_LO`
6	4	DIP	Линия 1 порта `EXT_LO`
7	5	DIP	Линия 2 порта `EXT_LO`
8	6	DIP	Линия 3 порта `EXT_LO`
9	7	DIP	Линия 4 порта `EXT_LO`
10	8	DIP	Линия 5 порта `EXT_LO`
11	9	DIP	Линия 6 порта `EXT_LO`
12	10	DIP	Линия 7 порта `EXT_LO`

ким образом, из лабораторного стенда можно передавать достаточно большие объемы информации для отображения на экране или для записи в файл на компьютере. И наоборот, из компьютера передавать массивы данных для размещения в памяти на стенде. Это позволяет исследовать содержимое областей памяти микроконтроллера и различных периферийных устройств, входящих в состав лабораторного стенда.

Следующая программа иллюстрирует технологию пересылки данных с использованием УАПП. В данном примере ASCII код клавиши, нажатой на клавиатуре компьютера, принимается УАПП, отображается на светодиодных индикаторах стенда и отсылается обратно в COM-порт компьютера для отображения на экране монитора T2, работающего в режиме эмуляции терминала. Можно наблюдать принятые байты в шестнадцатеричном коде (`1 term`) или в бинарном формате (`0 term`).

Подпрограмма UART_INI иллюстрирует технологию начальной инициализации УАПП. В данном примере УАПП инициализируется для работы со скоростью 9.6 кбит/с с запретом прерываний для работы по интерфейсу RS232 в режиме опроса флагов. Эта программа должна быть выполнена до начала любых пересылок данных с использованием УАПП.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Пример пр., кот.принимает байт от компьютера,
; выводит его на светодиодную линейку и отсылает
; обратно в COM-порт компьютера (эхо-печать).
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        org     2050h           ;
        call    uart_ini1       ; Инициализация aduc812.
;       call    uart_ini2       ; Инициализация aduc842.
; Прием байта с выводом на линейку светодиодов.
loop:   jnb     scon.0,$        ; Ожидание байта.
        mov     a,sbuf          ; Прием байта.
        clr     scon.0          ; Сбр. фл. приемника.
        call    svdisp          ; Вывод на светодиоды.
; Передача байта обратно в COM порт компьютера.
        mov     sbuf,a          ; Передача байта.
        jnb     scon.1,$        ; Ожид. конца передачи.
        clr     scon.1          ; Сбр. фл. передатчика.
        sjmp    loop            ; Конец программы.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Подпрограмма UART_INI для ADuC812 (9600 baud)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
S9600   equ     0fdh            ; Скорость 9.6 kbod.
uart_ini1: mov  th1,#S9600      ; Скорость UART.
        orl     tmod,#20h       ; T/C1 -- в реж.autorel.
        anl     pcon,#7fh       ; Скорость не удваивать.
        orl     tcon,#40h       ; Запуск таймера 1.
        mov     scon,#50h       ; Настройка УАПП.
        clr     ie.4            ; Запрет прерыв. УАПП.
        ret                     ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Подпрограмма UART_INI для ADuC842 (9600 baud)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
pllcon  data    0D7h            ;
t3con   data    09Eh            ;
t3fd    data    09Dh            ;
uart_ini2:
        mov     pllcon,#03h     ; Core f=2.097152 MHz.
        mov     t3con,#83h      ;
        mov     t3fd,#2dh       ;
        mov     scon,#52h       ;
        ret                     ;
        end                     ;
```

3.6. Периферийные устройства, доступные через I²C

Микроконтроллер ADuC8xx содержит I²C/SPI-последовательный порт. В лабораторном стенде SDK-1 на основе этого порта организована шина последовательного интерфейса I²C для подключения периферийных устройств. Шина I²C – это двунаправленная последователь-

Таблица 3.7. Спецификация адресов I²C

Адрес I²C								Периферийное устройство
1	0	1	0	0	0	1	r/w	EEPROM данных
1	0	1	0	0	0	0	r/w	Часы реального времени

Примечание. r/w – выбор режимов чтения (1) или записи (0).

ная шина, соединяющая между собой различные интегральные схемы или модули. Она содержит две линии: линию передачи данных (SDA) и линию синхронизации (SCL). В стенде SDK-1 к шине I²C подключены только два встроенных периферийных устройства: часы-календарь реального времени и внешняя EEPROM данных. Взаимодействие с микроконтроллером оба периферийных устройства осуществляют через интерфейс I²C. Адреса I²C этих устройств приведены в табл. 3.7.

3.6.1. Часы-календарь реального времени

Часы-календарь типа PCF8583 являются I²C – устройством с внутренней встроенной памятью объемом 256 байт и работают от кварцевого резонатора с частотой 32.768 кГц. Из 256 байт памяти собственно часами используется только первые 16: 8 постоянно обновляемых регистров-защелок на установку/чтение даты/времени и 8 – на обслуживание будильника. Остальные 240 байт доступны для хранения данных пользователя. Точность измерения времени – до сотых доле секунды. Спецификация регистров и форматы данных приведены в руководстве пользователя [4].

Отметим только, что информация о текущем времени содержится в 4 регистрах часов с адресами 1…4: сотые доли секунды, секунды, минуты и часы соответственно. Данные приведены в двоично-десятичном коде. Два старших бита регистра 4 используются для служебных целей – для организации переключателя 12 часовой или 24-часовой шкалы часов. В регистрах 5 и 6 в упакованном виде хранится обновляемая информация о дате: год, месяц, число и день недели.

3.6.2. Внешняя EEPROM данных

Внешняя EEPROM данных типа AT24C01A имеет объем 128 байтов. В стенде предусмотрена возможность установки EEPROM большего объема до 32 Кбайт. Взаимодействие с микроконтроллером осуществляется через интерфейс I²C. EEPROM допускает перезапись до 1 млн раз, дает возможность побайтной и постраничной записи. В текущей конфигурации размер страницы составляет 8 байт.

3.6.3. Программный доступ к I²C-устройству

Краткое описание устройства и функционирования шины I²C приведено в руководстве пользователя стенда SDK-1 [4]. Для выполнения элементарных рутинных операций взаимодействия с шиной I²C подготовлен набор подпрограмм на языке ассемблера, оформленный в объектную библиотеку **`I2C.LIB`**. Эту библиотеку необходимо подключить к своему проекту (п. 3.2.1). Отметим, что следует различать I²C-адрес устройства, подключенного к шине I²C (табл. 3.7), и адреса внутренних регистров устройства, находящиеся во внутреннем адресном пространстве этого устройства. Библиотека **`I2C.LIB`** предназначена для организации доступа во внутреннее адресное пространство размером не более 256 адресов: т.е. она рассчитана на 8-разрядный внутренний адрес. Микроконтроллер при этом функционирует в режиме ведущего (master). Приведем описание трех процедур из библиотеки I2C.LIB, используемых при доступе в регистры внутреннего адресного пространства ведомых периферийных устройств, подключенных к шине I²C. Все процедуры модифицируют флаг F0.

GetAck – осуществляет проверку готовности (пинг) ведомого (slave) I²C-устройства к обмену данными. С этой процедуры следует начинать любое обращение к I²C-устройству. Спецификация входных параметров в регистрах микроконтроллера:

A – I²C-адрес устройства.

Результат проверки готовности фиксируется в виде состояния флага F0: 0 – устройство не готово; 1 – устройство готово к обмену.

ReceiveBlock – позволяет получить блок данных от ведомого (slave) I²C-устройства с 8-битным внутренним адресным пространством. Блок данных при получении размещается в резидентной памяти данных (idata). Спецификация входных параметров в регистрах микроконтроллера:

A – I²C-адрес устройства;

B – длина блока данных, т.е. количество получаемых байтов;

R0 – адрес начала блока-источника во внутреннем адресном пространстве I²C-устройства;

R1 – адрес области в `idata`, где будет размещен принятый блок.

При успешном завершении процедуры приема блока данных флаг F0 сброшен. При любой ошибке во время приема `F0` = 1. По значению флага `F0` возможна организация обработчика ошибки. Если обработка ошибки не планируется, то анализ состояния флага можно исключить.

SendBlock – позволяет передать блок данных в ведомое (slave) I²C-устройство с 8-битным внутренним адресным пространством. Блок данных для отправки должен быть размещен в резидентной памяти данных (idata). Спецификация входных параметров в регистрах микроконтроллера:

A – I²C-адрес устройства;

B – длина блока данных, т.е. количество байтов для передачи;

R0 – адрес начала блока-приемника во внутреннем адресном пространстве I²C-устройства;

R1 – адрес области в `idata`, где размещен блок данных для пересылки.

При успешном завершении процедуры приема блока данных флаг `F0` сброшен. При любой ошибке во время приема `F0` = 1.

Приведем примеры программ, осуществляющих чтение и установку времени в часах. Директива **EXTRN** декларирует, что программы (code) с именами «receiveblock», «sendblock», «getack» содержатся в другом файле. В данном случае они содержатся в объектной библиотеке I2C.LIB. Директивы **SEGMENT** и **RSEG** служат для оформления сегмента с именем **PROG**, в котором будет размещена приведенная ниже программа. Подробное описание данных директив приведено в [1]. Отметим, что подобное декларирование структуры программы требуется редактору связей для сборки загрузочного модуля из нескольких объектных модулей: в данном случае для объединения приведенной ниже программы и библиотеки I2C.LIB.

В подпрограммах GetTime (получение времени) и PutTime (установка времени) использованы описанные выше процедуры доступа к I²C-устройствам. Программы выдают содержимое регистров в «сыром» виде без какого-либо преобразования или форматирования.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Пример программы для получения времени
; (4-байта) из часов по шине I2C (или для установки
; времени в часах отправкой 4 байтов по шине I2C)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Ф-ии из библиотеки I2C.LIB
        extrn   code (receiveblock,sendblock,getack)
prog    segment code     ; Декларация сегмента прог.
        rseg    prog     ; Выбор текущего сегмента
        org     2050h    ; Установка адреса сегмента
Begin:  mov     r1,#40h
        call    gettime  ; Получить 4 байта из часов
        call    puttime  ; Отправить 4 байта в часы
        jnb     F0,done  ; F0=0, нет ошибки
; Здесь можно разместить обработчик ошибки при F0=1
done:   sjmp    $        ; Конец основной программы
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Подпрограмма GetTime
; Получение времени в виде 4 байтов ДДК из часов
; r1 -- адрес в idata для размещения 4 байтов
; Результат в F0:
; 0 -- успешно получено; 1 - часы не откликаются
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
gettime:
        push    acc      ;
        push    b        ;
; Выбор банка 0, F0=0
        anl     psw,#01100011b
        push    0h       ; Сохранение r0
        mov     a,#0a0h  ; I2C-адрес часов
        call    getack   ; F0=1 -- часы готовы
        cpl     F0       ; Иначе -- не готовы
        jb      F0,time_err
        mov     b,#4     ; Длина блока байтов
        mov     r0,#1h   ; Адрес рег. в часах
        call    receiveblock
time_err:
        pop     0h       ;
        pop     b        ;
        pop     acc      ;
        ret              ;
;Конец подпрограммы GetTime
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Подпрограмма PutTime
; Установка времени в виде 4 байтов ДДК кода
; r1 -- адрес области в idata исходных 4 байтов
; Результат в F0:
; 0 -- успешно установлен; 1 -- часы не откликаются
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
puttime:
        push  acc              ;
        push  b                ;
        anl   psw,#01100011b   ; выбор банка 0, F0=0
        push  0h               ; сохранение r0
        mov   a,#0a0h          ; I2C-адрес часов
        call  getack           ; F0=1 -- часы готовы
        cpl   F0               ; иначе -- не готовы
        jb    F0,put_err_      ; обработка ошибки
        mov   b,#4             ; длина блока байтов
        mov   r0,#1h           ; адрес рег. в часах
        call  sendblock        ; передать в часы
put_err_:
        pop   0h               ;
        pop   b                ;
        pop   acc              ;
        ret                    ;
        end                    ;
;Конец подпрограммы PutTime
```

3.7. Примерные темы и порядок выполнения работ

В данном разделе приведены примерные темы лабораторных работ, проводимых с использованием стенда SDK-1, а также порядок выполнения работ и методические указания к ним.

Лабораторная работа 1. Доступ к периферийным устройствам через регистры ПЛИС (8 ч)

Цель работы – изучить технологию и получить навыки программирования периферийных устройств стенда SDK-1, доступных через регистры ПЛИС. Продолжительность работы – 8 ч. Работа рассчитана на два занятия по 4 часа. Первое занятие включает в себя задания с 1 по 4, а второе занятие – задания с 5 по 7.

Порядок выполнения лабораторной работы 1

1. Изучить особенности организации лабораторного стенда SDK-1 (п. 3.1).

2. Провести предварительные операции по созданию рабочей директории и открытию проекта в интегрированной среде Keil μVision (п. 3.2.1).

3. Программа для вывода информации на линейку светодиодных индикаторов: открыть проект, набрать программу (п. 3.4.1) и подпрограмму PUTBYTE. Разобраться в организации программы, провести ассемблирование, получить HEX-файл. Используя монитор T2 (п. 3.2.2), загрузить программу в лабораторный стенд SDK-1 и получить правильный результат выполнения программы. В исходном тексте изменить содержимое выводимого байта данных и пронаблюдать результат на лабораторном стенде.

Предостережение: подпрограммы в файле следует всегда располагать после головной программы. Это избавит от возможной проблемы со стартовым адресом программы.

4. Программа для вывода информации на жидкокристаллический дисплей. Изучить организацию подпрограммы PUTCHAR и технологию управления жидкокристаллическим индикатором (п. 3.4.2). Открыть новый проект, набрать подпрограмму PUTCHAR, составить и набрать головную программу для вывода своего имени или фамилии на ЖК-дисплее с использованием подпрограммы PUTCHAR. Дисплей имеет встроенный знакогенератор, поддерживающий символы в ASCII-кодах и кириллицу в нестандартной кодировке (см. стр. 40 Руководства пользователя [4]). Ассемблировать программу и получить результат. Изучить команды управления курсором дисплея и вывести текст во вторую строчку ЖК-дисплея.

5. Опрос матричной клавиатуры (п. 3.4.3). Открыть новый проект, набрать подпрограмму GETKEY, составить головную программу для опроса клавиатуры с помощью подпрограммы GETKEY и вывода кода нажатой клавиши на светодиодные индикаторы. Составить таблицу кодов клавиатуры.

6. Звуковой излучатель (п. 3.4.4). Открыть новый проект, набрать программу генерации звука и получить звук. Программно изменить громкость и высоту тона (частоту).

7. Программирование УАПП (п. 3.5). Набрать программу для инициализации УАПП UART_INI, набрать приведенную в пособии головную программу, принимающую байт с линии, передающую этот байт обратно в линию и на светодиодные индикаторы (эхо-печать). Изучить работу монитора T2 в режиме эмуляции терминала (п. 3.2.3).

Переключив монитор T2 в режим эмуляции терминала, наблюдать передачу данных с клавиатуры компьютера (при нажатии клавиш) через стенд SDK-1 на экран компьютера. Провести данный эксперимент в двух режимах: при отображении принятых данных в бинарном и шестнадцатеричном кодах. Изучить режим работы T2 при записи в файл информации, приходящей со стенда (п. 3.2.3). Записать эхо-печать в текстовый файл. Выйдя из монитора T2 (команда bye), просмотреть содержимое этого текстового файла средствами программы Блокнот (Notepad.exe).

Лабораторная работа 2. Программирование периферийных устройств, доступных через регистры ПЛИС (4 ч)

Цель работы – закрепить полученные навыки программирования периферийных устройств стенда SDK-1, доступных через регистры ПЛИС. Продолжительность работы – 4 ч. Работа содержит индивидуальные задания.

Варианты заданий к лабораторной работе 2

1. Модификация подпрограммы GETKEY: используя возможности битового процессора микроконтроллера, составить подпрограмму для преобразования четырехбитного кода нажатой клавиши в четырехбитный код, в точности соответствующий двоичному представлению изображенного на клавише символа (0…F). Использовать не табличное преобразование кода, а синтез рассчитанной переключательной функции.

2. Модификация подпрограммы GETKEY: используя возможности битового процессора микроконтроллера, модифицировать подпрограмму GETKEY так, чтобы четырехбитовые компоненты ROW и COL преобразовывались в четырехбитовый код нажатой клавиши, в точности соответствующий двоичному представлению изображенного на клавише символа (0…F). Использовать не табличное преобразование кода, а синтез рассчитанной переключательной функции.

3. Счетчик внешних событий: составить программу для подсчета внешних событий и отображения состояния счетчика на ЖК-дисплее. Внешнее событие: нажатие на заданную клавишу клавиатуры лабораторного стенда. Отображение информации в десятичном коде на двух первых знакоместах первой строки ЖК-дисплея. Число внешних событий – не более 99d. Указание: при использовании подпрограммы опроса клавиатуры необходимо контролировать не только нажатие заданной клавиши, но и ее отжатие (возврат в исходное состояние).

Кроме того, необходимо помнить о проблеме устранения «дребезга» контактов. Решение проблемы достигается программным способом.

4. Счетчик внешних событий: составить программу для подсчета внешних событий и отображения состояния счетчика на ЖК-дисплее. Внешнее событие: нажатие на заданную клавишу клавиатуры компьютера. Отображение информации в десятичном коде на двух первых знакоместах второй строки ЖК-дисплея. Число внешних событий – не более 99D.

5. Модификация программы PUTCHAR: составить подпрограмму для вывода строки текстовой информации на верхнюю строчку ЖК-дисплея, начиная всегда с первого знакоместа. Строка информации для вывода расположена во внешней памяти контроллера, адрес первого байта задан в регистре DPTR, признак конца строки – символ '$'. Особенность: перед выводом каждого последующего символа строки необходимо проверять готовность ЖК-дисплея к приему нового байта данных. В регистре состояния ЖК-дисплея поддерживается специальный флаг занятости, который необходимо опрашивать в режиме чтения дисплея. Обратите внимание, что режим чтения возможен только при высоком уровне строба E.

6. Составить программу, которая после запуска выводит символ '+' на первое знакоместо верхней строки ЖК-дисплея. Далее этот символ можно перемещать по знакоместам ЖК-дисплея при нажатии на клавиши: [#] – перемещение на одно знакоместо вправо; [*] – перемещение на одно знакоместо влево; [0] – перемещение на строчку вниз; [8] – перемещение на строчку вверх. Указание: при использовании подпрограммы опроса клавиатуры необходимо контролировать не только нажатие заданной клавиши, но и ее отжатие (возврат в исходное состояние). Кроме того, необходимо помнить о проблеме устранения «дребезга» контактов. Решение проблемы достигается программным способом.

7. Составить программу, которая после запуска ждет нажатия клавиш: [#] – перемещение на одно знакоместо вправо; [*] – перемещение на одно знакоместо влево; [0] – перемещение на строчку вниз; [8] – перемещение на строчку вверх. При первом нажатии на [#] в текущем знакоместе печатается «стрелка вправо» (→), при каждом последующем нажатии на [#] «стрелка вправо» перемещается на одну позицию вправо. Аналогично для клавиш [*], [0] и [8], только в соответствующем направлении. Указание: при использовании подпрограммы опроса клавиатуры необходимо контролировать не только нажатие заданной клавиши, но и ее отжатие (возврат в исходное состояние). Необходимо также помнить о проблеме устранения «дребезга» контактов.

8. Составить программу, которая после запуска ждет нажатия клавиш на клавиатуре лабораторного стенда: на клавиатуре можно набрать пару шестнадцатеричных цифр (код), затем после нажатия клавиши [#] на ЖК-дисплее отображается символ, соответствующий этому коду по таблице знакогенератора. Далее можно набирать новый код, который отобразится в следующем знакоместе ЖК-дисплея. И так – до заполнения строки. Пример: набираем [4], [B], [#] – при этом на ЖК-дисплее загорелся символ 'K'. Указание: при использовании подпрограммы опроса клавиатуры необходимо контролировать не только нажатие заданной клавиши, но и ее отжатие (возврат в исходное состояние). Кроме того, необходимо помнить о проблеме устранения «дребезга» контактов.

9. Составить программу опроса АЦП с выводом байта кода на светодиодные индикаторы. В качестве источника сигнала использовать программу вывода на ЦАП. На стенде есть переключатель, соединяющий выход ЦАП со входом АЦП.

10. Составить программу вывода на ЦАП кода, поступающего из компьютера по последовательному каналу (T2 в режиме эмуляции терминала). Результат работы ЦАП контролировать с помощью программы опроса АЦП. На стенде есть переключатель, соединяющий выход ЦАП со входом АЦП.

11. Модифицировать программу работы с УАПП путем использования режима прерываний. На компьютере работает программа T2 в режиме эмуляции терминала, ASCII коды нажатых на компьютере клавиш передаются на лабораторный стенд по последовательному каналу, УАПП настроен для работы по прерываниям. Подпрограмма обслуживания прерываний от УАПП принимает байт из SBUF, выводит его на светодиодную линейку и издает короткий звуковой сигнал. Основная программа находится в режиме бесконечного цикла.

12. Модифицировать программу работы с УАПП путем использования режима прерываний. На компьютере работает программа T2 в режиме эмуляции терминала, содержимое заданной области памяти лабораторного стенда передается на компьютер по последовательному каналу, УАПП настроен для работы по прерываниям. Подпрограмма обслуживания прерываний от УАПП передает байт в BUF и проверяет, не достигнут ли конец массива. Если конец массива достигнут, то выдается короткий звуковой сигнал и зажигаются светодиоды. Основная программа находится в режиме бесконечного цикла.

13. Модифицировать программу генерации звука так, чтобы задержка формировалась таймером с программным опросом флага переполнения. Заданная частота звука 440 Гц.

14. Модифицировать программу генерации звука так, чтобы задержка формировалась таймером в режиме прерываний. Основная программа находится в режиме бесконечного цикла. Заданная частота звука 1000 Гц.

15. Составить программу «кодовый замок». Секретный код из трех символов набирается на клавиатуре компьютера (Т2 в режиме эмуляции терминала) и передается на лабораторный стенд по последовательному каналу. Программа, работающая на лабораторном стенде, анализирует принятый код: если код правильный, то издается короткий звуковой сигнал и зажигается линейка светодиодов.

16. Составить программу «кодовый замок». Секретный код из трех символов набирается на клавиатуре лабораторного стенда и завершается нажатием клавиши [#]. Программа, работающая на лабораторном стенде, анализирует принятый код: если код правильный, то издается короткий звуковой сигнал, зажигается линейка светодиодов и на компьютер по последовательному каналу передается текстовая строка «OK!». На компьютере работает программа Т2 в режиме эмуляции терминала с отображением принятых байтов в бинарном коде.

17. Программа отображает на ЖК-дисплее символы (цифры, знаки, буквы латиницы), приходящие через УАПП, работающий в режиме прерываний. На компьютере работает монитор Т2 в режиме эмуляции терминала. Символы набираются на клавиатуре компьютера, а отображаются на ЖК-дисплее стенда. Особенность: для отображения символов на ЖК-дисплее использовать обе строки. Сначала заполняется верхняя строка, затем нижняя.

18. Программа отображает на дисплее компьютера символы (цифры, знаки, буквы латиницы), приходящие через УАПП, работающий в режиме прерываний. На компьютере работает монитор Т2 в режиме эмуляции терминала. Символы набираются на клавиатуре стенда, а отображаются на дисплее компьютера в окне терминала Т2.

19. Отобразить на ЖК-дисплее в двоичном виде состояние 8 битов внешнего порта, подключенных к микропереключателям стенда. Опрос порта делать по прерываниям с периодом 120 миллисекунд.

20. Бегущие огни на линейке светодиодов с периодом перескоков 0.5 с. Использовать трехбайтовую неповторяющуюся последовательность и таймер в режиме прерываний для задания временного интервала.

21. Монитор ячеек памяти резидентной памяти программ: в верхней строке выводить адрес с префиксом D:0x0, а в нижней – содержимое ячейки памяти в шестнадцатеричном коде. Перемещение по адресам памяти с помощью клавиш стенда '*' и '#'.

Лабораторная работа 3. Доступ к периферийным устройствам через шину I2C (4 ч)

Цель работы – получить навыки программирования периферийных устройств стенда SDK-1, подключенных к микропроцессорной системе по шине I^2C.

Порядок выполнения лабораторной работы 3

1. Теоретический материал по организации шины I2C (см. руководство пользователя стенда SDK-1 и п. 3.6).
2. Изучить технологию подключения объектных библиотек на примере библиотеки I2C.LIB.
3. Изучить и набрать библиотечные (I2C.LIB) подпрограммы GetAck, ReceiveBlock и SendBlock.
4. На примере программ GetTime и PutTime изучить технологию доступа к часам реального времени и разобраться с форматом (4 байта) представления времени в часах.
5. Используя PutTime, установить в часах текущее время.
6. Написать программу, которая на основе GetTime читает содержимое часов (4 байта) и выводит на экран компьютера средствами T2 в режиме эмуляции терминала. Здесь необходимо использовать созданные в лабораторной работе N1 программы работы с УАПП. Добиться вывода содержимого часов на экран компьютера и убедиться, что время соответствует заданному.

Лабораторная работа 4. Программирование периферийных устройств, доступных через шину I2C (4 ч)

Цель работы – закрепить навыки программирования периферийных устройств стенда SDK-1, подключенных к микропроцессорной системе по шине I2C. Работа содержит индивидуальные задания.

Варианты заданий к лабораторной работе 4

Задание 1. Спроектировать электронный секундомер на основе БИС часов-календаря, входящей в состав лабораторного стенда: старт-стопный счетчик секунд с отображением результата в виде двухразрядного десятичного числа. Интервал счета: от нуля до «Nmax». Сигналы ПУСК/СТОП – нажатие на клавишу «КЛАВИША» на клавиатуре компьютера (комп. – T2 в режиме эмуляции терминала), либо на клавиатуре лабораторного стенда (стенд). Отображение – на устройстве

«УСТРОЙСТВО ОТОБРАЖЕНИЯ», в качестве которого выступает либо дисплей компьютера (Т2 в режиме эмуляции терминала), либо ЖК-дисплей лабораторного стенда.

Таблица 3.8. Варианты индивидуального задания 1

Варианты	Nmax	Клавиша	Устройство отображения
1	59	z комп.	Компьютер
2	59	# стенд	Компьютер
3	59	z комп.	ЖКД
4	59	# стенд	ЖКД
5	99	w комп.	Компьютер
6	99	* стенд	Компьютер
7	99	w комп.	ЖКД
8	99	* стенд	ЖКД

Задание 2. Спроектировать часы на основе БИС часов-календаря, входящей в состав лабораторного стенда: вывод часов и минут, разделенных символом «СИМВОЛ» на ЖК-дисплее в позиции «ПОЗИЦИЯ», используя 12- или 24-часовую шкалу «ШКАЛА» и будильник «БУДИЛЬНИК». Разделительный символ ’-’ статический, а ’:’ динамический. Динамический символ мигает (тикает), отображая ход секунд.

Таблица 3.9. Варианты индивидуального задания 2

Варианты	Символ	Позиция	Шкала	Будильник
1	-	0	24	ЕСТЬ
2	-	40	12	НЕТ
3	-	Справа вверху	24	НЕТ
4	-	Справа внизу	12	НЕТ
5	:	0	24	НЕТ
6	:	40	12	НЕТ
7	:	Справа вверху	24	НЕТ
8	:	Справа внизу	12	НЕТ
9	:	Посредине внизу	12	НЕТ
10	:	Посредине внизу	24	НЕТ

Лабораторная работа 5. Практическая реализация индивидуального домашнего задания на стенде SDK-1 (4 ч)

Цель работы – практическая реализация выполненного ранее индивидуального домашнего задания на стенде SDK-1.

Порядок выполнения лабораторной работы 5

Индивидуальное домашнее задание, выполненное в п. 2, необходимо реализовать на лабораторном стенде SDK-1. При этом может потребоваться некоторая корректировка разработанной ранее программы. Причина этого – отсутствие в лабораторном стенде свободных параллельных портов P0–P3 микроконтроллера. Один из возможных путей такой корректировки – замена параллельного порта, фигурирующего в индивидуальном домашнем задании, на последовательный порт УАПП микроконтроллера. Программу-монитор T2 в режиме эмуляции терминала при этом можно использовать как внешний источник входного потока данных и внешний приемник выходного потока данных. Индивидуальное домашнее задание оформляется в виде подпрограммы. Для ее испытания необходимо разработать головную программу, задача которой состоит в подготовке входных данных, вызове подпрограммы, передаче выходных данных на терминал. Далее необходимо подготовить тестовый пример и выполнить его на лабораторном стенде. Написать отчет, к отчету приложить пояснительную записку к индивидуальному домашнему заданию.

4. СТАНДАРТНЫЕ БИБЛИОТЕКИ KEIL μVISION

В данном разделе рассмотрены общие вопросы применения стандартных библиотек Keil μVision при программировании на языке ассемблера A51 (ASM-51), приведено описание форматов целых и вещественных чисел, основных библиотечных функций компилятора C51 и особенностей обращения к ним из программы, написанной на ассемблере, приведены примеры программ.

4.1. Общие положения

Восьмибитовые контроллеры платформы x51 не имеют встроенных машинных команд для непосредственного манипулирования многобайтовыми данными. Для разработки программного обеспечения широко используются стандартные библиотеки, поддерживающие математические (арифметические) операции над целыми и вещественными числами. В этом случае структура, форматы и порядок размещения многобайтных данных уже предопределены внутренней организацией используемой библиотеки.

В состав интегрированной среды разработки Keil μVision входят библиотеки подпрограмм для работы с целыми и вещественными многобайтовыми данными. Доступ к этим библиотекам может быть осуществлен как из программ, написанных на языке ассемблера, так и из программ, написанных на языках высокого уровня (C51, PL/M и т.п.).

Возможность доступа к библиотекам из программ, написанных на языке ассемблера, является очень важным фактором при разработке прикладных программ. Использование стандартных библиотек не только экономит время на разработку программ, но и унифицирует технологию выполнения математических (арифметических) операций над целыми и вещественными числами в микропроцессорной системе. Широкая распространенность стандартных библиотек Keil μVision позволяет рассматривать их как своеобразный технический стандарт, который является важным элементом инженерной подготовки в области микропроцессорной техники.

4.2. Архитектурно-зависимые особенности

Типы памяти. Типы памяти определяются особенностями архитектуры микроконтроллеров платформы x51.

`CODE` – память программ для чтения объемом до 64 Кбайт. Память типа code может быть как резидентной, так и внешней.

`DATA` – резидентная память данных, доступная по прямым адресам от 0 до 127.

`IDATA` – резидентная память данных, доступная по косвенным адресам от 00H до FFH. Младшие 128 адресов относятся к тем же самым ячейкам памяти, что и в DATA. Косвенный доступ обычно медленнее прямого.

`BDATA` – бит-адресуемая память, расположенная в байтовой области РПД от 20H до 2FH. Предоставляет доступ к отдельным битам по специальным адресам битовой памяти от 00H до 7FH.

`XDATA` – внешняя память данных объемом до 64 Кбайт.

`PDATA` – одна страница (256 байт) внешней памяти данных.

`FAR` и `CONST FAR` – тип внешней памяти для новых моделей x51, использующих трехбайтовый регистр-указатель. Области памяти `far` и `const far` относятся к ОЗУ и ПЗУ соответственно. Объем памяти до 16 Мбайт. Пример таких моделей – микроконвертеры ADuC8xx.

`SFR` – область памяти, относящаяся к регистрам специальных функций 08H…FFH.

Модели памяти. Модель памяти определяет типы памяти, используемые по умолчанию в программе для аргументов функций, автоматических переменных и т.п. Три предопределенные модели памяти:

`SMALL` – все переменные размещаются в резидентной памяти данных (data). Доступ самый быстрый, но все объекты, включая стек, должны разместиться в резидентной памяти данных. Размер стека здесь – самый критичный параметр. Если редактор связей (компоновщик) настроен на режим оверлея переменных в резидентной памяти данных, то эта модель памяти является наилучшим выбором.

`COMPACT` – все переменные располагают на одной странице внешней памяти данных (pdata). Объем страницы составляет 256 байтов. Это обусловлено использованием косвенной адресации через 8-битовые регистры R0 или R1 (R0, R1).

`LARGE` – все переменные располагают во внешней памяти объемом до 64 Кбайт (xdata). Для косвенной адресации используется регистр DPTR. Самая неэффективная модель, как по быстродействию, так и по объему генерируемого кода.

Порядок следования байтов в многобайтовых данных. К сожалению, не существует единого стандарта на порядок следования (размещения в памяти по возрастанию адресов) байтов. Исторически сложилось несколько альтернативных подходов к размещению байтов в памяти.

1. *Интеловский порядок* (*Intel order* или *little-endian*) – это запись многобайтных данных «младшими байтами вперед». При этом младший байт записывают по самому младшему адресу, остальные байты записывают подряд по возрастанию адресов в порядке возрастания старшинства байтов. Этот порядок записи принят в операционной системе Windows, в персональных компьютерах с x86-процессорами.

Возьмем для примера 32-битное целое значение в шестнадцатеричном коде 57415244H. Это число будет размещено в памяти по возрастанию адресов таким образом:

Адрес	+0	+1	+2	+3
Двоичный	01000100	01010010	01000001	01010111
HEX	44H	52H	41H	57H

Существенным достоинством little-endian по сравнению с другими порядками записи считается возможность «неявной типизации» целых чисел при чтении меньшего объёма байтов (при условии, что читаемое число помещается в диапазон). Так, если в памяти содержится число 00000022H, то прочитав его как двухбайтовое целое, мы получим число 0022H, прочитав один байт – число 22H.

2. *Мотороловский порядок* (*Motorola order* или *big-endian*) – это запись многобайтных данных «старшими байтами вперед». При этом старший байт записывают по самому младшему адресу, остальные байты записывают подряд по возрастанию адресов в порядке убывания старшинства байтов.

Этот порядок является стандартным для протоколов TCP/IP, он используется в заголовках пакетов данных и во многих протоколах более высокого уровня, разработанных для использования поверх TCP/IP. Поэтому порядок байтов от старшего к младшему часто называют *сетевым порядком* байтов (*network byte order*). Этот порядок байтов используется операционной системой UNIX, процессорами IBM 360/370/390, Motorola 68000, SPARC.

Возьмем для примера 32-битовое целое значение в шестнадцатеричном коде 57415244H. Это число будет размещено в памяти по возрастанию адресов таким образом:

Адрес	+0	+1	+2	+3
Двоичный	01010111	01000001	01010010	01000100
HEX	57H	41H	52H	44H

3. *Смешанный порядок* (*middle-endian*) иногда используется при работе с числами, длина которых превышает машинное слово. Производится факторизация числа не на байты, а на машинные слова, которые записываются в формате, естественном для данной архитектуры, но сами слова записываются в обратном порядке.

Классический пример middle-endian – представление четырехбайтовых целых чисел на 16-битовых процессорах семейства PDP-11 (способ известен как «PDP-endian»). Для представления двухбайтовых значений использовался порядок big-endian, но четырехбайтовое двойное слово записывалось от младшего слова к старшему. В процессорах VAX и ARM используется также смешанное представление для длинных вещественных чисел.

Возьмем для примера 32-битовое целое значение в шестнадцатеричном коде 57415244H. Для PDP-endian это число будет размещено в памяти по возрастанию адресов таким образом:

Адрес	+0	+1	+2	+3
Двоичный	01000001	01010111	01000100	01010010
HEX	41H	57H	44H	52H

Термины *big-endian* и *little-endian* первоначально не имели никакого отношения к информатике. В сатирическом произведении Джонатана Свифта «Путешествия Гулливера» описываются вымышленные государства *Лилипутия* и *Блефуску*, в течение многих лет ведущие между собой войны из-за разногласия по поводу того, с какого конца следует разбивать варёные яйца. Тех, кто считал, что яйца нужно разбивать с тупого конца, в произведении называли «Big-endians» («тупоконечники»). Споры между сторонниками big-endian и little-endian в информатике также часто носят характер «религиозных войн».

Порядок следования байтов в контроллерах x51. Для двухбайтовых адресов, используемых некоторыми командами, приняты следующие соглашения:

Команда LCALL запоминает адрес следующей команды (точку возврата – ТВ) в стеке, используя интеловский порядок, т.е. младшими байтами вперед.

Команды LJMP и LCALL содержат двухбайтовый прямой адрес, записанный в мотороловском порядке, т.е. старшими байтами вперед.

Адрес	+0	+1	+2
ТВ в стеке		Младший байт	Старший байт
Команда	LCALL	Старший байт	Младший байт
Команда	LJMP	Старший байт	Младший байт

Восьмибитовые контроллеры платформы x51 не имеют встроенных машинных команд для непосредственного манипулирования многобайтовыми данными. Поэтому не существует никаких принципиальных (в том числе – аппаратных) ограничений на порядок размещения многобайтовых данных при разработке программного обеспечения для контроллеров x51.

Для разработки программного обеспечения широко используются стандартные библиотеки, поддерживающие математические (арифметические) операции над целыми и вещественными числами. В этом случае структура, форматы и порядок размещения многобайтных данных предопределены внутренней организацией используемой библиотеки.

4.3. Стандартная библиотека C51 Keil Software

При программировании на языке C51 все преобразования форматов скрыты от пользователя и осуществляются компилятором с использованием стандартных библиотек. Комплект файлов стандартной библиотеки для компилятора C51, входящей в состав среды μVision фирмы Keil Software, содержит, в частности, шесть файлов для трех различных моделей памяти. Для каждой модели памяти присутствуют два библиотечных файла.

Файл `C51x.lib` представляет собой библиотеку стандартных функций языка C51. В этой библиотеке содержится большое число различных функций, в частности арифметические функции для работы с целыми числами.

Файл `C51FPx.lib` содержит библиотечные функции для выполнения операций над числами с плавающей точкой (ПТ).

`C51S.LIB` – модель `small`;

`C51FPS.LIB` – поддержка арифметики с ПТ для модели `small`;

`C51C.LIB` – модель `compact`;

`C51FPC.LIB` – поддержка арифметики с ПТ для модели `compact`;

`C51L.LIB` – модель `large`;

`C51FPL.LIB` – поддержка арифметики с ПТ для модели `large`.

Полный список всех библиотечных функций для модели памяти `small` приведен в руководстве стандартных библиотек.

Все целые числа со знаком представлены в дополнительном коде (дополнение до двойки).

Все двух- и четырехбайтовые объекты записывают в моторолловском порядке, т.е. старшими байтами вперед.

Битовые переменные. Тип данных `bit`. Битовые переменные хранятся в бит-адресуемой резидентной памяти `bdata`. В библиотеке не поддерживаются битовые массивы и указатели на битовые переменные.

Тип данных `sbit` позволяет декларировать статические битовые переменные по их абсолютным битовым адресам. Таким способом задают, например, индивидуально адресуемые биты регистров SFR.

Однобайтовые двоичные целые со знаком и без знака. Тип данных без знака: `unsigned char`. Диапазон изменения чисел 0…255. Тип `enum` также может входить в эту группу, если он представим в байтовом формате.

Типы данных со знаком: `signed char`. Диапазон изменения чисел −128…+127.

Двухбайтовые двоичные целые со знаком и без знака. Тип данных без знака: `unsigned short`, `unsigned int` и `enum`. Диапазон изменения чисел 0…65535.

Типы данных со знаком: `signed short`, `signed int`. Диапазон изменения чисел −32768…+32767.

Четырехбайтовые двоичные целые со знаком и без знака. Тип данных без знака: `unsigned long`. Диапазон изменения чисел этого формата 0…4294967295.

Типы данных со знаком: `signed long`. Диапазон изменения чисел −2147483648…+2147483647.

Числа с плавающей точкой. Тип данных `float`. Диапазон изменения чисел этого формата:

$$\pm 1.175494 \cdot 10^{-38} \ldots \pm 3.402823 \cdot 10^{+38}.$$

Четырехбайтовое (32 бита) число в стандарте IEEE-754 состоит из двух компонентов: мантиссы (M) и порядка (P)

$$\pm M \times 2^{P}.$$

Порядок (экспонента) – это однобайтовое двоичное целое со знаком (−127…+128), представленное в *смещенном коде*. Смещенный код E числа P, находящегося в диапазоне −127…+128, получают следующим образом:

$$E = P + 127.$$

Мантисса (`M`) – это трехбайтовое (24 бита) число $1 \le |\mathtt{M}| < 2$ с фиксированной двоичной точкой, представленное в прямом коде со знаком.

В нормализованной мантиссе старший (наиболее значимый) бит всегда равен единице, а следом за ним всегда идет двоичная точка. При этом нет никакой необходимости сохранять старший бит и двоичную точку: достаточно сохранить только оставшиеся 23 бита. Это дает до семи значащих десятичных цифр. Высвободившийся бит используют для хранения знака мантиссы (S): 1 – минус, 0 – плюс. На диаграмме биты порядка обозначены буквой `E`, а биты мантиссы – буквой `M`.

Адрес	+0	+1	+2	+3
Формат	SEEEEEEE	EMMMMMMM	MMMMMMMM	MMMMMMMM

Машинный нуль в формате с плавающей точкой представлен наименьшим возможным положительным числом из диапазона `float`: $P = -127$, $M = 1.0$, $S = 0$. Легко видеть, что машинный нуль представлен шестнадцатеричным кодом 00000000H.

Рассмотрим пример представления числа (-12.5) в формате `float`. Вначале запишем число в нормализованном виде:

$$-12.5 = -1.5625 \times 2^3.$$

Очевидно, что порядок $P = 3$, а в смещенном коде $E = 130$, т.е. 10000010B. Преобразуем модуль мантиссы в двоичный код:

$$1.5625 = 1 + 1/2 + 1/16 = 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-4}.$$

Мантисса M = 11001000 00000000 00000000B, знак $S = 1$. Старший бит всегда равен 1, его никогда не сохраняют. Оставшиеся 23 бита мантиссы дают 1001000 00000000 00000000B. Собирая все вместе, получаем, что в формате `float` число -12.5 представлено шестнадцатеричным четырехбайтовым кодом C1480000H.

Адрес	+0	+1	+2	+3
Формат	SEEEEEEE	EMMMMMMM	MMMMMMMM	MMMMMMMM
Двоичный	11000001	01001000	00000000	00000000
HEX	C1	48	00	00

4.4. Интерфейс между C51 и ассемблером

Технология передачи параметров в библиотеках Keil μVision оптимизирована для работы с компилятором C51. В этой связи взаимодействие библиотечных функций и программ, написанных на ассемблере, осуществляется по тем же правилам, что и интерфейс между C51 и ассемблером.

Макроассемблер A51, входящий в состав Keil μVision, при генерации кода формирует объектные модули в формате OMF-51. Соблюдение нескольких простых правил обеспечивает взаимную доступность модулей, подготовленных на C51 и A51. Переменные, декларированные как public в ассемблерной программе, доступны из модулей, подготовленных на C51 как external, и наоборот.

Функция, написанная на C51, по умолчанию передает до трех параметров через регистры микроконтроллера. Если количество параметров больше трех, то остальные параметры передаются через фиксированную область памяти. Директива NOREGPARMS позволяет запретить передачу параметров через регистры. В этом случае все параметры будут переданы только через фиксированную область памяти. Компилятор C51 при генерации кода маркирует функции, передающие параметры через регистры, значком подчеркивания '_' перед именем функции. Функции, передающие параметры только через фиксированную область памяти, не маркируются таким значком. При работе с арифметическими библиотеками C51 обычно используются один или два параметра. Технология передачи параметров через регистры микроконтроллера представлена в табл. 4.1.

Таблица 4.1. Передача параметров функций через регистры

Номер аргумента	char, 1-byte ptr	int, 2-byte ptr	long, float	generic ptr
1	R7	R6_R7	R4–R7	R3–R1
2	R5	R4_R5	R0–R3	R3–R1
3	R3	R2_R3	–	R3–R1

Примечание. Старший байт записан всегда слева. Для generic ptr: R3 – тип памяти, R2_R1 – указатель. R0–R3 – только для библиотечных функций, перечисленных в табл. 4.3.

При передаче параметров через фиксированную область памяти библиотечные функции C51 используют специальные сегменты с именами:

```
?function_name?BYTE          ?function_name?BIT
```

Вместо «function_name» в имена сегментов подставляют имя соответствующей функции. Первый из сегментов используют для передачи байтовых переменных, а второй – для передачи битовых переменных. Для всех параметров резервируют место в этих сегментах, даже если эти параметры передаются через регистры. Параметры размещаются в порядке их декларирования в функции.

Если функция возвращает значение, то оно всегда передается через регистры (табл. 4.2).

Таблица 4.2. Возврат значения функции через регистры

Тип значения	Регистры	Описание
Бит	C	Возврат бита через флаг переноса
char или 1-byte ptr	R7	Возврат байта через R7
int или 2-byte ptr	R6_R7	R6 – старший байт
long или float	R4–R7	R4 – старший байт
generic ptr	R3–R1	R3 – тип памяти, R2 – старший байт

4.5. Арифметические функции библиотеки

4.5.1. Арифметические действия с целыми и вещественными числами

Для операций сложения и вычитания многобайтных целых чисел генерируется простой ассемблерный код на основе команд `ADDC` и `SUBB` соответственно. Для остальных случаев используют соответствующие функции из библиотеки C51 (табл. 4.3). На рис. 4.1, 4.2 и 4.3 приведены примеры вызова библиотечной функции деления чисел в формате с ПТ, а также других функций с одним и двумя аргументами. Обязательным при этом является оформление модуля программы с учетом сегментной структуры, а также объявления функций как `EXTRN` (внешняя функция, вызываемая из данного модуля) и `PUBLIC` (внутренняя функция, которую разрешено вызывать из других модулей). В проекте обязательно должна быть одна головная функция с именем `main`.

Рассмотрим текст программы (рис. 4.1) подробнее. В первой строке директива `SEGMENT` декларирует новый перемещаемый сегмент. Директива содержит два обязательных параметра (имя сегмента и класс памяти), а также может содержать необязательные параметры, управляющие размещением сегмента в памяти. Имя сегмента `?PR?primer`

Таблица 4.3. Библиотечные функции арифметических операций

Тип	Операция	Библиотечная функция	Операнд 1	Операнд 2	Результат
int	×	?C?IMUL	R6_R7	R4_R5	R6_R7
	÷	?C?SIDIV	R6_R7	R4_R5	R6_R7
	÷	?C?UIDIV	R6_R7	R4_R5	R6_R7
long	×	?C?LMUL	R4–R7	R0–R3	R4–R7
	÷	?C?SLDIV	R4–R7	R0–R3	R4–R7
	÷	?C?ULDIV	R4–R7	R0–R3	R4–R7
float	+	?C?FPADD	R4–R7	R0–R3	R4–R7
	–	?C?FPSUB	R4–R7	R0–R3	R4–R7
	×	?C?FPMUL	R4–R7	R0–R3	R4–R7
	÷	?C?FPDIV	R4–R7	R0–R3	R4–R7

Примечание. Старший байт всегда записан слева. Функции деления целых чисел со знаком – `?C?SxDIV`, без знака – `?C?UxDIV`.

выбрано для примера. Можно взять любое другое имя. Приставка `?PR?` (*программа*) в имени сегмента не является строго обязательной, но некоторые библиотечные модули ожидают ее наличия для правильной организации оверлейного процесса. Класс памяти `CODE` соответствует программной памяти (п. 4.2). Отметим, что директива `SEGMENT` только лишь декларирует будущий сегмент и его свойства, но не создает его реально.

Во второй и третьей строках при помощи директивы `EXTRN` декларируют две библиотечные функции (`?C_STARTUP` и `?C?FPDIV`), которые далее будут использованы в нашей программе. Эти функции размещены в библиотеке Keil μVision и являются *внешними* по отношению к модулю (файлу) нашей программы. Описание библиотечной функции `?C?FPDIV` приведено в табл. 4.3. Стартовая функция `?C_STARTUP` выполняет ряд подготовительных действий (очистка памяти, инициализация стека и т.п.), общих для всех библиотечных функций, а затем передает управление головной программе `main`. Библиотечные функции рассчитаны на то, что до обращения к ним стартовая функция уже провела эти подготовительные действия. Игнорирование функции `?C_STARTUP` при работе с библиотечными функциями может привести к негарантированному результату. Функция `?C_STARTUP` имеет абсолютный стартовый адрес `00000H`, который всегда следует использовать для запуска программы пользователя.

В четвертой строке директива `PUBLIC` декларирует, что головная функция программы (функция `main`) доступна для внешних вызо-

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Пример вызова библиотечной функции.
; Деление чисел в формате с ПТ.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Декларация сегмента
?PR?primer
        SEGMENT  CODE
;Стартовые настройки
        EXTRN    CODE (?C_STARTUP)
;Функция библиотеки
        EXTRN    CODE (?C?FPDIV)
        PUBLIC   main          ; Имя головной ф-ии
        RSEG     ?PR?primer    ; Начало сегмента
main:                          ; Начало ф-ции main
        USING    0             ; Выбор 0-банка
        CLR      A             ; Запись нуля в A
;------- a = -12.5; = /C1/48/00/00------------
        MOV      R7,A          ; Младший байт
        MOV      R6,A          ;
        MOV      R5,#048H      ;
        MOV      R4,#0C1H      ; Старший байт
;------- b = +2.0 = /40/00/00/00--------------
        MOV      R3,A          ; Младший байт
        MOV      R2,A          ;
        MOV      R1,A          ;
        MOV      R0,#040H      ; Старший байт
;------- вызов функции деления -12.5/2.0------
        ACALL    ?C?FPDIV      ;
; Результат деления в регистрах /R4/R5/R6/R7
; На месте первого операнда /C0/C8/00/00/
        SJMP     $             ; Конец функции main
        END                    ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Рис. 4.1. Вызов библиотечной функции деления чисел в формате с ПТ

вов из других модулей. Это необходимо, в частности, чтобы функция `?C_STARTUP` по окончании своей работы могла передать управление нашей головной программе `main`. Имя головной программы фиксировано и не может быть изменено. В проекте обязательно должна быть

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Пример вызова библиотечной функции.
; Вызов функции с одним аргументом.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Декларация сегмента
?PR?primer
       SEGMENT CODE
;Стартовые настройки
       EXTRN   CODE (?C_STARTUP)
;Функция библиотеки
       EXTRN   CODE (_sin)
       PUBLIC  main        ; Имя головной ф-ции
       RSEG    ?PR?primer  ; Начало сегмента
main:                      ; Начало ф-ции main
       USING   0           ; Выбор 0-банка
;------- x = Pi/2; /3F/C9/0E/56/---------------
       MOV     R7,#056H    ;
       MOV     R6,#0EH     ;
       MOV     R5,#0C9H    ;
       MOV     R4,#03FH    ;
;------- y = sin (x)-----------------------------
       ACALL   _sin        ;
; Результат в регистрах /R4/R5/R6/R7
; На месте первого операнда /3F/80/00/00/
       SJMP    $           ; Конец функции main
       END                 ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Рис. 4.2. Вызов библиотечной функций с одним аргументом

одна функция с именем main, которая и будет головной функцией.

В пятой строке директива RSEG назначает текущим сегментом перемещаемый сегмент с именем ?PR?primer. Декларация действует до следующего применения директивы RSEG. Отметим, что свойства сегмента ?PR?primer были описаны ранее директивой SEGMENT.

Далее идет текст головной программы main, в которой согласно табл. 4.3 производится подготовка двух операндов и вызов библиотечной функции деления ?C?FPDIV. Результат получается на месте первого операнда.

При отсутствии операционной системы выход из головной про-

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Пример вызова библиотечной функции.
; Вызов функции с двумя аргументами.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Декларация сегмента
?PR?primer
       SEGMENT CODE
;Стартовые настройки
       EXTRN   CODE (?C_STARTUP)
;Функция библиотеки
       EXTRN   CODE (_pow)
;Область памяти для передачи второго аргумента
       EXTRN   DATA (?_pow?BYTE)
       PUBLIC  main         ; Имя головной ф-ции
       RSEG    ?PR?primer   ; Начало сегмента
main:                       ; Начало ф-ции main
       USING   0            ; Выбор 0-банка
;------- x = 3.1415; /40/49/0E/56/-------------
       MOV     ?_pow?BYTE+07H,#056H
       MOV     ?_pow?BYTE+06H,#0EH
       MOV     ?_pow?BYTE+05H,#049H
       MOV     ?_pow?BYTE+04H,#040H
;------- x = 2.5; /40/20/00/00/----------------
       MOV     R7,#00H      ; Младший байт
       MOV     R6,#00H      ;
       MOV     R5,#020H     ;
       MOV     R4,#040H     ; Старший байт
;------- y = pow (2.5,x) -----------------------
       ACALL   _pow         ;
; Результат в регистрах /R4/R5/R6/R7
; На месте первого операнда /41/8E/4D/F2/
       SJMP    $            ; Конец функции main
       END                  ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Рис. 4.3. Вызов библиотечной функций с двумя аргументами

граммы не предусмотрен. Программа должна быть продолжена дальше или каким-либо образом остановлена. Один из вариантов завершения головной программы – бесконечная петля sjmp $.

Библиотека `C51FPx.lib` позволяет также проводить вычисление элементарных функций вещественных аргументов. Угловые переменные в тригонометрических функциях задаются в радианах. Большинство элементарных функций имеет только один аргумент. Такие функции называются *унарными* функциям. Вызов унарных функций происходит почти так же, как в предыдущем случае (рис. 4.2). Первый аргумент функции (четырехбайтовое вещественное число) размещают в регистрах R4–R7, однако второй аргумент при этом не используется и отсутствует. Значение функции получают на месте первого операнда после ее вызова. К функциям с одним аргументом относятся:

_EXP – экспонента (e^x);

_LOG, _LOG10 – натуральный и десятичный логарифмы;

_SQRT – квадратный корень ($\sqrt{x}$);

_SIN, _COS, _TAN – тригонометрические функции;

_SINH, _COSH, _TANH – гиперболические функции;

_ACOS, _ASIN, _ATAN – обратные тригонометрические функции.

В функциях с двумя аргументами первый операнд передают как в предыдущем случае, а второй операнд размещают в четырех байтах специально выделенной области памяти (рис. 4.3), используя мотороловский порядок байтов:

`?function_name?BYTE+04H... ?function_name?BYTE+07H`

Результат получают на месте первого операнда. К функциям с двумя аргументами (*бинарным* функциям) относятся:

_POW – показательная функция (a^x), первый аргумент – основание (a), второй – показатель степени (x);

_ATAN2 – арктангенс отношения двух величин (x/y), первый аргумент – (x), второй – (y).

4.5.2. Преобразование форматов чисел

Библиотека `C51FPx.lib` содержит в своем составе также функции для преобразования целых чисел в вещественные и, наоборот, вещественных чисел в целые. Отметим, что подобные преобразования могут приводить к потере точности вследствие округления чисел. Особенно при преобразовании вещественных чисел в целые. В табл. 4.4 приведены некоторые функции для преобразования форматов целых и вещественных чисел из библиотеки C51.

Преобразование знаковых и беззнаковых чисел осуществляется одними и теми же функциями. Однако при преобразовании целых чисел в вещественные аккумулятор перед вызовом функции должен быть установлен в определенное значение:

(A)← 00H – для unsigned char, unsigned int, unsigned long;
(A)← (R4) – для signed char, signed int, signed long.

Преобразование вещественных чисел в целые не требует такой предварительной подготовки аккумулятора.

Приведенный список функций не является полным. В стандартных библиотеках содержится множество других функций, описанных в руководстве программиста C51.

4.5.3. Особенности работы с библиотеками на стенде SDK-1

Описание стартового механизма. Стандартные библиотеки Keil μVision содержат библиотечные функции в объектном формате, подготовленные для компоновки в перемещаемые модули. Это означает, что в библиотечных функциях в большинстве случаев изначально отсутствует привязка к каким-либо абсолютным адресам: они могут быть размещены компоновщиком (*Linker*, *редактор связей*) в любом подходящем месте резидентной или внешней памяти программ. Управление размещением модулей может быть осуществлено различными путями как через директивы ассемблера (макроассемблера), так и через настройки компоновщика.

Директивы `SEGMENT` и `RSEG` на рис. 4.1–4.3 как раз необходимы для оформления перемещаемого модуля программы пользователя. Вызываемые библиотечные функции библиотек оформлены с помощью таких же директив и точно так же компонуются в отдельные перемеща-

Таблица 4.4. Библиотечные функции преобразования форматов чисел

Преобразование	Библиотечная функция	Операнд	Результат
`(char)` → `(float)`	`?C?FCASTC`	R4	R4–R7
`(int)` → `(float)`	`?C?FCASTI`	R4_R5	R4–R7
`(long)` → `(float)`	`?C?FCASTL`	R4–R7	R4–R7
`(float)` → `(char)`	`?C?CASTF`	R4–R7	R7
`(float)` → `(int)`	`?C?CASTF`	R4–R7	R6_R7
`(float)` → `(long)`	`?C?CASTF`	R4–R7	R4–R7

Примечание. Старший байт всегда записан слева.

емые модули, которые компоновщик размещает в памяти программ. Компоновщик стремится размещать перемещаемые модули в памяти программ подряд по самым младшим свободным адресам. В этой связи в перемещаемом модуле программы пользователя не рекомендуется использовать какие-либо директивы, осуществляющие привязку к абсолютным адресам, например, директиву ORG. В противном случае модуль программы пользователя будет «оторван» от остальной части программы. Например, указание `org 2000h` в программе пользователя приведет к размещению этого модуля в указанной области памяти программ, тогда как библиотечные функции могут оказаться размещенными, начиная с адреса 0000H. Справедливости ради отметим, что даже в таких случаях компоновщик пытается все же выправить ситуацию и разместить модули программы максимально компактно. Однако результат при этом может быть негарантированным.

Одна из причин этого – «привязка» стартовой функции `?C_STARTUP` к абсолютному адресу. Такая привязка необходима для фиксации стартового адреса: любая программа запускается с адреса 0000H через стартовую функцию `?C_STARTUP`. С точки зрения внутренней организации стартовая функция состоит из двух частей: абсолютного и перемещаемого модулей.

Абсолютный модуль оформлен директивой `CSEG AT 0H` и всегда размещается с нулевого адреса памяти программ. Обычно он состоит из единственной команды `LJMP STURTUP1`, занимающей три байта памяти.

Все остальные команды стартовой функции находятся в перемещаемом модуле, который доступен из абсолютного модуля по символическому адресу `STURTUP1` и размещается компоновщиком в любом подходящем месте памяти программ.

Практические рекомендации. Исходный текст стартовой функции `?C_STARTUP` находится в библиотечном архиве в файле `STARTUP.A51` и доступен для редактирования. Это дает возможность осуществить корректную настройку для работы с библиотеками в конкретных условиях.

В качестве примера оптимизируем настройки Keil μVision для корректной работы с библиотеками на лабораторном стенде SDK-1.

1. В настройках инструментальной среды Keil μVision для нашего проекта ‘Target 1’ укажем начальный адрес внешней памяти SRAM лабораторного стенда `2000H`. Для этого на вкладке

`Project\Options for 'Target 1'\Target`

в окне `Off-Chip Code memory/Start` наберем 0x2000. Как видно из примера, адреса на вкладке набираются не в *постфиксном* виде как

в ассемблере, а в *префиксном* виде как в языке C51. Отметим, что на этой же вкладке в окне `Off-Chip X-data memory/Start` можно при необходимости задать начальный адрес `XDATA` для размещения сегментов данных. На другой вкладке

```
Project\Options for 'Target 1'\BL51 Locate
```

следует выбрать `Use Memory Layout from Target Dialog`.

2. В копии файла `STARTUP.A51` необходимо заменить директиву `CSEG AT 0H` на директиву `CSEG AT 2000H`. Отредактированный файл `STARTUP.A51` можно подключить к проекту одним из трех возможных способов.

Во-первых, этот файл можно ассемблировать до объектного формата и заменить им исходную версию функции `?C_STARTUP` в библиотеке. Однако мы не рекомендуем этот путь, так как в этом случае библиотеки Keil μVision перестанут быть *стандартными* библиотеками и появится проблема с переносимостью программ.

Во-вторых, к проекту можно прямо подключить исходный текст отредактированной копии файла `STARTUP.A51`. Здесь важно соблюдать правильный порядок подключения файлов. `STARTUP.A51` подключается первым, потом идут файлы с программой пользователя, затем библиотечные файлы в строго установленном порядке – сначала `C51FPx.lib`, потом `C51x.lib`. Смысл такого порядка: компоновщик в указанном порядке будет искать внешние функции *до их первого обнаружения*. Например, при поиске внешней функции `?C_STARTUP` компоновщик обнаружит ее в файле `STARTUP.A51` и прекратит дальнейший поиск этой функции. Таким образом, стандартная версия этой функции, содержащаяся в файле `C51x.lib`, будет проигнорирована.

Данный способ дает возможность вносить правки, не модифицируя ни файлы стандартных библиотек, ни программы пользователя. Однако данный способ имеет серьезный недостаток: если мы забыли подключить к проекту отредактированную копию файла `STARTUP.A51`, то это не вызовет ошибки ни при ассемблировании, ни при компоновке. Компиляция пройдет успешно, но получившаяся программа будет неработоспособной. Такую ошибку найти чрезвычайно трудно. Она может быть выявлена только при анализе исполняемого файла в бинарных кодах или загрузочного файла в HEX-кодах.

В-третьих, можно воспользоваться директивой макроассемблера `#include`. Для этого отредактированную копию стартового файла переименовываем в `SDK-startup.inc` и помещаем в рабочую директорию проекта. В тексте программы (рис. 4.4) указываем директиву `#include "SDK-startup.inc"`. Обратите внимание, что ранее использовавшаяся директива

```
EXTRN CODE (?C_STARTUP),
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Пример вызова библиотечной функции.
; Модификация заголовка программы.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Стартовые настройки
#include "SDK-startup.inc"
;Декларация сегмента
?PR?primer
        SEGMENT  CODE
;Функция библиотеки
        EXTRN    CODE (?C?FPDIV)
        PUBLIC   main          ; Имя головной ф-ции
        RSEG     ?PR?primer    ; Начало сегмента
main:                          ; Начало ф-ции main
;.....................................................
;.....................................................
        SJMP     $             ; Конец функции main
        END                    ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Рис. 4.4. Модификация заголовка программы (рис. 4.1)

в этом файле отсутствует, поскольку функция `?C_STARTUP` теперь содержится в файле `SDK-startup.inc` и является внутренней функцией данного модуля. Это наиболее надежный способ модификации стартовой функции. В этом случае, если программа ассемблер не найдет модифицированный файл `SDK-startup.inc`, то мы получим сообщение о соответствующей ошибке. Во всех трех случаях стартовым адресом при запуске программ пользователя будет адрес `2000H`.

Программирование векторов прерываний при использовании сегментной структуры имеет особенность (рис. 4.5), поскольку векторы прерываний необходимо располагать по абсолютным адресам. Для стенда SDK-1 пользовательская таблица векторов прерываний должна быть размещена в области 2000H–204AH, поэтому при программировании по абсолютным адресам необходимо сначала объявить абсолютный сегмент, начинающийся с адреса 2003H. Выбор адреса объясняется тем, что модифицированная стартовая программа уже заняла адреса 2000H–2002H под команду `LJMP STARTUP1`. После этого вектора прерываний программируются обычным образом с помощью

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Пример вызова библиотечной функции.
; Программирование векторов прерываний.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Стартовые настройки
#include "SDK-startup.inc"
;Декларация сегмента
?PR?primer
       SEGMENT CODE
;Функция библиотеки
       EXTRN   CODE (?C?FPDIV)
       PUBLIC  main          ; Имя головной ф-ции
;-- Начало программирования векторов прерываний --
       CSEG    AT 2003h      ; Абс. сегмент с 2003H
       AJMP    vector1       ; П/п обсл. вектора 1
       ORG     200BH         ; Вектор прерываний 2
       AJMP    vector2       ; П/п обсл. вектора 2
;.............................................
;---Конец программирования векторов прерываний----
       RSEG    ?PR?primer    ; Начало сегмента
main:                        ; Начало ф-ции main
;.............................................
;.............................................
       SJMP    $             ; Конец функции main
       END                   ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

Рис. 4.5. Программирование векторов прерываний

директив ORG. Для первого вектора прерываний (адрес 2003H) директива ORG не нужна, так как абсолютный сегмент начинается с этого адреса. Отметим, что директиву ORG при этом можно не использовать совсем, а начинать новый абсолютный сегмент для каждого вектора прерываний. Это позволит получить вполне работоспособный вариант программы, но компилятор при этом попытается заполнить все свободные адреса между этими сегментами. Это не всегда приемлемо, так как многочисленные лишние команды SJMP могут замедлить работу программы. В этой связи настоятельно рекомендуется первый способ, а именно – объявить абсолютный сегмент, начинающийся с адреса 2003H, и векторы прерываний с помощью директивы ORG.

При использовании данной технологии программирования пользователь избавлен от проблем со стартовым адресом программы, порядком размещения основной программы и подпрограмм, таблицей векторов прерываний и т.д. Все эти вопросы будут решаться программой-компоновщиком. Стартовым адресом при запуске программ пользователя во всех случаях по-прежнему будет адрес 2000H.

Задания на выполнение курсовой работы по микропроцессорной технике подразумевают обязательное использование стандартных библиотек Keil μVision, поскольку предполагают операции с элементарными функциями и числами в формате с плавающей точкой.

5. КУРСОВАЯ РАБОТА ПО ПРОЕКТИРОВАНИЮ ИМПУЛЬСНОЙ И МИКРОПРОЦЕССОРНОЙ ТЕХНИКИ

Современное развитие науки и техники немыслимо без широкого использования микропроцессорной техники в любой отрасли народного хозяйства. При этом знания, умения, навыки и соответствующие компетенции оказываются важнейшими критериями оценки эффективности подготовки бакалавров, специалистов и магистров.

Курсовая работа по проектированию микропроцессорной техники завершает практический курс изучения микропроцессорной техники. В данном разделе представлены методические указания к выполнению курсовой работы, приведены необходимые требования к построению и содержанию курсовой работы, её оформлению, а также рекомендации по выбору темы работы.

5.1. Общие положения

Объем и содержание курса «Микропроцессорная техника», составной частью которого является курсовая работа по учебной дисциплине «Проектирование импульсной и микропроцессорной техники», определены Государственными стандартами по направлениям «Прикладные ядерная физика и технологии», «Биомедицинская техника», «Биотехнические системы и технологии», «Биомедицинская инженерия» и учебными планами соответствующих специальностей и направлений подготовки бакалавров и магистров.

Приведенные в данном разделе методические указания предназначены в первую очередь для студентов специальности «Электроника и автоматика физических установок», выполняющих курсовую работу по дисциплине «Проектирование импульсной и микропроцессорной техники». Однако данные методические указания могут быть полезны также студентам всех упомянутых выше специальностей и направлений, использующих элементы микропроцессорной техники при выполнении курсовых работ и проектов по смежным дисциплинам.

Курсовому проектированию должно предшествовать аудиторное и самостоятельное изучение базового курса «Микропроцессорная техника», а также дисциплин «Цифровая электроника», «Аналоговая схемотехника» и «Физические основы электронной техники».

5.2. Цели и задачи курсовой работы

Курсовая работа (проект) – это учебный документ, выполняемый студентами по учебному плану на промежуточных этапах обучения. Процесс выполнения курсовой работы (проекта) является одним из видов самостоятельной работы студентов, выполняемой в течение времени, отводимом на изучение данной дисциплины.

Выполнение курсовой работы имеет целью:

- систематизировать и закрепить теоретические и практические знания студентов по отдельным дисциплинам, помочь на практике применить полученные теоретические знания при проектировании микропроцессорных контрольно-измерительных и управляющих систем, а также систем сбора и обработки информации;
- развить навыки самостоятельной работы, проверить способности студента самостоятельно, творчески применять на практике теоретические знания, полученные при изучении базовой дисциплины «Микропроцессорная техника»;
- освоить технологии проектных и научных работ;
- обучить методам выбора и обоснования технических решений, приобрести необходимые навыки и умения в проектировании контрольно-измерительных и управляющих устройств на однокристальных микроконтроллерах;
- изучить современные стандарты, системы автоматизированного проектирования и подготовки конструкторской документации.

В процессе курсового проектирования студенты приобретают навыки по составлению алгоритма заданных операций и доведению поставленной задачи до конкретного программно-аппаратного технического решения. Для этого необходимо хорошо освоить теоретический материал и проявить творческую инициативу в подборе и изучении специальной литературы.

5.3. Тематика курсовых работ

Курсовая работа является самостоятельной работой студента. К ее выполнению необходимо отнестись с особым вниманием и тщательностью. Курсовая работа выполняется по индивидуальным заданиям, которые выдаются преподавателем. Задания на курсовое проектирование предусматривают решение специализированных задач с использованием микропроцессорных средств обработки информации.

Тематика курсовых работ связана, как правило, с проектированием контроллерных устройств, выполняющих разнообразные контрольно-измерительные и управляющие функции, а также функции по обработке данных, поступающих в режиме реального времени. Тема работы может быть связана также с научными или производственными интересами студента, лежащими в русле изучаемой дисциплины.

Исходные данные для курсовой работы задаются в виде, стимулирующем поиск прогрессивных схемных и алгоритмических решений и получение высоких показателей при обработке потоков данных. При формулировке содержания расчетно-пояснительной записки основной акцент делается на получение студентом в процессе проектирования навыков в выборе алгоритма решения задач, возникающих перед разработчиком аппаратно-программных средств.

Тематика курсовой работы разработана в соответствии с учебным планом дисциплины «Микропроцессорная техника» и предусматривает выполнение работы с учетом специализации студентов.

Кроме предлагаемых преподавателем вариантов курсовых работ, возможны темы, связанные с госбюджетными и хоздоговорными научно-исследовательскими и опытно-конструкторскими работами, выполняемыми на кафедре, а также с заказами предприятий и организаций соответствующего профиля. Студенты, активно участвующие в научно-исследовательской работе (НИРС), могут предложить свою тему, если она соответствует программе курса.

Темы курсовых работ в зависимости от объёма решаемых задач могут быть индивидуальными, рассчитанными на выполнение одним студентом, или комплексными. Для выполнения последних привлекается несколько студентов, каждому из которых отводится самостоятельная часть из общей работы. Предпочтительными являются комплексные темы, позволяющие наиболее полно решить задачу, причем работа приобретает законченный характер.

5.4. Порядок выполнения курсовой работы

Приступить к выполнению курсовой работы студент должен сразу же после получения задания, не дожидаясь, когда рассматриваемый вопрос будет изучен в лекционном курсе или лабораторном практикуме. Для качественного и своевременного выполнения курсовой работы рекомендуется придерживаться следующей последовательности:

1. Ознакомиться с выданной преподавателем темой и провести анализ задания на курсовую работу.

2. Провести самостоятельное изучение данного вопроса по совре-

менным литературным источникам и установить функциональное назначение системы.

3. Подготовить предварительный план намечаемой работы и обсудить его с руководителем.

4. Изучить поставленный вопрос по специальной технической литературе. Особая роль отводится здесь работе студента со специальной технической литературой, в том числе и на иностранном языке.

5. Составить с учетом изученной специальной литературы эскизный проект функциональной схемы аппаратной части курсовой работы, провести обоснование и выбор структурной схемы микропроцессорной системы и интегральных схем и других электронных компонентов, входящих в систему.

6. Разработать блок-схему алгоритма работы контроллера и приступить к описанию.

7. В соответствии с блок-схемой алгоритма и принципами структурного программирования приступить к проектированию на языке ассемблера основных модулей программы. На данном этапе следует использовать интегрированную среду разработки Keil μVision, уже знакомую студентам по лабораторному практикуму.

8. Рекомендуется отлаживать и тестировать каждый функционально законченный блок программы до его фактического размещения в общем программном модуле.

9. Добиться работоспособности всей программы. Продемонстрировать ее работу преподавателю и зафиксировать результаты испытания. Для доказательства правильности функционирования проектируемого контроллера может потребоваться подключение дополнительного лабораторного оборудования, имеющегося в лаборатории микропроцессорной техники, а также разработка тестовых примеров, наборов тестовых данных или дополнительных тестовых программ.

10. Приступить к оформлению пояснительной записки и графической части курсовой работы.

Для курсовых работ по разработке и изготовлению макетов и лабораторных образцов допускается по согласованию с преподавателем оформлять только пояснительную записку (объем 10–15 страниц рукописного текста) с включением в нее в качестве приложений необходимых структурных, принципиальных или аналогичных им по смыслу схем.

Курсовые работы, направленные на решение самостоятельных вопросов в рамках госбюджетных и хоздоговорных работ, допускается защищать без оформления текстовых и графических материалов в случае, если эти материалы включены в отчеты по соответствующей научно-исследовательской или опытно-конструкторской работе.

5.5. Требования к структуре и оформлению работы

В состав курсовых работ (проектов) входят текстовые и графические документы, а также может входить программная и технологическая документация. Правила оформления курсовых работ (проектов) есть в стандарте предприятия СТП УПИ 1-96. Стандарт предприятия опирается на государственные стандарты Российской Федерации. На текущий момент времени это основные стандарты: ГОСТ 2.105-95, ГОСТ 6.30-97 и ГОСТ 7.32-2001. Конкретизация специфики оформления учебной документации по профилю обучения содержится в методических рекомендациях по оформлению курсовых и дипломных работ [3]. Методические рекомендации не подменяют действующих государственных стандартов Российской Федерации, которые обязательны для изучения и играют главенствующую роль.

Законченная курсовая работа состоит из расчетно-пояснительной записки и графического материала, которые должны в совокупности давать достаточно полное представление о разрабатываемом устройстве, основных принципах его работы, о решениях, положенных в основу разработки функциональных схем и т.д.

Студент несет ответственность за правильность вычислений, работоспособность программного кода, качество оформления расчетно-пояснительной записки и графических материалов, за своевременное выполнение работы и представление ее к защите.

Хорошо выполненные курсовые работы могут быть рекомендованы для участия в конкурсах или студенческих конференциях.

Ниже приводится более подробное описание основных разделов курсовой работы.

Расчетно-пояснительная записка является основным документом, представляемым к защите. Качество ее оценивается полнотой, точностью и логикой изложения, а также аккуратностью и правильностью заполнения составляющих частей: текста, рисунков, таблиц и библиографического списка.

Объем записки составляет 20–30 страниц стандартного формата А4 (210×297 мм^2), включая схемы и иллюстрации. В отдельных случаях допускается использование большего объема записки и графики большего формата. По краям листа оставляются поля: слева – 25 мм, справа – 10 мм, сверху и снизу – по 20 мм.

Все страницы записки, в том числе с графиками, рисунками, листингом программы и т.п., нумеруются и переплетаются в единую папку.

Схемы, диаграммы, рисунки и другие иллюстративные материалы

выполняются на отдельных листах миллиметровой бумаги единообразно по всей работе, с указанием номера рисунка и подрисуночной подписью; все формулы следует пронумеровать.

Расчетно-пояснительная записка в общем случае может состоять из нижеследующих составных частей.

1. Титульный лист, на котором указывается Министерство образования и науки Российской Федерации, полное наименование университета, факультета и кафедры, а также тема курсовой работы, ФИО студента и номер группы, ФИО преподавателя, город и год (1 страница). Пример титульного листа приведен в прил. 2.

2. Бланк задания на курсовую работу, подписанный преподавателем и утвержденный заведующим кафедрой. Задание на выполнение курсовой работы (проекта) является нормативным документом, устанавливающим границы и глубину исследования (разработки) темы, а также сроки представления работы на кафедру в завершенном виде. Бланк задания имеет стандартную форму и содержит данные о студенте, название темы работы и краткие исходные данные. Здесь кратко приводится основное содержание расчетно-пояснительной записки и графического материала. Задание с указанием даты его выдачи подписывается преподавателем и утверждается заведующим кафедрой. Задание на курсовое проектирование необходимо поместить в начале пояснительной записки сразу после титульного листа (1 страница).

После выполнения курсовой работы руководитель делает отметку на бланке о ее завершении, а по окончании защиты комиссия проставляет оценку. Без правильно заполненного бланка задания и отметки о ее завершении курсовая работа к защите не допускается. Форма бланка заданий на курсовую работу приведена в прил. 3.

3. Оглавление включает название разделов и номера страниц, соответствующих началу каждого из них (1 страница).

4. Перечень использованных сокращений и обозначений, который содержит составленный в алфавитном порядке список малораспространенных сокращений, каждое из которых должно быть расшифровано. Перечень обозначений содержит список составленных в алфавитном порядке обозначений, использованных в тексте и на чертежах, с их расшифровкой (1 страница).

5. Введение содержит краткую вводную информацию, касающуюся предметной области курсовой работы (1–2 страницы).

6. Инженерная формулировка проектного задания и техническое обоснование выбранных решений со ссылками на литературные источники. Проектное задание – это полное и точное определением всех требований к разрабатываемому устройству. Обязательно приводится связь проектируемого устройства с внешними для него источниками

потоков данных, т.е. интерфейс проектируемого устройства. Техническое обоснование выбранного решения представляет собой аналитический обзор технической литературы по теме курсовой работы, включающий периодические издания и Интернет-источники. Основное назначение этого обзора – показать преимущества и недостатки выбранных технических решений в сравнении с известными альтернативными решениями. При необходимости рассматриваются также теоретические аспекты данного вопроса (2–3 страницы).

7. Разработка и описание функциональной схемы, выбор и обоснование технологических параметров, необходимых для последующего программирования (адреса устройств, векторы прерываний, режимы функционирования встроенных узлов, управляющие слова, флаги, сигналы квитирования и т.п.). В этом разделе необходимо описать принципы построения функциональной схемы контроллера, взаимодействие основных его устройств при вводе, обработке и выводе информации, способы формирования управляющих сигналов и организации с их помощью управления, порядок программирования и перепрограммирования БИС, а также выполнить расчет отдельных его элементов. При необходимости привести временные диаграммы работы элементов контроллера либо таблицы их состояний. На чертежах все выводы БИС обозначить в соответствии с общепринятой методикой их обозначения, взятой из ЕСКД и стандартов на микросхемы.

Проработка этого этапа необходима даже в том случае, когда в качестве прототипа выбирается стандартное аппаратное решение, например стенд SDK-1 (5–7 страниц).

8. Разработка, графическое представление и описание блок-схемы алгоритма программы. Алгоритм работы устройства – один из основных документов, к его разработке следует отнестись особенно внимательно, поскольку при неверно выбранном алгоритме теряют смысл самые изысканные технические решения (3–4 страницы).

9. Разработка программы на языке ассемблера и результаты ассемблирования. Описание программы и использованных библиотечных процедур (2–3 страницы). Листинг программы с подробными комментариями поместить в приложение.

10. Результаты испытаний, содержательные выводы и рекомендации по дальнейшему развитию работы (1–2 страницы).

11. Список использованных литературных источников (1 страница).

12. Приложение, содержащее графический материал и листинг программы (количество страниц – по необходимости).

В процессе выполнения перечисленных этапов работы формируются текстовый и графический материалы, которые в дальнейшем составят основу курсовой работы.

Состав разделов расчетно-пояснительной записки для конкретного задания уточняется во время консультации с преподавателем.

Графическая часть курсовой работы выполняется обычно на листах формата А3. Допускается оформление графических материалов на масштабно-координатной (миллиметровой) бумаге – на лицевой или изнаночной стороне – карандашом, но с обязательным использованием необходимых чертежных инструментов. Условные обозначения, шрифты и масштабы чертежа должны соответствовать требованиям Единой системы конструкторской документации ЕСКД. Допускается использование специализированных компьютерных пакетов разработки и оформления конструкторской документации, таких как, например, КОМПАС или Autocad.

Рекомендуются следующие минимальные нормы представления графических материалов:

– функциональная схема контроллера – 1 лист;

– блок-схема алгоритма программы – 1 лист.

Перечень графического материала задан в объеме, предусматривающем обучение студента оформлению технической документации. Конкретное содержание графической части уточняется при выдаче задания. В случае необходимости по согласованию с руководителем допускается изменение норм представления графического материала, внесение части из них в расчетно-пояснительную записку и т.п. при непременном сохранении общего объема не менее двух листов.

5.6. Контроль за выполнением работы

Руководитель контролирует все этапы выполнения курсовой работы студентом. Кроме того, руководитель

1) проводит консультации и обсуждения со студентом правильности принимаемых решений;

2) контролирует соблюдение плановых сроков выполнения отдельных этапов;

3) проверяет все материалы, составляющие работу, на предмет их готовности к защите.

За содержание и форму работы, за ее качество ответственность несет исполнитель – студент.

Устанавливаются три контролируемых рубежа выполнения курсовой работы. К первому рубежному контролю должны быть: сформу-

лировано проектное задание, выполнен аналитический обзор литературы по теме, разработан эскиз функциональной схемы (т.е. 25–30 % от общего объема курсовой работы).

Ко второму рубежному контролю следует представить полностью проработанную функциональную схему и блок-схему ее функционирования (т.е. должно быть готово 60 % общего объема).

К третьему рубежному контролю работа должна быть закончена. Объекты проектирования следует испытать в лаборатории микропроцессорной техники и продемонстрировать их работу руководителю. После одобрения руководителя вся документация должна быть оформлена и предъявлена преподавателю на проверку. Готовность курсовой работы определяется руководителем и подтверждается его подписью на расчетно-пояснительной записке и графическом материале. Курсовая работа считается готовой, если выполнены все пункты, предусмотренные заданием. Работы, не прошедшие натурные испытания в лаборатории, в исключительных случаях по решению заведующего кафедрой могут быть также допущены к защите. Однако такие работы как правило не могут претендовать на получение высших оценок.

Все материалы курсовой работы после установления её готовности представляются комиссии, назначаемой кафедрой, и защищаются студентом по всем разделам, предусмотренным заданием. По результатам выполнения работы и защиты выставляется оценка с учетом:

1) объема и качества выполнения работы, оригинальности и самостоятельности решений;

2) знаний по вопросам, связанным с разработкой функциональной схемы контроллера, блок-схемы алгоритма его функционирования и программы на языке ассемблера;

3) умения излагать результаты работы, обосновывать и защищать принятые решения и отвечать на заданные вопросы.

Полностью подготовленная и оформленная курсовая работа сдается на проверку преподавателю. При наличии замечаний в рецензии на работу студент должен дать на них письменный ответ, а в случае необходимости внести в работу исправления. После доработки исправлений курсовая работа допускается к защите. Качество доработки оценивает преподаватель, который на титульном листе ставит визу о допуске работы к защите. Защита работы производится индивидуально каждым студентом на заседании комиссии, назначаемой заведующим кафедрой. На изложение основного содержания работы студенту дается около 7 мин, доклад иллюстрируется графическим материалом. Оценка работы – дифференцированный зачет (зачет с оценкой).

В процессе защиты студенту могут быть заданы разнообразные вопросы по содержанию работы, по теоретической ее части, по алгоритму и функциональным схемам отдельных узлов.

Комиссия оценивает работу по ее качеству и уровню защиты и проставляет оценку на титульном листе работы и в зачетной книжке студента. При получении неудовлетворительной оценки студент защищает курсовую работу повторно. Повторная защита с целью получения повышенной оценки как правило не допускается. В исключительных случаях по разрешению заведующего кафедрой студент может получить новое задание и выполнить курсовую работу заново с последующей ее защитой.

По завершении защиты расчетно-пояснительная записка и графические материалы остаются для хранения на кафедре.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Огородников И.Н. Микропроцессорная техника : учебник /И.Н. Огородников. Екатеринбург: УГТУ-УПИ, 2007. 380 с.

2. Каспер Э. Программирование на языке ассемблера для микроконтроллеров семейства i8051 /Э. Каспер. М.: Горячая линия-Телеком, 2003. 192 с.

3. Кичигин В.Н. Оформление курсовых и дипломных проектов : методические указания для студентов технических специальностей /В.Н. Кичигин, И.Е. Мясников, С.И. Тимошенко. Екатеринбург: УГТУ-УПИ, 2005. 80 с.

4. Учебный стенд SDK-1.1 : руководство пользователя. СПб.: ЛМТ, 2006. 100 с. [Электронный ресурс]. Режим доступа: http://lmt.cs.ifmo.ru. (файл `sdk11_userm_v1_0_11.pdf`)

5. Спецификация ADuC812 /пер. с англ. Б.Л. Горшкова, Ю.М. Зайцева, В.И. Силантьева. СПб.: АВТЭКС, 1999. 30 с. [Электронный ресурс]. Режим доступа: http://www.autex.spb.ru. (файл `aduc812_rus.pdf`)

6. Огородников И.И. Программный модуль "Программируемая инструментальная среда (загрузчик, терминал) HEX202ldr (HEX202ldr)" : свидетельство о государственной регистрации программы для ЭВМ №2010611096 зарегистрировано в едином реестре программ для ЭВМ 05 февраля 2010 г. /И.И. Огородников, И.Н. Огородников.

7. Лукичев А.Н. Отличия в программировании SDK-1.1 с ADuC842, ADuC831 и ADuC812 /А.Н. Лукичев. СПб.: ЛМТ, 2004. 10 с. [Электронный ресурс]. Режим доступа: http://www.lmt.ifmo.ru. (файл `sdk11_appnote1.pdf`)

8. ADuC841/ADuC842/ADuC843: MicroConverter, 12-Bit ADCs and DACs with Embedded High Speed 62-kB Flash MCU Data Sheet (Rev 0, 11/2003). Norwood: One Technology Way, 2003. 88 p. [Электронный ресурс]. Режим доступа: http://embedded.ifmo.ru/index.php/support/sdk-11. (файл `ADuC841_2_3_0.pdf`)

ПРИЛОЖЕНИЯ

Приложение 1. Пример выполнения ИДЗ

Министерство образования и науки Российской Федерации

ФГАОУ ВПО «Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина»

Физико-технологический институт
Кафедра экспериментальной физики

ПРЕОБРАЗОВАТЕЛЬ КОДОВ

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ
по дисциплине «Микропроцессорная техника»

Пояснительная записка

Руководитель проф., д.ф.-м.н. И.Н. Огородников

Студент гр. Фт-47031 И.И. Иванов

Екатеринбург 2012

ФГАОУ ВПО «Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина»
Кафедра экспериментальной физики

«УТВЕРЖДАЮ»
Зав.кафедрой

_____В.Ю. Иванов
“ *18* ” *декабря* 2012 г.

Задание № *010*
индивидуальное домашнее
по дисциплине «*Микропроцессорная техника*», 7 семестр

Студент группы *Фт-47031* специальность *140306 – ЭАФУ*
Фамилия *Иванов* Имя *Иван* Отчество *Иванович*
Преподаватель *проф., д.ф.-м.н. Огородников И.Н.*
Срок выполнения задания с *18 декабря 2010 г.* по *25 декабря 2010 г.*

1. Тема работы: *«Преобразователь кодов»*

2. Содержание работы (какие графические работы и расчеты должны быть выполнены) *Пояснительная записка и графические материалы:*
2.1. Схема обработки данных (эскиз), ее описание, карта памяти
2.2. Блок-схема алгоритма подпрограммы и ее описание
2.3. Текст подпрограммы на языке ассемблера с подробными комментариями
2.4. Длина подпрограммы в байтах, оценка времени ее выполнения
2.5. Тестовый пример–программа для проверки подпрограммы

Принял к исполнению. Студент_____

3. Индивидуальное задание выполнено_____

4. Оценка выполнения_____

Руководитель_____

1. Цель работы

Целью индивидуального домашнего задания является разработка подпрограммы, осуществляющей процесс приема четырехбитовых данных, преобразования их в семисегментный код и размещения байтов семисегментного кода во внешней памяти данных, начиная с адреса 5000H.

2. Схема обработки потока данных

На рис. 1 представлена схема обработки потока данных, поступающих от внешнего устройства ПУ1. Четырехбитовые данные поступают от устройства ПУ1 на линии младшей тетрады порта P0. Одновременно с этим периферийное устройство выставляет единичный сигнал квитирования «Данные готовы» на линию P0.5 микроконтроллера. По этому стробу микроконтроллер считывает четырехбитовые данные P0.0–P0.3 и начинает их обработку. Обработка заключается в поиске по таблице готовых решений соответствующего семисегментного кода и его размещении во внешней памяти данных.

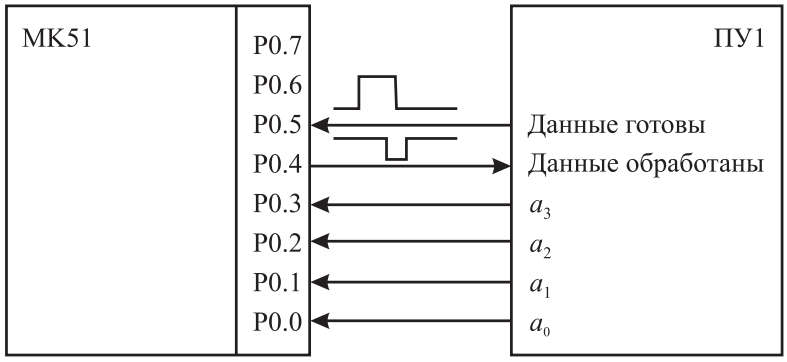

Рис. 1. Схема потоков данных

После завершения обработки четырех битов данных микроконтроллер выставляет нулевой сигнал квитирования «Данные обработаны» на линию P0.4 и ожидает снятия сигнала периферийного устройства «Данные готовы». После этого микроконтроллер снимает свой сигнал «Данные обработаны» и переходит в исходный режим ожидания новых данных от периферийного устройства, если еще не достигнут конец потока данных.

Поиск по таблице готовых решений можно осуществить с помощью специальной команды микроконтроллера `MOVC A,@A+PC`, а сама таблица (массив констант) может располагаться непосредственно в памяти программ. Соответствие двоичных и семисегментных кодов приведено в табл. 1, а схема формирования символов семисегментного индикатора показана на рис. 2.

Таблица 1. Соответствие двоичного и семисегментного кодов

Символ	Двоичный код	Семисегментный код
0	0000	00111111
1	0001	00000110
2	0010	01011011
3	0011	01001111
4	0100	01100110
5	0101	01101101
6	0110	01111101
7	0111	00000111
8	1000	01111111
9	1001	01101111
A	1010	01110111
B	1011	01111100
C	1100	00111001
D	1101	01011110
E	1110	01111001
F	1111	01110001

Рис. 2. Формирование цифровых символов семисегментного кода

3. Карта памяти

Процесс преобразования кодов предполагает использование следующих регистров: аккумулятор (A), указатель данных (DPTR), счетчик цикла (R0) и порт ввода-вывода (P0). Таблица преобразователя кодов занимает 16 байтов и располагается в памяти программ, результат преобразования размещается во внешней памяти данных, начиная с адреса 5000H.

4. Блок-схема алгоритма подпрограммы

На рис. 3 приведена блок-схема алгоритма подпрограммы, реализующей техническое задание. Особенностью алгоритма является вынесение таблицы готовых решений в отдельную подпрограмму CODE7. Блок-схема подпрограммы CODE7 приведена на рис. 4.

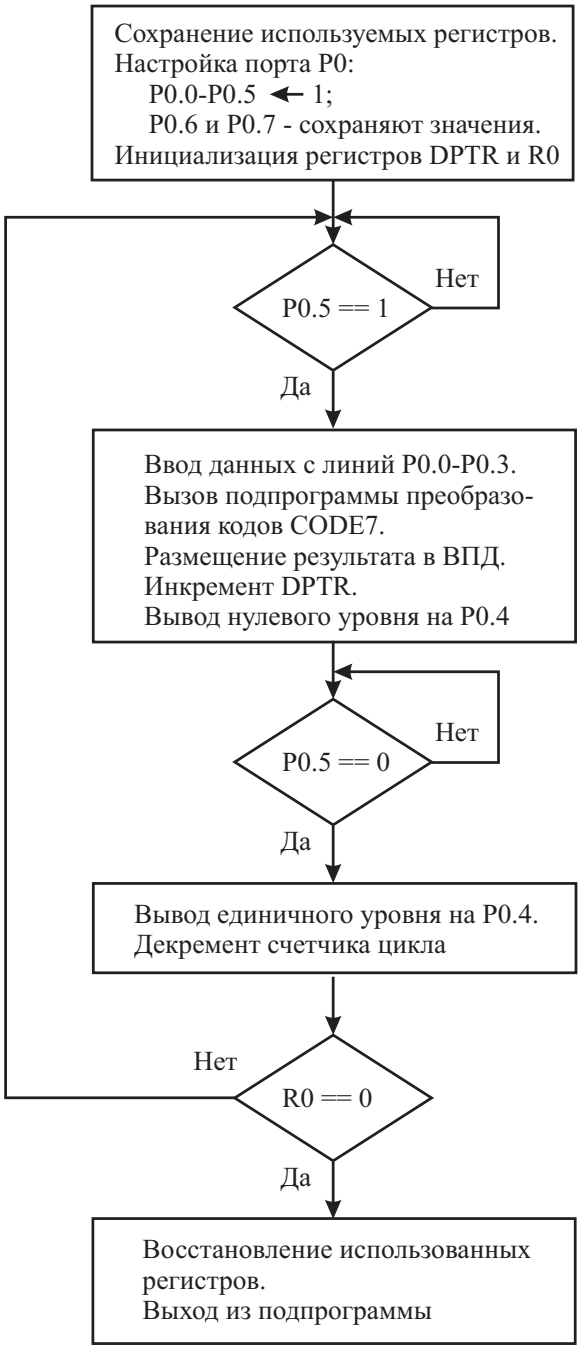

Рис. 3. Блок-схема алгоритма основной подпрограммы

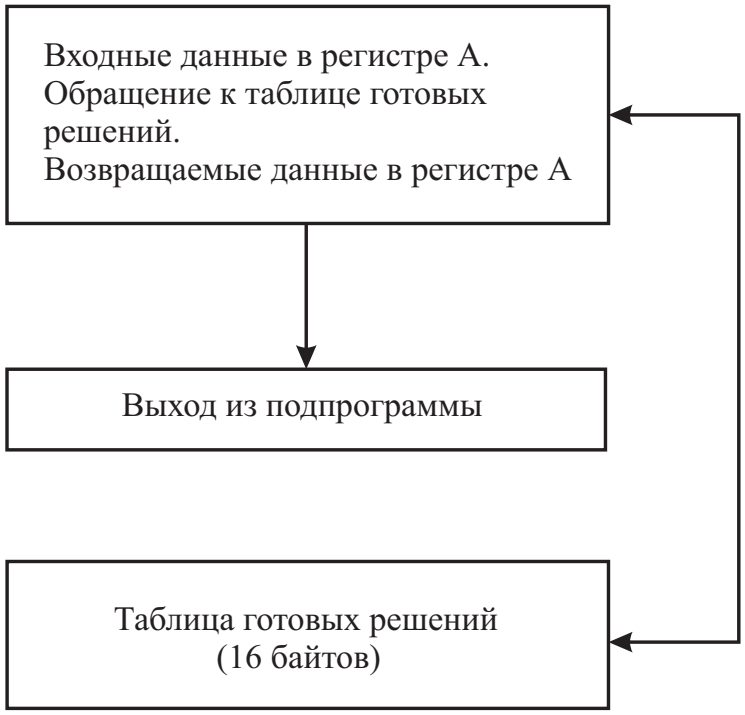

Рис. 4. Блок-схема алгоритма подпрограммы CODE7

Особенностью подпрограммы CODE7 является использование команды `MOVC A,@A+PC`. Это позволяет организовать обращение к таблице готовых решений без использования регистра DPTR, который уже задействован в обращении к внешней памяти данных. Таблица готовых решений должна располагаться в памяти программ непосредственно за командой MOVC. Команда MOVC получает входные данные через регистр (A), в этом же регистре будет размещен результат преобразования кодов. Однако необходимость использования однобайтной команды возврата из подпрограммы приводит к тому, что таблица решений располагается со смещением на один байт от команды MOVC. Поэтому входные данные должны быть предварительно увеличены на единицу командой инкремента.

Контроль количества преобразованных байтов осуществляется счетчиком R0, который декрементируют после каждого преобразования. По достижении нуля в R0 происходит выход из основной подпрограммы. Для локализации переменных используемые регистры сохраняются в стеке и восстанавливаются перед выходом из основной подпрограммы.

5. Листинг подпрограммы на языке ассемблера

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Подпрограмма преобразователя кодов SEVNSG
; Входные данные: (A) -- кол-во тетрад в потоке
; Выходные данные: адреса ВПД 5000H -- 50FFH
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ORG    2100H        ; Начальный адрес п/п
SEVNSG: PUSH   DPL          ; Сохранение в стеке
        PUSH   DPH          ;
        PUSH   PSW          ;
        PUSH   ACC          ;
        XCH    A,R0         ; Обмен (A) и (R0)
        PUSH   ACC          ; Сохранение R0 в стеке
        MOV    DPTR,#5000H  ; Загрузка указателя
;Программирование линий P0.0--P0.5 порта P0
        ORL    P0,#00111111B
;Ожидание сигнала "Данные готовы" на линии P0.5
LOOP:   JNB    P1.5,$       ;
        MOV    A,P0         ; Ввод байта данных
        ANL    A,#0FH       ; Выделение мл. тетрады
        LCALL  CODE7        ; Вызов п/п преобразования
        MOVX   @DPTR,A      ; Размещение рез-та в ВПД
        INC    DPTR         ; Инкремент указателя
        CLR    P0.4         ; Уст. "Данные обработаны"
;Ожидание снятия сигнала "Данные готовы" на P0.5
        JB     P0.5,$       ;
        SETB   P0.4         ; Сброс "Данные обработаны"
        DJNZ   R0,LOOP      ; Организация цикла
        POP    ACC          ; Восстановление R0
        MOV    R0,A         ;
        POP    ACC          ; Восстановление регистров
        POP    PSW          ;
        POP    DPH          ;
        POP    DPL          ;
        RET                 ; Возврат из основной п/п
;Конец основной подпрограммы преобразователя кодов
```

7

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Вспомогательная подпрограмма преобразователя
; кодов CODE7
; Входные данные: (A)--тетрада двоичного кода
; Выходные данные: (A)--байт семисегментного кода
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
CODE7:  INC   A           ; Инкремент данных
        MOVC  A,@A+PC     ; Обращение к таблице
        RET               ; Возврат из подпрограммы
; Таблица готовых семисегментных кодов
TABL:   DB    00111111B   ; 0
        DB    00000110B   ; 1
        DB    01011011B   ; 2
        DB    01001111B   ; 3
        DB    01100110B   ; 4
        DB    01101101B   ; 5
        DB    01111101B   ; 6
        DB    00000111B   ; 7
        DB    01111111B   ; 8
        DB    01101111B   ; 9
        DB    01110111B   ; A
        DB    01111100B   ; B
        DB    00111001B   ; C
        DB    01011110B   ; D
        DB    01111001B   ; E
        DB    01110001B   ; F
;Конец вспомогательной подпрограммы CODE7
        END               ;
```

6. Оценка требуемых ресурсов и времени выполнения

Суммарная длина подпрограмм и таблицы готовых решений составляет 68 байтов: 49 байтов (подпрограмма SEVNSG) + 3 байта (подпрограмм CODE7) + 16 байтов (таблица готовых семисегментных кодов TABL) [1, 2]. Глубина стека составляет 9 байтов. Время одного преобразования без учета ожидания сигналов квитирования составляет около 45 мкс.

7. Программа для проверки и тестовый пример

Ниже приведены листинг программы для проверки задания и тестовый пример для испытания программы (табл. 2).

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Программа для тестирования выполненного задания
; MAIN
; FORMAT -- задает кол-во тетрад во входном потоке
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ORG   2000H      ; Начальный адрес программы
FORMAT  EQU   16D        ; Длина тестового примера
MAIN:   MOV   A,#FORMAT  ; Передача параметра п/п
        CALL  SEVNSG     ; Вызов п/п SEVNSG
        SJMP  $          ; Конец основной программы
;Конец программы для тестирования.
        END              ;
```

Таблица 2. Тестовый пример для испытания программы

Входные данные	Выходные данные	
	Адрес ВПД	Семисегментный код
0000	5000H	00111111
0001	5001H	00000110
0010	5002H	01011011
0011	5003H	01001111
0100	5004H	01100110
0101	5005H	01101101
0110	5006H	01111101
0111	5007H	00000111
1000	5008H	01111111
1001	5009H	01101111
1010	500AH	01110111
1011	500BH	01111100
1100	500CH	00111001
1101	500DH	01011110
1110	500EH	01111001
1111	500FH	01110001

Заключение

В домашнем задании выполнена разработка программного обеспечения для выполнения преобразования четырехбитовых данных, поступающих от периферийного устройства, в соответствующие семисегментные коды. Реализована программная обработка сигналов квитирования, преобразование кодов с помощью таблицы готовых решений и размещение полученных семисегментных кодов во внешней памяти данных, начиная с адреса 5000H. Дана оценка требуемых ресурсов и ожидаемого быстродействия программы.

Для испытания программного обеспечения разработана тестовая программа и подготовлен тестовый пример. Реальное испытание программы запланировано в рамках лабораторного практикума на специализированном микропроцессорном стенде. При испытании программы на стенде может потребоваться уточнение начальных адресов размещения компонентов программы.

Изучены также правила оформления учебного документа [3], на основании которых оформлена пояснительная записка.

Библиографический список

1. Каспер Э. Программирование на языке ассемблера для микроконтроллеров семейства i8051 /Э. Каспер. М.: Горячая линия-Телеком, 2003. 192 с.

2. Огородников И.Н. Микропроцессорная техника : учебник /И.Н. Огородников. Екатеринбург: УГТУ-УПИ, 2007. 380 с.

3. Кичигин В.Н. Оформление курсовых и дипломных проектов : методические указания для студентов технических специальностей / В.Н.Кичигин, И.Е. Мясников, С.И. Тимошенко. Екатеринбург: УГТУ-УПИ, 2005. 80 с.

4. Лукичев А.Н. Отличия в программировании SDK-1.1 с ADuC842, ADuC831 и ADuC812 /А.Н. Лукичев. СПб.: ЛМТ, 2004. 10 с. [Электронный ресурс] файл `sdk11_appnote1.pdf`, http://www.lmt.ifmo.ru.

Приложение 2. Пример титульного листа

Министерство образования и науки Российской Федерации

ФГАОУ ВПО «Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина»
Физико-технологический институт
Кафедра экспериментальной физики

Оценка работы

Члены комиссии

КОНТРОЛЛЕР ТЕРМОРЕГУЛЯТОРА

КУРСОВАЯ РАБОТА
по дисциплине «Проектирование
импульсной и микропроцессорной техники»

Пояснительная записка

Руководитель проф., д.ф.-м.н. И.Н. Огородников

Студент гр. Фт-43031 И.И. Иванов

Екатеринбург 2012

Приложение 3. Пример бланка задания

ФГАОУ ВПО «Уральский федеральный университет имени первого Президента России Б.Н.Ельцина»

«УТВЕРЖДАЮ»
Зав.кафедрой_____
“____”_____2012 г.

Задание №_____
по курсовому проектированию

Студент группы_____специальность_____
Фамилия_____Имя_____Отчество_____
Руководитель курсового проектирования_____
Срок проектирования с_____по_____

1. Тема курсового проекта_____

2. Содержание проекта (какие графические работы и расчеты должны быть выполнены)_____

3. Особые дополнительные сведения_____

4. План выполнения курсового проекта

Наименование элементов проектной работы	Сроки	Примечание	Отметка о выполнении

5. Курсовое проектирование закончено_____
6. Оценка проекта_____

Руководитель_____

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

П

Р

С

Т

У

Ф

Ч

Ш

Э

Я

Учебное издание

Огородников Игорь Николаевич

Микропроцессорная техника: практический курс

Учебное пособие

ISBN 978-5-321-02171-2

9 785321 021712

Редактор *И. В. Коршунова*
Корректор *И. В. Коршунова*
Компьютерная верстка *И. Н. Огородникова*

План выпуска 2012 г. Подписано в печать 25.06.2012. Формат 60×84/16.
Бумага писчая. Цифровая печать. Усл. печ. л. 8,14.
Уч.-изд. л. 7,61. Тираж 100 экз. Заказ 2426

Отпечатано в типографии Издательско-полиграфического центра УрФУ
620000, Екатеринбург, ул. Тургенева, 4
Тел.: +7 (343) 350-56-64, 350-90-13
Факс: +7 (343) 358-93-06
E-mail: press.info@usu.ru