Министерство общего и профессионального образования
Российской Федерации
Уральский государственный технический университет

# ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРА INTEL 8051 НА ЯЗЫКЕ АССЕМБЛЕРА

Методические указания к лабораторной работе №3
по курсу “Цифровые устройства и микропроцессоры”
для студентов всех форм обучения специальностей
200700 – Радиотехника;
201500 – Бытовая радиоэлектронная аппаратура

Екатеринбург 1999

УДК 681.322

Составители В.А.Добряк, В.К.Рагозин

Научный редактор доц., канд. техн. наук В.И.Елфимов

**ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРА INTEL 8051 НА ЯЗЫКЕ АССЕМБЛЕРА**: Методические указания к лабораторной работе №3 по курсу “Цифровые устройства и микропроцессоры”/ В.А.Добряк, В.К.Рагозин. Екатеринбург: Изд-во УГТУ, 1999. 6 с.

Методические указания предназначены для использования при выполнении лабораторного практикума. Содержат основные правила программирования на языке ассемблера, описание часто используемых директив, порядок выполнения домашнего и лабораторного заданий, контрольные вопросы и задания для самостоятельной работы.

Библиогр.: 4 назв. Табл. 2. Прил.

Подготовлено кафедрой радиоприёмных устройств.

## ОГЛАВЛЕНИЕ

# 1. ЦЕЛЬ И СОДЕРЖАНИЕ РАБОТЫ

Целью работы является изучение основ языка ассемблера микроконтроллеров семейства Intel 8051, а также продолжение начатого в лабораторных работах №1 [3] и №2 [4] изучения интегрированной среды ProView фирмы Franklin Software Inc., которая предназначена для разработки программного обеспечения этого семейства. Работа рассчитана на 4 часа домашней подготовки и 4 часа занятий в лаборатории.

При домашней подготовке к работе изучаются основные правила программирования на языке ассемблера и наиболее употребительные директивы. Далее в соответствии с номером варианта студенты составляют программы на языке ассемблера.

Перед началом лабораторной работы проводится коллоквиум. Студенты, успешно ответившие на поставленные вопросы, допускаются к лабораторной части работы.

При выполнении лабораторного задания осуществляется ввод исходных текстов программ, ассемблирование и отладка программ.

После выполнения работы оформляется отчёт с указанным ниже содержанием.

# 2. ЗАДАНИЯ ДЛЯ ДОМАШНЕЙ ПОДГОТОВКИ

## 2.1. Изучите методику разработки прикладного программного обеспечения микроконтроллерных систем

Формализованный подход к разработке прикладных программ. Этапы формализации в разработке алгоритмов. Модульный принцип построения прикладных программ, процедуры и подпрограммы. Вызов подпрограмм, сохранение параметров основной программы, передача параметров в подпрограммы. Методика отладки прикладных программ.

Правила записи программ на языке ассемблера. Поля метки, операции, операндов и комментария. Обработка выражений в процессе трансляции. Директивы ассемблера: BIT, DATA, DB, DS, DW, END, EQU, ORG, RSEG, SEGMENT, SET, XDATA [1, 2].

## 2.2. Составьте подпрограмму

Подпрограмма должна реализовать процедуру скалярного произведения двоичных векторов размерности 8 в соответствии с формулой

$$scall = \vec{X}\vec{Y} = \sum_{i=0}^{7} x_i y_i ,$$

где $x_i$ - значение бита *i*-го разряда вектора $\vec{X}$; $y_i$ - значение бита *i*-го разряда вектора $\vec{Y}$.

В тексте подпрограммы необходимо обеспечить сохранение содержимого регистров микроконтроллера, так как регистры могут использоваться основной программой. Рассмотрите различные способы передачи параметров в подпрограмму. В максимально возможной степени используйте директивы ассемблера.

Составьте программу, обеспечивающую тестирование подпрограммы.

## 2.3. Составьте программу

Составьте программу на языке ассемблера, которая реализует алгоритм решения задачи в соответствии с одним из десяти вариантов, указанных в табл. 1. Номер варианта выбирается по последней цифре номера студенческого билета (зачётной книжки).

***Таблица 1***

*Варианты заданий на разработку программы*

| Вариант | Программа |
|---|---|
| 0 | Вычисление среднего арифметического $\overline{X}$ элементов массива $\{X_i\}$: $$\overline{X} = \frac{1}{N}\sum_{i=1}^{N} X_i .$$ |
| 1 | Вычисление энергии сигнала $S(t)$, заданного своими отсчётами $\{S_i\}$ через интервал времени 1 с на сопротивлении нагрузки $R_н$ = 1 Ом: $$E_S(Дж) = \sum_{i=1}^{N} S_i^2 \left( \frac{B^2 сек}{Ом} \right).$$ |
| 2 | Определение минимального элемента массива $\{X_i\}$: $$X_{\min} = \min_{i=1}^{N}\{X_i\}.$$ |
| 3 | Определение максимального элемента массива $\{X_i\}$: $$X_{\max} = \max_{i=1}^{N}\{X_i\}.$$ |
| 4 | Сортировка элементов массива $\{X_i\}$ в порядке возрастания. |
| 5 | Сортировка элементов массива $\{X_i\}$ в порядке убывания. |
| 6 | Попарное сравнение элементов двух массивов $\{X_i\}$, $\{Y_i\}$ и выборка минимального элемента в паре: $Z_i = \min(X_i, Y_i), \quad i = \overline{1,N}$. Запись элементов $\{Z_i\}$ в память данных. |
| 7 | Попарное сравнение элементов двух массивов $\{X_i\}$, $\{Y_i\}$ и выборка максимального элемента в паре: $Z_i = \max(X_i, Y_i), \quad i = \overline{1,N}$. Запись элементов $\{Z_i\}$ в память данных. |
| 8 | Программа вычисления значений булевой функции: $Y = f(X_1, X_2, X_3, X_4, X_5) = X_1 \overline{X}_2 X_4 \vee X_2 X_3 \overline{X}_5 \vee \overline{X}_1 \overline{X}_3 X_5$. |
| 9 | Программа вычисления значений булевой функции: $Y = f(X_1, X_2, X_3, X_4, X_5) = (X_1 \vee \overline{X}_2 \vee X_4)(X_2 \vee X_3 \vee \overline{X}_5)(X_1 \vee \overline{X}_3 \vee X_5)$. |

Исходный текст программы должен содержать не только операторы машинных команд, реализующие алгоритм предложенной задачи, но и директивы задания начальных значений памяти и регистров. В максимально возможной степени используйте директивы ассемблера.

Результаты работы программы должны фиксироваться в памяти.

## 2.4. Контрольные вопросы

1. Перечислите и охарактеризуйте этапы разработки прикладной программы.
2. Сформулируйте понятие функциональной спецификации прикладной программы.
3. Сформулируйте понятие модульной декомпозиции задачи прикладной программы.
4. Укажите достоинства и недостатки языка ассемблера.
5. Какова структура строки программы, написанной на языке ассемблера, какие поля строки являются обязательными?
6. Укажите правила выбора имени метки.
7. Каким образом выделяется комментарий?
8. Перечислите основные директивы языка ассемблера.
9. Какие директивы используются для указания начала и конца программного модуля?
10. Перечислите директивы, используемые для резервирования памяти.
11. Какие директивы используются для определения программных сегментов?
12. Укажите функции следующих директив: ORG, EQU, SET, BIT. В чём состоит отличие директив EQU и SET?
13. Укажите назначение директив DATA, XDATA.
14. Укажите назначение директив DB, DS, DW, END.
15. Укажите назначение директив RSEG, SEGMENT.
16. Каким образом определяется макрокоманда?
17. Укажите достоинства и недостатки подпрограмм по сравнению с макрокомандами. В каких случаях целесообразно использовать подпрограммы, а в каких - макрокоманды?
18. Какие команды можно использовать для создания циклической программы?
19. Какими обязательными свойствами должна обладать подпрограмма?
20. Каким образом используется стек при выполнении подпрограмм?
21. От чего зависит глубина вложенности подпрограмм?
22. Для чего может быть использован приём переключения регистровых банков?
23. Поясните механизм передачи параметров подпрограммы через память.
24. Поясните механизм передачи параметров подпрограммы через регистры общего назначения.
25. Поясните механизм передачи параметров подпрограммы через регистр признаков.
26. Поясните механизм передачи параметров подпрограммы через стек.
27. Укажите основные этапы работы с ассемблерной программой.
28. Как рассчитать время выполнения ассемблерной программы?
29. Укажите основные приемы, используемые при отладке программы.

# 3. МЕТОДИКА РАЗРАБОТКИ ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ МИКРОКОНТРОЛЛЕРНЫХ СИСТЕМ

## 3.1. Общие сведения

Если задача на разработку прикладной программы для микроконтроллера поставлена, то для получения текста исходной программы необходимо выполнить ряд последовательных действий.

1. Подробно описать задачу.
2. Проанализировать задачу.
3. Выполнить инженерную интерпретацию задачи, желательно с привлечением того или иного аппарата формализации (граф автомата, сети Петри, матрицы состояний и связности и т.п.).
4. Разработать общую схему алгоритма работы контроллера.
5. Разработать детализированные схемы отдельных процедур, выделенных на основе модульного принципа составления программ.
6. Детально проработать интерфейс контроллера и внести исправления в общую и детализированные схемы алгоритмов.
7. Распределить рабочие регистры и память.
8. Сформировать текст исходной программы.

В результате работы по трем первым пунктам данного перечня получают так называемую функциональную спецификацию прикладной программы, в которой основное внимание уделяется детализации способов формирования входной и выходной информации.

На языке схем алгоритмов разработчик описывает метод, выбранный им для решения поставленной задачи. Довольно часто бывает, что одна и та же задача может быть решена различными методами. Способ решения задачи, выбранный на этапе её инженерной интерпретации, на основе которого формируется схема, определяет не только качество разрабатываемой прикладной программы, но и качественные показатели конечного изделия.

Алгоритм есть точно определенная процедура, предписывающая контроллеру однозначно определенные действия по преобразованию исходных данных в обработанные выходные данные. Поэтому разработка схемы требует предельной точности и однозначности используемой атрибутики: символических имен переменных, констант, подпрограмм (модулей), символических адресов таблиц, портов ввода вывода и т.п. Основное внимание при разработке следует уделить тому разделу функциональной спецификации прикладной программы, в котором приводится описание аппаратуры сопряжения с объектом управления. Это описание должно быть детализировано вплоть до электрических и временных характеристик каждого входного и выходного сигнала или устройства.

Успех разработки прикладной программы заключается в использовании метода декомпозиции, при котором вся задача последовательно разделяется на меньшие функциональные модули. Каждый из модулей можно анализировать, разрабатывать и отлаживать отдельно от других. При выполнении прикладной программы в микроконтроллере управление без всяких двусмысленностей передается от одного функционального модуля к другому. Схема связности этих функциональных

модулей, каждый из которых реализует некоторую процедуру, образует общую схему алгоритма прикладной программы. Язык графических образов схемы алгоритма можно использовать на любом уровне детализации описания модулей вплоть до того, что каждому оператору схемы будет соответствовать единственная команда микроконтроллера.

Структурное программирование есть процесс построения прикладной программы из набора программных модулей, каждый из которых реализует определенную процедуру обработки данных. Программные модули должны иметь только одну точку входа и одну точку выхода. Только в этом случае отдельные модули можно разрабатывать и отлаживать независимо, а затем объединять в законченную прикладную программу с минимальными проблемами их взаимосвязи.

Источником подавляющего большинства ошибок программирования является использование модулей, имеющих один вход и несколько выходов. При необходимости организации множественных ветвлений в программе декомпозицию задачи выполняют таким образом, чтобы каждый функциональный модуль имел только один вход и один выход. Для этого условные операторы (имеющие два выхода) или включают внутрь модуля, объединяя их с операторами обработки, или выносят в систему межмодульных связей, формируя тем самым схему алгоритма более высокого ранга.

В международном стандарте на программный продукт HIPO (Hierarchy-Input-Process-Output) (“хай-по”) декларируется аналогичный подход к разработке прикладных программ.

Разработка схемы алгоритма функционального модуля программы имеет ярко выраженный итеративный характер, т.е. требует многократных проб, прежде чем возникает уверенность, что алгоритм реализации процедуры правильный и завершённый. Вне зависимости от функционального назначения процедуры при разработке её схемы необходимо придерживаться следующей очередности работы.

1. Определить, что должен делать модуль.
2. Определить способы получения модулем исходных данных (от датчиков через порты ввода, из таблиц в памяти или через рабочие регистры).
3. Определить необходимость какой-либо предварительной обработки введенных исходных данных (маскирование, сдвиг, масштабирование, перекодировка).
4. Определить метод преобразования входных данных в требуемые выходные. Используя операторы процедур и условные операторы принятия решения, отобразить на языке схемы алгоритма выбранный метод.
5. Определить способы выдачи из модуля обработанных данных (передать в память, в вызывавшую программу или в порты вывода).
6. Определить необходимость какой-либо вторичной обработки выводимых данных (изменение формата, перекодирование, масштабирование, маскирование).
7. Вернуться к пункту 1 настоящего перечня и проанализировать полученный результат. Выполнить итеративную корректировку схемы алгоритма с целью сделать её простой, логичной, стройной и обладающей чётким графическим образом.
8. Проверить работоспособность алгоритма на бумаге путем подстановки в него действительных данных. Убедиться в его сходимости и результативности.

9. Рассмотреть предельные случаи и попытаться определить граничные значения информационных объектов, с которыми оперирует алгоритм, за пределами которых он теряет свойства конечности, сходимости или результативности. Особое внимание при этом следует уделить анализу возможных ситуаций переполнения разрядной сетки, изменения знака результата операции, деления на переменную, которая может принять нулевое значение.
10. Провести мысленный эксперимент по определению работоспособности алгоритма в реальном масштабе времени, когда стохастические события, происходящие в объекте управления, могут оказать влияние на работу алгоритма. При этом самому тщательному анализу следует подвергнуть реакцию алгоритма на возможные прерывания с целью определения критических операторов, которые необходимо защитить от прерываний. Кроме того, в ходе этого мысленного эксперимента следует проанализировать логику алгоритма с целью определения таких последовательностей операторов, при выполнении которых микроконтроллер может “не заметить” кратковременных событий в объекте управления. При обнаружении таких ситуаций в логику следует внести коррективы.

Практика разработки программного обеспечения показала, что последовательное использование описанной поэтапной процедуры, составляющей основу метода структурного программирования, позволяет уверенно получать работоспособные прикладные программы.

Преобразование разработанной схемы алгоритма в исходный текст программы дело несложное. Но прежде чем приступить к написанию программы необходимо специфицировать память и выбрать язык программирования.

Спецификация памяти и рабочих регистров заключается в определении адреса первой команды прикладной программы, действительных начальных адресов стека, таблиц данных, буферных зон передачи параметров между процедурами, подпрограмм обслуживания прерываний и т.д. При этом следует помнить, что в микроконтроллерах память программ и память данных физически и логически разделены.

Диапазон языков написания исходного текста прикладной программы простирается от машинного кода до почти естественного языка. В машинном коде или на языке ассемблера программировать труднее, чем на алгоритмическом языке высокого уровня, но зато получается более короткий код программы, требуется меньшая ёмкость памяти программы и выполняется такая программа быстрее.

Объектные коды, полученные путем трансляции исходных программ, написанных на алгоритмическом языке высокого уровня, занимают в памяти больше места и требуют большего времени на исполнение. Выбор языковых средств составления исходных программ в каждом конкретном случае зависит от характеристик прикладной задачи, опыта программиста и допустимых затрат на разработку.

По мнению [1], огромное большинство прикладных задач управления объектами вследствие того, что они должны решаться в реальном времени, предъявляет столь высокие требования по быстродействию, что для их решения основным языковым средством написания прикладных программ еще долгие годы будет оставаться язык ассемблера. Это положение о преимущественном использовании языка ассемблера подкрепляется и тем обстоятельством, что однокристальные микроконт-

троллеры имеют ограниченный объем резидентной памяти программ и, следовательно, критичны к длине прикладных программ.

## 3.2. Процедуры и подпрограммы

При разработке микроконтроллерных систем могут быть использованы два способа организации прикладных программ: монолитный и модульный. При первом способе вся прикладная программа разрабатывается как единое целое. При втором она строится из отдельных программных блоков, каждый из которых реализует некоторую процедуру обработки данных или управления. Взаимосвязь блоков определяется разработчиком при монтаже из этих блоков законченной прикладной программы.

Отдельные фрагменты прикладной программы могут быть получены в виде линейной последовательности блоков, другие (многократно используемые) обычно оформляются в виде подпрограмм, к которым прикладная программа, называемая основной, имеет возможность обратиться по мере необходимости. Подпрограмма должна обладать следующими свойствами: выполнять законченную процедуру обработки данных, иметь только один вход и один выход и не обладать эффектом последействия, при котором текущее выполнение подпрограммы оказывало бы влияние на её последующие выполнения.

### *3.2.1. Вызов подпрограммы*

Обращение к подпрограмме осуществляется по команде вызова CALL MARK, где MARK - символическое имя процедуры. Имя процедуры используется в качестве метки, отмечающей одну из команд (чаще всего первую) подпрограммы. Для Intel 8051 мнемоническое значение CALL является обобщенным и транслируется в одну из команд ACALL или LCALL в зависимости от адресного расстояния вызываемой подпрограммы.

По команде CALL в стеке сохраняется значение счётчика команд, и возврат из подпрограммы осуществляется в то место основной программы, откуда был осуществлен вызов (к команде основной программы, следующей за командой CALL). Для этого любая подпрограмма должна заканчиваться командой возврата RET, осуществляющей восстановление содержимого программного счётчика из стека.

Достаточно часто возникает необходимость такой организации вычислительного процесса, при которой подпрограмма вызывает другую подпрограмму, та в свою очередь вызывает следующую и т.д. Этот процесс называется вложением подпрограмм. Число подпрограмм, которые могут быть вызваны таким образом (глубина вложенности подпрограмм), ограничивается только ёмкостью стека.

### *3.2.2. Сохранение параметров основной программы*

Иногда при обращении к подпрограмме возникает необходимость сохранить не только адрес возврата в основную программу, но и содержимое отдельных рабочих регистров. Удобным способом для этого является переключение банка регистров. Например, если основная программа использует банк регистров 0, то подпрограмма может использовать банк регистров 1. Однако переключение банка регистров не обеспечивает сохранение содержимого аккумулятора, что приводит к

необходимости создавать в одном из рабочих регистров или в памяти копию аккумулятора.

### *3.2.3. Передача параметров*

Для успешной работы любой подпрограммы необходимо однозначно определить способ передачи в неё исходных данных и способ вывода результата её работы. Подпрограмма, которой требуется дополнительная информация в виде параметров её настройки или операндов, называется параметризуемой.

Получили распространение четыре способа передачи параметров: через память, через регистры общего назначения, через регистр признаков и через стек.

При передаче входных параметров через память основная программа обязательно содержит команды загрузки некоторых ячеек памяти, а подпрограмма - команды считывания из этих ячеек. При передаче входных параметров подпрограмма должна загрузить некоторые ячейки памяти, а основная программа - считать.

Передача параметров через регистры осуществляется аналогичным образом.

Третий способ передачи параметров - через регистр признаков - удобно использовать при передаче выходных параметров (например, в подпрограммах сравнения чисел). В этом случае подпрограмма должна установить (или сбросить) соответствующие признаки, а основная программа - проанализировать их значение. Intel 8051 обладает большими возможностями для передачи параметров через признаки. В нём имеется 128 флагов пользователя, доступных для модификации и анализа.

Способ передачи через стек позволяет использовать в качестве параметра содержимое счётчика команд.

Использование процедур, оформленных в виде подпрограмм, при разработке программного обеспечения имеет ряд достоинств. Прежде всего относительно простые модули, выделенные из сложной программы, могут программироваться несколькими разработчиками с целью сокращения времени проектирования. Еще более важным является то, что любая подпрограмма допускает автономную отладку. Это, как правило, многократно сокращает время отладки всего прикладного программного обеспечения. И, наконец, механизм использования подпрограмм уменьшает длину прикладной программы, что имеет своим следствием уменьшение требующейся ёмкости памяти программ.

Существенным является и то обстоятельство, что отлаженные процедуры организуются разработчиками в библиотеки параметризуемых подпрограмм и могут быть многократно использованы в проектной работе. Библиотека подпрограмм должна строиться на основе соглашения о едином способе обмена параметрами.

## 3.3. Правила записи программ на языке ассемблера

Исходный текст программы на языке ассемблера имеет определенный формат. Каждая команда и директива представляет собой строку:

МЕТКА ОПЕРАЦИЯ ОПЕРАНД(Ы) КОММЕНТАРИИ

Поля могут отделяться друг от друга произвольным числом пробелов и табуляцией.

### *3.3.1. 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 A51 допускает использование в метках символа подчеркивания (_). Метка всегда завершается двоеточием (:).

Директивы ассемблера не преобразуются в двоичные коды, а потому не могут иметь меток. Исключение составляют директивы резервирования памяти и определения данных (DS, DB, DW). У директив, определяющих символические имена, в поле метки записывается определяемое символическое имя, после которого двоеточие не ставится.

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

### *3.3.2. Операция*

В поле операции записывается мнемоническое обозначение команды или директивы ассемблера, которое является сокращением (аббревиатурой) полного английского наименования выполняемого действия. Например: MOV - move - переместить, JMP - jump - перейти, DB - define byte - определить байт.

Для микроконтроллера Intel 8051 используется строго определенный и ограниченный набор мнемонических кодов. Любой другой набор символов, размещенный в поле операции, воспринимается ассемблером как ошибочный.

### *3.3.3. 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 указателем 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 (JMP @A) или выражение (JMP $ - 2, где $ - текущее содержимое счётчика команд).

Используемые в качестве операндов символические имена и метки должны быть определены, а числа представлены с указанием системы счисления, для чего используется суффикс (буква, стоящая после числа): B – для двоичной, Q – для восьмеричной, D – для десятичной и H – для шестнадцатиричной. Число без суффикса по умолчанию считается десятичным.

Ассемблер A51 допускает использование выражений в поле операндов, значения которых вычисляются в процессе трансляции.

Выражение представляет собой совокупность символических имен и чисел, связанных операторами ассемблера. Операторы ассемблера обеспечивают выпол-

нение арифметических (“+” - сложение, “-” - вычитание, “*” - умножение, “/” - целое деление, MOD – деление по модулю) и логических (OR - ИЛИ, AND - И, XOR - исключающее ИЛИ, NOT - отрицание) операций в формате 2-байтных 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 вычислить младший и старший байты 2-байтного операнда.

#### *3.3.4. Комментарий*

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

### 3.4. Директивы ассемблера

Ассемблер транслирует исходную программу в объектные коды. Хотя он берет на себя многие из рутинных задач программиста, такие как присвоение действительных адресов, преобразование чисел, присвоение действительных значений символьным переменным и т.п., программист всё же должен указать ей некоторые параметры: начальный адрес прикладной программы, конец ассемблируемой программы, форматы данных и т.п. Всю эту информацию программист вставляет в исходный текст прикладной программы в виде директив, которые только управляют процессом трансляции и не преобразуются в коды объектной программы.

Ассемблер поддерживает ряд директив, которые позволяют дать символическое определение переменным, резервируют и инициализируют пространство памяти, определяют расположение сгенерированного объектного кода в памяти. За исключением DB и DW директивы не производят объектный код. Директивы используются, чтобы изменить состояние ассемблера, определить объекты и добавить информацию к объектному файлу.

Директивы ассемблера могут быть разделены на ряд категорий:

- символические определения,
- резервирование пространства памяти,
- инициализация данных,
- управление состоянием ассемблера,
- выбор сегментов,
- определение макрокоманд.

Далее перечисляются все директивы по категориям и кратко описываются результаты их действия. Основные часто используемые директивы перечислены в алфавитном порядке в приложении. Информацию по остальным можно найти в справочной системе ProView.

#### *3.4.1. Директивы символических определений*

Директивы символических определений могут быть использованы для того, чтобы резервировать пространство памяти, поставить в соответствие символиче-

ским именам определённые числовые значения, регистры процессора и сегменты. Эти директивы требуют, чтобы имя символа было определено наряду с адресом, числовым значением, регистром или типом сегмента.

| Директива | Описание |
|---|---|
| **BIT** | Определяет символическое имя, ссылающееся на адрес бита. |
| **CODE** | Определяет символическое имя, ссылающееся на адрес кода. |
| **DATA** | Определяет символическое имя, ссылающееся на адрес резидентной памяти данных. |
| **EQU** | Назначает символическому имени числовое значение или имя регистра. |
| **IDATA** | Определяет символическое имя, ссылающееся на косвенно адресуемый адрес резидентной памяти данных. |
| **SEGMENT** | Объявляет имя перемещаемого сегмента, его тип и расположение. |
| **SET** | Назначает символическое имя числовому значению или регистру. Имя может быть впоследствии изменено с помощью директивы SET. |
| **XDATA** | Определяет символическое имя, ссылающееся на адрес внешней памяти данных. |

### *3.4.2. Директивы резервирования и инициализации памяти*

Эти директивы используются для резервирования и инициализации слов, байтов или битов. В абсолютном сегменте зарезервированное пространство начинается с текущего адреса. В перемещаемом сегменте зарезервированное пространство начинается с текущего смещения. Указатель расположения поддерживается отдельно для каждого сегмента, к нему можно обращаться, используя символ ($).

| Директива | Описание |
|---|---|
| **DB** | Заносит в память программ байтовую константу. |
| **DBIT** | Резервирует пространство в битовом сегменте. |
| **DS** | Резервирует пространство памяти в текущем сегменте. |
| **DW** | Инициализирует память значением слова. |

### *3.4.3. Директивы компоновки программы*

Вы можете использовать директивы компоновки программы для того, чтобы дать объектному модулю имя и определить общие и внешние символы. Эти директивы используются в L51 для объединения отдельных объектных модулей в единый абсолютный объектный модуль.

| Директива | Описание |
|---|---|
| **EXTRN** | Определяет символические имена, которые объявлены в других объектных модулях. |
| **NAME** | Определяет имя объектного модуля. |
| **PUBLIC** | Определяет символические имена, которые могут использоваться в других объектных модулях. |

### *3.4.4. Директивы управления состоянием ассемблера*

Эти директивы используются для того, чтобы сообщить о конце трансляции программы, выбрать начальный адрес или смещение для сегмента, определить используемый банк регистров.

| Директива | Описание |
|---|---|
| **END** | Сообщает о конце транслируемого модуля. |
| **ORG** | Изменяет значение ассемблерного счётчика адреса текущего сегмента программы. |
| **USING** | Выбирает номер банка регистров общего назначения. |

### *3.4.5. Директивы выбора сегмента*

Следующие директивы определяют сегменты данных и кода.

| Директива | Описание |
|---|---|
| **BSEG** | Выбирает абсолютный битовый сегмент. |
| **CSEG** | Выбирает сегмент программы в машинном коде. |
| **DSEG** | Выбирает абсолютный сегмент резидентной памяти данных. |
| **ISEG** | Выбирает абсолютный косвенно адресуемый сегмент резидентной памяти данных. |
| **RSEG** | Выбирает предварительно определенный перемещаемый сегмент. |
| **XSEG** | Выбирает абсолютный сегмент внешней памяти данных. |

### *3.4.6. Директивы макроопределений*

Следующие директивы используются для определения макрокоманд.

| Директива | Описание |
|---|---|
| **ENDM** | Заканчивает макроопределение. |
| **EXITM** | Заставляет макрорасширение немедленно завершиться. |
| **IRP** | Определяет список аргументов. |
| **IRPC** | Определяет аргумент. |
| **LOCAL** | Определяет до 16 локальных символов, используемых внутри макрокоманды. |
| **MACRO** | Начало макроопределения, определяет имя макрокоманды и параметров, которые могут быть переданы макрокоманде. |
| **REPT** | Определяет количество повторений последующих строк. |

## 3.5. Отладка прикладного программного обеспечения микроконтроллеров

После получения объектного кода прикладной программы наступает этап отладки, т.е. установления факта её работоспособности, а также выявления, локализации и устранения ошибок. Без этого этапа никакое программное обеспечение вообще не имеет права на существование. Отладка программного обеспечения представляет собой отдельную сложную задачу, которая почти не поддается формализации и требует для своего выполнения высокого профессионализма и глубоких знаний разработчика.

Обычно отладка прикладного программного обеспечения осуществляется в несколько этапов. Простые синтаксические ошибки выявляются уже на этапе трансляции. Далее необходимо выполнить:

- автономную отладку каждой процедуры в статическом режиме, позволяющую проверить правильность проводимых вычислений, правильность последовательности переходов внутри процедуры (отсутствие “зацикливания”) и т.п.;
- комплексную отладку программного обеспечения в статическом режиме, позволяющую проверить правильность алгоритма управления (по последовательности формирования управляющих воздействий);
- комплексную отладку в динамическом режиме без подключения объекта для определения реального времени выполнения программы и её отдельных фрагментов.

Следует иметь в виду, что автономная отладка отдельных модулей настолько проще и эффективнее отладки всей прикладной программы, что переходить к этапу комплексной отладки целесообразно только после исчерпания всех средств автономной отладки.

Вышеперечисленные этапы осуществляются обычно с использованием кросс-систем, к которым и относится изучаемый пакет ProView.

В состав кросс-систем входят программы-отладчики, моделирующие выполнение программ, написанных для микроконтроллеров. Такие программные имитаторы позволяют эффективно отлаживать вычислительные процедуры, а также сам алгоритм функционирования контроллера.

Разработчику предоставлен доступ к любому ресурсу микроконтроллера, имеется возможность покомандного и пофрагментного исполнения программ и останова по условию, а также подсчёта числа тактов выполнения тех или иных фрагментов программы, инициирования прерывания, дизассемблирования содержимого памяти программ и т.п.

Кросс-отладчики позволяют промоделировать практически все возможные варианты работы программы и тем самым убедиться в её работоспособности. На этом этапе возможна проверка работоспособности программы в нештатных ситуациях, в условиях поступления некорректных входных воздействий.

Мощные имитаторы должны позволять моделировать объекты и датчики, подключаемые к микроконтроллеру. При этом появляется возможность выполнять комплексную отладку программного обеспечения, не опасаясь, что возможные ошибки в программе, алгоритме или некорректные действия оператора приведут к выходу из строя технических средств разрабатываемой системы.

Главным недостатком кросс-систем является невозможность прогона программы в реальном масштабе времени.

Наиболее полная и комплексная отладка прикладного программного обеспечения совместно с аппаратными средствами контроллера может быть произведена на инструментальном компьютере с так называемым внутрисхемным эмулятором.

# 4. ЛАБОРАТОРНЫЕ ЗАДАНИЯ

## 4.1. Отладка подпрограммы вычисления скалярного произведения

С помощью пакета ProView выполните отладку подпрограммы вычисления скалярного произведения двоичных векторов, составленную при выполнении домашнего задания. Номер варианта исходных данных выбирается по последней цифре номера студенческого билета (зачётной книжки).

***Таблица 2***

*Варианты исходных данных*

| Вариант | X | Y |
|---|---|---|
| 0 | 10110010 | 10001001 |
| 1 | 11010101 | 11000011 |
| 2 | 01101010 | 10101010 |
| 3 | 00111100 | 11100111 |
| 4 | 10101010 | 00111100 |
| 5 | 11001111 | 00010111 |
| 6 | 11100111 | 00110011 |
| 7 | 11110101 | 01010101 |
| 8 | 01000111 | 10011001 |
| 9 | 00011110 | 11011011 |

Для отладки подпрограммы:

- создайте файл исходного текста программы на языке ассемблера, создайте файл проекта задач и добавьте к проекту файл исходного текста;
- выполните ассемблирование исходного текста и исправьте синтаксические ошибки;
- загрузите объектный код в процессе запуска отладчика;
- задайте начальные значения регистров и памяти;
- осуществите пробный пуск программы на контрольном примере;
- при наличии ошибок перейдите в пошаговый режим, локализуйте и исправьте имеющиеся ошибки;
- после исправления ошибок повторите пуск программы на контрольном примере и зафиксируйте полученный результат;
- выполните генерацию листинга программы, который включите в отчёт о выполнении работы, изучите все компоненты стандартного листинга.

## 4.2. Отладка варианта программы

С помощью пакета ProView выполните отладку вашего варианта программы, которая была подготовлена при выполнении домашнего задания. Для этого:

- создайте файл исходного текста программы на языке ассемблера, создайте

файл проекта задач и добавьте к проекту файл исходного текста;

- выполните ассемблирование исходного текста и исправьте синтаксические ошибки;
- загрузите объектный код в процессе запуска отладчика;
- задайте начальные значения регистров и памяти;
- осуществите пробный пуск программы на контрольном примере;
- при наличии ошибок перейдите в пошаговый режим, локализуйте и исправьте имеющиеся ошибки;
- после исправления ошибок повторите пуск программы на контрольном примере и зафиксируйте полученный результат;
- выполните генерацию листинга программы, который включите в отчёт о выполнении работы, изучите все компоненты стандартного листинга.

### 4.3. Задания для углублённого изучения

С помощью справочной системы пакета ProView изучите директивы ассемблера, описание которых не включено в приложение. Изучите раздел справки “Macro Processor”.

Модифицируйте Ваши программы с учётом новых знаний и выполните отладку программ.

## 5. СОДЕРЖАНИЕ ОТЧЁТА

Отчёт о лабораторной работе должен содержать:

- титульный лист;
- цель и задачи работы;
- листинги с исходными текстами и объектным кодом отлаженных программ с дополнениями, обеспечивающими тестирование и отладку;
- перечень ошибок, выявленных при отладке;
- результаты решения контрольных примеров;
- выводы по работе.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК


1. Сташин В.В., Урусов А.В., Мологонцева О.Ф. Проектирование цифровых устройств на однокристальных микроконтроллерах. М.: Энергоатомиздат, 1990. 224 с.
2. Однокристальные микроЭВМ/ А.В.Боборыкин, Г.П.Липовецкий, Г.В.Литвинский и др. М.: МИКАП, 1994. 400 с.
3. Ваша первая программа для микроконтроллера Intel 8051: Методические указания к лабораторной работе №1 по курсу “Микропроцессоры и вычислительные устройства”/ Добряк В.А. Екатеринбург: УГТУ, 1999. 32 с.
4. Система команд микроконтроллера Intel 8051: Методические указания к лабораторной работе №2 по курсу “Цифровые устройства и микропроцессоры”/ Добряк В.А., Рагозин В.К. Екатеринбург: УГТУ, 1999. 32 с.

## Приложение
## Директивы ассемблера в алфавитном порядке

Ниже дано описание основных, часто используемых директив, доступных в ассемблере A51 пакета ProView. Директивы могут включать ряд факультативных полей или аргументов:

| Имя | Описание |
|---|---|
| *Address* | Допустимый код или адрес данных. |
| *Argument* | Значение или выражение, заменяющее формальное имя параметра. |
| *Bit-address* | Допустимый адрес бита. |
| *Expression* | Допустимое выражение. |
| *Label* | Допустимый код программы или метка данных. |
| *Number* | Числовая константа, составленная только из цифр. |
| *Parameter* | Символическое имя формального параметра. |
| *Register* | Имя регистра: A, R0, R1, R2, R3, R4, R5, R6, или R7. |
| *String* | Строка символов и цифр. |
| *Symbol* | Допустимое символическое имя. |

### BIT

**Описание** Директива **BIT** назначает символическое имя адресу бита. Формат директивы:

*symbol* **BIT** *bit-address*

где ***symbol*** - символическое имя,
***bit-address*** - адрес бита в резидентной памяти данных.

Символические имена, определенные директивой **BIT,** не могут быть изменены или переопределены.

**Пример**

```
RSEG          DATA_SEG       ;выбор сегмента
CTRL:         DS   1         ;однобайтовая переменная (CTRL)
ALARM         BIT  CTRL.0    ;бит в перемещаемом байте
SHUT          BIT  ALARM+1   ;следующий бит
ENABLE_FLAG   BIT  60H       ;абсолютный бит
DONE_FLAG     BIT  24H.2     ;абсолютный бит
```

### DATA

**Описание** Директива **DATA** назначает символическое имя адресу резидентной памяти данных. Формат директивы:

*symbol* **DATA** *address*

где ***symbol*** - символическое имя, которое может использоваться во всей программе,
***address*** - адрес резидентной памяти данных, должен находится в

диапазоне от 0 до 255.

Символические имена, определенные этой директивой, не могут быть изменены или переопределены.

**Пример**

```
SERBUF  DATA SBUF
RESULT  DATA 40H
RESULT2 DATA RESULT + 2
PORT1   DATA 90H
```

## DB

**Описание** Директива **DB** заносит в память программ 8-разрядное значение байта. Директива имеет следующий формат:

*label:* **DB** *expression , expression ...*

где ***label:*** - метка, адрес инициализированной памяти,
***expression*** - значение байта, которое может быть символом, символьной строкой или выражением.

Директива DB может быть определена только внутри сегмента кода. Если директива используется в другом сегменте, ассемблер A51 генерирует сообщение об ошибке.

**Пример**

```
REQUEST:     DB   'PRESS ANY KEY TO CONTINUE', 0
TABLE:       DB   0, 1, 8, 'A', '0', Low(TABLE), ';'
ZERO:        DB   0, ''''
CASE_TAB:    DB   Low(REQUEST), Low(TABLE), Low(ZERO)
```

## DS

**Описание** Директива **DS** резервирует определенное число байтов в резидентной памяти данных, внешней памяти данных или адресном пространстве кода программы. Директива имеет следующий формат:

*label:* **DS** *expression*

где ***label:*** - метка, присвоенная адресу зарезервированной памяти,
***expression*** - количество зарезервированных байтов, не может содержать форвардные ссылки, перемещаемые символы или внешние символы.

Директива резервирует пространство в текущем сегменте по текущему адресу. Затем текущий адрес увеличивается на значение выражения. Сумма счётчика адреса и значения выражения не может превышать границу текущего адресного пространства.

***Примечание.*** *A51 - ассемблер с двумя проходами по исходному тексту программы. В первом проходе обрабатываются символы и определяется длина каждой команды. Во втором проходе обрабатываются форвардные ссылки и генери-*

*руется объектный код. По этим причинам выражение, используемое в директиве, не может содержать форвардные ссылки.*

**Пример**

```
GAP:    DS   (($ + 16) AND 0FFF0H) - $
        DS   20
TIME:   DS   8
```

## DW

**Описание** Директива **DW** инициализирует память программ 16-разрядными значениями слова. Директива имеет следующий формат:

*label:* **DW** *expression , expression ...*

где ***label:*** - метка, присвоенная адресу зарезервированной памяти,
***expression*** - выражения - данные, которые могут содержать символ, символьную строку или выражение.

Директива может быть определена только внутри сегмента кода. Если директива используется в другом сегменте, ассемблер A51 генерирует сообщение об ошибке.

**Пример**

```
TABLE:      DW   TABLE, TABLE + 10, ZERO
ZERO:       DW   0
CASE_TAB:   DW   CASE0, CASE1, CASE2, CASE3, CASE4
            DW   $
```

## END

**Описание** Директива **END** сообщает о конце ассемблерного модуля. Любой текст в ассемблерном файле, который появляется после этой директивы, игнорируется. Директива требуется в каждом исходном ассемблерном файле. Если директива отсутствует, ассемблер генерирует сообщение о фатальной ошибке.

**Пример**

```
END
```

## EQU

**Описание** Директива **EQU** назначает числовому значению или регистру символическое имя. Формат директивы:

*symbol* **EQU** *expression*

*symbol* **EQU** *register*

где ***symbol*** - символическое имя, которое заменяется на выражение или регистр во всей ассемблерной программе,

***expression*** - числовое выражение, не содержащее форвардных ссылок,

***register*** - одно из следующих имен регистра: A, R0-R7.

Символические имена, определенные директивой, могут использоваться в операндах, выражениях или адресах. Символы, которые определены как имя регистра, могут использоваться во всех командах, работающих с регистрами. Символические имена, определенные директивой, не могут быть изменены или переопределены.

**Пример**

```
LIMIT   EQU  1200
VALUE   EQU  LIMIT - 200 + 'A'
SERIAL  EQU  SBUF
ACCU    EQU  A
COUNT   EQU  R5
```

## ORG

**Описание** Директива **ORG** определяет адрес начала последующего кода программы или данных. Формат директивы:

**ORG** *expression*

где ***expression*** - должно быть абсолютным или простым перемещаемым выражением и не может иметь форвардных ссылок; символы, используемые в выражении, могут ссылаться только на текущий или абсолютный сегмент.

Когда ассемблер сталкивается с этой директивой, он вычисляет значение выражения и изменяет счётчик адресов (внутренняя переменная ассемблера) для текущего сегмента. Если директива находится в абсолютном сегменте, счётчику назначается значение истинного адреса. Если директива находится в перемещаемом сегменте, счётчику назначается смещение, определяемое выражением.

Директива изменяет счётчик адресов, но не производит новый сегмент. Неиспользованный диапазон адресов можно включить в текущий сегмент. Обратите внимание, что при использовании абсолютных сегментов счётчик не может ссылаться на адрес до начала смещения.

***Примечание.*** *A51 - ассемблер с двумя проходами по исходному тексту программы. В первом проходе обрабатываются символы и определяется длина каждой команды. Во втором проходе обрабатываются форвардные ссылки и генерируется объектный код. По этим причинам выражение, используемое в директиве, не может содержать форвардные ссылки.*

**Пример**

```
ORG     100H
ORG     RESTART
ORG     EXTI1
ORG     ($ + 16) AND 0FFF0H
```

## RSEG

**Описание** Директива **RSEG** выбирает перемещаемый сегмент, который был предварительно объявлен директивой SEGMENT. Формат директивы:

**RSEG** *segment*

где ***segment*** - имя сегмента, предварительно определенное директивой SEGMENT.

Для получения дополнительной информации относительно использования сегментов обратитесь к разделу справочной системы ProView “Assembly Programs”.

**Пример**

```
MYPROG  SEGMENT   CODE      ;объявление сегмента
        RSEG      MYPROG    ;выбор сегмента
        MOV       A, #0
        MOV       P0, A
```

## SEGMENT

**Описание** Директива **SEGMENT** используется для того, чтобы объявить перемещаемый сегмент. Тип сегмента может быть определен в объявлении сегмента. Директива имеет следующий формат:

*segment* **SEGMENT** *segtype reloctype*

где ***segment*** - символическое имя, назначенное сегменту,
***segtype*** - тип сегмента, определяющий адресное пространство сегмента; для получения дополнительной информации см. таблицу ниже,
***reloctype*** - тип перемещения для сегмента, который определяет параметры перемещения для компоновщика L51; обратитесь к таблице ниже для получения дополнительной информации.

Имя каждого сегмента внутри модуля должно быть уникально. Однако L51 объединяет сегменты одинакового типа. Это также применимо к сегментам, объявленным в других исходных модулях.

Переменная segtype определяет адресное пространство для сегмента и может быть любой из следующих:

| **Segtype** | **Описание** |
|---|---|
| BIT | Битовое адресное пространство (пространство резидентной памяти данных с адреса 20H по 2FH). |
| CODE | Пространство кода программы. |
| DATA | Пространство резидентной памяти данных (адреса с 0 по 127). |
| IDATA | Косвенно адресуемое пространство резидентной памяти данных (с 0 по 127 или с 0 по 255). |
| XDATA | Пространство внешней памяти данных. |

Факультативный параметр *reloctype* - тип перемещения определяет действия компоновщика L51. Следующая таблица содержит список допустимых типов настройки:

| **Reloctype** | **Описание** |
|---|---|
| BITADDRESSABLE | Определяет сегмент, который будет перемещен L51 внутрь битовой адресуемой области памяти (адреса с 20H по 2FH в резидентной памяти данных). Разрешён только для сегментов DATA, которые по длине не превышают 16 байтов. |
| INBLOCK | Определяет сегмент, который должен содержаться в 2048-байтовом блоке. Этот тип допустим только для сегментов CODE. |
| INPAGE | Определяет сегмент, который должен содержаться в 256-байтовом блоке. Этот тип настройки допустим только для сегментов CODE и XDATA. |
| OVERLAYABLE | Определяет, что сегмент может использовать память совместно с другими сегментами этого же типа. При использовании этого типа настройки имя сегмента должно быть объявлено согласно правилам C51 или PL/M-51. |
| PAGE | Определяет сегмент, чей начальный адрес должен быть в 256-байтовой границе. Размещение сегмента выполняется компоновщиком L51. Этот тип настройки допустим только для сегментов CODE и XDATA. |
| UNIT | Этот тип размещения задан по умолчанию как стандартный тип. Он определяет сегмент, который начинается в границе модуля. Модуль - байт для сегментов CODE, DATA, IDATA и XDATA и бит - для сегмента BIT. |

***Примечание.*** *Сегментные символы, используемые в выражениях, представляют собой базовый адрес объединенного сегмента, вычисляемый компоновщиком L51.*

Для получения дополнительной информации относительно использования сегментов обратитесь к разделу справочной системы ProView “Assembly Programs”.

**Пример**

```
STACK   SEGMENT   IDATA
        RSEG      STACK           ;выбор сегмента
        DS        10H             ;резервирование 16 байтов
        MOV       SP, #STACK - 1 ;инициализация SP
```

## SET

**Описание** Директива **SET** назначает символическое имя числовому значению или регистру. Формат директивы:

*symbol* **SET** *expression*

*symbol* **SET** *register*

где ***symbol*** - символическое имя, которое заменяется на выражение или регистр во всей ассемблерной программе,
***expression*** - числовое выражение, не содержащее форвардных ссылок,
***register*** - одно из следующих имен регистра: A, R0-R7.

Символические имена, определенные директивой, могут использоваться в операндах, выражениях или адресах. Символы, которые определены как имя регистра, могут использоваться во всех командах, работающих с регистрами. Имена, определенные директивой, могут быть изменены последующими директивами **SET**.

**Пример**

```
VALUE   SET  100
VALUE   SET  VALUE / 2
COUNTER SET  R1
TEMP    SET  COUNTER
TEMP    SET  VALUE * VALUE
```

## XDATA

**Описание** Директива **XDATA** назначает символическое имя адресу внешней памяти данных. Формат директивы:

*symbol* **XDATA** *address*

где ***symbol*** - символическое имя, которое может использоваться во всей программе,
***address*** - адрес внешней памяти данных, должен находиться в диапазоне от 0 до 65535.

Символические имена, определенные этой директивой, не могут быть изменены или переопределены.

**Пример**

```
RSEG    XSEG1
ORG     100H
DTIM:   DS        6    ;резервирует 6 байтов для DTIM
TIME    XDATA     DTIM + 0
DATE    XDATA     DTIM + 3
```

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРА
INTEL 8051 НА ЯЗЫКЕ АССЕМБЛЕРА

Составители Добряк Вадим Алексеевич
Рагозин Владимир Константинович

Редактор Н.П.Кубыщенко

Подписано в печать 08.02.99 Формат 60x84 1/16
Бумага типографская Офсетная печать Усл. п. л. 1,63
Уч.-изд. л. 1,44 Тираж 100 Заказ 40 Цена “С”

Издательство УГТУ
620002, Екатеринбург, Мира, 19
ЗАО УМЦ УПИ. 620002, Екатеринбург, Мира, 17