Министерство образования и науки Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра телевидения и управления (ТУ)

В.А. Кормилин

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

(1 часть)

Учебно-методическое пособие по организации самостоятельной работы по дисциплине

2018

Кормилин В.А.
Вычислительная техника и информационные технологии (1 часть): Учебно-методическое пособие по организации самостоятельной работы по дисциплине. Томск: Томский государственный университет систем управления и радиоэлектроники (ТУСУР), 2018. 22 с.

Учебно-методическое пособие предназначено для студентов радиотехнических направлений подготовки ТУСУРа, обучающихся на всех формах обучения и содержит учебный материал и методические указания для организации самостоятельной работы при изучении дисциплины.

© Кормилин В.А., 2018

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

При разработке указаний по организации самостоятельной работы учитывалась необходимость формирования и оценки достижения общепрофессиональных компетенций, связанных с данной дисциплиной в рабочем учебном плане.

Формируемые знания, умения и навыки связаны со следующими компетенциями:

ОПК-2 – способностью решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением инфокоммуникационных технологий и с учетом основных требований информационной безопасности;

ОПК-3 – способностью владеть основными методами, способами и средствами получения, хранения, переработки информации.

1 СОСТАВ ДИСЦИПЛИНЫ

Дисциплина «Вычислительная техника и информационные технологии» рассчитана на два семестра и читается в четвертом и пятом семестрах.

Объем дисциплины «Вычислительная техника и информационные технологии» (1 часть) и виды учебной деятельности в 4-м семестре по учебному плану 2016 года набора составляют:

Вид учебной деятельности	4 семестр	Единицы
Лекции	20	часов
Практические занятия	16	часов
Лабораторные работы	16	часов
Всего аудиторных занятий	52	часов
Самостоятельная работа	56	часов
Общая трудоемкость по дисциплине	108	часов
Дифференцированный зачет		часов
Зачетные единицы	3	З.Е.

Формой промежуточной аттестации по учебному плану 2016 года набора в 4 семестре по дисциплине является дифференцированный зачет, т.е. зачет с оценкой , выставляемой по итогам обучения в семестре.

Объем дисциплины «Вычислительная техника и информационные технологии» (1 часть) и виды учебной деятельности в 4-м семестре по учебному плану 2018 года и последующих лет набора составляют:

Вид учебной деятельности	4 семестр	Единицы
Лекция	20	часов
Практическая работа	16	часов
Лабораторная работа	16	часов
Всего аудиторных занятий	52	часов
Самостоятельная работа	56	часов
Трудоемкость без экзамена	108	часов
Подготовка и сдача экзамена	36	часов
Общая трудоемкость по дисциплине	144	часов
Зачетные единицы	4	З.Е.

Формой промежуточной аттестации по учебному плану 2018 года и последующих лет набора является экзамен.

Объем самостоятельной работы составляет 56 часов независимо от года набора.

2 СОДЕРЖАНИЕ САМОСТОЯТЕЛЬНОЙ ПОДГОТОВКИ

Самостоятельная работа включает: проработку материала лекций, подготовку к практическим занятиям, подготовку к контрольным работам, подготовку к выполнению лабораторных работ и написанию отчетов по ним, изучение тем теоретической части курса, отводимых на самостоятельную проработку.

Количество часов, отводимых на самостоятельную работу, и формы контроля самостоятельной работы сведены в таблицу.

№	Наименование работы	Часы	Формы контроля
1.	Подготовка к лекциям	10 час	устный выборочный опрос по 5 мин перед лекцией
2.	Подготовка к практическим занятиям	8 час	устный выборочный опрос по 5 мин перед практикой
3.	Подготовка к лабораторным работам и оформление отчетов по ЛР	16 час	Допуск к лаборат. работам, защита отчетов по ЛР
4.	Изучение тем (вопросов) теоретической части курса, отводимых на самостоятельную проработку: 1. Выполнение арифметических действий в модифицированном дополнительном коде. 2. Программные и аппаратные средства создания и отладки программного обеспечения. 3. Система команд микроконтроллера MCS-51.	12 час	устный выборочный опрос по 5 мин перед лекцией
5.	Подготовка к контрольным работам: 1. Преобразование чисел между системами счисления. 2. Арифметические действия в разных системах счисления. 3. Микропроцессоры в системах управления. 4. ОМК MCS-51. 5. Периферийные устройства вычислительной техники	10 час	20 мин. контрольные работы перед практикой
	ИТОГО объем СРС	56 час	
	Подготовка и сдача экзамена	36 час	Оценка на экзамене

3 ПОДГОТОВКА К ЗАНЯТИЯМ

Подготовка к лекционным занятиям выполняется по материалу учебного пособия «Вычислительная техника и информационные технологии (1 часть)» Учебное пособие.

Подготовка к практическим занятиям выполняется по материалу пособия «Вычислительная техника и информационные технологии»(1 часть)» Учебно-методическое пособие по организации практических занятий.

Подготовка к выполнению лабораторных работ выполняется по материалу пособия «Методические указания по практическим занятиям по дисциплине «Вычислительная техника и информационные технологии (1 часть)».

4 ТЕМЫ ДЛЯ САМОСТОЯТЕЛЬНОЙ ПРОРАБОТКИ

4.1 Выполнение арифметических действий в модифицированном дополнительном коде

На самостоятельную проработку выносится тема «Выполнение арифметических действий в модифицированном дополнительном коде», описанная в учебном пособии в разделе «Системы счисления в ЭВМ» в подразделе «Арифметические операции с числами».

Краткое изложение раздела.

Сложение в модифицированном коде.

Модифицированный код отличается наличием двух одинаковых знаковых битов. В модифицированном дополнительном коде у положительных чисел знаковые разряды равны нулю, а у отрицательных – единице. При выполнении операций правильным результатом будет число с одинаковыми знаковыми битами. Наличие комбинации 01 или 10 в знаковых разрядов однозначно показывает переполнение разрядной сетки. А это убережет от использования ошибочного результата в дальнейших вычислениях. Рассмотрим сложение в модифицированном дополнительном коде. Сначала проверим предыдущий пример:

```
          [00]1100001
          [00]0111101
          ----------
           [01]0011110
переносы        11     1
```

Мы видим, что результат содержит разные знаковые биты, поэтому он ошибочен. Ошибка объясняется тем, что для размещения значащей части результата необходимо 8, а не 7 разрядов, поскольку $N_1 = 97_{10}$, а $N_2 = 61_{10}$. Результат $N_1 + N_2 = 158_{10}$ больше предельного значения 127.

Рассмотрим сложение отрицательных чисел в модифицированном дополнительном коде. Числа уже преобразованы в дополнительный код и равны N_1 = [11]0110100 и N_2 = [11]0101101.

```
          [11]0110100
        + [11]0101101
          [10]1100001
переносы    11   1111
```

В результирующем числе знаковые биты 10 сигнализируют о переполнении разрядной сетки, т.е. результат не помещается в семи двоичных разрядах. Действительно, если перевести числа в десятичную систему счисления, мы получим $N_1 = -76_{10}$, а $N_2 = -83_{10}$. Результат $N_1 + N_2 = -159_{10}$. Это действительно превышает значение -128_{10}, которое является наибольшим по модулю отрицательным числом, занимающим семь двоичных разрядов.

Рассмотрим другие примеры.

```
          [00]0110100
        + [00]0101101
          [00]1100001
переносы      1111
```

Суммирование положительных чисел дало правильный результат. Действительно, $N_1 = 52_{10}$, а $N_2 = 45_{10}$. Результат $N_1 + N_2 = 97_{10}$. Рассмотрим еще один пример.

```
          [11]1100001
        + [11]0111101
          [11]0011110
переносы    11 11    1
```

Результат правильный и в десятичных числах записывается как $N_1 = -31_{10}$, а $N_2 = -67_{10}$. Результат $N_1 + N_2 = -98_{10}$. Для следующего примера возьмем числа с разными знаками $N_1 = -52_{10}$, а $N_2 = +45_{10}$. Результат $N_1 + N_2 = -7_{10}$.

```
          [11]1001100
        + [00]0101101
          [11]1111001
переносы      11
```

Результат получился правильным и представлен в модифицированном дополнительном коде. Можете поверить на слово, что и сложение большого по модулю положительного числа с маленьким (по модулю) отрицательным числом даст правильный положительный результат.

4.2 Программные и аппаратные средства создания и отладки программного обеспечения

На самостоятельную проработку выносится тема «Программные и аппаратные средства создания и отладки программного обеспечения».

Краткое изложение раздела.

Кросс-технология разработки программного обеспечения. Микро-ЭВМ и микроконтроллеры (МК) для радиоэлектронных устройств и систем предназначены для решения задач управления и обработки сигналов. От обычных микро-ЭВМ они отличаются способом разработки программного обеспечения. Микро-ЭВМ и МК, встроенные в оборудование, не имеют соответствующего набора необходимых внешних устройств (дисплеи, принтеры, внешние накопители и т.д.) и поэтому не пригодны для разработки и отладки программного обеспечения. Часто в этом случае программы разрабатываются и отлаживаются на универсальных ЭВМ, с использованием пакета специальных программ. При этом приходится разрабатывать программы на языке ассемблера. Отладку выполняют с помощью симулятора – программной модели микро-ЭВМ или контроллера.

Этот способ разработки программного обеспечения называется кросс-технологией, в противовес резидентной технологии. При разработке программ на языке ассемблера используют различные служебные программы – редакторы текста, трансляторы, компоновщики, отладчики программ.

В кросс-технологии применяются: кросс-транслятор, кросс-компоновщик, кросс-отладчик. В минимальный пакет кросс-программ разработки программного обеспечения (ПО) для управляющей микро-ЭВМ входят:

- ***редактор текста***, обеспечивающий ввод программы на языке ассемблера микро-ЭВМ или МК, редактирование и запись файла программы;
- ***кросс-транслятор***, преобразующий исходный текст программы в файл объектного кода, содержащего коды команд управляющей микро-ЭВМ (МК), информацию для редактора связей, предупреждения и сообщения об ошибках;
- ***кросс-компоновщик объектных модулей (кросс-редактор связей)***. Компоновщик связывает несколько объектных модулей в единую программу, добавляет вызываемые из программы библиотечные файлы, подставляет вместо ссылок на объекты их реальные или относительные адреса и формирует исполняемый файл в кодах микро-ЭВМ или МК;
- ***кросс-отладчик (симулятор)***, описывающий программную логическую модель микро-ЭВМ или микроконтроллера. При работе симулятор дает эффективный и удобный механизм покомандной и модульной отладки компонентов программы с доступом к программно эмулируемым ресурсам микро-ЭВМ: регистрам, ячейкам памяти, таймерам, портам и т.д.

Перед составлением текста программы необходимо разработать алгоритм решения поставленной задачи в виде последовательного набора шагов (элементарных действий), позволяющий решить требуемую задачу.

Каждую строку алгоритма распишем в виде команд – текст программы готов. С этого момента нужны специальные программные средства.

Общая процедура создания и отладки программ для микро-ЭВМ или МК включает следующие этапы, показанные на рисунке **1.**

Исходный текст программы с помощью редактора текста вводится в память универсальной ЭВМ и записывается в файл. Затем с помощью кросс-транслятора этот файл переводится в промежуточную форму – объектный файл. Объектный файл требует дополнительной обработки, которая выполняется с помощью программы – кросс-редактора связей. На выходе редактора связей формируется двоичный файл, который может быть выполнен на реальной микро-ЭВМ или МК. Для проверки функционирования созданной программы можно выполнить ее отладку в симуляторе. Отлаженный файл программы в кодах микро-ЭВМ (МК) записывается с помощью программатора в постоянную память (ПЗУ) микропроцессорной системы.

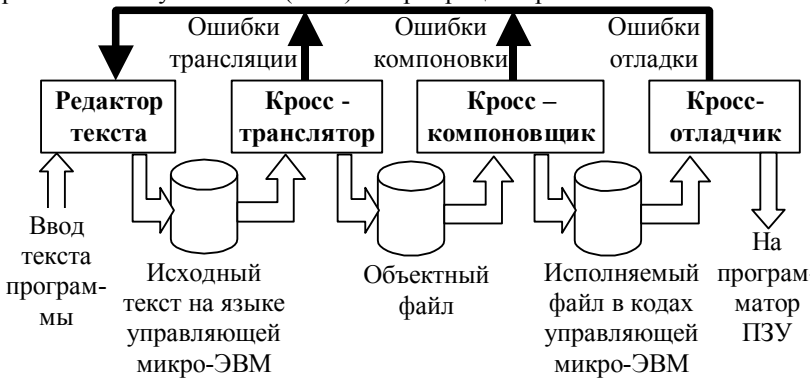

Рисунок 1 – Этапы процедуры создания ПО для управляющей микро-ЭВМ

В простейших случаях операции трансляции и компоновки могут выполняться совместно без записи объектного файла на диск. При этом транслятор и компоновщик объединяются в одной программе, которая выполняет несколько последовательных проходов (просмотров) файла, и преобразует исходный текст программы в исполняемый код.

На каждом из этапов создания программного обеспечения могут быть обнаружены различные ошибки. Это приводит к необходимости возврата к исходному тексту программы, исправления его и повторения всей процедуры преобразования.

Ошибки трансляции связаны с неправильными типами операндов команд, пропуском необходимых элементов оформления текста программы,

опечатками, синтаксическими ошибками записи команд и другими ошибками, обнаруживаемыми по внешнему виду команды. Эти ошибки являются самыми простыми и легко устраняются при внимательном прочтении указанной транслятором ошибочной строки.

Ошибки компоновки проявляются на этапе работы редактора связей и означают неправильные указания для объединения нескольких модулей подпрограмм в единую общую программу. Здесь встречаются ссылки на неизвестные программы и метки, ошибки в именах программных модулей, неправильное использование библиотечных функций. Осознание и устранение подобных ошибок, как правило, требует простой внимательности и не вызывает больших трудностей.

Нередко компоновщики при обнаружении ошибок, которые на их взгляд не являются критическими, выдают просто предупреждения. При этом они создают исполняемый файл. В таком файле ошибочные ссылки заменяются стандартными значениями, например, нулями. Опасность скрывается в игнорировании предупреждений компоновщика и использовании исполняемого файла с ошибками связей. В этом случае результат работы программы непредсказуем.

Ошибки исполняемого файла связаны с ошибками алгоритма, неправильной работой программы, зацикливаниями, ошибками счета, ошибками преобразования и другими ошибками. Такие ошибки проявляются и обнаруживаются только при исполнении программы. Как правило, это самые трудные для диагностирования и устранения ошибки. Анализ сложной и громоздкой управляющей или обрабатывающей программы требует внимательности, памяти, хороших знаний поведения объекта управления и занимает продолжительное время.

Ситуация несколько упрощается при использовании программных моделей – симуляторов. Перед записью программы в ПЗУ, ошибки данного этапа можно обнаружить, исследовать и найти пути устранения с помощью программы кросс-отладчика. Программная модель микро-ЭВМ позволяет проводить анализ исполняемого кода с использованием большого числа сервисных возможностей, которые отсутствуют в реально работающей аппаратной ЭВМ.

Перечисленные выше возможности обеспечивают на разных этапах создания и преобразования программ выявление и устранение большинства ошибок. Однако нельзя гарантировать устранение всех ошибок на 100%. Кроме перечисленных, остаются еще динамические ошибки, проявляющие себя лишь при реальной работе аппаратной микро-ЭВМ. Эти ошибки могут быть связаны с реальными режимами прерываний, условиями функционирования портов, таймеров, ячеек памяти. По этой причине достаточно сложные программные модули даже после тщательной отладки могут еще содержать не выявленные ошибки и обычно их содержат.

4.3 Система команд микроконтроллера MCS-51

На самостоятельную проработку выносится тема «Система команд микроконтроллера MCS-51», описанная в учебном пособии в разделе «Однокристальный микроконтроллер семейства MCS-51» в подразделе «Система команд ОЭВМ КР1816ВЕ51».

Краткое изложение раздела.

Система команд MCS-51 содержит 111 базовых команд. По функциональному признаку команды можно разделить на пять групп:

a) команды передачи данных;
b) арифметические команды;
c) логические команды;
d) команды передачи управления;
e) команды операций с битами.

Большинство команд (94) имеют формат один или два байта и выполняются за один или два машинных цикла. Остальные команды – трехбайтные и выполняются за два машинных цикла. Только на выполнение команд умножения и деления требуется четыре машинных цикла.

Первый байт всех 13 типов команд содержит код операции (КОП). Второй и третий байты содержат либо адреса операндов, либо непосредственные данные.

В системе команд MCS-51 используется четыре способа адресации данных.

1. Регистровая адресация. В команде указано символическое название регистра.
2. Прямая адресация. В команде явно указан адрес операнда в памяти.
3. Непосредственная адресация. В команде задан сам операнд.
4. Косвенная адресация. В команде задано имя регистра или регистровой пары, где содержится адрес, по которому и происходит обращение к операнду в памяти.

В системе команд используются следующие обозначения (в скобках показаны варианты обозначений, встречающиеся в командах, с буквой h обозначаются старшие байты слов, с буквой l – младшие байты слов):

Rn, n=0÷7 – регистры R0-R7 текущего банка регистров;

direct (ad, add, ads) – 8-битный прямой адрес ячейки резидентной памяти данных (0-127) или регистра специальных функций (128-255);

@Ri, i=0,1 – 8-битная ячейка РПД или регистра специальных функций, косвенно адресуемая через регистр R0 или R1;

#data (#d) – 8-битная константа, входящая в состав команды;

#data16 (#d16h, #d16l) – 16-битная константа, входящая в состав команды (непосредственная адресация);

addr16 (ad16h, ad16l) – 16-битовый адрес назначения в командах длинного перехода LJMP и вызова подпрограмм LCALL, позволяющий охватить полное адресное пространство в 64 Кбайт;

addr11 – 11-битовый адрес назначения в командах абсолютного перехода AJMP и вызова подпрограмм ACALL, позволяющий выполнить переход в пределах 2 Кбайт страницы от адреса текущей команды;

rel – 8-битное смещение со знаком. Смещение с учетом знака суммируется с текущим содержимым счетчика команд и позволяет выполнить переход в пределах –128 +127 байт относительно текущего адреса;

bit – 8-битный прямой адрес бита, находящегося в РПД (0-127) или в регистре специальных функций (128-255).

Микроконтроллер оперирует данными четырех типов: биты, тетрады (4-битные цифры), байты и 16-битные слова.

Биты. В MCS-51 определены 256 битов, из них 128 находятся в РПД и используются как флаги пользователя, а остальные 128 битов расположены в блоке регистров специальных функций и входят в состав некоторых регистров этого блока. Для обращения к битам используется прямой 8-битный адрес (bit).

Тетрады. Операции с тетрадами определены только в командах обмена тетрад в аккумуляторе и между регистром и аккумулятором.

Байты. Байтовым операндом может быть адрес внутренней и внешней памяти данных и программ, непосредственный операнд, константа, адрес регистра специальных функций или порта ввода/вывода.

Слова. Двухбайтовые операнды могут быть или константами, или прямыми адресами памяти и занимают второй и третий байты команды.

В слове состояния процессора PSW имеются четыре флага, отражающие результат операции в АЛУ: C – перенос, AC – вспомогательный перенос, OV – переполнение и P – паритет. Только некоторые команды изменяют флаги результата.

Команды передачи данных

Команды передачи данных не влияют на флаги результата.

Большую часть команд данной группы составляют команды передачи и обмена байтами.

На графе возможных путей передачи данных в МК (рисунок **2 22**) аккумулятор представлен отдельной вершиной, поскольку многие команды используют его при передаче данных.

Полный перечень команд передачи данных представлен в приложении А «Список команд ОЭВМ MCS-51» в подразделе «Группа команд передачи данных».

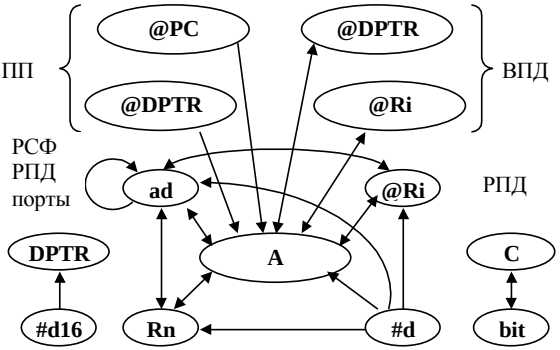

Рисунок 2 – Граф путей передачи данных

Арифметические команды

В данную группу команд входят команды сложения, вычитания, умножения и деления байтов. Имеются команды десятичной коррекции результата сложения, увеличения на единицу как 8-битовых, так и 16-битовых операндов, уменьшения на единицу для 8-битовых операндов.

Все команды этой группы используют АЛУ и влияют на состояние флагов.

Полный перечень арифметических команд представлен в приложении А «Список команд ОЭВМ MCS-51» в подразделе «Группа арифметических операций».

Логические команды

В данную группу входят команды, выполняющие логические операции вида «И», «ИЛИ», «Исключающее ИЛИ» над операндом в аккумуляторе и в любом байте РПД или в регистрах специальных функций. Сюда же входят команды сдвига аккумулятора влево и вправо, а также сброса и инверсии аккумулятора.

Все команды этой группы используют АЛУ и влияют на состояние флагов.

Полный перечень логических команд представлен в приложении А «Список команд ОЭВМ MCS-51» в подразделе «Группа логических операций».

Команды операций с битами

Данные команды оперируют с однобитовыми операндами. Биты расположены в области битовой адресации (128 битов пользователя) и в области регистров специальных функций. Для адресации бит используется прямой 8-битовый адрес (bit). Косвенная адресация бит невозможна.

Команды не затрагивают флагов, кроме флага переноса в отдельных случаях.

Полный перечень команд операций с битами представлен в приложении А «Список команд ОЭВМ MCS-51» в подразделе «Группа битовых операций».

Команды передачи управления

Данная группа команд содержит команды условных и безусловных переходов, вызовы подпрограмм и возврат из них, команды сравнения и цикла. Для выполнения условных переходов можно проверять разнообразные условия, а безусловные переходы и вызовы подпрограмм различаются по предельной удаленности перехода

Полный перечень команд передачи управления представлен в приложении А «Список команд ОЭВМ MCS-51» в подразделе «Группа команд передачи управления».

5 ПОДГОТОВКА К КОНТРОЛЬНЫМ РАБОТАМ

Для подготовки к контрольным работам следует проработать следующие разделы лекционных разделов учебного пособия.

Для контрольной работы «Преобразование чисел между системами счисления» следует проработать раздел 2.2 учебного пособия.

Для контрольной работы «Арифметические действия в разных системах счисления» следует проработать раздел 2.3 учебного пособия.

Для контрольной работы «Микропроцессоры в системах управления» следует проработать раздел 3 учебного пособия.

Для контрольной работы «ОМК MCS-51» следует проработать раздел 4 учебного пособия.

Для контрольной работы «Периферийные устройства вычислительной техники» следует проработать раздел 5 учебного пособия.

ПРИЛОЖЕНИЕ А

(справочное)

СПИСОК КОМАНД ОЭВМ MCS-51

Мнемокод	Название	айт	икл	Операция
Группа команд передачи данных				
MOV @Ri,#d	Пересылка в РПД константы (i=0,1)	2	1	**((Ri))←#d**
MOV @Ri,A	Пересылка в РПД байта из аккумулятора	1	1	**((Ri))←(A)**
MOV @Ri,ad	Пересылка в РПД байта с прям.адреса ad	2	2	**((Ri))←(ad)**
MOV A,#d	Загрузка в аккумулятор константы	2	1	**(A)←#d**
MOV A,@Ri	Пересылка в аккумулятор из РПД (i=0,1)	1	1	**(A)←((Ri))**
MOV A,ad	Пересылка в аккумулятор байта с адреса ad	2	1	**(A)←(ad)**
MOV A,Rn	Пересылка в аккумулятор байта из регистра (n=0÷7)	1	1	**(A)←(Rn)**
MOV ad,#d	Пересылка константы по прям.адресу	3	2	**(ad)←#d**
MOV ad,@Ri	Пересылка по прямому адресу байта из РПД (i=0,1)	2	2	**(ad)←((Ri))**
MOV ad,A	Пересылка аккумулятора по прямому адресу	2	1	**(ad)←(A)**
MOV ad,Rn	Пересылка регистра по прямому адресу	2	2	**(ad)←(Rn)**
MOV add,ads	Пересылка прямоадресуемого байта по прямому адресу	3	2	**(add)←(ads)**
MOV DPTR,#d16	Загрузка указателя данных словом	3	2	**(DPTR)←#d16**
MOV Rn,#d	Загрузка в регистр (n=0÷7) константы	2	1	**(Rn)←#d**
MOV Rn,A	Пересылка в регистр (n=0÷7) байта из аккумулятора	1	1	**(Rn)←(A)**
MOV Rn,ad	Пересылка в регистр с прямого адреса	2	2	**(Rn)←(ad)**

Мнемокод	Название	байт	цикл	Операция
MOVC A,@A+DPTR	Пересылка в аккумулятор байта из ПП	1	2	**(A)←((A)+(DPTR))**
MOVC A,@A+PC	Пересылка в аккумулятор байта из ПП	1	2	**(A)←((A)+(PC))**
MOVX @DPTR,A	Пересылка в расшир. ВПД из аккумулятора	1	2	**((DPTR))←(A)**
MOVX A,@DPTR	Пересылка в аккумулятор из расшир. ВПД	1	2	**(A)←((DPTR))**
MOVX @Ri,A	Пересылка в ВПД из аккумулятора	1	2	**((Ri))←(A)**
MOVX A,@Ri	Пересылка в аккумулятор байта из ВПД	1	2	**(A)←((Ri))**
POP ad	Извлечь из стека	2	2	**(ad)←((SP))** **(SP)←(SP)-1**
PUSH ad	Загрузить в стек	2	2	**(SP)←(SP)+1** **((SP))←(ad)**
XCH A,@Ri	Обмен аккумулятора с байтом из РПД	1	1	**(A)↔((Ri))**
XCH A,ad	Обмен аккумулятора и байта с адреса ad	2	1	**(A)↔(ad)**
XCH A,Rn	Обмен аккумулятора с регистром	1	1	**(A)↔(Rn)**
XCHD A,@Ri	Обменять младш. тетраду аккумулятора с мл. тетрадой байта РПД	1	1	**$(A_{0-3})\leftrightarrow((Ri)_{0-3})$**
SWAP A	Обменять тетрады в аккумуляторе	1	1	**$(A_{0-3})\leftrightarrow(A_{4-7})$**
Группа арифметических операций				
ADD A,#d	Сложить аккумулятор с константой	2	1	**(A)←(A)+#d**
ADD A,@Ri	Сложить аккумулятор с байтом из РПД	1	1	**(A)←(A)+((Ri))**
ADD A,ad	Сложить аккумулятор с байтом по адресу ad	2	1	**(A)←(A)+(ad)**
ADD A,Rn	Сложить аккумулятор с регистром	1	1	**(A)←(A)+(Rn)**
ADDC A,#d	Сложить аккумулятор с константой и переносом	2	1	**(A)←(A)+#d+(C)**
ADDC A,@Ri	Сложить аккумулятор с байтом из РПД (i=0,1) и переносом	1	1	**(A)←(A)+((Ri))+(C)**

Мнемокод	Название	байт	цикл	Операция
ADDC A,ad	Сложить аккумулятор с байтом по адресу ad и переносом	2	1	**(A)←(A)+(ad)+(C)**
ADDC A,Rn	Сложить аккумулятор с регистром (n=0÷7) и переносом	1	1	**(A)←(A)+(Rn)+(C)**
DA A	Десятичная коррекция аккумулятора	1	1	
SUBB A,#d	Вычесть из аккумулятора константу и заем	2	1	**(A)←(A)-#d-(C)**
SUBB A,@Ri	Вычесть из аккумулятора байт из РПД и заем	1	1	**(A)←(A)-((Ri))-(C)**
SUBB A,ad	Вычесть из аккумулятора байт с адреса ad и заем	2	1	**(A)←(A)-(ad)-(C)**
SUBB A,Rn	Вычесть из аккумулятора регистр и заем	1	1	**(A)←(A)-(Rn)-(C)**
INC @Ri	Инкремент байта в РПД	1	1	**((Ri))←((Ri))+1**
INC A	Инкремент аккумулятора	1	1	**(A)←(A)+1**
INC ad	Инкремент байта по адр. ad	2	1	**(ad)←(ad)+1**
INC DPTR	Инкремент указателя данных	1	2	**(DPTR)←(DPTR)+1**
INC Rn	Инкремент регистра	1	1	**(Rn)←(Rn)+1**
DEC @Ri	Декремент байта в РПД	1	1	**((Ri))←((Ri))-1**
DEC A	Декремент аккумулятора	1	1	**(A)←(A)-1**
DEC ad	Декремент байта по адресу ad	2	1	**(ad)←(ad)-1**
DEC Rn	Декремент регистра	1	1	**(Rn)←(Rn)-1**
MUL AB	Умножение аккумулятора на регистр B	1	4	**(B)(A)←(A)·(B)**
DIV AB	Деление аккумулятора на регистр B	1	4	**(A).(B)←(A)/(B)**
Группа логических операций				
ANL A,#d	Логическое И аккумулятора и константы	2	1	**(A)←(A)∧#d**
ANL A,@Ri	Логическое И аккумулятора с байтом из РПД (i=0,1)	1	1	**(A)←(A)∧((Ri))**
ANL A,ad	Логическое И аккумулятора с байтом по адресу ad	2	1	**(A)←(A)∧(ad)**
ANL A,Rn	Логическое И аккумулятора с регистром (n=0÷7)	1	1	**(A)←(A)∧(Rn)**
ANL ad,#d	Логическое И байта по адресу ad с константой	3	2	**(ad)←(ad)∧#d**

Мнемокод	Название	байт	цикл	Операция
ANL ad,A	Логическое И байта по адресу ad с аккумулятором	2	1	**(ad)←(ad)∧(A)**
ORL A,#d	Логическое ИЛИ аккумулятора с константой	2	1	**(A)←(A)∨#d**
ORL A,@Ri	Логическое ИЛИ аккумулятора с байтом из РПД (i=0,1)	1	1	**(A)←(A)∨((Ri))**
ORL A,ad	Логическое ИЛИ аккумулятора с байтом по адресу ad	2	1	**(A)←(A)∨(ad)**
ORL A,Rn	Логическое ИЛИ аккумулятора с регистром (n=0÷7)	1	1	**(A)←(A)∨(Rn)**
ORL ad,#d	Логическое ИЛИ байта по адресу ad с константой	3	2	**(ad)←(ad)∨#d**
ORL ad,A	Логическое ИЛИ байта по адресу ad с аккумулятором	2	1	**(ad)←(ad)∨(A)**
XRL A,#d	Исключ. ИЛИ аккумулятора с константой	2	1	**(A)←(A)⊕#d**
XRL A,@Ri	Исключающее ИЛИ аккумулятора с байтом из РПД (i=0,1)	**1**	**1**	**(A)←(A)⊕((Ri))**
XRL A,ad	Исключающее ИЛИ аккумулятора с байтом по адресу ad	2	1	**(A)←(A)⊕(ad)**
XRL A,Rn	Исключающее ИЛИ аккумулятора с регистром (n=0÷7)	1	1	**(A)←(A)⊕(Rn)**
XRL ad,#d	Исключающее ИЛИ байта по адресу ad с константой	3	2	**(ad)←(ad)⊕#d**
XRL ad,A	Исключающее ИЛИ байта по адресу ad с аккумулятором	2	1	**(ad)←(ad)⊕(A)**
CLR A	Сброс аккумулятора	1	1	**(A)←0**
CPL A	Инверсия аккумулятора	1	1	**(A)←($\overline{A}$)**
RL A	Сдвиг аккумулятора влево по циклу	1	1	**(A_{n+1})←(A_n), n=0÷6 (A_0)←(A_7)**
RR A	Сдвиг аккумулятора вправо по циклу	1	1	**(A_n)←(A_{n+1}), n=0÷6 (A_7)←(A_0)**
RLC A	Сдвиг аккумулятора влево через перенос	1	1	**(A_{n+1})←(A_n), n=0÷6 (A_0)←(C) (C)←(A7)**
RRC A	Сдвиг аккумулятора вправо через перенос	1	1	**(A_n)←(A_{n+1}), n=0÷6 (A_7)←(C) (C)←(A_0)**

Мнемокод	Название	байт	цикл	Операция
Группа битовых операций				
MOV bit,C	Пересылка разряда переноса в бит	2	2	**(b)←(C)**
MOV C,bit	Пересылка бита в разряд переноса	2	1	**(C)←(b)**
ANL C,bit	Логическое И бита и переноса	2	2	**(C)←(C)∧(b)**
ANL C,/bit	Логическое И инверсии бита и переноса	2	2	**(C)←(C)∧($\overline{b}$)**
ORL C,bit	Логическое ИЛИ бита и переноса	2	2	**(C)←(C)∨(b)**
ORL C,/bit	Логическое ИЛИ инверсии бита и переноса	2	2	**(C)←(C)∨($\overline{b}$)**
CLR bit	Сброс бита	2	1	**(b)←0**
CLR C	Сброс переноса	1	1	**(C)←0**
SETB bit	Установка бита	2	1	**(b)←1**
SETB C	Установка переноса	1	1	**(C)←1**
CPL bit	Инверсия бита	2	1	**(b)←($\overline{b}$)**
CPL C	Инверсия переноса	1	1	**(C)←($\overline{C}$)**
Группа команд передачи управления				
LCALL ad16	Длинный вызов программы	3	2	**(PC)←(PC)+3, (SP)←(SP)+2, (SP-1)←(PC_{0-7}), (SP)←(PC_{8-15}), (PC)←ad16**
ACALL ad11	Абсолютный вызов подпрограммы в странице 2 Кбайта	2	2	**(PC)←(PC)+2, (SP)←(SP)+2, (SP-1)←(PC_{0-7}), (SP)←(PC_{8-15}), (PC_{0-10})←ad11**
LJMP ad16	Длинный переход	3	2	**(PC)←ad16**
AJMP ad11	Абсолютный переход в странице размером 2 Кбайт	2	2	**(PC_{0-10})←ad11**
SJMP rel	Короткий относительный переход	2	2	**(PC)←(PC)+2+rel**
CJNE A,ad,rel	Сравнение аккумулятора с байтом по адресу ad и переход, если не равно	3	2	**(PC)←(PC)+3, если (A)≠(ad), то (PC)←(PC)+rel**
CJNE @Ri,#d,rel	Сравнение байта из РПД с константой и переход, если не равно	3	2	**(PC)←(PC)+3, если ((Ri))≠#d, то (PC)←(PC)+rel**

Мнемокод	Название	байт	цикл	Операция
CJNE A,#d,rel	Сравнение аккумулятора с константой и переход, если не равно	3	2	**(PC)←(PC)+3, если (A)≠#d, то (PC)←(PC)+rel**
CJNE Rn,#d,rel	Сравнение регистра с константой и переход, если не равно	3	2	**(PC)←(PC)+3, если (Rn)≠#d, то (PC)←(PC)+rel**
DJNZ ad,rel	Декремент байта по адресу ad и переход, если не нуль	3	2	**(PC)←(PC)+2, (ad)←(ad)-1, если (ad)≠0, то (PC)←(PC)+rel**
DJNZ Rn,rel	Декремент регистра и переход, если не нуль	2	2	**(PC)←(PC)+2, (Rn)←(Rn)-1, если (Rn)≠0, то (PC)←(PC)+rel**
JB bit,rel	Переход, если бит равен единице	3	2	**(PC)←(PC)+3, если (b)=1, то (PC)←(PC)+rel**
JBC bit,rel	Переход, если бит установлен, с последующим сбросом бита	3	2	**(PC)←(PC)+3, если (b)=1, то (b)←0, (PC)←(PC)+rel**
JC rel	Переход, если перенос равен единице	2	2	**(PC)←(PC)+2, если (C)=1, то (PC)←(PC)+rel**
JMP @A+DPTR	Косвенный относительный переход	1	2	**(PC)←(A)+(DPTR)**
JNB bit,rel	Переход, если бит равен нулю	3	2	**(PC)←(PC)+3, если (b)=0, то (PC)←(PC)+rel**
JNC rel	Переход, если перенос равен нулю	2	2	**(PC)←(PC)+2, если (C)=0, то (PC)←(PC)+rel**
JNZ rel	Переход, если аккумулятор не равен нулю	2	2	**(PC)←(PC)+2, если (A)≠0, то (PC)←(PC)+rel**
JZ rel	Переход, если аккумулятор равен нулю	2	2	**(PC)←(PC)+2, если (A)=0, то (PC)←(PC)+rel**

Мнемокод	Название	ай т	и кл	Операция
RET	Возврат из подпрограммы	1	2	**$(PC_{8-15})\leftarrow((SP))$ $(PC_{0-7})\leftarrow((SP)-1)$ $(SP)\leftarrow(SP)-2$**
RETI	Возврат из подпрограммы обработки прерывания	1	2	**$(PC_{8-15})\leftarrow((SP))$ $(PC_{0-7})\leftarrow((SP)-1)$ $(SP)\leftarrow(SP)-2$**
NOP	Нет операции	1	1	**$(PC)\leftarrow(PC)+1$**