МИНИСТЕРСТВО СЕЛЬСКОГО ХОЗЯЙСТВА РФ

БРЯНСКАЯ ГОСУДАРСТВЕННАЯ
СЕЛЬСКОХОЗЯЙСТВЕННАЯ АКАДЕМИЯ

КАФЕДРА ИНФОРМАТИКИ

ИЗУЧЕНИЕ МИКРОПРОЦЕССОРНОЙ ТЕХНИКИ НА ПРИМЕРЕ МИКРОЭВМ СЕМЕЙСТВА МК51

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ
С МЕТОДИЧЕСКИМИ УКАЗАНИЯМИ
К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

БРЯНСК 2009

УДК 621. 38: 681. 32 (075)
ББК 32.973.26-04
Б 39

Безик Д.А. Изучение микропроцессорной техники на примере микроЭВМ семейства МК51. Учебно-методическое пособие с методическими указаниями к выполнению лабораторных работ. Брянск. Издательство Брянской ГСХА, 2009г.- 100 с.

Учебно-методическое пособие с методическими указаниями предназначено для изучения основных понятий микропроцессорной техники на примере однокристальных микроЭВМ семейства МК51. Предназначено для студентов, обучающихся по специальности 110302 – электрификация и автоматизация сельского хозяйства, а также аспирантов и лиц, занимающихся изучением и разработкой микропроцессорной техники.

Рецензенты:
1. К. э. н., доцент, зав. кафедрой информационных систем и технологий Ульянова Н. Д.
2. К. т. н., доцент кафедры электротехнологий, В. А. Башлыков.

Рекомендовано к изданию методической комиссией факультета энергетики и природопользования от 16 марта 2009 г. протокол № 21.

© Брянская ГСХА, 2009
© Безик Д.А., 2009

Содержание

Введение

Характерной особенностью сегодняшнего этапа развития электронной техники является широкое применение микропроцессорной техники. Сейчас она применяется не только в сфере обработки данных, но и там, где раньше господствовала аналоговая техника – регуляторы, фильтры и др. Однокристальные микроЭВМ получили большое распространение благодаря простоте применения, гибкости построенных на их основе систем, универсальности применения. Сейчас на одном кристалле формируется не только процессор, память программ и данных, но и много других устройств – последовательные порты, цифроаналоговые и аналого-цифровые преобразователи, энергонезависимая память, системы для внутрисхемного программирования и т. п.

Ранее программы писались для ЭВМ на языках высокого уровня, а затем преобразовывались в машинные коды. Результат был не всегда оптимален и зависел от компилятора. Сейчас же появилась тенденция оптимизации микроЭВМ под конкретные языки программирования высокого уровня (например, PIC-микроконтроллёры под язык СИ).

МикроЭВМ семейства МК51 появились давно, но успешно выпускаются и функционируют по сей день. Их изучение позволяет понять основные принципы построения ЭВМ, их программирования и использования.

Изучение микроЭВМ МК51 следует базировать на основе знаний по основам цифровой техники и рассматривать как первый этап освоения сложных микропроцессорных систем.

Представленное учебно-методическое пособие предназначено для студентов, изучающих курсы «электроника, микропроцессорная техника и техника связи», «технические средства ПЭВМ», «архитектура вычислительных систем». Также оно может оказаться полезным для аспирантов и лиц, занимающихся изучением и разработкой микропроцессорной техники.

Лабораторная работа №1

«Исследование основ программирования микропроцессора семейства МК51»

Цель работы: изучение основ программирования микропроцессора семейства МК51.

Приборы и принадлежности: стенд СУ-МК для изучения микропроцессоров серии МК51, интегрированная среда программирования ProView, ПК, справочные материалы по ОМЭВМ МК51.

Теоретическое введение

Микропроцессоры серии МК51 представляют собой восьмиразрядные однокристальные микроЭВМ, выполненные по n-МОП или КМОП технологии. Их разработчиком была фирма Intel (микропроцессор 8051), но аналоги этой ИС выпускают многие другие производители, в том числе и в нашей стране. В таблице 1 приведены краткие сведения по ИС производимых у нас.

Таблица 1 - Микросхемы серии МК51

Микросхема	Аналог	Объём внутренней памяти программ	Тип памяти программ	Объём внутренней памяти данных	Максимальная тактовая частота тактовых сигналов	Ток потребления
КР1816ВЕ31	8031АН	-	внеш.	128	12 МГц	150 мА
КР1816ВЕ51	8051АН	4 кБайт	ПЗУ	128	12 МГц	150 мА
КР1816ВЕ751	8751Н	4 кБайт	ППЗУ	128	12 МГц	220 мА
КР1830ВЕ31	80С31ВН	-	внеш.	128	12 МГц	18 мА
КР1830ВЕ31	80С51ВН	4 кБайт	ПЗУ	128	12 МГц	18 мА

Однокристальные микроЭВМ (ОМЭВМ) серии МК51 содержат все узлы, необходимые для автономной работы:

1. центральный восьмиразрядный процессор;
2. память программ 4 кбайт (не у всех);
3. память данных 128 байт;
4. 4 параллельных восьмиразрядных порта ввода-вывода;
5. два шестнадцатиразрядных таймера-счётчика;
6. систему прерываний (два уровня, 5 векторов);
7. последовательный интерфейс;
8. тактовый генератор.

Структурная схема описываемого микропроцессора представлена на рисунке 1. Из рисунка следует, что ОМЭВМ состоит из нескольких блоков, соединённых между собой восьмиразрядными шинами адреса и данных, а также линиями управления. Основа ОМЭВМ – арифметико-логическое устройство (АЛУ), которое выполняет основные операции – сложение вычитание, умножение и т. д. АЛУ содержит в своём составе два основных регистра (ячейки памя-

ти) – аккумулятор А и вспомогательный регистр В. Большинство операций микропроцессора производится с участием аккумулятора, например при сложении одно из слагаемых обязательно должно быть помещено в него, также перенос данных во внешнюю память возможен только из аккумулятора. Регистр В используется АЛУ в частности при умножении и т. п.

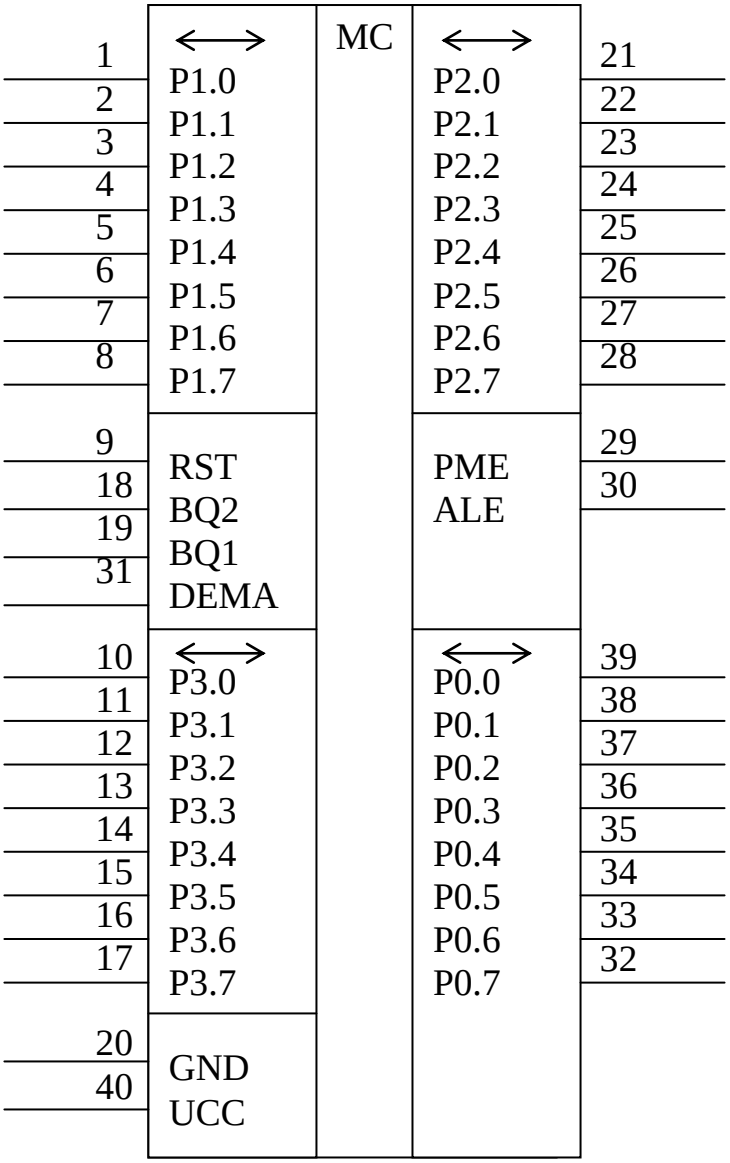
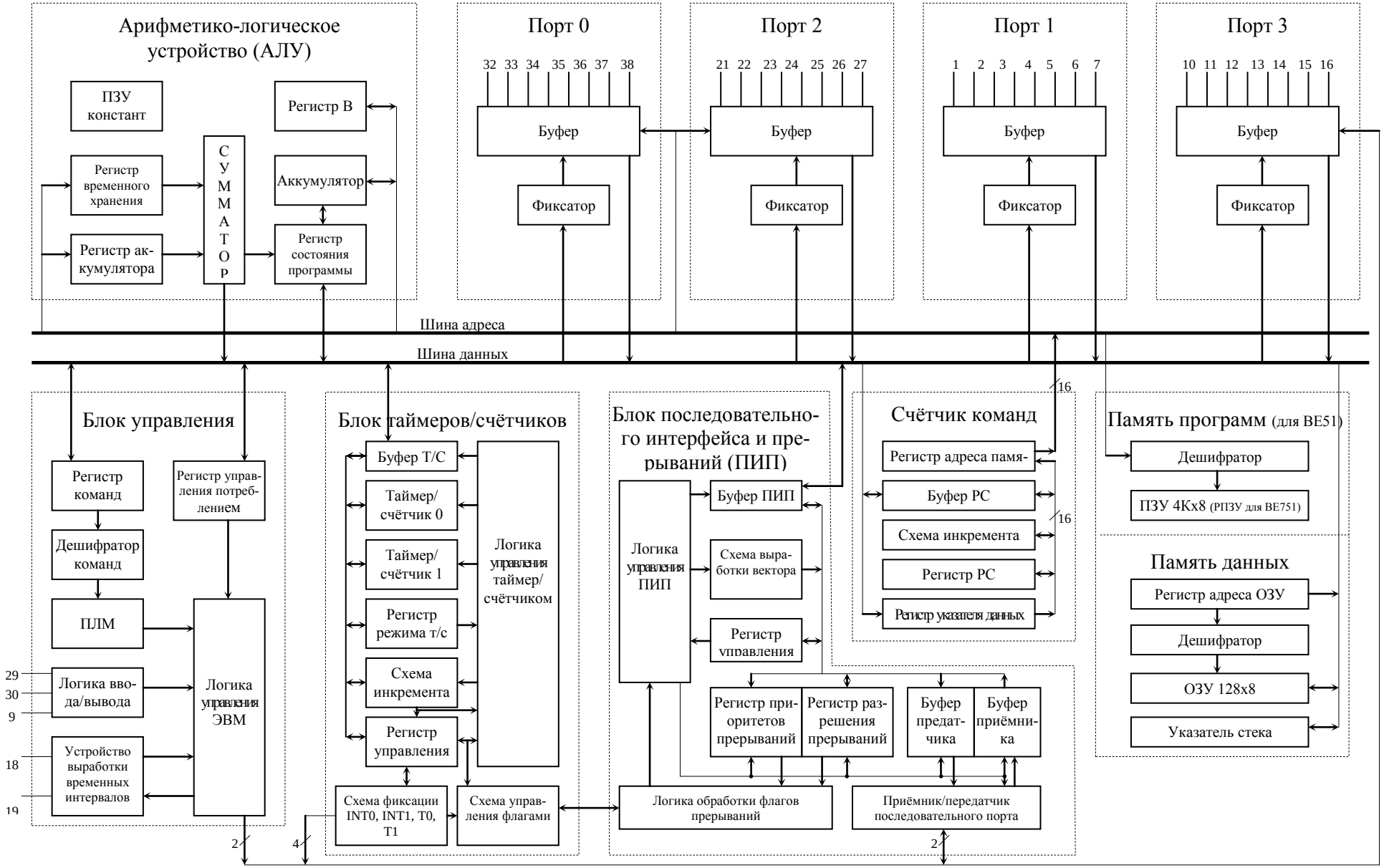


Рисунок 2 - Условное обозначение ОМЭВМ.

Обмен информацией между ОМЭВМ и внешним миром осуществляется по четырём параллельным восьмиразрядным портам, что хорошо видно из рисунка 2, на котором показано условное обозначение ОМЭВМ (эти порты обозначены как P0, P1, P2, P3 с указанием через точку номеров битов). Обмен последовательными данными, подсчёт числа внешних событий, приём прерываний, обмен с внешней памятью происходит также через эти выводы.

СТРУКТУРНАЯ СХЕМА ОМЭВМ СЕРИИ МК-51



ОМЭВМ серии МК51 кроме 32 выводов для параллельного ввода-вывода имеет выводы питания (+5 В - UCC, 0 В - GND), выводы для подключения внешнего кварцевого резонатора для задания тактовой частоты (BQ1, BQ2), вход сброса RST и три входа для организации обмена с внешней памятью (PME, ALE, DEMA).

В простейшем случае для управления внешними устройствами с помощью этой ОМЭВМ надо источник питания, кварцевый резонатор и необходимые цепи внешних устройств. Для примера на рисунке 3 приведёна простейшая схема, управляющая реле. В этом примере имеется минимум внешних цепей (RC-цепь для сброса при включении питания и резонатор с конденсаторами для задания тактовой частоты) и ключ на биполярном транзисторе, коммутирующий реле. Программа хранится в ОМЭВМ и, например, производит отсчёт времени (10 мин), в течение которого контакты реле замкнуты. Изменив программу можно изменить интервал времени, а добавив кнопку и один резистор, можно сделать отсчёт времени при её нажатии, и т. д. Важно то, что путём небольших схемных изменений (или вообще без них) путём изменения программы можно менять свойства устройства. Именно для этого и создавались микропроцессоры.

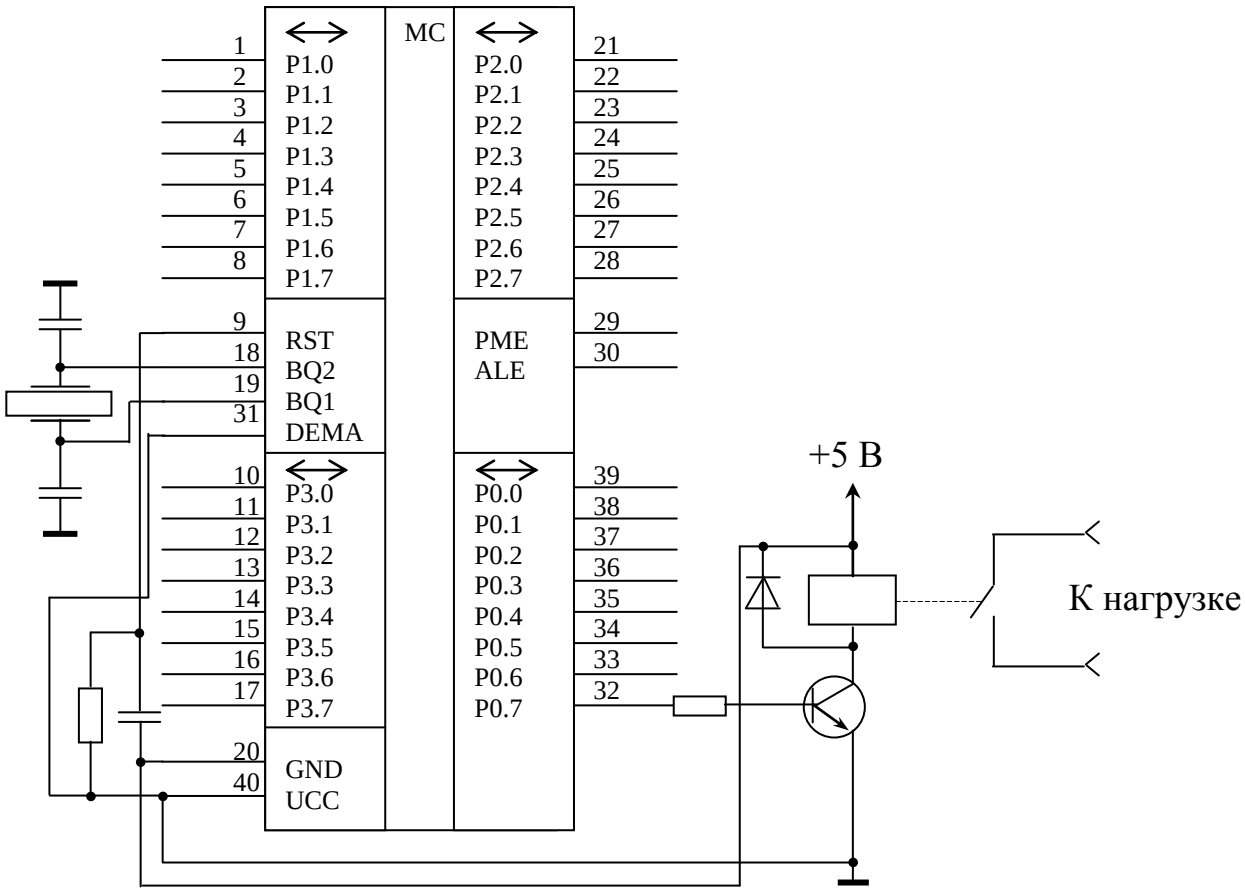


Рисунок 3 - Пример микропроцессорной системы, управляющей реле.

Сказанное выше относится к взгляду на ОМЭВМ электронщика. С точки зрения программиста важна система команд, структурная организация ОМЭВМ, организация памяти и управляющих устройств.

Система команд МК51 включает в себя 111 команд. Многие команды имеют похожее назначение и мнемоническое обозначение, так команда **mov** для байтных команд имеется в 15 вариантах, а **da** только в одном. Всего есть 42 мнемонических обозначения (аббревиатур) команд и их несложно запомнить. Все команды приведены в приложении 1. В этой лабораторной работе понадобятся только 6 команд.

Для написания программы важно знать, как распределена память данных. У семейства МК51 память данных и память программ разделены. Распределением памяти программ занимается компилятор, который программу на языке ассемблера (или на каком-либо другом языке) переводит в программу в машинных кодах, которая затем записывается в память программ МП. Поэтому в несложных случаях можно не обращать внимание на распределение программы в памяти.

Внутренняя память данных адресуется 8 битами, поэтому всего возможна адресация 2^8 =256 ячеек памяти. Обращение к внутренней и внешней памяти производится с помощью разных команд, поэтому их ошибочное пересечение невозможно. Структура внутренней памяти показана на рисунке 4. Все младшие 128 байт можно использовать как ячейки памяти для хранения данных, а вот верхние байты с адресами от 080h до 0ffh образуют блок регистров специальных функций (SFR).

В области адресов от 080h до 0ffh расположены основные регистры арифметико-логического устройства, порты ввода-вывода, регистры конфигурации и т. п. В принципе их можно использовать и для хранения данных, но им отведены другие функции и прибегать к хранению в них данных следует очень осторожно. Надо заметить, что если ячейки с адресами 00h-7Fh физически присутствуют все, то от 80h до 0FFh есть только 11 ячеек, к которым возможно обращение. Попытка обращения к несуществующим ячейкам ни к чему, ни приведёт.

Обращение к ячейкам памяти возможно по их адресам, например 07h или 65h, но к некоторым возможно обращение и по символическим именам. Эти имена указаны справа от ячеек на рисунке 4. Так запись в аккумулятор можно произвести записав данные в ячейку с адресом 0E0h или в A (при обращении к аккумулятору как к байту в ассемблере записывается A, а при обращении к его битам – ACC).

ОМЭВМ МК51 имеют команды обработки отдельных битов. Для упрощения работы с отдельными битами некоторые из них имеют собственные битовые адреса, они указаны внутри байтов на рисунке 4. Так чтобы обратиться к пятому биту ячейки с адресом 25h, следует использовать адрес бита 2Dh, а к пятому биту аккумулятора – 0E5h. Кроме того, для обращения к битам регистров специальных функций можно использовать их символические имена (указаны сверху над битами), а также запись названия регистра и номера бита через точку. Например, для того чтобы обратиться к флагу переноса можно использовать три варианта – C, 0D7h, PSW.7.

Адресное пространство в 32 байта от 00h до 1Fh разбито на четыре банка по восемь ячеек памяти (регистров). Кроме непосредственного обращения к этим ячейкам по их адресам возможно и обращение по именам R0…R7. К какому банку регистров в данный момент ведётся обращение определяется *указателем банка рабочих регистров* - битами RS0 и RS1 регистра состояния программы PSW (см. рис. 4). Например, запись в регистр R2 при RS0=0 и RS1=0 будет осуществлена в ячейку 02h (банк 0), а при RS0=0 и RS1=1 – в ячейку 12h (банк 2).

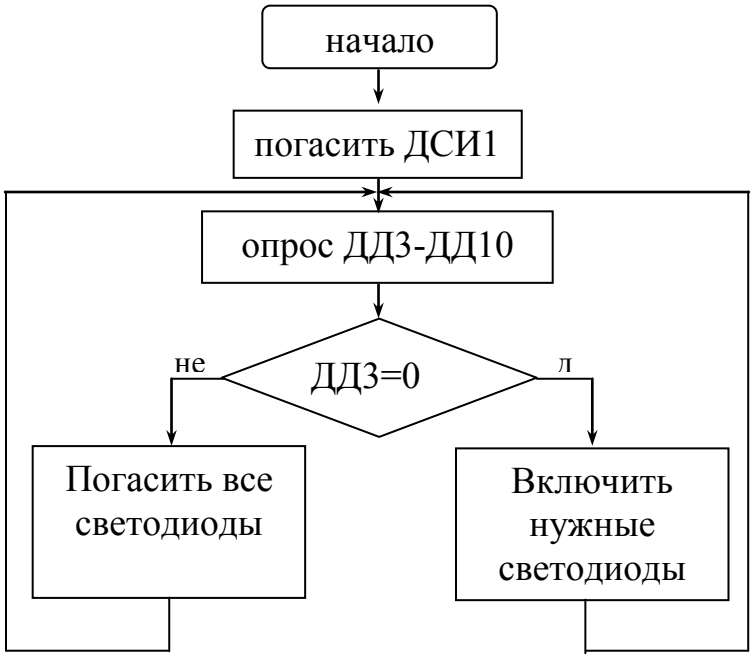
Ошибка! Раздел не указан.

Рисунок 4 - Внутренняя память ОМЭВМ МК51.

Пример выполнения лабораторной работы

Разберём пример написания простейшей программы для МК51, использующий приёмы побитного и побайтного ввода-вывода. Допустим необходимо запрограммировать стенд так, чтобы при выключенном переключателе ДД3 не горели светодиоды ДСИ1-ДСИ10 (все), а при его включении светодиоды загорались через один, т. е. 2, 4, 6, 8 и 10-й.

Для начала составим алгоритм программы. Первым делом надо погасить светодиод №1, т. к. неясно, в каком он будем состоянии при включении стенда (состояние остальных зависит от ДД3, поэтому оно будет определено далее). Затем надо опросить состояние переключателей ДД3-ДД10 (из-за особенностей конструктивного исполнения стенда опрос производится сразу для всех переключателей ДД3-ДД10). После этого если ДД3 выключен то надо погасить светодиоды, а если включён – то включить нужные. Чтобы стенд всегда реагировал на ДД3, а не только после включения, этот процесс надо повторять циклически, т. е. повторять сначала вновь и вновь. Схематично алгоритм изображён справа.



В программе придётся обращаться к отдельным светодиодам и переключателям. Покажем, как это делается.

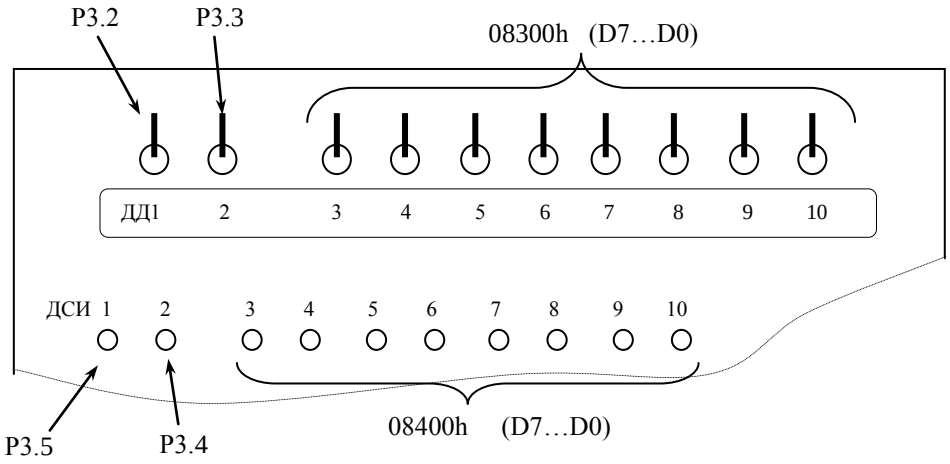


Рисунок 5 - Адреса индикаторов и переключателей стенда СУ-МК.

Переключатели ДД1 и ДД2 подключены непосредственно к линиям порта P3, а точнее к второму и третьему разряду (с точки зрения программы – биту) этого порта. Поэтому обращаться к ним надо как к битам байта P3, т. е. P3.2 и P3.3 (см. рисунок 5). Если из бита считано значение «0», то это означает, что переключатель выключен, а «1» - включен. Аналогично и светодиоды ДСИ1 и ДСИ2. Они подключены к разрядам 5 и 4 того же порта и обращаться к ним надо по именам P3.5 и P3.4. Так, чтобы включить индикатор ДСИ1 надо записать единицу (установить бит) в P3.5, а чтобы его погасить – записать «0» (сбросить бит). Для установки и сброса бита можно использовать команды **setb bit** и **clr bit**, где вместо **bit** должно стоять имя бита (1, 5 и 10 строки программы приведённой ниже).

Индикаторы ДСИ3…ДСИ10 и переключатели ДД3…ДД10 подключены через вспомогательную микросхему к ОМЭВМ и обращение к ним ведётся как к внешним ячейкам памяти. Для индикаторов адрес – 08400h, а для переключателей – 08300h. Это шестнадцатиразрядные шестнадцатеричные адреса и действительны только правые 4 цифры. Ноль слева добавлен только для того, чтобы было ясно, что это числа, а не строка символов. Обращение к внешней памяти возможно через команду **movx**. Одним из её операндов обязательно должен быть аккумулятор. Вторым операндом в нашем случае будет регистр указателя данных **DPTR.** В него предварительно надо занести адрес ячейки внешней памяти, в которую (из которой) будут извлекаться данные. Так, чтобы включить индикаторы ДСИ3-ДСИ10 надо выполнить три строки программы:

```
mov DPTR,#08400h     ; загрузка в указатель данных адреса
                      ;светодиодных индикаторов
mov A,#0FFh
movx @DPTR,A         ; запись байта по адресу индикаторов
```

В этом фрагменте сначала в регистр указателя данных DPTR заносится адрес индикаторов в шестнадцатеричном виде (показывает буква h). Решётка # указывает, что 08400 константа, а не адрес ячейки. Затем в аккумулятор заносится константа FF (с нулём слева, чтобы было ясно, что это число, а не строка). И, наконец, число FF из аккумулятора переписывается во внешнюю ячейку памяти, адрес которой находится в DPTR. Символ @ показывает, что запись будет произведена не в сам регистр, указанный в команде, а в ячейку с адресом, хранящейся в этом регистре. Из этих трёх команд видно, что после имени команды идут операнды, причём сначала пункт назначения, а затем источник.

В алгоритме, приведённом выше, надо осуществлять ветвление программы в зависимости от состояния переключателя ДД3. После считывания по адресу 08300h в аккумулятор будет помещён байт, каждый бит которого соответствует одному переключателю. ДД3 будет соответствовать старший седьмой бит и обращаться к нему следует по имени ACC.7, хотя можно и непосредственно по битовому адресу 0E7h (см. рисунок 4). Для ветвления можно использовать команду **jb bit,metka**. По ней происходит переход на метку с именем **metka**, если бит **bit** установлен, то есть равен единице, в противном случае никакого перехода не осуществляется и происходит выполнение команд, следующих за командой **jb**. В нашей программе эта команда выглядит так:

```
jb ACC.7,vpravo
```

Здесь если седьмой бит аккумулятора равен единице, то выполнение программы перейдёт к строке с меткой **vpravo**. Имя метки может быть составлено из букв и цифр и заканчиваться двоеточием в той строке, на которую она указывает.

Чтоды осуществить безусловный переход (у нас переход от действий со светодиодами к опросу переключателей), можно использовать команду **sjmp metka**. Когда программа доходит до строки с этой командой происходит переход на метку **metka.**

В конце программы следует указать ключевое слово **end.**

Полный текст программы нашего примера приведён ниже.

```
      clr P3.5         ; предварительное выключение ДСИ1
nachalo: mov DPTR,#08300h  ; загрузка в указ. данных адреса дискретных входов
      movx A,@DPTR      ; считывание состояния переключателей 3-10
      jb ACC.7,vpravo  ; переход, если ACC.7=1 на метку vpravo
      clr P3.4         ; выключение ДСИ2
      mov DPTR,#08400h ; загрузка в указ. данных адреса свет. индикаторов
      mov A,#0h        ; загрузка в аккумулятор 0-й для гашения ДСИ3-ДСИ10
      movx @DPTR,A     ; запись байта по адресу мндикаторов
      sjmp nachalo     ; переход в начало программы
vpravo:  setb P3.4        ; включение ДСИ2
      mov DPTR,#08400h ; загрузка в указ. данных адреса свет. индикаторов
      mov A,#01010101b ; будут включены ДСИ4, ДСИ6, ДСИ8 и ДСИ10
      movx @DPTR,A     ; запись байта по адресу мндикаторов
      sjmp nachalo     ; переход в начало программы
      end
```

Порядок выполнения работы

1. Ознакомиться с теоретическим введением по методическим указаниям к лабораторной работе №1
2. На языке ассемблера написать программу, реализующую задание (согласно варианту).
3. Отладить программу с помощью отладчика, входящего в систему ProView. Убедиться в работоспособности программы.

Задания для самостоятельной работы

1. Написать программу, которая при включении переключателя 4 включает светодиоды 3 и 7.
2. Написать программу, которая при выключении переключателя 3 включает светодиоды 1 и 5.
3. Написать программу, которая при включении переключателя 1 включает светодиоды 7-10.
4. Написать программу, которая при выключении переключателя 2 включает светодиоды через один.
5. Написать программу, которая при включении переключателя 1 включает светодиоды парами через один.
6. Написать программу, которая при выключении переключателя 10 включает светодиоды 3 и 4.
7. Написать программу, которая при выключении переключателя 5 включает светодиоды 1 и 2.
8. Написать программу, которая при выключении переключателя 1 включает светодиоды 3-10.
9. Написать программу, которая при выключении переключателя 8 включает светодиоды 3-6.
10. Написать программу, которая при выключении переключателя 9 включает светодиоды 1 и 10.

Лабораторная работа №2
«Изучение команд условного перехода, реализация комбинационных схем на МК51»

Цель работы: изучение битовых команд МК51, команд условного перехода, реализация комбинационной схемы с использованием дискретных входов и выходов.

Приборы и принадлежности: стенд для изучения микропроцессоров МК51, интегрированная среда программирования ProView, ПК.

Теоретическое введение

ОМЭВМ семейства МК51 обладают широкими возможностями и гибкой системой команд. Это 8-разрядные микропроцессоры, оперирующие однобайтными операндами. Все регистры и память кроме регистра указателя данных DPTR[1] и счётчика команд PC у них имеют 8 разрядов. Кроме оперирования байтами МК51 способны работать с битовыми переменными и имеют для этого ряд команд, приведённых в таблице №1. В ней применены следующие обозначения:

C – флаг переноса;
bit – 128 программно доступных бита, любой бит управления, состояния или ввода/вывода;
/bit – то же, что и bit, но с инверсией;
rel8 – байт относительного смещения (условный переход осуществляется в диапазоне от -128 до +127 байтов относительно адреса байта следующей команды); в программе на ассемблере – метка строки перехода.

С отдельными битами и байтами, как группами бит, можно производить ряд логических операций, которые приведены ниже.

Основные логические операции

Логическое сложение (дизъюнкция) (обозначается «+» или «∨»):
0 + 0 = 0; 1 + 0 = 1; 0 + 1 = 1; 1 + 1 = 1.
Логическое умножение (конъюнкция) (обозначается «·» или «∧»):
0 · 0 = 0; 1 · 0 = 0; 0 · 1 = 0; 1 · 1 = 1.
Логическое отрицание (инверсия) (обозначается чертой над логической переменной или выражением):
$\overline{0} = 1$; $\overline{1} = 0$.

[1] Регистр DPTR можно рассматривать как два 8-разрядных регистра DPH и DPL и обращаться к ним по отдельности.

Основные законы алгебры логики

$a+0=a$, $a+1=1$, $a+a+a+...=a$, $a+\bar{a}=1$,
$a\cdot 0=0$, $a\cdot 1=a$, $a\cdot a\cdot a\cdot ...=a$, $a\cdot\bar{a}=0$, $\bar{\bar{a}}=a$.
Переместительный закон: $a+b=b+a$, $a\cdot b=b\cdot a$.
Сочетательный закон: $(a+b)+c=a+(b+c)$, $(a\cdot b)\cdot c=a\cdot(b\cdot c)$.
Распределительный закон: $a\cdot(b+c)=ab+ac$, $a+bc=(a+b)(a+c)$.
Закон поглощения: $a+ab=a$, $a(a+b)=a$.
Закон склеивания: $ab+a\bar{b}=a$, $(a+b)(a+\bar{b})=a$.
Закон отрицания: $\overline{a+b}=\bar{a}\cdot\bar{b}$, $\overline{ab}=\bar{a}+\bar{b}$.

Таблица 1 - Битовые команды МК51

Мнемоническое обозначение	Описание команды	Число байтов	Число циклов
SETB C	Установка флага переноса	1	1
SETB bit	Установка бита	2	1
CLR C	Сброс флага переноса	1	1
CLR bit	Сброс бита	2	1
CPL C	Инверсия бита	1	1
MOV C,bit	Пересылка бита в флаг переноса	2	1
MOV bit,C	Пересылка флага переноса в бит	2	1
ANL C,bit	«Логическое И» бита и флага переноса	2	2
ANL C,/bit	«Логическое И» инвертированного бита и флага переноса	2	2
ORL C,bit	«Логическое ИЛИ» бита и флага переноса	2	2
ORL C,/bit	«Логическое ИЛИ» инвертированного бита и флага переноса	2	2
JC rel8	Переход, если флаг переноса установлен	2	2
JNC rel8	Переход, если флаг переноса сброшен	2	2
JB bit,rel8	Переход, если бит установлен	3	2
JNB bit,rel8	Переход, если бит сброшен	3	2
JBC bit,rel8	Переход, если бит установлен и сброс этого бита	3	2

Синтез комбинационных цепей

Комбинационные логические цепи – цепи, выходные сигналы которых не зависят от предыстории, выходное значение которых однозначно определяются текущими сигналами на входе.

*Пример***.** Рассмотрим для примера систему управления автономным отопителем на жидком топливе. Свеча накала (воспламеняющая топливо) должна быть включена если выполнены следующие условия – подаётся топливо (логическая переменная X_1), подаётся воздух (логическая переменная X_2), топливо горит (X_3). Если обозначить логическую переменную «включена свеча накала» как Y, то её можно выразить через указанные условия так: $Y = X_1 \cdot X_2 \cdot \overline{X}_3$. Причем важно текущее состояние условий, а их предыдущее состояние не оказывает влияния на Y.

Синтез комбинационных цепей, т. е. составление в виде формулы зависимости выходной величины от заданных состояний входов, обычно производится в следующем порядке.

1. Составляется таблица истинности, т. е. таблица, которая содержит все возможные комбинации входных сигналов и заданные выходные сигналы.
2. Составляется т. н. дизъюнктивная нормальная форма (ДНФ), которая равна сумме конъюнкций тех строк таблицы истинности, для которых значение выходной функции равно “1”. Конъюнкция для строки равна логическому произведению её элементов, причем если значение на входе равно «1», то соответствующая логическая переменная берётся без инверсии, а если «0», то с инверсией.
3. Полученное выражение (ДНФ) упрощается (минимизируется) используя законы логики и приводится к виду, удобному для реализации на имеющейся элементной базе.

Пример.

Составить цепь с приведённой ниже таблицей истинности

№	X_1	X_2	X_3	Y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

Значение выходной функции равно единице для строк №3, 5, 6, 7. Конъюнкция для строки №3 равна $\overline{X}_1 X_2 X_3$, так как X_1 для строки №3 равна «0», а для X_2 и X_3 равна «1». Аналогично составляются логические произведения (конъюнкты) для строк №5, 6, 7. Затем составляется ДНФ, т. е. логическая сумма конъюнктов, которая и равна выходной функции:

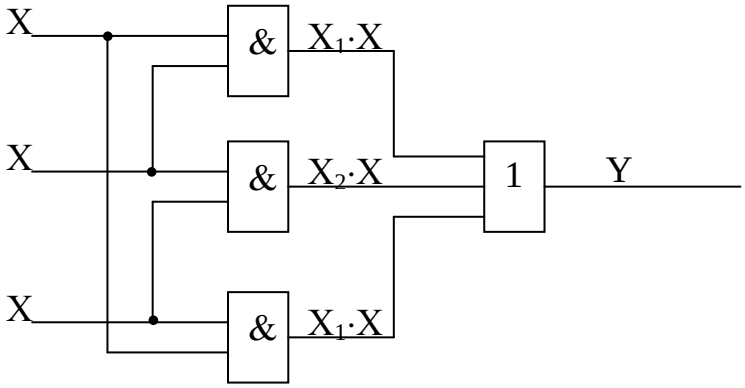
$$Y = \overline{X}_1 X_2 X_3 + X_1 \overline{X}_2 X_3 + X_1 X_2 \overline{X}_3 + X_1 X_2 X_3 .$$

Это логическое выражение можно реализовать и непосредственно с помощью логических элементов, но это будет нерационально. Поэтому упростим полученное выражение:

$$Y = \overline{X}_1 X_2 X_3 + X_1 X_2 X_3 + X_1 \overline{X}_2 X_3 + X_1 X_2 X_3 + X_1 X_2 \overline{X}_3 + X_1 X_2 X_3 =$$
$$= \left(\overline{X}_1 + X_1\right) X_2 X_3 + \left(\overline{X}_2 + X_2\right) X_1 X_3 + \left(\overline{X}_3 + X_3\right) X_1 X_2 = X_2 X_3 + X_1 X_3 + X_1 X_2$$

Здесь добавлено два слагаемых равных четвёртому, а затем, после вынесения за скобки, учтено правило $a + \overline{a} = 1$.

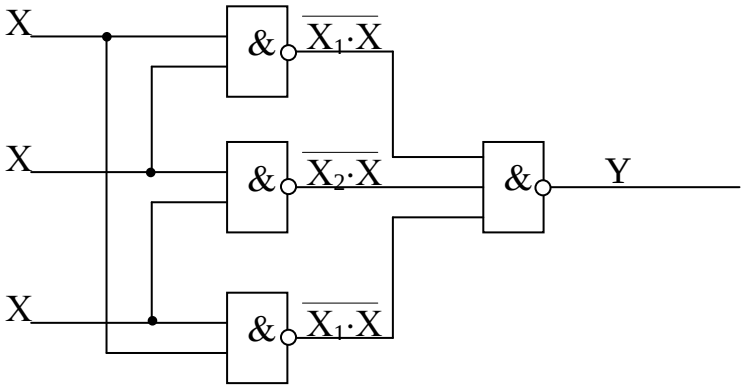
Итак, заданную логическую функцию можно реализовать с помощью трёх умножений (логических элементов, реализующих функцию «И») и одного логического сложения (логического элемента, реализующего функцию «ИЛИ»). Схема приведена ниже.



Эта же схема может быть реализована и другим способом. Преобразуем логическую функцию:

$$Y = X_1 X_2 + X_2 X_3 + X_1 X_3 = \overline{\overline{X_1 X_2} \cdot \overline{X_2 X_3} \cdot \overline{X_1 X_3}}.$$

Теперь для реализации данной логической функции надо использовать элементы «И-НЕ»:



Пример выполнения лабораторной работы

Требуется изготовить комбинационную цепь в соответствии с таблицей истинности, приведённой в теоретическом введении. Это, безусловно, можно сделать с помощью дискретных логических элементов, но в этом случае схему будет сложно изменять, модернизировать и т. д. Другой возможный вариант – моделирование схемы с использованием микропроцессора. Конечно, для моделирования простейших схем использование МП нецелесообразно, но при усложнении комбинационных цепей применение МП даёт большие преимущества.

Итак в в теоретическом введении была приведена таблица истинности и приведена схема на цифровых ИС, реализующая её. По таблице истинности составлена формула, реализующая эту таблицу.

Приведём программу для ОМЭВМ семейства МК51, работающую в соответствии с заданной таблицей истинности:

```
X1    bit 00h          ; присваиваем биту 00h обозначение X1
X2    bit 01h          ; присваиваем биту 01h обозначение X2
X3    bit 02h          ; присваиваем биту 02h обозначение X3

start: mov DPTR,#08B00h    ; загрузка в указатель данных адреса
                           ;управляющего регистра КР580ВВ55
      mov A,#10011011b     ; управляющее слово для КР580ВВ55
                           ;(PA - режим 0; PA - ввод; PCH - ввод
                           ; PB - режим 0; PB - ввод; PCL - ввод
      movx @DPTR,A         ; загрузка управляющего слова в КР580ВВ55
qwe:                       ; qwe - цикл опроса датчико и вывода их состояния
      mov DPTR,#8300h      ; загрузка в указатель данных адреса порта B
                           ; КР580ВВ55, к которому подключены дискретные входы
      movx A,@DPTR         ; чтение дискретных входов

                           ;------------------моделирование логической схемы
      mov 020h,A           ; запись состояния дискретных индикаторов в
                           ; битовые переменные
      mov C,X1
      anl C,X2             ; в бите C теперь находится логическое произведение X1 и X2
      mov 03h,C            ; загрузка логического произведения X1 и X2 в бит 03h
      mov C,X2
      anl C,X3             ; в бите C теперь находится логическое произведение X2 и X3
      mov 04h,C            ; загрузка логического произведения X3 и X2 в бит 04h
      mov C,X1
      anl C,X3             ; в бите C теперь находится логическое произведение X1 и X3
      orl C,04h            ; в C теперь логическая сумма X1*X3+X2*X3
      orl C,03h            ; в C теперь логическая сумма X1*X3+X2*X3+X1*X2
      clr A                ; очистка аккумулятора
      rlc A                ; сдвиг справа налево содержимого аккумулятора через бит C

      mov DPTR,#08400h     ; загрузка в указатель данных адреса
                           ; регистра дискретных индикаторов
      movx @DPTR,A         ; вывод на индикаторы содержимого аккумулятора
      sjmp qwe             ; зацикливание
      end                  ; конец программы
```

Порядок выполнения работы

1. На языке ассемблера написать программу, реализующую заданную комбинационную схему. В качестве входных сигналов использовать переключатели ДД7-ДД10, имитирующие дискретные входы. Выходной сигнал подать на светодиодный индикатор ДСИ10.
2. Отладить программу с помощью отладчика, входящего в систему ProView.
3. Сгенерировать HEX-файл, загрузить его в лабораторный стенд с помощью загрузчика «Загрузчик СУ-МК». Убедиться в работоспособности программы.

Задания для самостоятельной работы

1 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

2 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

3 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

4 Вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	1

5вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	1

6вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

7 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

8 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	1

9 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	0

10 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

11 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

12 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

13 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

14 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	1

15 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	0

16 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

17 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

18 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

19 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

20 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	1

21 вариант

№	X_1	X_2	X_3	X_4	Y
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

Лабораторная работа №3

«Работа с параллельными портами ввода-вывода»

Цель работы: изучение параллельного ввода-вывода МП-систем, изучение динамической индикации.

Приборы и принадлежности: стенд для изучения микропроцессоров МК51, интегрированная среда программирования ProView, ПК.

Теоретическое введение

Для ввода-вывода информации в микропроцессорах серии МК51 предусмотрено 4 восьмиразрядных порта. Обращаться к ним можно как к байтам, так и к отдельным битам. При этом в ассемблере используется обозначение P0, P1, P2 и P3 для обращения к портам, и P0.0, P0.1 … P3.6, P3.7 - при обращении к отдельным битам. Так, например, в МП 8051 линии P3.1 (передача данных - TXD) и P3.0 (чтение данных - RXD) используются для организации последовательного интерфейса.

Однако, далеко не всегда достаточно 4×8 линий ввода-вывода. Так при большой программе, не умещающейся в ПЗУ МП, приходится использовать внешнюю память программ. Также возможно использование внешней памяти данных. В обоих этих случаях уже практически невозможно использовать порты P0 и P2. Возможны и другие случаи недостаточного количества линий ввода-вывода. В этих случаях используют внешние ИС регистров (например К555ИР23) или расширителей ввода-вывода (например К580ВВ55).

В учебном стенде СУ-МК-1 используются все возможные способы организации обмена данными.

Ввод-вывод через линии порта

Линии порта P3.2 и P3.3 непосредственно подключены к дискретным переключателям SW1и SW2, с помощью которых эти линии подключаются к 0 В или 5 В. Подключение осуществляется через резисторы, чтобы не вывести из строя МП при случайном выводе данных через этим линии.

Линии P3.5 и P3.4 подключены к ключам на биполярных транзисторах, коммутирующих светодиоды VD1 и VD2. Для зажигания светодиода надо в эти линии вывести «1».

Приведём программу, которая включает или выключает светодиод VD1 соответственно положению переключателя SW1 (битовые команды МК51 предусматривают обмен битами только через флаг C):

```
asd:  mov C,P3.2    ; считывание состояния бита P3.2 и
                    ;  сохранение его в флаге переноса
      mov P3.5,C    ; запись бита C в бит P3.5
      sjmp asd      ; зацикливание программы
      end
```

Адресация внешних устройств в СУ-МК

При обращении к внешним устройствам и к памяти в семействе МК-51 предусмотрен один и тот же протокол. То есть обращение к ячейкам памяти осуществляется так же как и к внешним устройствам ввода-вывода (гарвардская архитектура). Адрес передаётся через порты P0 и P2, причём младший байт адреса и данные передаются через P0 по очереди. Сначала передаётся младший байт адреса. Он фиксируется во внешнем регистре по сигналу ALE, а затем передаётся байт данных (см. рисунок 1 и 2). Старший байт адреса всё это время присутствует на линиях P2.

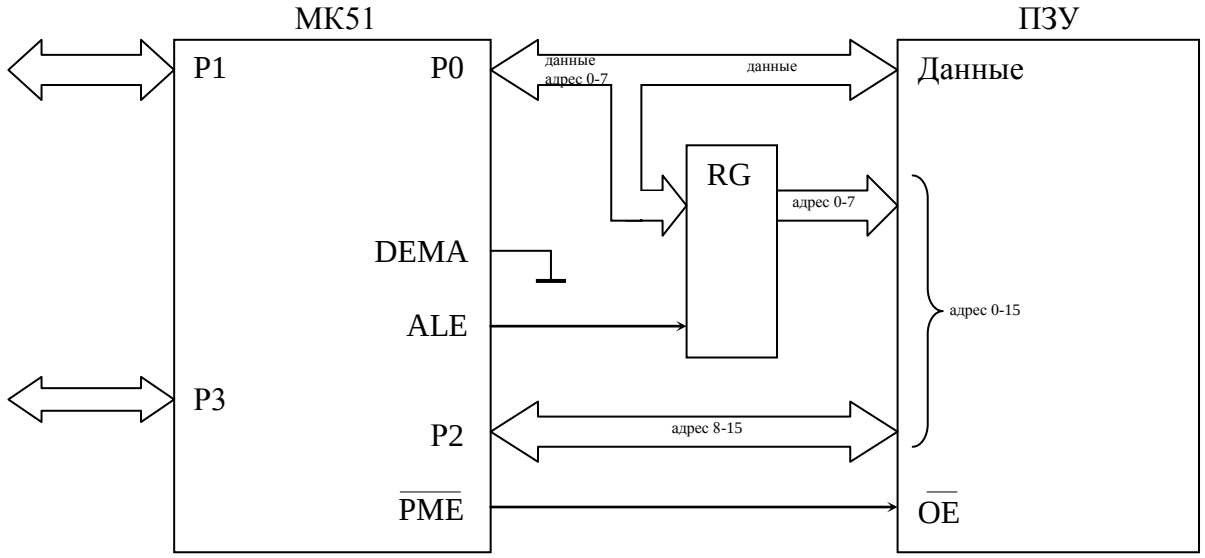


Рисунок 1 - Работа с внешним ПЗУ

Отличие при работе с внешней памятью программ и данных состоит в стробирующих сигналах. Когда идёт обращение к памяти программ (всегда чтение), то активизируется (⎍) сигнал PME – program memory enabled. А когда происходит обращение к памяти данных используется сигнал чтения RD или сигнал записи WR (они же выходы порта P3 – P3.7 и P3.6 соответственно).

В любом случае порты P0 и P2 используются для выдачи адреса. Для распознавания своего адреса необходим дешифратор адреса. В ИС памяти уже встроен дешифратор, который по заданному адресу записывает или считывает из нужной ячейки памяти. Для ИС в большинстве случаев ввода-вывода требуется отдельный дешифратор. В СУ-МК для этого используется ИС К555ИД7 (DD11). На её входы поступает сигнал с адресных выходов A9, A10. Линия A15 подключена к входу выборки ИС, поэтому дешифрация адреса осуществляется только при A15=1.

При A15=1 и A9=0, A10=0 активизируется (⎍) выход «0» и этот сигнал запишет данные в регистр К555ИР22 (DD7), выходы которого подключены к семисегментному индикатору. Соответствующий адрес – 1000000000000000_b = 8000_h.

Аналогично при 15=1 и A9=0, A10=1 активизируется (⎍) выход «1» и этот сигнал запишет данные в регистр К555ИР22 (DD9), выходы которого подключены к дискретным индикаторам. Соответствующий адрес – $1000010000000000_b = 8400_h$.

Рассуждая таким образом можно определить все адреса устройств ввода вывода. Следует заметить, что значение имеют только линии A15, A10, A9, все стальные не дешифрируются и могут принимать произвольные значения, поэтому обращение по адресам 8000_h, 8020_h и 8001_h приведёт к одинаковому результату. Хотя при обращении к К580ВВ55 для дешифрации используются также линии адреса A8 и A11. Ниже, в таблице, приведены адреса устройств ввода-вывода учебного стенда СУ-МК.

Таблица 1 - Адресное пространство стенда СУ-МК

Устройство	Адрес
Регистр управления К580ВВ55	$8B00_h$
Порт А К580ВВ55 (символьный индикатор)	8200_h
Порт В К580ВВ55 (имитатор дискретных сигналов)	8300_h
Порт С К580ВВ55 (клавиатура)	$8A00_h$
Регистр семисегментного индикатора	8000_h
Регистр дискретных индикаторов	8400_h

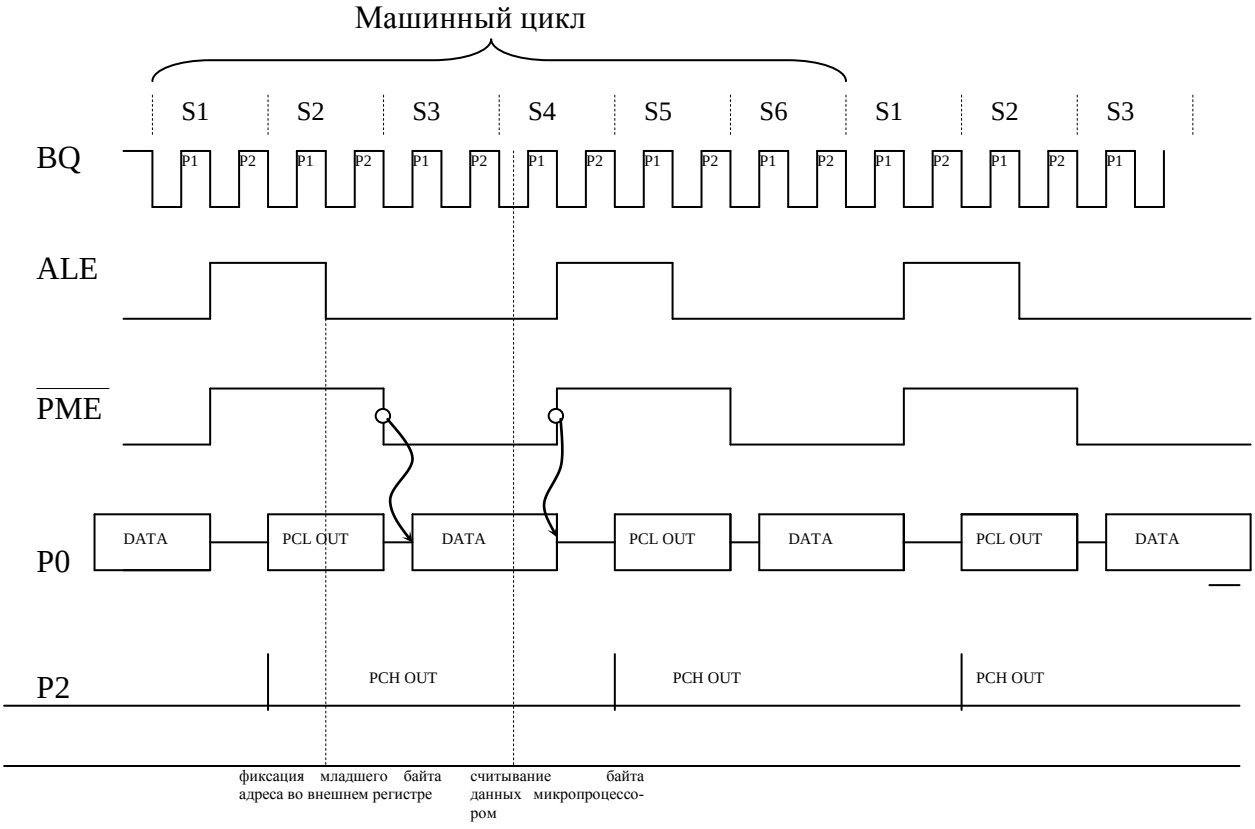


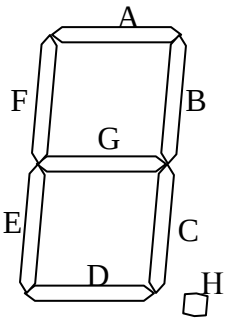
Рисунок 2 - Работа с внешней памятью программ

Цикл обмена с внешними устройствами аналогичен циклу показанному на рисунке 2. Только вместо сигнала PME используются либо WR, либо RD. Для адресации используется 16 разрядов адреса. Адрес внешнего устройства должен быть занесён в регистр указателя данных DPTR, а затем с помощью команды movx в ячейку с указанным адресом надо записать или считать нужные данные. Приведём пример:

```
…
mov DPTR,#8400h  ; занесение адреса дискретных индикаторов в регистр DPTR
mov A,#10101010b ;  загрузка в аккумулятор данных для вывода
movx @DPTR,A     ; загрузка данных по адресу, указываемому DPTR
```

После выполнения этого фрагмента программы будут включены светодиоды дискретного индикатора через один.

Семисегментный индикатор



Для вывода числовой информации в МП системах часто применяются семисегментные светодиодные индикаторы. Они представляют собой восемь светодиодов такой формы, чтобы обеспечить отображение семи элементов цифры и точки. Светодиоды обозначаются как A, B, C, D, E, F, G, H. Для отображения нужной цифры включаются соответствующие светодиоды, например для отображения 2 надо включить светодиоды A, B, G, E, D.

Для управления семисегментными индикаторами есть микросхемы дешифраторов (например К514ИД2), на вход которых поступает двоичный код (1-2-4-8), а к семи выходам – соответствующие светодиоды.

Можно обойтись и без ИС дешифраторов, подключив светодиоды непосредственно к восьми линиям порта ввода-вывода. Именно так сделано в стенде СУ-МК. Это позволяет выводить любые возможные комбинации включённых светодиодов, а не только цифры. Нагрузочная способность линий вывода ИС 89С51, использованной в стенде, не позволяет непосредственно подключить светодиоды индикатора к портам. Поэтому в нём применён регистр DD3 К1533ИР22 для хранения выведенного кода символа. Эта ИС также выполняет роль буферного элемента, увеличивающего нагрузочную способность. Для обращения к этому регистру зарезервирован адрес 08000h. Чтобы включить соответствующий сегмент надо в нужный разряд записать «0». Например, фрагмент программы для вывода числа «4» будет выглядеть так:

сегмент	H	G	F	E	D	C	B	A
разряд	7	6	5	4	3	2	1	0

```
mov DPTR,#8000h   ; адрес семисегментного индикатора
mov A,#10011001b  ; код цифры 4
movx @DPTR,A      ; запись кода цифры по адресу,
                  ; указанному в DPTR
```

или

```
mov DPTR,#8000h  ; адрес семисегментного индикатора
mov A,#099h      ; код цифры 4
movx @DPTR,A     ; запись кода цифры по адресу,
                 ; указанному в DPTR
```

Динамическая индикация

Понятно, что при выводе нескольких цифр надо очень много линий ввода-вывода. Для уменьшения их количества применяют так называемую динамическую индикацию. При этом сначала кратковременно в первом разряде отображается первая цифра, затем во втором вторая и т. д. Потом процесс повторяется. При большой частоте переключения создаётся впечатление, что все индикаторы светятся одновременно и без мерцания.

В СУ-МК для включения одного из восьми семисегментных индикаторов используются линии порта P1: P1.0, P1.1, P1.2. При выводе на них, например, числа 010 будет включён третий справа индикатор (в соответствии с нумерацией 7, 6, 5, 4, 3, 2, 1, 0).

Для примера приведём программу, выводящую на индикатор число 2008. После отображения каждой цифры с помощью цикла сделана небольшая задержка чтобы дать время светодиодам отобразить свою цифру.

```
nachalo:  mov P1,#07h        ; вывод будет осуществлён в крайний правый разряд индикатора
          mov DPTR,#8000h   ; адрес семисегментного индикатора
          mov A,#10100100b  ; код цифры 2
          movx @DPTR,A      ; запись кода цифры по адресу, указанному в DPTR

          mov  R0,#0FFh     ; эти три строки обеспечивают задержку для
a1:         dec R0          ; отображения цифры некоторое время
          cjne R0,#0h,a1

          mov P1,#06h       ; вывод будет осуществлён во второй разряд индикатора
          mov DPTR,#8000h   ; адрес семисегментного индикатора
          mov A,#11000000b  ; код цифры 0
          movx @DPTR,A      ; запись кода цифры по адресу, указанному в DPTR

          mov  R0,#0FFh
a2:         dec R0
          cjne R0,#0h,a2

          mov P1,#05h       ; вывод будет осуществлён в третий разряд индикатора
          mov DPTR,#8000h   ; адрес семисегментного индикатора
          mov A,#11000000b  ; код цифры 0
          movx @DPTR,A      ; запись кода цифры по адресу, указанному в DPTR

          mov  R0,#0FFh
a3:         dec R0
```

```
        cjne R0,#0h,a3

        mov P1,#04h       ; вывод будет осуществлён в четвёртый разряд индикатора
        mov DPTR,#8000h   ; адрес семисегментного индикатора
        mov A,#10000000b  ; код цифры 8
        movx @DPTR,A      ; запись кода цифры по адресу, указанному в DPTR

        mov  R0,#0FFh
a4:      dec R0
        cjne R0,#0h,a4

        ajmp nachalo
        end
```

Порядок выполнения работы

1. На языке ассемблера написать программу, реализующую задание.
2. Написать на языке ассемблера программу. Отладить её с помощью отладчика, входящего в систему ProView.
3. Сгенерировать HEX-файл, загрузить его в лабораторный стенд с помощью загрузчика «Загрузчик СУ-МК». Убедиться в работоспособности программы.

Задания для самостоятельной работы

1. Вывести на семисегментный индикатор (в десятичном коде) число включённых выключателей.

2. Отобразить на семисегментном индикаторе состояние дискретных входов – если переключатель находится во включённом положении , то в соответствующем разряде отображается «1», если выключен, то «0».

3. В левом разряде семисегментного индикатора сначала отобразить «0». После каждого переключения переключателя ДД3 увеличивать показания индикатора от 0 до 9.

4. В левом разряде семисегментного индикатора сначала отобразить «9». После каждого переключения переключателя ДД3 уменьшать показания индикатора от 9 до 0.

5. Если переключатель ДД3 включён, то отобразить On, в противном случае - «OFF».

6. Найти поразрядную логическую сумму двух четырёхразрядных чисел, образованных считанным байтом с дискретных датчиков (ДД3-ДД6 и ДД7-ДД10). Отобразить это в четырёх разрядах семисегментного индикатора.

7. Найти поразрядное логическое произведение двух четырёхразрядных чисел, образованных считанным байтом с дискретных датчиков (ДД3-ДД6 и ДД7-ДД10). Отобразить это в четырёх разрядах семисегментного индикатора.

8. Найти поразрядное логическое «исключающее или» двух четырёхразрядных чисел, образованных считанным байтом с дискретных датчиков (ДД3-ДД6 и ДД7-ДД10). Отобразить это в четырёх разрядах семисегментного индикатора.

9. Включить сегмент «d» в левом разряде семисегментного индикатора. При включении-выключении ДД3 включённый сегмент должен перемещаться вправо.

10. Включить сегмент «a» в правом разряде семисегментного индикатора. При включении-выключении ДД3 включённый сегмент должен перемещаться влево.

11. Вывести на семисегментный индикатор десятичное двухразрядное число, заданное в двоичном коде на дискретных входах.

12. Вывести на семисегментный индикатор число, равное сумме включённых дискретных входов.

13. Вывести на семисегментный индикатор числа 1, 2, … 8.

14. Вывести на семисегментный индикатор (в десятичном коде) сумму двух чисел в двоичном коде, заданных на дискретных входах.

15. Вывести на семисегментный индикатор (в десятичном коде) произведение двух чисел в двоичном коде, заданных на дискретных входах.

16. Вывести на семисегментный индикатор (в десятичном коде) частное двух чисел в двоичном коде, заданных на дискретных входах.

17. Вывести на семисегментный индикатор (в шестнадцатиричном коде) двухразрядное число, первый разряд которого задаётся переключателями ДД3-ДД6, а второй – ДД7-ДД10.

18. Вывести на семисегментный индикатор (в десятичном коде) число, соответствующее номеру самого левого включённого выключателя.

19. Включить на семисегментном индикаторе сегменты A, B, C… при включении переключателя 1, 2, 3…

20. Вывести на семисегментный индикатор слово ГАННОВЕР.

Лабораторная работа №4

«Работа с подпрограммами. Клавиатура»

Цель работы: получение навыков написания подпрограмм применительно к организации опроса клавиатуры.

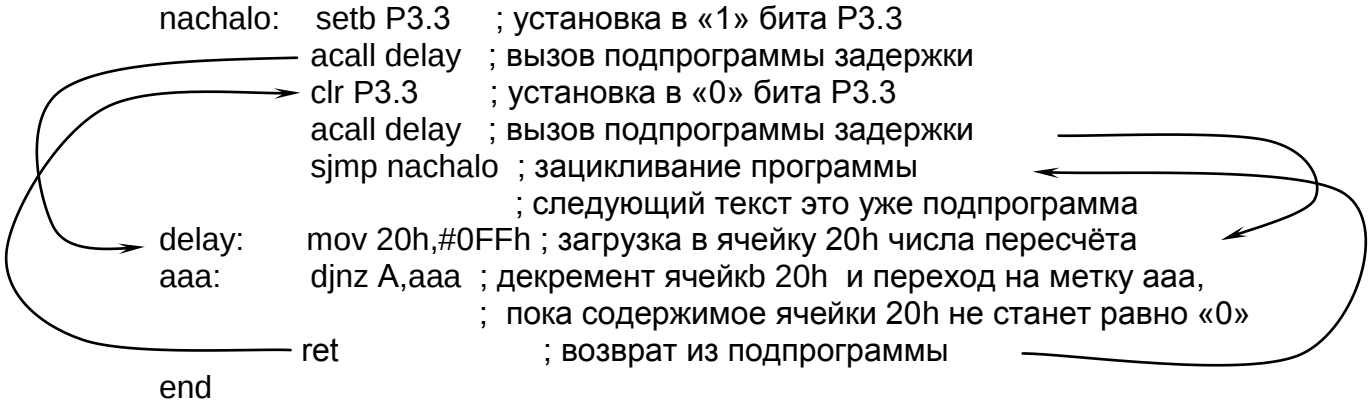
Приборы и принадлежности: стенд для изучения микропроцессоров МК51, интегрированная среда программирования ProView, ПК.

Теоретическое введение

При частом выполнении одних и тех же действий в программе текст её получается громоздким. Чтобы этого не происходило, прибегают к выделению части часто повторяемого кода в отдельные подпрограммы.

Подпрограммы представляют собой небольшие программы, вызываемые из основной программы по мере необходимости и выполняющие некоторую специализированную функцию. Хорошим примером является организация задержки. Допустим при работе в некоторых местах программы надо делать задержку в 100 мкс. Можно во всех местах программы, где это необходимо вставить циклы, реализующих задержку. Но удобнее оформить подпрограмму задержки и вызывать её по мере необходимости.

Вызов подпрограммы осуществляется командой *acall*, после которой следует метка, именующая подпрограмму. После выполнения этой команды программа продолжает своё выполнение с указанной метки. Для возврата из подпрограммы используется команда *ret*. После этой команды программа возвращается к тому месту, от которого она начала выполнение подпрограммы. Приведём пример:



Эта программа попеременно выводит на выход P3.3 то «1», то «0», причём время единичного и нулевого состояний определяется задержкой, которая определяется временем выполнения команд acall (1 раз), mov (1 раз), djnz (255 раз), ret (1 раз). Команда djnz выполняется 2 машинных цикла, это около 2 мкс (при тактовой частоте 12 МГц). Поэтому задержка будет около 310 мкс + 5мкс на остальные команды. В результате на выводе P3.3 будет выводиться прямоугольный сигнал с периодом 730 мкс.

Работа со стеком.

Чтобы подпрограмма не изменяла содержимое регистров надо все переменные (регистры ячейки памяти) сохранять на время выполнения подпрограммы. Обычно это делается с помощью стека.

Для того чтобы сохранить в стеке содержимое регистра или ячейки памяти надо выполнить команду *push reg*, где *reg* – прямо адресуемый регистр или ячейка памяти. Регистры R_i нельзя сохранять в стеке, их можно только сохранить по их абсолютным адресам (например R0 первого банка соответствует ячейке 00h). Возврат чисел из стека осуществляется командой *pop reg*, при этом последнее сохранённое число считывается из стека и помещается в *reg*. Так в предыдущем примере подпрограмму задержки лучше дополнить так:

```
delay:    push 020h    ; сохранение содержимого ячейки 020 в стеке
          mov 20h,#0FFh ; загрузка в ячейку 20h числа пересчёта
aaa:      djnz A,aaa  ; декремент ячейкb 20h  и переход на метку aaa,
                      ;  пока содержимое ячейки 20h не станет равно «0»
          pop 020h    ; возврат содержимого ячейки 020 в стеке
          ret               ; возврат из подпрограммы
end
```

Теперь эту подпрограмму можно использовать и в других программах, не опасаясь, что из-за совместного использования ячейки 020 возникнут ошибки.

Следует знать, что стек начинается с ячейки 08h (по умолчанию) и растёт в сторону увеличения адресов. Адрес ячейки, в которую была произведена последняя запись, хранится в указателе стека SP. В начале SP=07h. После записи в стек содержимое SP увеличивается на единицу и в указываемую SP ячейку производится запись. При считывании из стека содержимое ячейки, указываемое SP помещается в указываемый командой *pop* регистр и содержимое SP уменьшится на единицу. Приведём пример (запись (08h)=00 означает, что ячейка с адресом 08h содержит 00):

```
                ; исходное состояние – SP=07h, A=24h, B=8Fh,
                ;                        (08h)=00, (09h)=00
 push ACC      ; SP=08h (увеличилось на 1), A=24h (не изменилось)
                ; (08h)=24h,  (09h)=00
push B          ; SP=09h (увеличилось на 1), A=24h (не изменилось)
                ; B=8Fh (не изменилось),  (08h)=24h,  (09h)=8Fh
                ; теперь содержимое A и B может произвольно меняться
                 ; например A=00h, B=00h,
        …
pop B          ; SP=08h (уменьшилось на 1), A=00h (не изменилось)
                ; B=8Fh (возвратил из стека),  (08h)=24h,  (09h)=8Fh
pop ACC        ; SP=07h (уменьшилось на 1), B=8Fh (не изменилось)
                ; A=24h (возвратил из стека),  (08h)=24h,  (09h)=8Fh
```

Заметьте, что вынимать из стека надо сначала то, что положили в него последним.

Приведём пример программы, в которой последовательно вводятся два трёхразрядных числа, которые затем складываются, и результат выводится на семисегментный индикатор. Клавиша «*» используется для отмены ввода числа, а «#» для подтверждения.

```
        ; программа работы с клавиатурой

        ; по адресам 30, 31, 32 хранится первое введённое число
        ; по адресам 40, 41, 42 хранится первое введённое число
        ; по адресам 50, 51, 52 хранится их сумма

          acall init     ; вызов подпрограммы инициализации

          setb P3.5      ; включить первый светодиод
          clr P3.4       ; выключить второй светодиод
          mov R0,#030h   ; загрузка начального адреса для ввода числа
          acall vvodS    ; ввод первого числа по адресу, указанному R0

          setb P3.4      ; включить второй светодиод
          clr P3.5       ; выключить первый светодиод
          mov R0,#040h   ; загрузка начального адреса для ввода второго числа
          acall vvodS    ; ввод второго  числа по адресу, указанному R0

          acall summa       ; вызов подпрограммы суммирования
                            ;двух трёхразрядных чисел
          mov R0,#050h   ; загрузка начального адреса суммы для индикации

start:    acall indic       ; зацикливание индикации суммы, если перенести
                            ; метку  start в начало (на 9 комманд вверх),
                            ; то начнётся ввод новых слагаемых
          sjmp start

;---------------------- далее идут подпрограммы ----------------------------------------------------
summa:       ; подпрограмма сложения двух трёхразряддных чисел,
             ; адреса заданы в подпрограмме
          mov A,032h     ; младший байт первого слагаемого
          add A,042h     ; прибавляем младший байт второго слагаемого
          da A           ; десятичная коррекция суммы
          mov C,ACC.4
          anl A,#0Fh
          mov 052h,A     ; результат помещаем в ячейку 052h

          mov A,031h
          addc A,041h
          da A
          mov C,ACC.4
          anl A,#0Fh
          mov 051h,A

          mov A,030h
          add A,040h
```

```
        da A
        mov C,ACC.4
        anl A,#0Fh
        mov 050h,A

        ret
;-------------------------------------------------------------------------------------------
init:    ; подпрограмма инициализации
        push ACC        ; сохранение в стеке аккумулятора
        mov DPTR,#08B00h   ;адрес управляющего слова вв55
        mov A,#10010011b    ;загрузка в вв55 управляющего слова (A,B,PCL - ввод,
PCH - вывод, режим 0)
        movx @DPTR,A
        mov R0,#030h      ; очистка памяти начиная с 30 по 52 ячейку
aaa:      mov @R0,#00h
        inc R0
        cjne R0,#052h,aaa
        pop ACC
        ret             ; возврат из подпрограммы
;-------------------------------------------------------------------------------------------
opros:    ; подпрограмма опроса клавиатуры, опрошенный код содержится в
          ; аккумуляторе звёздочке соответствует 0A, а решётке - OB,
          ; если клавиша не нажата, то возвращается FF
 bbb1:  mov A,#00010000b   ; сканируется первая линия
        mov DPTR,#08A00h   ; адрес порта C вв55, к которому подключена клава
        movx @DPTR,A      ; сканирование первой линии
        movx A,@DPTR      ; считываение линии
        anl A,#07h        ; сброс старших разрядов
        cjne A,#01h,bbb2   ; если нажата клавиша "1", то будет считано ACC=01
        mov A,#01h      ; загрузка в аккумулятор числа, соответствующего "1"
        sjmp vihod      ; выход из подпрограммы
 bbb2:  cjne A,#02h,bbb3   ; если нажата клавиша "2", то будет считано ACC=02
        mov A,#02h
        sjmp vihod
 bbb3: cjne A,#04h,bbb4   ; если нажата клавиша "3", то будет считано ACC=04
        mov A,#03h
        sjmp vihod

 bbb4:   mov A,#00100000b   ; сканируется вторая линия
         movx @DPTR,A      ; опрос второй линии
         movx A,@DPTR       ; считываение линии
         anl A,#07h        ; сброс старших разрядов
         cjne A,#01h,bbb5   ; если нажата клавиша "4", то будет считано ACC=01
         mov A,#04h
         sjmp vihod
 bbb5:   cjne A,#02h,bbb6   ; если нажата клавиша "5", то будет считано ACC=02
         mov A,#05h
         sjmp vihod
 bbb6:   cjne A,#04h,bbb7   ; если нажата клавиша "6", то будет считано ACC=04
         mov A,#06h
         sjmp vihod
```

```
  bbb7:  mov A,#01000000b
         movx @DPTR,A       ; опрос третьей линии
         movx A,@DPTR       ; считываение линии
         anl A,#07h         ; сброс старших разрядов
         cjne A,#01h,bbb8   ; если нажата клавиша "7", то будет считано ACC=01
         mov A,#07h
         sjmp vihod
  bbb8:  cjne A,#02h,bbb9   ; если нажата клавиша "8", то будет считано ACC=02
         mov A,#08h
         sjmp vihod
  bbb9:  cjne A,#04h,bbb10  ; если нажата клавиша "6", то будет считано ACC=04
         mov A,#09h
         sjmp vihod

  bbb10: mov A,#10000000b
         movx @DPTR,A       ; опрос четвёртой линии
         movx A,@DPTR       ; считываение линии
         anl A,#07h         ; сброс старших разрядов
         cjne A,#01h,bbb11  ; если нажата клавиша "*", то будет считано ACC=01
         mov A,#0Ah         ; тогда возвращается 0A
         sjmp vihod
  bbb11: cjne A,#02h,bbb12 ; если нажата клавиша "0", то будет считано ACC=02
         mov A,#00h         ; тогда возвращается 00
         sjmp vihod
  bbb12: cjne A,#04h,bbb13 ; если нажата клавиша "#", то будет считано ACC=04
         mov A,#0Bh         ; тогда возвращается 0B
         sjmp vihod
  bbb13:  mov A,#0FFh       ; ести ничего не нажато решётка, то ACC = 0FF

vihod:   mov DPTR,#08400h
         movx @DPTR,A
         ret
;-------------------------------------------------------------------------------------
opros1:    ; подпрограмма несколько раз опрашивает клавиатуру и выдаёт
           ; результат только если 16 раз подряд будет считано один и тот же
           ; результат тем самым устраняет дребезг контактов
 zzz1:  acall opros       ; опрос клавиатуры
        mov 70h,A         ; сохранение считанного числа в промежуточной ячейке
        mov R6,#0Fh       ; регистр R6 служит счётчиком числа опросов
 zzz:   acall opros       ; цикл опрос клавиатуры
        cjne A,70h,zzz1 ; если не совпадает с первым значением, то опрос с начала
        djnz R6,zzz       ; цикл, пока R6 не станет равным нулю
        ret
;-------------------------------------------------------------------------------------
vvodS:   ; подпрограмма ввода трёхразрядного числа
      ; В R0 должен находиться начальный адрес ячеек для ввода
    ; R1 используется для хранения адреса текущей ячейки, в него вводится число
     mov 60h,R0        ; сохраняем содержимое R0 в ячейке 60h
     mov A,60h
     mov R1,60h
     add A,#03h ; прибавляем к адресу 3, чтобы знать сколько чисел вводить
     mov 61h,A  ; теперь в ячейке 61 находится адрес последней ячейки с данными
```

```
 ccc0:  acall opros1      ; опрос клавиатуры, результат опроса в ACC
        mov B,A     ; в регистре B будет храниться предыдущее считанное значение
 ccc1:  acall indic  ; вывод на индикацию числа, находящегося по адресу,
                     ; начальный адрес в R0
        acall opros1      ; опрос клавиатуры, результат опроса в ACC
        cjne A,#0FFh,ccc5 ; ести ничего не нажато, то продолжаем опрашивать
        mov B,#0FFh
        sjmp ccc1
 ccc5:  cjne A,B,ccc4     ; проверка, удерживается клавиша или нажата другая
      sjmp ccc1 ; если не изменилось состояние клавы, то снова индикация и опрос
 ccc4:    mov B,A            ; загрузка в регистр B кода нажатой клавиши
        cjne A,#0Ah,ccc2  ; ести нажата звёздочка, то начинаем ввод сначала
        mov R1,60h
        mov @R0,#00h      ; обнуление введённого трёхзначного числа
        inc R0
        mov @R0,#00h
        inc R0
        mov @R0,#00h
        mov R0,60h
        sjmp ccc0
 ccc2:    cjne A,#0Bh,ccc3 ; ести нажата решётка, то ввод закончен
          sjmp vihod1
 ccc3:    mov @R1,A        ; вводим число с клавиатуры в ячейку с адресом в R1
          inc R1           ; переходим к следующему разряду
          mov A,R1
          cjne A,61h,ccc1  ; если ещё не всё число введено, то продолжаем,
                           ;иначе конец подпрограммы
          mov R1,060h      ; ввод числа  сначала
          sjmp ccc1
vihod1:    ret
;-------------------------------------------------------------------------------------
indic:     ; подпрограмма индикации трёхразрядного числа,
        ; которое содержится по адресам начиная с указанного в R0
        push ACC  ; сохранение в стеке ACC, R0 и R7
        mov A,R0
        push ACC
        mov A,R7
        push ACC

        mov A,@R0             ; считываем первое число
        acall semiseg         ; преобразуем его в семисегментный код
        mov DPTR,#8000h   ; адрес семисегментного индикатора
        movx @DPTR,A      ; запись кода цифры по адресу, указанному в DPTR
        mov P1,#06h           ; вывод будет осуществлён в крайний правый разряд
                              ; индикатора

        mov  R7,#0FFh     ; эти три строки обеспечивают задержку для
a1:     dec R7                ; отображения цифры некоторое время
        cjne R7,#0h,a1

        inc R0                    ; переход к следующему разряду
```

```
        mov A,@R0           ; считываем второе число
        acall semiseg       ; преобразуем его в семисегментный код
        mov DPTR,#8000h     ; адрес семисегментного индикатора
        movx @DPTR,A        ; запись кода цифры по адресу, указанному в DPTR
        mov P1,#05h         ; вывод будет осуществлён во второй разряд индикатора

        mov  R7,#0FFh       ; эти три строки обеспечивают задержку для
a2:     dec R7              ; отображения цифры некоторое время
        cjne R7,#0h,a2

        inc R0              ; переход к следующему разряду
        mov A,@R0           ; считываем третье число
        acall semiseg       ; преобразуем его в семисегментный код
        mov DPTR,#8000h     ; адрес семисегментного индикатора
        movx @DPTR,A        ; запись кода цифры по адресу, указанному в DPTR
        mov P1,#04h         ; вывод будет осуществлён в третий разряд индикатора

        mov  R7,#0FFh       ; эти три строки обеспечивают задержку для
a3:     dec R7              ; отображения цифры некоторое время
        cjne R7,#0h,a3

        mov DPTR,#8000h     ; адрес семисегментного индикатора
        mov A,#0FFh         ; гашение индикатора
        movx @DPTR,A

        pop ACC
        mov R7,A
        pop ACC
        mov R0,A
        pop ACC
        ret
;-------------------------------------------------------------------------------
semiseg:   ; подпрограмма, переводит содержимое аккумулятора из численного
           ; выражения  в код для семисегментного индикатора
    push B
    cjne A,#0ah,rrr     ; проверка, не превышает ли число в аккумуляторе 10
rrr:  jnc endsemiseg    ; если превышает, то ничего не делаем
    mov B,#04           ; загрузка шага для перехода (4 цикла на вывод одной цифры)
    mul AB
    mov DPTR,#TABLE     ; загружается в DPTR начальный адрес переходов
    jmp @A+DPTR         ; в аккумулятор загружается адреса перехода как сумма
                        ; аккумулятора и начального адреса
TABLE: mov A,#0C0h
        ajmp endsemiseg
         mov A,#0F9h
        ajmp endsemiseg
        mov A,#0A4h
        ajmp endsemiseg
        mov A,#0B0h
        ajmp endsemiseg
        mov A,#099h
        ajmp endsemiseg
```

```
        mov A,#092h
        ajmp endsemiseg
        mov A,#082h
        ajmp endsemiseg
        mov A,#0F8h
        ajmp endsemiseg
        mov A,#080h
        ajmp endsemiseg
        mov A,#090h
endsemiseg:     pop B
        ret
;-------------------------------------------------------------------------------------
    end         ; конец программы.
```

Порядок выполнения работы

1. На языке ассемблера написать программу, реализующую задание.
2. Написать на языке ассемблера программу. Отладить её с помощью отладчика, входящего в систему ProView.
3. Сгенерировать HEX-файл, загрузить его в лабораторный стенд с помощью загрузчика «Загрузчик СУ-МК». Убедиться в работоспособности программы.

Задания для самостоятельной работы

1) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифр 7, 8, 9 включала бы светодиоды 7, 8, 9 (светодиод горит пока нажата кнопка).
   b) Программу, которая вводит два одноразрядных числа в клавиатуры, перемножает их и выводит результат на светодиодном индикаторе (два разряда справа).

2) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифр 1, 4, 7 включала бы светодиоды 1, 4, 7 (светодиод горит пока нажата кнопка).
   b) Программу, которая вводит два одноразрядных числа в клавиатуры, складывает их и выводит результат на светодиодном индикаторе (два разряда справа).

3) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифр 1, 4, 7 включала бы светодиоды 1, 4, 7 (светодиод последовательно включается и выключатся при каждом нажатии на кнопку).
   b) Программу, которая вводит два одноразрядных числа в клавиатуры, вычитает их и выводит результат на светодиодном индикаторе (два разряда справа).

4) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифр 1, 2, 3 включала бы светодиоды 1, 2, 3 (светодиод последовательно включается и выключатся при каждом нажатии на кнопку).
   b) Программу, которая вводит два одноразрядных числа в клавиатуры, делит их и выводит результат на светодиодном индикаторе (два разряда справа).

5) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифры 1 - 9, 0 включала бы светодиоды 1 - 9, 0 (светодиод последовательно включается и выключатся при каждом нажатии на кнопку).
   b) Программу, которая вводит восьмиразрядное число на семисегментный индикатор, причём каждое последующее вводимое число меньше или равно предыдущему.

6) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифры 1 - 9, 0 включала бы светодиоды 1 - 9, 0 (светодиод горит пока нажата кнопка).
   b) Программу, которая вводит восьмиразрядное число на семисегментный индикатор, причём каждое последующее вводимое число больше или равно предыдущему.

7) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии «*» перемещает включённый светодиод влево, а «#» - вправо..
   b) Программу, которая вводит последовательно два двухразрядных числа, складывает их и выводит результат на семисегментный индикатор.

8) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифры вызывает моргание светодиода такое число раз , какое указано на клавише.
   b) Программу, которая вводит последовательно два двухразрядных числа, вычитает их и выводит результат на семисегментный индикатор.

9) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифры вызывает перемещение включённого светодиода вправо такое число раз, какое указано на клавише.
   b) Программу, которая вводит последовательно два двухразрядных числа, делит их и выводит результат на семисегментный индикатор.

10) Написать две программы:
   a) Программу опроса клавиатуры, которая отображает на индикаторе число, обозначенное на нажатой клавише.
   b) Программу, которая вводит последовательно два двухразрядных числа, умножает их и выводит результат на семисегментный индикатор.

11) Написать две программы:
   a) Программу опроса клавиатуры, которая отображает на индикаторе число «10 - обозначенное на нажатой клавише».
   b) Программу, которая вводит двухразрядное число, а затем светодиод «3» быстро моргает число раз, равное первой введённой цифре и медленно – число раз, равное второй введённой цифре.

12) Написать две программы:
   a) Программу опроса клавиатуры, которая отображает на дискретном светодиодном индикаторе (3-4-5-6) число в двоичном коде, введённое с клавиатуры.
   b) Программу, которая вводит с клавиатуры число, а затем светодиод «3» моргает число раз, равное введённой цифре, причём длительность включённого состояния каждое включение удваивается.

13) Написать две программы:
   a) Программу опроса клавиатуры, которая отображает на дискретном светодиодном индикаторе (1-2-3-4) число в двоичном коде, введённое с клавиатуры.
   b) Программу, которая вводит с клавиатуры число, а затем светодиод «3» моргает число раз, равное введённой цифре, причём длительность включённого состояния каждое включение уменьшается вдвое.

14) Написать две программы:
   a) Программу опроса клавиатуры, которая включает на дискретном светодиодном индикаторе светодиод, соответствующий нажатой клавише (после нажатия он остаётся гореть). При нажатии на «*» все светодиоды гаснут.
   b) Программу, которая вводит с клавиатуры число, а затем светодиод «3» загорается и затем включённый светодиод «перемещается» вправо столько раз, какая была нажата цифра. Например при нажатии «4» последовательно загораются светодиоды 3-4-5-6.

15) Написать две программы:
   a) Программу опроса клавиатуры, которая включает на семисегментном светодиодном индикаторе светодиод, соответствующий нажатой клавише (A – 1, B – 2, C – 3 … H – 8). При нажатии на «*» все светодиоды гаснут.
   b) Программу, которая вводит с клавиатуры число, а затем светодиод сегмент «D» семисегментного индикатора включается на левом индикаторе, затем горящий сегмент пробегает слева направо и делает это столько раз какая цифра была нажата.

16) Написать две программы:
   a) Программу опроса клавиатуры, которая включает на семисегментном светодиодном индикаторе светодиод, соответствующий нажатой клавише (A – 1, B – 2, C – 3 … H – 8). При нажатии на «*» все светодиоды гаснут.

b) Программу, которая при нажатии цифры на клавиатуре прибавляет её к предыдущему числу (начиная от 0) и выводит его на экран. Выводимое число должно быть трёхразрядным (значит сначала выводится 000, после нажатия на 5 выводится 005, после следующего нажатия например на 3 выводится 008 и т. д.).

17) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифры вызывает перемещение включённого сегмента «A» семисегментного индикатора вправо такое число раз, какое указано на клавише.
   b) Программу, которая вводит с клавиатуры число, умножает его на число, равное номеру включённого переключателя (3, 4, … 10) и выводит его на семисегментный индикатор (значение имеет только самый левый включённый переключатель).

18) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифры 1, 2, или 3 выводит на семисегментный индикатор «A», «B» или «C».
   b) Программу, которая вводит с клавиатуры число, прибавляет его к числам, равным номерам включённых переключателей (3, 4, … 10) и выводит результат на семисегментный индикатор.

19) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифр 7, 8, 9, 5, 1, 2, 3 включает на семисегментном индикаторе сегменты «F», «A», «B», «G», «E», «D», «C». При нажатии на «*» все сегменты гаснут.
   b) Программу, которая при нажатии на числовую клавишу устанавливает яркость свечения индикатора, соответствующее нажатой клавише.

20) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифровых клавиш включает соответствующий светодиод на дискретном светодиодном индикаторе. При повторном нажатии светодиод должен погаснуть.
   b) Программу, которая при нажатии на клавишу «*» увеличивает яркость свечения некоторого числа на семисегментном индикаторе, а при нажатии на «#» - уменьшает.

Лабораторная работа №5

«Прерывания. Работа с таймерами»

Цель работы: Знакомство с принципами работы МП по прерываниям. Изучение работы таймеров микропроцессоров серии МК51.

Приборы и принадлежности: стенд для изучения микропроцессоров МК51, интегрированная среда программирования ProView, ПК.

Теоретическое введение

Прерывания

При работе микропроцессорной системы осуществляется взаимодействие её с внешними устройствами. Так, например, компьютер отслеживает движение мыши, нажатие клавиш на клавиатуре и т. п.

Осуществлять это взаимодействие можно разными способами. Во-первых, можно периодически обращаться к внешним устройствам и опрашивать их текущее состояние. Для этого при написании программы в неё вводится подпрограмма опроса соответствующего устройства и эта подпрограмма периодически вызывается. Достоинство этого метода – простота реализации и гибкость (можно опрашивать любые устройства в пределах адресного пространства, в любом порядке, также при необходимости внесения изменений в МПС достаточно изменить только программу). К недостаткам этого метода надо отнести большую загруженность микропроцессора порою ненужной работой по опросу внешних устройств, количество которой растет в геометрической прогрессии при добавлении новых устройств. Это может существенно замедлить работу МП, поэтому в сложных устройствах (как, например, в персональном компьютере) этот метод почти не применяется, хотя для простых систем он оказывается вполне приемлем.

Второй путь – использование т. н. *прерываний*. В этом случае внешнее устройство при необходимости обмена данными выставляет микропроцессору *запрос прерывания* по некоторой линии (IRQ –interrupt request). Обычно МП имеет несколько линий запроса прерывания: у МК51 – 5 линий, у PC XT – 8, PC AT – 16 и т. д. Причём каждое внешнее устройство имеет свою линию, которые обычно пронумерованы, например для PC XT – IRQ0 – IRQ15. При поступлении запроса прерывания МП заканчивает выполнение текущей команды и переходит к выполнению *подпрограммы обслуживания прерывания*. В языках высокого уровня эти подпрограммы оформляются соответствующим образом, а в программе, написанной в машинных кодах они должны располагаться во вполне определённом месте. Адрес точки входа в подпрограмму обработки прерывания называется *вектором прерывания*. Этот адрес может быть жёстко закреплён за соответствующей линией прерывания или хранится в некоторой области памяти, ставя в соответствие номеру прерывания свой вектор (т. о. в

этом случае вектор можно менять). Обычно по адресу, указываемому вектором, располагается команда безусловного перехода к подпрограмме обслуживания прерывания. После выполнения подпрограммы обработки прерывания МП возвращается к выполнению основной программы. Из сказанного следует основное преимущество работы МП по прерываниям – общение с внешним устройством (ВУ) происходит только при необходимости и не тратится время на пустые опросы ВУ.

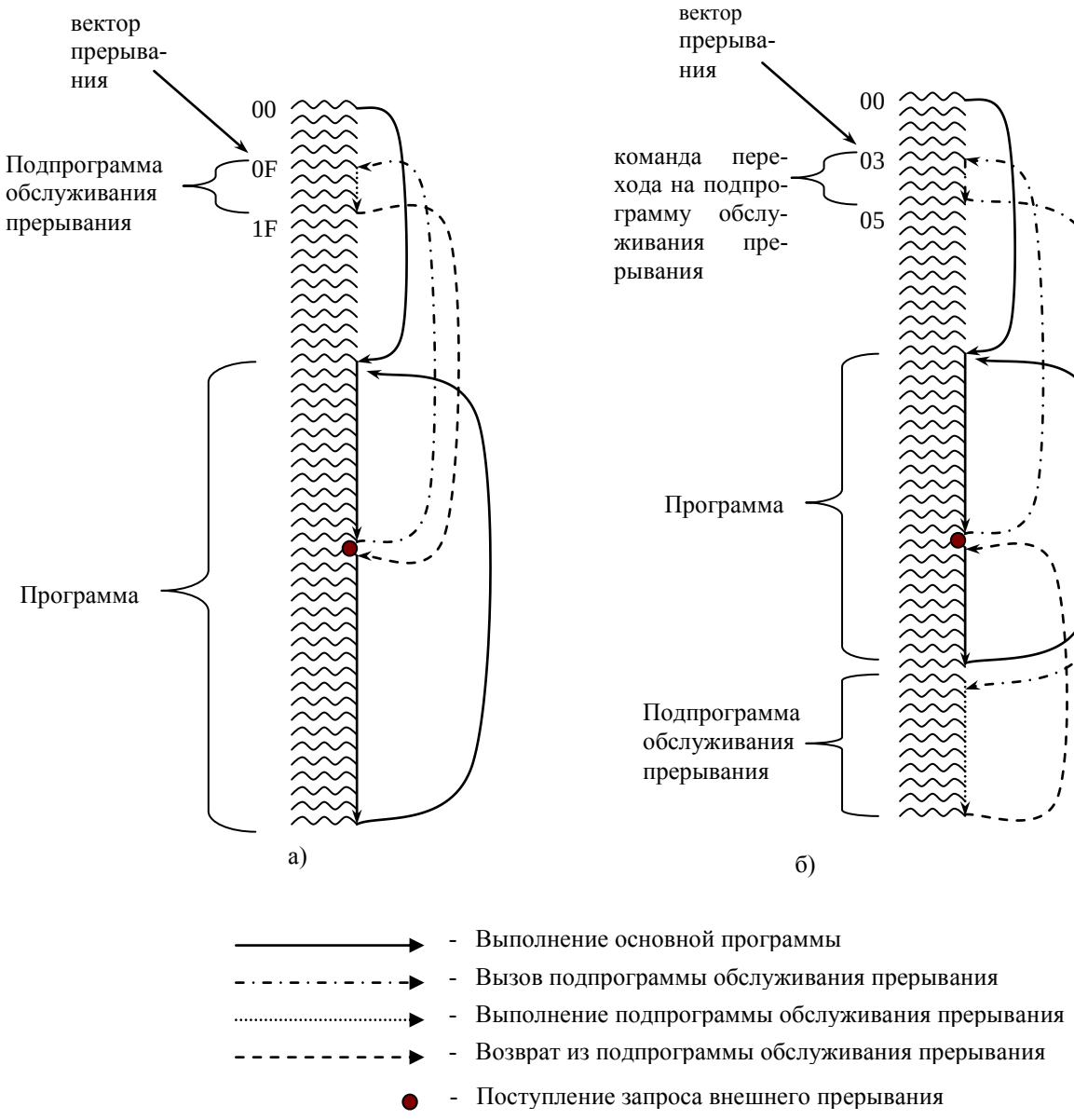


Рисунок 1 - Работа МП по прерываниям:

а) упрощённая схема при коротких подпрограммах обработки прерывания;
б) реальная схема при больших подпрограммах обработки прерывания.

Под прерыванием понимается прекращение выполнения одной программы и переключение процессора на выполнение другой программы. При необходимости выполнение прерванной программы может быть продолжено.

ОМЭВМ серии МК51 имеют 5 линий прерываний:

1. Внешний вход запроса прерывания INT0 (вывод 12 ИС, альтернативное назначение вывода третьего порта P3.2). Вектор 03h.
2. Флаг переполнения таймера-счётчика T/C0. Вектор 0Bh.
3. Внешний вход запроса прерывания INT1 (вывод 13 ИС, альтернативное назначение вывода третьего порта P3.3). Вектор 013h.
4. Флаг переполнения таймера-счётчика T/C1. Вектор 01Bh.
5. Прерывание от последовательного порта (при приёме байта или после передачи байта). Вектор 023h.

Флаги прерываний

У МК51 запрос на прерывание от внешних устройств осуществляется путем установки сигнала на внешнем выводе ИС. В зависимости от битов IT0 и IT1 регистра TCON (см. ниже) возможны два способа запроса на прерывание - *спадом* - при изменении с «1» на «0» на внешнем выводе или *уровнем* – при появлении нуля на соответствующем выводе ИС.

При поступлении запроса на прерывание от внешнего источника или при переполнении таймера в регистре TCON устанавливается в «1» соответствующий бит (*флаг*) запроса прерывания – IE0, IE1, TF0, TF1. Если же запрос поступает от последовательного порта, то устанавливается флаг запроса прерывания TI (после передачи восьмого бита) или RI (после приёма восьмого бита) в регистре SCON.

Если соответствующее прерывание разрешено, то при установке флага процессором автоматически вызывается подпрограмма обслуживания прерывания.

Флаги TF0 и TF1 сбрасываются аппаратно при передаче управления подпрограмме. Флаги IE0 и IE1 сбрасываются аппаратно только при инициировании прерывания спадом, если же вызов прерывания с внешнего входа осуществляется уровнем, то подпрограмма должна так воздействовать на источник прерывания, чтобы к моменту её окончания сигнал запроса прерывания был снят. Флаги TI и RI подпрограмма обслуживания прерывания должна сама устанавливать в «0».

Флаги запроса прерываний в каждый момент доступны программно, поэтому можно, например, вызвать прерывание от таймера 1 записав «1» в бит TF1.

Разрешение прерываний

Каждое прерывание может быть разрешено или запрещено. Регистр IE (interrupt enabled) определяет, какие прерывания разрешены, а какие нет. Для разрешения прерывания в соответствующий бит надо записать единицу.

Таблица 1 - Назначение битов регистров IE и IP

Биты	7	6	5	4	3	2	1	0
Регистр IE	EA			ES	ET1	EX1	ET0	EX0
Регистр IP				PS	PT1	PX1	PT0	PX0
Источник прерывания	все			последова-тельный порт	таймер/ счётчик1	внешний вывод INT1	таймер/ счёт-чик1	внешний вывод INT0

Каждое прерывание независимо от других можно как запретить, так и разрешить. Можно также сразу запретить все прерывания, установив седьмой бит EA регистра IE в ноль (чтобы МП реагировал на прерывания должно быть EA=1).

При работе подпрограммы обработки прерывания может придти запрос на то же прерывание, поэтому, как правило, в начале подпрограммы прерывания надо запретить это прерывание (или все сразу).

Приоритеты прерываний

МП может реагировать на запросы нескольких источников, но их важность не одинакова. Конечно, при одновременном поступлении запросов на прерывание надо обработать сначала тот, который наиболее важен. Для этих целей у МК51 предусмотрен регистр *приоритетов прерываний* IP (interrupt priority). Всего МК51 имеет два уровня прерываний – высокий и низкий. Для определения уровня приоритета каждого прерывания в IP записывается единица (высокий уровень приоритета) или ноль (низкий уровень).

Если во время обработки прерывания с низким уровнем приоритета приходит запрос на прерывание с высоким уровнем приоритета, то подпрограмма обслуживания текущего прерывания прерывается и МП переходит к обработке прерывания более высокого уровня. В случае прихода прерывания с низким уровнем приоритета МП сначала заканчивает обработку текущего прерывания, а только потом начинает обработку поступившего запроса. Если обрабатывается прерывание с высоким уровнем приоритета, то его работа не может быть прервана другим прерыванием.

Схема работы прерываний МК51 и алгоритм её работы приведены на рисунках ниже. На схеме также указан порядок опроса прерываний. Так если запрос на прерывание приходит одновременно от обоих таймеров и они имеют одинаковый уровень приоритета, то сначала будет обработан запрос на прерывание от таймера 0, так как он опрашивается в первую очередь.

Оформление подпрограмм прерываний

Когда вызывается прерывание, то счётчик команд PC, показывающий номер выполняемой команды, загружается в стек (побайтно, сначала старший байт, затем младший), а на его место помещается вектор прерывания (для

МК51 0003h, 000Bh, 0013h, 001Bh, 0023h) и программа продолжает выполняться с указанного адреса. Поэтому подпрограммы обслуживания прерываний должны располагаться начиная с указанных адресов.

В ассемблере для указания компилятору адресов, по которым надо разместить подпрограмму, служит директива ORG. После этой директивы надо указать адрес, начиная с которого будут размещены последующие машинные команды.

Заканчиваться подпрограмма должна командой *RETI,* по которой в счётчик команд загружается из стека старое значение счётчика команд PC и восстанавливается логика прерываний. Дело в том, что при вызове подпрограммы запрещаются все прерывания с таким же или более низким приоритетом, а по команде *RETI* они разрешаются.

Для примера, если будут использоваться прерывания от таймера 1, то подпрограмму следует оформить так:

```
ORG 01Bh          ; Обработчик прерывания от таймера 1 (вектор 01Bh)
                  ; Далее идёт тело подпрограммы
reti              ; конец подпрограммы
```

Если используется только одно прерывание, то подпрограмма обработки прерывания в состоянии уместиться по младшим адресам памяти программ. А когда используется несколько прерываний, то это не всегда возможно, например при использовании двух таймеров размер обработчика прерываний от таймера 0 может быть не более 001Bh-000Bh=0010h, т.е только 16 байт. Если учесть команду *RETI* (1 байт), то остаётся только 15 байт, что составляет около 7 двухбайтных команд. В таком случае по вектору прерывания можно поставить команду безусловного перехода к подпрограмме, расположенной в произвольном месте:

```
ORG 00h       ; Выполнение программы начинается с адреса 00h
ajmp proga    ; Переход к основной программе, минуя обработчики прерываний
ORG 0Bh       ; Обработчик прерывания от таймера 0 (вектор 0Bh)
ajmp timer0   ; Переход к подпрограмме прерывания от таймера 0
ORG 01Bh      ; Обработчик прерывания от таймера 1 (вектор 01Bh)
ajmp timer1   ; Переход к подпрограмме прерывания от таймера 1

proga:              ; Далее идёт тело подпрограммы

timer0:     ; Далее идёт тело подпрограммы обработчика прерываний от таймера 0
            …
        reti ; конец подпрограммы
            …
timer1:      ; Далее идёт тело подпрограммы обработчика прерываний от таймера 1
            …
        reti ; конец подпрограммы
```

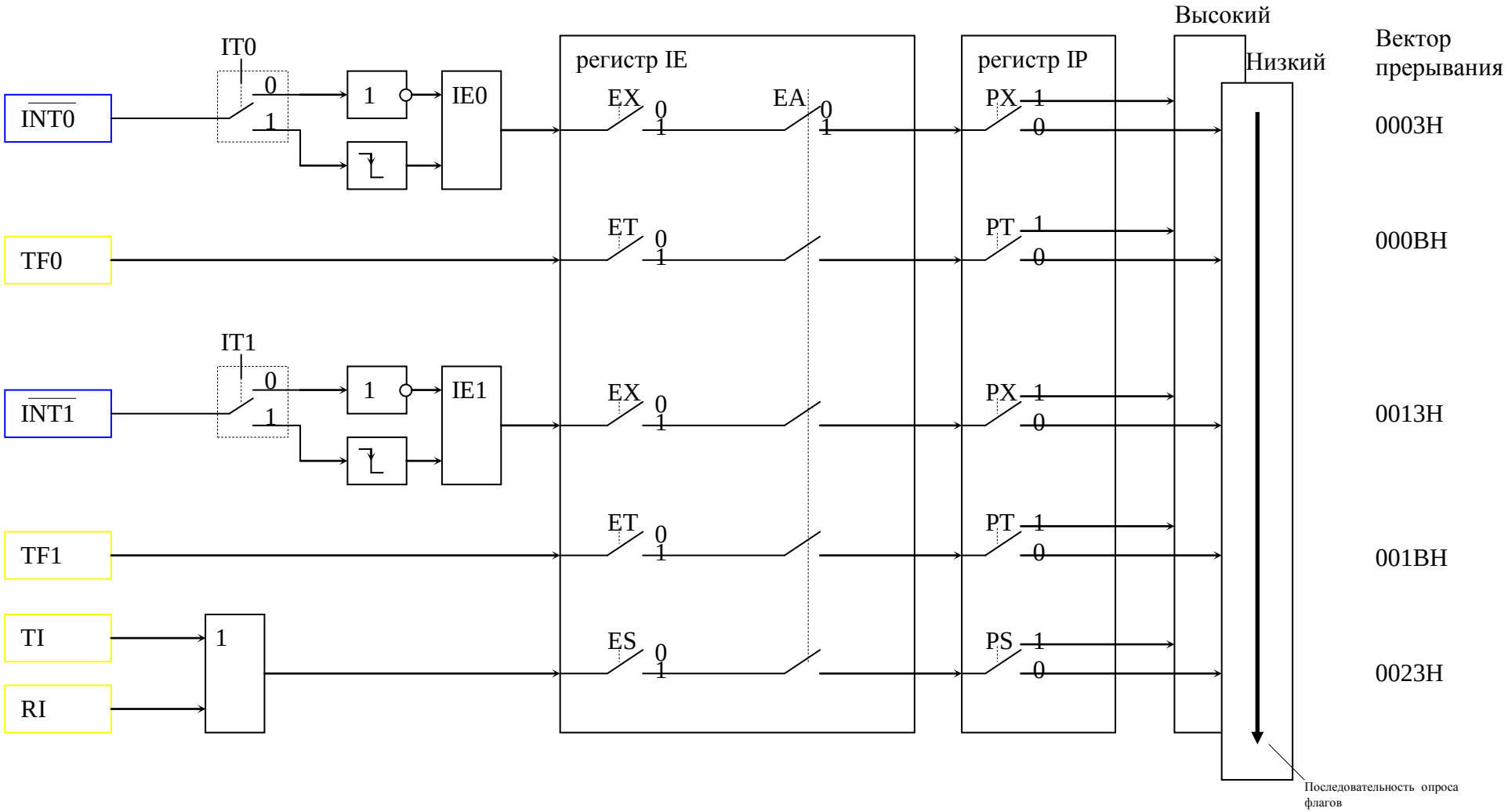


Рисунок 2 - Схема работы прерываний МК51.

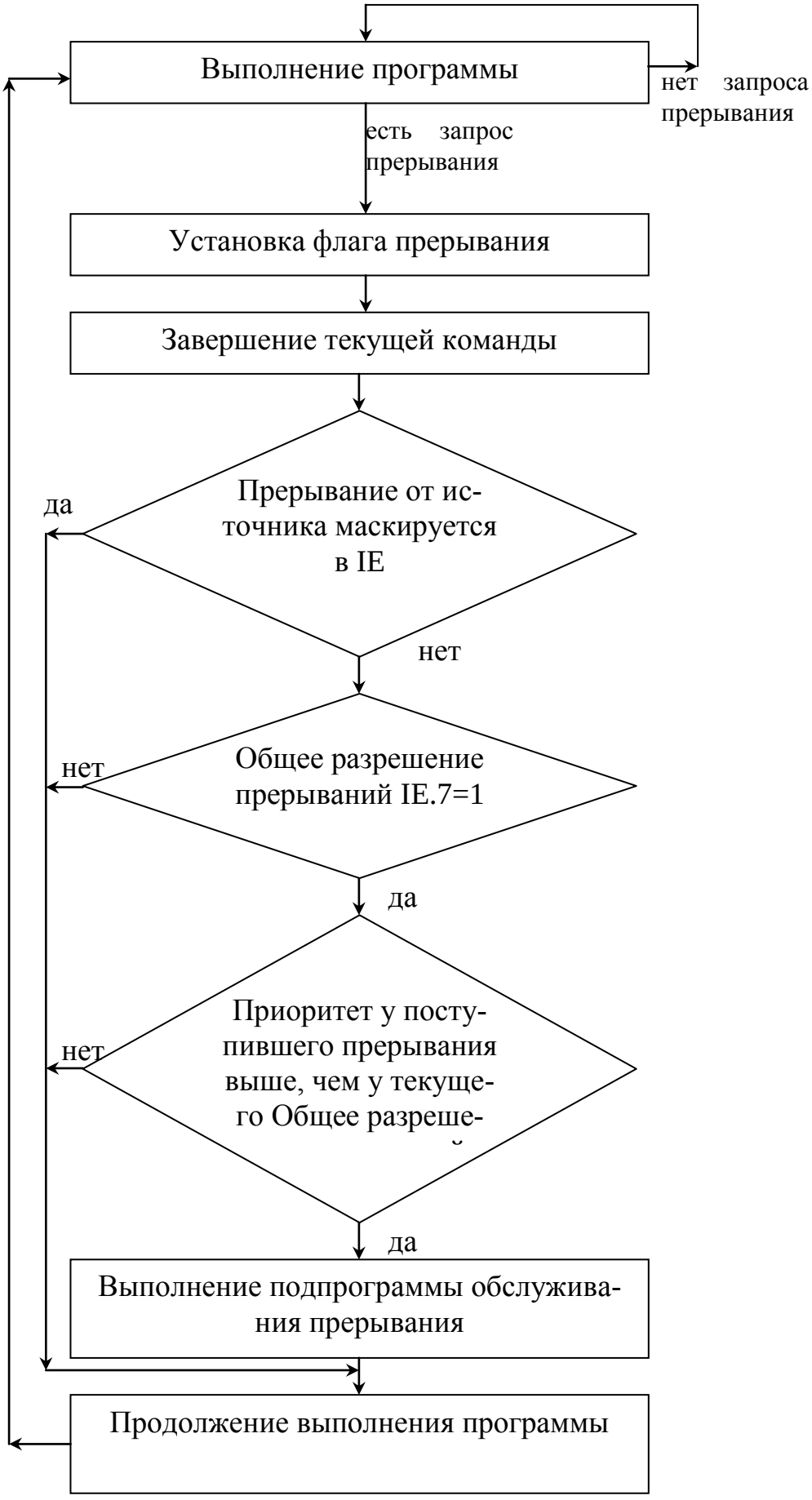


Рисунок 3 - Алгоритм обработки прерываний МК51.

Таймеры МК51

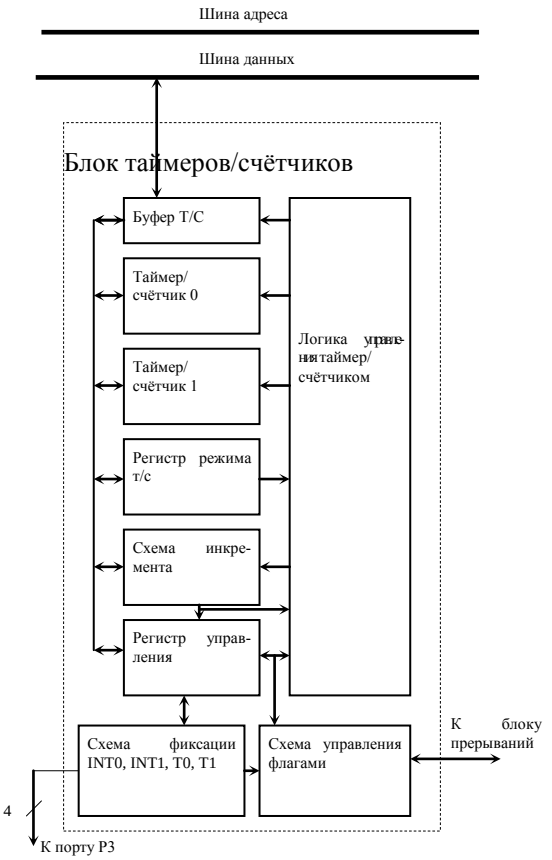
Работа микропроцессора состоит в последовательном выполнении некоторой последовательности команд - программы. Время выполнения программы зависит от микропроцессора, тактовой частоты, выбранного алгоритма, используемых команд. Соответственно и время потраченное на выполнение некоторой процедуры может быть различным. В простейших случаях его можно предсказать (посчитать время на выполнение каждой команды и сложить их), а во многих случаях это невозможно, особенно если МП работает в режиме реального времени и его задача – реагировать на внешние воздействия, которые заранее не известны.

Чтобы можно было отсчитывать промежутки времени в микропроцессорных системах практически всегда присутствуют счётчики-таймеры. Конкретная реализация зависит от архитектуры МП-системы.

Таймеры МП серии МК-51.

Блок таймеров/счётчиков (Т/С) содержит следующие узлы:

1. буфер Т/С,
2. таймер/счётчик 0,
3. таймер/счётчик 1,
4. регистр режима (TMOD),
5. регистр управления (TCON),
6. схема инкремента,
7. логика управления Т/С,
8. схема фиксации,
9. схема управления флагами.



МК-51 содержит два таймера/счётчика (Т/С), каждый из которых имеет два восьмиразрядных регистра (старшие байты TH0 и TH1, младшие байты TL0 и TL1). Соответственно максимальный коэффициент пересчёта – 2^{16}=65536. Указанные регистры доступны программно для записи и чтения. Если какой-нибудь таймер не используется, то его регистры можно использовать по своему усмотрению как регистры общего назначения (РОН).

Для изменения коэффициента пересчёта в регистры Т/С программно заносится нужное число, а затем содержимое Т/С инкрементируется (+1). Признаком конца счёта является переполнение Т/С, т. е. переход из состояния 11111111 11111111 в 00000000 00000000 (хотя можно и программно определять окончание счёта). Переполнение Т/С приводит к установке в «1» флага TF0 или TF1 в регистре TCON и вызове подпрограммы обработки прерывания (т. е. пе-

реходу к выполнению программы начиная с соответствующего вектора – 0Bh для T/C0 или 01Bh для T/C1).

Инкремент регистров происходит либо от внутреннего тактового генератора, либо от внешнего входа. В первом случае инкремент происходит в каждом машинном цикле и поэтому максимальная частота счёта – *f*/12 (в 12 раз меньше частоты кварцевого резонатора) Во втором случае инкремент происходит при изменении входного сигнала на счётном входе с «1» на «0» (P3.4 = T0 – вход для T/C0, P3.5 = T1 – вход для T/C1), при этом проверка состояния входа производится один раз в машинном цикле и поэтому максимальная частота воспринимаемых сигналов - *f*/24.

Регистр режимов TMOD (адрес 089h) определяет режим работы T/C. Название и назначение битов TMOD приведены в таблице 1.

Таблица 1 - Назначение битов регистра TMOD

<table>
<tr><th>Биты</th><th>Наименование</th><th>Назначение</th><th>Примечание</th></tr>
<tr><td>0-1
4-5</td><td>M0.0-M1.0
M0.1-M1.1</td><td>Определяют режим работы T/C, отдельно для каждого таймера.
<table>
<tr><th>M1</th><th>M0</th><th>Режим</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>2</td></tr>
<tr><td>1</td><td>1</td><td>3</td></tr>
</table></td><td rowspan="3">Все биты устанавливаются программно. Биты 0-3 определяют работу T/C0, биты 4-7 – T/C1</td></tr>
<tr><td>
2
6</td><td>
C/T0
C/T1</td><td>Работа соответствующего T/C в качестве
таймера – «0»
счётчика – «1»</td></tr>
<tr><td>
3
7</td><td>
GATE0
GATE1</td><td>Разрешение управления таймером с внешнего входа:
упр. запрещено – «0»
упр. разрешено – «1»</td></tr>
</table>

Согласно таблице, каждый T/C может работать в одном из 4-х режимов. Также их работа возможна или в качестве таймеров или счётчиков внешних событий.

Бит GATE разрешает управления таймером с внешнего входа. Если, например, сигнал GATE0 = 1, то управление T/C0 возможно с внешнего входа третьего порта МП – входа P3.2 (обозначается как INT0).

Регистр управления TCON (088h) хранит код управляющего слова. Он также доступен программно для записи и чтения. Название и назначение битов TMOD приведены в таблице 2.

Биты TR определяют, будет ли использоваться T/C или нет.

Флаги TF устанавливаются при переполнении счётчика. Если при этом не запрещены прерывания от Т/С, то начинается обработка соответствующего прерывания, а флаг прерывания аппаратно сбрасывается в «0».

Для восприятия внешних событий у МП МК-51 есть два входа внешних прерываний – INT0 и INT1 (соответственно P3.2 и P3.3). Они могут реагировать либо на нулевой уровень входного сигнала (IT=0), или на спад с «1» на «0» (IT=1). Длительность сигнала прерывания должна быть не менее 2 машинных циклов.

Таблица 2 - Назначение битов TCON

Биты	Наименование	Назначение
6 4	TR1 TR0	Биты включения Т/С: «0» - Т/С выключен «1» - Т/С включен
7 5	TF1 TF0	Флаги переполнения Т/С, устанавливаются в «1» при переполнении соответствующего счётчика, сбрасываются в «0» при передаче управления подпрограмме обработки соответствующего прерывания.
2 0	IT1 IT0	Определяют вид прерывания по внешнему входу: прерывание уровнем – «0» прерывание спадом – «1»
3 1	IE1 IE0	Флаги запроса внешних прерываний. Устанавливаются в «1» при поступлении на внешний вход сигнала прерывания.

При поступлении внешнего сигнала прерывания на соответствующий вход и разрешении этого прерывания устанавливается флаг прерывания IE и вызывается подпрограмма обслуживания этого прерывания. Флаг прерывания IE сбрасывается аппаратно только в том случае, если прерывание вызвано спадом сигнала (IT=1). Если же оно вызвано уровнем, то реакция МП на прерывание должна быть такой, чтобы внешнее устройство само сняло сигнал прерывания.

Режимы работы Т/С

Режим работы Т/С определяется битами M0 и M1 в регистре TMOD. Режимы работы задаются отдельно для каждого Т/С и не зависят друг от друга, кроме случая когда Т/С0 в режиме 3.

1. *Режим работы 0.*

В этом режиме Т/С работает как тринадцатиразрядный счётчик, используется 8 разрядов старшего регистра (TH0 и TH1) и пять младших бит младшего регистра (TL0 и TL1). Три старших бита регистров TL0 и TL1 не используются. Это ограничение и сам режим 0 сделаны лишь для совместимости с МП серии МК48.

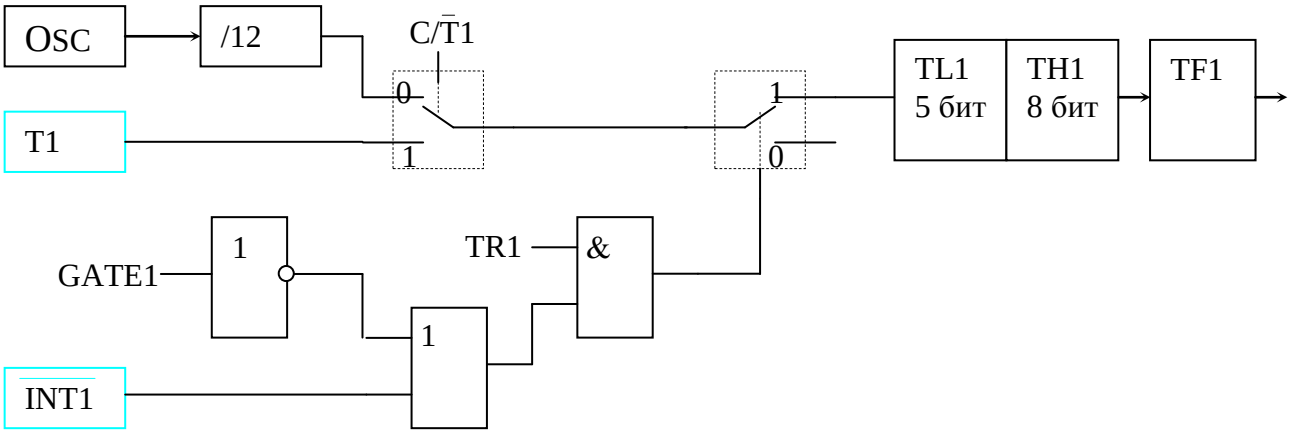


Рисунок 4 - Логика работы T/C1 в режиме 0.

Счёт разрешается при TR=1 (на рисунке 4 при TR1 = «1» на выходе элемента «И» возможна «1», разрешающая поступление импульсов на вход счётчика). Значит, чтобы счётчик T/C1 считал, надо установить шестой бит TCON = «1». Кроме того, надо чтобы и на втором входе элемента «И» была единица. Это можно сделать сбросив в «0» бит GATE1, либо, при GATE=«1», установив «1» на внешнем входе INT1 (вход P3.3 третьего порта). В последнем случае таймер считает при INT1 = «1» и останавливается при INT1 = «0».

Импульсы, которые считает счётчик, поступают либо с внешнего входа T1 (вход P3.5 третьего порта), либо с делителя на 12 тактовой частоты процессора. Откуда поступают счётные импульсы, зависит от бита C/T1 (шестой бит регистра TMOD): при C/T1 = «1» с внешнего входа (работа в режиме счётчика внешних событий), при C/T1 = «0» - с делителя тактовой частоты (работа в режиме таймера).

При работе микропроцессора с резонатором на 12 МГц делитель на двенадцать делит её до 1 МГц. В этом случае длительность импульсов после делителя – 1 мкс. Максимальный коэффициент пересчёта в режиме 0 - 2^{13} = 8192, значит максимальная длительность отсчитываемых интервалов времени – 8,192 мс (около 122 Гц).

Приведём пример настройки T/C1 в режим 0:

```
mov TMOD,#00000000b    ; загрузка регистра режима
                       ; GATE1=0 – управление с внешнего входа запрещено
                       ; C/T1=0 – работа в режиме таймера
                       ; M1.1=0, M1.0=0 –задание режима 0
                       ; те же параметры заданы для T/C0
mov TCON,#01000000b    ; загрузка регистра управления
                       ; TR1=1 – T/C1 включен (TR0=0 – T/C0 выключен)
                       ; все биты кроме 4 и 6 устанавливать при настройке не нужно
                       ; хотя прибегать к их установке можно, например чтобы программно
                       ; вызвать прерывание  от таймера
```

Приведённые команды можно использовать в процедуре инициализации перед началом выполнения основной программы.

2. *Режим работы 1.*

В этом режиме T/C работает как шестнадцатиразрядный счётчик. Всялогика работы аналогична режиму 0, с той лишь разницей, что оба регистра TH и TL используются полностью (8+8 бит).

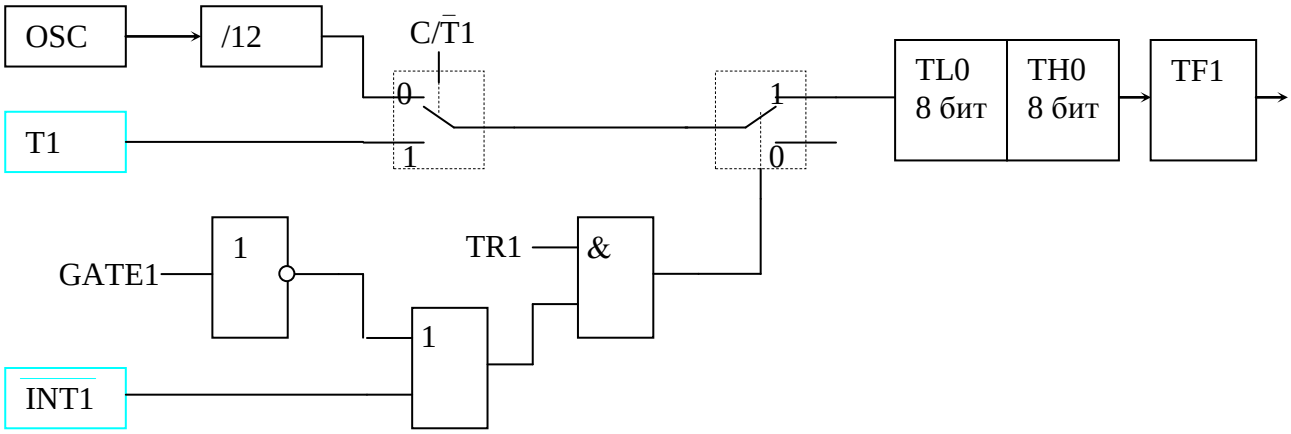


Рисунок 5 - Логика работы T/C1 в режиме 1.

3. *Режим работы 2.*

Это режим с аппаратной перезагрузкой счётного регистра. Логика работы показана на рисунке 6 для T/C1, T/C0 работает аналогично.

В режиме 2 младший регистр TL1 является счётным, т. е. при каждом поступлении на его вход импульса происходит инкремент (увеличение на единицу). Логика работы, определяющая откуда приходит этот импульс – с предделителя на 12 или внешнего входа - определяется регистрами TMOD и TCON так же как и в режимах 0 и 1. Различие состоит в том, что при переполнении регистра TL1 (т. е. переходе 1111111→00000000) происходит не только установка флага TF1, но и перезагрузка 8 бит из регистра TH1 в TL1. В регистр TH1 предварительно можно программно занести любое нужное число. При загрузке содержимое TH1 не меняется. С помощью такого механизма можно менять коэффициент пересчёта и вызывать прерывание от таймера с нужной частотой.

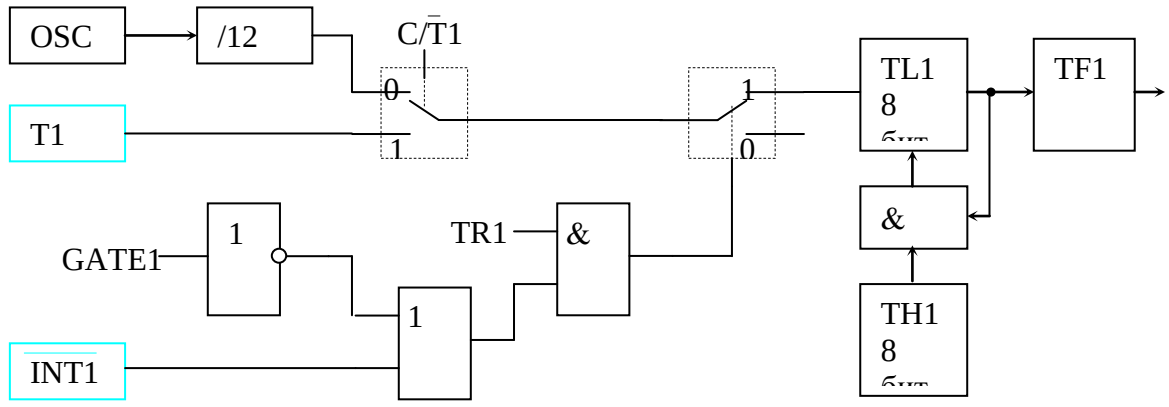


Рисунок 6 - Логика работы T/C1 в режиме 2.

Приведём пример. Допустим нам нужно один раз в 100 мкс вызывать прерывание от таймера, чтобы программа периодически выполняла некоторое действие. Будем использовать таймер T/C0 в режиме 2.

Пусть микропроцессор тактируется с частотой 12 МГц. Тогда при работе в режиме таймера счётные импульсы с предделителя будут поступать с частотой 1 МГц. Чтобы получить продолжительность времени между прерываниями 100 мкс надо загрузить в регистр TH0 число 156 = 9Ch. После его загрузки в TL0 будет подсчитано 99 импульсов, пока не будет достигнуто число 255 = FFh. При приходе следующего импульса будет установлен флаг TF0 и будет вызвано соответствующее прерывание.

```
mov TMOD,#00000010b    ; загрузка регистра режима
                       ; GATE0=0 – управление с внешнего входа запрещено
                       ; C/T0=0 – работа в режиме таймера
                       ; M0.1=1, M0.0=0 –задание режима 0
                       ; те же параметры заданы для T/C0
mov TCON,#00010000b    ; загрузка регистра управления
                       ; TR0=1 – T/C0 включен (TR1=0 – T/C1 выключен)
mov TH0,#09Ch          ; загрузка в TH0 числа 9Ch
```

После выполнения этой части программы будет включён таймер T/C0 и если прерывание от таймера разрешено, то при каждой установке флага TF0 будет вызвана подпрограмма его обслуживания.

4. *Режим работы 3.*

Этот режим различен для T/C0 и T/C1.

T/C0 в режиме 3 имеет два счётных восьмиразрядных регистра - TH0 и TL0. Причём счётчик на основе TL0 работает аналогично режимам 0 и 1 (см. рисунок 4). Регистр TH0 может использоваться только как счётчик таймера, причём он включается битом управления TR1 от T/C1.

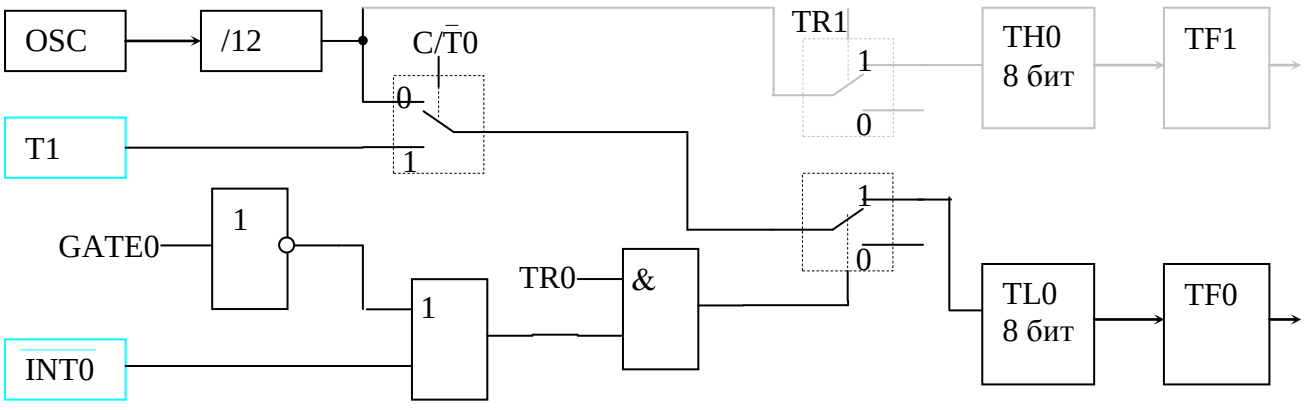


Рисунок 7 - Логика работы T/C1 в режиме 3.

T/C1 при работе T/C0 в режиме 3 всегда включен при GATE1=0 и выборе его режима 0, 1 или 2. Кроме потери бита управления TR1 T/C1 также лишается и флага TF1, который теперь устанавливается при переполнении TH0. Таймер-счётчик T/C1 аппаратно связан с блоком последовательного интерфейса (ПИ) и

хотя при режиме 3 у T/C0 он не вырабатывает флага прерывания, он используется для отсчёта временных интервалов ПИ. Чтобы выключить T/C1 (при работе T/C0 в режиме 3) надо также перевести его в режим 3.

Итак, режим 3 позволяет использовать восьмибитный таймер на TH0, таймер или счётчик на TL0, последовательный интерфейс на T/C1.

Пример

Приведём пример использования таймера и работы МП по прерываниям.

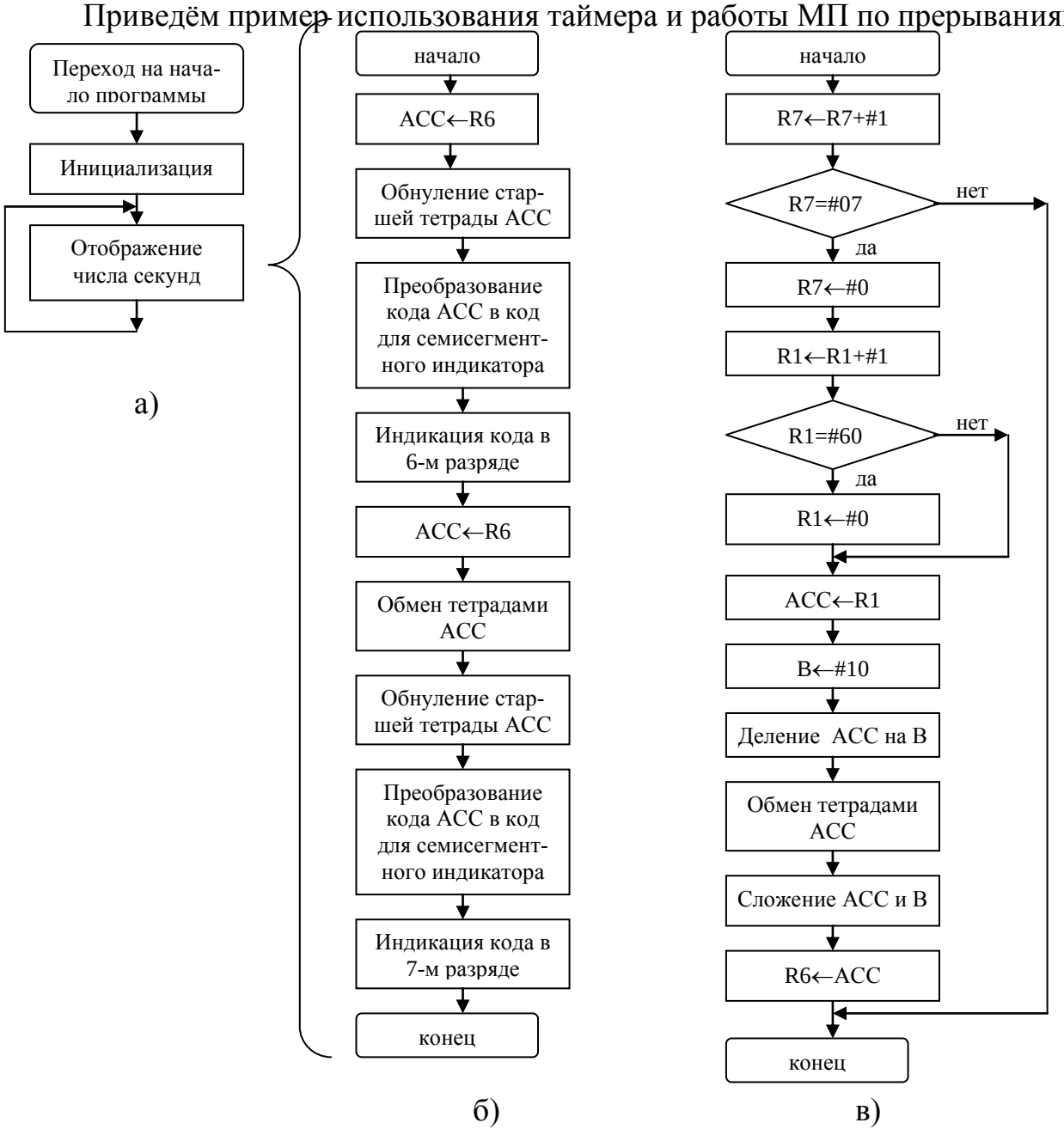


Рисунок 8. Алгоритмы работы программы «секундомер»:

а) алгоритм работы программы;
б) алгоритм индикации двухразрядного числа, хранящегося в регистре R6;
в) алгоритм обработчика прерываний.

В примере программируется работа стенда СУ-МК (используется МП 89С51) в качестве секундомера. В двух разрядах семисегментного индикатора отображается число секунд от 0 до 59. При достижении 60 счёт начинается сначала.

В программе регистр R7 используется для счёта количества поступивших прерываний. Когда это количество достигает 07Ah (т. е. проходит одна секунда) увеличивается на единицу содержимое регистра R1, хранящего шестнадцатиразрядный код числа секунд. Тут же в обработчике прерываний код из R1 преобразуется в двоично-десятичный код (старшая тетрада – число десятков секунд от 0000 до 1001 и младшая тетрада – число единиц секунд от 0000 до 1001) и помещается в регистр R6.

Индикация производится следующим образом. Из регистра R6 число записывается в аккумулятор, его старшая тетрада обнуляется, полученный код числа единиц секунд переводится в код для семисегментного индикатора с помощью подпрограммы semiseg, а затем записывается в регистр индикатора. В порт P1 записывается 6, чтобы отображение было в 6-м разряде индикатора. После задержки отображается число десятков секунд в 7-м разряде индикатора. Для этого снова из регистра R6 число записывается в аккумулятор, его тетрады меняются местами. Теперь его младшая тетрада хранит число десятков. Старшая тетрада обнуляется, полученный код числа десятков секунд переводится в код для семисегментного индикатора с помощью подпрограммы semiseg, а затем также записывается в регистр индикатора. В порт P1 записывается 7, чтобы отображение было в 7-м разряде индикатора. Далее следует задержка.

После отображения единиц и десятков опять повторяется индикация. Этот процесс иногда прерывается таймером, обработчик прерываний от которого меняет код отображаемого регистра R6.

```
        ; Программа циклического счёта от 0 до 59
        ; Регистр R6 содержит отображаемое на семисегментном индикаторе
        ;    число секунд
        ;  R7 - чило прерываний от таймера
        ;  R1 - число секунд в шестнадцатиричном коде

ORG 00h          ; директива компилятора, размещающая следующие команды
                 ; начиная с указанного адреса
        jmp init     ; переход на основную программу (инициализацию)
;-------------------------------------------------------------------------------
ORG 01Bh              ; Обработчик прерывания от таймера 1 (вектор 01Bh)
        inc R7        ; Прерывание вызывается каждые 8,192 мс и каждый раз R7
                      ; увеличивается на 1
        cjne R7,#07Ah,qwe  ; Переход на qwe если R7 ещё не равен указанному
                           ; значению 07Ah ,  (8,192 мс x 122 = 1 с )
        mov R7,#00h       ; При достижении R7 = 07Ah = 122 этот регистр
                          ; обнуляется и содержимое R1
        inc R1            ;  увеличивается на 1 (R1 хранит число секунд в
                          ;шестнадцатиричном коде)
        cjne R1, #60 ,asd  ; Проверка на достижение 1 минуты
        mov R1,#0          ; Обнуление R1
asd:    mov A,R1           ; Далее идёт десятичная коррекция содержимого R1
```

```
        mov B,#10
        div AB              ; A - число десятков < 6, B - число единиц < 9
        swap A              ; перемещаем число десятков в старшую тетраду
        add A,B             ; складываем и в аккумуляторе скорректированное число
        mov R6,A            ; возвращаем в R6 скорректированное число
        cpl P3.4            ; Инвертируем разряд порта P3.4, чтобы светодиод
                            ; мигал с частотой 1 Гц
qwe:    reti                ; Выход из подпрограммы обработки прерывания
;-------------------------------------------------------------------------------
init:   mov tmod,#00000000B ; режим 0 для таймера,
                            ;управление с внешнего входа запрещено
        mov tcon,#01000101B ; включён таймер 1
        mov ip,#00001000B   ; высший приоритет прерывания у T/C1
        mov ie,#10001000B   ; разрешение прерываний от таймера 1
        mov A,#0FFh
        mov DPTR,#8000h     ;загрузка адреса семисегментного индикатора
                            ;в указатель данных
        movx @dptr,A        ;гашение семисегментного индикатора
        mov R1,#00h         ; R1 - число секунд в шеснадцатиричном коде

start:                  ; цикл отображения содержимого регистра R6, в котором
                        ; содержится число секунд
        mov A,R6        ; помещаем в ACC число секунд, хранящееся в R6
        anl A,#0Fh      ; выделяем в ACC младшую тетраду
        lcall semiseg   ; преобразуем код ACC в семисегментный код
        movx @DPTR,A    ; отображение младшей тетрады ACC (младший
                        ; разряд числа секунд)
        mov P1,#06h     ; индикация числа единиц в 6 разряде
        acall delay     ; задержка

        mov A,R6        ; помещаем в ACC число секунд, хранящееся в R6
        swap A          ; меняем местами тетрады ACC
        anl A,#0Fh      ; выделяем в ACC младшую тетраду
        lcall semiseg   ; преобразуем код ACC в семисегментный код
        movx @DPTR,A    ; отображение младшей тетрады ACC (младший
                        ; разряд числа секунд)
        mov P1,#07h     ; индикация числа десятков в 7 разряде
        acall delay     ; задержка

        jmp start
;-------------------------------------------------------------------------------
semiseg:  ; подпрограмма преобразует код числа  из ACC в код для
          ; семисегментного индикатора
          ; чисдо должно быть в ACC, туда же возвращается и код для индикатора
        mov ie,#00001000B   ; запрещение прерываний от таймера
        jz a0           ;если содержимое акк=0, то на a0
        dec A
        jz a1
        dec A
        jz a2
        dec A
        jz a3
```

```
        dec A
        jz a4
        dec A
        jz a5
        dec A
        jz a6
        dec A
        jz a7
        dec A
        jz a8
        ajmp a9
a0:       mov A,#0C0h
        ajmp endsemiseg
a1:       mov A,#0F9h
        ajmp endsemiseg
a2:       mov A,#0A4h
        ajmp endsemiseg
a3:       mov A,#0B0h
        ajmp endsemiseg
a4:       mov A,#099h
        ajmp endsemiseg
a5:       mov A,#092h
        ajmp endsemiseg
a6:       mov A,#082h
        ajmp endsemiseg
a7:       mov A,#0F8h
        ajmp endsemiseg
a8:       mov A,#080h
        ajmp endsemiseg
a9:       mov A,#090h
endsemiseg:  mov ie,#10001000B     ; разрешение прерываний от таймера
        ret                ; конец подпрограммы semiseg
;-------------------------------------------------------------------------------
delay:   ; подпрограмма задержки
      push ACC
      mov A,#0FFh
del:    djnz ACC,del
      pop ACC
      ret
;-------------------------------------------------------------------------------
      end
```

В приведённой программе шестнадцатеричное число преобразуется в двоично-десятичный код путём деления шестнадцатеричного числа на 10 для выделения числа десятков. Получаемый при этом остаток содержит число единиц. Затем эти числа объединяются, и полученное двоично-десятичное число помещается в R6. Заметим, что такой перевод можно сделать с помощью команды десятичной коррекции аккумулятора, как сделано в фрагменте обработчика прерывания ниже:

```
;-------------------------------------------------------------------------------
ORG 01Bh                    ; Обработчик прерывания от таймера 1 (вектор 01Bh)
        inc R7              ; Прерывание вызывается каждые 8,192 мс и каждый
                            ; раз R7 увеличивается на 1
        cjne R7,#07Ah,qwe   ;Переход если R7 ещё не равен указанному значению
                            ; 07Ah ,    (8,192мс x 122 = 1с)
        mov R7,#00h         ; При достижении R7 = 07Ah = 122 этот регистр
                            ; обнуляется и содержимое R1 увеличивается на 1
        inc R1              ; (R1 хранит число секунд в шестнадцатиричном коде)
        cjne R1,#05Ah,asd   ; Проверка на достижение 1 минуты
        mov R1,#0           ; Обнуление R1
asd:    mov A,R1            ; Далее идёт десятичная коррекция содержимого R1
        clr C               ; Команда cjne меняет флаг C, а он влияет на
                            ; десятичную коррекциюю с помощью команды da,
                            ; поэтому его надо обнулить
        da A                ; Десятичная коррекция
        mov R1,A            ; возвращаем в  R1 и R6 скорректированное число
        mov R6,A
        cpl P3.4            ; Инвертируем разряд порта P3.4, чтобы светодиод
                            ; мигал с чатотой 1 Гц
qwe:    reti                ; Выход из подпрограммы обработки прерывания
```

Порядок выполнения работы

1. На языке ассемблера написать программу, реализующую задание.
2. Отладить программу с помощью отладчика, входящего в систему ProView.
3. Сгенерировать HEX-файл, загрузить его в лабораторный стенд с помощью загрузчика «Загрузчик СУ-МК». Убедиться в работоспособности программы.

Задания для самостоятельной работы

1) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифр 7, 8, 9 включала бы светодиоды 7, 8, 9 (светодиод горит 1 с).
   b) Программу «бегущие огни»: светодиоды 3…0 по очереди включаются и гаснут, имитируя перемещение включённого светодиода. Время между переключениями – 1 с.

2) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифр 1, 4, 7 включала бы светодиоды 1, 4, 7 (светодиод горит 5 с).
   b) Программу «бегущие огни»: светодиоды 3…0 по очереди включаются и гаснут, имитируя перемещение включённого светодиода. Время между переключениями регулируется - скорость переключения задаётся переключателями 7…10 в двоичном коде.

3) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифр 1, 4, 7 включала бы светодиоды 1, 4, 7 (светодиод включается на время, соответствующее нажатой клавише – 1 с, 4 с и 7 с).
   b) Программу «бегущие огни»: светодиоды 3…9, 0 по очереди включаются, а потом все одновременно гаснут (горит 3, горит 3 и 4, горит 3, 4 и 5 и т. д.). Время между переключениями регулируется - скорость переключения задаётся переключателями 7…10 в двоичном коде.

4) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифр 1, 2, 3 включала бы светодиоды 1, 2, 3 (светодиод три раза включается и выключается, время включённого состояния 0.5 с).
   b) Программу «бегущие огни»: среди светодиодов 3…9, 0 постоянно включено два соседних светодиода, которые «перемещаются вправо» (горит 3 и 4, горит 4 и 5, горит 5 и 6 и т. д.). Время между переключениями регулируется - скорость переключения задаётся переключателями 7…10 в двоичном коде.

5) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифры 1 - 9, 0 включала бы светодиоды 1 - 9, 0 (время включённого состояния соответствует нажатой кнопке).
   b) Программу «бегущие огни»: среди светодиодов 3…9, 0 постоянно включены все светодиоды, кроме одного, который «перемещается вправо» (не горит только 3, не горит только 4, не горит только 5 и т. д.). Время между переключениями регулируется - скорость переключения задаётся переключателями 7…10 в двоичном коде.

6) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифры 1 - 9, 0 включала бы светодиоды 1 - 9, 0 (светодиод горит пока нажата кнопка и продолжает гореть после её отпускания ещё 2 с).
   b) Программу «секундомер», которая выводит на семисегментный индикатор время от 0 до 1 с с точностью 0.01 с.

7) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии «*» перемещает включённый светодиод влево, а «#» - вправо (если нажатие короткое, то перемещает на один шаг, если длительное, то на количество позиций, соответствующих длительности нажатого состояния в секундах).
   b) Программу «секундомер», которая выводит на семисегментный индикатор время от 0 до 10 мин с с точностью 1 с.

8) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифры вызывает моргание светодиода такое число раз, какое указано на клавише. Время включённого и выключенного состояния – 1 с.
   b) Программу «секундомер», которая выводит на семисегментный индикатор время от 0 до 60 с точностью 1 с. Секундомер включается кнопкой «*» и выключается - «#».

9) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифры вызывает перемещение включённого светодиода вправо такое число раз, какое указано на клавише. Длительность одного шага - 0.5 с.
   b) Программу «секундомер», которая выводит на семисегментный индикатор время от 0 до 60 с точностью 1 с. Секундомер включается и выключается кнопкой «#».

10) Написать две программы:
   a) Программу опроса клавиатуры, которая отображает на индикаторе число, обозначенное на нажатой клавише. После отображения числа оно каждую секунду уменьшается на 1 до нуля.
   b) Программу «секундомер», которая выводит на семисегментный индикатор время от 0 до 60. Время за которое показания увеличиваются на единицу определяется дискретными переключателями 7…10 от 1/16 до 1 с.

11) Написать две программы:
   a) Программу опроса клавиатуры, которая отображает на индикаторе число «10 - обозначенное на нажатой клавише». Время отображения -3 с.
   b) Программу, которая плавно меняет яркость свечения светодиода 1 от минимальной до максимальной в течении 1 с. Затем светодиод гаснет и процесс повторяется.

12) Написать две программы:
   a) Программу опроса клавиатуры, которая отображает на дискретном светодиодном индикаторе (3-4-5-6) число в двоичном коде, введённое с клавиатуры. Время отображения – 2 с.
   b) Программу, которая вводит с клавиатуры число, а затем светодиод «3» моргает число раз, равное введённой цифре, причём длительность включённого состояния каждое включение удваивается (время первого включения 0.2 с).

13) Написать две программы:
   a) Программу опроса клавиатуры, которая отображает на дискретном светодиодном индикаторе (1-2-3-4) число в двоичном коде, введённое с клавиатуры. Время отображения – 2 с.

b) Программу, которая вводит с клавиатуры число, а затем светодиод «3» моргает число раз, равное введённой цифре, причём длительность включённого состояния каждое включение уменьшается вдвое. (время первого включения 10 с).

14) Написать две программы:
   a) Программу опроса клавиатуры, которая включает на дискретном светодиодном индикаторе светодиод, соответствующий нажатой клавише (после нажатия он остаётся гореть). При нажатии на «*» все светодиоды плавно гаснут в течении 3 с.
   b) Программу, которая вводит с клавиатуры число, а затем светодиод «3» загорается и включённый светодиод «перемещается» вправо столько раз, какая была нажата цифра. Например, при нажатии «4» последовательно загораются светодиоды 3-4-5-6. Время между переключениями – 1 с.

15) Написать две программы:
   a) Программу опроса клавиатуры, которая включает на семисегментном светодиодном индикаторе светодиод, соответствующий нажатой клавише (A – 1, B – 2, C – 3 … H – 8). При нажатии на «*» все светодиоды плавно гаснут в течении 2 с.
   b) Программу, которая вводит с клавиатуры число, а затем светодиод сегмента «D» семисегментного индикатора включается на левом индикаторе, затем горящий сегмент пробегает слева направо, и делает это столько раз, какая цифра была нажата. Время переключения – 0.5 с.

16) Написать две программы:
   a) Программу опроса клавиатуры, которая включает на семисегментном светодиодном индикаторе светодиод, соответствующий нажатой клавише (A – 1, B – 2, C – 3 … H – 8). При нажатии на «*» все светодиоды гаснут. Включение должно происходить плавно в течении 1 с.
   b) Программу «бегущие огни»: светодиоды 3…0 по очереди включаются и гаснут, имитируя перемещение включённого светодиода. Светодиоды включаются и выключаются плавно, в течении 0.5 с.

17) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифры вызывает перемещение включённого сегмента «A» семисегментного индикатора вправо такое число раз, какое указано на клавише. Перед перемещением делается задержка 2 с.
   b) Программу «бегущие огни»: светодиоды 3…0 по очереди включаются и гаснут, имитируя перемещение включённого светодиода. Яркость свечения регулируется - нарастает за 20 с до максимума, а затем падает тоже за 20 с. Скорость переключения -0.8 с.

18) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифры 1, 2, или 3 выводит на семисегментный индикатор «A», «B» или «C». Яркость свечения плавно увеличивается в течении 5 с, а затем цифра резко гаснет.
   b) Программу «бегущие огни»: среди светодиодов 3…9, 0 постоянно включено два соседних светодиода, которые «перемещаются вправо» (горит 3 и 4, горит 4 и 5, горит 5 и 6 и т. д.). Яркость свечения регулируется - нарастает за 20 с до максимума, а затем падает тоже за 20 с. Скорость переключения -0.5 с.

19) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифр 7, 8, 9, 5, 1, 2, 3 включает на семисегментном индикаторе сегменты «F», «A», «B», «G», «E», «D», «C». При нажатии на «*» все сегменты плавно гаснут в течении 3 с.
   b) Программу, которая при нажатии на числовую клавишу устанавливает яркость свечения индикатора, соответствующую нажатой клавише. Изменение яркости должно происходить в течении 1 с, а отображается число в левом разряде семисегментного индикатора, соответствующее коду дискретных переключателей 7…10 в двоичном коде.

20) Написать две программы:
   a) Программу опроса клавиатуры, которая при нажатии цифровых клавиш включает соответствующий светодиод на дискретном светодиодном индикаторе. При повторном нажатии светодиод должен погаснуть. Включение и выключение светодиода должно быть плавным в течении 2 с.
   b) Программу, которая при нажатии на клавишу «*» увеличивает яркость свечения числа на семисегментном индикаторе от половины до максимума, а при нажатии на «#» - уменьшает. Изменение яркости происходит в течении 1 с. Число отображается в левом разряде семисегментного индикатора и соответствует коду дискретных переключателей 7…10 в двоичном коде.

Литература:

1. Боборыкин А.В. и др. Однокристальны микроЭВМ. М.:МИКАП, 1994. – 400 с.

2. http://vsystem.ulz.ru/doc/mk51/start_mk51.shtml

3. А. Фрунзе, С. Хоркин. Однокристальные микроЭВМ. РАДИО, 1994 г., №№ 8–12/1994, №№ 1—3/1995.

4. В. Я. Нерода, В. Э. Торбинский, Е. Л. Шлыков. Однокристальные микро-ЭВМ MCS51. Архитектура. – М., изд. Диджитал Компонентс. 1995, с. 164.

Приложение 1

Правила записи программ на языке ассемблера

Программа, по которой работает микропроцессор, записывается в виде последовательности двоичных чисел, которыми кодируются команды. Каждой команде соответствует свой двоичный код (машинный код). В принципе можно составлять программу путём написания последовательности команд в виде двоичных чисел, но это очень не удобно. Ниже для примера приведена программа для МК51 в машинных кодах.

```
:10000000908B00749BF0908300E0F520A2008201A9
:100010009203A20182029204A2008202720472037D
:08002000E433908400F080DE5F
:00000001FF
```

Для облегчения написания программ существуют языки высокого уровня. Самым простым и приближённым к программированию в машинных кодах является *ассемблер*. В нём каждой команде микропроцессора ставится в соответствие *мнемоническое обозначение* команды, например AJMP. Аббревиатура AJMP – мнемоническое обозначение команды «абсолютный переход по указанному адресу». Машинный код, соответствующий этой команде – 10000001. Понятно, что гораздо проще и понятней запись AJMP (Absolute JuMP), чем 10000001.

Каждый микропроцессор имеет свою систему команд и свою внутреннюю структуру. Поэтому обозначения команд для разных МП разные и программы, написанные на языке ассемблера для разных микропроцессоров, не подходят друг для друга.

Программа микропроцессора в машинных кодах и на языке ассемблера представляет собой последовательность определённым образом записанных команд. В простейшем случае команды программы выполняются последовательно – сначала первая команда, затем вторая и т. д.

В более сложном случае при выполнении программы команды выполняются последовательно, но может происходить переход от одного места программы к другому. Для указания, с какого места далее нужно выполнить программу, надо указать адрес нужной команды. В программной памяти МП адрес перехода указывается в двоичном виде, а в программе на языках высокого уровня применяются т. н. *метки*, которые ставятся в нужном месте программы. При переводе программы в машинные коды в те места, которые ссылаются на метки, подставляется адрес команды, стоящей сразу после метки и записанный в двоичном коде.

Если нет специальных указаний в программе, то первая команда программы располагается начиная с нулевого адреса, за ней располагается следующая команда и т. д. Однако с помощью некоторых средств (см. далее) порядок расположения команд в памяти программ можно менять.

Текст программы на языке ассемблера имеет определенный формат. Каждая команда (и псевдокоманда) представляет собой строку, состоящую от одного до четырех элементов:

МЕТКА: ОПЕРАЦИЯ ОПЕРАНД1,ОПЕРАНД2 ;КОММЕНТАРИЙ

Элементы (поля) могут отделяться друг от друга произвольным числом пробелов.

МЕТКА. Поле метки содержит символическое имя, по которому программа может обратиться к команде, следующей за меткой. При переводе программы в машинные коды (компиляции) *компилятор*[2] определяет адрес строки, где стоит метка и подставляет его в те места программы, которые ссылаются на эту метку. Метка может содержать буквы латинского алфавита и цифры (начинается с буквы). Ассемблер МК51 допускает использование в метках символа подчеркивания (_). Длина метки не должна превышать 31 символ. Метка всегда завершается двоеточием (:).

Псевдокоманды ассемблера не преобразуются в двоичные коды, а потому не могут иметь меток. Исключение составляют псевдокоманды резервирования памяти и определения данных (DS, DB, DW). У псевдокоманд, осуществляющих определение символических имен, в поле метки записывается определяемое символическое имя, после которого двоеточие не ставится.

В качестве меток не могут быть использованы мнемонические обозначения команд, псевдокоманд, операторов ассемблера, регистров и других внутренних блоков МП.

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

Для МК51 используется строго определенный и ограниченный набор мнемонических кодов (42 аббревиатуры).

ОПЕРАНД. В этих полях определяются операнды, участвующие в операции, т. е. данные с которыми будут производиться некоторые действия. Команды ассемблера могут быть без- , одно- или двухоперандными. Операнды разделяются запятой (,). В случае использования двух операндов на первом месте стоит операнд-приёмник, а на втором – операнд-источник. Так при пересылке данных из одной ячейки в другую на первом месте стоит адрес ячейки куда надо переслать данные, а на втором – откуда.

Операнд может быть задан непосредственно или в виде его адреса (прямого или косвенного), Непосредственный операнд представляется числом (MOV A, #15)[3] или символическим именем (ADDC A,#OPER2)[4] с обязательным указанием префикса непосредственного операнда (#).

---

[2] Компилятор – программа, переводящая программу, написанную на языке высокого уровня в программу в машинных кодах. Этот процесс называется комапиляцией.

[3] Для этой команды происходит пересылка десятичного числа 15 в регистр **A**.

Прямой адрес операнда может быть задан мнемоническим обозначением (MOV A, P1)[5], числом (INC 40)[6], символическим именем (MOV A, MEMORY). В этом случае необходимые данные берутся по указанному адресу.

Указанием на косвенную адресацию служит префикс @. В этом случае в указанной ячейке хранится не сами данные, а адрес, по которому они хранятся. Например MOVX A, @DPTR[7].

В командах передачи управления операндом может являться число (LCALL 0135h), метка (JMP LABEL), косвенный адрес (JMP @A) или выражение (JMP $-2, где $ - текущее содержимое счетчика команд). В этом случае операнд указывает адрес, по которому нужно осуществить переход.

Используемые в качестве операндов символические имена и метки должны быть определены. Числа должны быть 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 шестнадцатеричной (можно b, q, d, h). Число без суффикса по умолчанию считается десятичным.

Обработка **выражений** в процессе трансляции. Ассемблер МК51 допускают 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 выполнение 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 выделить младший и старший байты 2-байтного операнда.

КОММЕНТАРИЙ. Поле комментарий начинается с точки с запятой (;) и может содержать любой текст поясняющий программу. Поле комментария полностью игнорируется ассемблером, а потому в нем допустимо использовать любые символы.

Псевдокоманды ассемблера. Ассемблирующая программа транслирует исходную программу в объектные (машинные) коды. Хотя транслирующая программа берет на себя многие из рутинных задач программиста, такие как присвоение действительных адресов, преобразование чисел, присвоение действительных значений символьным переменным и т.п., программист все же должен указать ей некоторые параметры: начальный адрес прикладной про-

---

[4] В этой команде производится сложение числа, определённого в переменной OPER2 с числом, находящимся в аккумуляторе. Результат сложения будет помещён в первый операнд – аккумулятор **A**.

[5] Для этой команды происходит пересылка содержимого порта **P1** в регистр-аккумулятор **A**.

[6] По этой команде происходит увеличение на единицу (инкремент) содержимого ячейки с адресом 40.

[7] Здесь происходит пересылка в регистр **A** из ячейки внешней памяти, а адрес этой ячейки находится в регистре **DPTR**.

граммы, конец ассемблируемой программы, форматы данных и т.п. Всю эту информацию программист вставляет в исходный текст своей прикладной программы в виде псевдокоманд (директив) ассемблера, которые только управляют процессом трансляции и не преобразуются в коды объектной программы.

Псевдокоманда **ORG 10H** задает ассемблеру адрес ячейки памяти (10H), в которой должна быть расположена следующая за ней команда прикладной программы.

Псевдокомандой **EQU** можно любому символическому имени, используемому в программе, поставить в соответствие определенный операнд. Например, запись **GOD EQU 2008** приводит к тому, что в процессе ассемблирования всюду, где встретится символическое имя GOD, оно будет заменено числом 2008.

Символические имена операндов, переопределяемых в процессе исполнения программы, определяются псевдокомандой SET:

ALFA SET 3

ALFA SET ALPA+1

Ассемблер МК51 позволяет определить символическое имя как адрес внутренних (псевдокоманда DATA), внешних (XDATA) данных или адрес бита (псевдокоманда BIT). Например, директива

ERROR_FLAG BIT 25H.3

определяет символическое имя ERROR_FLAG как третий бит ячейки ОЗУ с адресом 25H.

Псевдокоманда DB обеспечивает занесение в память программ константы, представляющей собой байт.

Псевдокомандой END программист дает ассемблеру указание об окончании трансляции.

В результате трансляции должна быть получена карта памяти программ, где каждой ячейке памяти поставлен в соответствие хранящийся в ней код.

Каждая команда после перевода в объектные коды представляется одним, двумя или тремя байтами (записываемыми затем в ячейки памяти программ). В первом байте всегда располагается код операции, во втором и третьем - непосредственный операнд, адрес прямоадресуемого операнда, адрес перехода или смещение.

В приложении 2 приведены команды ассемблера МК51, там же указаны машинные коды, соответствующие этим командам.

Приложение 2

Система команд МК51

Таблица 1 - Условные обозначения в описании команд

Обозначение, символ	Назначение
A	Аккумулятор
Rr	Регистры текущего выбранного банка регистров.
r	Номер загружаемого регистра, указанного в команде.
direct	Прямо адресуемый 8-битовый внутренний адрес ячейки данных, который может быть ячейкой внутреннего ОЗУ данных (0-127) или SFR (128-255).
@Rr	Косвенно адресуемая 8-битовая ячейка внутреннего ОЗУ данных.
data 8	8-битовое непосредственное данное, код операции входящее в код операции (КОП).
data 16	16-битовое непосредственное данное входящее в код операции КОП.
data H	Старшие биты (15-8) непосредственных 16-битовых данных.
data L	Младшие биты (7-0) непосредственных 16-битовых данных.
addr 11	11-битовый адрес назначения.
addr 16	16-битовый адрес назначения.
addr L	Младшие биты адреса назначения.
disp 8	8-битовый байт смещения со знаком.
bit	1 бит с прямой адресацией, адрес которого содержит КОП, находящийся во внутреннем ОЗУ данных или SFR.
al5,a14…a0	Биты адреса назначения.
(X)	Содержимое элемента X
((X)) (X)[M]	Содержимое по адресу, хранящемуся в элементе X Разряд M элемента X
(X)[M1-M2]	Группа разрядов M1-M2 элемента X
	Операции:
+	сложения
-	вычитания
*	умножения
/	деления
AND	логического умножения (операция И)
OR	логического сложения (операция ИЛИ)
XOR	сложения по модулю 2 (операция "Исключающее ИЛИ")
/X	инверсия элемента X

Команда ACALL <addr 11 >

Команда "абсолютный вызов подпрограммы" вызывает безусловно подпрограмму, размещенную по указанному адресу. При этом счетчик команд увеличивается на 2 для получения адреса следующей команды, после чего полученное 16-битовое значение PC помещается в стек (сначала следует младший байт), и содержимое указателя стека также увеличивается на два. Адрес перехода получается с помощью конкатенации старших бит увеличенного содержимого счетчика команд, битов старшего байта команды и младшего байта команды.

Ассемблер: ACALL <метка>

Код: A10 A9 A8 1 0 0 0 1 | A7 A6 A5 A4 A3 A2 Al A0

Время: 2 цикла

Алгоритм: (PC):=(PC)+2
(SP):=(SP)+1
((SP)):=(PC[7-0])
(SP):=(SP)+1
((SP)):=(PC[l5-8))
(PC[10-0]):=A10A9A8 II A7A6A5A4A3A2A1A0,
(где II - знак конкатенации (сцепление))

Пример: ;ДО ВЫПОЛНЕНИЯ КОМАНДЫ ACALL
;(SP)=07H
;метка MT1 соответствует адресу: 0345H,
;т.е. (PC)=0345H
ACALL MT1 ;расположена по адресу 028DH, т.е.
;(PC)=028DH
;ПОСЛЕ ВЫПОЛНЕНИЯ КОМАНДЫ
;(SP)=09H, (PC)=0345H,
;ОЗУ [08]=8FH, ОЗУ [09]=02H.

Команда ADD A,<байт-источник>

Эта команда ("сложение") складывает содержимое аккумулятора A с содержимым байта-источника, оставляя результат в аккумуляторе. При появлении переносов из разрядов 7 и 3, устанавливаются флаги переноса (C) и дополнительного переноса (AC) соответственно, в противном случае эти флаги сбрасываются. При сложении целых чисел без знака флаг переноса "C" указывает на появление переполнения. Флаг переполнения (OV) устанавливается, если есть перенос из бита 6 и нет переноса из бита 7, или есть перенос из бита 7 и нет — из бита 6, в противном случае флаг OV сбрасывается. При сложении целых чисел со знаком флаг OV указывает на отрицательную величину, полученную при суммировании двух положительных операндов или на положительную сумму для двух отрицательных операндов.

Для команды сложения разрешены следующие режимы адресации байта-источника:

1) регистровый;
2) косвенно-регистровый;

3) прямой;
4) непосредственный.

1) Ассемблер: ADD A,Rn ; где n=0-7
Код: 00101 rrr , где rrr=000-111
Время: 1 цикл
Алгоритм: (A):=(A)+(Rn), где n=0-7
C:=X, OV:=X, AC:=X, где X=(0 или 1)
Пример: ;(A)=C3H, (R6)=AAH
ADD A,R6 ;(A)=6DH, (R6)=AAH
;(AC)=0, (C)=l, (OV)=1

2) Ассемблер: ADD A,@Ri ; где i=0,l
Код: 0010011i , где i=0,1
Время: 1 цикл
Алгоритм: (A):=(A)+((Ri)), где 1=0,1
C:=X, OV:=X, AC:=X, где X=(0 или 1)
Пример: ;(A)=95H, (Rl)=31H, (ОЗУ [31])=4CH
ADD A,@R1 ;(A)=E1H, (ОЗУ [31])=4CH,
; (C)=0, (AC)=1, (OV)=0

3) Ассемблер: ADD A,<direct>
Код: 00100101 direct address
Время: 1 цикл
Алгоритм: (A):=(A)+(direct)
C:=X, OV:=X, AC:=X, где X=(0 или 1)
Пример: ;(A)=77H, (ОЗУ [90])=FFH
ADD A,90H ;(A)=76H, (ОЗУ [90])=FFH,
;(C)=1, (OV)=0, (AC)=1

4) Ассемблер: ADD A,<#data>
Код: 00100100 #data
Время: 1 цикл
Алгоритм: (A):=(A)+#data
C:=X, OV:=X, AC:=X, где X=(0 или 1)
Пример: ;(A)=09H
ADD A,#0D3H ;(A)=DCH,
;(C)=0, (OV)=0, (AC)=0

Команда ADDC A. <байт-источник>

Эта команда ("сложение с переносом") одновременно складывает содержимое байта-источника, флаг переноса и содержимое аккумулятора A, оставляя результат в аккумуляторе. При этом флаги переноса и дополнительного переноса устанавливаются, если есть перенос из бита 7 или бита 3, и сбрасываются в противном случае. При сложении целых чисел без знака флаг переноса указывает на переполнение. Флаг переполнения (OV) устанавливается, если имеется перенос бита 6 и нет переноса из бита 7 или есть перенос из бита 7 и нет — из бита 6, в противном случае OV сбрасывается. При сложении целых чисел со знаком OV указывает на отрицательную величину, полученную при суммиро-

вании двух положительных операндов или на положительную сумму от двух отрицательных операндов.

Для этой команды разрешены следующие режимы адресации байта-источника:

1) регистровый;
2) косвенно-регистровый;
3) прямой;
4) непосредственный.

1) Ассемблер: ADDC A,Rn ; где n=0-7
Код: 00111rrr , где rrr=000-111
Время: 1 цикл
Алгоритм: (A):=(A)+(C)+(Rn)
(C):=X, (AC):=X, (OV):=X, где X=(0 или 1)
Пример: ;(A)=B2H, (R3)=99,
;(C)=1
ADDC A,R3 ;(A)=4CH, (R3)=99,
;(C)=1, (AC)=0, (OV)=1

2) Ассемблер: ADDC A,@Ri ; где i=0, 1
Код: 0011011i , где i=0,1
Время: 1 цикл
Алгоритм: (A):=(A)+(C)+((Ri))
(C):=X, (AC):=X, (OV):=X, где X=(0 или 1)
Пример: ;(A)=D5H, (R0)=3AH,
;(ОЗУ[3A])=1AH, (C)=1
ADDC A,@R0 ;(A)=F0H, (ОЗУ[3A]=1AH),
;(C)=0, (AC)=1, (OV)=0

3) Ассемблер: ADDC A,<direct>
Код: 00110101 direct address
Время: 1 цикл
Алгоритм: (A):=(A)+(C)+(direct)
(C):=X, (AC):=X, (OV):=X, где X=(0 или 1)
Пример: ;(A)=11H, (ОЗУ[80])=DFH, (C)=l
ADDC A,80H ;(A)=F1H, (C)=0, (AC)=1, (OV)=0 4) Ассемблер: ADDC A,#data
Код: 001101001 #data
Время: 1 цикл
Алгоритм: (A):=(A)+(C)+(direct)
(C):=X, (AC):=X, (OV):=X, где X=(0 или 1)
Пример: ;(A)=55H; (C)=0
ADDC A,#55H ;(A)=AAH, (C)=0, (AC)=0, (OV)=1

Команда AJMP <addr11>

Команда "абсолютный переход", передает управление по указанному адресу, который получается при конкатенации пяти старших бит счетчика команд PC (после увеличения его на два), 7—5 битов кода операции и второго байта коман-

ды. Адрес перехода должен находится внутри одной страницы объемом 2 Кбайт памяти программы, определяемой пятью старшими битами счетчика команд.

Ассемблер: AJMP <метка>
Код: A10 A9 A8 0 0 0 0 1 A7 A6 A5 A4 A3 A2 Al A0
Время: 2 цикла
Алгоритм: (PC[15-0]):=(PC[15-0])+2, (PC[10-0]):=<addr11>
Пример: ;(PC)=028FH
;Метке MT2 соответствует адрес 034AH
AJMP MT2 ;(PC)=034AH

Команда ANL <байт-назначения>,<байт-источник>

Команда "логическое "И" для переменных-байтов" выполняет операцию логического "И" над битами указанных переменных и помещает результат в байт-назначения. Эта операция не влияет на состояние флагов.

Два операнда обеспечивают следующие комбинации шести режимов адресации:

—байтом назначения является аккумулятор (A):

1) регистровый;
2) прямой;
3) косвенно-регистровый;
4) непосредственный;

—байтом назначения является прямой адрес (direct):

5) прямой аккумуляторный;
6) непосредственный (байт-источник равен константе).

1) Асемблер: ANL A,Rn ; где n=0-7
Код: 01011rrr , где rrr=000-111
Время: 1 цикл
Алгоритм: (A):=(A) AND (Rn)
Пример: ;(A)=FEH, (R2)=C5H
ANL A,R2 ;(A)=C4H, (R2)=C5H
2) Ассемблер: ANL A,<direct>
Код: 01010101 direct address
Время: 1 цикл
Алгоритм: (A):=(A) AND (direct)
Пример: ;(A)=A3H, (PSW)=86H
ANL A,PSW ;(A)=82H, (PSW)=86H
3) Ассемблер: ANL A,@Ri ; где i=0,l
Код: 0101011i ,где i=0,l
Время: 1 цикл
Алгоритм: (A):=(A) AND ((Ri))
Пример: ;(A)=BCH, (ОЗУ[35])=47H, (R0)=35H
ANL A,@R0 ;(A)=04H, (ОЗУ[35])=47H

4) Ассемблер: ANL A,#data
Код: 01010100 #data8
Время: 1 цикл
Алгоритм: (A):=(A) AND #data
Пример: ;(A)=36H
ANL A,#0DDH ;(A)=14H
5) Ассемблер: ANL <direct>,A
Код: 01010010 direct address
Время: 1 цикл
Алгоритм: (direct):=(direct) AND (A)
Пример: ;(A)=55H, (P2)=AAH
ANL P2,A ;(P2)=00H, (A)=55H
6) Ассемблер: ANL <direct>,#data
Код: 01010011 direct address #data8
Время: 2 цикла
Алгоритм: (direct):=(direct) AND #data
Пример: ;(P1)=FFH
ANL P1,#73H ;(P1)=73H

Примечание. Если команда "ANL" применяется для изменения содержимого порта, то значение, используемое в качестве данных порта, будет считываться из "защелки" порта, а не с выводов БИС.

Команда ANL C,<бит источника>

Команда "логическое "И" для переменных-битов", выполняет операцию логического "И" над указанными битами. Если бит-источник равен "0", то происходит сброс флага переноса, в противном случае флаг переноса не изменяет текущего значения. "/" перед операндом в языке ассемблера указывает на то, что в качестве значения используется логическое отрицание адресуемого бита, однако сам бит источника при этом не изменяется. На другие флаги эта команда не влияет.

Для операнда-источника разрешена только прямая адресация к битам.

1) Ассемблер: ANL C,<bit>
Код: 10000010 bit address
Время: 2 цикла
Алгоритм: (C):=(C) AND (bit)
Пример: ;(C)=1, P1[0]=0
ANL C,P1.0 ;(C)=0, P1[0]=0
2) Ассемблер: ANL C,</bit>
Код: 10110000 bit address
Время: 2 цикла
Алгоритм: (C):=(C) AND (/bit)
Пример: ;(C)=1, (AC)=0
ANL C,/AC ;(C)=1, (AC)=0

Команда CJNE <байт назначения>,<байт источник>,<смешение>

Команда "сравнение и переход, если не равно" сравнивает значения первых двух операндов и выполняет ветвление, если операнды не равны. Адрес перехода (ветвления) вычисляется при помощи сложения значения (со знаком), указанного в последнем байте команды, с содержимым счетчика команд после увеличения его на три.

Флаг переноса "С" устанавливается в "1", если значение целого без знака <байта назначения> меньше, чем значение целого без знака <байта источника>, в противном случае перенос сбрасывается (если значения операндов равны, флаг переноса сбрасывается). Эта команда не оказывает влияния на-операнды.

Операнды, стоящие в команде, обеспечивают комбинации четырех режимов адресации:

—если байтом-назначения является аккумулятор:

1) прямой
2) непосредственный,

—если байтом-назначения является любая ячейка ОЗУ с косвенно-регистровой или регистровой адресацией:

3) непосредственный к регистровому
4) непосредственный к косвенно-регистровому

1) Ассемблер: CJNE A,<direct>,<метка>
Код: 10110101 direct address rel8
Время: 2 цикла
Алгоритм: (PC):=(PC)+3,
если (direct)<(A) то (PC):=(PC)+<rel8>, C:=0
если (direct)>(A), то (PC):=(PC)+<re18>, C:=l
Пример: ;(A)=97H, (P2)=F0H, (C)=0
CJNE A,P2,MT3
MT3: CLR A ;(A)=97H, (P2)=F0H, (C)=l
;Адрес, соответствующий метке
;MT3 вычисляется , как
;(PC):=(PC)+3+(re18)

2) Ассемблер: CJNE A,#data,<метка>
Код: 10110100 #data8 rel8
Время: 2 цикла
Алгоритм: (PC):=(PC)+3,
если #data<(A), то (PC)+<re18>, C:=0
если #data8>(A), то (PC):=(PC)+<re18>, C:=l
Пример: ;(A)=FCH, (C)=l
CJNE A,#0BFH,MT4
MT4: INC A ;(A)=FDH, C=0,
;(PC):=(PC)+3+(rel8)

3) Ассемблер: CJNE Rn,#data,<метка> ; где n=0-7
Код: 1 0 1 1 1 rrr #data8 rel8
Время: 2 цикла
Алгоритм: (PC):=(PC)+3,

если #data<(Rn), то (PC):=(PC)+<rel8>, C:=0
если #data8>(Rn), то (PC)+<rel8>, C:=l

Пример: ;(R7)=80H, (C)=0
CJNE R7,#81H,MT5
MT5: NOP ;(R7)=80H, (C)=l,
;(PC):=(PC)+3+(rel8)

4) Ассемблер: CJNE @Ri,#data,<метка> ; где i=0,l
Код: 1011011i #data8 rel8
Время: 2 цикла
Алгоритм: (PC):=(PC)+3,
если #data<((Ri)), то (PC)+<re18>,C:=0
если #data8>((Ri)), то (PC)+<rel8>, C:=l

Пример: ;(R0)=41H, (C)=l, (ОЗУ[411)=57H
CJNE @R0,#29H,MT6
MT6: DEC R0
;(ОЗУ[41])=57H, (C)=0, ;(PC):=(PC)+3+(rel8)

Команда CLR A

Команда "сброс аккумулятора" сбрасывает (обнуляет) содержимое аккумулятора А. На флаги команда не влияет.

Ассемблер: CLR A
Код: 11100100
Время: 1 цикл
Алгоритм: (A):=0

Пример: ;(A)=6DH, (C)=0, (AC)=1
CLR A ;(A)=00H, (C)=0, (AC)=1

Команда CLR <bit>

Команда "сброс бита" сбрасывает указанный бит в нуль. Эта команда работает с флагом переноса "С" или любым битом с прямой адресацией.

1) Ассемблер: CLR C
Код: 11000011
Время: 1 цикл
Алгоритм: (C):=0

Пример: ;(C)=1
CLR C ;(C)=0

2) Ассемблер: CLR <bit>
Код: 11000010 bit address
Время: 1 цикл
Алгоритм: (bit):=0

Пример: ;(P1)=5EH (01011110B)
CLR P1.3 ;(P1)=56H (01010110B)

Команда CPL A

Команда "инверсия аккумулятора" каждый бит аккумулятора инвертирует (изменяет на противоположный). Биты, содержащие "единицы", после этой команды будут содержать "нули", и наоборот. На флаги эта операция не влияет.

```
Ассемблер:   CPL  A
Код:         11110100
Время:       1 цикл
Алгоритм:    (A):=/(A)
Пример:                ;(A)=65H  (01100101B)
             CPL   A  ;(A)=9AH  (10011010B)
```

Команда CPL <bit>

Команда "инверсия бита" инвертирует (изменяет на противоположное значение) указанный бит. Бит, который был "единицей", изменяется в "нуль" и наоборот. Команда CPL может работать с флагом переноса или с любым прямо адресуемым битом. На другие флаги команда не влияет.

```
1) Ассемблер:  CPL <bit>
Код:           10110010  bit address
Время:         1 цикл
Алгоритм:      (bit):=/(bit)
Пример:                    ;(P1)=39H (00111001B)
               CPL   P1.1
               CPL   P1.3  ;(P1)=33H  (00110011B)
2) Ассемблер:  CPL   C
Код:           10110011
Время:         1 цикл
Алгоритм:      (C):=/(C)
Пример:                ;(C)=0, (AC)=1, (OV)=0
               CPL C ;(C)=1, (AC)=1, (OV)=0
```

Примечание: Если эта команда используется для изменения информации на выходе порта, значение, используемое как исходные данные, считывается из "защелки" порта, а не с выводов БИС.

Команда DA A

Команда "десятичная коррекция аккумулятора для сложения" упорядочивает 8-битовую величину в аккумуляторе после выполненной ранее команды сложения двух переменных (каждая в упакованном двоично-десятичном формате). Для выполнения сложения может использоваться любая из типов команд ADD или ADDC. Если значение битов 3—0 аккумулятора (A) превышает 9 (XXXX 1010—XXXX 1111) или, если флаг AC равен "1", то к содержимому (A) прибавляется 06, получая соответствующую двоично-десятичную цифру в младшем полубайте. Это внутреннее побитовое сложение устанавливает флаг переноса, если перенос из поля младших четырех бит распространяется через все старшие биты, а в противном случае — не изменяет флаг переноса. Если после этого флаг переноса равен "1", или если значение четырех старших бит

(7—4) превышает 9 (1010 XXXX- 1111 XXXX), значения этих старших бит увеличивается на 6, создавая соответствующую двоично-десятичную цифру в старшем полубайте. И снова при этом флаг переноса устанавливается, если перенос получается из старших битов, но не изменяется в противном случае. Таким образом, флаг переноса указывает на то, что сумма двух исходных двоично-десятичных переменных больше чем 100. Эта команда выполняет десятичное преобразование с помощью сложения 06, 60, 66 с содержимым аккумулятора в зависимости от начального состояния аккумулятора и слова состояния программы (PSW).

Ассемблер: DA A
Код: 11010100
Время: 1 цикл
Алгоритм: если ((A[3-0])>9 или (AC)=1), то A[3-0]:=A[3-0]+6
если ((A[7-4])>9 или (C)=1), то A[7-4]:=A[7-4]+6
Примеры:

```
а)                          ;(A)=56H, (R3)=67H, (C)=l
   ADDC A,R3
   DA  A                    ;(A)=24H, (R3)=67H, (C)=l
б)                          ;(A)=30H, (C)=0
   ADD A, #99H
   DA A                     ;(A)=29, (C)=1
```

Примечание: Команда DA A не может просто преобразовать шестнадцатеричное значение в аккумуляторе в двоично-десятичное представление и не применяется, например, для десятичного вычитания.

Команда DEC <байт>

Команда "декремент" производит вычитание "1" из указанного операнда. Начальное значение 00H перейдет в 0FFH. Команда DEC не влияет на флаги. Этой командой допускается четыре режима адресации операнда:

1) к аккумулятору
2) регистровый
3) прямой
4) косвенно-регистровый

1) Ассемблер: DEC A
Код: 00010100
Время: 1 цикл
Алгоритм: (A):=(A)-1
Пример:

```
                ;(A)=11H, (C)=1, (AC)=1
DEC A    ;(A)=10H, (C)=1, (AC)=1
```

2) Ассемблер: DEC Rn ; где n=0-7
Код: 00011 rrr , где rrr=000-111
Время: 1 цикл
Алгоритм: (Rn):=(Rn)-l
Пример:

```
                ;(R1)=7FH,
                ;(ОЗУ[7F])=40H, (ОЗУ[7E])=00H
```

```
                DEC @R1
                DEC Rl
                DEC @R1 ;(R1)=7EH,
                        ;(ОЗУ[7F])=3FH, (ОЗУ[7E])=FFH
```

3) Ассемблер: DEC <direct>
Код: 00010101 direct address
Время: 1 цикл
Алгоритм: (direct):=(direct)-l
Пример:
```
                ;(SCON)=A0H, (C)=1, (AC)=1
DEC SCON ;(SCON)=9FH, (C)=1, (AC)=1
```
5) Ассемблер: DEC @Ri ; где i=0,l
Код: 0001011i
Время: 1 цикл
Алгоритм: ((Ri)):=((Ri)-l)
Пример:
```
                ;(R1)=7FH,
                ;(ОЗУ[7F])=40H, (ОЗУ[7E])=00H
 DEC @R1
 DEC Rl
 DEC @R1  ;(R1)=7EH,
                ;(ОЗУ[7F])=3FH, (ОЗУ[7E])=FFH
```

Примечание: Если эта команда используется для изменения информации на выходе порта, значение, используемое как исходные данные, считывается из "защелки" порта, а не с выводов БИС.

Команда DIV AB

Команда "деление" делит 8-битовое целое без знака из аккумулятора А на 8-битовое целое без знака в регистре В. Аккумулятору присваивается целая часть частного (старшие разряды), а регистру В — остаток. Флаги переноса (С) и переполнения (OV) сбрасываются. Если (А) < (В), то флаг дополнительного переноса (АС) не сбрасывается. Флаг переноса сбрасывается в любом случае.

Ассемблер: DIV AB
Код: 10000100
Время: 4 цикла
Алгоритм: (A): = ((A)/(B))[15-8], (B):=((A)/(B))[7-0]

Пример: Пусть аккумулятор содержит число 251 (0FBH или 11111011B), а регистр В - число 18 (12H или 00010010B). После выполнения команды DIV AB в аккумуляторе будет число 13 (0DH или 00001101B), а в регистре В - число 17 (11H или 00010001B), т.к. 251=(13*18)+17. Флаги С и OV будут сброшены.

Примечание. Если В содержит 00, то после команды DIV содержимое аккумулятора А и регистра В будут не определены. Флаг переноса сбрасывается, а флаг переполнения устанавливается в "1".

Команда DJNZ <байт>, <смещение>

Команда «декремент и переход, если не равно нулю» выполняет вычитание "1" из указанной ячейки и осуществляет ветвление по вычисляемому адре-

су, если результат не равен нулю. Начальное значение 00H перейдет в 0FFH. Адрес перехода (ветвления) вычисляется сложением значения смещения (со знаком), указанного в последнем байте команды, с содержимым счетчика команд, увеличенным на длину команды DJNZ. На флаги эта команда не влияет и допускает следующие режимы адресации:

1) регистровый
2) прямой

1) Ассемблер: DJNZ Rn,<метка> ; где n=0-7
Код: 11011rrr rel8
Время: 2 цикла
Алгоритм: (PC):=(PC)+2, (Rn):=(Rn)-l,
если ((Rn)>0 или (Rn)<0), то (PC):=(PC)+<rel8>

```
Пример:                          ;(R2)=08H, (P1)=FFH (11111111B)
        LAB4: CPL P1.7
              DJNZ R2,LAB4 ;(R2)={07-00}
              ;Эта последовательность команд переключает P1.7
              ;восемь раз и приводит к появлению четырех
              ;импульсов на выводе БИС,
              ;соответствующем биту P1.7.
```

2) Ассемблер: DJNZ <direct>, <метка>
Код: 11010101 direct address rel8
Время: 2 цикла
Алгоритм: (PC):=(PC)+3,
(direct):=(direct)-1,
если ((direct)>0 или (direct)<0), то (PC):=(PC)+<re18>

```
Пример:                    ;(ОЗУ[40])=01H, (ОЗУ[50])=80H,
                           ;(ОЗУ[60])=25H
              DJNZ 40H, LAB1 ;(ОЗУ[40]):=00H
              DJNZ 50H, LAB2 ;(ОЗУ[50]):=7FH
              DJNZ 60H, LAB3 ;(ОЗУ[60]):=25H
      LAB1: CLR  A
      LAB2: DEC Rl           ;осуществился переход на
                             ;метку LAB2
```

Примечание. Если команда DJNZ используется для изменения выхода порта, значение, используемое как еперанд, считывается из "защелки" порта, а не с выводов БИС.

Команда INC <байт>

Команда "инкремент" выполняет прибавление "1" к указанной переменной и не влияет на флаги. Начальное значение 0FFH перейдет в 00H. Эта команда допускает четыре режима адресации:

1) к аккумулятору
2) регистровый
3) прямой
4) косвенно-регистровый

1) Ассемблер: INC A
Код: 00000100
Время: 1 цикл
Алгоритм: (A):=(A)+1
Пример: ;(A)=1FH, (AC)=0
INC A ;(A)=20H, (AC)=0
2) Ассемблер: INC Rn ; где n=0-7
Код: 00001 rrr , где rrr=000-111
Время: 1 цикл
Алгоритм: (Rn):=(Rn)+l
Пример: ;(R4)=FFH, (C)=0, (AC)=0
INC R4 ;(R4)=00H, (C)=0, (AC)=0
3) Ассемблер: INC <direct>
Код: 00000101 direct address
Время: 1 цикл
Алгоритм: (direct):=(direct)+l
Пример: ;(ОЗУ[43])=22H
INC 43H ;(ОЗУ[43])=23H
4) Ассемблер: INC @Ri ; где i=0,1
Код: 00000111 , где i=0,1
Время: 1 цикл
Алгоритм: ((Ri)):=((Ri))+l
Пример: ;(R1)=41H, (ОЗУ[41])=4FH, (AC)=0
INC @R1 ;(R1)=41H, (ОЗУ[41])=50H, (AC)=0

Примечание. При использовании команды INC для изменения содержимого порта, величина, используемая как операнд, считывается из "защелки" порта, а не с выводов БИС.

Команда INC DPTR

Команда "инкремент указателя данных" выполняет инкремент (прибавление "1") содержимого 16-битового указателя данных (DPTR). Прибавление "1" осуществляется к 16 битам, причем переполнение младшего байта указателя данных (DPL) из FFH в 00H приводит к инкременту старшего байта указателя данных (DPH). На флаги эта команда не влияет.

Ассемблер: INC DPTR
Код: 1 0 1 0 0 0 1 1
Время: 2 цикла
Алгоритм: (DPTR):=(DPTR)+1
Пример: ;(DPH)=12H, (DPL)=FEH
INC DPTR
INC DPTR ;(DPH)=13H, (DPL)=01H

Команда JB <bit>,<rel8>

Команда "переход, если бит установлен" выполняет переход по адресу ветвления, если указанный бит равен "1", в противном случае выполняется следующая команда. Адрес ветвления вычисляется с помощью прибавления относительного смещения со знаком в третьем байте команды (rel 8) к содержимому счетчика команд после прибавления к нему 3. Проверяемый бит не изменяется. Эта команда на флаги не влияет.

```
Ассемблер:  JB (bit),<метка>
Код:        00100000  bit address  rel8
Время:      2 цикла
Алгоритм:   (PC):=(PC)+3,
            если (bit)=1, то (PC):=(PC)+<rel8>
Пример:                          ;(A)=96H (10010110B)
            JB  ACC.2,LAB5 ;эта команда обеспечивает переход
                           ;на метку LAB5
      LAB5: INC A
```

Команда JBC <bit>,<rel8>

Команда "переход, если бит установлен и сброс этого бита", выполняет ветвление по вычисляемому адресу, если бит равен "1". В противном случае выполняется следующая за JBC команда. В любом случае указанный бит сбрасывается. Адрес перехода вычисляется сложением относительного смещения со знаком в третьем байте команды (rel8) и содержимого счетчика команд после прибавления к нему 3. Эта команда не влияет на флаги.

```
Ассемблер:  JBC (bit),<метка>
Код:        00010000  bit address  rel8
Время:      2 цикла
Алгоритм:   (PC):=(PC)+3,
            если (bit)=1, то (bit):=0, (PC):=(PC)+<rel8>
Пример:                          ;(A)=76H (0111 0110B)
            JBC ACC.3,LAB6  ;Перехода на LAB6 нет, т.к.
                            ;(A[3]) = 0
             JBC ACC.2,LAB7  ; (A)=72H (0111 0010B) и переход
                             ;на адрес, соответствующий
                             ;метке LAB7.
```

Примечание. Если эта команда используется для проверки бит порта, то значение, используемое как операнд, считывается из "защелки" порта, а не с вывода БИС.

Команда JC <rel8>

Команда "переход, если перенос установлен" выполняет ветвление по адресу, если флаг переноса равен "1", в противном случае выполняется следующая команда. Адрес ветвления вычисляется с помощью сложения относительного смещения со знаком во втором байте команды (rel8) и содержимого счетчика команд, после прибавления к нему 2. Эта команда на флаги не влияет.

Ассемблер: JC <метка>
Код: 01000000 rel8
Время: 2 цикла
Алгоритм: (PC):=(PC)+2,
если (C)=1, то (PC):=(PC)+<re18>
Пример: ;(C)=0
JC LAB8 ;нет перехода на метку LAB8
CPL C ;(C):=1
LAB8: JC LAB9 ;переход на метку LAB9, т.к. (C)=1
LAB9: NOP

Команда JMP @A+DPTR

Команда "косвенный переход" складывает 8-битовое содержимое аккумулятора без знака с 16-битовым указателем данных (DPTR) и загружает полученный результат в счетчик команд, содержимое которого является адресом для выборки следующей команды. 16-битовое сложение выполняется по модулю 2 , перенос из младших восьми бит распространяется на старшие биты программного счетчика. Содержимое аккумулятора и указателя данных не изменяется. Эта команда на флаги не влияет.

Ассемблер: JMP @A+DPTR
Код: 0 1 1 1 0 0 1 1
Время: 2 цикла
Алгоритм: (PC):=(A)[7-0]+(DPTR)[15-0]
Пример: ;(PC)=034EH, (A)=86H, (DPTR)=0329H
JMP @A+DPTR ;(PC)=03AFH, (A)=86H,
;(DPTR)=0329H

Команда JNB <bit>,<rel8>

Команда "переход, если бит не установлен" выполняет ветвление по адресу, если указанный бит равен "нулю", в противном случае выполняется следующая команда. Адрес ветвления вычисляется с помощью сложения относительного смещения со знаком в третьем байте команды (rel8) и содержимого счетчика команд после прибавления к нему 3. Проверяемый бит не изменяется. Эта команда на флаги не влияет.

Ассемблер: JNB (bit),<метка>
Код: 00110000 bit address rel8
Время: 2 цикла
Алгоритм: (PC):=(PC)+3, если (bit)=0, то (PC):=(PC)+<re18>
Пример: ;(P2)=CAH (11001010B),
;(A)=56H (0101 0110B)
JNB P2.3,LAB10 ;нет перехода на LAB10
JNB ACC.3,LAB11 ;переход на метку LAB11
LAB11: INC A

Команда JNC <rel8>

Команда "переход, если перенос не установлен" выполняет ветвление по адресу, если флаг переноса равен нулю, в противном случае выполняется следующая команда. Адрес ветвления вычисляется с помощью сложения относительного смещения со знаком во втором байте команды (rel8) и содержимого счетчика команд после прибавления к нему 2. Флаг переноса не изменяется. Эта команда на другие флаги не влияет.

```
Ассемблер:  JNC <метка>
Код:        01010000 rel8
Время:      2 цикла
Алгоритм:   (PC):=(PC)+2,
            если (C)=0,  то (PC):=(PC)+<re18>
Пример:                  ;(C)=1
            JNC  LAB12   ;нет перехода на LAB12
            CPL  C
      LAB12: JNC  LAB13  ;переход на метку LAB13
```

Команда JNZ <rel8>

Команда "переход, если содержимое аккумулятора не равно нулю" выполняет ветвление по адресу, если хотя бы один бит аккумулятора равен "1", в противном случае выполняется следующая команда. Адрес ветвления вычисляется сложением относительного смещения со знаком во втором байте команды (re8) и содержимого счетчика команд (PC) после прибавления к нему 2. Содержимое аккумулятора не изменяется. Эта команда на флаги не влияет.

```
Ассемблер:  JNZ  <метка>
Код:        01110000 rel8
Время:      2 цикла
Алгоритм:   (PC):=(PC)+2,
            если (A)≠0 то (PC):=(PC)+<re18>
Пример:                  ;(A)=00H
            JNZ  LAB14   ;нет перехода на LAB14
            INC  A
      LAB14: JNZ  LAB15  ;переход на метку LABI5
      LAB15: NOP
```

Команда JZ <rel8>

Команда "переход, если содержимое аккумулятора равно "0" выполняет ветвление по адресу, если все биты аккумулятора равны "0", в противном случае выполняется следующая команда. Адрес ветвления вычисляется сложением относительного смещения со знаком во втором байте команды (rel8) и содержимым счетчика команд после прибавления к нему 2. Содержимое аккумулятора не изменяется. Эта команда на флаги не влияет.

```
Ассемблер:  JZ <метка>
Код:        01100000 rel8
Время:      2 цикла
```

Алгоритм: (PC):=(PC)+2,
если (A)=0, то (PC):=(PC)+<rel8>
Пример: ;(A)=01H
JZ LAB16 ;нет перехода на LAB16
DEC A
LAB16: JZ LAB17 ;переход на метку LAB17
LAB17: CLR A

Команда LCALL <addr16>

Команда "длинный вызов" вызывает подпрограмму, находящуюся по указанному адресу. По команде LCALL к счетчику команд (PC) прибавляется 3 для получения адреса следующей команды и после этого полученный 16-битовый результат помещается в СТЕК (сначала следует младший байт, за ним — старший), а содержимое указателя СТЕКа (SP) увеличивается на 2. Затем старший и младший байты счетчика команд загружаются соответственно вторым и третьим байтами команды LCALL. Выполнение программы продолжается командой, находящейся по полученному адресу. Подпрограмма, следовательно, может начинаться в любом месте адресного пространства памяти программ объемом до 64 Кбайт. Эта команда на флаги не влияет.

Ассемблер: LCALL <метка>
Код: 00010010 addr[15-8] addr[7-0]
Время: 2 цикла
Алгоритм: (PC):=(PC)+3
(SP):=(SP)+1
((SP)): = (PC[7-0])
(SP):=(SP)+1
((SP)):=(PC[15-81)
(PC):=<addr[15-0]>
Пример: ;(SP)=07H,
;метке PRN соответствует адрес 1234H,
;по адресу 0126H находится команда LCALL
LCALL PRN ;(SP)=09H, (PC)=1234H,
;(ОЗУ[08])=26H, (ОЗУ[09])=01H

Команда LJMP <addr16>

Команда "длинный переход" выполняет безусловный переход по указанному адресу, загружая старший и младший байты счетчика команд (PC) соответственно вторым и третьим байтами, находящимися в коде команды. Адрес перехода, таким образом, может находиться по любому адресу пространства памяти программ в 64 Кбайт. Эта команда на флаги не влияет.

Ассемблер: LJMP <метка>
Код: 00000010 addr[15-8] addr[7-0]
Время: 2 цикла
Алгоритм: (PC):=<addr[15-0]>

Команда MOV <байт-назначения>,<байт-источника>

Команда "переслать переменную-байт" пересылает переменную-байт, указанную во втором операнде, в ячейку, указанную в первом операнде. Содержимое байта источника не изменяется. Эта команда на флаги и другие регистры не влияет. Команда "MOV" допускает 15 комбинаций адресации байта-источника и байта-назначения.

1) Ассемблер: MOV A,Rn ; где n=0-7
Код: 11101 rrr , где rrr=000-111
Время: 1 цикл
Алгоритм: (A):=(Rn)
Пример: ;(A)=FAH, (R4)=93H
MOV A,R4 ;(A)=93H, (R4)=93H

2) Ассемблер: MOV A,<direct>
Код: 11100101 direct address
Время: 1 цикл
Алгоритм: (A):=(direct)
Пример: ;(A)=93H, (ОЗУ[40])=10H, (R0)=40H
MOV A,40H ;(A)=10H, (ОЗУ[40])=10H, (R0)=40H

3) Ассемблер: MOV A,@Ri ; где i=0,1
Код: 11100111
Время: 1 цикл
Алгоритм: (A):=((Ri))
Пример: ;(A)=10H, (R0)=41H, (ОЗУ[41])=0CAH
MOV A,@R0 ;(A)=CAH, (R0)=41H, (ОЗУ[41])=0CAH

4) Ассемблер: MOV A,#data
Код: 01110100 #data8
Время: 1 цикл
Алгоритм: (A):=<#data8>
Пример: ;(A)=C9H (11001001B)
MOV A,#37H ;(A)=37H (00110111B)

5) Ассемблер: MOV Rn,A ; где n=0-7
Код: 11111 rrr , где rrr=000-111
Время: 1 цикл
Алгоритм: (Rn):=(A)
Пример: ;(A)=38H, (R0)=42H
MOV R0,A ;(A)=38H, (R0)=38H

6) Ассемблер: MOV Rn,<direct> ; где n=0-7
Код: 10101 rrr direct address
Время: 2 цикла
Алгоритм: (Rn):=(direct)
Пример: ;(R0)=39H, (P2)=0F2H
MOV R0,P2 ;(R0)=F2H

7) Ассемблер: MOV Rn,#data ; где n=0-7
Код: 0 1 1 1 1 rrr #data8

Время: 1 цикл
Алгоритм: (Rn):=<#data8>
Пример: ;(R0)=0F5H
MOV R0,#49H ;(R0)=49H
8) Ассемблер: MOV <direct>,A
Код: 11110101 direct address
Время: 1 цикл
Алгоритм: (direct):=(A)
Пример: ;(P0)=FFH, (A)=4BH
MOV P0,A ;(P0)=4BH, (A)=4BH
9) Ассемблер: MOV <direct>,Rn ; где n=0-7
Код: 10001 rrr direct address , где rrr=000-111
Время: 2 цикла
Алгоритм: (direct):=(Rn)
Пример: ;(PSW)=C2H, (R7)=57H
MOV PSW,R7 ;(PSW)=57H, (R7)=57H
10) Ассемблер: MOV <direct>,<direct>
Код: 10000101 direct address direct address
Время: 2 цикла
Алгоритм: (direct):=(direct)
Пример: ;(ОЗУ[45])=33H, (ОЗУ[48])=0DEH
MOV 48H,45H ;(ОЗУ[45])=33H, (ОЗУ[48])=33H
11) Ассемблер: MOV <direct>,@Ri ; где i=0,1
Код: 10000111 direct address
Время: 2 цикла
Алгоритм: (direct):=((Ri))
Пример: ;(R1)=49H, (ОЗУ[49])=0E3H
MOV 51H,@R1 ;(ОЗУ[51])=0E3H, (ОЗУ[49])=0E3H
12) Ассемблер: MOV <direct>,#data
Код: 01110101 direct address #data8
Время: 2 цикла
Алгоритм: (direct):=<#data8>
Пример: ;(ОЗУ[5F])=9BH
MOV 5FH,#07H ;(ОЗУ[5F])=07H
13) Ассемблер: MOV @Ri,A ; где i=0,1
Код: 1111011i ,где i=0,1
Время: 1 цикл
Алгоритм: ((Ri)):=(A)
Пример: ;(R1)=48H, (ОЗУ[48])=75H, (A)=0BDH
MOV @R1,A ;(0ЗУ[48])=0BDH
14) Ассемблер: MOV @Ri,<direct> ; где i=0,1
Код: 10100111 direct address
Время: 2 цикла
Алгоритм: ((Ri)):=(direct)

Пример: ;(R0)=51H, (ОЗУ[51])=0E3H, (P0)=0ACH
MOV @R0,P0 ;(ОЗУ[51])=0ACH

15) Ассемблер: MOV @Ri,#data ; где i=0,1
Код: 0111011i
Время: 1 цикл
Алгоритм: ((Ri)):=<#data8>
Пример: ;(ОЗУ[7E])=67H, (R1)=7EH
MOV @R1, #0A9H ;(ОЗУ[7E])=0A9H,(R1)=7EH

Команда MOV <бит назначения>, <бит источника»

Команда "переслать бит данных" битовую переменную, указанную во втором байте, копирует в разряд, который указан в первом операнде. Одним из операндов должен быть флаг переноса С, а другим может быть любой бит, к которому возможна прямая адресация.

1) Ассемблер: MOV C,<bit>
Код: 10100010 bit address
Время: 1 цикл
Алгоритм: (C):=(bit)
Пример: ;(C)=0, (P3)=D5H (11010101B)
MOV C,P3.0 ;C:=1
MOV C,P3.3 ;C:=0
MOV C,P3.7 ;C:=1

2) Ассемблер: MOV <bit>,C
Код: 10010010 bit address
Время: 2 цикла
Алгоритм: (bit):=(C)
Пример: ;(C)=1, (P0)=20H (00100000B)
MOV P0.1,C
MOV P0.2,C
MOV P0.3.C ;(C)=1, (P0)=2EH (00101110B)

Команда MOV DPTR,#datal6

Команда "загрузить указатель данных 16-битовой константой" загружает указатель данных DPTR 16-битовой константой, указанной во втором и третьем байтах команды. Второй байт команды загружается в старший байт указателя данных (DPH), а третий байт — в младший байт указателя данных (DPL). Эта команда на флаги не влияет и является единственной командой, которая одновременно загружает 16 бит данных.

Ассемблер: MOV DPTR,#<datal6>
Код: 10010000 #data[15-8] #data[7-0]
Время: 2 цикла
Алгоритм: (DPTR):=#data[15-0], причем DPH:=#data[15-8],
DPL:=#data[7-0]
Пример: ;(DPTR)=01FDH
MOV DPTR,#1234H ;(DPTR)=1234H,
;(DPH)=12H, (DPL)=34H

Команда MOVC A,@A+(<R16>)

<R16> — 16-разрядный регистр.

Команда "переслать байт из памяти программ" загружает аккумулятор байтом кода или константой из памяти программы. Адрес считываемого байта вычисляется как сумма 8-битового исходного содержимого аккумулятора без знака и содержимого 16-битового регистра. В качестве 16-битового регистра может быть:

1) указатель данных DPTR;
2) счетчик команд PC.

В случае, когда используется PC, он увеличивается до адреса следующей команды перед тем, как его содержимое складывается с содержимым аккумулятора. 16-битовое сложение выполняется так, что перенос из младших восьми бит может распространяться через старшие биты. Эта команда на флаги не влияет.

```
1) Ассемблер:  MOVC  A,@A+DPTR
Код:           10010011
Время:         2 цикла
Алгоритм:      (A):=((A)+(DPTR))
Пример:                              ;(A)=1BH, (DPTR)=1020H,
                                     ;(ПЗУ[103B])=48H,
               MOVC A,@A+DPTR  ;(A)=48H,  (DPTR)=1020H
2) Ассемблер:  MOVC A,@A+PC
Код:           10000011
Время:         2 цикла
Алгоритм:      (A): = ((A) + (PC)
Пример:                            ;(A)=FAH, (PC)=0289,
                                   ;(ПЗУ[0384])=9BH
               MOVC A,@A+PC  ;(A)=9BH, (PC)=028AH
```

Команда MOVX <байт приемника>, <байт источника>

Команда "переслать во внешнюю память (из внешней памяти) данных" пересылает данные между аккумулятором и байтом внешней памяти данных. Имеется два типа команд, которые отличаются тем, что обеспечивают 8-битовый или 16-битовый косвенный адрес к внешнему ОЗУ данных.

В первом случае содержимое R0 или R1 в текущем банке регистров обеспечивает 8-битовый адрес, который мультиплексируется с данными порта P0. Для расширения дешифрации ввода-вывода или адресации небольшого массива ОЗУ достаточно восьми бит адресации. Если применяются ОЗУ, немного больше чем 256 байт, то для фиксации старших битов адреса можно использовать любые другие выходы портов, которые переключаются командой, стоящей перед командой MOVX.

Во втором случае, при выполнении команды MOVX указатель данных DPTR генерирует 16-битовый адрес. Порт P2 выводит старшие восемь бит адреса (DPH), а порт P0 мультиплексирует младшие 8 бит адреса (DPL) с данными. Эта форма является эффективной при доступе к большим массивам данных

(до 64К байт), так как для установки портов вывода не требуется дополнительных команд.

```
1) Ассемблер:  MOVX  A,@Ri     ;   где i=0,1
Код:           11100011
Время:         2 цикла
Алгоритм:      (A):=((Ri))
Пример:                              ;(A)=32H, (R0)=83H, ячейка
                                     ;внешнего ОЗУ по адресу 83H
                                     ;содержит B6H
               MOVX A,@R0     ;(A)=B6H, (R0)=83H
2) Ассемблер:  MOVX A,@DPTR
Код:           11100000
Время:         2 цикла
Алгоритм:      (A):=((DPTR))
Пример:                                ;(A)=5CH, (DPTR)=1ABEH,
                                       ;ячейка внешнего ОЗУ по адресу
                                       ;1ABEH содержит 72H
               MOVX A,@DPTR   ;(A)=72H, (DPTR)=2ABEH
3) Ассемблер:  MOVX @Ri,A ; где i=0,1
Код:           11100111
Время:         2 цикла
Алгоритм:      ((Ri)):=(A)
Пример:                            ;(A)=95H, (R1)=FDH,
                                   ;ячейка внешнего ОЗУ с адресом
                                   ;FDH содержит 00
               MOVX @R1,A  ;(A)=95H, .(R1)=FDH,
                                   ;ячейка внешнего ОЗУ с адресом
                                   ;FDH содержит 95H
4) Ассемблер:  MOVX @DPTR,A
Код:           11110000
Время:         2 цикла
Алгоритм:      ((DPTR)):=(A)
Пример:                                 ;(A)=97H,   (DPTR)=1FFFH,
                                        ;ячейка внешнего ОЗУ с адресом
                                        ;1FFFH содержит 00
               MOVX    @DPTR,A     ;(A)=97H,
                                        ;ячейка внешнего ОЗУ с адресом
                                        ;1FFFH содержит 97H
```

Команда MUL AB

Команда "умножение" умножает 8-битовые целые числа без знака из аккумулятора и регистра B. Старший байт 16-битового произведения помещается в регистр B, а младший — в аккумулятор A. Если результат произведения больше, чем 0FFH (255), то устанавливается флаг переполнения (OV), в противном случае он сбрасывается. Флаг переноса всегда сбрасывается.

Ассемблер: MUL AB
Код: 10100100
Время: 4 цикла
Алгоритм: (A)[7-0]=(A)*(B),
(B)[15-8] = (A)*(B)

```
Пример: а)              ;(A)=50H (50H=80), (C)=l,
                        ;(B)=0A0H (A0H=160), (OV)=0
        MUL   AB        ;(A)=00H, (B)=32H, (C)=0, (OV)=1
      б)                ;(A)=2HH, (OV)=1, (B)=06H, (C)=1
        MUL   AB        ;(A)=0D8H, (B)=00H, (OV)=0, (C)=0
```

Команда NOP

Команда "нет операции" выполняет холостой ход и не влияет на регистры и флаги, кроме как на счетчик команд (PC).

Ассемблер: NOP
Код: 00000000
Время: 1 цикл
Алгоритм: (PC):=(PC)+1

Пример: Пусть требуется создать отрицательный выходной импульс на порте P1.6 длительностью 3 цикла. Это выполнит следующая последовательность команд:

```
CLR   P1.6        ;P1[6]:=0
NOP
NOP
NOP
SETB  P1.6        ;P1[6]:=1
```

Команда ORL <байт назначения>.<байт источника>

Команда "логическое "ИЛИ" для переменных-байтов" выполняет операцию логического "ИЛИ" над битами указанных переменных, записывая результат в байт назначения. Эта команда на флаги не влияет. Допускается шесть комбинаций режимов адресации:

— если байтом назначения является аккумулятор:

1) регистровый
2) прямой
3) косвенно-регистровый
4) непосредственный

— если байтом назначения является прямой адрес:

5) к аккумулятору
6) к константе

1) Ассемблер: ORL A,Rn ; где n=0-7
Код: 01001rrr , где rrr=000-111
Время: 1 цикл
Алгоритм: (A):=(A) OR (Rn),
где OR - операция логического "ИЛИ"

Пример: ;(A)=15H, (R5)=6CH
ORL A,R5 ;(A)=7DH, (R5)=6CH
2) Ассемблер: ORL A,<direct>
Код: 01000101 direct address
Время: 1 цикл
Алгоритм: (A):=(A) OR (direct)
Пример: ;(A)=84H, (PSW)=C2H
ORL A,PSW ;(A)=C6H, (PSW)=C2H
3) Ассемблер: ORL A,@Ri ; где i=0,1
Код: 0100011i
Время: 1 цикл
Алгоритм: (A):=(A) OR ((Ri))
Пример: ;(A)=52H, (R0)=6DH, (ОЗУ[6D])=49H
ORL A,@R0 ;(A)=5BH, (ОЗУ[6D])=49H
4) Ассемблер: ORL A,#<data>
Код: 01000100 #data8
Время: 1 цикл
Алгоритм: (A):=(A) OR #<data>
Пример: ;(A)=F0H
ORL A,#0AH ;(A)=FAH
5) Ассемблер: ORL (direct),A
Код: 01000010 direct address
Время: 1 цикл
Алгоритм: (direct):=(direct) OR (A)
Пример: ;(A)=34H, (IP)=23H
ORL IP,A ;(IP)=37H, (A)=34H
6) Ассемблер: ORL (direct),#<data>
Код: 01000011 direct address #data8
Время: 2 цикла
Алгоритм: (direct):=(direct) OR #<data>
Пример: ;(P1)=00H
ORL P1,#0C4H ;(P1)=11000100B (C4H)

Примечание. Если команда используется для работы с портом, величина, используемая в качестве исходных данных порта, считывается из "защелки" порта, а не с выводов БИС.

Команда ORL C,<бит источника>

Команда "логическое "ИЛИ" для переменных-битов" устанавливает флаг переноса C, если булева величина равна логической "1", в противном случае устанавливает флаг C в "0". Косая дробь ("/") перед операндом на языке ассемблера указывает на то, что в качестве операнда используется логическое отрицание значения адресуемого бита, но сам бит источника не изменяется. Эта команда на другие флаги не влияет.

1) Ассемблер: ORL C,<bit>
Код: 01110010 bit address
Время: 2 цикла
Алгоритм: (C): = (C) OR (bit.)
Пример: ;(C)=0, (P1)=53H (01010011B)
ORL C,P1.4 ;(C)=1, (P1)=53H (01010011B)
2) Ассемблер: ORL C,/<bit>
Код: 10100000 bit address
Время: 2 цикла
Алгоритм: (C):=(C) OR /(bit)
Пример: ;(C=0, (ОЗУ[25H])=39H (00111001B)
ORL C,/2AH ;(C=1, (ОЗУ[25H])=39H (00111001B)

Команда POP <direct>

Команда "чтение из стека" считывает содержимое ячейки, которая адресуется с помощью указателя стека, в прямо адресуемую ячейку ОЗУ, при этом указатель стека уменьшается на единицу.

Эта команда не воздействует на флаги и часто используется для чтения из стека промежуточных данных.

Ассемблер: POP <direct>
Код: 11010000 direct address
Время: 2 цикла
Алгоритм: (direct):=((SP)), (SP):=(SP)-1
Пример: ;(SP)=32H, (DPH)=01, (DPL)=ABH,
;(ОЗУ[32])=12H, (ОЗУ[31])=56H, ;(ОЗУ[30])=20H
POP DPH
POP DPL ;(SP)=30H, (DPH)=12H, (DPL)=56H,
;(ОЗУ[32])=12H, (ОЗУ[31))=56H
POP SP ;(SP)=20H, (ОЗУ[30])=20H

Команда PUSH <direct>

Команда "запись в стек" увеличивает указатель стека на единицу и после этого содержимое указанной прямо адресуемой переменной копируется в ячейку внутреннего ОЗУ, адресуемого с помощью указателя стека. На флаги эта команда не влияет и используется для записи промежуточных данных в стек.

Ассемблер: PUSH <direct>
Код: 11000000 direct address
Время: 2 цикла
Алгоритм: (SP):=(SP)+1, ((SP)):=(direct)
Пример: ;(SP)=09H, (DPTR)=1279H
PUSH DPL
PUSH DPH ;(SP)=0BH, (DPTR)=1279H,
;(ОЗУ[0A])=79H, (ОЗУ[0B])=12H,

Команда RET

Команда "возврат из подпрограммы" последовательно выгружает старший и младший байты счетчика команд из стека, уменьшая указатель стека на 2. Выполнение основной программы обычно продолжается по адресу команды, следующей за ACALL или LCALL. На флаги эта команда не влияет.

Ассемблер: RET
Код: 00100010
Время: 2 цикла
Алгоритм: (PC)[15-8]:=((SP)), (SP):=(SP)-1,
(PC)[7-0]:=((SP)), (SP):=(SP)-1
Пример: ;(SP)=0DH, (ОЗУ[0C])=93H, (ОЗУ[00])=02H
RET ;(SP)=0BH, (PC)=0293H

Команда RETI

Команда "возврат из прерывания" выгружает старший и младший байты счетчика команд из стека и устанавливает "логику прерываний", разрешая прием других прерываний с уровнем приоритета, равным уровню приоритета только что обработанного прерывания. Указатель стека уменьшается на 2. Слово состояния программы (PSW) не восстанавливается автоматически. Выполнение основной программы продолжается с команды, следующей за командой, на которой произошел переход к обнаружению запроса на прерывание. Если при выполнении команды RETI обнаружено прерывание с таким же или меньшим уровнем приоритета, то одна команда основной программы успевает выполниться до обработки такого прерывания.

Ассемблер: RETI
Код: 00110010
Время: 2 цикла
Алгоритм: (PC)[15-8]:=((SP)), (SP):=(SP)-1,
(PC)[7-0]:=((SP)), (SP):=(SP)-1
Пример: ;(SP)=0BH, (ОЗУ[0A])=2AH, (ОЗУ[0B])=03H,
;(PC)=XXXXH, где X=0 - FH
RETI ;(SP)=09H, (PC)=032AH

Команда RL A

Команда "сдвиг содержимого аккумулятора влево", сдвигает восемь бит аккумулятора на один бит влево, бит 7 засылается на место бита 0. На флаги эта команда не влияет.

Ассемблер: RL A
Код: 00100011
Время: 1 цикл
Алгоритм: (A[N+1]):=(A[N]), где N=0-6
(A[0]):=(A[7])
Пример: ;(A)=0D5H (11010101B), (C)=0
RL A ;(A)=0ABH (10101011B), (C)=0

Команда RLC A

Команда "сдвиг содержимого аккумулятора влево через флаг переноса" сдвигает восемь бит аккумулятора и флаг переноса влево на один бит. Содержимое флага переноса помещается на место бита 0 аккумулятора, а содержимое бита 7 аккумулятора переписывается в флаг переноса. На другие флаги эта команда не влияет.

Ассемблер: RLC A
Код: 00110011
Время: 1 цикл
Алгоритм: (A[N+1]):=(A[N]), где N=0-6,
(A[0]):=(C), (C): = (A[7])
Пример: ;(A)=56H (01010110B), (C)=1
RLC A ;(A)=0ADH (10101101B), (C)=0

Команда RR A

Команда "сдвиг содержимого аккумулятора вправо" сдвигает вправо на один бит все восемь бит аккумулятора. Содержимое бита 0 помещается на место бита 7. На флаги эта команда не влияет.

Ассемблер: RR A
Код: 00000011
Время: 1 цикл
Алгоритм: (A[N]):=(A[N+1]), где N=0-6
(A[7]):=(A[0])
Пример: ;(A)=0D6H (11010110B), (C)=1
RR A ;(A)=6BH (01101011B), (C)=1

Команда RRC A

Команда "сдвиг содержимого аккумулятора вправо через флаг переноса" сдвигает восемь бит аккумулятора и флаг переноса на один бит вправо. Бит 0 перемещается в флаг переноса, а исходное содержимое флага переноса помещается в бит 7. На другие флаги эта команда не влияет.

Ассемблер: RRC A
Код: 00010011
Время: 1 цикл
Алгоритм: (A[N]):=(A[N+1]), где N=0-6 (A[7]):=(C), (C):=(A[0])
Пример: ;(A)=95H (10010101B), (C)=0
RRC A ;(A)=4AH (01001010B), (C)=1

Команда SETB <бит>

Команда "установить бит" устанавливает указанный бит в "1". Адресуется:

1) к флагу переноса (C);
2) к биту с прямой адресацией.

1) Ассемблер: SETB C
Код: 11010011
Время: 1 цикл
Алгоритм: (C):=1

```
Пример:                 ;(C)=0.
                SETB C  ;(C)=1
2) Ассемблер:   SETB (bit)
Код:            11010010  bit address
Время:          1 цикл
Алгоритм:       (bit):=1
Пример:                         ;(P2)=38H  (00111000B)
                SETB    P2.0
                SETB    P2.7
                                ;(P2)=B9H  (10111001B)
```

Команда SJMP <метка>

Команда "короткий переход" выполняет безусловное ветвление в программе по указанному адресу. Адрес ветвления вычисляется сложением смещения со знаком во втором байте команды с содержимым счетчика команд после прибавления к нему 2.

Таким образом, адрес перехода должен находиться в диапазоне от 128 байт, предшествующих команде, до 127 байт, следующих за ней.

```
Ассемблер:  SJMP  <метка>
Код:        10000000  rel8
Время:      2 цикла
Алгоритм:   (PC):=(PC)+2,      (PC):=(PC)+(re18)
Пример:                 ;(PC)=0418H,
                        ;метка MET1 соответствует адресу 039AK
            SJMP MET1  ;(PC)=039AH,   где  (rel8)=80H=-128
            SJMP MET2   ;(PC)=041AH,   где метка MET2 соответ-
                        ;ствует адресу 041AH,
                        ;(rel8)=7DH=+125
```

Команда SUBB A,<байт источника>

Команда "вычитание с заемом" вычитает указанную переменную вместе с флагом переноса из содержимого аккумулятора, засылая результат в аккумулятор. Эта команда устанавливает флаг переноса (заема), если при вычитании для бита 7 необходим заем, в противном случае флаг переноса сбрасывается. Если флаг переноса установлен перед выполнением этой команды, то это указывает на то, что заем необходим при вычитании с увеличенной точностью на предыдущем шаге, поэтому флаг переноса вычитается из содержимого аккумулятора вместе с операндом источника. (AC) устанавливается, если заем необходим для бита 3 и сбрасывается в противном случае. Флаг переполнения (OV) устанавливается, если заем необходим для бита 6, но его нет для бита 7, или есть для бита 7, но нет для бита 6.

При вычитании целых чисел со знаком (OV) указывает на отрицательное число, которое получается при вычитании отрицательной величины из положительной, или положительное число, которое получается при вычитании положительного числа из отрицательного.

Операнд источника допускает четыре режима адресации:

1) регистровый
2) прямой
3) косвенно-регистровый
4) непосредственный (к константе).

1) Ассемблер: SUBB A,Rn ; где n=0-7
Код: 10011 rrr , где rrr=000-111
Время: 1 цикл
Алгоритм: (A):=(A)-(C)-(Rn);
(C):=X, (AC):=X, (OV):=X, где X=(0 или 1)
Пример: ;(A)=C9H, (R2)=54H, (C)=l
SUBB A,R2 ;(A)=74H, (R2)=54H, (C)=0,
;(AC)=0, (OV)=1

2) Ассемблер: SUBB A,<direct>
Код: 10010101 direct address
Время: 1 цикл
Алгоритм: (A):=(A)-(C)-(direct);
(C):=X, (AC):=X, (OV):=X, где X=(0 или 1)
Пример: ;(A)=97H, (B)=25H, (C)=0
SUBB A,B ;(A)=72H, (B)=25H, (C)=0,
;(AC)=0, (OV)=1

3) Ассемблер: SUBB A,@Ri ; где i=0,1
Код: 10010111
Время: 1 цикл
Алгоритм: (A): = (A)-(C)-((Ri));
(C):=X, (AC):=X, (OV):=X, где X=(0 или 1)
Пример: ;(A)=49H, (C)=1, (R0)=33H,
;(ОЗУ[33])=68H
SUBB A,@R0 ;(A)=E0H, (C)=1, (AC)=0, (OV)=0

4) Ассемблер: SUBB A,#data
Код: 10010100 #data8
Время: 1 цикл
Алгоритм: (A):=(A)-(C)-(#data8);
(C):=X, (AC):=X, (OV):=X, где X=(0 или 1)
Пример: ;(A)=0BEH, (C)=0
SUBB A,#3FH ;(A)=7FH, (C)=0, (AC)=1, (OV)=1

Команда SWAP A

Команда "обмен тетрадами внутри аккумулятора" осуществляет обмен между младшими четырьмя и старшими четырьмя битами аккумулятора (между старшей и младшей тетрадами).

Эта команда может рассматриваться так же, как команда четырехбитового циклического сдвига. На флаги эта команда не влияет.

Ассемблер: SWAP A
Код: 11000100
Время: 1 цикл
Алгоритм: (A[3-0]):=(A[7-4]), (A[7-4]):=(A[3-0])
Пример: ;(A)=0D7H (11010111B)
SWAP A ;(A)=7DH (01111101B)

Команда XCH A, <байт>

Команда "обмен содержимого аккумулятора с переменной-байтом" осуществляет обмен содержимого аккумулятора с содержимым источника, указанным в команде. Операнд источника может использовать следующие режимы адресации:

1) регистровый
2) прямой
3) косвенно-регистровый.

1) Ассемблер: XCH A,Rn ; где n=0-7
Код: 11001 rrr , где rrr=000-111
Время: 1 цикл
Алгоритм: (A):=(Rn), (Rn):=(A)
Пример: ;(A)=3CH , (R4)=15H
XCH A,R4 ;(A)=15H, (R4)=3CH

2) Ассемблер: XCH A,<dirrect>
Код: 11000101 direct address
Время: 1 цикл
Алгоритм: (A):=(direct), (direct):=(A)
Пример: ;(A)=0FEH, (P3)=0DAH
XCH A,P3 ;(A)=0DAH, (P3)=0FEH

3) Ассемблер: XCH A,@Ri , где i=0,1
Код: 11000111
Время: 1 цикл
Алгоритм: (A):=((Ri)), ((Ri)):=(A)
Пример: ;(R1)=39H, (ОЗУ[39])=44H, (A)=0BCH
XCH A,@R1 ;(ОЗУ[39])=0BCH, (A)=44H

Команда XCHD A,@Ri

Команда "обмен тетрадой" выполняет обмен младшей тетрады (биты 3—0) аккумулятора с содержимым младшей тетрады (биты 3—0) ячейки внутреннего ОЗУ, косвенная адресация к которой производится с помощью указанного регистра. На старшие биты (биты 7—4) эта команда не влияет (так же, как и на флаги).

Ассемблер: XCHD A,@Ri ; где i=0,1
Код: 1101011i
Время: 1 цикл
Алгоритм: (A[3-0]):=((Ri[3-0])),
((Ri[3-0])):=(A[3-0])
Пример: ;(R0)=55H, (A)=89H, (ОЗУ[55])=0A2H
XCHD A,@R0 ;(A)=82H, (ОЗУ[55])=0A9H

Команда XRL <байт назначения>,<байт источника>

Команда "логическое "ИСКЛЮЧАЮЩЕЕ ИЛИ" для переменных-байтов" выполняет операцию "ИСКЛЮЧАЮЩЕЕ ИЛИ" над битами указанных переменных, записывая результат в байт назначения. На флаги эта команда не влияет.

Допускается шесть режимов адресации:

— байтом назначения является аккумулятор:

1) регистровый
2) прямой
3) косвенно-регистровый
4) непосредственный

— байтом назначения является прямой адрес:

5) к аккумулятору
6) к константе.

1) Ассемблер: XRL A,Rn ; где n=0-7
Код: 01101 rrr , где rrr=000-111
Время: 1 цикл
Алгоритм: (A):=(A) XOR (Rn)
Пример: ;(A)=C3H, (R6)=0AAH
XRL A,R6 ;(A)=69H, (R6)=0AAH

2) Ассемблер: XRL A,<direct>
Код: 01100101 direct address
Время: 1 цикл
Алгоритм: (A):=(A) XOR (direct)
Пример: ;(A)=0FH, (P1)=0A6H
XRL A,P1 ;(A)=A9H, (P1)=0A6H

3) Ассемблер: XRL A,@Ri ; где i=0,1
Код: 0110011i
Время: 1 цикл
Алгоритм: (A):=(A) XOR ((Ri))
Пример: ;(A)=55H, R1=77H, (ОЗУ[77])=5AH
XRL A,@R1 ;(A)=0FH, (ОЗУ[77])=5AH

4) Ассемблер: XRL A,#data
Код: 01100100 #data8
Время: 1 цикл
Алгоритм: (A):=(A) XOR <data>
Пример: ;(A)=0C3H
XRL A,#0F5H ;(A)=36H

5) Ассемблер: XRL <direct>,A
Код: 01100010 direct address
Время: 1 цикл
Алгоритм: (direct):=(direct) XOR (A)
Пример: ;(A)=31H, (P1)=82H
XRL P1,A ;(A)=31H, (P1)=B3H

6) Ассемблер: XRL <direct>,#data
Код: 01100011 direct address #data8
Время: 2 цикла
Алгоритм: (direct):=(direct) XOR #data
Пример: ;(IP)=65H
XRL IP,#65H ;(IP)=00H

Примечание. Если эта команда используется для работы с портами, то значение, используемое в качестве операнда, считывается из "защелки" порта, а не с выводов БИС.

Методическое пособие

Безик Дмитрий Александрович

ИЗУЧЕНИЕ МИКРОПРОЦЕССОРНОЙ ТЕХНИКИ НА ПРИМЕРЕ МИКРОЭВМ СЕМЕЙСТВА МК 51

учебно-методическое пособие с методическими указаниями
к выполнению лабораторных работ

Редактор Лебедева Е.М.

Подписано к печати 04.06.2009 г. Формат 60х84. 1/16.
Бумага офсетная. Усл.п.л. 5.81. Тираж 100 экз. Изд.№ 1414.

Издательство Брянской государственной сельскохозяйственной академии
243365, Брянская обл., Выгоничский район, с. Кокино, Брянская ГСХА.