Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(ВлГУ)**

Ю.Е. МИШУЛИН

МИКРОПРОЦЕССОРНАЯ ТЕХНИКА

Лабораторный практикум

Владимир 2016

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»**

Ю.Е. МИШУЛИН

МИКРОПРОЦЕССОРНАЯ ТЕХНИКА

Лабораторный практикум

Владимир 2016

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С достижениями в области микропроцессоров ассоциируется, в первую очередь имя Intel. Действительно, начиная с выпуска в 1971 году первого микропроцессора i4004, корпорация Intel является мировым лидером в разработке и производстве этих изделий. Достаточно напомнить, что более 80% современных персональных компьютеров реализовано на базе микропроцессоров Intel.

Однако сфера интересов Intel не ограничивается микропроцессорами. Выпускается несколько семейств микроконтроллеров, множество типов специализированных микросхем для интерфейса с внешними и периферийными устройствами для реализации локальных вычислительных сетей и систем передачи данных, большая номенклатура микросхем памяти различной емкости, включая последние разработки электрически репрограммируемой FLASH-памяти.

Все задачи, решаемые микропроцессорными системами управления, можно разделить на два больших класса: управление событиями в реальном времени и управление потоками данных. Каждый класс задач предъявляет свои специфические требования к микропроцессору или микроконтроллеру, что отражается в наборе функций, реализуемых в микропроцессоре, и в системе команд.

К первому классу относятся задачи, требующие быстрой реакции микропроцессорной системы на изменение внешних условий (на срабатывание технологических датчиков, изменение параметров и т.д.). В основном это системы управления приводами, энергетическими установками, роботами, а также системы распределенной автоматизации. Эти задачи требуют применения микроконтроллеров с большим объемом интегрированной на кристалл периферии, включая реализацию на кристалле памяти

программ, памяти данных и устройств ввода-вывода, что сокращает аппаратные затраты и удешевляет изделие со встроенной системой управления. Чаще всего в системах управления этого класса для реализации алгоритма управления требуется память относительно небольшого объема (до 32 Кбайт).

Ко второму классу задач относятся задачи, требующие быстрой обработки значительных объемов информации, например, в микропроцессорных системах поддержки компьютерных сетей, в системах управления летательными аппаратами, в системах обработки видеоизображения. Встроенный процессор должен выполнять множество различных вычислительных операций, в том числе операций с плавающей запятой. Как правило, для решения таких задач требуется уже высокопроизводительный 32- или 64-разрядный процессор.

Микропроцессоры принесли корпорации Intel *мировую известность, но необходимо обратить внимание на другую ветвь микропроцессорной техники, предназначенную решать именно первый класс задач – встраиваемые микроконтроллеры (*embedded microcontrollers*).*

По количеству выпускаемых изделий этого класса, а значит, и по числу их пользователей, микроконтроллеры почти в 10 раз превосходят традиционные микропроцессоры. Современный микроконтроллер (МК) – это размещенная на кристалле сложная цифровая система, в состав которой входит 8-, 16- или 32- разрядный процессор, внутренняя память программ (до десятков килобайт), широкий набор интерфейсных и периферийных устройств: порты ввода-вывода, таймеры, аналого-цифровые преобразователи и другие. Эти однокристальные системы ориентированы, в первую очередь, на выполнение функций управления различными устройствами и процессами, почему и названы микроконтроллерами. Сфера их применения весьма широка — от современной бытовой техники до сложнейших систем управления технологическими процессами и робототехнических комплексов.

Основной базой для построения встроенных систем управления нижнего уровня являются именно однокристальные микроЭВМ и микроконтроллеры, могут применяться также и законченные одноплатные системы управления на их основе, выпускаемые рядом фирм в качестве кон-

троллеров-прототипов. Сегодня разработчики встроенных систем управления стоят перед непростым выбором: какое изделие и какой фирмы использовать в проекте. В настоящее время Intel производит несколько семейств микроконтроллеров: MCS-51/151, MCS-251, MCS-96/196/296, включающих более сотни моделей. Набор их характеристик позволяет удовлетворить запросы широкого круга производителей разнообразной электронной аппаратуры. Однако наряду с Intel производством данного семейства микроконтроллеров занимаются и другие известные фирмы, такие как Atmel, Philips Infineon Technologies (Siemens), Silicon Storage Technology, Dallas Semiconductor, Triscend и другие.

В приведенной таблице индекса популярности микроконтроллеров [8] *можно отметить, что микроконтроллеры семейства MCS-51 занимают одно из лидирующих положений как по архитектуре, так и по производителям (Atmel, Philips), поэтому лабораторный практикум ориентирован на изучение именно этого микроконтроллера.*

Таблица

Индекс популярности микроконтроллеров

По производителям, %		*По архитектурам, %*	
Atmel	*40,3*	*AVR*	*27,9*
Microchip	*19,3*	*PIC*	*19,1*
Texas	*11,6*	*MCS-51*	*17,3*
Philips	*10,8*	*ARM7*	*15,4*
Other	*6,1*	*MSP430*	*10*
Cygnal	*4,2*	*Other*	*3,5*
Fujitsu	*2,5*	*ARM9*	*3*
Samsung	*1,7*	*MB90*	*2,2*
Renesas	*0,9*	*M16C*	*0,9*
AD	*0,8*	*MB91*	*0,3*
ST	*0,8*	*68HC1x*	*0,3*
Dallas	*0,6*	*68HC0x*	*0,2*
Motorola	*0,5*		

Лабораторные занятия строятся таким образом, чтобы студенты, выполняя задания, отрабатывали навыки программирования и отлад-

ки программного обеспечения микроконтроллеров. Каждая лабораторная работа содержит краткие теоретические сведения, задания и контрольные вопросы.

Начинается практикум с изучения среды программирования PROVIEW32 с целью освоения функциональных возможностей программной среды и изучения порядка программирования и принципов работы микроконтроллеров семейства MCS-51. Вторая лабораторная работа посвящена изучению системы команд микроконтроллера, а именно форматов представления данных и команд, способов адресации операндов, команд операций с данными, признаков результата операций, команд операций управления.

Цель следующих работ направлена на получение навыков построения микропроцессорных систем управления техническими объектами и их программирования.

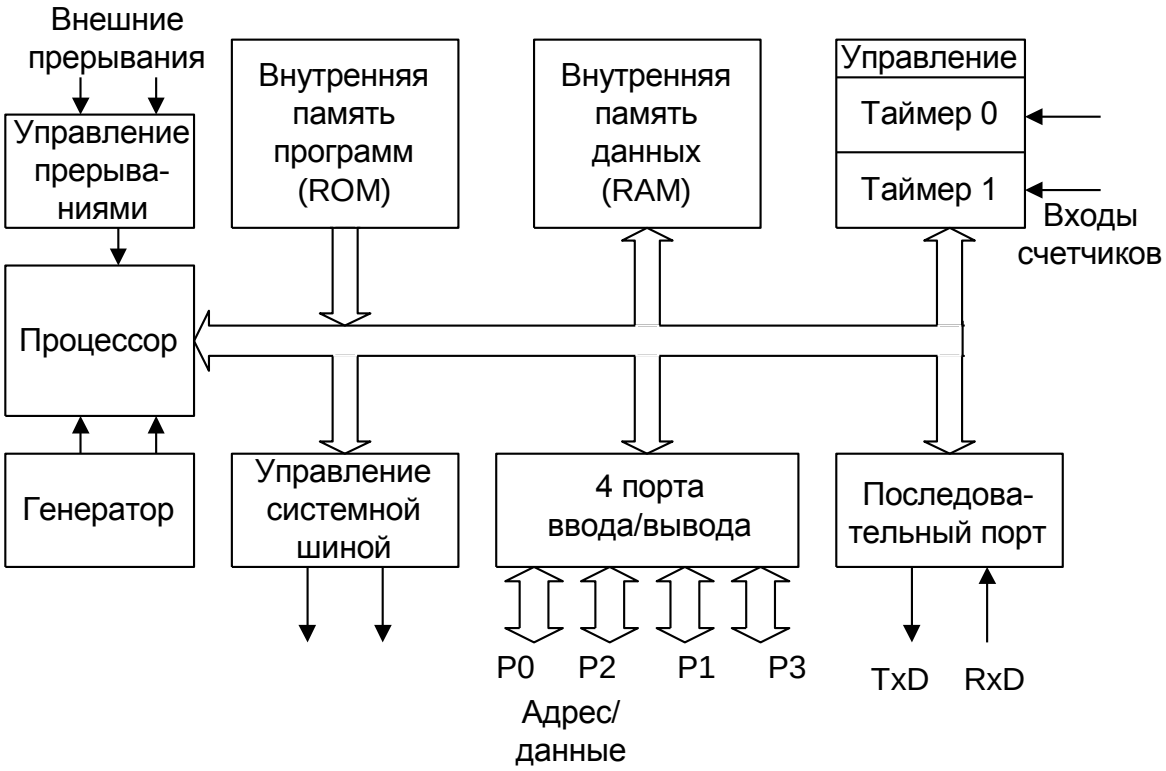
В состав объекта управления входят информационно-измерительные устройства (датчики) и исполнительные элементы как дискретного, так и непрерывного действия. Лабораторные работы 3…6 позволяют освоить приемы программирования микроконтроллера и построения устройств ввода и вывода дискретных и аналоговых сигналов, а также формирования управляющих сигналов сложной формы, например сигнала ШИМ.

При проектировании сложной технической системы используют многоуровневые мультипроцессорные системы управления. На верхнем уровне управления наряду с управляющим промышленным контроллером может использоваться промышленный компьютер. Для обеспечения взаимодействия контроллеров на нижнем уровне управления, а также связи с компьютером верхнего уровня используются различные последовательные каналы связи (интерфейсы), например RS-232, RS-485, USB и др. В последней работе рассматривается программирование и последовательного канала микроконтроллера для обеспечения связи с компьютером по интерфейсу RS-232 или RS-485.

Практикум предназначен для студентов технических специальностей, изучающих дисциплины «Микропроцессорная техника», «Микропроцессорные средства и системы».

1. СТРУКТУРА МИКРОКОНТРОЛЛЕРА СЕМЕЙСТВА MCS-51

В качестве примера рассмотрим базовую конфигурацию микроконтроллера 8051. Структурная схема микроконтроллера приведена на рис.



1.

Основу структурной схемы микроконтроллера образует внутренняя двунаправленная 8-битная шина, которая связывает между собой основные узлы и устройства: процессор, внутреннюю (резидентную) память, устройства управления и порты ввода/вывода. Для работы MCS-51 требуется один источник электропитания +5 В.

Кварцевый резонатор, подключаемый к внешним выводам корпуса микроконтроллера, управляет работой внутреннего генератора, который, в свою очередь, формирует сигналы синхронизации.

Устройство управления MCS-51 на основе сигналов синхронизации формирует машинный цикл фиксированной длительности, равный 12 периодам резонатора. Весь машинный цикл состоит из 12 фаз.

Большинство команд MCS-51 выполняется за один машинный цикл. Команды, оперирующие с 2-байтными словами, или связанные с обращением к внешней памяти, а также команды передачи управления выполняются за два машинных цикла. Только команды деления и умножения тре-

Рис. 1. Структурная схема микроконтроллера

буют четырех машинных циклов. На основе этих особенностей работы устройства управления MCS-51 рассчитывается время исполнения прикладных программ.

Все четыре порта микроконтроллера предназначены для ввода или вывода информации побайтно. Каждый порт содержит управляемые регистр-защелку, входной буфер и выходной драйвер.

Выходные драйверы портов P0 и P2, а также входной буфер порта P0 используются при обращении к внешней памяти (ВП). При этом через порт P0 в режиме временного мультиплексирования сначала выводится младший байт адреса ВП, а затем выдается или принимается байт данных. Через порт P2 выводится старший байт адреса в тех случаях, когда разрядность адреса равна 16 бит.

Все выводы порта P3 могут быть использованы для реализации альтернативных функций, перечисленных в табл. 1. Альтернативные функции могут быть задействованы путем записи 1 в соответствующие биты регистра-защелки (P3.0—P3.7) порта P3.

Таблица 1.

Альтернативные функции порта P3

Символ	Позиция	Имя и назначение
$\overline{RD}$	P3.7	Чтение. Активный сигнал низкого уровня формируется аппаратурно при обращении к ВПД
$\overline{WR}$	P3.6	Запись. Активный сигнал низкого уровня формируется аппаратурно при обращении к ВПД
T0	P3.5	Вход таймера/счетчика 1 или тест-вход
T1	P3.4	Вход таймера/счетчика 0 или тест-вход
$\overline{INT}1$	P3.3	Вход запроса прерывания 1. Воспринимается сигнал низкого уровня или срез
$\overline{INT}0$	P3.2	Выход запроса прерывания 0. Воспринимается сигнал низкого уровня или срез
TXD	P3.1	Выход передатчика последовательного порта в режиме УАПП. Выход синхронизации в режиме сдвигающего регистра
RXD	P3.0	Вход приемника последовательного порта в режиме УАПП. Ввод/вывод данных в режиме сдвигающего регистра

Порт P0 является двунаправленным, а порты P1, P2 и P3 — квазидвунаправленными. Каждая линия портов может быть использована независимо для ввода или вывода информации. Для того чтобы некоторая ли-

ния порта использовалась для ввода, в соответствующий разряд порта должна быть записана единица, которая закрывает МОП-транзистор выходной цепи.

По сигналу RST в регистры-защелки всех портов автоматически записываются единицы, настраивающие их тем самым на режим ввода. Все порты могут быть использованы для организации ввода/вывода информации по двунаправленным линиям передачи. Однако порты P0 и P2 не могут быть использованы для этой цели в случае, если система имеет внешнюю память, связь с которой организуется через общую разделяемую шину адреса/данных, работающую в режиме временного мультиплексирования.

Основным узлом микроконтроллера является процессор, в состав которого входит арифметико-логическое устройство (АЛУ) и устройство управления. 8-битное АЛУ может выполнять арифметические операции сложения, вычитания, умножения и деления, логические операции И, ИЛИ, исключающее ИЛИ, а также операции циклического сдвига, сброса, инвертирования и т.п. В АЛУ имеются регистры, предназначенные для временного хранения операндов, схема десятичной коррекции и схема формирования признаков операции.

Простейшая операция сложения используется в АЛУ для инкрементирования содержимого регистров, продвижения регистра-указателя данных и автоматического вычисления следующего адреса памяти программ. Простейшая операция вычитания используется в АЛУ для декрементирования регистров и сравнения переменных. Простейшие операции автоматически образуют «тандемы» для выполнения в АЛУ таких операций, как, например, инкрементирование 16-битных регистровых пар. В АЛУ реализуется механизм каскадного выполнения простейших операций для реализации сложных команд.

Важной особенностью арифметико-логического устройства является его способность оперировать не только байтами, но и битами. Отдельные программно-доступные биты могут быть установлены, сброшены, инвертированы, переданы, проверены и использованы в логических операциях.

Таким образом, АЛУ может оперировать четырьмя типами информационных объектов: булевыми (1 бит), цифровыми (4 бита), байтными (8 бит) и адресными (16 бит). В АЛУ выполняется 51 различная операция пересылки или преобразования этих данных. Так как используются 11 режимов адресации, то базовое число команд расширяется до 255.

Организация памяти

Память программ и память данных размещаются на кристалле и физически и логически разделены (гарвардская архитектура), имеют различные механизмы адресации, работают под управлением различных сигналов и выполняют разные функции. Память программ, так же как и память данных, может быть расширена до 64 Кбайт путем подключения внешних БИС.

Память программ (ПЗУ или СППЗУ) предназначена для хранения команд, констант, управляющих слов инициализации, таблиц перекодировки входных и выходных переменных и т.п. Память программ (ПП) имеет 16-битную шину адреса, через которую обеспечивается доступ из счетчика команд или из регистра-указателя данных. Последний выполняет функции базового регистра при косвенных переходах по программе или используется в командах, оперирующих с таблицами.

Организация ПП показана на рис. 2. Полный объём адресуемой памяти составляет 64 Кбайта. Объём внутренней памяти программ может составлять от 1 Кбайт до 32 Кбайт. На рисунке показано распределение адресов для базовой конфигурации контроллера 80С51, у которого объём составляет 4 Кбайта. Память программ может быть целиком внешней (сигнал EA=0), либо при обращении к младшим 4К адресам код извлекается из ячеек внутренней памяти, а содержимое старших 60К берется из внешней памяти (сигнал EA=1).

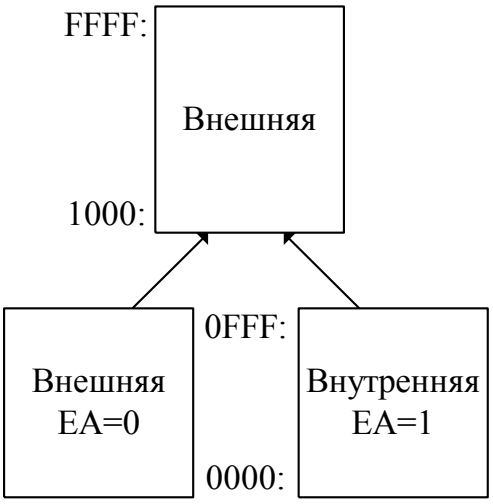


Рис.2. Память программ

Для обращения к памяти программ (внутренней и внешней) используются две инструкции: MOVC A,@A+DPTR и MOVC A,@A+PC. Они выполняют загрузку аккумулятора из памяти программ. Адрес байта памяти определяется сложением содержимого аккумулятора с содержимым 16-битного регистра указателя данных DPTR или программного счетчика PC. В зависимости от значения сигнала EA данные будут прочитаны или из внешней памяти (EA=0), или из внутренней (EA=1).

Память данных (ОЗУ) предназначена для хранения переменных в процессе выполнения прикладной программы. Память данных (ПД) также может быть внутренней (резидентной РПД) и внешней (ВПД). При этом

внутренняя ПД обязательно существует у любого микроконтроллера. Организация ПД показана на рис.3. Полный объём внешней ПД составляет 64 Кбайта. Объём внутренней памяти программ может составлять от 64 байт до 256 байт. На рисунке показано распределение памяти для базовой конфигурации контроллера 80С51, у которого объём составляет 128 байт.

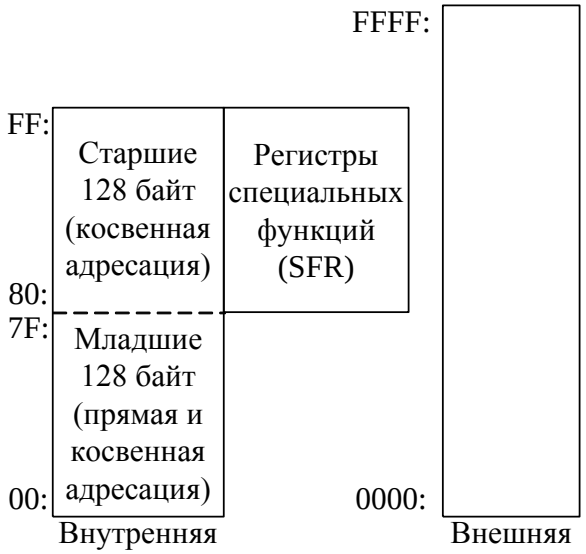


Рис.3. Память данных

Внутренняя память данных адресуется одним байтом и имеет емкость 128 байт. Старшие 128 байт имеются не у всех микроконтроллеров. Структура младших 128 байт памяти приведена на рис.4. По адресам 00h…1Fh расположены 4 банка регистров общего назначения R0…R7. При включении микроконтроллера по умолчанию выбирается нулевой банк. С 20h по 2Fh расположены бит-адресуемые ячейки памяти. Адреса этих ячеек 00h…7Fh. Байты памяти, имеющие адреса с 30h по 7Fh, составляют основную внутреннюю память данных.

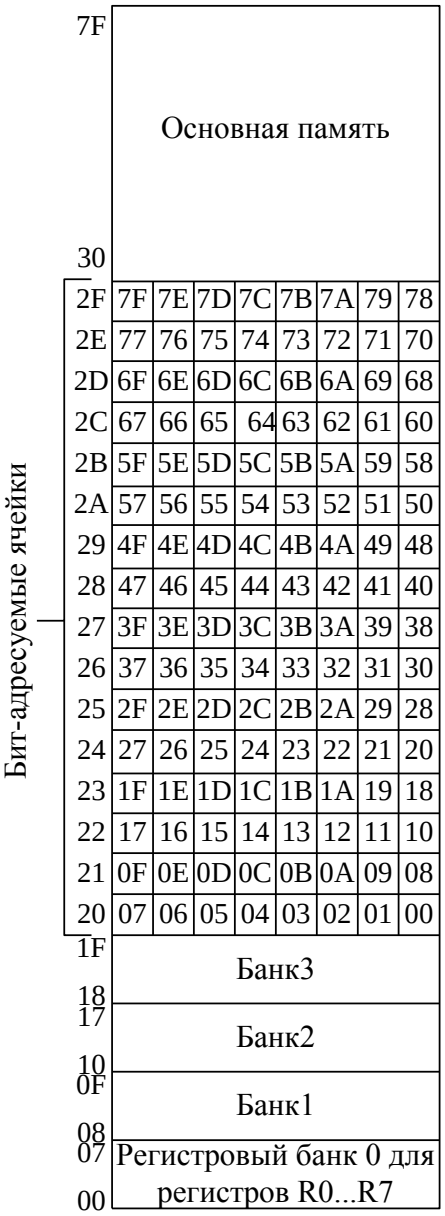


Рис. 4. Основная память данных

Кроме того, к адресному пространству внутренней ПД примыкают адреса регистров специальных функций (SFR), которые перечислены в табл. 2.

Доступ к ВПД обеспечивается управляющими сигналами WR и RD, которые формируются на линиях P3.7 и P3.6 при выполнении портом P3 альтернативных функций (см. табл. 1). Доступ к ВПД возможен с использованием 16-битного адреса:

MOVX A, @DPTR

или 8-битного адреса:

MOVX A, @ Ri.

Таблица 2

Блок регистров специальных функций

Символ	Наименование	Адрес
* B	Регистр-расширитель аккумулятора	F0H
* ACC	Аккумулятор	E0H
* PSW	Слово состояния программы	D0H
* IP	Регистр приоритетов	B8H
* P3	Порт 3	B0H
* IE	Регистр маски прерываний	A8H
* P2	Порт 2	A0H
SBUF	Буфер приемопередатчика	99H
* SCON	Регистр управления приемопередатчиком	98H
* P1	Порт 1	90H
TH1	Таймер 1 (старший байт)	8DH
TH0	Таймер 0 (старший байт)	8CH
TL1	Таймер 1 (младший байт)	8BH
TL0	Таймер 0 (младший байт)	8AH
TMOD	Регистр режима таймера/счетчика	89H
* TCON	Регистр управления/статуса таймера	88H
PCON	Регистр управления мощностью	87H
DPTR	Регистр-указатель данных (DPH)	83H
	(DPL)	82H
SP	Регистр-указатель стека	81H
* P0	Порт 0	80H

Примечание. Регистры, имена которых отмечены знаком *, допускают адресацию отдельных бит.

Аккумулятор и слово состояния программы (ССП)

Аккумулятор является источником операнда и местом фиксации результата при выполнении арифметических, логических операций и ряда операций передачи данных. Кроме того, только с использованием аккумулятора могут быть выполнены операции сдвигов, проверка на нуль, формирование флага паритета и т.п.

Формат слова состояния программы представлен в табл. 3. Наиболее «активным» флагом ССП является флаг переноса, который принимает участие и модифицируется в процессе выполнения множества операций, включая сложение, вычитание и сдвиги. Кроме того, флаг переноса (С) выполняет функции «булевого аккумулятора» в командах, манипулирующих с битами. Флаг переполнения (OV) фиксирует арифметическое переполнение при операциях над целыми числами со знаком и делает возможным использование арифметики в дополнительных кодах. АЛУ не управляет флагами селекции банка регистров (RSO, RS1 и их значение полностью определяется прикладной программой и используется для выбора одного из четырех регистровых банков.

Таблица 3

Формат слова состояния программы (ССП)

Символ	Позиция	Имя и назначение
C	PSW.7	Флаг переноса. Устанавливается и сбрасывается аппаратурными средствами или программой при выполнении арифметических и логических операций
AC	PSW.6	Флаг вспомогательного переноса. Устанавливается и сбрасывается только аппаратурными средствами при выполнении команд сложения и вычитания и сигнализирует о переносе или заеме в бите 3
F0	PSW.5	Флаг 0. Может быть установлен, сброшен или проверен программой как флаг, специфицируемый пользователем
RS1	PSW.4	Выбор банка регистров. Устанавливается и сбрасывается
RS0	PSW.3	программой для выбора рабочего банка регистров
OV	PSW.2	Флаг переполнения. Устанавливается и сбрасывается аппаратурно при выполнении арифметических операций
-	PSW.1	Не используется
P	PSW.0	Флаг паритета. Устанавливается и сбрасывается аппаратурно в каждом цикле команды и фиксирует нечетное/четное число единичных бит в аккумуляторе, то есть выполняет контроль по четности

Процессор в MCS-51 имеет в своей основе аккумулятор, однако он может выполнять множество команд и без участия аккумулятора. Например, данные могут быть переданы из любой ячейки РПД в любой регистр, любой регистр может быть загружен непосредственным операндом и т.д. Многие логические операции могут быть выполнены без участия аккумулятора. Кроме того, переменные могут быть инкрементированы, декрементированы и проверены без использования аккумулятора. Флаги и управляющие биты могут быть проверены и изменены аналогично.

Регистры-указатели

8-битный указатель стека (SP) может адресовать любую область РПД. Его содержимое инкрементируется прежде, чем данные будут запомнены в стеке в ходе выполнения команд PUSH и CALL. Содержимое SP декрементируется после выполнения команд POP и RET. Подобный способ адресации элементов стека называют прединкрементным/ постдекрементным. В процессе инициализации микроконтроллера после сигнала RST в SP автоматически загружается код 07H. Это значит, что если прикладная программа не переопределяет стек, то первый элемент данных в стеке будет располагаться в ячейке РПД с адресом 08H.

Двухбайтный регистр-указатель данных (DPTR) обычно используется для фиксации 16-битного адреса в операциях с обращением к внешней памяти. Командами MCS-51 регистр-указатель данных может быть использован или как 16-битный регистр, или как два независимых 8-битных регистра (DPH и DPL).

Начальное включение микроконтроллера

При начальном включении микроконтроллера необходимо сформировать единичный сигнал сброса путем подачи его на вход RST. Для уверенного сброса микроконтроллера этот сигнал должен быть удержан на входе RST по меньшей мере в течение двух машинных циклов (24 периода тактового генератора. Под воздействием сигнала RST сбрасывается содержимое регистров: PC, ACC, B, PSW, DPTR, TMOD, TCON, T/CO, T/C1, IE, IP и SCON, в регистре PCON сбрасывается только старший бит, в регистр-указатель стека загружается код 07H, а в порты P0-P3 - коды 0FFH. Состояние регистра SBUF неопределенное. Сигнал RST не воздействует на содержимое ячеек РПД.

После выполнения сигнала сброса в программный счетчик загружается адресом 0000H, который соответствует начальному пуску микроконтроллера при включении питания.

Исполняемая программа хранится в памяти программ, объём которой может достигать 64 Кбайта и может размещаться в любой области памяти. Но первая исполняемая команда должна располагаться по адресу 0000H.

Система прерываний

Система прерываний базового микроконтроллера включает 5 источников: 2 внешних прерывания, 2 прерывания от таймеров и прерывание от последовательного порта.

При поступлении сигнала от источника прерываний в программный счётчик (PC) загружается адрес вектора соответствующей подпрограммы обслуживания прерываний, а прежнее значение PC помещается в стек. Распределение адресов векторов прерываний в памяти программ показано в табл. 4. Таймер 2 присутствует у более развитых модификаций контроллеров, например 80C52.

Таблица 4

Адреса векторов прерываний

Адрес вектора	Обозначение	Источник прерываний
0003H	INT0	Внешнее прерывание 0
000BH	TF0	Прерывание от таймера 0
0013H	INT1	Внешнее прерывание 1
001BH	TF1	Прерывание от таймера 1
0023H	TI, RI	Прерывание от УАПП
002BH	TF2	Прерывание от таймера 2

Каждый из источников прерываний может быть индивидуально разрешён или запрещен установкой или сбросом соответствующего бита (флага) в SFR-регистре масок прерываний IE. Этот регистр также содержит бит, который может разрешать или запрещать все прерывания. Обозначение разрядов регистра IE показано в табл. 5, а их назначение указано ниже.

EA – управление всеми источниками прерываний одновременно. Если EA=0, то прерывания запрещены. Если EA=1, то прерывания могут быть разрешены индивидуальными разрешениями EX0, ET0, EX1, ET1, ES.

X – неиспользуемый разряд.

ES – управление прерыванием от последовательного порта. ES=1 – разрешение. ES=0 – запрещение.

ET1 – управление прерыванием от таймера/счетчика 1 (T/C1). ET1=1 – разрешение. ET1=0 – запрещение.

EX1 – управление прерыванием от внешнего источника INT1. EX1=1 – разрешение. EX1=0 – запрещение.

ET0 – управление прерыванием от таймера/счетчика 0 (T/C0). ET0=1 – разрешение. ET0=0 – запрещение.

EX0 – управление прерыванием от внешнего источника INT0. EX0=1 – разрешение. EX0=0 – запрещение.

Таблица 5

Регистр масок прерываний

Биты	IE.7	IE.6	IE.5	IE.4	IE.3	IE.2	IE.1	IE.0
Обозначение	EA	X	X	ES	ET1	EX1	ET0	EX0

Регистр приоритетов прерываний (IP) предназначен для установки уровня приоритета прерывания для каждого из пяти источников прерываний. Обозначение разрядов регистра IP показано в табл. 6, а их назначение указано ниже:

PX0 – установка уровня приоритета прерывания от внешнего источника INT0.

PT0 – установка уровня приоритета прерывания от T/C 0.

PX1 – установка уровня приоритета прерывания от внешнего источника INT1.

PT1 – установка уровня приоритета прерывания от T/C 1.

PS – установка уровня приоритета прерывания от последовательного порта.

X – неиспользуемый разряд.

Таблица 6

Регистр приоритетов прерываний

Биты	IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0
Обозначение	X	X	X	PS	PT1	PX1	PT0	PX0

Наличие в разряде IP единицы устанавливает для соответствующего источника высокий уровень приоритета, а наличие в разряде IP нуля – низкий уровень приоритета.

2. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Описание лабораторного стенда

Лабораторный стенд состоит из нескольких законченных функциональных модулей:

– модуль микроконтроллера;

– модуль дискретного вывода;

– модуль дискретного ввода;

– модуль аналогового вывода;

– модуль аналогового ввода;

– модуль индикации;

– интерфейсный модуль.

Функциональная схема лабораторного стенда приведена на рис.5.

Все модули имеют контактные гнезда входных и выходных сигналов, с помощью которых можно осуществлять соединение модулей между собой.

Модуль микроконтроллера состоит из панели, в которую устанавливается микроконтроллер. К его выводам подключен кварцевый резонатор, имеющий частоту 24 МГц, и схема начального сброса при включении питания. В качестве микроконтроллера можно использовать любой микроконтроллер семейства MCS-51 в 40 выводном корпусе, имеющий внутреннюю память программ. Порты P0, P1, P2, P3 микроконтроллера выведены на панель с помощью контактных гнезд.

Модуль дискретного вывода – это 8 светодиодов, аноды которых через токоограничивающие резисторы подключены к источнику питания 5В. Таким образом, если к катоду светодиода подключить «логический 0», то светодиод будет светиться, а при «логической 1» – не будет.

Модуль дискретного ввода представляет собой 8 двухпозиционных переключателей *SB*1…*SB*8. На выходных контактных гнездах переключателей в зависимости от их положения формируются сигналы «логического 0» и «логической 1».

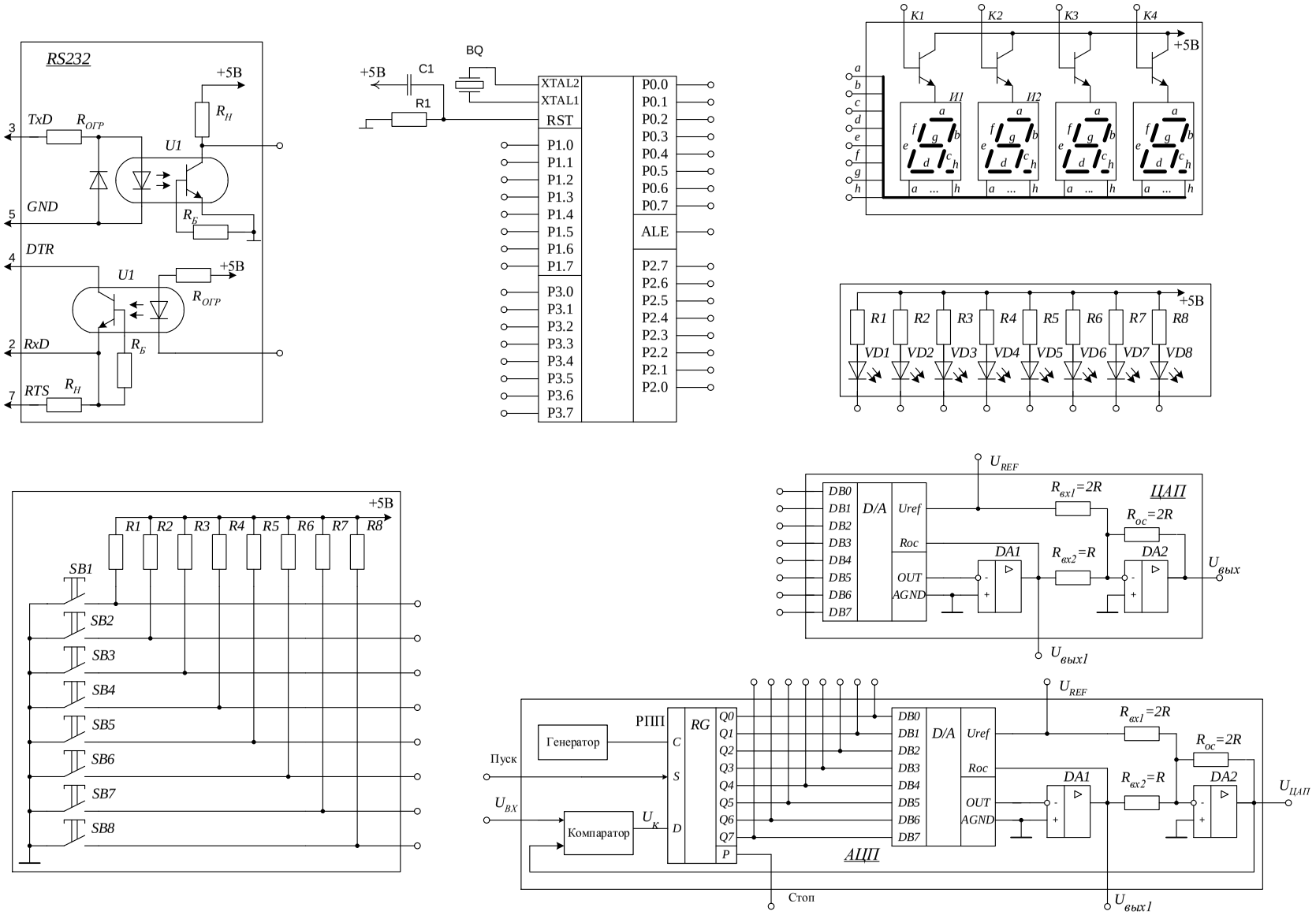


Рис. 5. Функциональная схема лабораторного стенда

Модуль аналогового вывода это цифро-аналоговый преобразователь (ЦАП). ЦАП имеет 8 разрядов и выполнен по схеме формирования двуполярного напряжения. Старший разряд – знаковый. Таким образом, на выходе ЦАП можно формировать напряжение в диапазоне от -10 до +10 В. Величина опорного напряжения равна 10,24 В. На контактные гнезда выведены 8 разрядов входных данных D0…D7 и два выходных напряжения с первого и второго усилителей.

Модуль аналогового ввода выполнен на основе аналого-цифрового преобразователя последовательного приближения (поразрядного уравновешивания). АЦП состоит из ЦАП и регистра последовательного приближения (РПП). Запуск АЦП осуществляется подачей сигнала «Пуск». По окончании преобразования формируется сигнал «Стоп» (конец преобразования). На контактные гнезда выведены входной сигнал АЦП, 8 разрядов данных D0…D7, сигналы «Пуск», «Стоп», выходное напряжение ЦАП.

Модуль индикации содержит 4 семисегментных индикатора. Одноименные сегменты всех индикаторов объединены и выведены на контактные гнезда. Аноды индикаторов через управляемые транзисторные ключи подключаются к источнику питания. Для вывода информации на все индикаторы необходимо использовать динамическую индикацию.

Интерфейсный модуль представляет собой интерфейс RS-232. Он состоит из двух транзисторных ключей с оптронной развязкой. Один ключ настроен на передачу данных от контроллера, другой – на прием. На контактные гнезда выведены сигналы TxD и RxD, которые подключаются к соответствующим выводам порта P3 микроконтроллера. Через стандартный 9-ти контактный разъем интерфейс можно подключать к COM-порту компьютера.

При выполнении лабораторной работы с использованием стенда необходимо выполнить соответствующие соединения модулей между собой и установить в панель микроконтроллер, с записанной в него программой. Программа разрабатывается с использованием программной среды «PROVIEW32 Franklin Software Inc», подробное описание которой и приемы программирования рассматриваются в первой лабораторной работе. После отладки программы и получения HEX файла осуществляется программирование микроконтроллера, например, на программаторе ChipProg,

производимым фирмой Phyton. Внешний вид программатора показан на рис. 6. Программатор подключается к компьютеру через параллельный LPT-порт и имеет зажимную панель для установки микроконтроллера.

Рис.6. Программатор

После запуска исполняемого файла uprog32.exe или выбора в меню кнопки «Пуск» программы Phyton xxxProg на экране компьютера появляется окно программы, показанное на рис. 7. Окно программы содержит строку меню, панель инструментов, а также три основных поля: поле кодов программы Buffer #0, поле информации о микросхеме и поле программирования.

Перед началом работы необходимо выбрать

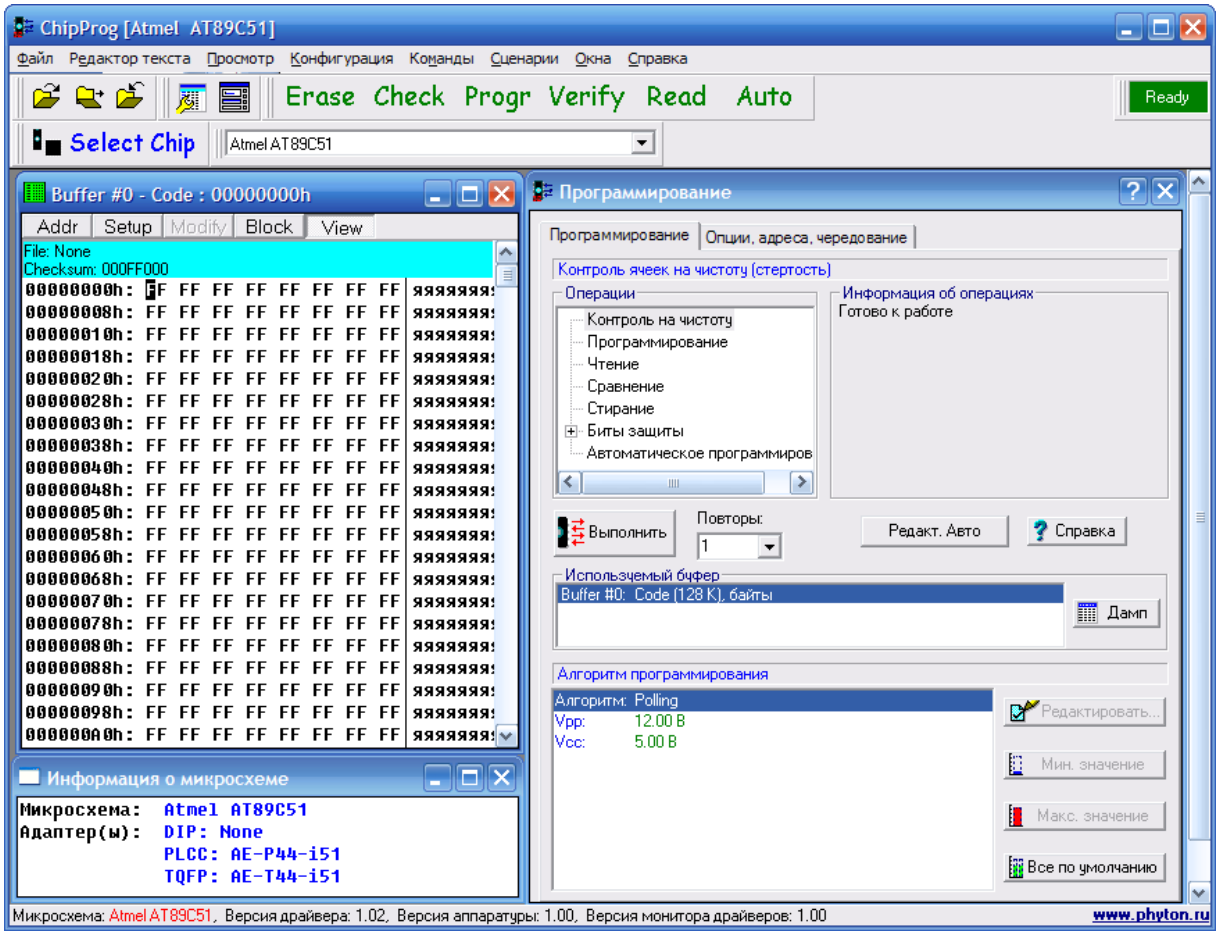


Рис.7. Окно программы ChipProg

тип используемого микроконтроллера. Для этого в меню «Конфигурация» выбрать команду «Выбор микросхемы». Операцию также можно вызвать нажатием клавиши F4.

В появившемся окне (рис.8) выбираем нужный микроконтроллер, например, AT89C51 фирмы Atmel. После этой процедуры в поле информации о микросхеме появляется название выбранного контроллера, а также сведения об используемых адаптерах для корпусов PLCC и TQFP микроконтроллера. При использовании DIP-корпуса адаптер не требуется и контроллер устанавливается непосредственно в панель программатора.

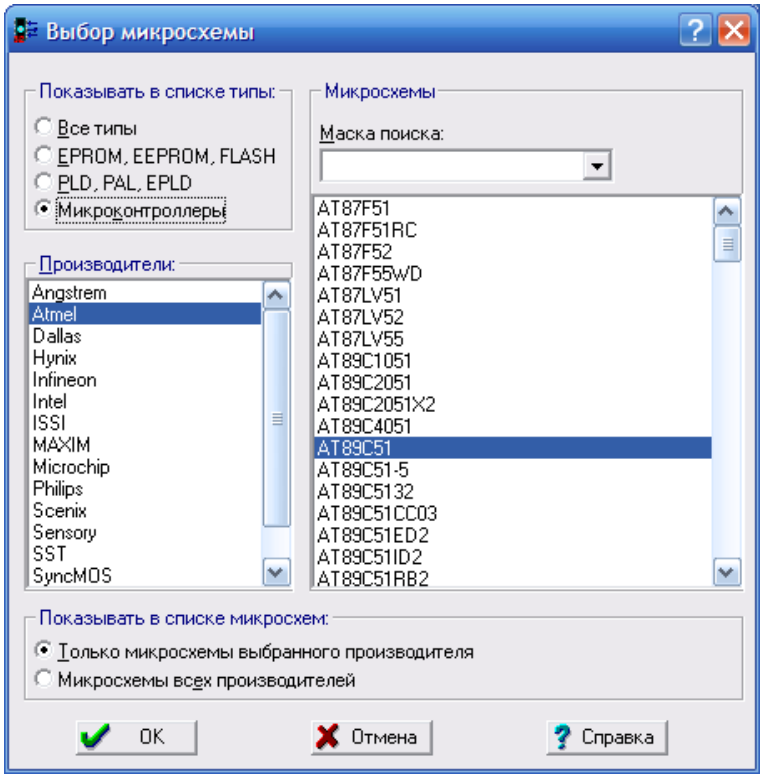


Рис.8. Окно выбора микросхемы

Следующим этапом работы с программатором является загрузка HEX-файла. Для этого в меню «Файл» выбрать команду «Загрузить…». Операцию также можно вызвать нажатием клавиши F2. В появившемся окне загрузки файла (рис.9) выбираем необходимый файл. Для поиска файла используется кнопка «Обзор…». При этом необходимо установить формат загружаемого файла Standard/Extended Intel HEX.

После загрузки в поле кодов программы Buffer #0 (рис.10) появляются значения кодов программы (дамп памяти).

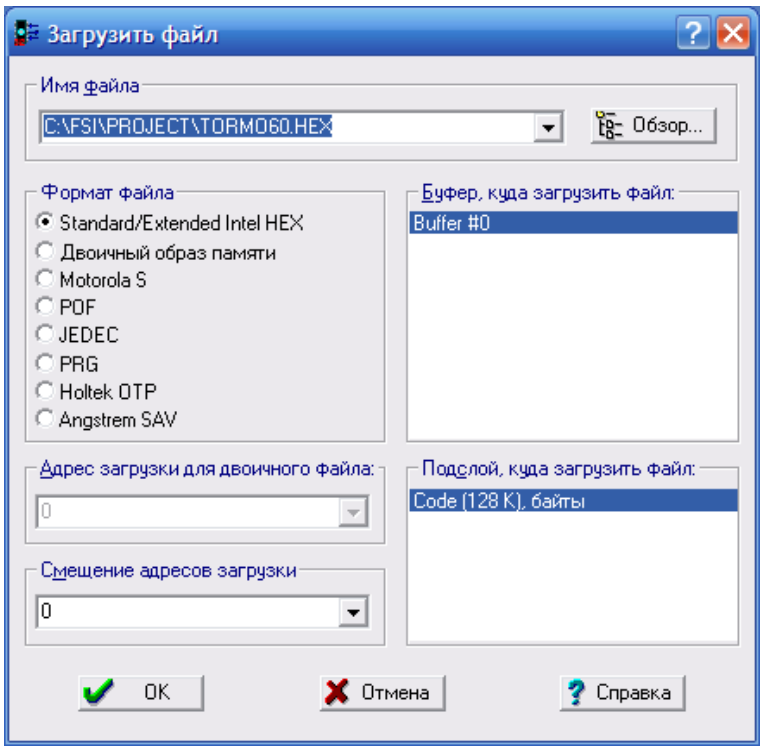


Рис. 9. Окно загрузки файла

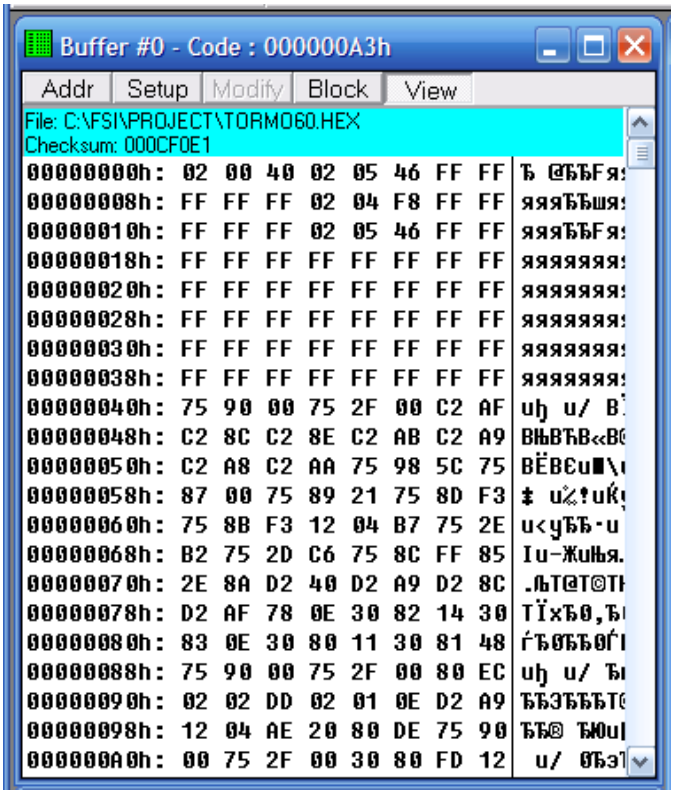


Рис. 10. Дамп памяти

Завершающий этап – программирование контроллера. Для этого в поле программирования (рис. 11) поочередно выполняются команды стирание, контроль на чистоту, программирование, сравнение. Более простым действием является выполнение команды автоматического программирования, по которой выполняются все указанные команды. Изменить набор команд при автоматическом программировании, а также последовательность их выполнения можно с помощью кнопки «Редакт.Авто».

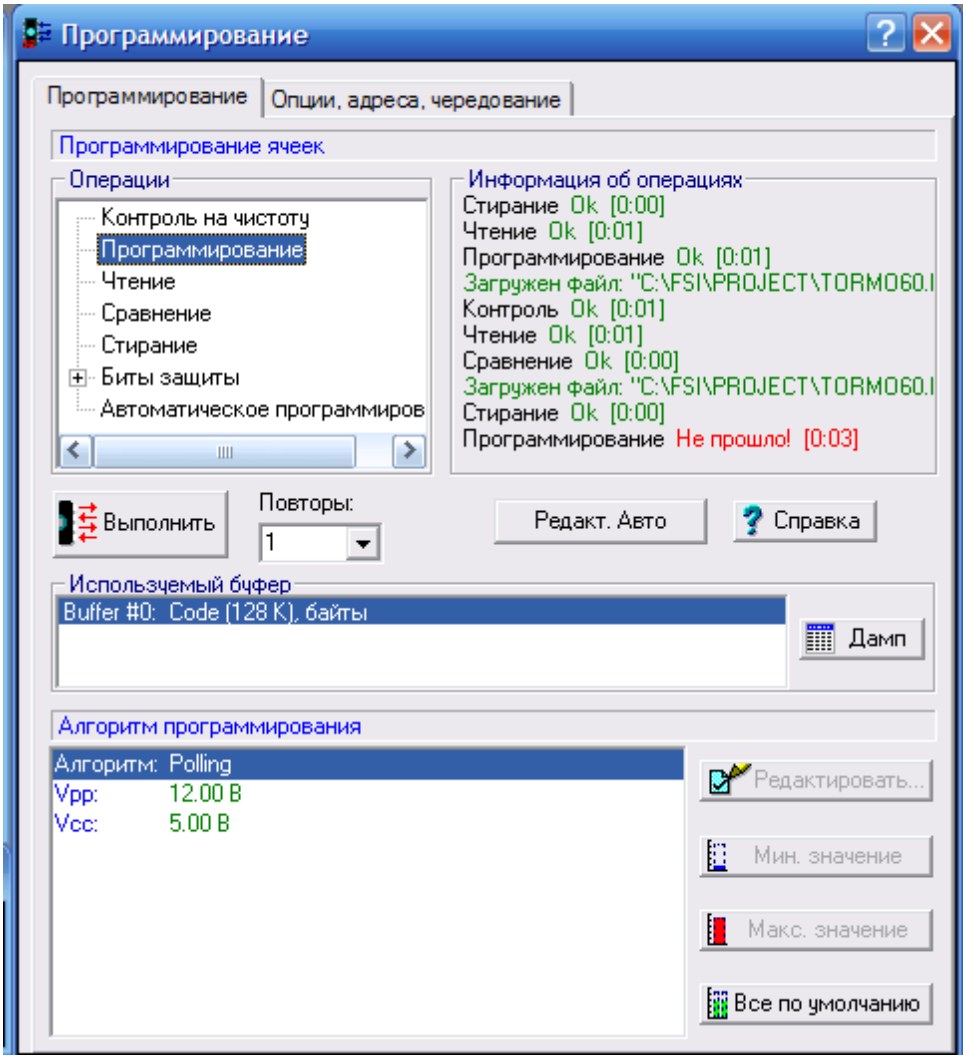


Рис. 11. Программирование микросхемы

ЛАБОРАТОРНАЯ РАБОТА №1
Изучение программной среды «PROVIEW32 Franklin Software Inc.»

Цель работы:

1. Изучить функциональные возможности программной среды «PROVIEW32 Franklin Software Inc.»

2. Изучить порядок программирования и принципы работы микроконтроллера AT89C51.

3. Приобрести навыки в подготовке программ с использованием программной среды «PROVIEW32 Franklin Software Inc.»

Общие правила написания программ.

Программирование контроллера осуществляется с помощью программной среды «PROVIEW32 Franklin Software Inc.» При написании программы можно использовать следующие директивы ассемблера:

EQU – создание символической переменной, которая при компиляции будет заменятся на число, ей соответствующее:

```
PWM  EQU   40h
```

ORG – директивное назначение текущего адреса компиляции:

```
ORG 0040h                ;программа будет расположена с адреса 40h
begin:  mov P0,#0FFh
        mov P1,#00011101b
```

(постфикс h обозначает шестнадцатеричное число, b – двоичное).

Символ решетки (#) перед любым операндом означает то, что этот операнд рассматривается как число, а не как адрес.

DB – компиляция последовательности байт:

```
TSQW1: DB 114,116,117,119,121,123,125,126,128,130,132,134,135,137,139
```

Примечание: Запрещается использовать в качестве символических имен, меток имена, зарезервированные программной средой. Это такие имена, как имена портов (P0…P3), имена регистров специальных функций (TMOD, TCON и др.), имена альтернативных функций порта P3 (TXD, INT и др.) и т.д. Полный перечень зарезервированных имен приведен в файле REG51.INC программной среды.

В начале программы указываются команды, определяющие тип используемого микроконтроллера:

```
$NOMOD51                 ; switch off 8051 controller defaults and use...
```

```
$INCLUDE (..\inc\reg52.inc)          ; ...another controller definition file (i.e. 8052).
```

Первая команда отключает используемый по умолчанию микроконтроллер 8051, а вторая подключает микроконтроллер 8052.

Следующая команда отключает генерацию листинга программы:

```
$NOLIST                              ; switch off generation of the listing file
```

Программу начинают следующими директивами и командами:

```
ORG 00h      ;прерывание от начального запуска (сброс по питанию)
JMP BEGIN    ; переход на начало программы
ORG 03h      ;прерывание от INT0
JMP INTR0    ;переход на подпрограмму обслуживания прерывания 0
ORG 0Bh      ;прерывание от таймера TMP0
JMP TIME0    ;переход на подпрограмму обслуживания прерывания от таймера 0
ORG 13h      ;прерывание от INT1
JMP INTR1    ;переход на подпрограмму обслуживания прерывания 1
ORG 1Bh      ;прерывание от таймера TMP1
JMP TIME1    ;переход на подпрограмму обслуживания прерывания от таймера 0
ORG 23h      ;прерывание от последовательного порта
JMP RS_PORT      ;переход на подпрограмму обслуживания прерывания от
                 ;последовательного порта
```

В поле векторов прерываний каждому вектору (определяется директивой ORG) ставится в соответствие символическое имя (метка) подпрограммы обслуживания. Для этого используется команда перехода JMP.

В области памяти программ можно хранить неизменяемые данные. Ниже приведен пример определения таблицы, для перевода десятичного числа в код семисегментного индикатора:

```
TABHG: DB 7Eh, 60h, 6Dh, 79h, 33h, 5Bh, 5Fh, 70h, 7Fh, 7Bh
```

К этой таблице можно обратиться следующими командами:

```
MOV A, #<число>          ;в аккумулятор записывается
                         ;десятичное число от 0 до 9
MOV DPTR, #TABHG         ;в указатель адреса записывается
                         ;адрес таблицы
MOVC A, @A+DPTR          ;в аккумулятор заносится код для
                         ;семисегментного индикатора
```

Для простой реализации мультизадачного режима можно определить подпрограммы установки одного из четырех банков РОН:

```
; Установка рабочего банка РОН
TASK4:    CLR RS0
          CLR RS1            ; адреса 0- 7H
```

```
            RET
TASK3:      CLR RS0
            SETB RS1            ; адреса 10- 17h
            RET
TASK2:      SETB RS0
            CLR RS1             ; адреса 8- 0Fh
            RET
TASK1:      SETB RS0
            SETB RS1            ; адреса 18- 1Fh
            RET
```

Начало функционирования программы может быть реализовано следующим образом:

```
BEGIN:      CLR IE              ; Запрет всех прерываний
            MOV PSW, #0         ; обнуление основных флагов
;----------------------
; обнуления внутренней памяти данных
            CLR A
            MOV R0, #1h         ; задание начального адреса памяти
BG2:        CJNE R0, #80h, BG1
            AJMP BG3
BG1:        MOV @R0, A
            INC R0
            AJMP BG2
;------------------------------
BG3:        MOV PCON, #0        ; отмена расширенных функций энергопотребления
            MOV IP, #00000011B ; установка приоритета INT0
                                ; установка приоритета первого таймера
                                ; низший приоритет остальных устройств
            MOV SCON, #01001100B ; установка режима УАПП
            MOV TMOD, 00100010B ; установка режимов таймеров
            MOV SP, #STACK      ; установка вершины стека
            MOV TH1, #0EBh      ; скорость обмена 2,4 Кбод
            SETB IT1            ; установить работу INT1 по срезу импульса.
            SETB IT0            ; установить работу INT0 по срезу импульса.
            SETB TR1            ; запустить первый таймер
```

А так же и другие команды, требуемые по логике работы алгоритма. Завершается этот фрагмент командой перехода на программу бесконечного цикла, рабочего цикла основной программы

```
            AJMP MAIN           ; переход на основной цикл
```

Далее описывается область подпрограмм. Для корректной работы

подпрограммы в многозадачной среде необходимо при выполнении каждой подпрограммы обработки прерывания сохранять основные регистры (естественно, если она их изменяет). Программный модуль обработки прерывания будет иметь вид:

```
MODUL:  PUSH PSW        ; сохранение основных регистров
        PUSH ACC
        PUSH B
        PUSH DPL
        PUSH DPH
        ACALL TASK1     ; выбор группы рабочих регистров
        . . .
        тело подпрограммы
        . . .
        POP DPH         ; восстановление основных регистров
        POP DPL
        POP B
        POP ACC
        POP PSW
        RETI
```

Примером подпрограммы не связанной с прерыванием может служить следующие программные модули:

```
; Перевод числа из двоичного в двоично-десятичное : R1R3=BIN
; Выход : R6 - десятки тысяч R5 - тысячи и сотни R4- десятки и единицы
BCD:    MOV FCH4, C
        ACALL CONV
        MOV A, R6
        MOV R4, A
        ACALL CONV
        MOV A, R6
        MOV R5, A
        MOV C, FCH4
        RET
CONV:   MOV R6, #0      ; Промежуточный результат
        MOV R7, #17
SB:     DJNZ R7, BC1
        RET
BC1:    CLR C
        MOV A, R3
        RLC A
```

```
                MOV R3, A
                MOV A, R1
                RLC A
                MOV R1, A
                MOV A, R6
                ADDC A, R6
                DA A
                MOV R6, A
                JNC SB
                MOV A, R3
                ADD A, #1
                MOV R3, A
                MOV A, R1
                ADDC A, #0
                MOV R1, A
                JMP SB          ; Сложение DPTR с R0.
NODIR:          MOV A, R0
                ADD A, DPL
                MOV DPL, A
                CLR A
                ADDC A, DPH
                MOV DPH, A
                RET
; Умножение 12 разрядных слов в R7/R6 на 4 разрядное в R5
; изменяет: R6, R7, R4, R3, A, B. Результат: R7/R6
; Работает только в TASK4 (нулевой банк)
TETMUL:         MOV A, R6
                ANL A, #0Fh
                MOV B, R5
                MUL AB
                MOV R4, A
                MOV A, R6
                ANL A, #0F0h
                SWAP A
                MOV B, R5
                MUL AB
                MOV R3, A
                ANL A, #0Fh
                SWAP A
                ADD A, R4
```

```
        PUSH PSW
        MOV R6, A
        MOV A, R7
        MOV B, R5
        MUL AB
        MOV R7, A
        MOV A, R3
        ANL A, #0F0h
        SWAP A
        POP PSW
        ADDC A, R7
        MOV R7, A
        RET
```

Также описывается основной цикл программы.

```
; Основной цикл программы.
MAIN:
        . . .
        тело основного цикла
        . . .
        AJMP MAIN
```

В случае если работа происходит только по прерываниям, тело основного цикла будет пустым – программа будет бесконечно выполнять команду AJMP MAIN, ожидая прерываний.

Таким образом, логически, программа может быть разбита на следующие блоки: таблица векторов прерываний; блок неизменных данных; команды начальной установки; подпрограммы обработки прерываний; подпрограммы; программу основного цикла.

После составления программы ее необходимо оттранслировать. Для этого в меню Project выбрать команду Translate. Операцию также можно вызвать одновременным нажатием клавиш Alt+F9. После выполнения операции открывается окно Message, в котором выводится информация об ошибках, например:

```
***ERROR #46 IN LINE 141 OF name_file.asm: UNDEFINED SIMBOL (PASS-2)`Pusk`
***ERROR #52 IN LINE 156 OF name_file.asm: TARGET OUT OF RANGE
```

Здесь указывается, что в 141 строке файла с именем name_file.asm произошла ошибка 46: неопределенный символ `Pusk`. Это означает что переменной с именем «Pusk» не было присвоено числовое значение дирек-

тивой EQU (например, Pusk EQU 40h). В строке 156 этого же файла – ошибка 52: переполнение ранга. В этой строке записана команда

```
jc bm3          ; переход на метку bm3, если есть перенос
```

Переход осуществляется внутри участка памяти, определяемой 8-разрядной величиной смещения (со знаком), указанной в последнем байте команды. В данном случае величина смещения превысила указанный диапазон.

После устранения ошибок создается HEX файл. Для этого в меню Options выбрать команду Project. В открывшемся окне развернуть строку L51 (щелчком левой кнопки мыши по значку +) и выбрать строку Linker. В правом поле в окне Misc установить галочку в позиции Intel Hex.

В меню Project выбрать команду Build all (или клавишами Shift+F9). В результате этих действий на диске будет создан файл с расширением HEX, который загружается в память программ микроконтроллера с помощью программатора.

Для отладки готовой программы с помощью программной среды «PROVIEW32 Franklin Software Inc.» используется встроенный симулятор. В меню Options выполняется команда Debug. В открывшемся окне выбираем Virtual Machine (Simulator), используемый тип микроконтроллера, например 80C51 и частоту задающего генератора (Crystal).

В меню Debug запускаем команду Start. Открываются окно Code, в котором содержатся коды выполняемой программы и отображается последовательность выполнения команд, и окно Main Registers. В этом окне выведены значения регистровой памяти, регистров специальных функций (SFR), портов микроконтроллера, которые изменяются в процессе выполнения программы. При необходимости их содержимое можно изменять вручную.

Для выполнения программы меню Debug запускаем команду Run. Наибольший интерес представляет режим пошагового выполнения программы с помощью команды Step (F7). В этом режиме удобно отслеживать ход выполнения программы и контролировать правильность ее исполнения по содержимому памяти данных, регистров специального назначения и стека, а также регистра флагов (PSW).

Имеются также другие возможности программной среды при отладке

программы, такие как назначение точек останова, отслеживания времени выполнения любого фрагмента программы, режимы анимации, трассировки и др.

Пример программы

```
$NOLIST                          ; switch off generation of the listing file
$NOMOD51                         ; switch off 8051 controller defaults and use...
$INCLUDE (..\inc\reg8s52.inc)    ; ...another controller definition file (i.e. 8052).
HUND        EQU 30h
TENONE      EQU 31h
NUMBER      EQU 32h
        ORG 00h
        JMP START
        ORG 03h
        JMP INT0
        ORG 0Bh
        JMP TIME0
        ORG 13h
        JMP INT1
        ORG 1Bh
        JMP TIME1
        ORG 23h
        JMP RS_PORT
;Выполняет преобразование 8-разрядного двоичного числа, хранящегося в
;регистре NUMBER в трехзначное число в двоично-десятичном упакованном формате.
;Число сотен размещено справа в поле переменной HUND,
;а числа десятков и единиц размещены в переменной TENONE.
INT0:       MOV A,NUMBER
            MOV B,#100
            DIV AB
            MOV HUND,A
            MOV A,#10
            XCH A,B
            DIV AB
            SWAP A
            ADD A,B
            MOV TENONE,A
            CLR    IE0
            RETI
TIME0:      RETI
```

```
INT1:       RETI
TIME1:      RETI
RS_PORT:    RETI

START:      MOV   IE,#00h           ; запрет всех прерываний
            MOV   IP,#00h           ; все прерывания низкого уровня
            MOV TH1,#0FBh           ; перезагружаемое число TH1
            MOV TL1,#0FBh
            MOV TMOD,#00100110B
;                                   Режим работы C/T0
;                                   Таймер/счетчик (1-счетчик)
;                                   1-управление от /INT0
;                                   Режим работы C/T1
;                                   Таймер/счетчик (1-счетчик)
;                                   1-управление от /INT1
            MOV SCON,#01010000B  ; УАПП-8 бит
;                                   Флаг прерывания приемника- RI
;                                   Флаг прерывания передатчика- TI
;                                    Девятый бит принят. данных в реж. 2 и 3 - RB8
;                                   Девятый бит перед. данных в реж. 2 и 3 - TB8
;                                   Разрешение приема последоват. данных - REN
;                                   Разрешение многопроцессорной работы - SM2
;                                   Режим работы послед. интерфейса - SM0,SM1
            MOV PCON,#10000000B
;                                   Бит удвоения скорости - SMOD
            MOV   TCON,#01010101B
;                                   Вид прерывания по входу /INT0
;                                   1- прерывание по фронту
;                                   0- прерывание по уровню ( бит IT0 )
;                                   Флаг запроса внешн. прер. по /INT0 (бит IE0 )
;                                   Вид прерывания по входу /INT1 ( бит IT1 )
;                                   Флаг запроса внешн. прер. по /INT1 ( бит IE1 )
;                                   Бит включения T/C0 (1-вкл) ( бит TR0 )
;                                   Флаг переполнения T/C0 ( бит TF0 )
;                                   Бит включения T/C1 ( бит TR1 )
;                                   Флаг переполнения T/C1 ( бит TF0 )
            MOV P1,#11000001B
            ACALL IND
            MOV P1,#0FFh
            MOV IE,#10000001B       ; разрешение прерываний от /INT0
            SETB TI                 ; установка бита передатчика
MAIN:       ACALL INBYTE
            CLR EA
            CJNE A,#2h,IKOM3
```

```
; ********** исполнение команды 2 ******************
                ACALL INBYTE
                MOV R0,A
                ACALL INBYTE
                MOV @R0,A
                SETB EA
                AJMP MAIN
IKOM3:          CJNE A,#03h,ERR; идентификация команды 3
; ********** исполнение команды 3 ******************
                ACALL INBYTE
                MOV R0,A
                MOV A,@R0
                ACALL OUTBYTE
                SETB EA
                AJMP MAIN
ERR:            AJMP MAIN
; ************** прием байта из RS-232 **************
INBYTE:         JNB RI,INBYTE ; ожидание флага приемника
                MOV A,SBUF
                CLR RI ; очистка флага
                RET
; ************** передача байта в RS-232 ************
OUTBYTE:        JNB TI,OUTBYTE ;проверка готовности передатчика
                CLR TI
                MOV SBUF,A
                RET
IND:            MOV R2,#07h
IND2:           MOV R1,#0FFh
IND1:           MOV R0,#0FFh
                DJNZ R0,$
                DJNZ R1,IND1
                DJNZ R2,IND2
                RET
END
```

Порядок выполнения работы

1. Создать новый файл.

2. С помощью операций над блоками текстовой информации скопировать пример рабочей программы, приведенный выше в данном описании.

3. Оттранслировать полученную программу.

4. При отсутствии ошибок создать HEX файл.

5. Выполнить несколько команд в пошаговом режиме и с точками

останова, обратив внимание на то, как изменяется содержимое регистровой памяти, регистров специального назначения и стека, а также регистра флагов (PSW).

6. Изменить некоторые ячейки памяти.

7. Завершить работу Вашей программы, выполнив ее до конца.

8. Завершить работу программной среды.

9. Продемонстрировать работу программы преподавателю.

Контрольные вопросы

1. Назовите директивы ассемблера и их назначение.

2. Как в программе задается тип используемого микроконтроллера?

3. Опишите структуру программы.

4. Можно ли модуль основной программы и модуль подпрограмм менять местами?

5. Обязательно ли в программе присутствие строки:

MOV TMOD, 00100010B ; установка режимов таймеров

6. Что означает ошибка в программе TARGET OUT OF RANGE?

7. Что означает ошибка в программе UNDEFINED SIMBOL `ffh` в строке программы

MOV 2fh,#ffh

8. Что означает ошибка в программе EXPRESSION TYPE DOES NOT MATCH INSTRUCTION в строке программы

MOV #2fh,20h

9. Что означает ошибка в программе ATTEMPT TO REDEFINE SYMBOL `BG15` в строке программы

bg15: DJNZ R3,bg16

10. Для чего применяется режим пошагового выполнения программы?

ЛАБОРАТОРНАЯ РАБОТА №2

Изучение системы команд микроконтроллера AT89C51: форматы представления данных и команд, способы адресации операндов, арифметические и логические операции

Цель работы:

1. Изучить систему команд микроконтроллера AT89C51: форматы представления данных и команд, способы адресации операндов, команды операций с данными, признаки результата операций, команды операций управления.

2. Приобрести навыки программирования микроконтроллера AT89C51 на языке Ассемблера.

Система команд MCS-51

Система команд MCS-51 содержит 111 базовых команд, которые удобно разделить по функциональному признаку на пять групп: арифметические операции, логические операции, команды передачи данных, команды операций с битами и команды передачи управления.

Большинство команд имеют формат один или два байта и выполняются за один или два машинных цикла. При тактовой частоте 12 МГц длительность машинного цикла составляет 1 мкс.

Методы адресации

Прямая адресация определяет операнд как 8-битный адрес в формате команды. Могут использоваться только адреса регистровой памяти данных и регистров специальных функций SFR.

Команда MOV A, 2Fh выполняет передачу данных из ячейки внутренней памяти данных с адресом 2Fh в аккумулятор (Acc).

В качестве прямого адреса может быть также указан один из регистров специальных функций. Например, команда MOV P1, A передает данные из аккумулятора на выходы порта P1.

При косвенной адресации в формате команды задается регистр, в котором находится адрес операнда. Могут использоваться адреса внутренней и внешней памяти данных. В качестве регистров для 8-битного адреса могут использоваться только регистры R0 и R1 выбранного регистрового банка. Для 16-ти битного адреса используется регистр указатель данных

DPTR. Команда MOV A, @R0 осуществляет передачу данных в аккумулятор из ячейки внутренней памяти данных, адрес которой находится в регистре R0. Команда MOVX A, @DPTR осуществляет передачу данных в аккумулятор из ячейки внешней памяти данных. Адрес ячейки находится в регистре DPTR.

Регистровая адресация определяет операнд в качестве одного из регистров R0…R7, выбранного регистрового банка.

Команда MOV A, R5 выполняет передачу данных из регистра R5 в аккумулятор.

Непосредственная адресация. Значение константы может быть указано непосредственно в коде команды. Например, MOV A, #100 загружает в аккумулятор десятичное число 100. Это число может быть также задано в шестнадцатеричном коде (64H) и в двоичном коде (01100100B).

Арифметические операции

Команды арифметических операций приведены в табл. 7, где показаны методы адресации, которые могут быть использованы в команде.

Таблица 7

Команды арифметических операций

Обозначение	Операция	Метод адресации				Время выпол., мкс
		Dir	Ind	Reg	Imm	
ADD A, <byte>	A=A+<byte>	+	+	+	+	1
ADDC A, <byte>	A=A+<byte>+C	+	+	+	+	1
SUBB A, <byte>	A=A-<byte>-C	+	+	+	+	1
INC A	A=A+1	Только аккумулятор				1
INC <byte>	<byte>=<byte>+1	+	+	+		1
INC DPTR	DPTR=DPTR+1	Только DPTR				2
DEC A	A=A-1	Только аккумулятор				1
DEC <byte>	<byte>=<byte>-1	+	+	+		1
MUL AB	B,A=B·A	Только аккумулятор и B				4
DIV AB	A=Int[A/B], B=Mod[A/B]	Только аккумулятор и B				4
DA A	Десятичная коррекция	Только аккумулятор				1

Например, команда ADD A, <byte> может быть записана в виде

ADD A, 7FH – прямая адресация (Dir);

ADD A, @R0 – косвенная адресация (Ind);

ADD A, R7 – регистровая адресация (Reg);

ADD A, #127 – непосредственная константа (Imm).

Логические операции

Команды выполнения логических операций приведены в табл. 8.

Здесь используются такие же обозначения, что и для арифметических операций. Исключение составляет запись вида ANL <byte>, #data, которая обозначает, что в качестве второго операнда всегда используется непосредственная константа. Например, ANL P1,#55H выполняет функцию логического «И» над содержимым порта P1 и константой 55H. Результат выводится в порт P1. Команда OR обозначает функцию логического «ИЛИ», а команда XOR – функцию «исключающего ИЛИ».

Таблица 8

Команды логических операций

Обозначение	Операция	Метод адресации				Время выполнения, мкс
		Dir	Ind	Reg	Imm	
ANL A, <byte>	A=A AND <byte>	+	+	+	+	1
ANL <byte>, A	<byte>=<byte> AND A	+				1
ANL <byte>, #data	<byte>=<byte> AND #data	+				2
ORL A, <byte>	A=A OR <byte>	+	+	+	+	1
ORL <byte>, A	<byte>=<byte> OR A	+				1
ORL <byte>, #data	<byte>=<byte> OR #data	+				2
XRL A, <byte>	A=A XOR <byte>	+	+	+	+	1
XRL <byte>, A	<byte>=<byte> XOR A	+				1
XRL <byte>, #data	<byte>=<byte> XOR #data	+				2
CRL A	Очистка Acc. A=00H	Только аккумулятор				1
CPL A	Инверсия Acc. A=NOT A	Только аккумулятор				1
RL A	Сдвиг Acc на 1 бит влево	Только аккумулятор				1
RLC A	Сдвиг Acc на 1 бит влево через перенос	Только аккумулятор				1
RR A	Сдвиг Acc на 1 бит вправо	Только аккумулятор				1
RRC A	Сдвиг Acc на 1 бит вправо через перенос	Только аккумулятор				1
SWAP A	Обмен полубайтами в Acc	Только аккумулятор				1

Команды передачи данных

Внутренняя память

Команды передачи данных приведены в табл. 9. Здесь используются следующие обозначения: <src> – источник данных, <dest> – приемник данных.

Таблица 9

Команды передачи данных

Обозначение	Операция	Метод адресации				Время выполнения, мкс
		Dir	Ind	Reg	Imm	
MOV A, <src>	A=<src>	+	+	+	+	1
MOV<dest>, A	<dest>=A	+	+	+		1
MOV <dest>, <src>	<dest>=<src>	+	+	+	+	2
MOV DPTR, #data16	Загрузка DPTR 16-ти битной константой				+	2
PUSH <src>	Сохранение в стековой памяти источника	+				2
POP <dest>	Загрузка приемника из стековой памяти	+				2
XCH A, <byte>	Обмен данными Acc и <byte>	+	+	+		1
XCHD A, @Ri	Обмен младшими полубайтами Acc и байта РПД		+			1

Внешняя память

Команды обмена данными с внешней памятью данных приведены в табл. 10, а команды чтения данных из внешней памяти программ – в табл. 11.

Таблица 10

Команды обмена данными с ВПД

Обозначение	Операция	Время выполнения, мкс
MOVX A, @Ri	Чтение внешней ПД по адресу @Ri	2
MOVX @Ri, A	Запись во внешнюю ПД по адресу @Ri	2
MOVX A, @DPTR	Чтение внешней ПД по адресу @DPTR	2
MOVX @DPTR,A	Запись во внешнюю ПД по адресу @DPTR	2

Таблица 11

Команды чтения данных из ВПП

Обозначение	Операция	Время выполнения, мкс
MOVC A, @A+DPTR	Чтение в Acc внешней ПП по адресу @A+DPTR	2
MOVC A, @A+PC	Чтение в Acc внешней ПП по адресу @A+PC	2

Команды операций с битами

Команды операций с битами приведены в табл. 12.

Таблица 12

Команды операций с битами

Обозначение	Операция	Время выполнения, мкс
ANL C, bit	C=C AND bit	2
ANL C, /bit	C=C AND NOT bit	2
ORL C, bit	C=C OR bit	2
ORL C, /bit	C=C OR NOT bit	2
MOV C, bit	C=bit	1
MOV bit, C	Bit=C	2
CLR C	C=0	1
CLR bit	Bit=0	1
SETB C	C=1	1
SETB bit	Bit=1	1
CPL C	Инверсия C	1
CPL bit	Инверсия bit	1
JC rel	Переход по адресу rel если C=1	2
JNC rel	Переход по адресу rel если C=0	2
JB bit, rel	Переход по адресу rel если bit=1	2
JNB bit, rel	Переход по адресу rel если bit=0	2
JBC bit, rel	Переход по адресу rel если bit=1 и очистка bit	2

Внутренняя ПД содержит 128 адресуемых бит, а также некоторые регистры специальных функций позволяют адресоваться к отдельным битам. Например, можно любые биты портов устанавливать в единицу или сбрасывать в ноль. Данные команды позволяют проводить установку,

сброс, инвертирование отдельных битов, а также выполнять пересылку, функции «И», «ИЛИ» над отдельными битами.

Например:

1) CLR P1.5 – сброс в 0 пятого разряда порта P1;

2) SETB 53H – установка в единицу бита 53H внутренней ПД, т.е. третьего разряда ячейки памяти с адресом 2AH;

3) MOV C, P3.7 – загрузка в признак C из порта P3.7
JC prod – если C=1, то переход на метку prod, если C=0, то
CPL C выполняется следующая команда (CPL C)
prod: MOV A, 25H

Команды передачи управления

В табл. 13 приведены команды безусловного перехода по программе. Команда JMP 2500h обозначает, что следующей будет выполняться команда, расположенная по адресу 2500h. В качестве addr можно указывать символические адреса, например JMP table.

Команда JMP addr при трансляции программы в среде «PROVIEW32» автоматически заменяется командой абсолютного перехода AJMP addr11, длинного перехода LJMP addr16 или короткого перехода SJMP <метка>. Формируется та команда, которая наиболее подходит для данного фрагмента программы. Соответственно команда CALL addr заменяется командой абсолютного вызова подпрограммы ACALL addr11 или длинного вызова подпрограммы LCALL addr16.

Таблица 13

Команды безусловной передачи управления

Обозначение	Операция	Время выполнения, мкс
JMP addr	Переход по адресу addr	2
JMP @A+DPTR	Переход по адресу A+DPTR	2
CALL addr	Вызов подпрограммы с адресом addr	2
RET	Возврат из подпрограммы	2
RETI	Возврат из прерываний	2
NOP	Пустая операция	1

В табл. 14 показаны команды условного перехода по программе.

Таблица 14

Команды условного перехода по программе

Обозначение	Операция	Метод адресации				Время, мкс
		Dir	Ind	Reg	Imm	
JZ rel	Переход, если A=0	Только аккумулятор				2
JNZ rel	Переход, если A≠0	Только аккумулятор				2
DJNZ <byte>, rel	Декремент байта и переход, если не ноль	+		+		2
CJNE A, <byte>, rel	Переход, если A≠<byte>	+			+	2
CJNE <byte>, #data, rel	Переход, если <byte>≠#data		+	+		2

Пример программы

Найти сумму положительных чисел, хранящихся в памяти программ, и результат разместить в аккумулятор.

```
$NOLIST                 ; switch off generation of the listing file
$NOMOD51                ; switch off 8051 controller defaults and use...
$INCLUDE (..\inc\reg8s52.inc) ; ...another controller definition file (i.e. 8052).
                ORG 0000h
                jmp begin
                ORG 0040h               ;установка адреса начала программы
begin:          mov R3,#20              ;размерность массива
                mov R1,#0
                mov R2,#0
                mov DPTR,#Mass          ;адрес массива
m1:             mov A,R1
                movc A,@A+DPTR          ;загрузка в акк. элемента массива
                jb Acc.7,m2
                add A,R2                ;прибавление положительного элемента
                mov R2,A                ;сохранение суммы в R2
m2:             inc R1
                djnz R3,m1
                mov A,R2
                ORG 0400h               ;установка адреса начала массива
Mass: DB 0,20,200,157,35,189,34,26,244,35,67,177,0,92,94,96,98,99,101,103,105
END
```

Порядок выполнения работы

1. Разработать алгоритм и написать программу в среде «PROVIEW32» в соответствии с заданием.

2. Оттранслировать полученную программу.

3. Выполнить программу в пошаговом режиме и с точками останова, обратив внимание на то, как изменяется содержимое регистровой памяти, а также регистра флагов (PSW).

4. Просмотреть полученный результат в регистровой памяти и убедиться в правильности выполнения программы.

5. В программе выделить команды, использующие признаки C, Z результата, прямую регистровую, косвенную и непосредственную адресацию операндов.

Задания.

1. В памяти программ размещены 64 числа. Отобрать положительные числа с нулем в третьем разряде и разместить в свободную область памяти данных.

2. В памяти программ размещены 64 числа. Отобрать положительные четные числа и разместить в свободную область памяти данных.

3. В памяти программ размещены 64 числа. Подсчитать количество положительных чисел и поместить результат в свободную область памяти данных.

4. В памяти программ размещены 64 числа. Найти среднее арифметическое положительных чисел и поместить результат в свободную область памяти данных.

5. В памяти программ размещены 64 числа. Отобрать отрицательные числа с нулем во втором разряде и разместить в свободную область памяти данных.

6. В памяти программ размещены 64 числа. Отобрать отрицательные четные числа и разместить в свободную область памяти данных.

7. В памяти программ размещены 64 числа. Подсчитать количество отрицательных чисел и поместить результат в свободную область памяти данных.

8. В памяти программ размещены 64 числа. Найти среднее арифметическое отрицательных чисел и поместить результат в свободную область памяти данных.

9. В памяти программ размещены 64 числа. Отобрать числа больше 25 и разместить в свободную область памяти данных.

10. В памяти программ размещены 64 числа. Отобрать числа меньше 40 и разместить в свободную область памяти данных.

11. В памяти программ размещены 64 числа. Отобрать числа кратные шести и разместить в свободную область памяти данных.

12. В памяти программ размещены 64 числа. Найти сумму чисел больших 40 и поместить результат в свободную область памяти данных.

13. В памяти программ размещены 64 числа. Подсчитать количество четных чисел и поместить результат в свободную область памяти данных.

14. В памяти программ размещены 64 числа. Подсчитать количество чисел с единицей в пятом разряде и поместить результат в свободную область памяти данных.

Содержание отчета:

Отчет по лабораторной работе должен содержать блок-схему алгоритма и текст программы.

Контрольные вопросы

1. Назовите методы адресации и объясните их действие.

2. Какие операнды могут использоваться в арифметических операциях?

3. Чем отличаются команды RL A и RLC A?

4. Как работает команда десятичной коррекции?

5. Как осуществляется обмен данными с внутренней памятью данных?

6. Как осуществляется обмен данными с внешней памятью данных и памятью программ?

7. Какое назначение команд PUSH <src> и POP <dest>?

8. Для каких операндов можно использовать битовые команды?

9. Как работает команда DJNZ R6, m1?

10. Чем отличаются команды RET и RETI ?

ЛАБОРАТОРНАЯ РАБОТА №3
Программирование устройств ввода-вывода дискретных сигналов.

Цель работы:

1. Изучить структуру и функциональные возможности параллельных портов микроконтроллера AT89C51.
2. Изучить порядок и особенности программирования портов микроконтроллера.
3. Подготовить программы с использованием программной среды.

Особенности работы портов

Обращение к портам ввода/вывода возможно с использованием команд, оперирующих с байтом, отдельным битом и произвольной комбинацией бит. При этом в тех случаях, когда порт является одновременно операндом и местом назначения результата устройство управления автоматически реализует специальный режим, который называется «чтение-модификация-запись». Этот режим обращения предполагает ввод сигналов не с внешних выводов порта, а из его регистра-защелки, что позволяет исключить неправильное считывание ранее выведенной информации. Подобный механизм обращения к портам реализован в следующих командах:

ANL – логическое И, например ANL P1, A;

ORL – логическое ИЛИ, например ORL P2, A;

XRL – исключающее ИЛИ, например XRL P3, A;

JBC – переход, если в адресуемом бите единица, и последующий сброс бита, например JBC P1.1, LABEL;

CPL – инверсия бита, например CPL P3.3;

INC – инкремент порта, например INC P2:

DEC – декремент порта, например DEC P2;

DJNZ – декремент порта и переход, если его содержимое не равно нулю, например DJNZ P3, LABEL;

MOV PX.Y, C – передача бита переноса в бит Y порта X;

SET PX.Y – установка бита Y порта X;

CLR PX.Y – сброс бита Y порта X.

При выполнении последних трех команд сначала считывается байт из порта, а затем записывается новый байт в регистр-защелку. Причиной, по которой команды «чтение-модификация-запись» обеспечивают раздельный доступ к регистру-защелке порта и к внешним выводам порта, является необходимость исключить возможность неправильного прочтения уровней сигналов на внешних выводах.

Примеры программирования портов

1. Составить программу, обеспечивающую вывод на светодиоды состояния переключателей *SB*1…*SB*8. Светодиоды подключим к линиям порта P2, а переключатели – к линиям порта P1.

Для включения светодиода на его катод необходимо подать логический нуль, поэтому данные, принятые с переключателей, необходимо инвертировать.

```
        ORG 0000h
        jmp begin
        ORG 0040h           ;установка адреса начала программы
begin:  mov A,P1            ;читаем состояние переключателей
        cpl A
        mov P2,A            ;выводим на светодиоды
        jmp begin
```

Как известно одним из недостатков механических переключателей является эффект «дребезга» контактов. Дребезг контактов у переключателей проявляется в силу их механических свойств и заключается в том, что при переключении замыкание контакта происходит не мгновенно, а происходит несколько срабатываний так, как показано на рис. 12.

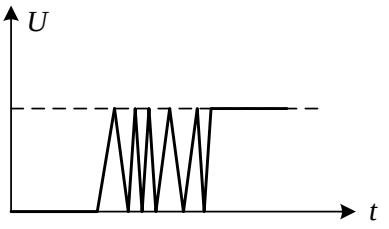


Рис. 12. Эффект дребезга контактов

Таким образом, при вводе информации с переключателей в цифровую систему возможно прохождение не одного сигнала о его срабатывании, а нескольких. Для устранения этого эффекта применяют различные устройства защиты от дребезга контактов, например реализованные на триггере.

Можно также реализовать программную защиту от дребезга контактов. Как показывает практика, время дребезга не превышает 20 мс. Поэто-

му при опросе состояния переключателя вводится временная задержка на время проявления эффекта дребезга.

Например, если ожидается изменение состояния переключателя *SB*1, подключенного к порту P1.0, из нуля в единицу, то программа защиты от дребезга выглядит следующим образом:

```
m1:     jnb P1.0,m1     ;читаем состояние переключателя
        call del        ;если состояние 1, то задержка на 20 мс
        jnb P1.0,m1     ;если состояние опять 0, то переход на m1
        …               ;если 1, то продолжение программы
```

2. Составить программу преобразования двоичного кода в двоично-десятичный. Двоичный код задается переключателями *SB*1…*SB*8, а двоично-десятичный отображается на семисегментных индикаторах, работающих в режиме динамической индикации.

Максимальное число, которое может быть задано восьмиразрядным двоичным кодом это число 11111111b, что соответствует десятичному числу 255. В результате преобразования получаем три числа – единицы, десятки и сотни, которые выводим на индикаторы. Для преобразования двоичного кода в двоично-десятичный используем команду деления.

Команда DIV производит деление содержимого аккумулятора A на содержимое регистра-расширителя B. После деления аккумулятор содержит целую часть от частного, а расширитель – остаток.

Исходное число, задаваемое переключателями *SB*1…*SB*8, которые подключим к порту P1, пересылаем в аккумулятор, а результат преобразования сохраним в регистрах.

Программа преобразования двоичного кода в двоично-десятичный

```
        mov A, P1       ;загрузка исходного числа
        mov B,#100      ;загрузка делителя
        div AB          ;деление на 100
        mov R3,A        ;в R3 число сотен
        mov A,B
        mov B,#10
        div AB          ; деление на 10
        mov R4,A        ;в R4 число десятков
```

```
mov R5,B          ;в R5 число единиц
```

Для вывода чисел на индикаторы необходимо собрать схему, показанную на рис. 13. Динамическая индикация осуществляется поочередным включением индикаторов И1…И4. В этом режиме код выводимой цифры подается одновременно на все индикаторы по линиям *a*…*h*, а питание подается только на индикатор, соответствующий выводимой цифре.

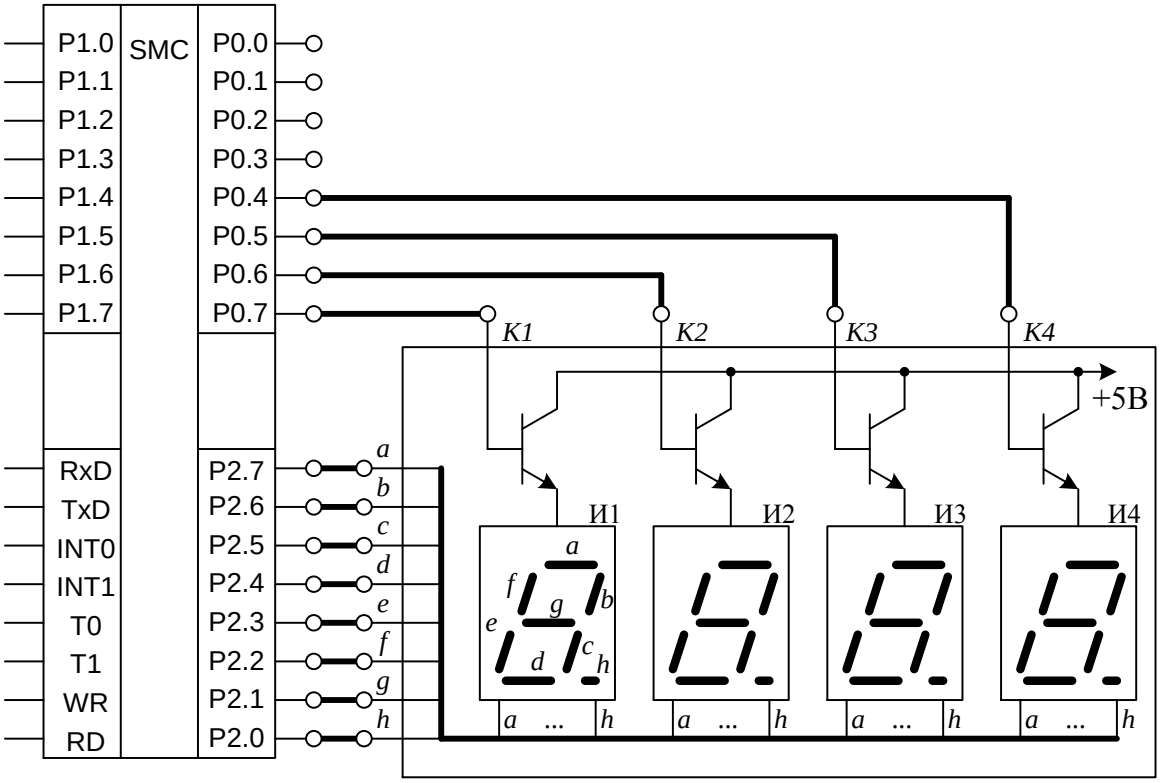


Рис. 13. Динамическая индикация

Например, если выводятся сотни, то включается индикатор И2, если десятки – И3, если единицы – И4. На индикаторе И1 выводятся тысячи. В нашем случае число тысяч всегда равно нулю, поэтому на индикатор И1 всегда выводится 0 или его можно не включать.

Для включения индикатора необходимо на базу соответствующего транзистора *Ki* подать уровень логической единицы. Временная диаграмма работы индикаторов приведена на рис. 14.

В каждый момент времени включен только один индикатор. Если частота переключения ключей более 50 Гц, то человеческий глаз не воспри-

нимает мелькание. В этом случае яркость свечения индикатора пропорциональна коэффициенту заполнения k, который определяется выражением $k = \frac{t_{откр}}{t_{закр}}$, где $t_{откр}$ – время открытия транзистора, $t_{закр}$ – время закрытия. Для задания времени открытия транзистора можно использовать программную задержку.

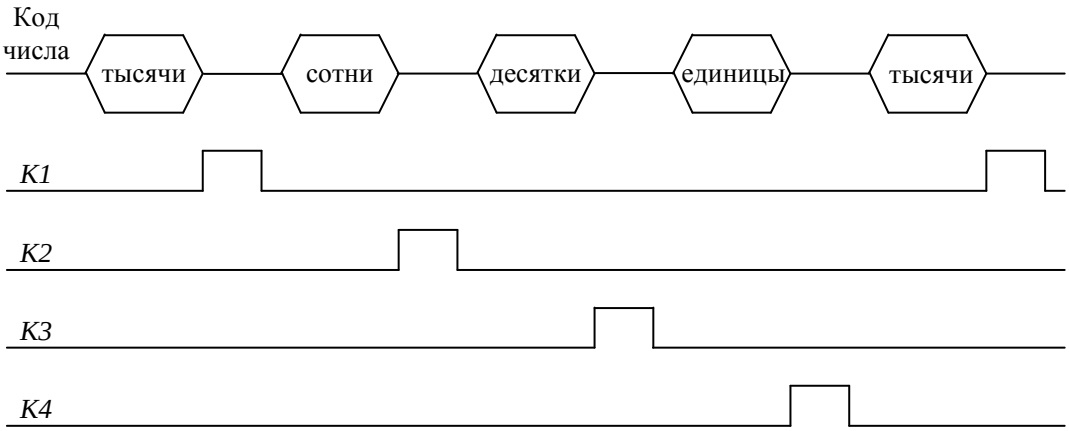


Рис. 14. Временная диаграмма

Как говорилось выше, в регистрах хранятся числа, соответствующие количеству сотен, десятков и единиц. Эти числа представлены двоичным кодом. Чтобы сформировать цифру на индикаторе нужно выполнить преобразование кода цифры, хранящейся в регистре, в код для семисегментного индикатора. Наиболее удобно для этой цели использовать таблицу. Таблица состоит из десяти строк. В каждой строке хранится семисегментный код, соответствующей цифры. В нулевой строке – код нуля, в первой – код единицы и так далее.

Для формирования цифры 0 необходимо включить сегменты *a*, *b*, *c*, *d*, *e*, *f* индикатора (см. рис. 13), цифры 1 – сегменты *b*, *c*, цифры 4 – *b*, *c*, *f*, *g* и так далее. Это осуществляется подачей логического нуля на соответствующий сегмент. Таким образом, для вывода цифр нужно подать следующие коды:

Цифра 0 – код 00000011b = 03h,
Цифра 1 – код 10011111b = 9Fh,
Цифра 2 – код 00100101b = 25h,
Цифра 3 – код 00001100b = 0Ch,

Цифра 4 – код 10011000b = 98h,
Цифра 5 – код 01001000b = 48h,
Цифра 6 – код 01000000b = 40h,
Цифра 7 – код 00011111b = 1Fh,
Цифра 8 – код 00000000b = 00h,
Цифра 9 – код 00001000b = 04h.

Полученные значения кодов для семисегментного индикатора разместим в области памяти программ в виде таблицы с символическим именем KOD:

```
KOD: DB   03h, 9Fh, 25h, 0Ch, 98h, 48h, 40h, 1Fh, 00h, 04h
```

Для чтения данных из памяти программ используется регистр указатель данных DPTR и команда пересылки movc. Далее приведен фрагмент программы преобразования десятичного кода, который хранится, например в регистре R5, в код семисегментного индикатора и передачи его в регистр R2.

```
        mov DPTR,KOD        ;загрузка адреса таблицы
        mov A,R5
        movc A,@A+DPTR
        mov R2,A
```

Таким образом, полная программа преобразования двоичного кода, задаваемого переключателями, и отображение его в двоично-десятичной форме на семисегментных индикаторах приведена ниже.

```
        ORG 0000h
        jmp begin
        ORG 0040h           ;установка адреса начала программы
```

Программа основного цикла

```
begin:  mov A,P1            ;читаем состояние переключателей
;преобразование двоичного кода в двоично-десятичный
        mov B,#100
        div AB
        mov R3,A            ;в R3 число сотен
        mov A,B
        mov B,#10
        div AB
```

```
        mov R4,A        ;в R4 число десятков
        mov R5,B        ;в R5 число единиц
        mov DPTR,KOD    ;загрузка адреса таблицы
        mov A,R3        ;формирование семисегментного кода сотен
        movc A,@A+DPTR
        mov P2,A
        setb P0.6       ;включение транзистора К2
        call delay
        clr P0.6        ;выключение транзистора К2
        mov A,R4    ;формирование семисегментного кода десятков
        movc A,@A+DPTR
        mov P2,A
        setb P0.5       ;включение транзистора К3
        call delay
        clr P0.5        ;выключение транзистора К3
        mov A,R5    ;формирование семисегментного кода единиц
        movc A,@A+DPTR
        mov P2,A
        setb P0.4       ;включение транзистора К4
        call delay
        clr P0.4        ;выключение транзистора К4
        jmp begin
```

Подпрограмма задержки на включение

```
delay:  mov R7,#20
del1:   mov R6,#150
        djnz R6,$
        djnz R7,del1
        ret
```

Блок неизменных данных

```
KOD:    DB   03h, 9Fh, 25h, 0Ch, 98h, 48h, 40h, 1Fh, 00h, 04h
```

Порядок выполнения работы

Часть I

1. Изучить устройство и порядок программирования портов ввода/вывода.

2. Создать программу, реализующую последовательное циклическое включение светодиодов «бегущий огонь». Время свечения светодиодов и задержку на переключения задавать с помощью программных задержек.

3. Произвести отладку программы в контроллере.

4. Продемонстрировать результат работы преподавателю.

Часть II

1. Реализовать вывод на семисегментные индикаторы в режиме динамической индикации количество включений переключателей *SB*1… *SB*3. Переключателем *SB*1 формировать единицы, *SB*2 – десятки, *SB*3 – сотни. При опросе состояния переключателей необходимо использовать защиту от дребезга контактов при переключении в ноль и в единицу.

2. Произвести отладку программы в контроллере.

3. Продемонстрировать результат работы преподавателю.

Содержание отчета:

Отчет по лабораторной работе должен содержать блок-схему алгоритмов и текст программ.

Контрольные вопросы

1. Объясните принцип работы режима «чтение-модификация-запись».

2. Какое назначение имеет регистр-защелка?

3. Какие команды используются для чтения данных из памяти программ?

4. Поясните принцип работы режима динамической индикации.

5. Как работают семисегментные индикаторы?

6. Как выполняется преобразование двоичного кода в двоично-десятичный?

7. Как выполняется преобразование двоичного кода в код семисегментного индикатора?

8. Что такое дребезг контактов и как осуществить защиту от него?

9. Как задается яркость свечения индикатора?

10. Что такое коэффициент заполнения импульса?

ЛАБОРАТОРНАЯ РАБОТА №4
Изучение режимов работы таймеров микроконтроллера AT89C51

Цель работы:

1. Изучить структуру и функциональные возможности таймеров микроконтроллера AT89C51.

2. Изучить порядок программирования таймеров микроконтроллера.

3. Подготовить программы с использованием программной среды.

Таймеры/счетчики

В составе средств MCS-51 имеются пары восьмиразрядных регистров с символическими именами THO, TLO и TH1, TL1, на основе которых функционируют два независимых программно-управляемых 16-битных таймера/счетчика событий. Код величины начального счета заносится в регистры Т/С программно. В процессе счета содержимое регистров Т/С инкрементируется. Признаком окончания счета, как правило, является переполнение регистра Т/С, т. е. переход его содержимого из состояния «все единицы» в состояние «все нули». Все регистры TH0, TH1, TL0, TL1 доступны по чтению и при необходимости контроль достижения требуемой величины счета может выполняться программно.

При работе в качестве таймера содержимое Т/С инкрементируется в каждом машинном цикле, т.е. через каждые 12 периодов резонатора. При работе в качестве счетчика содержимое Т/С инкрементируется под воздействием перехода из 1 в 0 внешнего входного сигнала, подаваемого на соответствующий (Т0, Т1) вывод микроконтроллера. Содержимое счетчика будет увеличено в том случае, если в предыдущем цикле был считан входной сигнал высокого уровня (1), а в следующем – сигнал низкого уровня (0). На распознавание перепада входного сигнала требуется два машинных цикла и максимальная частота входных сигналов равна 1/24 частоты резонатора. На длительность периода входных сигналов ограничений сверху нет.

Для управления режимами работы Т/С и для организации взаимодействия таймеров с системой прерывания используются два регистра специальных функций: регистр режима работы TMOD и регистр управления/статуса таймера TCON.

Обозначение разрядов регистров TMOD и TCON приведено в табл. 15 и 17. Назначение разрядов регистров приведено в табл. 16 и 18.

Таблица 15

Регистр режима работы

Биты	TMOD.7 для T/C1	TMOD.6 для T/C1	TMOD.5 для T/C1	TMOD.4 для T/C1
	TMOD.3 для T/C0	TMOD.3 для T/C0	TMOD.3 для T/C0	TMOD.3 для T/C0
Обозначение	GATE1	C/T1	Ml	M0

Таблица 16

Назначение разрядов регистра TMOD

<table>
<tr><th>Биты</th><th>Наименование</th><th>Назначение битов</th><th>Примечание</th></tr>
<tr><td>0,1
4,5</td><td>M0-M1</td><td>Определяют один из 4-х режимов работы, отдельно для T/C1 и T/C0
<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 режим работы T/C0, биты 4-7 – режим работы T/C1.</td></tr>
<tr><td>2,6</td><td>C/T 0
C/T 1</td><td>Определяют работу в качестве:
C/T0, C/T 1 = 0 – таймера
C/T0, C/T 1 = 1 – счетчика</td></tr>
<tr><td>3,7</td><td>GATE</td><td>Разрешает управлять таймером от внешнего вывода (INT0- для T/C0, INT1 - для T/C1).
GATE = 0 – управление запрещено
GATE = 1 – управление разрешено</td></tr>
</table>

Таблица 17

Регистр управления/статуса таймера

Биты	TCON.7	TCON.6	TCON.5	TCON.4	TCON.3	TCON.2	TCON.1	TCON.0
Обозначение	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Таблица 18

Назначение разрядов регистра TCON

Биты	Наименование	Назначение битов	Примечание
6 4	TR1 TR0	Биты управления таймерами. TR=1 – пуск таймера, TR=0 – стоп таймера	Биты устанавливаются и сбрасываются программно. Доступны по чтению.
7 5	TF1 TF0	Флаги переполнения Т/С.	Биты сбрасываются и устанавливаются аппаратно и программно. Доступны по чтению.
2 0	IT1 IT0	Биты, определяющие вид прерывания по входам INT0, INT1: IT=0 – прерывание по уровню (низкому) IT=1 – прерывание по фронту (переход из «1» в «0»)	Биты устанавливаются и сбрасываются программно Доступны по чтению.
3 1	IE1 IE0	Флаги запроса внешних прерываний INT0, INT1. Устанавливаются по срезу сигнала INT	Биты сбрасываются и устанавливаются аппаратно и программно. Доступны по чтению.

Примечание: биты 4,5 относятся к Т/С 0; биты 6, 7 - к Т/С 1. Биты 0,1 определяют внешние прерывания по входу INT0, биты 2,3 – по входу INT1.

Флаги переполнения TF0 и TF1 устанавливаются аппаратно при переполнении соответствующих Т/С (переход Т/С из состояния «все единицы» в состояние «все нули»). Если при этом прерывание от соответствующего Т/С разрешено, то установка флага TF вызовет прерывание. Флаги TF0 и TF1 сбрасываются аппаратно при передаче управления программе обработки соответствующего прерывания.

Флаги IE0 и IE1 устанавливаются аппаратно от внешних прерываний или программно и инициируют вызов программы обработки соответствующего прерывания. Сброс этих флагов выполняется аппаратно при обслуживании прерывания только в том случае, когда прерывание было вызвано по фронту сигнала. Если прерывание было вызвано уровнем сигнала на входе INT0 (INT1), то сброс флага IE должна выполнять программа обслуживания прерывания, воздействуя на источник прерывания для снятия им запроса.

Режимы работы Т/С

Режим работы каждого Т/С определяется значением битов М0, Ml в регистре TMOD. Т/С0 и Т/С1 имеют четыре режима работы. Режимы работы 0, 1, 2 одинаковы для обоих Т/С; Т/С0 и Т/С1 в этих режимах полностью независимы друг от друга. Работа Т/С0 и Т/С1 в режиме 3 различна. При этом установка режима 3 в Т/С0 влияет на режимы работы Т/С1.

Режим 0. Т/С в режиме 0 представляет собой устройство на основе 13-разрядного регистра и функционально совместим с таймером/счетчиком семейства МК48 (8-разрядный таймер/счетчик с предделителем на 32).

13-разрядный регистр состоит для Т/С0 из 8 разрядов регистра TH0 и 5 младших разрядов регистра TL0, а для Т/С1 – из 8 разрядов регистра TH1 и 5 младших разрядов регистра TL1. В этом режиме функцию делителя на 32 выполняют регистры TL0, TL1. Они являются программно доступными, но надо помнить, что значащими в режиме 0 являются только пять младших разрядов регистров TL0, TL1. Счет начинается при установке бита TR регистра TCON в состояние «1». При необходимости управления счетом извне бит GATE регистра TMOD устанавливается в состояние «1». Тогда при TR=1 счет будет разрешен, если на входе INT0 (для Т/С0) или INT1 (для Т/С1) установлено состояние «1» и будет запрещен, если установлено состояние «0». Установка бита TR0 для Т/С0 и TR1 для Т/С1 в состояние «0» выключает Т/С независимо от состояния других битов.

При переполнении Т/С (переход содержимого регистра Т/С из состояния «все единицы» в состояние «все нули») устанавливается флаг TF0 для Т/С0 или TF1 для Т/С1 в регистре TCON.

Режим 1 аналогичен режиму 0. Отличие состоит в том, что установка режима 1 превращает Т/С в устройство на основе 16-разрядного регистра. Для Т/С0 регистр состоит из программно доступных пар TL0, TH0, для Т/С1 из программно доступных пар TL1, TH1.

Режим 2. В этом режиме Т/С представляет собой устройство на основе восьмиразрядного регистра TL0 для Т/С0 и TL1 для Т/С1. При каждом переполнении TL0, кроме установки в регистре TCON флага TF0, происходит автоматически перезагрузка содержимого из TH0 в TL0. Соответственно для Т/С1 при переполнении TL1 в регистре TCON устанавли-

вается флаг TF1 и происходит перезагрузка TL1 из TH1. Регистры TH0 и TH1 загружаются программно. Перезагрузка TL0 из TH0 и TL1 из TH1 не влияет на содержимое регистров TH0 и TH1.

Режим 3. T/C1 в режиме 3 заблокирован и просто сохраняет свой счет (значение кода в регистре T/C). Эффект такой же, как при установке TR1=0.

T/C0 в режиме 3 представляет собой два независимых устройства на основе восьмиразрядных регистров TL0 и TH0. Устройство на основе регистра TL0 может работать в режиме таймера и в режиме счетчика. За ним сохраняются все биты управления T/C0, оно реагирует на воздействия по входам T0, INT0. При переполнении TL0 устанавливается флаг TF0. Устройство на основе регистра TH0 может работать только в режиме таймера. Оно использует бит включения TR1, при переполнении TH0 выставляет флаг TF1. Других битов управления устройство на основе TH0 в этом режиме не имеет.

Установка T/C0 в режим 3 лишает T/C1 бита включения TR1. Поэтому T/C1 в режимах 0, 1, 2 при GATE1=0 всегда включен и при переполнении в режимах 0 и 1 T/C 1 обнуляется, а в режиме 2 перезагружается не устанавливая флаг, если T/C0 находится в режиме 3.

Пример программирования таймеров

Составить программу, обеспечивающую формирование ШИМ сигнала на линии порта P0.3, который используется для управления транзисторным ключом. Программа должна обеспечивать регулирование коэффициента заполнения $K_{çàï}$ в %, который определяется как отношение длительности импульса к периоду повторения $K_{зап} = \frac{T_{отк}}{T_{шим}} 100\%$ – величина, обратная скважности *Q* импульсов.

Форма сигнала ШИМ показана на рис. 15, где $T_{шим}$ – период ШИМ, $T_{отк}$ – время открытия, $T_{закр}$ – время закрытия ключа. Период ШИМ должен оставаться постоянным, т.е. должно соблюдаться условие $T_{шим} = T_{отк} + T_{закр}$.

В качестве устройства, формирующего $T_{отк}$ и $T_{закр}$ используем таймер T0 микроконтроллера. В качестве устройства, задающего коэффициент заполнения, будем использовать двухпозиционные переключатели

*SB*1…*SB*4 лабораторного стенда, которые подключим к линиям порта P1. Так как используется четыре переключателя, то можно получить $2^4 = 16$ различных комбинаций, и, следовательно, 16 значений коэффициента заполнения. Каждому значению двоичного кода ставится в соответствие коэффициент заполнения, например, как показано в табл. 19.

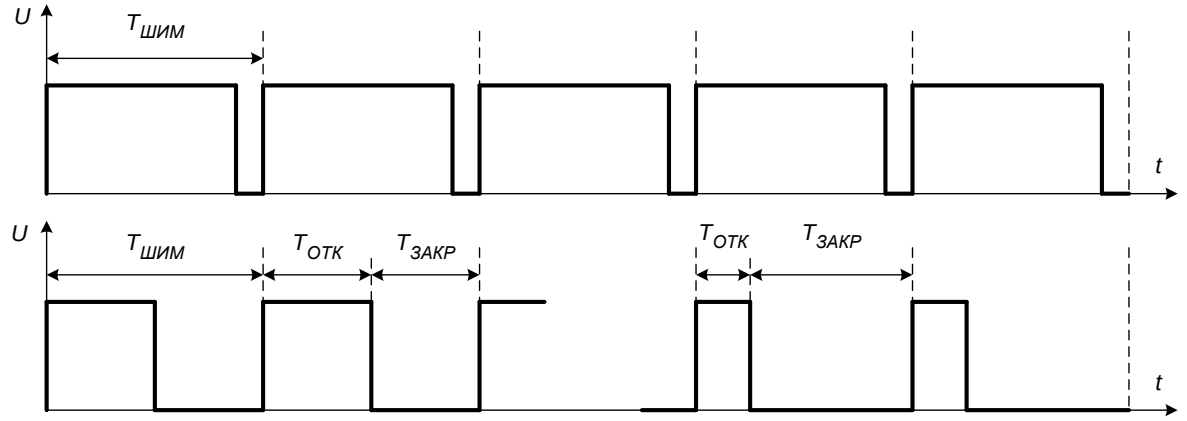


Рис. 15. Форма сигнала ШИМ

Таблица 19

Код	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$K_{çàï}$,%	0	7	13	20	27	33	40	47	53	60	67	73	80	87	93	100

Алгоритм работы программы состоит в следующем. С переключателей считывается значение кода коэффициента заполнения и осуществляется загрузка переменных для таймера, соответствующих коэффициенту. На линии порта P0.3 выставляется высокий уровень (лог.1) и запускается таймер на время, соответствующее времени открытия транзистора $T_{отк}$. По истечении этого времени таймер вызывает прерывание программы, на линии порта P0.3 выставляется низкий уровень (лог.0) и запускается таймер на время, соответствующее времени закрытия транзистора $T_{закр}$. Далее процесс повторяется.

Выполним расчет параметров, загружаемых в таймер. Предположим, что нам необходимо сформировать ШИМ с частотой 1 кГц, что соответствует периоду 1 мс. Для каждого значения коэффициента заполнения вычисляем значения $T_{отк}$ и $T_{закр}$.

Как говорилось выше, при работе таймера его содержимое инкрементируется в каждом машинном цикле, т.е. через каждые 12 периодов ге-

нератора. В стенде установлен задающий генератор с частотой 24 МГц (период 0,5 мкс). Следовательно, инкремент таймера выполняется каждые 6 мкс. Вычисляем количество отсчетов таймера для выполнения требуемых значений $T_{отк}$ и $T_{закр}$.

Таймер имеет четыре режима работы. Для данной задачи выбираем режим 1, в котором таймер работает на основе 16-разрядного регистра и состоит из программно доступных пар TH0, TL0. Если в таймер загрузить число 0, то он выполнит $2^{16} = 65536$ отсчетов и вызовет прерывание. Определяем число *N*, загружаемое в таймер, как разность между числом 65536 и вычисленным числом отсчетов таймера. Например, для $K_{зап}$=20% число отсчетов на открытие равно 33, тогда $N_{отк}$=65536 - 33=65503 (в шестнадцатеричном представлении FFDFh). Таким образом в старший регистр TH0 записываем число FFh=255, а в младший регистр TL0 – число DFh=223. Аналогично определяются значения $N_{закр}$. Результаты вычислений приведены в табл. 20, в которой указаны значения только для младшего регистра таймера.

Таблица 20

Код	$K_{зап}$,%	$T_{отк}$, мкс	$T_{закр}$, мкс	Число отсчетов на открытие	Число отсчетов на закрытие	$N_{отк}$	$N_{закр}$
0	1	10	990	2	165	254	91
1	7	67	933	11	156	245	100
2	13	133	867	22	144	234	112
3	20	200	800	33	133	223	123
4	27	267	733	44	122	212	134
5	33	333	667	56	111	200	145
6	40	400	600	67	100	189	156
7	47	467	533	78	89	178	167
8	53	533	467	89	78	167	178
9	60	600	400	100	67	156	189
10	67	667	333	111	56	145	200
11	73	733	267	122	44	134	212
12	80	800	200	133	33	123	223
13	87	867	133	144	22	112	234
14	93	933	67	156	11	100	245
15	99	990	10	165	2	91	254

Полученные значения $N_{отк}$ и $N_{закр}$ разместим в области памяти программ в виде таблиц с символическими именами TABO и TABZ.

Программа начинается с *описания символьных переменных*.

```
Kzaps   equ 32h     ;сохраненное значение Kзап
Kzap    equ 33h     ;считанное из порта P1 значение Kзап
Notk    equ 34h     ;число отсчетов таймера на открытие
Nzak    equ 35h     ;число отсчетов таймера на закрытие
PWM     equ 40h     ;признак ШИМ, PWM=0 – ключи закрыты.
                    ;PWM=1 - ключи открыты
```

Далее записывается *таблица векторов прерываний*. Так как другие прерывания не используются, указываем только вектор прерывания от TMP0.

```
        org 0000h
        jmp begin
        org 000bh       ;прерывание от TMP0
        jmp time0
```

Команды начальной установки

```
        org 0040h       ;установка адреса начала программы
begin:  clr P0.3        ;сброс линии порта
        clr EA          ;запрет всех прерываний
        mov PCON,#00000000B
        mov TMOD,#00100001B     ;8-битный TMP1 и 16-битный
                                ;TMP0 для ШИМ
        clr TR1         ;стоп таймера 1
        clr ET1         ;запрет прерываний от таймера 1
        clr EX0         ;запрет прерываний INT0
        clr EX1         ;запрет прерываний INT1
        mov TH0,#255    ;загрузка в TH0 (старший регистр) число 255
        mov TL0,#91     ;загрузка в TL0 (младший регистр) число 91
                        ;минимальное значение Kзап ШИМ
        setb ET0        ;разрешение прерываний от таймера 0
        setb TR0        ;пуск таймера 0
        setb EA         ;разрешение прерываний
```

Программа основного цикла

```
m1:     mov A,P1        ;чтение Kзап с переключателей
        anl A,#00001111b;очистка старших бит
        mov Kzap,A      ;сохранение в памяти Kзап
```

```
            clr C
            subb A,Kzaps      ;сравнение предыдущего значения K_зап
                              ;с текущим
            jz m1             ;если равны, то переход в начало
;если считано новое значение K_зап, то обновление N_отк и N_закр из таблиц
            mov DPTR,#TABO
            mov A,Kzap
            movc A,@A+DPTR
            mov Notk,A
            mov DPTR,#TABZ
            mov A,Kzap
            movc A,@A+DPTR
            mov Nzak,A
            jmp m1            ; переход в начало
```

Подпрограмма обработки прерываний

```
time0:      clr TR0           ;стоп таймера
            push PSW          ;сохранение слова состояния программы
            push Acc          ;сохранение аккумулятора
            jb PWM,tm2        ;если PWM=1, то переход на tm2
            setb P0.3         ;установка P0.3 (открытие ключа)
            setb PWM          ;установка PWM
tm5:        mov TL0, Notk     ;загрузка в таймер времени открытия
            mov TH0,#255
tm1:        pop Acc           ;восстановление аккумулятора
            pop PSW           ;восстановление слова состояния программы
            setb TR0          ;пуск таймера
            reti
tm2:        clr P0.3          ;сброс P0.3 (закрытие ключа)
            clr PWM           ;сброс PWM
            mov TL0, Nzak     ;загрузка в таймер времени закрытия
            mov TH0,#255
            pop Acc           ;восстановление аккумулятора
            pop PSW           ;восстановление слова состояния программы
            setb TR0          ;пуск таймера
            reti
```

Блок неизменных данных

```
TABO: DB  254,245,234,223,212,200,189,178,167,156,145,134,123,112,100,91
TABZ: DB  91,100,112,123,134,145,156,167,178,189,200,212,223,234,245,254
```

Порядок выполнения работы

Часть I

1. Изучить устройство и порядок программирования таймеров.

2. Создать программу, в которой с помощью таймера 1 сформировать временной интервал длительностью 1 секунда.

3. Оформить вывод числа секунд (10, 20) на шкалу светодиодов в двоично-десятичном виде.

4. Произвести отладку программы в контроллере.

5. Продемонстрировать результат работы преподавателю.

Часть II

1. Реализовать временные интервалы с использованием первого и второго таймеров, применяя программный опрос таймеров.

2. Реализовать на светодиодах контроллера включение светодиодов по заданному преподавателем алгоритму:

а) произвести последовательное циклическое включение светодиодов «бегущий огонь». Использовать первый таймер в качестве формирователя времени свечения светодиодов контроллера, вторым таймером задавать задержку переключения светодиодов;

в) сформировать свечение всех светодиодов с последовательным уменьшением яркости светодиодов.

Содержание отчета:

1. Алгоритмы программирования таймеров 1 и 2.
2. Схемы алгоритмов и реализующие их программы.

Контрольные вопросы

1. В чем различие таймера и счетчика?
2. Как определяется время счета таймера?
3. Чем отличаются режимы 0 и 1 работы таймеров?
4. В каком случае возникает прерывание от таймера?
5. Поясните принцип автоматической перезагрузки таймера.
6. Какие команды используются для разрешения и запрета прерываний от таймера?
7. Что такое скважность импульсов?
8. Что обозначают команды push PSW и pop PSW и для чего они используются?

ЛАБОРАТОРНАЯ РАБОТА №5
Программирование устройств вывода аналоговых сигналов

Цель работы:

1. Изучить функциональные возможности цифро-аналогового преобразователя.

2. Изучить порядок программирования и применения цифро-аналогового преобразователя.

Цифро-аналоговые преобразователи

Цифро-аналоговым преобразователем (ЦАП) называется устройство, предназначенное для преобразования цифровой информации в аналоговую. Они формируют сигнал в виде напряжения или тока, функционально связанного с управляющим кодом [4].

По виду выходного сигнала ЦАП делят на два вида: с токовым выходом и выходом по напряжению. Для преобразования выходного тока ЦАП в напряжение обычно используются операционные усилители.

По полярности выходного сигнала ЦАП принято делить на однополярные и двухполярные.

Управляющий код, подаваемый на вход ЦАП, может быть различным: двоичным, двоично-десятичным, Грея, унитарным и др.

Основные параметры ЦАП

Число разрядов (n) — число разрядов кода, отображающего исходную аналоговую величину, которое может подаваться на вход ЦАП.

Время установления ($t_{уст}$) — это интервал времени от подачи входного кода до вхождения выходного сигнала в заданные пределы, определяемые погрешностью.

Максимальная частота преобразования — наибольшая частота дискретизации, при которой все параметры ЦАП соответствуют заданным значениям.

В лабораторном стенде представлен восьмиразрядный ЦАП, реализованный на микросхеме К572ПА1. Опорное напряжение U_{REF}=10В. Условное обозначение и схема его подключения показаны на рис. 16. Такая схема обеспечивает двуполярный режим работы.

На цифровые входы *DB*0…*DB*7 подается двоичный код. Эта микросхема имеет внутренний резистор обратной связи R_{OC}, величина которого равна сопротивлению резистора матрицы, т.е. $R_{OC} = R$, и, следовательно, отношение $\frac{R_{OC}}{R} = 1$.

На выходе ЦАП формируется выходное напряжение $U_{вых}$ преобразователя, которое определяется по формуле

$$U_{вых} = U_{REF} \cdot \frac{R_{OC}}{R} \cdot \frac{(-2^{(n-1)} \cdot x_{n-1} + 2^{(n-2)} \cdot x_{n-2} + ...2^i \cdot x_i + ... + 2^0 \cdot x_0)}{2^n}.$$

Шаг квантования h, т.е. величина, на которую изменяется выходное напряжение при изменении кода на единицу младшего разряда, определяется по формуле $h = U_{REF} \cdot \frac{R_{OC}}{R \cdot 2^n}$.

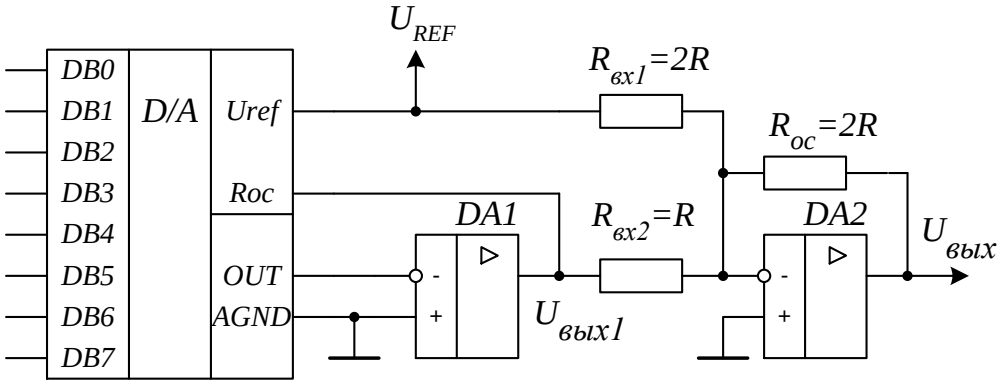


Рис. 16. Двухполярный ЦАП

Для получения положительных и отрицательных чисел, т.е. чисел со знаком используется формат чисел, в которых старший двоичный разряд является знаковым. Если в старший разряд равен нулю (0101101), то число положительное, а если единице (1101101), то отрицательное.

Таким образом, при изменении кода от 000...00 до 011...11 число является положительным, а далее от 100...00 до 111...11 число будет отрицательным, причем число 011...11 – максимальное положительное, а число 100...00 – минимальное отрицательное.

Для реализации в ЦАП этой зависимости старший разряд необходимо инвертировать. График выходного напряжения, которое в зависимости от входного двоичного кода изменяется от ($-U_{REF}$) до ($+U_{REF} - h$), показан на рис. 17.

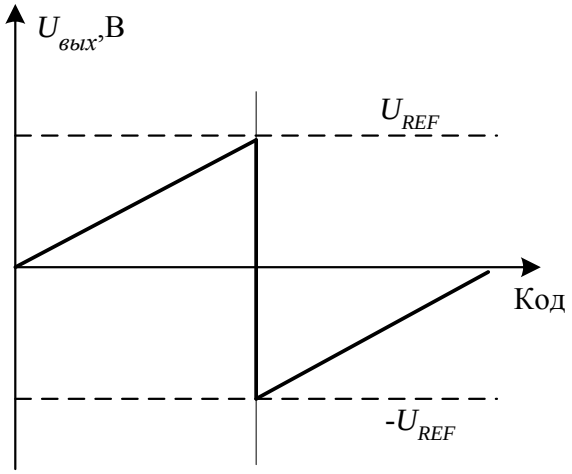


Рис. 17. Графики напряжений двухполярного ЦАП

Пример программирования ЦАП

Составить программу, формирующую на выходе ЦАП треугольную форму напряжения (рис.18). Переключателем *SB*1 изменять частоту повторения. Линии порта P2 будем использовать для задания двоичного кода на ЦАП, а переключатель *SB*1 подключим к порту P1.0.

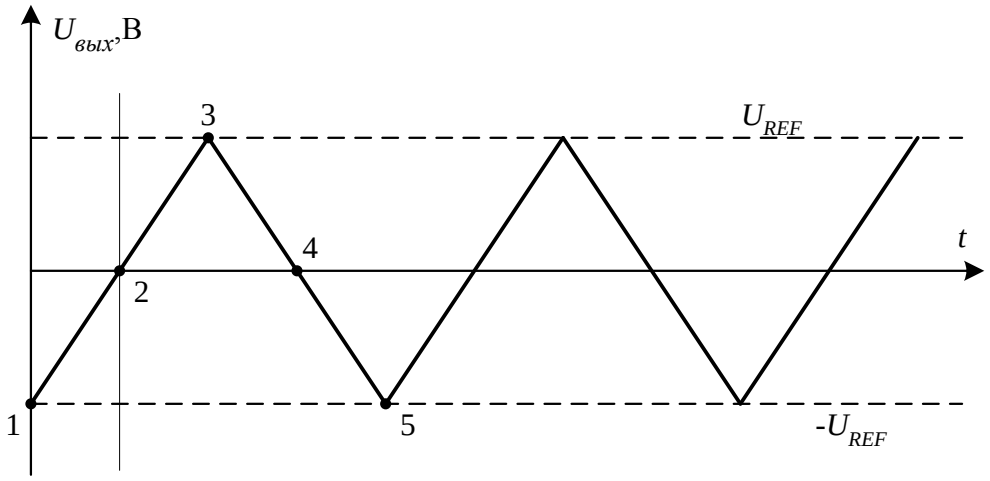


Рис. 18. Треугольная форма напряжения

В точке 1 выходное напряжение имеет минимальное отрицательное значение, что соответствует двоичному коду 10000000b. При походе к точке 2 значение кода увеличивается и достигает значения 11111111b. В точке 2 напряжение равно нулю, что соответствует нулевому двоичному коду. Далее при движении к точке 3 код увеличивается до значения 01111111 и

соответственно увеличивается напряжение до максимального положительного значения.

При движении от точки 3 к точке 5 выходное напряжение уменьшается до минимального отрицательного значения. Для организации этой зависимости значение двоичного кода изменяется в обратном порядке (01111111 … 00000000, 11111111 … 10000000).

Программа начинается с *описания символьных переменных*.

```
ST      equ 32h         ;признак направления счета.
FGN     equ 40h         ;частота генератора
```

Если бит ST=0, то значение двоичного кода увеличивается, если ST=1, то уменьшается. Переменой FGN присваивается значение адреса памяти, по которому хранится числовое значение, соответствующее периоду повторения сигнала треугольной формы.

Так как в программе прерывания не используются, то *таблица векторов прерываний* содержит только один вектор начального запуска.

```
        org 0000h
        jmp begin
```

Команды начальной установки

```
        org 0040h           ;установка адреса начала программы
begin:  mov P2,#0           ;обнуление ЦАП (порт P2)
        mov R1,#10000000b   ;начальное значение двоичного кода
                            ;(минимальное отрицательное напряжение)
        clr ST              ;сброс признака (счет на увеличение)
        clr EA              ;запрет всех прерываний
```

Программа основного цикла

```
m1:     jb P1.0,m2          ;проверка состояния переключателя
        mov FGN,#10         ;если P1.0=0, то период задается числом 10
        jmp m3
m2:     mov FGN,#50         ;если P1.0=1, то период задается числом 50
m3:     jb ST,m5            ;проверка направления счета
        mov P2,R1           ;вывод кода в ЦАП
        call delvar         ;задержка на формирования нового кода
```

Далее выполняется проверка значения кода. Если код достиг значения, соответствующего максимальному напряжению (01111111b), то выполняем переход на метку m4 где происходит счет на уменьшение. В противном случае продолжаем увеличивать значение кода командой inc R1.

```
        mov A,R1
        clr C
        subb A,# 01111111b
        jz m4
        inc R1
        jmp m1
```

Формирование кода для спадающего участка напряжения осуществляется аналогично нарастающему, но в этом случае выполняется проверка кода, соответствующего минимальному напряжению (10000000b).

```
m4:     setb ST          ;счет на уменьшение
m5:     mov P2,R1
        call delvar
        mov A,R1
        clr C
        subb A,# 10000000b
        jz m6
        dec R1
        jmp m1
m6:     clr ST           ;счет на увеличение
        jmp m1
```

Подпрограммы

```
delvar: mov r7,FGN
        djnz r7,$
        ret
```

Подпрограмма содержит изменяемую временную задержку с помощью которой можно изменять период (частоту) повторения сигнала.

Изменяя положение переключателя *SB*1, можно наблюдать с помощью осциллографа как изменяется частота повторения периодического сигнала треугольной формы.

Порядок выполнения работы

1. Изучить схему цифроаналогового преобразователя.

2. Составить программу, позволяющую получать напряжение на выходе ЦАП, соответствующую двоичному коду, задаваемому с помощью переключателей *SB*1… *SB*8.

3. Отладить программу и загрузить в контроллер. Запустить программу на выполнение. Выходное напряжение контролировать вольтметром или осциллографом.

4. Составить программу, формирующую на выходе ЦАП трапецеидальную форму напряжения. Переключателями *SB*1 и *SB*2 задавать четыре значения частоты повторения периодического сигнала.

5. Рассчитать величину шага квантования h и сравнить расчетное значение со значением, измеренным осциллографом.

6. Продемонстрировать работу преподавателю.

Содержание отчета:

1. Алгоритмы программирования ЦАП.
2. Тексты программ, реализующие разработанные алгоритмы.

Контрольные вопросы

1. Объясните принцип работы ЦАП с суммированием токов.
2. Как в ЦАП осуществляется преобразование тока в напряжение?
3. Перечислите основные параметры ЦАП?
4. Как определяется выходное напряжение ЦАП?
5. Что такое шаг квантования?
6. С какой целью необходимо инвертировать старший разряд ЦАП?
7. От чего зависит точность преобразования?
8. Сколько разрядов должен иметь ЦАП для получения точности установления выходного напряжения 0,1%?

ЛАБОРАТОРНАЯ РАБОТА №6
Программирование устройств ввода аналоговых сигналов

Цель работы:

1. Изучить функциональные возможности аналого-цифрового преобразователя.

2. Изучить порядок программирования и применения аналого-цифрового преобразователя.

Аналого-цифровые преобразователи

Аналого-цифровой преобразователь — устройство, предназначенное для преобразования непрерывно изменяющейся во времени аналоговой физической величины в эквивалентные ей значения числовых кодов.

В качестве аналоговой физической величины в общем случае могут фигурировать различные параметры, например угол поворота, линейное перемещение, скорость движения, температура, давление жидкости или газа и т.д. В дальнейшем под этой величиной будем понимать напряжение либо ток, которые, при необходимости, можно легко преобразовать в другие физические величины.

Процесс аналого-цифрового преобразования предполагает последовательное выполнение следующих операций:

- выборка значений исходной аналоговой величины в некоторые наперед заданные дискретные моменты времени, т. е. дискретизация сигнала по времени;
- квантование (округление до некоторых известных величин) полученной в дискретные моменты времени последовательности значений исходной аналоговой величины по уровню;
- кодирование – замена найденных квантованных значений некоторыми числовыми кодами.

В АЦП используют четыре основных типа кодов: натуральный двоичный, десятичный, двоично-десятичный и код Грея. Кроме этого, АЦП, предназначенные для вывода информации в десятичном коде, выдают на своем выходе специализированный код для управления семисегментными индикаторами.

Большинство АЦП работают с выходом в натуральном двоичном ко-

де, при котором каждому положительному числу N ставится в соответствие код $\{x_i\} = x_{n-1}x_{n-2}\ldots x_1x_0$,

где x_i принимает значения нуля или единицы. При этом положительное число в двоичном коде имеет вид

$$N = x_{n-1} \cdot 2^{n-1} + x_{n-2} \cdot 2^{n-2} + \ldots + x_1 \cdot 2^1 + x_0 \cdot 2^0 .$$

Такой код принято называть прямым: его крайний правый разряд является младшим, а крайний левый – старшим. Прямой код пригоден лишь для работ с однополярными сигналами. Полный диапазон преобразуемого сигнала равен 2^n, а $N_{\max} = 2^n - 1$.

Если АЦП должен работать с двухполярными числами, то наиболее часто используют дополнительный код.

Основные параметры АЦП

Время преобразования обычно определяют как интервал времени от начала преобразования до появления на выходе АЦП устойчивого кода входного сигнала. Для одних типов АЦП это время постоянное и не зависит от значения входного сигнала, для других АЦП это время зависит от значения входного сигнала.

Максимальная частота дискретизации – это частота, с которой возможно преобразование входного сигнала, при условии, что выбранный параметр (например, абсолютная погрешность) не выходит за заданные пределы. Иногда максимальную частоту преобразования принимают равной обратной величине времени преобразования. Однако это пригодно не для всех типов АЦП.

Все типы используемых АЦП можно разделить по признаку измеряемого значения напряжения на две группы: АЦП мгновенных значений напряжения и АЦП средних значений напряжения (интегрирующие АЦП).

АЦП мгновенных значений напряжений

АЦП последовательного приближения

Структурная схема АЦП последовательного приближения приведена на рис. 19. Их также называют АЦП с поразрядным уравновешиванием. По сравнению со схемой АЦП последовательного счета в ней сделано одно

существенное изменение – вместо счетчика введен регистр последовательного приближения (РПП). Это изменило алгоритм уравновешивания и сократило время преобразования.

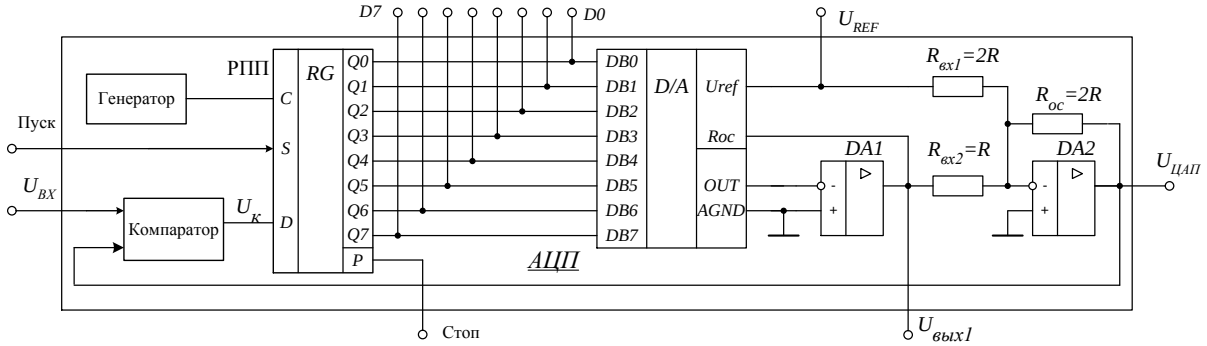


Рис. 19. АЦП последовательного приближения

В основе работы АЦП с регистром последовательного приближения лежит процедура последовательного сравнения преобразуемого напряжения $U_{\text{ВХ}}$ с набором фиксированных значений напряжений, равных 1/2, 1/4, 1/8, ... , $1/2^n$ от возможного максимального его значения $U_{\max} = U_{REF}$. Если число разрядов АЦП и соответственно РПП равно 10, то самое младшее фиксированное значение равно $1/2^{10} = 1/1024$.

Процесс преобразования начинается подачей сигнала «Пуск». При этом в регистр записывается двоичный код 1000...00 с единицей в старшем разряде, что соответствует установлению на выходе ЦАП напряжения $U_{\text{ЦАП}} = 1/2\ U_{REF}$ (рис. 20). Это напряжение сравнивается компаратором с входным напряжением. Если $U_{\text{ВХ}} > U_{\text{ЦАП}}$, то логическая единица остается в регистре, а если $U_{\text{ВХ}} < U_{\text{ЦАП}}$, то в регистр записывается ноль. Эта процедура на примере 10-ти разрядного АЦП показана в табл. 21. На следующем такте записывается единица во второй разряд регистра, что соответствует увеличению выходного напряжения ЦАП на 1/4 U_{REF}. Вновь производится сравнение напряжений и так процесс повторяется до тех пор, пока $U_{\text{ЦАП}}$ максимально не приблизится к входному напряжению $U_{\text{ВХ}}$.

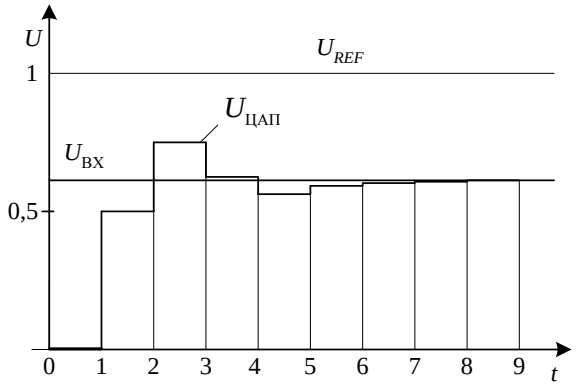


Рис. 20. Процесс преобразования АЦП последовательного приближения

Таблица 21

Процедура преобразования АЦП

N	Код	Результат	$U_К$
1	1000000000	$U_{ВХ} > U_{ЦАП}$	1
2	1100000000	$U_{ВХ} < U_{ЦАП}$	0
3	1010000000	$U_{ВХ} < U_{ЦАП}$	0
4	1001000000	$U_{ВХ} > U_{ЦАП}$	1
5	1001100000	$U_{ВХ} > U_{ЦАП}$	1
6	1001110000	$U_{ВХ} > U_{ЦАП}$	1
7	1001111000	$U_{ВХ} > U_{ЦАП}$	1
…	…	…	…

Таким образом, для *n*-разрядного АЦП процесс преобразования выполняется за *n* последовательных шагов приближения (итераций) вместо 2^n при использовании последовательного счета и получается существенный выигрыш в быстродействии.

Время преобразования АЦП последовательного приближения определяется выражением $T_{пр} = T \cdot (n+1)$, где T – период следования тактовых импульсов, n – число разрядов, n+1 – для формирования сигнала P (конец преобразования).

Если сигнал с выхода P (Стоп) подать на вход S (Пуск), то АЦП переходит в циклический режим работы.

Пример программирования АЦП

Составить программу, обеспечивающую вывод на светодиоды двоичного кода, эквивалентного величине постоянного напряжения, подаваемого на вход АЦП. К линиям порта P0 подключим светодиоды, порт P2 используется для ввода в микроконтроллер данных с АЦП. Сигнал «Пуск» формируется на линии порта P1.0.

Сигнал готовности АЦП (сигнал «Стоп») подключен к входу порта P3.2 ($\overline{INT0}$). Для опроса этого сигнала рассмотрим два варианта реализации программы.

1. Режим программного опроса сигнала готовности АЦП.

Так как в программе используется только вектор начального запуска, то *таблица векторов прерываний* содержит один.

```
        org 0000h
        jmp begin
```

Команды начальной установки

```
        org 0040h       ;установка адреса начала программы
begin:  mov P0,#0FFh    ;выключение светодиодов
        setb P1.0       ;включение сигнала «Пуск»
```

Программа основного цикла

```
m1:     clr P1.0        ;формирование сигнала «Пуск»
        call del1
        setb P1.0
        jb P3.2,$       ;ожидание сигнала готовности
        mov A,P2        ;вывод данных на индикаторы
        cpl A           ;инвертирование данных
        mov P0,A
        jmp m1
```

Подпрограмма задержки. Задает длительность сигнала «Пуск».

```
del1:   mov r7,#20
        djnz r7,$
        ret
```

При выводе данных на индикаторы используется команда инвертирования, так как включение светодиодов осуществляется низким уровнем сигнала.

Режим программного ввода/вывода данных имеет существенный недостаток. Практически всё основное время микроконтроллер находится в режиме ожидания сигнала готовности и не может выполнять никаких других действий. От этого недостатка свободен режим вводв/вывода по прерываниям.

2. Ввод сигнала готовности в режиме прерываний.

Таблица векторов прерываний содержит вектор начального запуска и вектор от внешнего прерывания INT0.

```
        org 0000h
```

```
        jmp begin
        org 0003h       ;прерывание от INT0
        jmp its0
```

Команды начальной установки

```
        org 0040h       ;установка адреса начала программы
begin:  mov P0,#0FFh    ;выключение светодиодов
        setb P1.0       ;ключение сигнала «Пуск»
        setb IT0        ;прерывание по срезу сигнала
        setb EX0        ;разрешение прерываний INT0
        setb EA         ;разрешение прерываний
```

Программа основного цикла содержит команды формирования первого запуска АЦП

```
m1:     clr P1.0        ;формирование сигнала «Пуск»
        call del1
        setb P1.0
        …
```

Далее может располагаться программа по обслуживанию любых других устройств

Вывод данных на индикаторы осуществляется в подпрограмме обслуживания внешних прерываний, которые вызываются при поступлении сигнала готовности АЦП.

Подпрограмма обработки прерываний

```
its0:   mov A,P2        ;вывод данных на индикаторы
        cpl A
        mov P0,A
        clr P1.0        ;формирование следующего сигнала «Пуск»
        call del1
        setb P1.0
        reti
```

Порядок выполнения работы

1. Изучить схему АЦП микроконтроллера.

2. Перевести АЦП в циклический режим работы. На вход АЦП подать постоянное напряжение и на выходе усилителя ЦАП наблюдать с по-

мощью осциллографа процесс преобразования.

3. Наблюдать процесс преобразования при подаче на вход АЦП гармонического сигнала, например синусоиды.

4. Выключить циклический режим работы. Запрограммировать режим работы АЦП. Контролировать работу АЦП на осциллографе. Предусмотреть средства вывода результатов работы в виде индикации на светодиодах.

5. Отладить программу и загрузить в контроллер. Запустить программу на выполнение и наблюдать результаты на светодиодах.

6. Подать на вход АЦП постоянное напряжение с выхода переменного резистора. Составить программу вывода на сегментные индикаторы измеренного напряжения. В программе предусмотреть масштабирование полученного двоичного кода в действительное значение напряжения. Диапазон изменения постоянного напряжения от -10В до +10В.

7. Продемонстрировать работу преподавателю.

Содержание отчета

1. Алгоритм программирования АЦП.
2. Схемы алгоритмов и реализующие их программы.
3. Графики процесса преобразования постоянного напряжения, полученные на осциллографе.

Контрольные вопросы

1. Какие операции необходимо выполнить при аналого-цифровом преобразовании?
2. Перечислите способы аналого-цифрового преобразования.
3. Какой АЦП является самым быстродействующим и почему?
4. Почему интегрирующий АЦП отличается повышенной точностью и помехозащищенностью?
5. Чем отличается программный ввод/вывод от ввода/вывода в режиме прерываний?
6. Какие команды необходимо использовать для настройки ввода данных в режиме прерываний?
7. Что означает команда setb IT0 прерывание по срезу сигнала?

ЛАБОРАТОРНАЯ РАБОТА №7
Изучение последовательного порта микроконтроллера AT89C51

Цель работы:

1. Изучить структуру и функциональные возможности последовательного порта микроконтроллера AT89C51.
2. Изучить порядок программирования последовательного порта микроконтроллера AT89C51.
3. Подготовить программы с использованием программной среды.

Последовательный интерфейс

Через универсальный асинхронный приемопередатчик (УАПП) осуществляется прием и передача информации, представленной последовательным кодом (младшими битами вперед), в полном дуплексном режиме обмена. В состав УАПП, называемого часто последовательным портом, входят принимающий и передающий сдвигающие регистры, а также специальный буферный регистр (SBUF) приемопередатчика. Запись байта в буфер приводит к автоматической переписи байта в сдвигающий регистр передатчика и инициирует начало передачи байта. Наличие буферного регистра приемника позволяет совмещать операцию чтения ранее принятого байта с приемом очередного байта. Если к моменту окончания приема байта предыдущий байт не был считан из SBUF, то он будет потерян.

Последовательный порт MCS-51 может работать в четырех различных режимах [3].

Режим 0. В этом режиме информация и передается, и принимается через внешний вывод входа приемника (RxD). Принимаются или передаются 8 бит данных (рис. 21). Через внешний вывод выхода передатчика (TxD) выдаются импульсы сдвига, которые сопровождают каждый бит. Частота передачи бита информации равна 1/12 частоты резонатора.

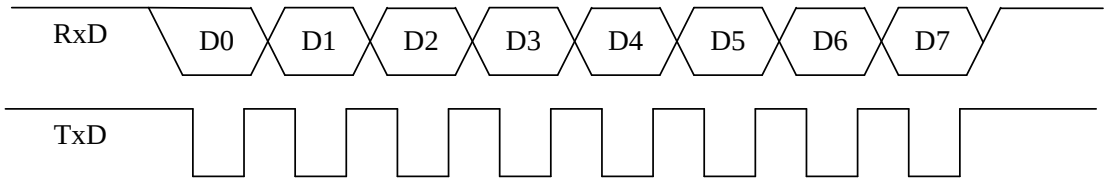


Рис. 21. Временная диаграмма работы УАПП в режиме 0

Режим 1. В этом режиме передаются через TxD или принимают RxD 10 бит информации: старт-бит (0), 8 бит данных и стоп-бит (рис. 22). Скорость приема/передачи — величина переменная и задается таймером.

Рис. 22. Временная диаграмма работы УАПП в режиме 1

Режим 2. В этом режиме через TxD передаются или из RxD принимаются 11 бит информации: старт-бит, 8 бит данных, программируемый девятый бит и стоп-бит. При передаче девятый бит данных может принимать значение 0 или 1, или, например, для повышения достоверности передачи путем контроля по четности в него может быть помещено значение признака паритета из слова состояния программы (PSW.0). Частота приема/передачи выбирается программой и может быть либо 1/32, либо 1/64 частоты резонатора в зависимости от управляющего бита SMOD.

Режим 3. Режим 3 совпадает с режимом 2 во всех деталях за исключением частоты приема/передачи, которая является величиной переменной и задается таймером.

Управление режимом работы УАПП осуществляется через регистр управления/статуса УАПП с символическим именем SCON.

Обозначение разрядов регистра SCON приведено в табл. 22. Все разряды регистра SCON программно доступны по записи и чтению. Разряды SM0, SM1 определяют режим работы УАПП, как указано в табл. 23.

Остальные биты регистра имеют следующее назначение:

SM2 – разрешение многопроцессорной работы. В режимах 2 и 3 при SM2=1 флаг RI не активизируется, если девятый принятый бит данных равен 0. В режиме 1 при SM2=1 флаг RI не активизируется, если не принят стоп-бит, равный 1. В режиме 0 бит SM2 должен быть установлен в 0.

REN – разрешение приема последовательных данных. Устанавливается и сбрасывается программным обеспечением соответственно для разрешения и запрета приема.

TB8 – девятый бит передаваемых данных в режимах 2 и 3. Устанавливается и сбрасывается программным обеспечением.

RB8 – девятый бит принятых данных в режимах 2 и 3. В режиме 1,

если SM2=0, RB8 является принятым стоп-битом. В режиме 0 бит RB8 не используется.

TI – флаг прерывания передатчика. Устанавливается аппаратно в конце времени выдачи 8-го бита в режиме 0 или в начале стоп-бита в других режимах. Сбрасывается программным обеспечением.

RI – флаг прерывания приемника. Устанавливается аппаратно в конце времени приема 8-го бита в режиме 0 или через половину интервала стоп-бита в режимах 1, 2, 3 при SM2=0.

Таблица 22

Регистр управления/статуса таймера

Биты	SCON.7	SCON.6	SCON.5	SCON.4	SCON.3	SCON.2	SCON.1	SCON.0
Обозначение	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Таблица 23

Режимы работы УАПП

SM0	SM1	Режим	Наименование	Скорость передачи
0	0	0	Сдвиговый регистр	f/12
0	1	1	8-битовый универсальный асинхронный прием-ник/передатчик	Переменная, задается Т/С1
1	0	2	9-битовый УАПП	f/64 или f/32
1	1	3	9-битовый УАПП	Переменная, задается Т/С1

Буфер приемника и буфер передатчика при программном доступе имеют одинаковые имя (SBUF) и адрес (99H). Если команда использует SBUF как регистр источника, то обращение происходит к буферу приемника. Если команда использует SBUF как регистр назначения, то обращение происходит к буферу передатчика.

Скорость приема/передачи, т.е. частота работы УАПП в различных режимах, определяется различными способами.

В режиме 0 частота передачи зависит только от резонансной частоты кварцевого резонатора $f_0 = f_{\text{рез}}/12$. За один машинный цикл последовательный порт передает один бит информации. Таким образом, при частоте резонатора 24 МГц частота приема/передачи составит 2 МГц.

В режимах 1, 2 и 3 скорость приема/передачи зависит от значения управляющего бита SMOD в регистре специальных функций PCON (табл. 24).

Таблица 24

Назначение разрядов регистра управления мощностью PCON

Биты	Обозначение	Назначение битов
PCON.7	SMOD	Удвоенная скорость передачи. Если бит установлен в 1, то скорость передачи вдвое больше, чем при SMOD = 0
PCON.6 PCON.5 PCON.4		Не используются
PCON.3	GF1	Флаги, специфицируемые пользователем (флаги общего назначения)
PCON.2	GF0	
PCON.1	PD	Бит пониженной мощности. При установке бита в 1 МК переходит в режим пониженной потребляемой мощности
PCON.0	IDL	Бит холостого хода. При установке бита в 1 МК переходит в режим холостого хода

В режиме 2 частота, передачи определяется выражением $f_2 = (2^{\text{SMOD}} / 64) f_{\text{рез}}$. Иными словами, при SMOD = 0 частота передачи равна $(1/64) f_{\text{рез}}$, а при SMOD = 1 равна $(1/32) f_{\text{рез}}$, то есть при частоте резонатора 24 МГц частота приема/передачи составит 375 МГц и 750 МГц соответственно.

В режимах 1 и 3 в формировании частоты передачи кроме управляющего бита SMOD принимает участие таймер 1. При этом частота передачи зависит от частоты переполнения f_{OVT1} таймера 1 и определяется выражением: $f_{1,3} = (2^{\text{SMOD}} / 32) f_{OVT1}$. Прерывание от таймера 1 в этом случае должно быть заблокировано.

Наиболее удобно использовать режим таймера с автоперезагрузкой. Старшая тетрада TMOD = 0010B (см. табл. 22 и 23). Частота передачи определяется выражением $f_{1,3} = (2^{\text{SMOD}} / 32)(f_{\text{рез}} / 12)(256 - (\text{TH1})$. В табл. 25 приведены примеры настройки T/C1 для получения типовых частот передачи данных через УАПП при различных значениях частот кварцевого резонатора.

Таблица 25

Настройка таймера 1 для управления частотой работы УАПП

f = 12 МГц

Скорость передачи, бит/с	Число в TH1	Действительная скорость передачи, бит/с	Погрешность, %	SMOD
19200	FEh	15625	18,62	0
9600	FDh	10416	-8,51	0
4800	F9h	4464	6,99	0
2400	F3h	2403	-0,125	0
1200	E6h	1201	-0,083	0
19200	FDh	20833	-8,51	1
9600	F9h	8928	6,99	1
4800	F3h	4807	-0,15	1
2400	E6h	2403	-0,125	1
1200	CCh	1201	-0,083	1

f = 24 МГц

Скорость передачи, бит/с	Число в TH1	Действительная скорость передачи, бит/с	Погрешность, %	SMOD
19200	FDh	20833	-8,51	0
9600	F9h	8928	6,99	0
4800	F3h	4807	-0,15	0
2400	E6h	2403	-0,125	0
1200	CCh	1201	-0,083	0
19200	F9h	17857	6,99	1
9600	F3h	9615	-0,16	1
4800	E6h	4807	-0,15	1
2400	CCh	2403	-0,125	1
1200	98h	1201	-0,083	1

f = 11,059 МГц

Скорость передачи, бит/с	Число в TH1	Действительная скорость передачи, бит/с	Погрешность, %	SMOD
19200	FFh	28799	-49,99	0
9600	FDh	9599	0,01	0
4800	FAh	4799	0,021	0
2400	F4h	2399	0,042	0
1200	E8h	1199	0,083	0
19200	FDh	19199	0,005	1
9600	FAh	9599	0,010	1
4800	F4h	4799	0,021	1
2400	E8h	2399	0,042	1
1200	D0h	1199	0,083	1

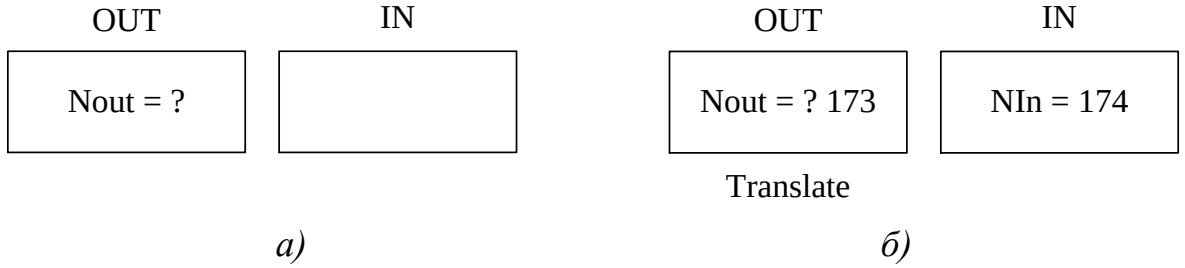
Пример программирования УАПП

На компьютере задается число в диапазоне от 0 до 255 (1 байт) и передается в микроконтроллер. Составить программу приема байта данных от компьютера с отображением на светодиодах двоичного кода принятого байта. Принятый байт увеличить на единицу и передать обратно в компьютер.

В данном примере приведена программа передачи данных от компьютера, составленная на Turbo Basic. Можно использовать любой другой язык программирования: Pascal, C++, Delphi, Visual Basic и другие.

Программа начинается оператором «open», который открывает порт компьютера com1 для передачи восьми бит со скоростью 9600 бит/с без контроля на четность.

Следующие операторы формируют на экране изображение, показанное на рис. 23*а*. В окно OUT вводится число, предназначенное для передачи в микроконтроллер. Если число выходит за пределы указанного диапазона, то выводится сообщение об ошибке: Error Number. При правильном вводе появляется изображение, показанное на рис. 23*б*. Информационное сообщение «Translate» указывает о том, что идет обмен данными с микроконтроллером. В окне IN появляется значение принятого числа.



а) *б)*

Рис. 23. Информационные сообщения на экране компьютера

Передача данных осуществляется командой «out &H3F8,N», а прием данных подпрограммой «Priem», в которой проверяется состояние флага готовности COM порта и ввод данных оператором «inp(&H3F8)».

Временные задержки delay 2 введены для удобства работы с программой, при необходимости их можно убрать.

Программа

```
open "com1:9600,n,8,2,rs,cs0,ds0,cd0" as #2:out &h3f9,0
screen 9
m1:
        cls
        line (100,100)-(230,130), ,B
        line (250,100)-(360,130), ,B
        locate 7,20
        print "OUT                IN"
        locate 9, 16
        input "Nout = "; N
        if N >= 0 and N <= 255 then m2
        locate 12,18
        print "Error Number"
        delay 2
        goto m1
m2 :
        locate 12,18
        print "Translate"
        delay 2
        out &H3F8, N
        gosub Priem
        locate 9, 34
        print "NIn = "; Nin
        delay 2
        goto m1
Priem:
        do
                flag = inp(&H3FD)
                loop while (flag AND 1) <> 1
                Nin = inp(&H3F8)
        return
```

Программа для микроконтроллера состоит из типовых осномных модулей. Команды начальной установки содержат настройки УАПП. В этой программе УАПП работает в режиме 8-битного асинхронный приема и передачи (режим 1). Старшая тетрада TMOD = 0010B. Скорость передачи устанавливается такая же, как в компьютере – 9600 бит/с. Для этого из табл. 25, при частоте резонатора 24 МГц выбираем значение числа, загру-

жаемого в TH1, равное F3h. При этом бит SMOD должен быть установлен в единицу.

Программа основного цикла содержит команду вызова подпрограммы приема (call in) данных. Данные, принятые от компьютера, размещаются в аккумуляторе. Включение светодиода осуществляется логическим нулем, поэтому принятые данные инвертируются (cpl A) и выводятся в порт (mov P2,A). Далее следуют команды: cpl A – восстановление принятого значения; inc A – увеличение на единицу; call out – вызова подпрограммы передачи числа в компьютер.

Программа

Таблица векторов прерываний

```
        org 0000h
        jmp begin
```

Команды начальной установки

```
        org 0040h               ;установка адреса начала программы
begin:  mov SCON,#01011100B     ;установка режима УАПП
        mov PCON,#10000000B     ;удвоенная скорость передачи
        mov TMOD,#00100001B     ;8-битный автоперегр. TMP1
        mov TH1,#0F3h           ;9600 - 24 МГц
        mov TL1,#0F3h
        mov P2,#0FFh            ;гашение светодиодов
        clr EA                  ;запрет всех прерываний
        setb TR1                ;пуск таймера 1
```

Программа основного цикла

```
m1:     call in                 ;прием числа
        cpl A                   ;инверсия аккумулятора
        mov P2,A                ;вывод на светодиоды
        cpl A
        inc A
        call out                ;передача числа
        jmp m1
```

Подпрограммы

```
out:    mov SBUF,A
        jnb ti,$                ;флаг прерывания передатчика
        clr ti
        ret
```

```
in:     jnb ri,$                ;флаг прерывания приемника
        mov A,SBUF
        clr ri
        ret
```

Порядок выполнения работы

1. Изучить программирование последовательного канала микроконтроллера. Особое внимание уделить программированию регистров управления последовательного канала.
2. Написать программу настройки последовательного порта и передачи одного байта и нескольких байт в последовательный канал микроконтроллера. Предусмотреть визуальный контроль вывода байта индикацией на светодиодах.
3. Отладить программу и загрузить в контроллер. Запустить программу на выполнение и наблюдать результаты на компьютере.
4. Изменить режимы работы последовательного канала.
5. Продемонстрировать работу преподавателю.

Содержание отчета

1. Алгоритм программирования последовательного канала.
2. Формат регистра управления для программирования последовательного канала.
3. Схемы алгоритмов и реализующие их программы.

Контрольные вопросы

1. Какой режим работы обеспечивает наибольшую скорость передачи данных?
2. Какие параметры определяют скорость приема/передачи данных?
3. Чем отличаются режимы 2 и 3 работы УАПП?
4. Какой таймер используется при программировании УАПП? Можно ли использовать другой таймер?
5. Что такое синхронный и асинхронный обмен данными? Какой режим работы УАПП обеспечивает синхронную прием/передачу данных?
6. В каком регистре сохраняются принятые данные?
7. Как задается скорость передачи в асинхронном режиме работы?

ПРИЛОЖЕНИЕ

ОПИСАНИЕ МАШИННЫХ КОМАНД

Описание каждой приводимой машинной команды микроконтроллера MCS-51 состоит из предложения языка ассемблера, кода, длины команды в байтах, времени ее выполнения, алгоритма и примера.

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[15-8])
(PC[10-0]):=A10A9A8 II A7A6A5A4A3A2A1A0,
где II - знак конкатенации (сцепление)
Пример:

```
;ДО ВЫПОЛНЕНИЯ КОМАНДЫ ACALL ;(SP)=07H
;метка HT1 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 «С» 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, (B6)=AAH
        ADD A,R6 ;(A)=6DH, (R6)=AAH
                ;(AC)=0, (0=1, (OV)=1
```

2) Ассемблер: ADD A,@Ri ; где i=0,1
Код: 00100111 , 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, (R1)=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 суммировании 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: 00111 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)=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: 00110111 , 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)=1
    ADDC A,80H      ;(A)=F1H, (C)=0, (AC)=1, (OV)=0
```

4) Ассемблер: ADDC A,#data
Код: 00110100 #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 <addrll>

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 команды. Адрес перехода должен находиться 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 A1 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) Ассемблер: ANL A,Rn ; где n=0-7
Код: 01011 rrr , 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: 01010111 , 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: 01010100 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: 01010100 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 логическое «И» над указанными битами. Если бит-источник равен «0», то происходит сброс флага переноса, в противном случае флаг переноса не изменяет своего 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: 01010100 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: 01010100 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>,<смещение>

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 комбинацию 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)+<rel8>, C:=1
Пример:

```
                        ;(A)=97H, (P2)=F0H, (C)=0
    CJNE A,P2,MT3
    …
MT3: CLR  A             ;(A)=97H, (P2)=F0H, (C)=1
                        ;адрес, соответствующий метке MT3 вычисляется, как
                        ; (PC):=(PC)+3+<rel8>
```

2) Ассемблер: CJNE A,#data,<метка>
Код: 10110100 #data rel8
Время: 2 цикла
Алгоритм: (PC):=(PC)+3,
если #data<(A), то (PC):=(PC)+<rel8>, C:=0
если #data8>(A), то (PC):=(PC)+<rel8>, C:=1
Пример:

```
                    ;(A)=FCH, (C)=1
    CJNE  A,#0BFH,MT4
    …
MT4: INC  A         ;(A)=FDH, (C)=1
                    ; (PC):=(PC)+3+<rel8>
```

3) Ассемблер: CJNE Rn,#data,<метка>
Код: 10111rrr #data rel8
Время: 2 цикла
Алгоритм: (PC):=(PC)+3,
если #data<(Rn), то (PC):=(PC)+<rel8>, C:=0
если #data8>(Rn), то (PC):=(PC)+<rel8>, C:=1
Пример:

```
                    ;(R7)=80H, (C)=0
    CJNE  R7,#81H,MT5
    …
MT5: NOP            ;(R7)=80H, (C)=1,
                    ;(PC):=(PC)+3+(rel8)
```

4) Ассемблер: CJNE @Ri,#data,<метка> ; где 1=0,1
Код: 10110111 #data8 rel8
Время: 2 цикла
Алгоритм: (PC):=(PC)+3,
если #data<((Ri)), то (PC)+<rel8>, C:=0
если #data8>((Ri)), то (PC)+<rel8>, C:=1
Пример:

```
                    ;(R0)=41H, (C)=1, (ОЗУ[41])=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)=3ЗH (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 «I», 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 «I», 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 (0=1), то A[7-4] :=A[7-4]+6
Пример:

```
а)          ;(A)=56H, (R3)=67H, (C)=1
    ADDC A,R3
    DA   A  ;(A)=24H, (R3)=67H, (C)=1
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: 00011rrr, где ггг=000-111
Время: 1 цикл
Алгоритм: (Rn):=(Rn)-l
Пример:

```
            ;(R1)=7FH,
            ;(ОЗУ[7F])=40H, (ОЗУ[7F])=00H
    DEC @R1
    DEC Rl
    DEC @R1 ;(R1)=7EH,
            ;(ОЗУ[7F])=3FH, (ОЗУ[7F])=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
```

4) Ассемблер: DEC @Ri ; где 1=0,1
Код: 00010111
Время: 1 цикл
Алгоритм: ((Ri)):=((Ri)-l)
Пример:

```
                ;(R1)=7FH,
                ;(ОЗУ[7F])=40H, (ОЗУ[7F])=00H
      DEC @R1
      DEC Rl
      DEC @R1   ;(R1)=7EH,
                ;(ОЗУ[7F])=3FH, (ОЗУ[7F])=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: (А):=((А)/(В))[15-8],
(В):=((А)/(В))[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 адресу, 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 PI: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)-l,
если ((direct)>0 или (direct)<0), то (PC):=(PC)+<rel8>

Пример:

```
                    ;(ОЗУ[40])=01H, (ОЗУ[50])=80H,
                    ;(ОЗУ[б0])=25H
      DJNZ 40H, LAB1 ;(ОЗУ[40]) =00H
      DJNZ 50H, LAB2 ;(ОЗУ[50]) =7FH
      DJNZ 60H, LAB3 ;(ОЗУ[60]) =25H
LAB1: CLR A
LAB2: DEC R1        ;осуществился переход на метку LAB2
```

Примечание: Если команда DJNZ используется для изменения выхода порта, значение, используемое как операнд, 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:             00001rrr, 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,l
Код:             00000111, где i=0,l
Время:           1 цикл
Алгоритм:        ((Ri)):=((Ri))+l
Пример:
                 ;(R1)=41H, (ОЗУ[41])=4FH, (AC)=0
      INC @Rl    ;(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: 10100011
Время: 2 цикла
Алгоритм: (DPTR):=(DPTR)+1
Пример:

```
                ;(DPH)=12H, (DPL)=FEH
    INC  DPTR
    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 (rel8) 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)=l, 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)=l, то (bit):=0, (PC):=(PC)+<rel8>
Пример: (A)=76H (0111 0110B)

```
JBC ACC.2.LAB6    ;Перехода на LAB6 нет, т.к. (A[3]) = 0
JBC ACC.1,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)+<rel8>
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^{10}, 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: 01110011
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)+<rel8>
Пример:

```
                        ;(P2)=CAH (11001010B),
                        ;(A)=56H (0101 0110B)
    JNB  P1.3,LAB10     ;нет перехода на LAB10
    JNB  ACC.3,LAB7     ;переход на метку LAB7
LAB7: 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)+<rel8>
Пример:

```
                    ;(C)=1
    JNC LAB6        ;нет перехода на LAB6
    CPL C
LAB6: 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 «I», в противном случае выполняется следующая команда. Адрес ветвления вычисляется сложением относительного смещения со знаком во втором байте команды (rel8) 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)+<rel8>
Пример:

```
        ;(A)=00H
    JNC LAB14 ;нет перехода на LAB14
    INC A
LAB14: JNZ LAB15 ;переход на метку LAB15
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 (re18) 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 <addrl6>

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 полу-

ченный 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[l5-8])
(PC):=<adddr[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 <addrl6>

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[l5-8] 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: 11101rrr , 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 1=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: 11111rrr , где rrr=000-lll
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: 10101rrr 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: 01111rrr #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: 10001rrr direct address , где rrr=000-lll
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,l
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[5Г])=9ВН
    MOV 5FH,#07H  ;(ОЗУ[5F])=07H
```

13) Ассемблер: MOV @Ri,A ; где i=0,l
Код: 11110111 , где 1=0,1
Время: 1 цикл
Алгоритм: ((Ri)):=(A)
Пример:

```
                    ;(R1)=48H, (ОЗУ[48])=75H, (A)=0BDH
    MOV @R1,A ;(ОЗУ[48])=0BDH
```

14) Ассемблер: MOV @Ri,<direct> ; где 1=0,1
Код: 10100111 direct address
Время: 2 цикла
Алгоритм: ((Ri)):=(direct)
Пример:

```
                    ;(R0)=51H, (ОЗУ[51])=0ЕЗН, (P0)=0ACH
    MOV@R0.P0 ;(ОЗУ[51])=0ACH
```

15) Ассемблер: MOV ©Ri,#data ; где 1=0,1
Код: 01110111
Время: 1 цикл
Алгоритм: ((Ri)):=<#data8>
Пример:

```
                    ;(ОЗУ[7E])=67H, (R1)=7EH
    MOV @R1, #0A9H    ;(ОЗУ[7E])=0A9H,(P1)=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.#data16

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[l5-8] #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-битовьд 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):=((R1))

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 1=0,1
Код: 11110011
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: 1111 0000
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 DEC), (C)=1,
                ;(B)=0A0H (A0H=160 DEC), (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-lll
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: 01000111
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 С, если булева величина равна логической «I», в противном случае устанавливает флаг С 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[25])=39H (00111001B)
        ORL C,/2A ;(C)=1, (ОЗУ[25])=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[0D])=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 восстанавливает-

ся 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)=УУУУН, где У=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+l]):=(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[71)

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+l]), 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+ll), 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):=l
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)+(rel8)
Пример:

```
        ;(PC)=0418H,
        ;метка MET1 соответствует адресу 039AH
    SJMP MET1   ;(PC)=039AH, где (rel8)=80H=-128 DEC
    SJMP MET2   ;(PC)=041AH, где метка MET2 соответствует
        ;адресу 041AH,
        ;(rel8)=7DH=+125 DEC
```

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: 10011rrr , 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)=1
        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, гле 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 1=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[7-41),
(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: 11001rrr , где rrr=000-lll
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,l
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: 11010111
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: 01101rrr, 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,PI    ;(A)=A9H, (P1)=0A6H
```

3) Ассемблер: XRL A,@Ri ; где i=0,l
Код: 01100111
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.5.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Сташин В.В., Урусов А.В., Мологонцева О.Ф. Проектирование цифровых устройств на однокристальных микроконтроллерах. – М.: Энергоатомиздат, 1990. – 224с.

2. Григорьев В.Л. Программирование однокристальных микропроцессоров. – М.: Энергоатомиздат, 1987. – 288 с.

3. Веселов О.В., Мишулин Ю.Е., Немонтов В.А., Кобзев А.А. Микропроцессорные устройства в системах автоматизации: Учеб. пособие: В 2 ч. Ч.1 / Владим. гос. ун-т, Владимир, 2003. 128 с. ISBN 5-89368-415-X

4. Мишулин Ю.Е. Цифровая схемотехника: Учеб. пособие. / Ю.Е. Мишулин, В.А.Немонтов; Владим. гос. ун-т. – Владимир: Изд-во Владим. гос. Ун-та, 2006. – 142 с. ISBN 5-89368-649-7

5. Злобин В.К., Григорьев В.Л. Программирование арифметических операций в микропроцессорах: Учеб. пособие для технических вузов. – М.: Выс. шк., 1991. – 303 с.

6. Науман Г., Майлинг В., Щербина А. Стандартные интерфейсы для измерительной техники: Пер. с нем. – М.: Мир, 1982. – 304 с.

7. Коффрон Дж. Технические средства микропроцессорных систем: Практ. курс: Пер. с англ. – М.: Мир, 1983. – 344 с.

8. http://mcu.cахара.ru. Сайт индекса популярности микроконтроллеров.

МИКРОПРОЦЕССОРНЫЕ СРЕДСТВА И СИСТЕМЫ
Лабораторный практикум

Составители
МИШУЛИН Юрий Евгеньевич

Подписано в печать . . .
Формат 60×84/16. Бумага для множит. техники. Гарнитура Таймс.
Печать на ризографе. Усл. печ. л. , . Уч.-изд. л. , . Тираж 150 экз.
Заказ .
Издательство
Владимирского государственного университета.
600000, Владимир, ул. Горького, 87.