Министерство образования и науки Украины

Донбасская государственная машиностроительная академия

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к самостоятельной работе

по дисциплине

«Микропроцессорные устройства»

для студентов специальности 7.092203

«Электромеханические системы автоматизации»

всех форм обучения

Утверждено
на заседании
методического совета
Протокол № от

Краматорск 2008

УДК

Методические указания к самостоятельной работе по дисциплине «Микропроцессорные устройства» для студентов специальности 7.092203 «Электромеханические системы автоматизации» всех форм обучения / Сост.: А. М. Наливайко, Д. С. Пономарев. – Краматорск: ДГМА, 2006. – 132 с.

В методических указаниях изложено краткое описание микроконтроллеров I8051, PIC16F877, основные методы и приемы программирования на языках ассемблера семейств MCS-51 и PICmicro. Приведены примеры составления программ и задания для самостоятельной работы студентов.

Составители: А. М. Наливайко, доц.,
Д. С. Пономарев, асс.

Отв. за выпуск А. М. Наливайко, доц.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Применение микроконтроллеров, начинавшееся в прошлом столетии с цифровых часов и микрокалькуляторов, в настоящее время продолжает расширяться и оправдано не только малыми размерами, весом и энергопотреблением при высокой надежности и низкой стоимости этих микросхем. Главной причиной эффективности использования микроконтроллеров является их функциональная универсальность, которая обусловлена возможностью их программирования.

Программирование неразрывно связано с компьютерами, хотя за последние десятилетия проделана большая работа, чтобы эта связь стала более опосредованной. Стремление к переносимости и удобству пользовательского интерфейса возносит прикладные программы во все более высокие слои математического обеспечения, отделяемые от аппаратно-зависимых программ растущим количеством оболочек. Такое усложнение математического обеспечения в свою очередь требует все больших аппаратных ресурсов. Прогресс технических характеристик микропроцессоров и связанных с ними наборов микросхем вызывает с одной стороны восхищение, а с другой – тревогу. Моральное старение персональных компьютеров значительно опережает их физический износ. Хорошо еще, что благодаря микроминиатюризации этот прогресс не требует чрезмерных затрат материальных ресурсов, но вовлекаемые финансовые средства огромны. Экспоненциальный рост не может продолжаться безгранично. Можно только гадать, каковы будут результаты столкновения растущих аппетитов потребителей с неизбежными технологическими ограничениями прогресса в разработке компьютеров.

По сравнению с персональными компьютерами микроконтроллеры не так известны, потому что скромно прячутся внутри самых разнообразных устройств, в том числе и в компьютерах, в которых без микроконтроллеров не могло бы работать ни одно из периферийных устройств, таких как клавиатура, мышь, дисковод, принтер, сканер и т.д. Да и программирование для микроконтроллеров не является престижным по сравнению с программированием для персональных компьютеров. Но эта работа не менее важна и интересна, хотя и не столь заметна широкой публике. Особенности программирования для микроконтроллеров, с одной стороны, связаны с ограничениями вычислительных ресурсов, поскольку аппаратура должна стоить как можно дешевле. С другой стороны, программист должен достаточно хорошо знать физику работы изделия, в котором используется микроконтроллер, поскольку разрабатываемые программы должны работать в реальном мире. В силу этих причин программы для микроконтроллеров являются штучным товаром, то есть непереносимы с одного типа изделия на другое, хотя тираж этих изделий может быть огромным. Ведь количество выпускаемых микроконтроллеров в десятки раз превышает производство микропроцессоров.

1 ОПИСАНИЕ МИКРОКОНТРОЛЛЕРА I8051

1.1 Организация памяти микроконтроллера

Микроконтроллеры семейства Intel MCS-51 выполнены по гарвардской архитектуре и имеют раздельную память команд и данных. Такая организации памяти определяет некоторые особенности при программировании микроконтроллеров данного типа. Кроме этого к микроконтроллеру можно подключить внешнюю память команд и данных. Внутренняя память команд и данных организована непосредственно в самом микроконтроллере на одном кристалле.

На рис. 1 изображена карта памяти команд микроконтроллера. Объем внутренней (резидентной) памяти программ (ROM, EPROM или OTP ROM), располагаемой на кристалле, в зависимости от типа микросхемы может составлять 0 (ROMless), 4К (базовый кристалл), 8К, 16К или 32К. При необходимости пользователь может расширять память программ установкой внешнего ПЗУ. Разрешение работы с внутренним или внешним ПЗУ определяется значением логического сигнала на выводе EA (External Access):

EA=1 – доступ к внутреннему ПЗУ;

EA=0 – доступ к внешнему ПЗУ.

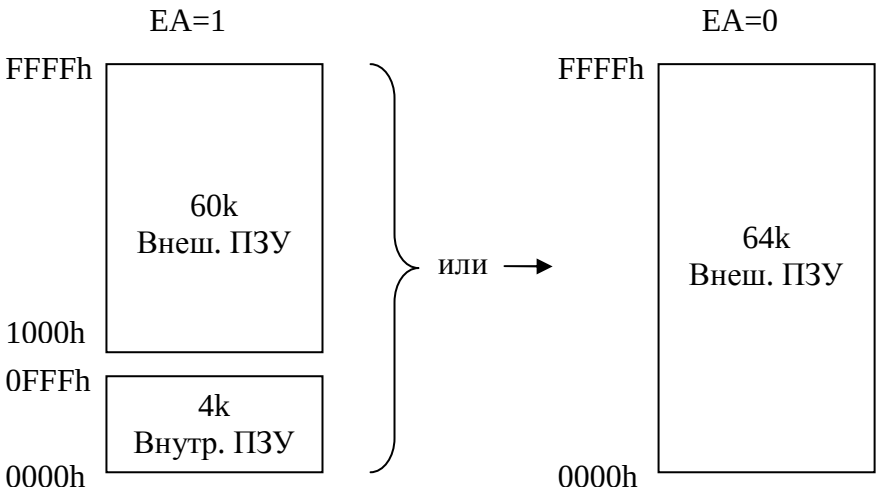


Рисунок 1 – Организация памяти команд

Область нижних адресов памяти программ используется системой прерываний. Архитектура микросхемы i8051обеспечивает поддержку пяти источников прерываний:

– двух внешних прерываний;

– двух прерываний от таймеров;

– прерывания от последовательного порта.

Распределение нижних адресов памяти команд указано на рис. 2. Данные адреса называют «Векторами прерываний». При подаче сигнала сброса RESET микроконтроллер выполняет начальную инициализацию и начинает извлекать команды по адресу 0000h. Поэтому, если предусматривается применять систему прерываний, по данному адресу следует записывать команду безусловного перехода на адрес 002Bh или выше. Это необходимо для обхода основной программой таблицы векторов прерываний.

При появлении соответствующего события, например переполнения таймера 0, и условии разрешения прерывания для данного события, основная программа прерывается и происходит извлечение команд по соответствующему адресу (для таймера 0 – 000Bh). Более подробно с системой прерываний можно познакомиться в главе 1.3 «Организация системы прерываний».

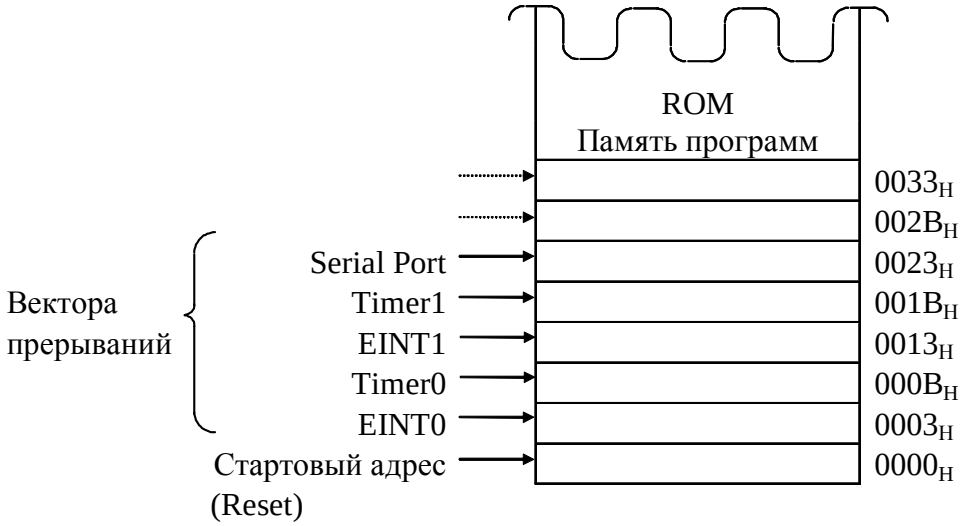


Рисунок 2 – Нижняя область памяти команд

Память данных отделена от памяти программ. В этой области возможна адресация 64К внешнего ОЗУ. При обращении к внешней памяти данных ЦП микроконтроллера генерирует соответствующие сигналы чтения $\overline{RD}$ или записи $\overline{WR}$. Взаимодействие с внутренней памятью данных осуществляется на командном уровне, при этом сигналы $\overline{RD}$ и $\overline{WR}$ не вырабатываются.

Внешняя память программ и внешняя память данных могут комбинироваться путем совмещения сигналов $\overline{RD}$ и $\overline{PSEN}$ по схеме «логического И» для получения строба доступа к внешней памяти (программ/данных).

Внутренняя память данных разделена на 2 участка: нижняя область памяти – lower memory (адреса 00h–7Fh) содержит РОН, которые выбираются из 4х банков и область памяти пользователя (Рис. 5); верхняя область памяти – upper memory (адреса 80h–FFh) содержит регистры

специальных функций (SFR). Данную область памяти нельзя использовать для хранения информации пользователя, и она используется только для доступа к регистрам SFR (Рис. 4). Организация памяти данных указана на рис. 3.

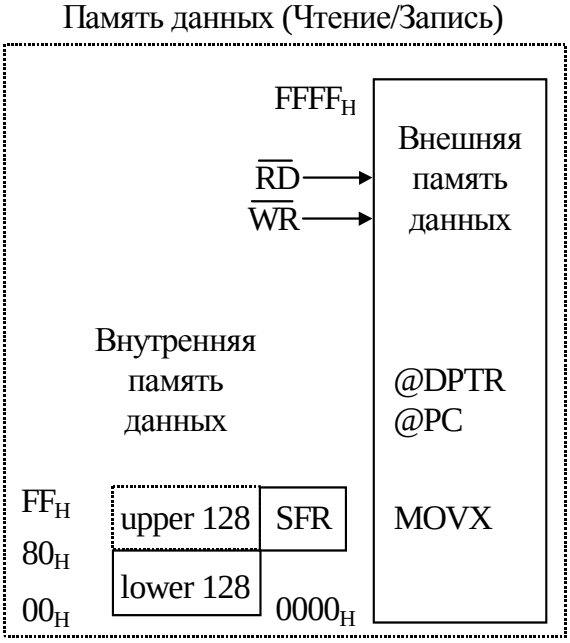


Рисунок 3 – Организация памяти данных

Первые 32 байта памяти lower memory представляют собой 4 банка (Register Bank) по 8 регистров R7...R0. Регистры R0 и R1 в любом из банков могут использоваться в качестве регистров косвенного адреса.

Следующие за регистровыми банками 16 байт образуют блок побитно-адресуемого пространства. Набор инструкций MCS-51 содержит широкий выбор операций над битами, а 128 бит в этом блоке адресуются прямо и адреса имеют значения от 00H до 7FH.

Все байты в нижней 128-байтной половине памяти могут адресоваться как прямо, так и косвенно.

Верхняя 128 байтная половина памяти ОЗУ (Upper 128) в микросхеме i8051 отсутствует, но имеется в версиях кристаллов с 256 байтами ОЗУ. В этом случае область «Upper 128» доступна только при косвенной адресации. Область SFR (Special Function Register) доступна только при прямой адресации.

Размещение регистров специальных функций в пространстве SFR показано на рис. 4. Они включают в себя регистры портов, таймеры, средства управления периферией и т. п.

Для 16 адресов в пространстве SFR имеется возможность как байтовой, так и битовой адресации. Для побитно адресуемых регистров шестнадцатеричный адрес заканчивается на «0H» или на «8H». Битовые адреса в этой области имеют значения от 80H до FFH.

	Битовая адресация	8 байт							
$F8_H$									FF_H
$F0_H$	B								$F7_H$
$E8_H$									EF_H
$E0_H$	ACC								$E7_H$
$D8_H$									DF_H
$D0_H$	PSW								$D7_H$
$C8_H$									CF_H
$C0_H$									$C7_H$
$B8_H$	IP								BF_H
$B0_H$	P3								$B7_H$
$A8_H$	IE								AF_H
$A0_H$	P2								$A7_H$
98_H	SCON	SBUF							$9F_H$
90_H	P1								97_H
88_H	TCON	TMOD	TL0	TL1	TH0	TH1			$8F_H$
80_H	P0	SP	DPL	DPH				PCON	87_H
	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

Рисунок 4 – Размещение регистров специальных функций в пространстве SFR

$7F_H$	Побайтно-адресуемая область ОЗУ (direct, indirect)							
30_H								
$2F_H$	$7F_H$	$7E_H$	$7D_H$	$7C_H$	$7B_H$	$7A_H$	79_H	78_H
$2E_H$	77_H	76_H	75_H	74_H	73_H	72_H	71_H	70_H
	Побитно-адресуемая область ОЗУ (direct)							
21_H	$0F_H$	$0E_H$	$0D_H$	$0C_H$	$0B_H$	$0A_H$	09_H	08_H
20_H	07_H	06_H	05_H	04_H	03_H	02_H	01_H	00_H
$1F_H$	Банк РОН 3							
18_H								
17_H	Банк РОН 2							
10_H								
$0F_H$	Банк РОН 1							
08_H								
07_H	⇐ **SP** после RESET							
00_H	Банк РОН 0(R7+R0)							

Рисунок 5 – Нижняя область внутренней памяти данных

Регистры SFR имеют следующее назначение:

Аккумулятор A (Accumulator, адрес $E0_H$).

Команды архитектуры MCS-51 используют аккумулятор как источник и как приемник при вычислениях и пересылках. Кроме обращения к аккумулятору командами, использующими мнемонику "A", имеется возможность побитовой или побайтовой адресации, как SFR-регистра.

Регистр B (Multiplication Register, адрес $F0_H$).

Регистр B используется как источник и как приемник при операциях умножения и деления, обращение к нему, как к регистру SFR, производится аналогично аккумулятору.

Слово состояния программы PSW (Program Status Word, адрес $D0_H$).

Данный регистр содержит биты, отражающие результаты исполнения операций, биты выбора регистрового банка и бит общего назначения, доступный пользователю. PSW отображен на область SFR и содержит:

Биты:	7	6	5	4	3	2	1	0
Обозначение:	CY	AC	F0	RS1	RS0	OV	-	P

Рисунок 6 – Расположение битов регистра PSW

Биты регистра PSW имеют следующее назначение:

CY (Carry) – признак переноса. Устанавливается, если в старшем бите аккумулятора при выполнении арифметической операции в результате возникает перенос бита или заем. При выполнении операций деления или умножения бит сбрасывается. Установка/сброс – аппаратно и программно;

AC (Auxiliary Carry) – признак дополнительного переноса. Используется при выполнении инструкций сложения или вычитания для указания переноса или заема из бита 3 при образовании младшего полубайта результата (D0-D3). Установка/сброс – аппаратно и программно;

F0 – флаг состояния, определяемый пользователем. Установка/сброс только программно;

RS0, RS1 – флаги-указатели банка рабочих регистров (РОН). Установка/сброс только программно;

OV (Overflow) – флаг переполнения. Устанавливается, если результат операции сложения/вычитания не укладывается в 7 битах и старший восьмой бит не может быть интерпретирован как знаковый. При выполнении операции деления флаг сбрасывается, а в случае деления на 0 устанавливается. При умножении флаг устанавливается, если результат больше 255. Возможен программный сброс/установка;

P (Parity) – флаг четности единичных битов в аккумуляторе. Является дополнением содержимого аккумулятора до четности. В 9-разрядном слове, состоящем из 8 разрядов аккумулятора и бита P всегда содержится четное число единичных битов. Если все биты аккумулятора равны 0, то и бит P сброшен. Он программно доступен только для чтения.

Разряд 1 PSW зарезервирован и может использоваться для программной записи/чтения.

Регистры портов P0-P3 (адреса 80_H, 90_H, $A0_H$, $B0_H$).

Каждый порт является фиксатором-защелкой и может адресоваться как побайтно, так и побитно. Помимо работы в качестве обычных портов ввода/вывода, линии портов могут выполнять ряд альтернативных функций:

– Через порт 0 (в мультиплексном режиме) выводится младший байт адреса, а также выдается и принимается в микроконтроллер байт данных при работе с внешней памятью программ/данных;

– Через порт 2 выводится старший байт адреса внешней памяти программ и данных;

– Порт 3 имеет следующие альтернативные функции:

P3.7 – строб чтения из внешней памяти данных (Read Data from External Memory, $\overline{RD}$);

P3.6 – строб записи во внешнюю память данных (Write Data to External Memory, $\overline{WR}$);

P3.5 – внешний вход T/C1 (Timer/Counter 1 External Input, T1);

P3.4 – внешний вход T/C0 (Timer/Counter 0 External Input, T0);

P3.3 – вход внешнего прерывания 1 (External Interrupt 1 Input Pin, $\overline{INT1}$);

P3.2 – вход внешнего прерывания 0 (External Interrupt 0 Input Pin, $\overline{INT0}$);

P3.1 – выход данных передатчика последовательного порта (Serial Port Transmit Pin, TxD);

P3.0 – вход данных передатчика последовательного порта (Serial Port Receive Pin, RxD).

Указатель стека SP (Stack Pointer, адрес 81_H).

Используется для указания на вершину стека в операциях записи в стек и чтения из него. Неявно используется такими командами, как PUSH, RET, RETI, POP. По аппаратному сбросу от ЦП устанавливается в значение 07_H (область стека в этом случае начинается с адреса внутренней памяти данных 08_H) и инкрементируется при каждой записи в стек. Запись в SFR-регистр SP (с использованием байтовой адресации) производится для предопределения положения стека во внутренней памяти данных.

Указатель данных DPTR (Data Pointer, адреса 82_H, 83_H).

Команды архитектуры MCS-51 используют DPTR для пересылки данных, пересылки кода и для переходов (JMP@A+DPTR). DPTR состоит из двух регистров: младшего – DPL и старшего – DPH, обращение к ним – только байтовое.

Регистр управления энергопотреблением PCON (Power Control Register, адрес 87_H).

Для кристаллов, выполненных по HMOS-технологии, данный регистр имеет только один значащий бит – SMOD, который управляет скоростью работы последовательного порта.

Регистры таймеров/счетчиков TL0, TL1, TH0, TH1 (адреса $8A_H$, $8B_H$, $8C_H$, $8D_H$).

Образуют 16-битные (Low/High) регистры таймеров/счетчиков T/C0 и T/C1. Обращение к регистрам только байтовое. Подробно описаны в подразделе «Организация таймеров/счетчиков».

Регистр режима таймеров/счетчиков TMOD (Timer/Counter Mode Control Register, адрес 89_H).

Регистр управления таймеров/счетчиков TCON (Timer/Counter Control Register, адрес 88_H).

Предназначены для управления работой таймерной секции микроконтроллера. Подробно описаны в подразделе «Организация таймеров/счетчиков».

Буфер последовательного порта SBUF (Serial Data Buffer, адрес 99_H).

Представляет собой два отдельных регистра. При записи в SBUF загружается «буфер передачи» последовательного порта, при чтении SBUF считывается содержимое «буфера приема» последовательного порта.

Регистр управления последовательным портом SCON (Serial Port Control Register, адрес 98_H)

Предназначен для управления работой последовательного порта. Обращение к данному регистру может быть как байтовым, так и побитным.

Регистр разрешения прерываний IE (Interrupt Enable Register, адрес $A8_H$).

Регистр управления приоритетом прерывания IP(Interrupt Priority Control Register, адрес $B8_H$).

Поддерживают работу системы прерываний микроконтроллера. Подробное описание работы с регистрами дано в подразделе «Организация системы прерываний».

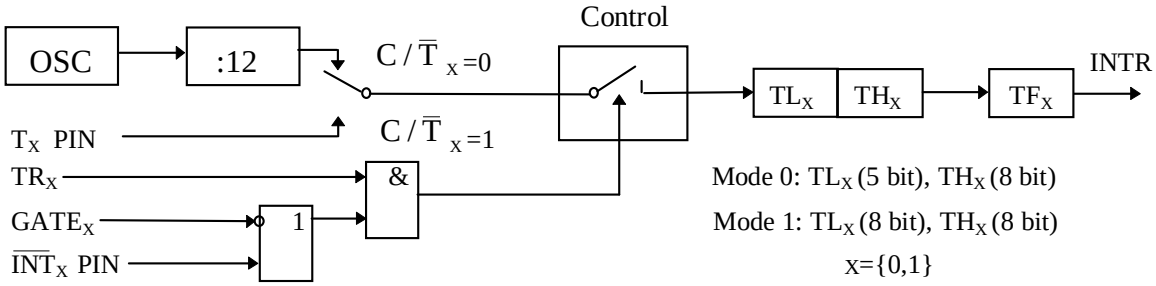
1.2 Организация таймеров/счетчиков

Таймеры/счетчики (T/C0 и T/C1) предназначены для подсчета внешних событий (выводы T0 и T1), организации программно-управляемых временных задержек и измерения временных интервалов. Таймер 1 может также служить генератором скорости передачи для последовательного порта.

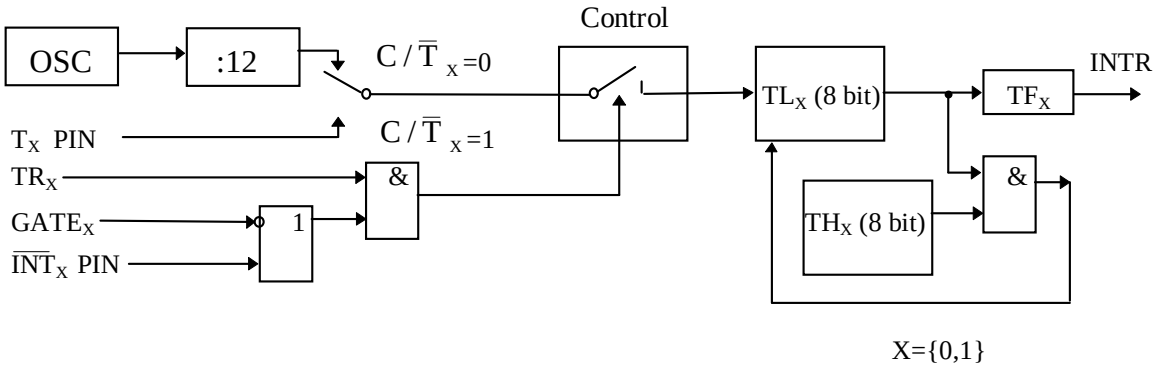
Таймер/счетчик, работая в режиме таймера, ведет подсчет тактов деленной системной частоты (запрограммированный промежуток времени) и выдает запрос прерывания. Регистр таймера инкрементируется один раз в каждом периферийном цикле. Поскольку цикл состоит из 12 тактов, то скорость счета таймера равна $F_{OSC}/12$.

В режиме счетчика регистр таймера ведет подсчет (предустановленного числа событий) отрицательных перепадов сигнала на внешнем входе и по окончании счета выдает запрос прерывания.

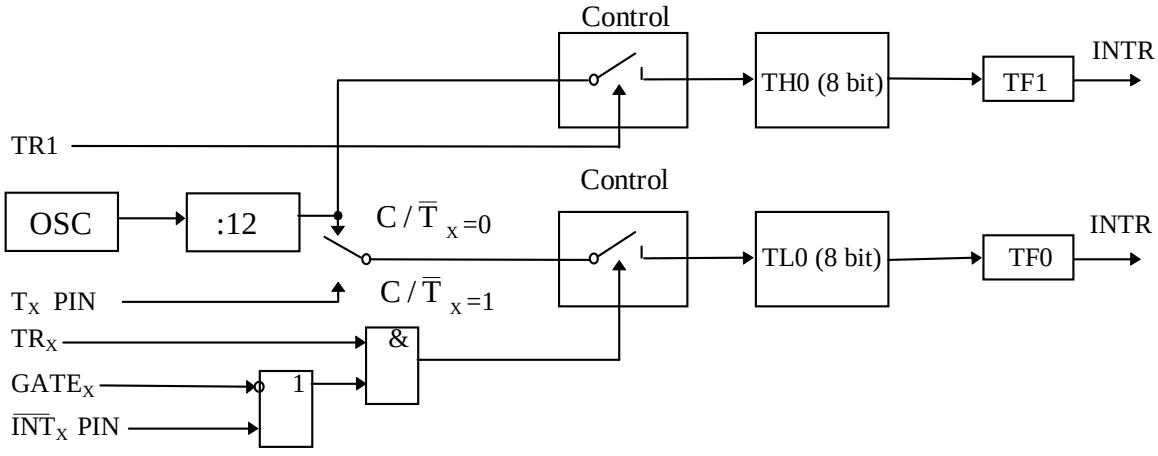
Поскольку распознавание отрицательного перехода внешнего сигнала занимает 24 периода тактовой частоты (2 цикла), то максимальная скорость счета равна $F_{OSC}/24$. Ограничений на рабочий цикл не накладывается, но чтобы гарантировать опрос конкретного уровня сигнала хотя бы один раз до момента его смены, он должен удерживаться на входе хотя бы в течение одного полного периферийного цикла.



а - логика работы T/C0 и T/C1 в режимах 0 и 1



б - логика работы T/C0 и T/C1 в режиме 2



в - логика работы T/C0 в режиме 3

Рисунок 7 – Логика работы T/C0 и T/C1в режимах 0, 1, 2 и 3

Программное управление функционированием T/C0 и T/C1 обеспечивают SFR-регистры TMOD и TCON. Возможны 4 режима работы

Т/С микроконтроллера, которые определяются установкой соответствующих битов регистра TMOD. Режимы 0 (13-битовый таймер), 1 (16-битовый таймер) и 2 (8-битовый таймер с автоперезагрузкой) полностью идентичны для обоих Т/С. В режиме 3 (два 8-битовых регистра) работает только Т/С0, Т/С1 в этом режиме заблокирован («лишен» бита управления запуском TR1 и флага переполнения TF1) и сохраняет содержимое своих регистров TL1 и TH1. Логика работы Т/С0 и Т/С1 в режимах 0,1,2,3 показана на рис. 7. Путем соответствующего программирования регистров TMOD и TCON осуществляется включение и выключение таймеров/счетчиков, выбор источника их тактирования и установка определенного режима их работы.

Функциональное назначение разрядов этих регистров следующее:

Биты:	7	6	5	4	3	2	1	0
Обозначение:	GATE1	$C/\overline{T}1$	M1.1	M1.0	GATE0	$C/\overline{T}0$	M0.1	M0.0

Рисунок 8 – Расположение битов регистра TMOD

Назначение битов:

GATEx – Если GATEx=1 и TRx=1, то включение и выключение соответствующего таймера осуществляется внешним сигналом на входе INTx. Когда GATEx=0, бит управления запуском TRx=1 разрешает прохождение входных сигналов от выбранного источника тактирования;

$C/\overline{T}x$ – выбирает функцию таймера, (подсчет импульсов деленной системной частоты) или выбирает функцию счетчика (подсчет отрицательных переходов сигнала на внешнем выводе Tx);

Mx.1, Mx.0 – биты выбора режима таймеров. Могут иметь следующие сочетания:

Mx.1	Mx.0	
0	0	Mode 0: 8-битовый таймер/счетчик (THx) с 5-битовым предделителем (TLx);
0	1	Mode 1: 16-битовый таймер/счетчик;
1	0	Mode 2: 8-битовый автоперезагружаемый таймер/счетчик (TLx). Константа перезагрузки предварительно заносится в THx.
1	1	Mode 3: TL0 – это 8-битовый таймер/счетчик; TH0 – 8-битовый таймер, использующий биты TR1 и TF1.

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

Рисунок 9 – Расположение битов регистра TCON

Назначение битов:

TRx – биты запуска соответствующего таймера/счетчика. Устанавливаются/сбрасываются только программно;

TFx – биты переполнения соответствующего счетчика (устанавливаются/сбрасываются программно и аппаратно при переходе значения таймера из $FFFF_H$ в 0000_H), очищается аппаратно, когда процессор переходит на подпрограмму обработки прерывания или программно;

IEx – флаг обнаружения внешнего прерывания. Устанавливается аппаратно, когда обнаружено внешнее прерывание (по фронту или уровню сигнала) на выводе INTx; сбрасывается аппаратно во время обработки прерывания только в том случае, когда прерывание было вызвано фронтом сигнала или программно;

ITx – выбор типа сигнала для обнаружения внешнего прерывания. Определяет тип воспринимаемого сигнала на входе INTx; для выбора срабатывания по фронту сигнала (высокий и низкий) нужно установить этот бит, для срабатывания по уровню (активный низкий уровень) нужно сбросить этот бит.

Пример настройки таймеров/счетчиков в режим 2:

```
 mov  TMOD, #20H     ; T/C1 в режиме 2
 mov  TL1, #data8    ; константы перезагрузки
 mov  TH1, #data8    ; таймера/счетчика 1
SETB  TR1            ; запуск таймера/счетчика на счет
```

1.3 Организация системы прерываний

Архитектура системы управления прерываниями для базовой модели I8051 показана на рис. 10:

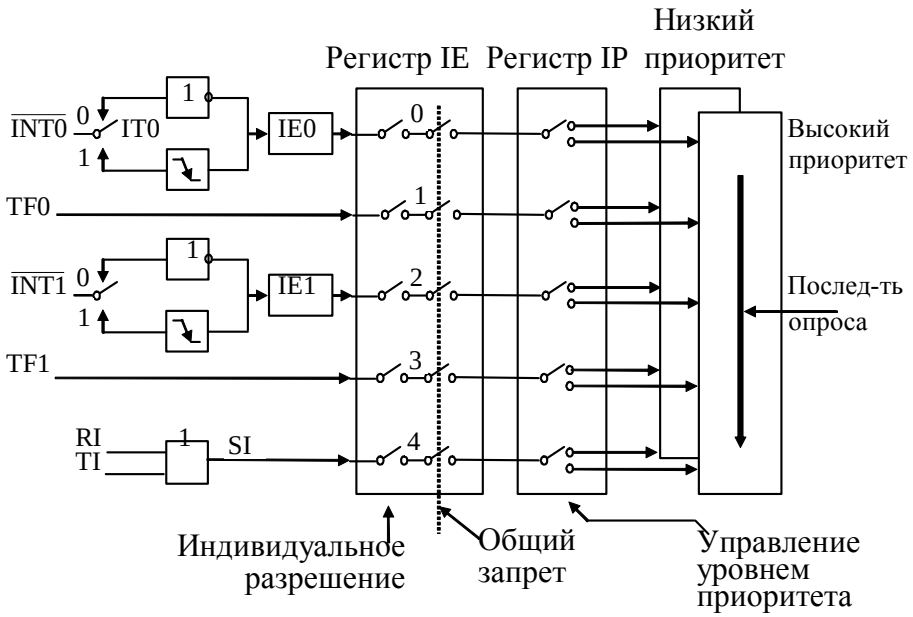


Рисунок 10 – Система прерываний ОМЭВМ

Каждое из внешних прерываний INT0, INT1 может быть активизировано по уровню «0» или по фронту (переход из «1» в «0») сигналов на выводах ОМЭВМ P3.2, P3.3, что определяется состоянием битов IT0 и IT1 регистра TCON. При поступлении запроса внешнего прерывания INTx устанавливается флаг IEx регистра TCON. Установка флагов IEx в регистре TCON вызывает соответствующее прерывание. Очистка флага IEx производится следующим образом. При прерывании по фронту IEx сбрасывается аппаратно (автоматически внутренними средствами ОМЭВМ) при обращении к соответствующей подпрограмме обработки прерывания. При прерывании по уровню флаг очищается при снятии запроса внешнего прерывания, то есть в IEx отслеживается состояние вывода INTx.

Прерывания от таймеров/счетчиков вызываются установкой флагов TF0 и TF1 регистра TCON, которые устанавливаются при переполнении соответствующих регистров таймеров/счетчиков (за исключением режима 3, см. раздел «Организация таймеров/счетчиков»). Отчистка флагов TF0 и TF1 производится внутренней аппаратурой ОМЭВМ при переходе к подпрограмме обслуживания прерывания.

Прерывание от последовательного порта вызывается установкой флага прерывания приемника RI или флага прерывания передатчика TI в регистре SCON. В отличие от всех остальных флагов, RI и TI сбрасываются только программным способом обычно в пределах подпрограммы обработки соответствующего прерывания.

Каждый из перечисленных источников прерываний может быть индивидуально разрешен или запрещен установкой или сбросом соответствующего бита в регистре разрешения прерываний IE. Регистр IE содержит также бит EA, сброс которого в «0» запрещает сразу все прерывания. Необходимым условием прерывания является его разрешение в регистре IE. Формат и описание регистра разрешения прерываний приведены ниже:

Биты:	7	6	5	4	3	2	1	0
Обозначение:	EA	-	-	ES	ET1	EX1	ET0	EX0

Рисунок 11 – Расположение битов регистра IE

Назначение битов:

EA (Enable All) – разрешение прерываний от всех источников;

Биты 6, 5 – зарезервированы;

ES (Enable Serial) – разрешение прерывания от последовательного порта;

ETx (Enable T/Cx) – разрешение прерывания по переполнению от соответствующего таймера;

EXx (Enable eXternal) – разрешение прерывания по внешнему сигналу на входе $\overline{INTx}$;

Все биты, которые вызывают прерывания (IE0, IE1, TF0, TF1, RI, TI), могут быть программно установлены или сброшены с тем же результатом, что и в случае их аппаратной установки или сброса. Т. е. прерывания могут программно вызываться или ожидающие обслуживания прерывания могут программно ликвидироваться. Кроме того, прерывания по входам $\overline{INT0}$, $\overline{INT1}$ могут вызываться программной установкой P3.2=0 и P3.3=0, как показано в приведенном ниже примере:

```
MAIN:     MOV  IE, #00000101B   ; Разрешение прерывания от INT0,INT1.
          MOV  IP, #04H         ; Присвоение INT1 старшего приоритета.
         SETB  EA               ; Общее разрешение прерываний.
          M0V  P3, #11110011B   ; Имитация внешних прерываний.
SUBR:                           ; Переход к подпрограмме обслуживания
   ORG 013H                     ; INT1.
```

В предложенном примере запросы прерывания INT0 и INT1, имеющие различный приоритет, поступают одновременно. При этом обслуживается прерывание с высшим приоритетом.

В случае, когда прерывание по $\overline{INTx}$ (x=0,1) вызывается уровнем сигнала на соответствующем входе ОМЭВМ, флаг IEx при переходе к подпрограмме обработки прерывания автоматически сбрасывается, а затем, если соответствующий вывод ОМЭВМ P3.2 или P3.3 все еще находится в состоянии логического «0», вновь устанавливается. Поэтому, в случае, когда прерывание по входам $\overline{INT0}$, $\overline{INT1}$ вызывается уровнем, программная установка в «1» флагов IE0, IE1 вызовет прерывание, после чего соответствующий флаг IEx (x=0,1) будет автоматически сброшен при переходе к подпрограмме обработки прерывания.

Флаги IE0, IE1, TF0, TF1, RI, TI устанавливаются независимо от того разрешено или нет соответствующее прерывание в регистре IE.

Структура приоритетов прерываний является двухступенчатой. Каждому источнику прерывания может быть индивидуально присвоен один из двух уровней приоритета: высокий или низкий. Выполняется это установкой (высокий уровень приоритета) или сбросом (низкий уровень приоритета) соответствующего бита в регистре приоритетов прерываний IP:

Биты:	7	6	5	4	3	2	1	0
Обозначение:	-	-	-	PS	PT1	PX1	PT0	PX0

Рисунок 12 – Расположение битов регистра IP

Назначение битов:

Биты 7, 6, 5 – зарезервированы;

PS (Priority of Serial) – Установка уровня приоритета от последовательного порта;

PTx (Priority of T/Cx) – Установка уровня приоритета от соответствующего T/C;

PXx (Priority of External) – Установка уровня приоритета от соответствующего внешнего источника прерываний.

Низкоприоритетное прерывание может прерываться высокоприоритетным, но никогда не прерывается запросом того же уровня приоритета. Поэтому, если одновременно возникают два прерывания с различным уровнем приоритета, то сначала выполняется высокоприоритетное. Если же подобная ситуация складывается для прерываний с одинаковым уровнем приоритета, то последовательность их обработки определяется специальной последовательностью опроса флагов прерываний (Interrupt Polling Sequence): IE0→TF0→IE1→TF1→RI+TI. Запрос IE0 – распознается первым, TI+RI – последним.

При выполнении прерывания основной программы для выполнения подпрограммы обработки прерывания внутри микропроцессора выполняется аппаратно-реализуемая команда LCALL, выполняющая переход по фиксированному адресу, значение которого зависит от источника прерывания. Данный адрес называют «вектором прерывания». Адреса векторов прерываний для базовой модели I8051 указаны на рис. 2. При этом подпрограмма будет выполняться до команды возврата из прерывания RETI. Данная команда снимает запрет на обработку других прерываний и загружает в счетчик команд из стека адрес следующей команды прерванной программы.

Программисту при использовании системы прерываний необходимо следить за тем, чтобы подпрограмма обработки прерываний не изменяла значения регистров и областей памяти, используемых для вычислений и хранения данных. Обычно это выполняется сохранением рабочих регистров в стеке при начале выполнения обработки прерывания и отказом использования рабочих областей памяти в подпрограмме. Пример программы сохранения состояния процессора при обработке прерываний:

```
LOC     EQU     $               ; Запоминание счетчика адреса.
        ORG     0003h           ; Начальный адрес программы прерывания.
          LJMP  SERV            ; Переход на обработчик прерывания.
;          ...  ...             ; Выполнение подпрограммы.
        ORG     LOC             ; Восстановление счетчика адреса.
SERV:     PUSH  PSW             ; Сохранение регистров в стек.
          PUSH  ACC
          PUSH  B
          PUSH  DPL
          PUSH  DPH
           MOV  PSW, #1000b     ; Выбор другого регистрового банка.
;         ...   ...             ; Тело обработчика прерывания.
           POP  DPH             ; Восстановление регистров из стека
           POP  DPL             ; в обратном порядке.
           POP  B
           POP  ACC
           POP  PSW
          RETI                  ; Возврат из прерывания.
```

2 ОСНОВЫ ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРНЫХ СИСТЕМ

2.1 Общие сведения

Диапазон языков написания исходного текста прикладной программы простирается от машинного кода до почти естественного языка. В машинном коде или на языке ассемблера программировать труднее, чем на алгоритмическом языке высокого уровня, Но, несмотря на данный недостаток, программа, написанная на данном языке, выполняется наиболее быстро и занимает наименьший объем памяти, что является весьма важным аргументом при выборе языка программирования для некоторых видов задач.

Объектные коды, полученные путем трансляции исходных программ, написанных на алгоритмическом языке высокого уровня, занимают в памяти больше места и требуют большего времени на исполнение. Огромное большинство прикладных задач управления объектами вследствие того, что они должны решаться в реальном времени, предъявляют столь высокие требования по быстродействию, что для их решения основным языковым средством написания прикладных программ еще долгие годы будет оставаться язык ассемблера. Это положение о преимущественном использовании языка ассемблера подкрепляется и тем обстоятельством, что однокристальные микроконтроллеры имеют ограниченный объем резидентной памяти программ и, следовательно, критичны к длине прикладных программ. Выбор языковых средств составления исходных программ в каждом конкретном случае зависит от характеристик прикладной задачи, опыта программиста и допустимых затрат на разработку.

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

– Подробно описать задачу.

– Проанализировать задачу.

– Выполнить инженерную интерпретацию задачи, желательно с привлечением того или иного аппарата формализации (граф автомата, сети Петри, матрицы состояний и связности и т.п.).

– Разработать общую схему алгоритма работы контроллера.

– Разработать детализированные схемы отдельных процедур, выделенных на основе модульного принципа составления программ.

– Детально проработать интерфейс контроллера и внести исправления в общую и детализированные схемы алгоритмов.

– Распределить рабочие регистры и память.

– Сформировать текст исходной программы.

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

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

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

Успех разработки прикладной программы заключается в использовании метода декомпозиции, при котором вся задача последовательно разделяется на меньшие функциональные модули. Каждый из модулей можно анализировать, разрабатывать и отлаживать отдельно от других. При выполнении прикладной программы в микроконтроллере управление без всяких двусмысленностей передается от одного функционального модуля к другому. Схема связности этих функциональных модулей, каждый из которых реализует некоторую процедуру, образует общую схему алгоритма прикладной программы. Язык графических образов схемы алгоритма можно использовать на любом уровне детализации описания модулей вплоть до того, что каждому оператору схемы будет соответствовать единственная команда микроконтроллера.

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

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

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

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

– Определить, что должен делать модуль.

– Определить способы получения модулем исходных данных (от датчиков через порты ввода, из таблиц в памяти или через рабочие регистры).

– Определить необходимость какой-либо предварительной обработки введенных исходных данных (маскирование, сдвиг, масштабирование, перекодировка и т. п.).

– Определить метод преобразования входных данных в требуемые выходные. Используя операторы процедур и условные операторы принятия решения, отобразить на языке схемы алгоритма выбранный метод.

– Определить способы выдачи из модуля обработанных данных (передать в память, в вызывавшую программу или в порты вывода).

– Определить необходимость какой-либо вторичной обработки выводимых данных (изменение формата, перекодирование, масштабирование, маскирование и т. п.).

– Вернуться к пункту 1 настоящего перечня и проанализировать полученный результат. Выполнить итеративную корректировку схемы алгоритма с целью сделать ее простой, логичной, стройной и обладающей четким графическим образом.

– Проверить работоспособность алгоритма на бумаге путем подстановки в него действительных данных. Убедиться в его сходимости и результативности.

– Рассмотреть предельные случаи и попытаться определить граничные значения информационных объектов, с которыми оперирует алгоритм, за пределами которых он теряет свойства конечности, сходимости или результативности. Особое внимание при этом следует уделить анализу возможных ситуаций переполнения разрядной сетки,

изменения знака результата операции, деления на переменную, которая может принять нулевое значение.

– Провести мысленный эксперимент по определению работоспособности алгоритма в реальном масштабе времени, когда стохастические события, происходящие в объекте управления, могут оказать влияние на работу алгоритма. При этом самому тщательному анализу следует подвергнуть реакцию алгоритма на возможные прерывания с целью определения критических операторов, которые необходимо защитить от прерываний. Кроме того, в ходе этого мысленного эксперимента следует проанализировать логику алгоритма с целью определения таких последовательностей операторов, при выполнении которых микроконтроллер может «не заметить» кратковременных событий в объекте управления. При обнаружении таких ситуаций в логику следует внести коррективы.

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

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

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

При написании программ на ассемблере для I8051 необходимо учитывать особенности архитектуры микроконтроллера и следовать определенным рекомендациям, которые сформулированы ниже:

– При сбросе микроконтроллер начинает выполнять команды с нулевого адреса, где располагается 8-байтный обработчик прерывания. Поэтому первая команда этого обработчика должна реализовать переход к основной программе и выполнить обход таблицы векторов прерываний.

– Желательно, чтобы программа работала с одним 8-байтовым банком регистров. Это сократит объем программы и ускорит ее выполнение.

– Часто используемые переменные следует размещать в первых 256 байтах памяти, что также сократит объем и ускорит выполнение программы.

– Желательно, чтобы обработчик прерывания имел объем не более 8 байт, чтобы его можно было разместить в соответствующем окне.

– При выполнении программ, которые контролируют время выполнения, прерывания должны быть запрещены. Например, при

выполнении участка кода, программно формирующего последовательность сигналов на каком-либо интерфейсе (к примеру, шина I^2C), где есть жесткие рамки временных характеристик сигналов.

– При использовании непосредственной адресации не забывайте ставить перед константой символ «#». Иначе константа будет интерпретироваться как адрес одного из первых 256 байт памяти.

Выполнение этих рекомендаций при разработке прикладных программ позволит свести к минимуму проблемы, которые могут возникнуть в дальнейшем при отладке программы.

Что касается дополнительной информации по написанию ассемблерных программ для I8051, то она зависит от конкретного используемого ассемблера. Формат написания программ практически одинаков для различных ассемблеров, но для нормальной их работы, возможно, понадобится ввести определенные директивы. Мы будем рассматривать свободно-распространяемый макро-ассемблер версии 2.2 производства Intel Corporation.

2.2 Процедуры и подпрограммы

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

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

Обращение к подпрограмме осуществляется по команде вызова CALL MARK, где MARK – символическое имя процедуры. Имя процедуры используется в качестве метки, отмечающей одну из команд (чаще всего первую) подпрограммы. Для I8051 мнемоническое значение CALL является обобщенным и транслируется в одну из команд ACALL или LCALL в зависимости от адресного расстояния текущей команды до вызываемой подпрограммы.

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

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

При таком интенсивном использовании стековой памяти указатель стека (SP) настраивают как можно дальше от области памяти, где хранятся данные или производятся временные записи. Настройка стекового указателя, при этом, должна выполнятся в самом начале программы – при инициализации. Для более точной величины используемой стековой памяти рассчитывают максимальную вложенность подпрограмм и сохранений регистров в стеке одновременно, а также необходимо учесть вызов прерывания и количество памяти для сохранения состояния процессора в обработчике прерываний, данных и других записей в стек.

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

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

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

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

Передача параметров через регистры осуществляется аналогичным образом.

Третий способ передачи параметров – через регистр признаков. Его удобно использовать при передаче выходных параметров (например, в подпрограммах сравнения чисел). В этом случае подпрограмма должна установить (или сбросить) соответствующие признаки, а основная программа – проанализировать их значение. I8051 обладает большими возможностями для передачи параметров через признаки. В нем имеется 128 флагов пользователя, доступных для модификации и анализа.

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

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

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

2.3 Основные правила записи исходного текста программы

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

Оператор – это базовая конструкция языка программирования, определяющая действия в программе. В языке программирования ASM-51 в одной строке может быть записан только один оператор. Максимальный размер строки – 255 символов. Признаком конца оператора является символ «возврат каретки».

Оператор состоит из трех полей:

<поле метки> <поле операции> <поле комментария>,

Любое из полей, в том числе и все поля, могут отсутствовать. Оператор, в котором все поля отсутствуют, называется пустым оператором. Он используется для увеличения наглядности программы.

Пример оператора, записанного на языке программирования ASM-51:

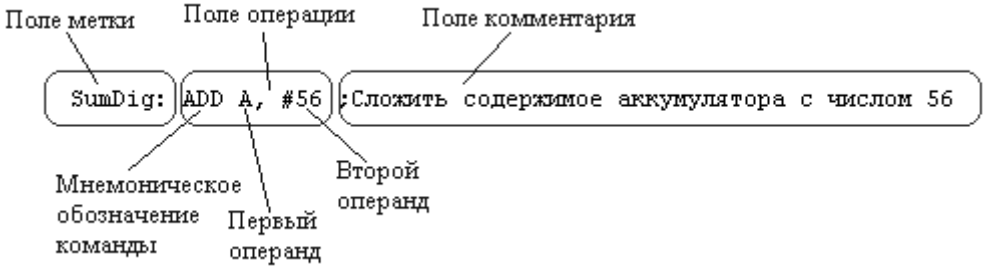


Рисунок 13 – Пример записи оператора на языке ASM-51

Поле метки используется для записи меток. Метки используются для организации условных и безусловных переходов, а также для объявления переменных и констант. Признаком конца поля метки является символ «двоеточие». Однако язык программирования ASM-51, в виде исключения, допускает использовать символы интервала, как признак конца поля метки.

Если в операторе присутствует только метка, то она помечает ближайший следующий оператор, в котором присутствует инструкция процессора или директива ассемблера. Использование оператора, содержащего только метку, может быть вызвано либо слишком большой длиной самой метки, либо необходимостью присвоить одному непустому оператору нескольких меток.

Пример использования оператора, содержащего только метку:

```
PodprogammaPeredachiDannix:  ; Помечается следующий оператор
                MOV  R0,A    ; <- Помечаемый оператор
                MOV  A,@R0
```

Поле операции используется для записи директивы языка или инструкции микроконтроллера, которые состоят из мнемонического обозначения команды микроконтроллера и одного или нескольких операндов. В качестве операндов могут использоваться адреса ячеек памяти, обозначения регистров или метки операторов. Операнды отделяются друг от друга запятыми. Вместе с запятыми для увеличения читаемости программы допускается использование символов интервала.

Поле комментария начинается с символа «точка с запятой». Это поле используется для записи пояснений к программе. Оператор, в котором присутствует только поле комментария, используется для увеличения наглядности программы.

Комментарий начинается с символа (;) и может содержать любые ASCII символы. Примеры комментариев:

```
;-----------------------------------------------------------------*
; ПОДПРОГРАММА ВЫЧИСЛЕНИЯ ФУНКЦИИ                                  |
;-----------------------------------------------------------------*
; X+Y*Z
```

Для набора текста программы используются следующие символы:

– Символы интервала.

– Буквы.

– Знаки.

– Цифры.

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

В качестве **букв** воспринимаются латинские буквы верхнего и нижнего регистра

Наименования знаков и их обозначение приведено в таблице 1:

Таблица 1 – Допустимые знаки

Наименование	Обозначение
Номер	#
Знак денежной единицы	$
Апостроф	'
круглая скобка левая	(
круглая скобка правая	)
Звездочка	*
Плюс	+
Запятая	,
Минус	-
Точка	.
дробная черта	/
Двоеточие	:
Точка с запятой	;
Меньше	<
Равно	=
больше	>
вопросительный знак	?
коммерческое эт	@

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

Из символов формируются идентификаторы и числа.

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

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

Примеры идентификаторов:

ADD5, FFFFH, ALFA_1.

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

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

В языке ASM-51 имеются следующие категории ключевых слов:

– Инструкции.

– Директивы.

– Встроенные имена.

– Операции.

Инструкции по форме записи совпадают с мнемоническими обозначениями команд микроконтроллеров семейства MCS-51 и совместно с операндами, составляют команды микроконтроллера.

Директивы совместно с **вспомогательными словами** определяют действия в программе, которые должны быть выполнены ассемблером в процессе преобразования исходного текста программы в объектный код.

Операции выполняются ассемблером в процессе вычисления выражений на этапе трансляции исходного текста программы для определения конкретного числа, которое используется в команде.

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

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

– Метки.

– Внутренние и внешние переменные адресного типа.

– Внутренние и внешние переменные числового типа.

– Имена сегментов.

– Названия программных модулей.

В языке программирования ASM-51 используются целые беззнаковые числа, представленные в двоичной, восьмеричной, десятичной и шестнадцатеричной формах записи. Для определения основания системы счисления используется суффикс (буква, следующая за числом):

B –двоичное число;

Q или O – восьмеричное число;

[D] – десятичное число (суффикс допускается пропускать);

H – шестнадцатеричное.

Для десятичного числа суффикс может отсутствовать. Количество символов в числе ограничено размером строки, однако значение числа определяется по модулю 2.

Примеры записи чисел:

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

Число всегда начинается с цифры. Это необходимо для того, чтобы отличать шестнадцатеричное число от идентификатора: ADCH – идентификатор; 0ADCH – число.

Часто бывает удобно выполнить некоторые вычисления для того, чтобы получить число. Язык программирования ASM-51 позволяет выполнять беззнаковые операции над числами. В таких выражениях допустимо использовать арифметические операции:

– «+» суммирование.

– «-» вычитание.

– «*» умножение.

– «/» деление.

– «mod» вычисление остатка от целочисленного деления

В языке программирования ASM-51 также определена одноместная операция «-». Для нее требуется один операнд, которому она предшествует. Для изменения порядка выполнения операций можно воспользоваться скобками. Кроме арифметических операций в выражениях допустимо использование логических операций:

– «not» побитовая инверсия операнда.

– «and» логическое «и».

– «or» логическое «или».

– «xor» «исключающее или» (суммирование по модулю два).

– Функций выделения старшего HIGH и младшего LOW байта шестнадцатиразрядного числа.

Пример использования выражений для определения числовой константы:

```
init:
;--- Настроить Timer 0 ---------------------------------------
  mov  TMOD, #00000001b    Выражение для определения константы
               ;    ||||
               ;    ||++--Выбрать режим 16-разрядного таймера
               ;    |+----Использовать внутреннюю синхронизацию
               ;    +-----Запретить управление таймером от INT0

  mov  TL0,  #LOW(-(F_ZQ/12)*10)    ;Настроить таймер
  mov  TH0,  #HIGH(-(F_ZQ/12)*10)   ;на период 10мс

  setb TR0                 ;Включить таймер 0
;-------------------------------------------------------------
```

Рисунок 14 – Использование выражений числовой константы на языке ASM-51

Часто число используется для представления символов. В этом случае для определения числа можно воспользоваться литеральной константой «’». Литеральная константа заключается в апострофы:

```
MOV  SBUF,#‘A’
```

Для записи фраз в памяти программ можно воспользоваться литеральными строками:

```
ERROR:          DB  ‘Ошибка при передаче’
```

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

2.4 Директивы языка ASM-51

2.4.1 Общие сведения

Язык программирования ассемблер всегда включает в себя машинные коды микроконтроллера, но этим не ограничивается набор команд этого языка. Дело в том, что нужно уметь управлять самим процессом трансляции программы. С полным списком директив можно познакомиться непосредственно в описании самого компилятора, а здесь будут подробно рассмотрены некоторые из них.

Первое, что неудобно при использовании только машинных команд – это необходимость помнить, какие данные в какой ячейке памяти находятся. При чтении программы трудно отличить константы от переменных, ведь они отличаются в командах только видом адресации. Преодолеть эту трудность можно при помощи идентификаторов. Можно назначить какой либо ячейке памяти идентификатор, и тем самым работать с этим идентификатором как с переменной. Такой способ применяется при разработке сложных и трудоемких программ.

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

Директивы ассемблера могут быть разделены на ряд категорий:

– Символические определения.
– Резервирование пространства памяти.
– Инициализация данных.
– Управление состоянием ассемблера.
– Выбор сегментов.
– Определение макрокоманд.

2.4.2 Директивы символических определений

Директивы символических определений могут быть использованы для того, чтобы резервировать пространство памяти, поставить в соответствие символическим именам определенные числовые значения, регистры процессора и сегменты. Эти директивы требуют, чтобы имя символа было определено наряду с адресом, числовым значением, регистром или типом сегмента:

BIT – Определяет символическое имя, ссылающееся на адрес бита.

CODE – Определяет символическое имя, ссылающееся на адрес кода (для объектного кода).

DATA – Определяет символическое имя, ссылающееся на адрес резидентной памяти данных.

EQU – Назначает символическому имени числовое значение или имя регистра.

IDATA – Определяет символическое имя, ссылающееся на косвенно-адресуемый адрес резидентной памяти данных (для объектного кода).

SEGMENT – Объявляет имя перемещаемого сегмента, его тип и расположение (для объектного кода).

SET – Назначает символическое имя числовому значению или регистру. Имя может быть впоследствии изменено с помощью директивы SET.

XDATA – Определяет символическое имя, ссылающееся на адрес внешней памяти данных (для объектного кода).

2.4.3 Директивы резервирования и инициализации памяти

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

DB – Заносит в память программ байтовую константу.

DBIT – Резервирует пространство в битовом сегменте (для объектного кода).

DS – Резервирует пространство памяти в текущем сегменте (для объектного кода).

DW – Инициализирует память значением слова.

2.4.4 Директивы компоновки программы

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

EXTRN – Определяет символические имена, которые объявлены в других объектных модулях.

NAME – Определяет имя объектного модуля.

PUBLIC – Определяет символические имена, которые могут использоваться в других объектных модулях.

2.4.5 Директивы управления состоянием ассемблера

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

END – Сообщает о конце транслируемого модуля.

ORG – Изменяет значение ассемблерного счетчика адреса текущего сегмента программы.

USING – Выбирает номер банка регистров общего назначения.

2.4.6 Директивы выбора сегмента

Следующие директивы определяют сегменты данных и кода (для объектного кода):

BSEG – Выбирает абсолютный битовый сегмент.

CSEG – Выбирает сегмент программы в машинном коде.

DSEG – Выбирает абсолютный сегмент резидентной памяти данных.

ISEG – Выбирает абсолютный косвенно адресуемый сегмент резидентной памяти данных.

RSEG – Выбирает предварительно определенный перемещаемый сегмент.

XSEG – Выбирает абсолютный сегмент внешней памяти данных.

2.4.7 Директивы макроопределений

Следующие директивы используются для определения макрокоманд (для макроассемблера):

ENDM – Заканчивает макроопределение.

EXITM – Заставляет макрорасширение немедленно завершиться.

IRP – Определяет список аргументов.

IRPC – Определяет аргумент.

LOCAL – Определяет до 16 локальных символов, используемых внутри макрокоманды.

MACRO – Начало макроопределения, определяет имя макрокоманды и параметров, которые могут быть переданы макрокоманде.

REPT – Определяет количество повторений последующих строк.

2.5 Примеры использования директив

Директива equ позволяет назначать имена переменных и констант. Теперь можно назначить переменной адрес в одном месте и пользоваться идентификатором переменной во всей программе. Правда за использование идентификатора именно в качестве переменной отвечает программист. Тем не менее, если в процессе написания программы потребуется изменить адрес переменной, это можно сделать в одном месте программы, а не просматривать всю программу, разбираясь, является ли в данной конкретной команде число 10 константой, адресом ячейки ли количеством повторов в цикле. Все необходимые изменения сделает сам транслятор. Пример назначения переменных приведен на примере описания интерфейса подключения жидкокристаллического (ЖКИ) к микроконтроллеру. Достоинство такого описания заключается в простом переносе программы на другую, сходную систему, имеющую другое подключение индикатора:

```
DispDat       EQU  P0     ; Шина данных ЖКИ на порте P0.
;
RDS           EQU  P1.2   ; Сигнал чтение команды ЖКИ подключен к
                          ; линии P1.2.
RW            EQU  P1.1   ; Сигнал выбора записи/чтения ЖКИ подключен к
                          ; линии P1.1
E             EQU  P1.0   ; Сигнал строб синхронизации ЖКИ подключен к
                          ; линии P1.0
BUSY          EQU  P0.7   ; Сигнал занятости ЖКИ подключен к линии P0.7
;
SetDD_Adr     EQU  80h    ; Адрес регистра данных/адреса внутри ЖКИ
FuncSet       EQU  20h    ; Команда ЖКИ установки функций
_8bit         EQU  10h    ; Режим 8 бит интерфейс ЖКИ
_2line        EQU  8      ; Режим 2 строчного ЖКИ
```

Как видно на приведенном примере, использование идентификаторов значительно повышает понятность программы, так как в названии переменной отображается функция, за которую отвечает данная переменная.

Один раз назначенный идентификатор уже не может быть изменен в дальнейшем и при повторной попытке назначения точно такого же имени идентификатора будет выдано сообщение об ошибке.

Директива set. Если требуется в различных местах программы назначать одному и тому же идентификатору различные числа, то нужно пользоваться директивой set. Использование этой директивы полностью идентично использованию директивы equ, поэтому иллюстрироваться примером не будет.

Константы, назначаемые директивой equ, могут быть использованы только в одной команде. Достаточно часто требуется работа с таблицей констант, такой как таблица перекодировки, таблицы элементарных функций или синдромы помехоустойчивых кодов. Такие константы используются не на этапе трансляции, а хранятся в памяти программ микроконтроллера. Для занесения констант в память программ микроконтроллера используются **директивы db и dw.**

Рассмотрим применение данных директив на примере. Пусть у нас имеется цифровой семисегментный индикатор с общим катодом, подключенный к микропроцессорной системе согласно рис. 15:

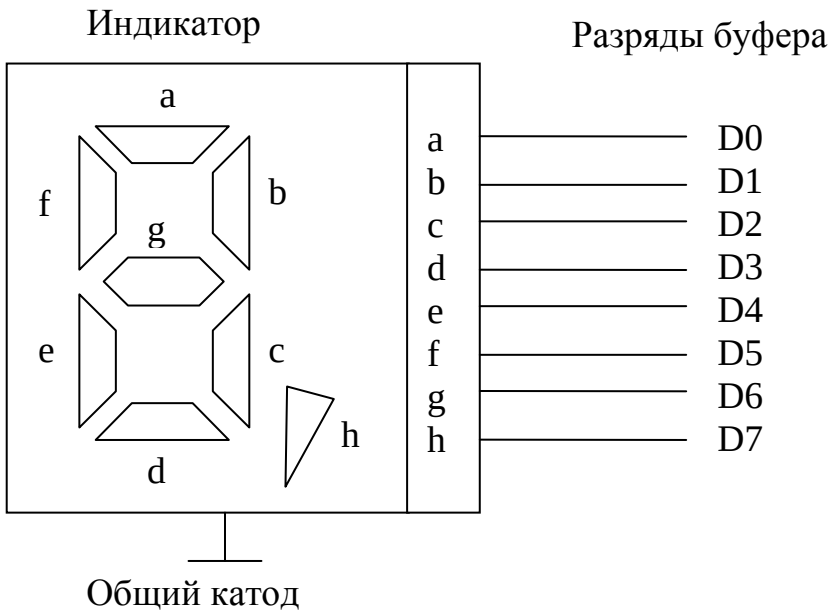


Рисунок 15 – Пример подключения индикатора к МПС

При установке соответствующего бита разряда выходного буфера в логическую «1», зажигается соответствующий сегмент. Например, для отображения числа 1 необходимо установить биты D1 и D2, соответствующие сегментам «b» и «c». Тогда, для вывода чисел на такой индикатор, необходимо выполнить программный дешифратор из числа в код семисегментного индикатора. Можно выполнить табличный перевод, напоминающий поиск значения функции по таблицам Брадиса. Непосредственно данные для свечения на индикаторе задаются директивой db:

```
;Подпрограмма перевода числа в код семисегментного индикатора
;Вход:  A - ДД число формата 0X
```

```
;Выход: A - семисегментный код числа
;---------------------------------------------------------------------
Decode:
            MOV  DPTR, #Table    ; Загрузка адреса таблицы.
           MOVC  A,@A+DPTR       ; Чтение записи со смещением в A.
            RET
;
Table:       ;  hgfedcba
             DB  00111111b       ; Символ ‘0’
             DB  00000110b       ; Символ ‘1’
             DB  01011011b       ; Символ ‘2’
             DB  01001111b       ; Символ ‘3’
             DB  01100110b       ; Символ ‘4’
             DB  01101101b       ; Символ ‘5’
             DB  01111101b       ; Символ ‘6’
             DB  00000111b       ; Символ ‘7’
             DB  01111111b       ; Символ ‘8’
             DB  01101111b       ; Символ ‘9’
```

Эта же директива позволяет легко записывать надписи, которые в дальнейшем потребуется высвечивать на встроенном дисплее или экране дисплея универсального компьютера, подключенного к разрабатываемому устройству через какой-либо интерфейс. Пример использования директивы db для занесения надписей в память программ микроконтроллера приведен ниже:

```
Decode:
     MOV  R7,#EndNadp-NadpSvjazUst   ;В R7 заносим число символов.
     MOV  DPTR,#NadpSvjazUst         ;Подготовить к передаче первый символ
PutNextChar:
     CLR  A
    MOVC  A,@A+DPTR                  ;Читаем очередной символ
     INC  DPTR
    CALL  PutChar                    ;Отправляем символ на передачу
    DJNZ  R7,PutNextChar             ;Последний символ?
     RET                             ;Да, возврат из подпрограммы

NadpSvjazUst: DB ‘Связь установлена’,10,13
EndNadp:
```

Директива dw позволяет заносить в память программ двухбайтные числа. В этой директиве, как и в директиве db числа можно заносить через запятую. Пример листинга фрагмента:

```
0023 0001              294  dw 1,2,0abh,’a’,’QW’
0025 0002
0027 00AB
0029 0061
002B 5157
```

Иногда требуется расположить команду по определенному адресу. Наиболее часто это требуется при использовании прерываний, когда первая команда программы-обработчика прерываний должна быть расположена точно на векторе прерывания. Это можно сделать, используя команду NOP для заполнения промежутков между векторами прерывания, но лучше воспользоваться директивой ORG.

Директива org предназначена для записи в счетчик адреса сегмента значения своего операнда. То есть при помощи этой директивы можно поместить команду (или данные) в памяти микроконтроллера по любому адресу. Пример использования директивы ORG для размещения подпрограмм обработки прерываний на векторах прерываний показан ниже:

```
Reset:
      LJMP Main          ; Переход на начало основной программы.
; Прерывание переполнения таймера 0. ----------------------------------
      ORG 0bh            ; Вектор прерывания таймера 0.
      LJMP IntT0         ; Переход на обработчик прерывания T/C0.
; Прерывание последовательного порта. ----------------------------------
      ORG 23h            ; Вектор прерывания последовательного порта.
      LJMP IntSerPort
;Для частоты кварцевого резонатора 12МГц.
IntT0:
      Mov TL0, #LOW(-(Fosc/12)*10-2)      ;Настройка таймера
      Mov TH0, #HIGH(-(Fosc/12)*10-2)     ;на 10мкс.
      Reti
;Начало основной программы микроконтроллера.
Main: Mov SP,#VershSteka                  ; Настройка указателя стека.
      Call Init          ; Выполнить п/п инициализации микроконтроллера.
;-------------------------------------------------------------------------
```

Необходимо отметить, что при использовании этой директивы возможна ситуация, когда программист приказывает транслятору разместить новый код программы по уже написанному месту, что приводит к неверной трансляции программы.

Директива using. При использовании прерываний критичным является время, занимаемое программой, обработчиком прерываний. Это время можно значительно сократить, выделив для обработки прерываний отдельный банк регистров. Выделить отдельный банк регистров можно при помощи директивы USING. Номер банка используемых регистров указывается в директиве в качестве операнда. Пример использования директивы USING для подпрограммы обслуживания прерываний от таймера 0 приведен ниже:

```
_code segment code
CSEG AT 0bh              ; Вектор прерывания от T/C0.
      Jmp TntT0

rseg _code
      USING 2
IntT0:
      Push PSW           ; Сохраняем регистры в стек.
      Push ACC
      Mov PSW,#00010000b                  ;Включаем банк 2 РОН.
      Mov TL0, #LOW(-(Fosc/12)*10-2)      ;Настройка таймера
      Mov TH0, #HIGH(-(Fosc/12)*10-2)     ;на 10мкс.
      Pop ACC
      Pop PSW
      Reti
```

Директива CALL. В системе команд микроконтроллера MCS-51 используется три команды безусловного перехода. Выбор конкретной команды зависит от расположения ее в памяти программ, однако программист обычно этого не знает. В результате во избежание ошибок приходится использовать самую длинную команду LJMP. Это приводит к более длинным программам и к дополнительной нагрузке на редактор связей. Транслятор сам может подобрать наилучший вариант команды безусловного перехода. Для этого вместо команды микроконтроллера следует использовать директиву call.

2.6 Реализация подпрограмм

При написании программ часто при реализации алгоритма работы устройства приходится повторять одни и те же операторы (например операторы, работающие с параллельным или последовательным портом). Было бы неплохо использовать один и тот же участок кода, вместо того, чтобы повторять одни и те же операторы несколько раз.

Участок программы, к которому можно обращаться из различных мест программы для выполнения некоторых действий называется **подпрограммой**.

Проблема, с которой приходится сталкиваться при многократном использовании участков кодов – это в какое место памяти программ возвращаться после завершения подпрограммы. Обращение к подпрограмме производится из нескольких мест основной программы. Описанную ситуацию иллюстрирует рис. 16. На этом рисунке изображено адресное пространство микроконтроллера. Младшие адреса адресного пространства на этом рисунке находятся в нижней части.

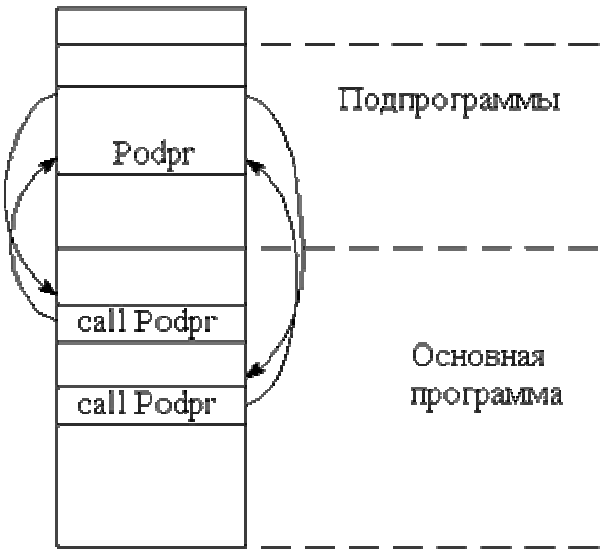


Рисунок 16 – Вызов подпрограммы и возврат к выполнению основной программы

Для обращения к подпрограмме и возврата из нее в систему команд микропроцессоров вводят специальные команды. В микроконтроллерах семейства MCS-51 это команды LCALL, ACALL для вызова подпрограммы и команда RET для возврата из подпрограммы. Эти команды не только осуществляют передачу управления на указанный адрес, но и запоминают адрес команды, следующей за командой вызова подпрограммы в стековой памяти. Команда возврата из подпрограммы RET передает управление команде, адрес которой был запомнен командой вызова подпрограммы. Пример использования подпрограммы на языке программирования ASM-51 приведен ниже:

```
      ...
      Mov G_Per,#56          ; Передать 56 через последовательный порт.
      Call PeredatByte
      ...
      Mov G_Per,#49          ; Передать 49 через последовательный порт.
      Call PeredatByte
      ...
;-------------------------------------------------------------------*
; Подпрограмма передачи байта по последовательному порту.           |
;-------------------------------------------------------------------*
PeredatByte:
      Jb TI,$          ; Ожидание конца передачи предыдущего байта.
      Mov SBUF,G_Per   ; Передача байта
      Ret
```

Ни в коем случае нельзя попадать в подпрограмму любым способом кроме команды вызова подпрограммы CALL. В противном случае команда возврата из подпрограммы передаст управление случайному адресу. По этому адресу могут быть расположены данные, которые в этом случае будут интерпретированы как программа, или обратиться к внешней памяти, откуда будут считываться случайные числа.

Очень часто требуется из одной подпрограммы обращаться к другой подпрограмме. Такое обращение к подпрограмме называется вложенным. Количество вложенных подпрограмм называется уровнем вложенности подпрограмм. Максимально допустимый уровень вложенности подпрограмм определяется количеством ячеек стековой памяти. Логически эти ячейки памяти организованы так, чтобы считывание последнего записанного адреса производилось первым, а первого записанного адреса производилось последним (буфер LIFO). Такая логическая организация формируется специальным счетчиком. Этот счетчик, как было указано выше, называется указателем стека **SP**. Ячейка памяти, в которую в данный момент может быть записан адрес возврата из подпрограммы, называется **вершиной стека**. Количество ячеек памяти, предназначенных для организации стека, называется глубиной стека. Последняя ячейка памяти, в которую можно производить запись называется **дном стека**. Логическая организация стека приведена на рис. 17.

В микроконтроллерах семейства MCS-51 при занесении информации в стек содержимое указателя стека увеличивается (стек растет вверх), поэтому стек размещается в самой верхней части памяти данных. Для того,

чтобы установить глубину стека 28 байт, необходимо вычесть из адреса максимальной ячейки внутренней памяти микроконтроллера глубину стека и записать полученное значение в указатель стека SP:

```
DnoSteka EQU 127        ; Объем внутреннего ОЗУ I8051 - 128 байт.
Mov SP,#DnoSteka-28     ; Установим глубину стека 28 байт.
```

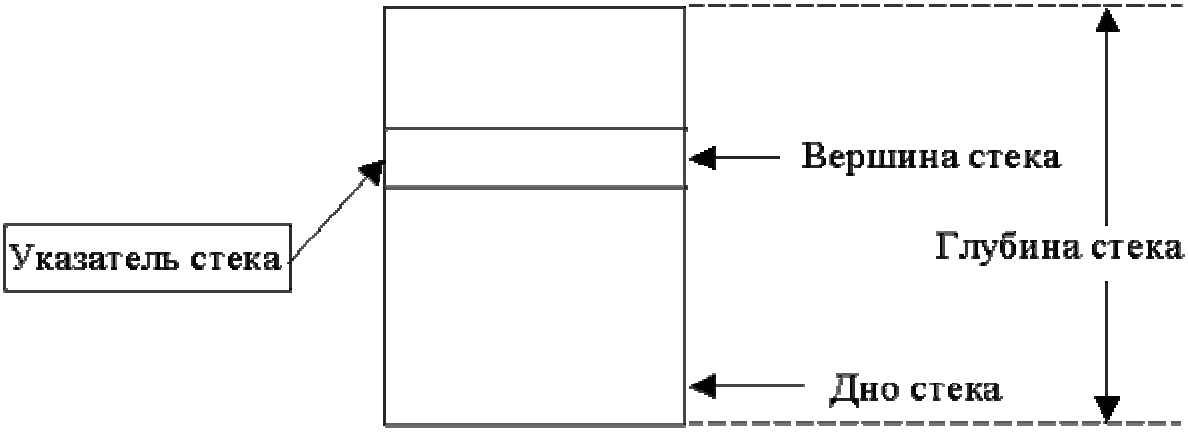


Рисунок 17 – Организация стека в памяти данных микропроцессора

Кроме содержимого программного счетчика часто требуется запоминать содержимое внутренних регистров и флагов процессора, локальных переменных подпрограммы. Стек оказался удобным средством и для этой задачи. Сохранение локальных переменных в стеке позволило осуществлять вызов подпрограммы самой из себя (реализовывать рекурсивные алгоритмы). Это привело к введению в систему команд специальных команд работы со стеком. В микроконтроллерах семейства MCS-51 это команды PUSH и POP. Использование этих команд показывается на следующем примере:

```
Podprogramma:
      PUSH PSW          ; Сохраняем используемые в подпрограмме
      PUSH ACC          ; регистры в стеке.
      PUSH R0

      ...               ; Код самой подпрограммы.

      POP R0            ; Извлекаем сохраненные регистры из стека
      POP ACC           ; в обратном порядке.
      POP PSW
      RET               ; Выход из подпрограммы.
```

В приведенном выше примере передачи байта через последовательный порт, сам байт передается в подпрограмму через глобальную переменную G_Per. Однако программа будет эффективнее при использовании подпрограммы с параметрами. Мы знаем, что параметр подпрограммы – это локальная переменная. В этом случае могут значительно снизиться требования к памяти данных. Для размещения локальных переменных лучше всего использовать внутренние регистры

процессора. На языке ASM51 для передачи параметра размерностью один байт обычно используется аккумулятор.

Если в подпрограмму нужно передать двухбайтовое значение, то в качестве параметра подпрограммы используется пара регистров (обычно регистры R6-старший байт и R7-младший байт). Пример программы, передающей в подпрограмму двухбайтовое число, написанной на языке программирования ASM-51 приведен ниже:

```
      ...
;Пример передачи в подпрограмму двухбайтового числа.
      Mov R7, #56       ; Передача младшего байта.
      Mov R6, #0        ; Передача старшего байта.
      Call Podprog      ; Вызвать подпрограмму.
      ...
```

Если в подпрограмму нужно передать четырехбайтовое значение (это требуется для переменной, соответствующей типу long или float), то используются регистры R4...R7 (регистр R4 – старший байт):

```
      ...
;Пример передачи в подпрограмму четырехбайтового числа.
      Mov R7, #56       ; Передача младшего байта.
      Mov R6, #0
      Mov R5, #0
      Mov R4, #0        ; Передача старшего байта.
      Call Podprog      ; Вызвать подпрограмму.
      ...
```

Регистры R0 и R1 обычно используются в качестве указателей обрабатываемых переменных, таких как строки или массивы. Если требуется, чтобы подпрограмма обработала значительный объем данных, то эти данные можно передать через параметр – указатель. В качестве указателя при обращении к внешней памяти данных или к памяти программ обычно используется регистр-указатель данных DPTR. Пример передачи в качестве параметра строки, написанный на языке программирования ASM-51 приведен ниже:

```
      ...
;Пример передачи в подпрограмму указателя данных.
      Mov DPTR, #Stroka ; Передача указателя на строку.
      Call Podprog      ; Вызвать подпрограмму.
      ...
Stroka: DB ‘Наша строка символов.’
```

При обращении к массивам или структурам, расположенным во внутренней памяти данных в качестве указателя адреса используется регистр R0 или R1:

```
      ...
;Пример передачи в подпрограмму указателя данных внутренней памяти.
      Mov R0, #Array    ; Передача указателя на данные.
      Call Obrabotka    ; Вызвать подпрограмму.
      ...
```

Часто требуется передавать результат вычислений из подпрограммы в основную программу. Для этого можно воспользоваться подпрограммой-функцией. Подпрограмма-функция возвращает вычисленное значение:

```
      Mov A,X           ; Передать в подпрограмму значение X.
```

```
Call Sin          ; Вызов функции Y=sin(X)
Mov Y,A           ; Сохранение функции в переменной Y.
```

При этом переменные X и Y должны быть заранее определены директивой EQU.

2.7 Реализация многомодульных программ

Разбиение исходного текста программы на несколько файлов делает этот текст более понятным для программиста или нескольких программистов, участвующих в создании программного продукта. Однако остается нерешенными еще несколько задач:

– Программа-транслятор работает со всем исходным текстом целиком, ведь она соединяет все файлы перед трансляцией вместе. Поэтому время трансляции исходного текста программы остается значительным (и даже возрастает). В то же самое время программа никогда не переписывается целиком. Обычно изменяется только небольшой участок программы.

– При назначении переменных их количество ограничено программой-транслятором и может быть исчерпано при написании программы.

– Различные программисты, участвующие в создании программного продукта могут назначать одинаковые имена для своих переменных и при попытке соединения файлов в единую программу обычно возникают проблемы.

Все эти проблемы могут быть решены при раздельной трансляции программы. То есть было бы неплохо уметь транслировать каждый файл с исходным текстом программы отдельно и соединять затем готовые оттранслированные участки программы.

Компиляторы, которые позволяют транслировать отдельные участки программы, называются **компиляторами с раздельной трансляцией.**

Исходный текст программы, который может быть отдельно оттранслирован, называется **программным модулем**.

Оттранслированный программный модуль сохраняется в виде отдельного файла в объектном формате, где кроме машинных команд сохраняется информация об именах переменных, адресах команд, требующих модификации при объединении модулей в единую программу и отладочная информация.

Раздельная трансляция программы возможна при использовании двух программ: **транслятора** исходного текста программы и **редактора связей**.

На первый взгляд раздельная трансляция не должна вызывать каких-либо проблем. Однако это не так. При компиляции исходного текста программы транслятор составляет таблицу ссылок на константы, переменные и команды. Если при втором просмотре исходного текста

программы, во время которого формируется объектный модуль, транслятор не обнаружит имени переменной или метки в своей таблице, то будет сформировано сообщение об ошибке и объектный модуль будет стерт с диска компьютера.

Для того, чтобы транслятор вместо формирования сообщения об ошибке записал в объектный модуль информацию, необходимую для редактора связей, нужно использовать специальные директивы ссылок на внешние переменные или метки. Обычно эти директивы называются PUBLIC (общие) и EXTRN (внешние). Для ссылки на переменную или метку используется директива EXTRN. В этой директиве перечисляются через запятую метки и переменные, точное значение которых редактор связей должен получить из другого модуля и модифицировать все команды, в которых эти метки или переменные используются. Пример использования директивы EXTRN на языке программирования ASM-51:

```
EXTRN DATA (BufInd, ERR)
EXTRN CODE (Podprog)
```

Для того, чтобы редактор связей мог осуществить связывание модулей в единую программу, переменные и метки, объявленные по крайней мере в одном из модулей как EXTRN, в другом модуле должны быть объявлены как доступные для всех модулей при помощи директивы PUBLIC. Пример использования директивы PUBLIC:

```
PUBLIC BufInd, Parametr
PUBLIC Podprogr, ?Podprog?Byte
```

Использование нескольких модулей при написании программы увеличивает скорость трансляции и, в конечном итоге, скорость написания программы. Однако объявления переменных и имен подпрограмм внешних модулей загромождают исходный текст модуля. Кроме того, при использовании чужих модулей трудно объявить переменные и подпрограммы без ошибок. Поэтому обычно объявления переменных, констант и предварительные объявления подпрограмм хранят во включаемых файлах, которые называются файлами-заголовками. Правилом хорошего тона считается при разработке программного модуля сразу же написать файл-заголовок для этого модуля, который может быть использован программистами, работающими с Вашим программным модулем.

Для объединения нескольких модулей в исполняемую программу имена всех модулей передаются в редактор связей rl51.exe в качестве параметров при запуске этой программы. Пример вызова редактора связей из командной строки DOS для объединения трех модулей:

rl51.exe progr.obj, modul1.obj, modul2.obj

В результате работы редактора связей в этом примере будет создан исполняемый модуль с именем progr. Формат записи информации в этом файле остается прежним – объектный. Это позволяет объединять модули по частям, то есть при желании можно из нескольких мелких модулей получить один более крупный.

Для получения из объектного файла HEX-файл необходимо объектный файл обработать программой oh.exe.

2.8 Использование сегментов в языке программирования ассемблер

Необходимо отметить, что даже когда мы не задумываемся о сегментах, в программе присутствует два сегмента: сегмент кода программы и сегмент данных. Если внимательно присмотреться к программе, то можно обнаружить, что кроме кодов команд в памяти программ хранятся константы, то есть в памяти программ микроконтроллера располагаются, по крайней мере, два сегмента: программа и данные. Чередование программы и констант в довольно сложной программе, может привести к нежелательным последствиям. Вследствие каких-либо причин данные могут быть случайно выполнены в качестве программы или наоборот программа может быть воспринята и обработана как данные.

Перечисленные выше причины приводят к тому, что желательно явным образом выделить, по крайней мере, четыре сегмента:

– Программы.
– Стека.
– Переменных.
– Констант.

Пример размещения сегментов в адресном пространстве памяти программ и внутренней памяти данных приведен на рис. 18:

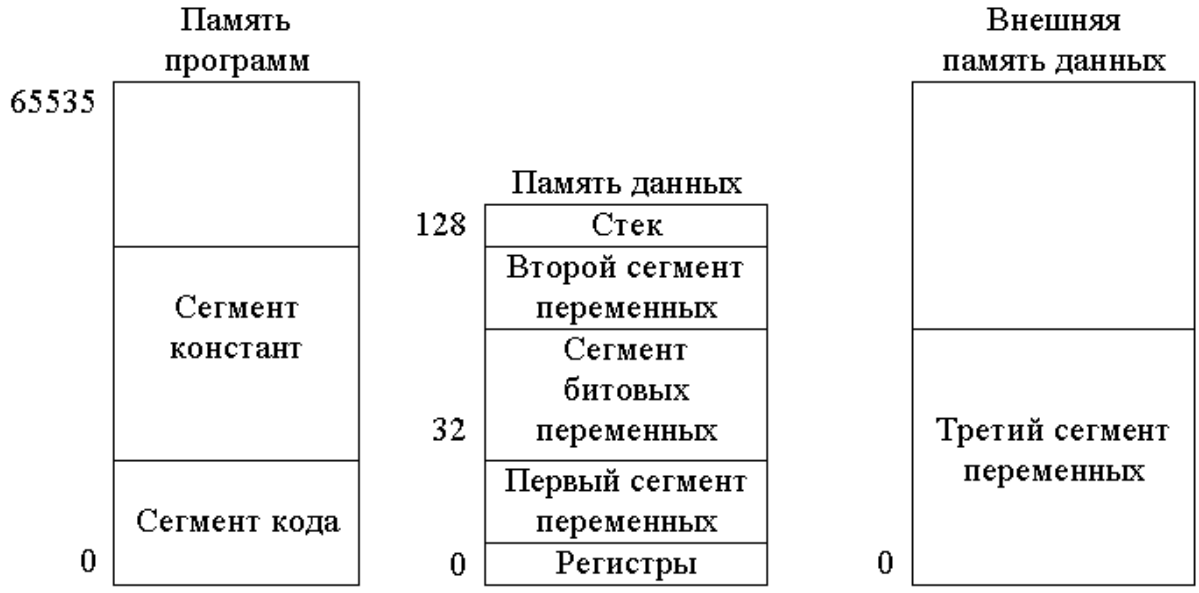


Рисунок 18 – Разбиение памяти программ и данных на сегменты

На этом рисунке видно, что при использовании нескольких сегментов переменных во внутренней памяти данных, редактор связей

может разместить меньший из них на месте неиспользованных банков регистров. Под сегмент стека обычно отводится вся область внутренней памяти, не занятая переменными. Это позволяет создавать программы с максимальным уровнем вложенности подпрограмм.

Наиболее простой способ определения сегментов это использование абсолютных сегментов памяти. При этом способе распределение памяти ведется вручную точно так же, как это делалось при использовании директивы EQU. В этом случае начальный адрес сегмента жестко задается программистом, и он же следит за тем, чтобы сегменты не перекрывались друг с другом в памяти микроконтроллера. Использование абсолютных сегментов позволяет более гибко работать с памятью данных, так как теперь байтовые переменные в памяти данных могут быть назначены при помощи директивы резервирования памяти DS, а битовые переменные при помощи директивы резервирования битов DBIT.

Для определения абсолютных сегментов памяти используются следующие директивы.

Директива BSEG позволяет определить абсолютный сегмент во внутренней памяти данных с битовой адресацией по определенному адресу. Эта директива не назначает имени сегменту, то есть объединение сегментов из различных программных модулей невозможно. Для определения конкретного начального адреса сегмента применяется атрибут AT. Если атрибут AT не используется, то начальный адрес сегмента предполагается равным нулю. Использование битовых переменных позволяет значительно экономить внутреннюю память программ микроконтроллера. Пример использования директивы BSEG для объявления битовых переменных:

```
      BSEG AT 8          ; Сегмент начинается с 8 бита.
RejInd:     DBIT 1       ; Флаг режима индикации.
RejPriem:   DBIT 1       ; Флаг режима приема.
Flag:       DBIT 1       ; Флаг общего назначения.
```

Директива CSEG позволяет определить абсолютный сегмент в памяти программ по определенному адресу. Эта директива не назначает имени сегменту, то есть объединение сегментов из различных программных модулей невозможно. Для определения конкретного начального адреса сегмента применяется атрибут AT. Если атрибут AT не используется, то начальный адрес сегмента предполагается равным нулю. Пример использования директивы CSEG для размещения подпрограммы обслуживания прерывания от таймера 0:

```
CSEG AT 0bh              ; Вектор прерывания от T/C0.
IntT0:
      Mov TL0, #LOW(-(Fosc/12)*10-2)     ;Настройка таймера
      Mov TH0, #HIGH(-(Fosc/12)*10-2)    ;на 10мкс.
      Reti
```

Директива DSEG позволяет определить абсолютный сегмент во внутренней памяти данных по определенному адресу. Предполагается, что к этому сегменту будут обращаться команды с прямой адресацией. Эта директива не назначает имени сегменту, то есть объединение сегментов из

различных программных модулей невозможно. Для определения конкретного начального адреса сегмента применяется атрибут AT. Если атрибут AT не используется, то начальный адрес сегмента предполагается равным нулю. Пример использования директивы DSEG для объявления байтовых переменных:

```
      DSEG AT 20h       ; Разместить сегмент с адреса, где возможна
                        ; битовая адресация (Для возможности одновремен-
                        ; ной битовой и байтовой адресации).
RejInd:     DS 1        ; Переменная, отображающая состояние программ
                        ; обслуживания аппаратуры.
Rejim:      DS 1        ; Переменная режима работы.
Massiv:     DS 10       ; Десятибайтовый массив.
```

В приведенном примере предполагается, что он связан с примером, приведенном выше. Т. е. команды, изменяющие битовые переменные RejInd, RejPriem или Flag одновременно будут изменять содержимое переменной Rejim, и наоборот команды, работающие с переменной Rejim, одновременно изменяют содержимое флагов RejInd, RejPriem или Flag. Такое объявление переменных позволяет написать наиболее эффективную программу управления контроллером и подключенными к нему устройствами.

Директива ISEG позволяет определить абсолютный сегмент во внутренней памяти данных по определенному адресу. Эта директива не назначает имени сегменту, то есть объединение сегментов из различных программных модулей невозможно. Для определения конкретного начального адреса сегмента применяется атрибут AT. Если атрибут AT не используется, то начальный адрес сегмента предполагается равным нулю. Пример использования директивы ISEG для объявления байтовых переменных:

```
      ISEG AT 40h       ; Расположить сегмент в адресах от 40h.
Buffer:     DS 10       ; Переменная Buffer объемом 10 байт.
Stack:      DS 245      ; Глубина стека 245 байт.
```

Если абсолютные адреса переменных или участков программ не интересны, то можно воспользоваться перемещаемыми сегментами. Имя перемещаемого сегмента задается директивой segment.

Директива segment позволяет определить имя сегмента и область памяти, где будет размещаться данный сегмент памяти. Для каждой области памяти определено ключевое слово:

Data – размещает сегмент во внутренней памяти данных с прямой адресацией;

Idata – размещает сегмент во внутренней памяти данных с косвенной адресацией;

Bit – размещает сегмент во внутренней памяти данных с битовой адресацией;

Xdata – размещает сегмент во внешней памяти данных;

Code – размещает сегмент в памяти программ;

Директива rseg. После определения имени сегмента можно использовать этот сегмент при помощи директивы rseg. Использование

сегмента зависит от области памяти, для которой он предназначен. Если это память данных, то в сегменте объявляются байтовые или битовые переменные. Если это память программ, то в сегменте размещаются константы или участки кода программы. Пример использования директив segment и rseg для объявления битовых переменных:

```
_data segment idata
      PUBLIC VershSteka, BufferKlav

; Определение переменных
      rseg _data
      BufferKlav DS 8   ; Объем буфера клавиатуры 8 байт.
      VershSteka:       ; Стек начинается здесь.
```

В этом примере объявлена строка buferKlav, состоящая из восьми байтовых переменных. Кроме того, в данном примере объявлена переменная VershSteka, соответствующая последней ячейке памяти, используемой для хранения переменных. Переменная VershSteka может быть использована для начальной инициализации указателя стека для того, чтобы отвести под стек максимально доступное количество ячеек внутренней памяти. Это необходимо для того, чтобы избежать переполнения стека при вложенном вызове подпрограмм.

Объявление и использование сегментов данных в области внутренней или внешней памяти данных не отличается от приведенного примера за исключением ключевого слова, определяющего область памяти данных.

Еще один пример использования директив segment и rseg:

```
_bits segment bit
      PUBLIC KnIzm, strVv

; Определение битовых переменных
      rseg _bits
      KnIzm: DBIT 1     ; Флаг нажатия кнопки.
      strVv: DBIT 1     ; Флаг введенной строки.
```

В этом примере директива segment используется для объявления сегмента битовых переменных.

Наибольший эффект от применения сегментов можно получить при написании основного текста программы с использованием модулей. Обычно каждый программный модуль оформляется в виде отдельного перемещаемого сегмента. Это позволяет редактору связей скомпоновать программу оптимальным образом. При использовании абсолютных сегментов памяти программ приходится это делать вручную, а так как в процессе написания программы размер программных модулей постоянно меняется, то приходится вводить защитные области неиспользуемой памяти между программными модулями.

Пример использования перемещаемых сегментов в исходном тексте программы:

```
_code segment code

CSEG AT 0               ; Начало программы.
```

```
Reset:
      Jmp Main

; Начало основной программы микроконтроллера----------------------------
rseg _code
Main:
      Movx @DPTR,A
      Mov SP,#VershSteka      ; Настроить указатель стека на вершину
      Call Init               ; Выполнить п/п инициализации процессора.
;-------------------------------------------------------------------------
```

В этом примере приведен начальный участок основной программы микроконтроллера, на который производится переход с нулевой ячейки памяти программ. Использование такой структуры программы позволяет в любой момент времени при необходимости использовать любой из векторов прерывания, доступный в конкретном микроконтроллере, для которого пишется эта программа. Достаточно поместить определение этого вектора с использованием директивы cseg.

В приведенном примере использовано имя перемещаемого сегмента _code. Оно было объявлено в самой первой строке исходного текста программы. Конкретное имя перемещаемого сегмента может быть любым, но как уже говорилось ранее оно должно отображать ту задачу, которую решает данный конкретный модуль.

3 СОЗДАНИЕ ПРОГРАММ ДЛЯ МИКРОКОНТРОЛЛЕРОВ

3.1 Структурное программирование

Создание программ для микроконтроллеров резко отличается от написания программ для универсального компьютера. При выполнении программы на универсальном компьютере запуск программ, взаимодействие с внутренними, внешними устройствами или человеком берет на себя операционная система. Программа, написанная для микроконтроллера, должна решать все эти задачи. Программа, написанная для компьютера, когда-нибудь запускается и завершается. Программа, управляющая микроконтроллером, запускается при включении устройства и не завершает свою работу пока не будет выключено питание.

Программа для микроконтроллера должна решать все перечисленные выше проблемы. Простейшим видом программы, которая может решать поставленные задачи является монитор.

Алгоритм программы-монитора приведен на рис. 19. После включения питания эта программа должна настроить микросхему под выполняемую программой задачу, то есть настроить определенные ножки микросхемы на ввод или вывод информации, включить и настроить внутренние таймеры микроконтроллера и так далее. Этот блок алгоритма программы-монитора называется инициализацией процессора.

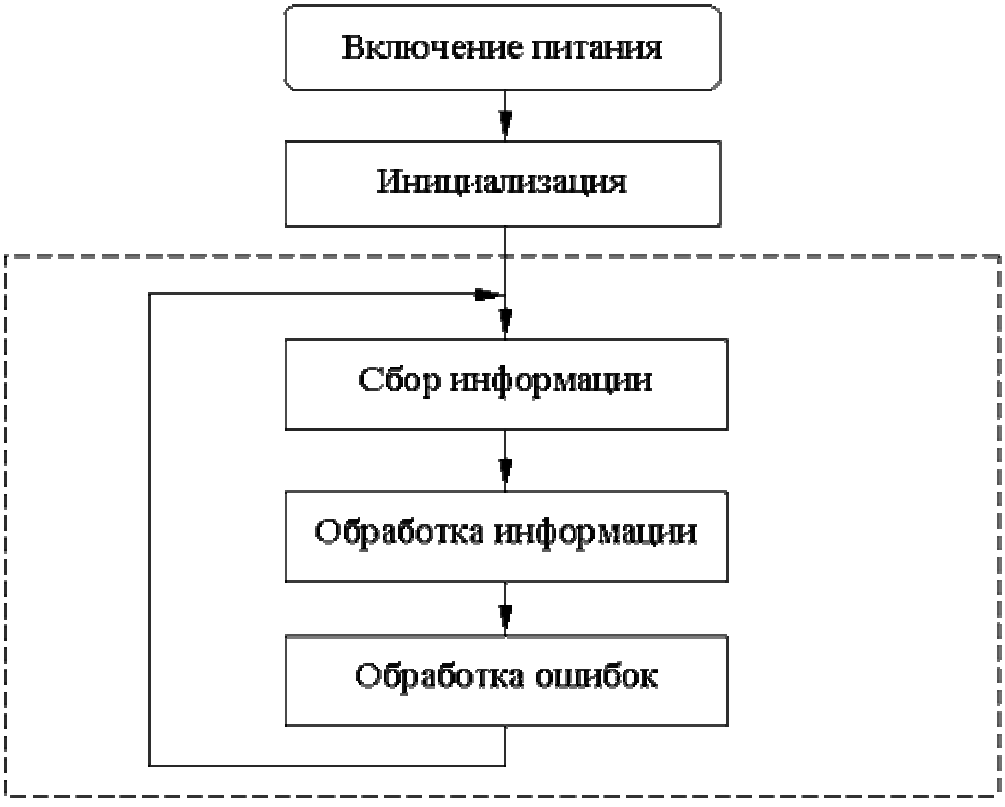


Рисунок 19 – Алгоритм программы-монитора

Основная часть программы начинает выполняться после настройки микроконтроллера на выполняемую работу. При этом необходимо понимать, что если в аппаратуре ввод, обработка и вывод информации производится различными блоками, то при выполнении программы эти же действия производятся последовательно одним и тем же устройством – микропроцессором. В этом же цикле предусмотрен блок обработки ошибок. Его предназначение сообщать оператору о непредвиденной ситуации, такой как неправильный ввод с клавиатуры или неправильные данные с подключенного к микроконтроллеру устройства.

При использовании нескольких подпрограмм встает проблема обмена информацией между этими подпрограммами. Как уже рассматривалось ранее, информация в подпрограмму может быть передана через параметры подпрограммы или через глобальные переменные. При создании программы-монитора может потребоваться передавать одну и ту же информацию нескольким подпрограммам, поэтому в мониторах информация обычно передается через глобальные переменные.

В настоящее время существует два способа написания программ: снизу вверх и сверху вниз. При написании программы снизу вверх приступить к отладке программы невозможно, не написав полностью всю программу. При написании программы сверху вниз на любом этапе написания программы она может быть оттранслирована и выполнена, при этом можно отследить все алгоритмические действия программы, написанные к этому времени. Процесс написания программы не отличается от процесса создания алгоритма. Более того, эти этапы создания программы можно объединить. Выполняемое алгоритмическое действие отображается в названии подпрограммы. Например:

```
Call ReadPort          ; Прочитать порт
Call IndOFF            ; Прочитать порт
```

Основная идея структурного программирования заключаются в том, что существует только четыре структурных оператора. Используя эти операторы, можно построить сколь угодно сложную программу.

Первый структурный оператор называется **линейная цепочка операторов**. Любая задача может быть разбита на несколько подзадач. Выполнение подзадач может быть поручено подпрограмме, в названии которой можно (и нужно) отразить подзадачу, которую должна решать эта подпрограмма. На момент написания алгоритма (и программы) верхнего уровня нас не интересует, как будет решаться эта задача, поэтому вместо настоящей подпрограммы поставим подпрограмму-заглушку (Подпрограмма-заглушка это подпрограмма, которая ничего не выполняет, а только возвращает управление главной программе; действие, которое в дальнейшем должна выполнять эта программа, отображается в названии подпрограммы-заглушки).

Алгоритмически такая цепочка операторов изображена на рис. 20.

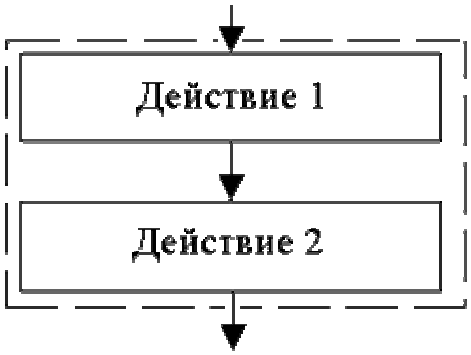


Рисунок 20 – Алгоритмическое изображение линейной цепочки операторов

Второй структурный оператор называется **условный оператор (оператор ветвления)**. Достаточно часто одна или другая задачи должны исполняться в зависимости от определенного условия, которое зависит от результатов выполнения предыдущей программы или от внешних устройств. Каждая из таких задач называется плечом условного оператора и изображена на рис. 21:

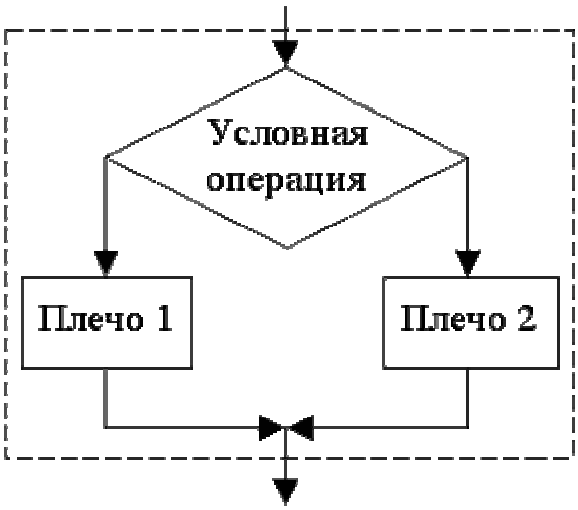


Рисунок 21 – Алгоритмическое изображение условного оператора

Условный оператор может использоваться в неполном виде, когда отсутствует одно из плеч оператора.

Третий структурный оператор – это **оператор цикла с проверкой условия после тела цикла**. Такой оператор легко реализуется на языке программирования ассемблер при помощи команды условного или безусловного перехода. Отличие от условного оператора заключается в том, что передача управления осуществляется не вперед, а назад. На языках программирования высокого уровня такой оператор входит в состав языка (оператор do..while в языке программирования C++, или оператор repeat..until в языке программирования PASCAL). В языке ассемблер MCS-51 – это команда DJNZ, перед которой пишется тело цикла.

Алгоритмически такой оператор изображен на рис. 22.

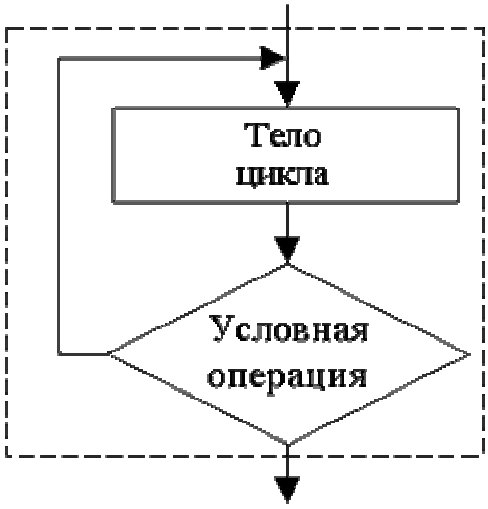


Рисунок 22 – Алгоритмическое изображение оператора цикла с проверкой условия после тела цикла

Четвертый структурный оператор – это **оператор цикла с проверкой условия до тела цикла**. В отличие от предыдущего оператора тело цикла в этом операторе может ни разу не выполниться, если условие цикла сразу же выполнено. Этот оператор, как и условный оператор, невозможно реализовать на одной машинной команде:

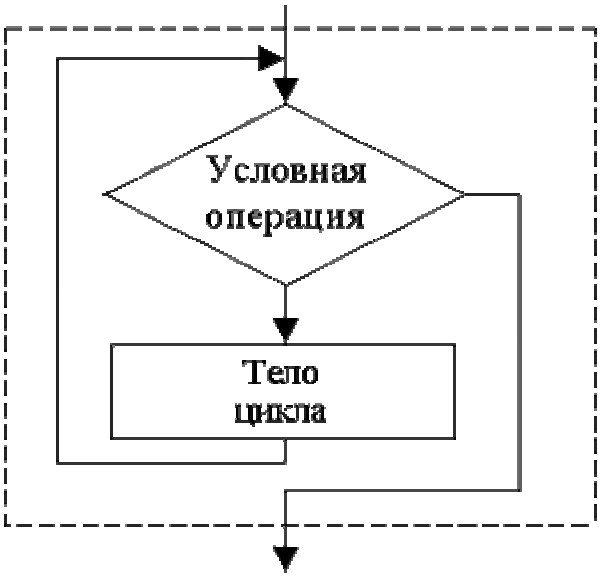


Рисунок 23 – Алгоритмическое изображение оператора цикла с проверкой условия до тела цикла

Условный оператор выполняется путем сравнения величин. Если это битовая величина, то в таком случае все просто – если бит установлен, выполняется одно плечо, если нет – другое. При сравнении байтовых и более размером величин выделяют следующие условия:

E (equal) – Равно;

NE (Not Equal) – Не равно;

GE (Great or Equal) – Больше или равно;

LE (Less or Equal) – Меньше или равно;

GT (Great Than) – Больше;

LT (Less Than) – Меньше.

У некоторых микропроцессоров операции данных условных переходов присутствуют. Для микропроцессоров, у которых нет таких операций (в том числе и I8051), данные условия строятся на анализе результатов регистра PSW после выполнения сравнения чисел.

Рассмотрим пример сравнения двух байтовых величин на микроконтроллере I8051. Т. к. данный контроллер имеет команду сравнения и перехода, если не равно (CJNE), то данный вариант мы рассматривать не будем.

Пусть имеется 2 байта, один в аккумуляторе, другой – в регистре R1. Нужно их сравнить и выполнить плечо условия №1, если они равны. Иначе – плечо №2:

```
      Clr C             ; Отчистим признак переноса перед сравнением.
      Subb A,R1         ; Сравнение вычитанием
      Jnz NeRavno       ; Переход на плечо №2.
      ...               ; Здесь выполняется плечо №1 равенства.
      Jmp Dalee
NeRavno:
      ...               ; Здесь выполняется плечо №2 не равно.
Dalee:                  ; Продолжение программы.
```

При выполнении мягких условий (GE и LE) задача усложняется из-за контроля двух признаков. Пусть нам необходимо проверить условие (A)≤(R1). Если оно соблюдается, то выполняем плечо №1, иначе – №2:

```
      Clr C             ; Отчистим признак переноса перед сравнением.
      Subb A,R1         ; Сравнение вычитанием.
      Jb ACC.7,Menwe    ; Переход на плечо №1.по условию меньше (A<0).
      Jz Menwe          ; Переход на плечо №1 по условию равенства (A=0).
      ...               ; Здесь выполняется плечо №2.
      Jmp Dalee
Menwe:
      ...               ; Здесь выполняется плечо №1.
Dalee:                  ; Продолжение программы.
```

По такому принципу можно построить конструкции и с помощью команды сложения. Пример контроля признаков результата сравнения в зависимости от разных условий при выполнении вычитания приведен в таблице 2. В процессе построения программы следует помнить, что условные переходы в языке ассемблер для микроконтроллера I8051 могут выполнить только относительный переход по адресному пространству в пределах -128…+127 адресов относительно самой команды. Поэтому желательно располагать плечи условий, которые выполняются по переходу, как можно ближе к участку программы, где выполняется сравнение. Если такой возможности нет и в процессе компиляции программы, компилятор выдал ошибку о невозможности выполнить переход, то приведенные выше

конструкции условных операторов нужно дополнить «длинными прыжками» LJMP к плечу оператора.

Таблица 2 – Ветвления и их команды после операции вычитания

Условие	Название условия	Выполнение ветвлений
A=R1	E	JZ
A≠R1	NE	JNZ
A<R1	LT	JC или JB ACC.7
A>R1	GT	JNC или JNB ACC.7
A≤R1	LE	(JC+JZ)
A≥R1	GE	(JNC+JZ)

3.2 Создание программы микропроцессорного устройства

Рассмотрим пример написания программы для микроконтроллера. Прежде всего, не нужно забывать, что программа не может существовать отдельно независимо от схемы устройства. Если при написании программы для универсального компьютера, такого как IBM PC можно не задумываться о схеме, так как она стандартная, то перед написанием программы для микроконтроллера необходимо разработать схему устройства, или изучить, если последняя имеется в наличии, в состав которого будет входить микроконтроллер.

Для программирования будем применять микропроцессорную систему, выполненную на основе учебно-отладочного стенда EV8031. Первая задача, которую нужно выполнить – определение оборудования системы, способы адресации к устройствам и распределение адресного пространства.

Распределение адресного пространства внешней памяти в стенде выполняет дешифратор адреса DD7 (Рис. 24). Своими входами он подключен к старшей тетраде 16-разрядного внешнего адреса. На входе управления E3 будет логическая «1», разрешающая работу дешифратора, если адресуемое устройство имеет адрес больше 7FFFh. Т. о. дешифратор распределяет адресное пространство старшей половины внешних адресов микроконтроллера. В зависимости от комбинации уровней на входах 0, 1 и 2 дешифратора, подключенных к шине адреса, появляется низкий уровень только на одном выходе, номер которого в двоичной форме задан на входах. Сигналы с выходов дешифратора называются $\overline{CS}$ (Chip Select) и имеют свой номер. Каждый сигнал $\overline{CS}$ соответствует только своему адресному пространству. Низкий уровень этого сигнала указывает на

адресацию к устройству, которое подключается к шинам микропроцессора данным сигналом. Адресные пространства, соответствующие сигналам $\overline{CS}$ указаны на рис. 24:

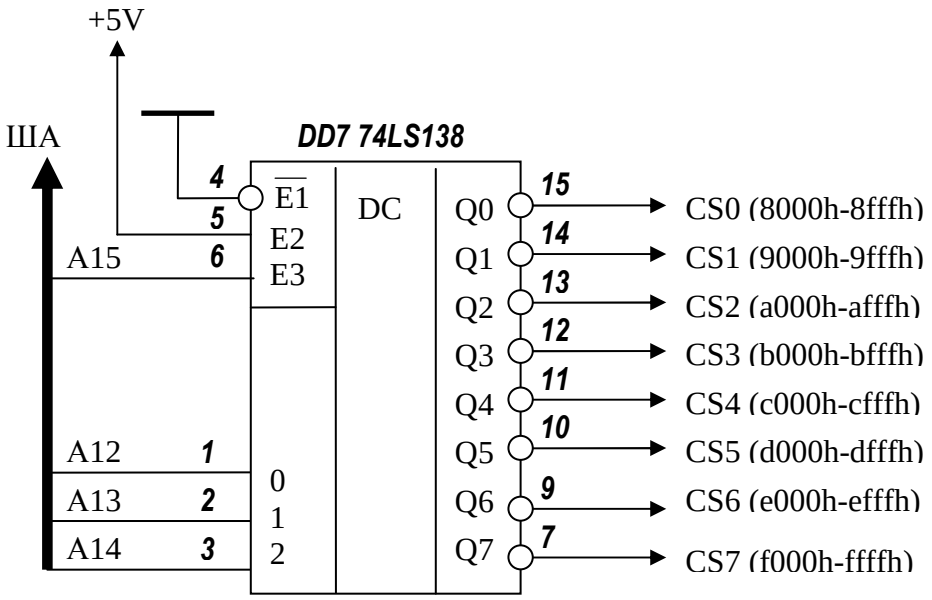


Рисунок 24 – Дешифратор адресного пространства внешней памяти стенда

Анализируя принципиальную схему стенда, можно увидеть, что к сигналу $\overline{CS0}$ подключена микросхема параллельного периферийного адаптера КР580ВВ55; к $\overline{CS1}$ – буфер чтения клавиатуры; $\overline{CS2}$ – старшая пара статических индикаторов; $\overline{CS3}$ – младшая пара статических индикаторов.

Для простоты схемы дешифрации адреса (в данном случае она построена только на микросхеме DD7), дешифратор выбирает из адресной шины некоторый промежуток адресов, в котором можно поместить множество устройств. Физически в таком диапазоне находится только одна (в некоторых случаях – несколько) ячеек. Поэтому для адресов обращения при программировании принимают только первую (несколько первых) ячеек. К примеру, для записи числа в регистр-защелку статического индикатора можно обратиться по адресу внешней памяти 0A000h, 0A001h и т. д. до 0AFFFh. Тогда при программировании принимают только адрес 0A000h.

Рассмотрим несколько примеров. Пусть нам необходимо засветить число на статических индикаторах HG3 и HG4. Для этого нам необходимо загрузить двоично-десятичное число в аккумулятор и записать эти данные в ячейку внешней памяти данных по адресу 0B000h согласно анализу

принципиальной схемы. Расположение статических индикаторов на стенде и их нумерация указана на рис. 25.

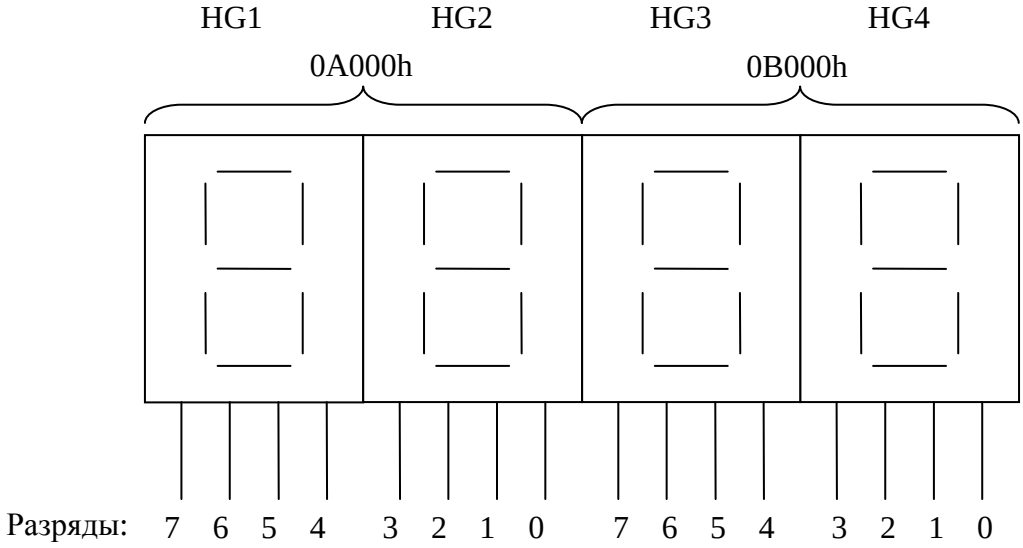


Рисунок 25 – Статические индикаторы

Пример участка программы:

```
Mov A,#56               ; Число, которое выводим на индикатор.
Mov DPTR,#0b000h        ; Адрес регистра индикатора.
Movx @DPTR,A            ; Вывод на индикатор.
```

Данные, выводимые на эти индикаторы, должны быть представлены в двоично-десятичной форме (тетрады имеют значения 0-9). При выводе шестнадцатеричного числа, код тетрады которого больше 9 (Ah-Fh) дешифратор индикатора не сможет преобразовать число в правильную комбинацию сигналов для отображения цифры и сегменты индикатора будут потушены.

На основе предыдущего примера можно изменить условие. Пусть нам теперь необходимо вывести тоже число, только индикатор HG3 должен не светиться. Для выполнения этой задачи необходимо, чтобы старшая тетрада числа была равна Fh. Этого можно добиться путем выполнения операции «логическое ИЛИ» над числом и константой F0h. Такой способ в программировании называется **маскирование**:

```
Mov A,#56               ; Число, которое выводим на индикатор.
Orl A,#0F0h             ; Налаживаем маску - тушим старшую тетраду.
Mov DPTR,#0b000h        ; Адрес регистра индикатора.
Movx @DPTR,A            ; Вывод на индикатор.
```

С помощью операции «логическое И» можно выполнять сброс битов числа или тетрады, а операцией «исключающее ИЛИ» – инверсию битов.

Теперь приведем пример написания программы секундомера. Индикаторы HG1 и HG2 отображают секунды, HG3 и HG4 – десятые и сотые секунды. Кнопки клавиатуры выполняют управление: «1» – старт; «2» – стоп; «3» – сброс показаний.

Секундомер обязательно должен содержать устройство измерения времени, которое в свою очередь всегда состоит из генератора эталонных интервалов времени и счетчика этих интервалов. Схема устройства измерения времени приведена на рис. 26:

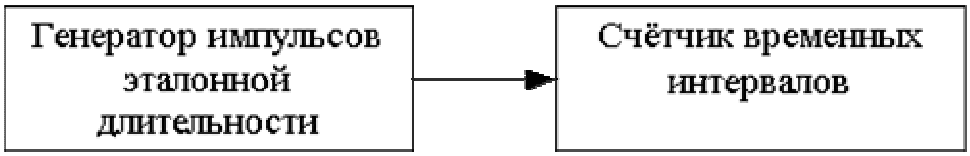


Рисунок 26 – Структурная схема устройства измерения времени

Генератор импульсов эталонной длительности должен вырабатывать импульсы с частотой 100 Гц, что соответствует дискретности времени 0,01 сек. Частота тактового генератора стенда составляет 7 372 800 Гц. Внутри микроконтроллера есть встроенный делитель импульсов на 12, который тактирует счетный вход таймеров. Для определения константы, загружаемой в таймер для формирования счетного импульса можно воспользоваться формулой:

$$K = 65535 - \frac{F_{OSC}}{12 \cdot F_T}$$

где F_{OSC} – частота кварцевого генератора;

F_T – необходимая частота после деления таймером.

Подставив числовые значения, получим:

$$65535 - \frac{7372800}{12 \cdot 100} = 59390 = 0E7FE_H .$$

Учитывая это, структурная схема устройства примет вид, указанный на рис. 27. Все элементы данного устройства, кроме дешифратора и индикаторов необходимо выполнить программно. Для такого выполнения разобьем устройство на составные части. Генератор эталонных импульсов (ГЭИ) представляет таймер микроконтроллера, генерирующий импульсы с частотой 100 Гц. Эти импульсы мы будем подавать на программный счетчик с коррекцией показаний секунд (после 59 секунды принимает значение 0). ГЭИ и схему сканирования клавиатуры выполним в одном потоке – подпрограмме прерывания от переполнения T/C0. Это позволит избавиться от алгоритма устранения дребезга контактов. Счетчик и схему управления – в другом потоке: это будет основная программа.

Для такой организации программы необходимы ячейки памяти для предоставления необходимой информации между потоками (синхронизация потоков). Такая ячейка у нас будет называться STATUS. Она будет представлять собой набор битов, устанавливающихся и

сбрасывающихся программой в определенных ситуациях. Необходимы следующие биты:

Count – Бит, устанавливаемый п/п обработки прерываний переполнения таймера, для отсчета сотых долей секунды; Сбрасывает основная программа после выполнения инкремента счетчика;

KeyRun – обнаружено нажатие кнопки «1». П/п прерывания сообщает основной программе о нажатии кнопки;

KeyStop – обнаружено нажатие кнопки «2». П/п прерывания сообщает основной программе о нажатии кнопки;

KeyReset – обнаружено нажатие кнопки «3». П/п прерывания сообщает основной программе о нажатии кнопки;

RunProcessed – флаг обработки кнопки «1». Необходим для однократного выполнения события нажатия при удержании нажатой кнопки. Сбрасывается когда KeyRun=0 (кнопка отжалась);

StopProcessed – флаг обработки кнопки «2». Сбрасывается когда KeyStop=0;

ResetProcessed – флаг обработки кнопки «3». Сбрасывается когда KeyReset=0;

Run – флаг, разрешающий счет времени.

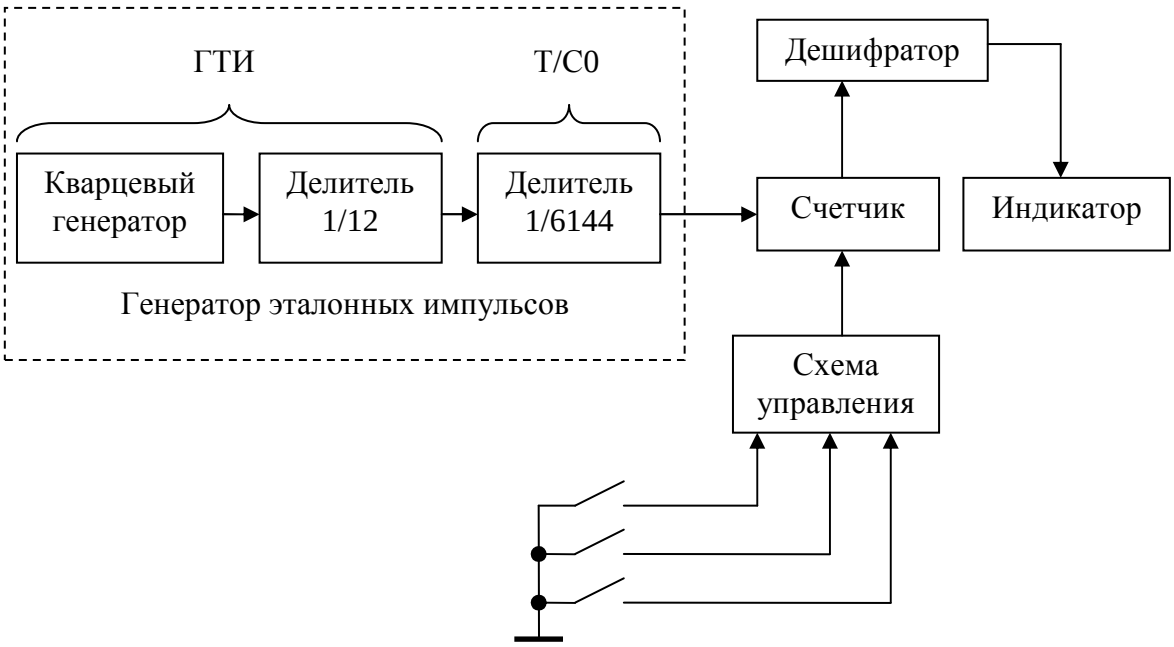


Рисунок 27 – Уточненная структурная схема устройства измерения времени

Кроме регистра STATUS нам необходимо определить, где будут храниться числа, выводимые на индикатор. Примем регистр R7 для счета долей секунд, R6 – для счета секунд. Содержимое этих регистров будем выводить на индикаторы. Для дальнейшего написания программы нам нужен алгоритм:

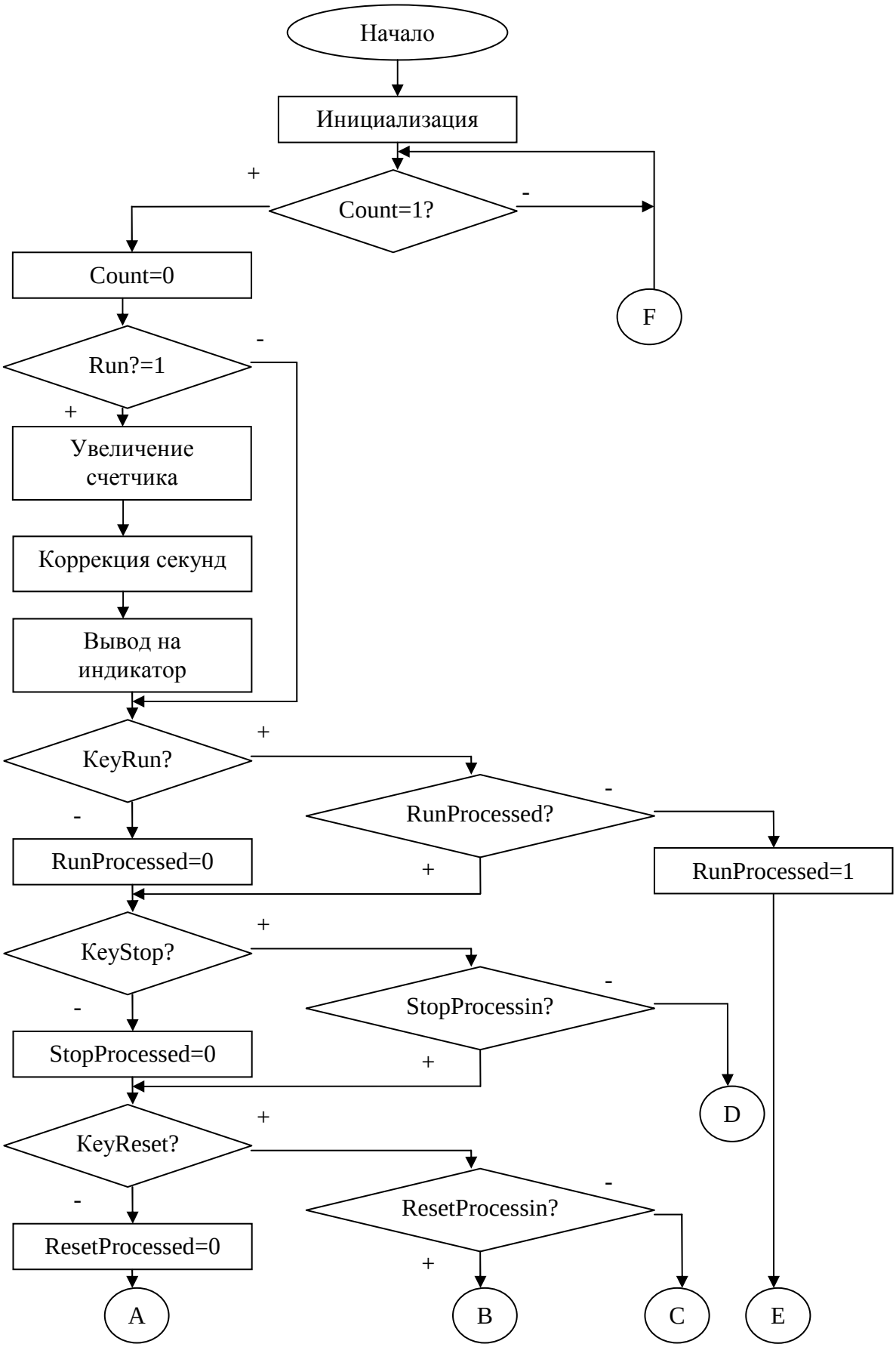


Рисунок 28 – Блок-схема алгоритма основной программы секундомера

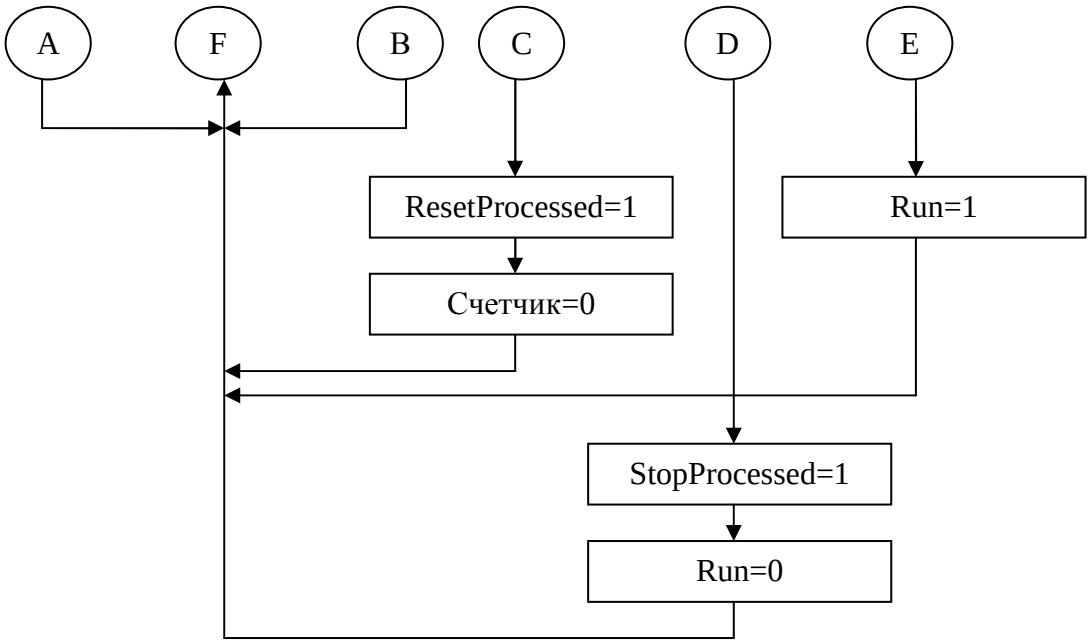


Рисунок 28, лист 2 – Блок-схема алгоритма основной программы секундомера

Алгоритм подпрограммы обслуживания прерывания приводить не будем. Он строится на аналогичных принципах. Данная подпрограмма должна выполнить перезагрузку счетчика T/C0, установить флаг Count, просканировать клавиатуру и, если нажата какая-либо клавиша, установить соответствующий бит в ячейке STATUS; если клавиша не нажата – сбросить соответствующий бит. С помощью этой ячейки выполняется своеобразный «мостик» для обмена информацией между потоком в прерывании и основной программой (Рис. 29). Иногда такие ячейки называют **регистрами событий**, а биты – **флагами событий**.

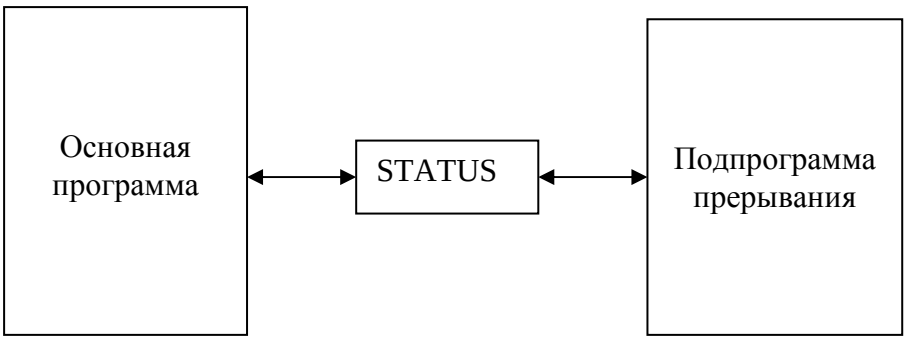


Рисунок 29 – Связь между потоком основной программы и потоком прерывания

При этом у каждого потока могут быть свои независимые переменные. В нашем случае это регистры счетчика R6 и R7 для основной программы и регистры таймера TH0 и TL0 для потока прерывания.

Составляем программу секундомера на языке ассемблер MCS-51:

```
$nolist
$include(mod51)
$list
;Программа реализации секундомера на стенде EV8031
;-------------------------------------------------------------------------
;Определение переменных
TMConst          set   6144  ;Общий коэффициент деления тактовой
                             ;частоты Fosc/(12*Ft)
                             ;Fosc=7372800 Гц, Ft=100 Гц
L_IND            equ   0a000h;Адреса С_Инд. во внешней памяти
R_IND            equ   0b000h
Low_Counter      equ   R7    ;Назначение имен регистрам
High_Counter     equ   R6
But1             equ   9006h ;Адрес обращения к кнопке 1
But2             equ   But1-1;Адрес обращения к кн.2=Адр.Кн1-1
But3             equ   But1-3;Адрес обращения к кн.3
DownStek         EQU   127   ;Определяем дно стека
;Битовая адресация к флагам в ячейке РПД 20h STATUS:
Count            bit   0     ;Флаг счета
KeyRun           bit   1     ;Флаг нажатия кнопки запуска
KeyStop          bit   2     ;Флаг нажатия кнопки останова
KeyReset         bit   3     ;Флаг нажатия кнопки сброса
RunProcessed     bit   4     ;Флаг обработки события запуска
StopProcessed    bit   5     ;Флаг обработки события останова
ResetProcessed   bit   6     ;Флаг обработки события сброса
Run              bit   7     ;Флаг разрешения счета времени
;-------------------------------------------------------------------------
;Точка входа в программу по вектору сброса процессора
      Org 0
      jmp Main
;-------------------------------------------------------------------------
;Точка входа в прерывание процессора по переполнению Т/С0
      Org 0bh
TMR0_Int:                    ;Точка входа в п/п прерывания Т/С0
      Push PSW               ;Сохранение состояния программы
      Push ACC
      Push DPL
      Push DPH
      CLR TR0                ;Останавливаем таймер
      Mov TL0,#LOW(NOT(TMConst+1))  ;загружаем константы
      Mov TH0,#HIGH(NOT(TMConst+1))
      SETB TR0               ;Запускаем таймер
      SETB Count             ;Сообщаем, что прошла 1/100 секунды
      Mov DPTR,#But1         ;Адрес обращения к буферу клавиатуры
      Movx A,@DPTR
      Jb ACC.0, NoRun        ; Кн.1 нажата?
      SETB KeyRun            ;Да, установим флаг
      sjmp KEY2
NoRun:CLR KeyRun             ;Нет, сбросим флаг
KEY2: Mov DPTR,#But2         ;Адрес обращения к буферу клавиатуры
      Movx A,@DPTR
      Jb ACC.0, NoStop       ; Кн.2 нажата?
      SETB KeyStop           ;Да, установим флаг
      sjmp KEY3
NoStop:CLR KeyStop           ;Нет, сбросим флаг
KEY3: Mov DPTR,#But3         ;Адрес обращения к буферу клавиатуры
      Movx A,@DPTR
      Jb ACC.0, NoReset      ; Кн.3 нажата?
```

```
      SETB KeyReset                ;Да, установим флаг
      sjmp DoneKey
NoReset:CLR KeyReset               ;Нет, сбросим флаг
DoneKey:Pop DPH                    ;Возврат состояния программы
      Pop DPL
      Pop ACC
      Pop PSW
      RETI
;------------------------------------------------------------------------
;Основная программа
;Здесь начинается инициализация
Main: Mov SP,#DownStek-28          ;Настройка указателя стека
      Mov TMOD,#00000001b          ;Т/С0, режим 1 (16-разрядов)
      Mov TL0,#LOW(NOT(TMConst+1)) ;загружаем константы
      Mov TH0,#HIGH(NOT(TMConst+1))
      SETB IT0                     ;Разрешение прерываний от Т/С0
      SETB TR0                     ;Запуск таймера
      SETB EA                      ;Разрешение прерываний
      CLR A
      Mov 20h,A                    ;Отчистка управляющих битов STATUS
Clear:Mov High_Counter,A           ;Отчистка регистров счетчика
      Mov Low_Counter,A
      Mov DPTR, #L_IND             ;Вывод на индикаторы 0
      Movx @DPTR,A
      Mov DPTR,#R_IND
      Movx @DPTR,A
;Конец инициализации. Начало основной программы
Wait: Jnb Count,$                  ;Ожидание 1/100 сек.
      CLR Count
      Jnb Run, KeyProcessed        ; Счет разрешен?
;Здесь начинается счетчик долей и секунд
      Inc Low_Counter              ;Да, увеличиваем сотые сек.
      Mov A,Low_Counter
      DA A                         ;Двоично-десятичная коррекция
      Mov Low_Counter,A
      JNC NoCarry                  ; Число больше 99?
      Mov A, High_Counter          ;Да, увеличим счетчик секунд
      INC A
      DA A                         ;Двоично-десятичеая коррекция
      CJNE A,#60h,Less60           ;Секунд 60?
      CLR A                        ;Да, сбросим старший разряд счетчика
Less60:Mov High_Counter,A          ;Нет, сохраним результаты
NoCarry:Mov A,High_Counter
      Mov DPTR, #L_IND             ;Вывод на индикаторы секунд
      Movx @DPTR,A
      Mov A, Low_Counter           ;и долей секунд
      Mov DPTR,#R_IND
      Movx @DPTR,A
;Обработка нажатий кнопок
KeyProcessed:
IsRun?:Jb KeyRun, SetRun           ;Нажата кнопка запуска?
      CLR RunProcessed             ;Нет, отчистить флаг обработки
IsStop?:Jb KeyStop, SetStop        ;Нажата кнопка стоп?
      CLR StopProcessed            ;Нет, отчистить флаг обработки
IsReset?:Jb KeyReset,SetReset      ;Нажата кнопка сброс?
      CLR ResetProcessed           ;Нет, отчистить флаг обработки
      Jmp Wait                     ;Кнопки не нажаты. Ожидаем 1/100 сек.

SetRun:Jb RunProcessed,IsStop?     ;Кнопка была отпущена?
      SETB Run                     ;Да, разрешим счет.
      SETB RunProcessed            ;и установим флаг обработки кнопки
      Jmp Wait

SetStop:Jb StopProcessed,IsReset?  ;Кнопка была отпущена?
```

```
      CLR Run                         ;Да, запретим счет.
      SETB StopProcessed              ;и установим флаг обработки кнопки
      Jmp Wait

SetReset:Jb ResetProcessed, Wait      ;Кнопка была отпущена?
      CLR A                           ;Да, отчистим аккумулятор
      SETB ResetProcessed             ;и установим флаг обработки кнопки
      Jmp Clear                       ;Прыжок на отчистку счетчика
;Конец программы
;-------------------------------------------------------------------------
END
```

Вводим исходный текст программы в редакторе текстовых файлов «Блокнот» (или любом другом) и сохраняем файл с расширением *.asm

Для компиляции программы в командной строке операционной системы вводим:

Asm51.exe [путь к файлу]SecMeter.asm

Результатом компиляции являются файлы, имеющие название исходной программы и расширения *.HEX и *.LST. Первый из них – полученный код программы в формате Intel Hex, а второй – листинг компиляции, в котором компилятор указывает на ошибки (если они есть) и другую служебную информацию. Если эти файлы не созданы, то процесс компиляции не мог начаться из-за ошибки. Наиболее частыми причинами таких ошибок бывают:

– Отсутствуют по указанному пути файлы, объявленные директивой $include;

– В пути к исходному файлу программы или ее модулей есть пробелы, русские символы или другие элементы, не подходящие под формат файлов DOS8.3.

Если программа пишется объектным кодом ассемблера (используются описатели сегментов кода, директивы резервирования переменных), то процесс компиляции выполняется в несколько этапов. На первом этапе с помощью компилятора получают объектный код каждого модуля программы с расширением *.obj. Далее объектный код модулей с помощью линкера соединяется в один файл, из которого получают *.hex файл.

3.3 Оптимизация кода программы и стиль программирования

Критерии оптимизации программ с точки зрения экономии ресурсов должны выбираться в зависимости от конкретных обстоятельств. Задачей программиста является экономия таких ресурсов, как время работы программы и используемые объемы ОЗУ и ПЗУ. В отличие от больших проектов программирование для микроконтроллеров осуществляется, как правило, или одним программистом, или небольшой группой программистов. Хороший стиль программирования всегда полезен. Он важен для экономии времени, затрачиваемого на разработку программы, ее отладку и поддержку, необходимую в процессе усовершенствования

изделия по результатам его эксплуатации или в случае введения доработок в аппаратуру.

Стиль программирования в значительной степени определяет успехи программиста и отражает его индивидуальность. Неряшливость в программировании тоже можно считать стилем, но это плохой стиль. Для улучшения стиля программисту время от времени необходимо осмысливать проделанную работу не только с точки зрения достигнутых результатов, но и с точки зрения затраченных на это средств. Усвоение хорошего стиля программирования должно идти параллельно с обучением программированию на ассемблере. Стиль характеризует творческую сторону работы программиста, поэтому строгих правил здесь нет. Однако каждый программисту позволено применять те или другие советы по-своему или не применять их. Важно не сотворить себе кумира и по мере накопления опыта создавать удобные для себя правила и приемы программирования.

4 ПРАКТИКУМ ПО АССЕМБЛЕРУ MCS-51

4.1 Требования к отчетам по практикуму

В процессе выполнения практических заданий каждый студент должен внимательно изучить задание своего варианта, выполнить анализ поставленной задачи и найти способы решения. В случае возникновения неясностей в задании, студент может проконсультироваться у преподавателя.

Каждый отчет должен содержать:

– Тему практического задания;

– Цель практического задания;

– Описание полученного задания;

– Блок-схему алгоритма решения задания;

– Описание переменных, используемых в программе, описание назначения регистров, хранящих данные. Описание регистров, которые используются временно для каких-либо целей, допускается не указывать. Пример описания переменных:

R6 – Слагаемое №1;

R7 – Слагаемое №2;

R5 – Хранит код единиц числа для отображения на семисегментном индикаторе;

KeyPress – флаг нажатия клавиш.

– Исходный текст программы на языке ассемблер;

– Выводы о проделанной работе.

При написании подпрограмм в программе в поле комментариев перед самой подпрограммой необходимо указывать входные и выходные данные:

```
;Decode-Подпрограмма перевода числа в код семисегментного индикатора
;Вход:  A - ДД число формата 0X
;Выход: A - семисегментный код числа
;Изменяет: DPTR
Decode:                              ;Здесь начинается подпрограмма
```

Практические задания построены для выполнения на учебно-отладочных стендах EV8031/AVR фирмы Open Systems.

4.2 Практическое задание №1

Тема: Система команд ОЭВМ MCS-51. Полноэкранный отладчик FD51.

Цель: Знакомство с ассемблером семейства микроконтроллеров MCS-51. Получение навыков в программировании и отладке программ для микроконтроллеров MCS-51.

Таблица 3 – Варианты заданий к самостоятельной работе

Вариант	**Задание**
1	Разработать программу подсчета положительных чисел в массиве.
2	Разработать программу подсчета отрицательных чисел в массиве
3	Разработать программу подсчета количества нулевых элементов в массиве
4	Разработать программу подсчета количества элементов массива, отличных от нуля
5	Разработать программу подсчета четных чисел в массиве
6	Разработать программу подсчета нечетных чисел в массиве
7	Разработать программу подсчета нулевых битов в байтах массива
8	Разработать программу подсчета единичных битов в байтах массива
9	Разработать программу поиска количества элементов массиве, равных первому элементу
10	Разработать программу сортировки по возрастанию массива
11	Разработать программу сортировки по убыванию в массиве
12	Разработать программу заполнения массива числами в арифметической прогрессии
13	Разработать программу заполнения массива нулями
14	Разработать программу, реализующую сумму всех элементов массива
15	Разработать программу, реализующую разность всех элементов массива
16	Разработать программу поиска наибольшего числа в массиве
17	Разработать программу поиска наименьшего числа в массиве
18	Разработать программу вычисления среднего арифметического массива (число элементов принять равное 8)
19	Разработать программу сложения элементов 2х массивов. Результат заносить в 1й массив.
20	Разработать программу перемещения блока памяти на 16 байт вперед
21	Разработать программу сравнения двух массивов. Выделить переменную для результата и приравнять ее к 1 если массивы одинаковы, иначе она равна 0
22	Разработать программу логического сдвига всех битов в массиве влево
23	Разработать программу логического сдвига всех битов в массиве вправо
24	Разработать программу вычисления суммы отдельно положительных и отрицательных элементов массива

Продолжение таблицы 3

25	Разработать программу вычисления суммы отдельно четных и нечетных элементов массива
26	Разработать программу преобразования числа из двоичной формы в двоично-десятичную
27	Разработать программу заполнения элементов массива по возрастанию
28	Разработать программу заполнения элементов массива по убыванию
29	Разработать программу нахождения суммы наибольшего и наименьшего элементов массива.
30	Разработать программу нахождения разности наибольшего и наименьшего элементов массива.

Примечание: если в задании варианта не указано количество элементов массива, то рекомендуется принять 10.

4.3 Практическое задание №2

Тема: Изучение стенда и команд ОЭВМ КР1816ВЕ31.

Цель: Изучение функциональных возможностей учебно-отладочного стенда, внутренней структуры и системы команд ОЭВМ КР1816ВЕ31.

Таблица 4 – Варианты заданий к самостоятельной работе

Вариант	**Задание**
1	Занести в регистр R4 ДД число 0X, в R6 – X0. Сумму чисел отобразить на С_Инд. HG1, HG2 с попеременным миганием тетрад с частотой 1 Гц.
2	Занести в регистр R4 число XXh. Выполнить двоично-десятичное преобразование числа и вывести результат на С_Инд. (HG2, HG3, HG4 – сотни, десятки и единицы соответственно)
3	Занести в регистр B ДД число XX, в R2 – 0X. Результат разницы чисел B-R2 с двоично-десятичной коррекцией результата вывести на С_Инд. HG1, HG2 с миганием 2 Гц.
4	Занести в A ДД число XX, в регистр R5 – X0. Число из A отобразить на С_Инд. HG1, HG2, число из R5 – попеременно отображать на С_Инд. HG3, HG4 с частотой 0,5 Гц.
5	Занести в регистр R2 ДД число 0X, в R5 X0. Сумму чисел попеременно отображать на С_Инд. HG1, HG2 и HG3, HG4 с частотой 1 Гц.

Продолжение таблицы 4

6	Занести в ячейку внутренней памяти с адресом 70h ДД число 0X, в регистр R3 – X0. Сумму чисел попеременно отображать на C_Инд. HG2, HG3 и HG1 HG4 с частотой 2 Гц.
7	Занести в регистр R0 число 0. прибавлять к регистру единицу с частотой 4 Гц до 99, выполнять двоично-десятичную коррекцию и выводить результат на C_Инд. HG1, HG2.
8	Занести в регистр B ДД число X0, в регистр R1 – XX. Число из B отображать на C_Инд. HG1 с частотой 1 Гц, число из R1 отображать на C_Инд. HG3, HG4 с частотой 0,5 Гц.
9	Прочитать значение регистра PSW, сбросить старшие биты тетрад прочитанного значения и отображать его на C_Инд. В следующем порядке: HG1, HG2→HG2, HG3→HG3, HG4 с временем смены индикаторов 0,5 сек.
10	Занести в регистр R4 ДД число 0X, в регистр R1 X0. Сумму чисел отображать на C_Инд. HG1, HG2 с плавным угасанием числа на протяжении 5 сек.
11	Занести в A ДД число X0, в регистр B – 0X. Младшую тетраду суммы постоянно отображать на C_Инд. HG4, а старшую – попеременно на индикаторах HG1, HG2 и HG3 с интервалом в 0,5 сек.
12	Занести в регистр B ДД число 0X, в регистр R5 – X0. Старшую тетраду суммы постоянно отображать на C_Инд. HG1, а младшую – попеременно на индикаторах HG4, HG3 и HG2 с интервалом в 1 сек.
13	Занести в регистр R1 ДД число 0X. Отнимая от числа единицу отображать результат на C_Инд. HG2 до нуля с частотой 1 Гц. После этого потушить индикатор на 2 сек. и выполнять программу заново.
14	Занести в регистр R3 ДД число XX, в регистр R5 – XX. Попеременно отображать эти числа на C_Инд. HG4,HG3 и HG2,HG1 с частотой 2 Гц. После 10 сек. потушить индикаторы на 2сек., зациклить программу.
15	Занести в регистр A ДД число 0X, в регистр R2 – X0. Число из A попеременно отображать на C_Инд. HG2, HG1 с частотой 1 Гц, из R2 – на C_Инд. HG3, HG4 с частотой 2 Гц.
16	Занести в регистр R4 ДД число X0, в R6 – 0X. Разницу чисел с двоично-десятичной коррекцией отобразить на C_Инд. HG1, HG2 с попеременным миганием тетрад с частотой 1 Гц.
17	Занести в регистр R1 ДД число 0X, в регистр R5 X0. Десятки суммы последовательно отображать на всех C_Инд. с задержкой времени 0,25 сек.

Продолжение таблицы 4

18	Занести в регистр А ДД число 0X, в R5 X0. Сумму чисел попеременно отображать на С_Инд. HG1, HG3 и HG2, HG4 с частотой 2 Гц.
19	Занести в ячейку внутренней памяти с адресом 20h ДД число X0, в регистр R3 – 0X. Разницу чисел с двоично-десятичной коррекцией попеременно отображать на С_Инд. HG2, HG3 и HG1 HG4 с частотой 2 Гц.
20	Занести в регистр R0 ДД число 10. отнимать от регистра единицу с частотой 2 Гц до 0, затем прибавлять до 10. Выводить результаты на С_Инд. HG1, HG2.
21	Занести в регистр В ДД число XX, в регистр R1 – 0X. Число из В отображать на С_Инд. HG1, HG2 с частотой 1 Гц, число из R1 отображать на С_Инд. HG3, HG4 с частотой 0,5 Гц.
22	Прочитать значение регистра SP, отображать его значение на С_Инд. В следующем порядке: HG3, HG4→HG2, HG3→HG1, HG2 с временем смены индикаторов 1 сек.
23	Занести в регистр R1 ДД число 0X, в регистр R2 X0. Сумму чисел отображать на С_Инд. HG1, HG4 с плавным зажиганием числа на протяжении 5 сек.
24	Занести в В ДД число 0X, в регистр R1 – X0. Младшую тетраду суммы постоянно отображать на С_Инд. HG1, а старшую – попеременно на индикаторах HG4, HG3 и HG2 с интервалом в 1 сек.
25	Занести в регистр TH0 ДД число X0, в регистр R2 – 0X. Старшую тетраду суммы постоянно отображать на С_Инд. HG1, а младшую – попеременно на индикаторах HG2, HG3 с интервалом в 1 сек.
26	Занести в регистр R1 число 0. Каждые 0,5 сек к числу прибавлять 1. Каждые 5 сек прибавлять 10. Счет вести в двоично-десятичном коде. Результат отображать на С_Инд. HG3, HG4.
27	Занести в регистр R1 число 0. Каждые 0,5 сек к числу прибавлять 1. Счет вести в двоично-десятичном коде. Результат отображать на С_Инд. HG3, HG4. По достижении числа 20 остановить счет и отображать результат с мерцанием в 0,5 сек.
28	Занести в регистр R2 ДД число XX, в регистр R3 – XX. Одновременно отображать эти числа на С_Инд. HG3,HG4 и HG1,HG2 с частотой 1 Гц. После 5 сек. потушить индикаторы на 1сек., зациклить программу.
29	Занести в регистр В ДД число X0, в регистр R1 – 0X. Сумму чисел отобразить на С_Инд. HG1, HG2, А разность с двоично-десятичной коррекцией – на С_Инд. HG3, HG4

Продолжение таблицы 4

30	Занести в регистры R1 и R4 числа 0Xh. После умножения выполнить двоично-десятичное преобразование результата и вывести результат на С_Инд. (HG2, HG3, HG4 –сотни, десятки и единицы соответственно).

Примечания: 1. ДД – двоично-десятичное число (числа в тетрадах 0…9);
2. XXh – произвольное шестнадцатеричное число;
3. ДД 0X – двоично-десятичное число старшая тетрада которого равна 0, а младшая имеет значения 0…9;
4. Индикаторы HG1 и HG2 имеют адреса во внешней памяти данных 0A000h (старшая и младшая тетрады соотв.);
5. Индикаторы HG3 и HG4 имеют адреса во внешней памяти данных 0B000h (старшая и младшая тетрады соотв.).

4.4 Практическое задание №3

Тема: Программирование параллельного интерфейса КР580ВВ55.

Цель: Изучение схем динамической индикации.

Таблица 5 – Варианты заданий к самостоятельной работе

Вариант	**Задание**
1	Занести в регистр R1 число XXh, отнимая от числа единицу отображать на Д_Инд. HG5, HG6 полученное значение до нуля с частотой 1 сек. С частотой 0,5 Гц. Переключать Свет. HL3, HL4.
2	Занести в В ДД число X0, в регистр R1 XXh, число из В. отображать на С_Инд. HG1 с частотой 1 Гц, число из R1 отображать на Д_Инд. HG5, HG6 с частотой 0,25 Гц.
3	Вкл. Свет. HL1. Занести в регистр В ДД число 0X, в регистр R5 X0, два разряда суммы (десятки и единицы) поочередно отображать на С_Инд. HG1, HG2 и на Д Инд. HG5, HG6 с частотой 1 Гц.
4	Занести в рег. R6 ДД число XX, в рег.R5 ДД XX, в регистр R0 ДД XX, последовательно отображать эти числа по одному разряду на на Д_Инд. HG5, HG6, С_Инд. HG1, HG2, HG3, HG4.
5	Попеременно вкл. Свет. HL8-HL7 с частотой 2 Гц. Занести в регистр R2 ДД число 0X, в регистр R1 0X, сумму чисел отобразить на Д_Инд. HG5, HG6.
6	Занести в регистр А ДД число 0X, в регистр R2 X0, число из А отобразить на С_Инд. HG3, число из регистра R2 отображать на Д_Инд. HG5, HG6 с частотой в 0.25 Гц.

Продолжение таблицы 5

7	Занести в А ДД число XX, в регистр R1 XX, младшие два разряда суммы чисел отобразить на Д_Инд. HG5, HG6, старшую тетраду С Инд. HG3.
8	Занести в регистр R6 число XXh. Число из регистра R6 вывести в шестнадцатеричной форме на Д_Инд. HG5, HG6, а его двоично-десятичный эквивалент на С_Инд. HG2-HG4.
9	Занести в регистр В ДД число XX, в регистр R3 XX, разность чисел отобразить на Д_Инд. HG2, HG3 в ДД форме.
10	Вкл. Свет. HL5. Занести в А ДД число XX, в регистр R5 X0, число из А отобразить на С_Инд. HG1, HG2, число из R5 отобразить на Д_Инд. HG5,HG6.
11	Занести в регистр R0 ДД число XX, попеременно отображать мл. и ст. тетраду на Д_Инд. HG5, HG6 с частотой 0,25 Гц.
12	Занести в регистр R2 ДД число 0X, в регистр R5 0X, сумму чисел отобразить на Д_Инд. HG2, HG3.
13	Занести в регистр В ДД число, с частотой 2 Гц выводить это число на С_Инд. HG1, HG2 и одновременно на Д_Инд. HG5, HG6.
14	Попеременно вкл. Свет. HL4-HL6. Занести в ячейку с адресом 0010h внешней памяти ОЭВМ ДД число 0X, в регистр R3 XXh, сумму чисел отобразить на Д_Инд. HG5, HG6.
15	Занести в регистр R1 ДД число 0X, в регистр R3 XX, младшие два разряда суммы отобразить на Д_Инд. HG5, HG6 с изменением частоты смены этих индикаторов, старшую на С Инд. HG1.

4.5 Практическое задание №4

Тема: Система прерываний. Опрос дискретных датчиков.

Цель: Изучение системы прерываний ОЭВМ КР1816ВЕ31 и программ опроса дискретных датчиков.

Таблица 6 – Варианты заданий к самостоятельной работе

Вариант	**Задание**
1	Прерывание INT0 увеличивает показания счетчика на 1, состояние которого выводится на С_Инд. HG3, HG4, а прерывание INT1 – уменьшает.
2	Реализовать опрос клавиатуры. Номер нажатой клавиши отображать зажиганием соответствующей точки на матричном индикаторе HG7.

Продолжение таблицы 6

3	Реализовать опрос клавиатуры. Номер нажатой клавиши отображать на С_Инд.
4	Прерывание INT0 запускает бегущий огонь на светодиодах HL1-HL8 с частотой 2 Гц, а INT1 – тушит все светодиоды.
5	Программа выполняет мерцание любого числа на С_Инд. Прерывание INT0 выполняет сдвиг бегущего огня на светодиодах HL1-HL8 на один шаг влево, а INT1 – вправо.
6	Реализовать опрос клавиатуры. Номер клавиши указывать позиционным кодом на светодиодах HL1-HL8.
7	Реализовать программу ввода двухзначного числа с клавиатуры используя С_Инд. для отображения.
8	Реализовать опрос клавиатуры. Номер клавиши указывать на Д_Инд. HG5, HG6.
9	Программа выполняет мерцание любого числа на С_Инд. Прерывание INT0 запускает бегущий огонь на матрице HG7, а INT1 – тушит все точки.
10	Программа выполняет бегущий огонь на светодиодах HL1-HL8. Прерывание INT0 останавливает огонь. Повторное INT0 запускает бегущий огонь
11	Прерывание INT0 засвечивает все точки на матрице HG7. Прерывание INT1 – тушит все точки.
12	Прерывание INT0 засвечивает точки в шахматном порядке на матрице HG7. Прерывание INT1 – тушит все точки.
13	Реализовать программу введения трехзначного числа с клавиатуры, используя С_Инд.
14	Прерывание INT1 выполняет запуск бегущей тени на матрице HG7. Повторный вызов прерывания тушит все точки.
15	Реализовать опрос клавиатуры. Номер клавиши указывать двоичным кодом на светодиодах HL1-HL8.

4.6 Практическое задание №5

Тема: Цифроаналоговое преобразование. Работа таймеров.

Цель: Изучение методов цифроаналогового преобразования. Знакомство с таймерами ОЭВМ КР1816ВЕ31.

Таблица 7 – Варианты заданий к самостоятельной работе

Вариант	**Задание**
1	Сформировать пилообразное напряжение с частотой повторения 50 Гц. Отображать на С_Инд число сгенерированных импульсов.
2	По нажатию S1 сформировать треугольные импульсы, передний фронт 20 мсек, задний 10 мсек.
3	По нажатию S2 сформировать трапециевидные импульсы, передний фронт 13 мксек. задний 15 мсек.
4	Сформировать синусоиду с частотой повторения 120 Гц.
5	Сформировать пилообразное напряжение с частотой повторения 200 Гц и длительностью переднего фронта 2 мсек.
6	По нажатию S3 сформировать синусоиду с частотой повторения 100 Гц.
7	Сформировать прямоугольные импульсы, с длительностью 25мсек и скважностью 4
8	По нажатию S4 сформировать треугольные импульсы, передний фронт 25 мсек., задний 5 мсек.
9	Сформировать синусоиду с частотой повторения 300 Гц. По нажатию S5 изменить частоту на 100 Гц.
10	Сформировать два прямоугольных импульса, один максимальной амплитудой длительностью и второй 2/3 амплитуды максимальной с периодом повторения 40 Гц.
11	Сформировать прямоугольные импульсы, с длительностью 25 мсек. по нажатию S6 сформировать треугольные импульсы.
12	Сформировать синусоиду с частотой повторения 70 Гц, по нажатию S7 прямоугольные импульсы, с длительностью 25мсек и скважностью 2.
13	По нажатию S8 сформировать треугольные импульсы, передний фронт 15 мсек., задний 40 мсек.
14	По нажатию S9 сформировать трапециевидные импульсы, передний фронт 40 мсек., задний 20 мсек.
15	Сформировать три прямоугольных импульса, один 1/3 макс. амплитуды, 2-ой 2/3 амп. макс, 3-ий макс. амп. с периодом повторения 100 Гц.

Примечание: Для формирования интервалов времени использовать внутренние таймеры микроконтроллера.

4.7 Практическое задание №6

Тема: Аналого-цифровое преобразование.

Цель: Научиться измерять аналоговую величину с помощью цифровых устройств.

Таблица 8 – Варианты заданий к самостоятельной работе

Вариант	**Задание**
1	При нажатии на кнопку S10 запустить АЦП последовательного уравновешивания. Результат преобразования выводить на С_Инд в десятичной форме.
2	При нажатии на кнопку S11 запустить АЦП половинных приближений. Результат преобразования выводить на С_Инд в десятичной форме.
3	Запустить следящий АЦП последовательного уравновешивания. Результат преобразования выводить на С_Инд в десятичной форме.
4	Запустить следящий АЦП половинных приближений. Результат преобразования выводить на С_Инд в десятичной форме.
5	По нажатию кнопки S1 запустить АЦП половинных приближений. Результат преобразования выводить на С_Инд в десятичной форме.
6	По нажатию кнопки S2 запустить АЦП последовательного уравновешивания. Результат преобразования выводить на С_Инд в десятичной форме.
7	По нажатию кнопки S3 запустить следящий АЦП половинных приближений. Результат преобразования выводить на С_Инд в десятичной форме.
8	По нажатию кнопки S4 запустить следящий АЦП последовательного уравновешивания. Результат преобразования выводить на С_Инд в десятичной форме.
9	По нажатию кнопки S5 запустить АЦП половинных приближений. Результат преобразования выводить на С_Инд в десятичной форме по нажатию кнопки S1.
10	По нажатию кнопки S6 запустить АЦП последовательного уравновешивания. Результат преобразования выводить на С_Инд в десятичной форме по нажатию кнопки S2.
11	По нажатию кнопки S7 запустить следящий АЦП последовательного уравновешивания. Результат выводить на С_Инд в десятичной форме по нажатию S3.

Продолжение таблицы 8

12	По нажатию кнопки S8 запустить следящий АЦП половинных приближений. Результат выводить на С_Инд в десятичной форме по нажатию S4.
13	При нажатии S9 запустить АЦП половинных приближений. Номер уравновешиваемого разряда выводить в качестве точки на матрицу HG7. После преобразования вывести результат на С_Инд в десятичной форме.
14	При нажатии S10 запустить АЦП последовательного уравновешивания. Номер шага уравновешивания выводить на матрицу HG7 в двоичном коде. После преобразования вывести результат на С_Инд в десятичной форме.
15	При нажатии кнопки S11 запустить АЦП половинных приближений. По нажатию S2 вывести результат на Д_Инд в шестнадцатеричной форме. Допускается использовать С_Инд для старшего разряда (HG1).

5 ОПИСАНИЕ МИКРОКОНТРОЛЛЕРА PIC16F877

5.1 Общая характеристика микроконтроллера

PIC16F877 – 8-разрядный микроконтроллер, выпускаемый фирмой Microchip Technology. Это специализированный микропроцессор, предназначенный в основном для программного управления автоматизированными системами, автомобильными и электрическими двигателями, устройствами передачи информации и измерительными приборами. В отличие от универсальных процессоров, он имеет развитые средства взаимодействия с внешними устройствами и более простую систему команд.

PIC16F877 представляет собой микросхему с 40 выводами, из которых 32 предназначены для передачи информации от внешнего устройства либо к внешнему устройству. Выполняемая программа хранится в перепрограммируемом ПЗУ, куда она заносится специальным устройством – программатором. Необходимые данные, переменные, результаты несложных расчетов и счетчики циклов хранятся в ОЗУ и теряются при выключении питания. Чтобы избежать потери данных при этом, можно использовать 256 ячеек энергонезависимой памяти данных.

Микроконтроллер PIC16F877 имеет следующие технические характеристики:

– Микроконтроллер выполнен по высокоскоростной RISC технологии. Высокая производительность достигается за счет применения конвейерной архитектуры и малого числа команд (всего 35);

– Тактовая частота МК составляет 20 МГц, при этом время длительности машинного цикла достигает 200 нс;

– 8Кх14 слов FLASH памяти программ;

– 368х8 байт памяти данных (ОЗУ);

– 256х8 байт EEPROM памятью данных;

– Система прерываний имеет 14 источников.

Характеристика периферийных модулей:

– Два 8-разрядных таймера/счетчика;

– Один 16-разрядный таймер/счетчик с возможностью подключения внешнего кварцевого резонатора;

– Два модуля захват/сравнение/ШИМ (CCP):

1) 16-разрядный захват (максимальная разрешающая способность 12.5 нс);

2) 16-разрядное сравнение (максимальная разрешающая способность 200нс);

3) 10 разрядный ШИМ.

– 8-канальный 10-разрядный АЦП;

– Последовательный синхронный порт;

– Ведущий/ведомый режим SPI;

– Ведущий/ведомый режим I^2C;

– Последовательный асинхронный приемопередатчик USART с поддержкой детектирования адреса;

– Ведомый 8-разрядный параллельный порт PSP с поддержкой внешних сигналов $\overline{RD}$, $\overline{WR}$, $\overline{CS}$.

5.2 Внутреннее устройство ядра микроконтроллера

Внутреннее устройство показано на рис. 30. МК можно условно разделить на две части: вычислительное ядро (серый цвет) и периферийные модули (белый цвет).

Вычислительное ядро работает следующим образом. Программа работы МК находится в FLASH памяти программ. Программа выполняется последовательно до тех пор, пока не встретится команда перехода. Регистр команд (РК) содержит текущую команду на время ее дешифрации и выполнения, а программный счетчик (PC) предназначен для хранения адреса следующей команды. Когда текущая команда завершена, то:

– По адресу из PC производится выборка команды из памяти программ в РК;

– При дешифрации этой команды, производится инкремент PC на единицу и PC адресует следующую команду;

– Когда выполнение данной команды заканчивается, содержимое PC выдается памяти программ и цикл повторяется.

Команды безусловного перехода позволяют изменить естественный порядок следования команд путем замещения содержимого PC (т. е. адреса следующей по порядку команды) адресом, определяемым самой командой перехода.

Команды условных переходов замещают или не замещают содержимое PC в зависимости от признаков результатов предыдущих команд. Признаки результатов предыдущих команд находится в регистре STATUS. В этом регистре имеются биты, показывающие такие условия, как получение в предыдущих операциях положительного, отрицательного или нулевого результата. Когда реализован переход, начинается новая последовательность команд с адреса, к которому осуществлен переход.

Циклы реализуются с помощью команд условных переходов.

Действия, связанные с вызовом подпрограммы, как и у других МК, также заменяет содержимое PC на адрес перехода, и также при этом запоминается текущее содержимое PC в стеке. Команда возврата должна восстановить в PC адрес возврата, чтобы после завершения подпрограммы продолжалось последовательное выполнение основной программы.

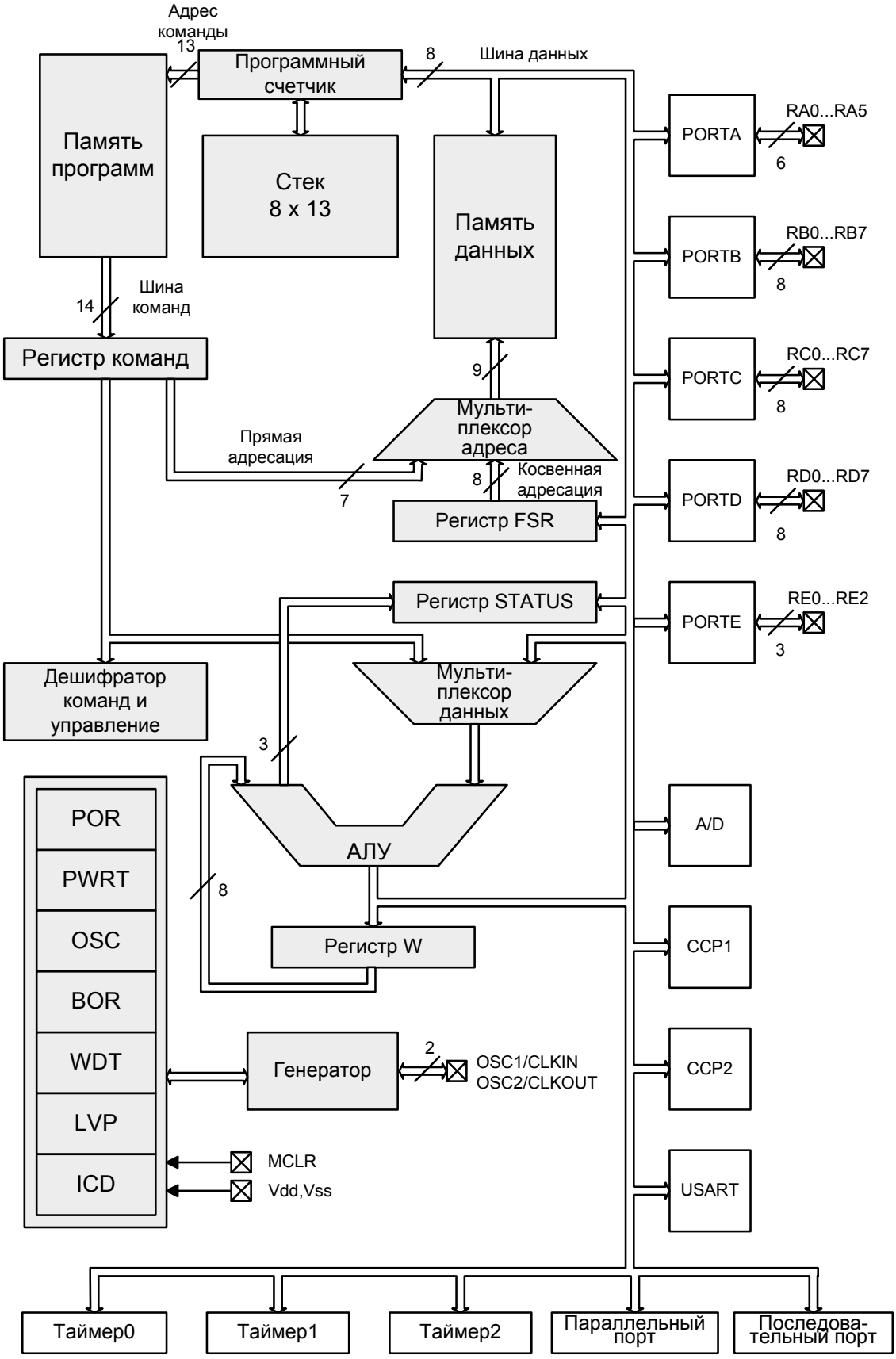


Рисунок 30 – Внутреннее устройство микроконтроллера PIC16F877

Извлечение команд из памяти команд и их выполнение ядром происходит за 4 машинных такта, благодаря конвейерной обработке, из

которых состоит машинный цикл. Диаграмма выборки команд и их исполнение представлена на рис. 31:

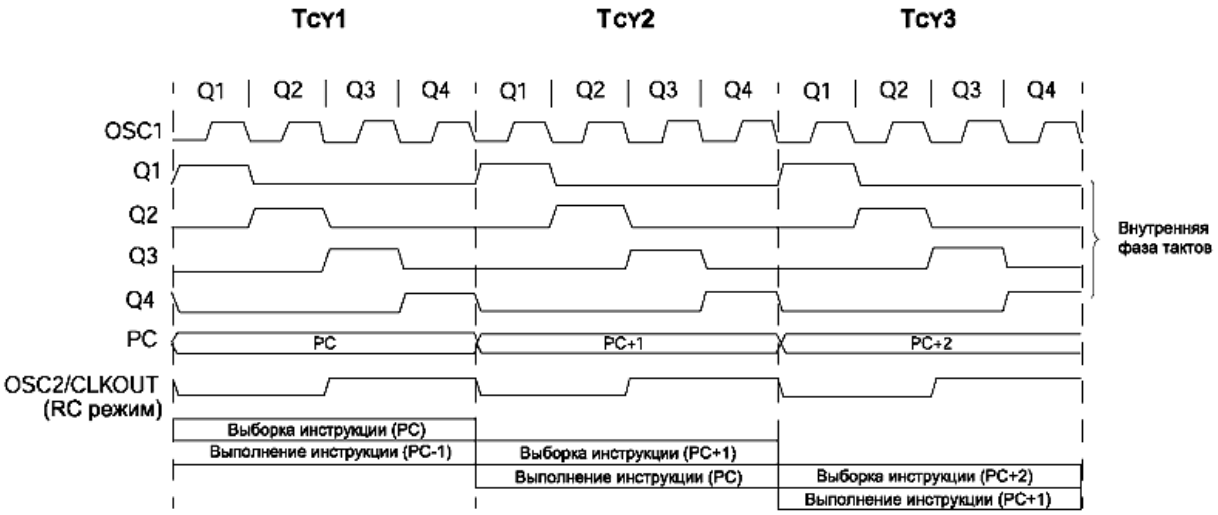


Рисунок 31 – Диаграмма машинных циклов

Микроконтроллеры PIC содержат 8-разрядный универсальный арифметический модуль (АЛУ) и 8-разрядный рабочий регистр (W). АЛУ выполняет арифметические и булевы операции между рабочим регистром и любым регистром памяти данных.

Входные данные АЛУ в зависимости от кода операции могут находиться:

– В регистре W;

– В регистре команд;

– В памяти данных.

Результат выполнения операции может сохраняться как в рабочий регистр W, так и в любой регистр памяти данных (файловый регистр):

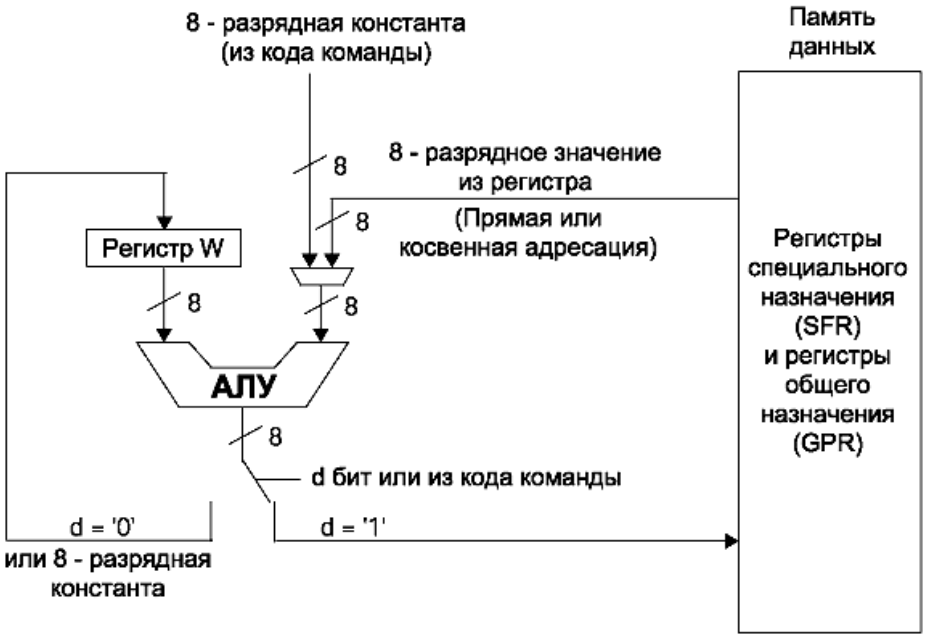


Рисунок 32 – Арифметико-логическое устройство и рабочий регистр

8-разрядное АЛУ может выполнять сложение, вычитание, поразрядный сдвиг и логические операции. Арифметические операции выполняются по принципу дополнения до двух, если не указано иначе. В командах с двумя операндами: первый операнд находится в рабочем регистре W, а второй операнд расположен в регистре памяти данных или константа. В командах с одним операндом: операндом является регистр W или регистр памяти данных.

Регистр W – не адресуемый 8-разрядный рабочий регистр, который используется в операциях АЛУ. В зависимости от типа команды и результат команды АЛУ может воздействовать на следующие флаги состояния в регистре STATUS: перенос (C), полуперенос (DC), флаг нулевого результата (Z). Биты C и DC работают как биты заема и десятичного заема при выполнении команд вычитания.

В регистре STATUS содержатся флаги состояния АЛУ. Флаги причины сброса микроконтроллера и биты управления банками памяти данных.

Регистр STATUS (Рис. 33) может быть адресован любой командой, как и любой другой регистр памяти данных. Если обращение к регистру STATUS выполняется командой, которая воздействует на флаги Z, DC и C, то изменение этих трех битов командой заблокировано. Эти биты сбрасываются или устанавливаются согласно логике ядра микроконтроллера. Команды изменения регистра STATUS также не воздействуют на биты -TO и -PD. Поэтому результат выполнения команды с регистром STATUS может отличаться от ожидаемого. Например, команда CLRF STATUS сбросит три старших бита и установит бит Z.

При изменении битов регистра STATUS рекомендуется использовать команды, не влияющие на флаги АЛУ (SWAPF, MOVWF, BCF и BSF).

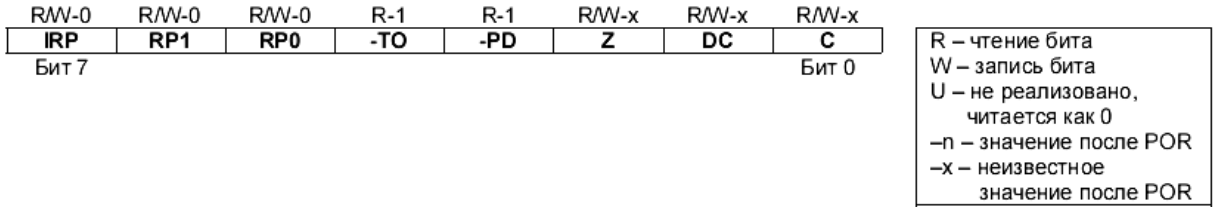


Рисунок 33 – Регистр STATUS

Назначение битов регистра STATUS:

IRP – Бит выбора банка при косвенной адресации (0 – банк РОН 0,1; 1 – банк РОН 2,3);

RP0, RP1 – Биты выбора банка РОН при прямой адресации;

-TO – Флаг переполнения сторожевого таймера;

-PD – Флаг включения питания (0 – после команды SLEEP);

Z – Флаг нулевого результата операции;

DC – Флаг десятичного переноса/заема в младшей тетраде;

C – Флаг переноса/заема при выполнении операции.

5.3 Слово конфигурации микроконтроллера

Биты конфигурации позволяют настроить некоторые режимы работы микроконтроллера в соответствии с требованиями конкретного приложения. При включении питания состояние этих битов определяет режим работы микроконтроллера. Биты конфигурации расположены по адресу 2007h в памяти программ. Программа пользователя не может изменять и читать состояние битов конфигурации (эта операция возможна только в режиме программирования микроконтроллера).

Биты конфигурации могут быть запрограммированы (читаются как «0») или оставлены без изменения (читаются как «1»), чтобы выбрать режим работы микроконтроллера.

Расположение битов конфигурации микроконтроллера представлено на рис. 34:

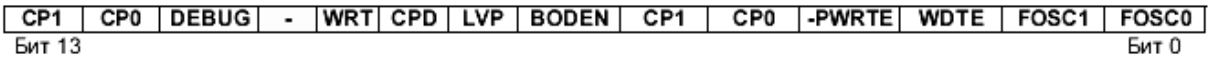


Рисунок 34 – Слово конфигурации

Биты данного слова выполняют следующие функции:

CP1, CP0 – Биты защиты памяти программ:

11 – Защита памяти программ выключена;

10 – Защищена память программ с адресами 1F00h-1FFFh;

01 – Защищена память программ с адресами 1000h-1FFFh;

00 – Защищена память программ с адресами 0000h-1FFFh.

DEBUG – Бит включения внутрисхемной отладки (ICD):

1 – Отладка отключена;

0 – Отладка включена (линии RB6, RB7 используются отладчиком).

WRT – Бит разрешения записи во FLASH память программ:

1 – Запись в FLASH память программ разрешена через регистр управления EECON;

1 – Запись в FLASH память программ запрещена через регистр управления EECON;

CPD – Бит защиты от записи в EEPROM данных:

0 – Защита записи в EEPROM память данных включена.

LVP – Бит разрешения низковольтного программирования:

1 – Низковольтное программирование включено (вывод RB3/PGM используется для программирования).

BODEN – Бит разрешения сброса по снижению напряжения питания:

1 – Разрешен сброс BOR.

-PWRTE – Бит разрешения работы таймера включения питания:

0 – PWRTE включен;

WDTE – Бит разрешения работы сторожевого таймера:

1 – Работа сторожевого таймера разрешена.

FOSC1, FOSC0 – Биты выбора режима работы тактового генератора:

11 – RC-генератор;

10 – HS-генератор (высокочастотный резонатор 4<f<20 МГц);

01 – XT-генератор (обычный резонатор 0,2<f<4 МГц);

00 – LP-генератор (низкочастотный резонатор f<200 кГц).

В макроассемблере MPASM предоставляется возможность определить биты конфигурации в исходном тексте программы с помощью директивы CONFIG. Использование директивы CONFIG гарантирует запись битов конфигурации при программировании микроконтроллера, что уменьшает риск запрограммировать неправильно слово конфигурации.

Пример использования директивы CONFIG:

```
      LIST  p = p16F877             ; Указываем тип процессора
      INCLUDE <P16F877.INC>         ; Подключаем определения регистров
; Настройка битов конфигурации
      _CONFIG _XT_OSC & _PWRTE_ON & _BQDEN_OFF & _CP_OFF & _WDT_ON
org 0x00                            ; Начало памяти программ
RESET_ADDR                          ; Первая команда после сброса
            ...                     ; Наша программа
end
```

Список стандартных имен, используемых в директиве CONFIG для процессора PIC16F877, указан в таблице 9:

Таблица 9 – Стандартные имена директивы CONFIG

Назначение	**Символ**
Тактовый генератор	_RC_OSC
	_LP_OSC
	_XT_OSC
	_HS_OSC
Сторожей таймер WDT	_WDT_ON
	_WDT_OFF
Таймер включения питания PWRT	_PWRTE_ON
	_PWRTE_OFF
Сброс по снижению напряжения питания	_BODEN_ON
	_BODEN_OFF
Защита кода программы	_CP_ALL
	_CP_ON
	_CP_75
	_CP_50
	_CP_OFF
Защита EEPROM памяти данных	_DP_ON
	_DP_OFF

По адресу 2000h-2003h расположена память, предназначенная для хранения контрольной суммы памяти программ и другой служебной информации (ID). Они доступны для чтения и записи только в режиме программирования микроконтроллера. Используются только младшие 4 бита в каждой ячейке.

5.4 Организация памяти программ

Микроконтроллеры среднего семейства имеют 13-разрядный счетчик команд, способный адресовать 8 Кх14 слов памяти программ, и 14-разрядную шину данных памяти программ. Все команды микроконтроллера состоят из 14-разрядного слова.

Вся память программ разделена на 4 страницы по 2 Кслов каждая (0000h-07FFh. 0800h-0FFFh. 1000h-17FFh. 1800h-1FFFh). На рис. 36 показана карта памяти программ и 8-уровневый аппаратный стек.

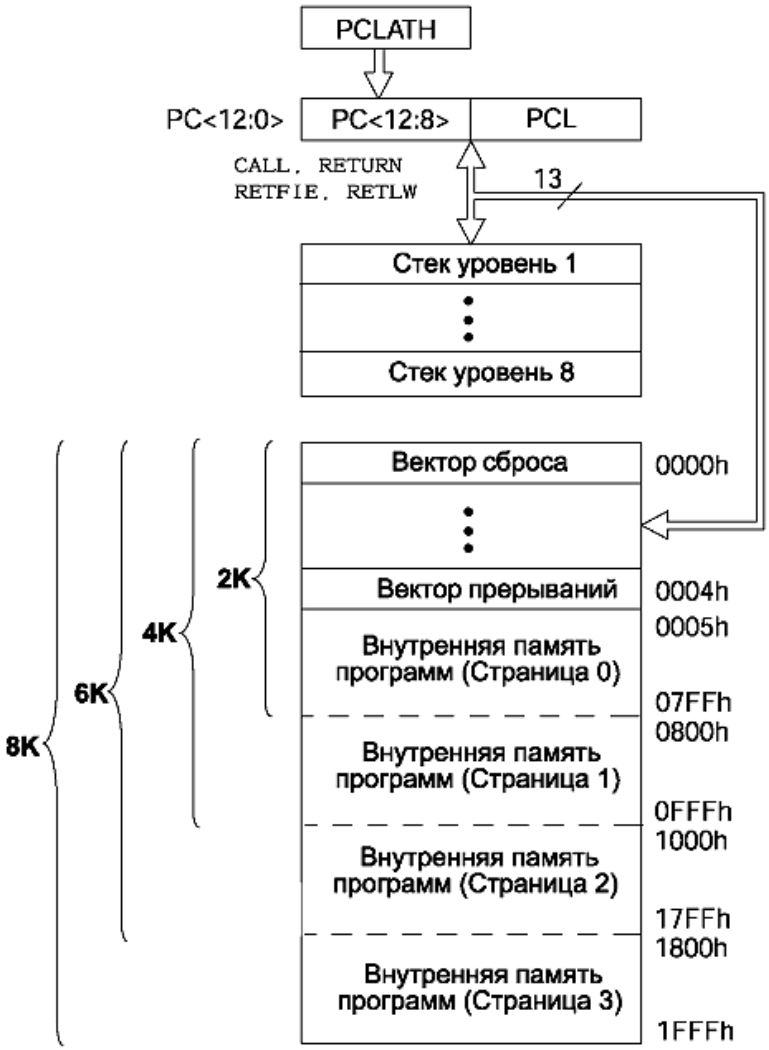


Рисунок 36 – Карта памяти программ и стек

Для перехода между страницами памяти программ необходимо изменить старшие биты регистра счетчика команд PC, записью в регистр специального назначения PCLATH (старший байт счетчика команд). Изменив значение регистра PCLATH, и выполнив команду ветвления, счетчик команд PC пересечет границу страницы памяти программ без дополнительного вмешательства пользователя.

В любом микроконтроллере PICmicro сброс приведет к очистке счетчика команд (PC), устанавливая адрес 0h. Адрес 0000h называется

«адрес вектора сброса», т. к. будет выполнен переход по этому адресу при сбросе микроконтроллера. Вместе со счетчиком команд (PC) очищается регистр PCLATH. устанавливая рабочую страницу памяти программ 0.

Когда возникает разрешенное прерывание, в счетчик команд PC записывается адрес 0004h, называемый «адрес вектора прерываний», при этом значение регистра PCLATH не изменяется.

Если в подпрограмме обработки прерываний требуется выполнять команды ветвления, то необходимо предварительно записать в регистр PCLATH значение, определяющее нужную страницу памяти программ. Прежде чем регистр PCLATH будет изменен, его значение должно быть сохранено в другом регистре памяти данных, а затем восстановлено перед выходом из подпрограммы обработки прерываний.

Счетчик команд PC.

13-разрядный регистр счетчика команд PC указывает адрес выбираемой команды для выполнения. Младший байт счетчика программ PCL доступен для чтения и записи. Старший байт PCH, содержащий биты счетчика команд PC <12:8>, не доступен для чтения и записи. Все операции с регистром PCH происходят через дополнительный регистр PCLATH.

Вычисляемый переход.

Вычисляемый переход может быть выполнен командой приращения к регистру PCL (например, ADDWF PCL). При выполнении вычисляемого перехода следует заботиться о том, чтобы значение PCL не пересекло границу блока памяти (каждый блок 256 байт).

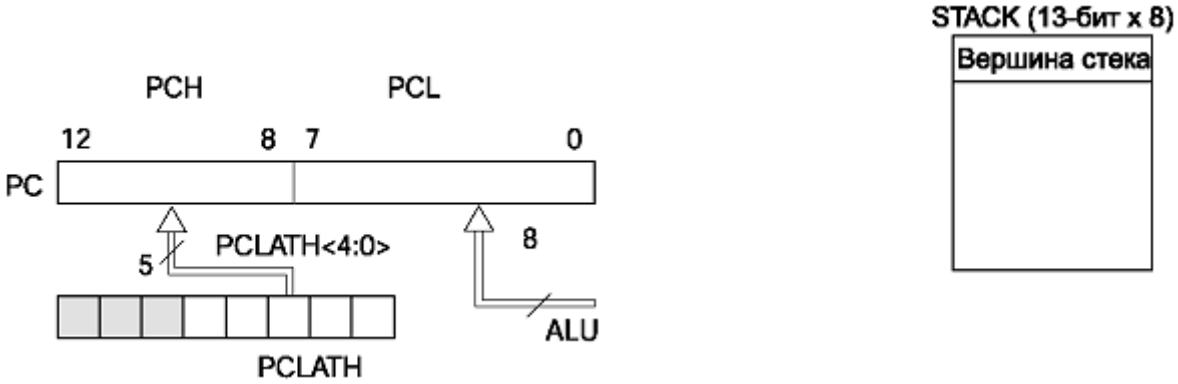
При записи значения в регистр PCL, автоматически происходит перезапись 5 младших бит из регистра PCLATH<4:0> в регистр PCH.

Аппаратный стек.

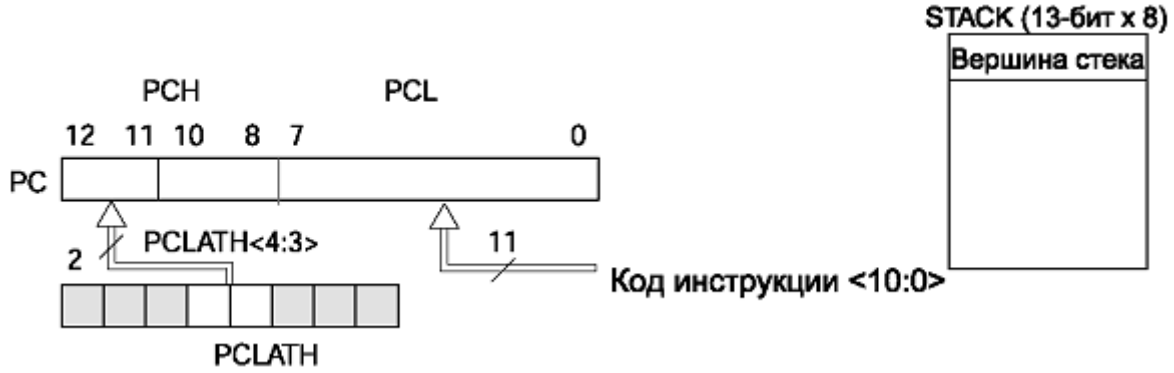
Стек поддерживает до 8 уровней вложенности подпрограмм пользователя, включая обработку прерываний. В стеке сохраняется адрес возврата в основную программу.

В микроконтроллерах среднего семейства PICmicro реализован 8-уровневый 13-разрядный аппаратный стек. Стек не имеет отображения на память программ и память данных, нельзя записать или прочитать данные из стека. Значение счетчика команд заносится в вершину стека при выполнении инструкций перехода на подпрограмму (CALL) или обработку прерываний. Чтение из стека и запись в счетчик команд PC происходит при выполнении инструкций возврата из подпрограммы или обработки прерываний (RETURN, RETLW, RETFIE), при этом значение регистра PCLATH не изменяется.

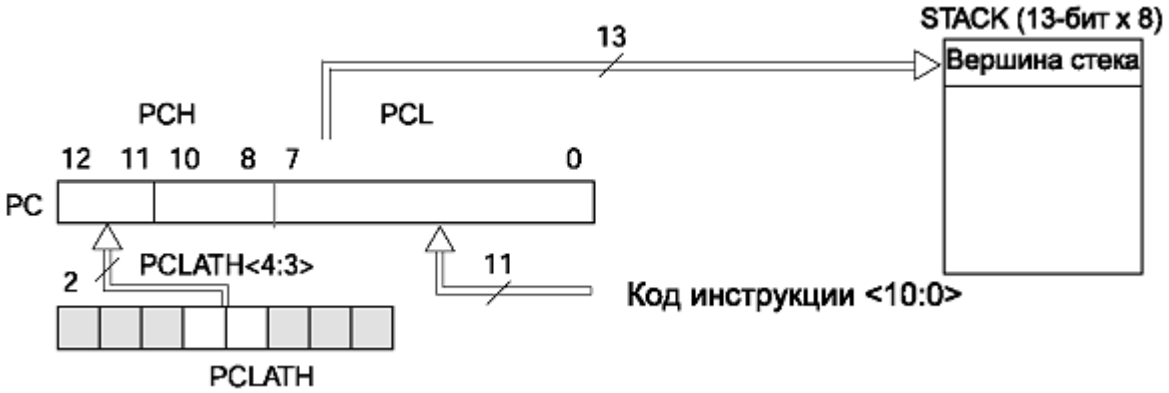
После 8 записей в стек, девятая запись запишется на место первой, а десятая запись заменит вторую и так далее, что нарушает последовательность хранимых данных. Программист должен сам следить за вложенностью подпрограмм и учитывать, что в любой момент времени, возможно, будет обрабатываться прерывание, которому необходима 1 запись в стеке.



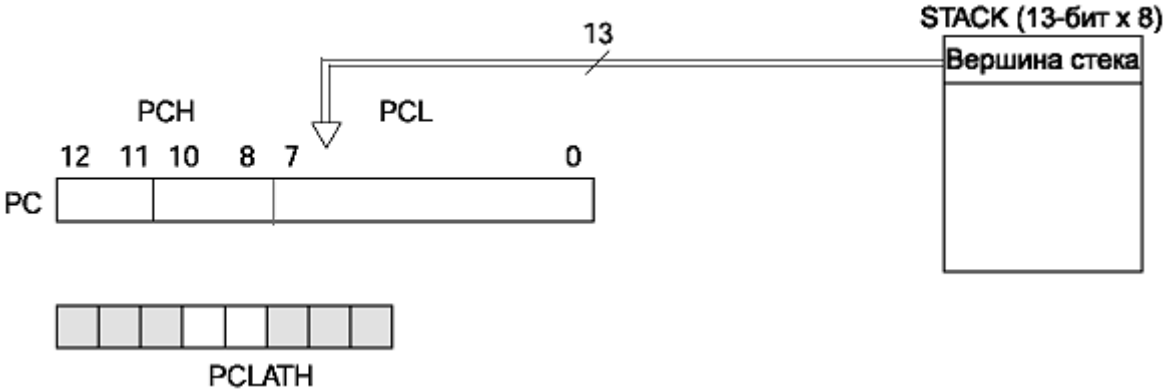
а) Непосредственная запись в регистр PCL (PLATH<4:0> →PCH)



б) Изменение значения PC при выполнении инструкции GOTO (PLATH<4:3> →PCH).



в) Изменение значения PC при выполнении инструкции CALL (PLATH<4:3> →PCH). Старое значение PC сохраняется в стеке.



г) Возвращение из подпрограммы (RETURN, RETFIE или RETLW). Значение PC возвращается со стека.

Рисунок 37 – Возможные варианты изменения состояния PC

Команды переходов (CALL, GOTO) в микроконтроллерах среднего семейства PICmicro имеют 11-разрядное поле для указания адреса, что позволяет непосредственно адресовать 2Кслов памяти программ. Микроконтроллер PIC16F877 имеет память программ 8Кслов. Для адресации верхних страниц памяти программ используются 2 бита в регистре PCLATH<4:3>. Перед выполнением команды перехода (CALL или GOTO) необходимо запрограммировать биты регистра PCLATH<4:3> для адресации требуемой страницы (Рис. 37 б, в).

При выполнении инструкций возврата из подпрограммы 13-разрядное значение для счетчика программ PC берется с вершины стека, поэтому манипуляция битами регистра PCLATH<3:4> не требуется (Рис. 37 г).

Пример переключения с 0 в 1 страницу памяти программ для выполнения подпрограммы в 1й странице:

```
ORG 0x500
      BSF PCLATH,3            ; Выбор страницы 1 (800h-FFFh)
      CALL SUBI_PI            ; Переход на страницу 1 (800h-FFFh)
            ...
;
ORG 0x900
SUBI_PI                       ; Страница 1 (800h-FFFh)
           ...
      RETURN                  ; Возврат на страницу 0 (000h-7FFh)
```

5.5 Организация памяти данных

Память данных разделяется на регистры двух типов:

– Регистры специального назначения (SFR), управляющие работой микроконтроллера;

– Регистры общего назначения (GPR), предназначены для хранения данных программы.

Память данных разделена на банки, содержащие регистры общего и специального назначения. Регистры общего назначения размещаются в разных банках памяти данных для того, чтобы была возможность организовать более 96 байт ОЗУ. Регистры специального назначения предназначены для управления периферийными модулями и функциями микроконтроллера. Управление банками памяти выполняется битами в регистре STATUS<7:5>. На рис. 38 представлена карта памяти данных.

Чтобы передать данные из одного регистра в другой, необходимо использовать дополнительный регистр W. Эта операция выполняется двумя командами за два машинных цикла микроконтроллера.

Обращение ко всем регистрам памяти данных может быть выполнено прямой или косвенной адресацией. **При прямой адресации** для указания банка памяти данных необходимо использовать биты PR1-PR0 регистра STATUS. **В случае косвенной адресации** адрес регистра сохраняется в FSR (Рис. 39), а в бите IRP регистра STATUS указывается, к какой паре

банков памяти данных выполняют обращение (0/1 или 2/3). Для выполнения косвенной адресации необходимо обратиться к регистру INDF. Обращение к регистру INDF фактически вызовет действие с регистром, адрес которого указан в FSR. Косвенное чтение регистра INDF (FSR=0) даст результат значения ячейки 00h. Косвенная запись в регистр INDF не вызовет никаких действий (вызывает воздействия на флаги АЛУ в регистре STATUS).

Адрес		Адрес		Адрес		Адрес	
Регистр косвенной адресации	0x000	Регистр косвенной адресации	0x080	Регистр косвенной адресации	0x100	Регистр косвенной адресации	0x180
TMR0	0x001	OPTION_REG	0x081	TMR0	0x101	OPTION_REG	0x181
PCL	0x002	PCL	0x082	PCL	0x102	PCL	0x182
STATUS	0x003	STATUS	0x083	STATUS	0x103	STATUS	0x183
FSR	0x004	FSR	0x084	FSR	0x104	FSR	0x184
PORTA	0x005	TRISA	0x085		0x105		0x185
PORTB	0x006	TRISB	0x086	PORTB	0x106	TRISB	0x186
PORTC	0x007	TRISC	0x087		0x107		0x187
PORTD	0x008	TRISD	0x088		0x108		0x188
PORTE	0x009	TRISE	0x089		0x109		0x189
PCLATH	0x00A	PCLATH	0x08A	PCLATH	0x10A	PCLATH	0x18A
INTCON	0x00B	INTCON	0x08B	INTCON	0x10B	INTCON	0x18B
PIR1	0x00C	PIE1	0x08C	EEDATA	0x10C	EECON1	0x18C
PIR2	0x00D	PIE2	0x08D	EEADR	0x10D	EECON2	0x18D
TMR1L	0x00E	PCON	0x08E	EEDATH	0x10E	Резерв	0x18E
TMR1H	0x00F		0x08F	EEADRH	0x10F	Резерв	0x18F
T1CON	0x010		0x090		0x110		0x190
TMR2	0x011	SSPCON2	0x091		0x111		0x191
T2CON	0x012	PR2	0x092		0x112		0x192
SSPBUF	0x013	SSPADD	0x093		0x113		0x193
SSPCON	0x014	SSPSTAT	0x094		0x114		0x194
CCPR1L	0x015		0x095		0x115		0x195
CCPR1H	0x016		0x096	Регистры общего назначения 16 байт	0x116	Регистры общего назначения 16 байт	0x196
CCP1CON	0x017		0x097		0x117		0x197
RCSTA	0x018	TXSTA	0x098		0x118		0x198
TXREG	0x019	SPBREG	0x099		0x119		0x199
RCREG	0x01A		0x09A		0x11A		0x19A
CCPR2L	0x01B		0x09B		0x11B		0x19B
CCPR2H	0x01C		0x09C		0x11C		0x19C
CCP2CON	0x01D		0x09D		0x11D		0x19D
ADRESH	0x01E	ADRESL	0x09E		0x11E		0x19E
ADCON0	0x01F	ADCON1	0x09F		0x11F		0x19F
Регистры общего назначения	0x020	Регистры общего назначения 80 байт	0x0A0	Регистры общего назначения 80 байт	0x120	Регистры общего назначения 80 байт	0x1A0
			0x0EF		0x16F		0x1EF
		Доступ к 0x070-0x07h	0x0F0	Доступ к 0x070-0x07h	0x170	Доступ к 0x070-0x07h	0x1F0
	0x07F		0x0FF		0x17F		0x1FF
Банк 0		Банк 1		Банк 2		Банк 3	

Рисунок 38 – Карта памяти микроконтроллера PIC16F877

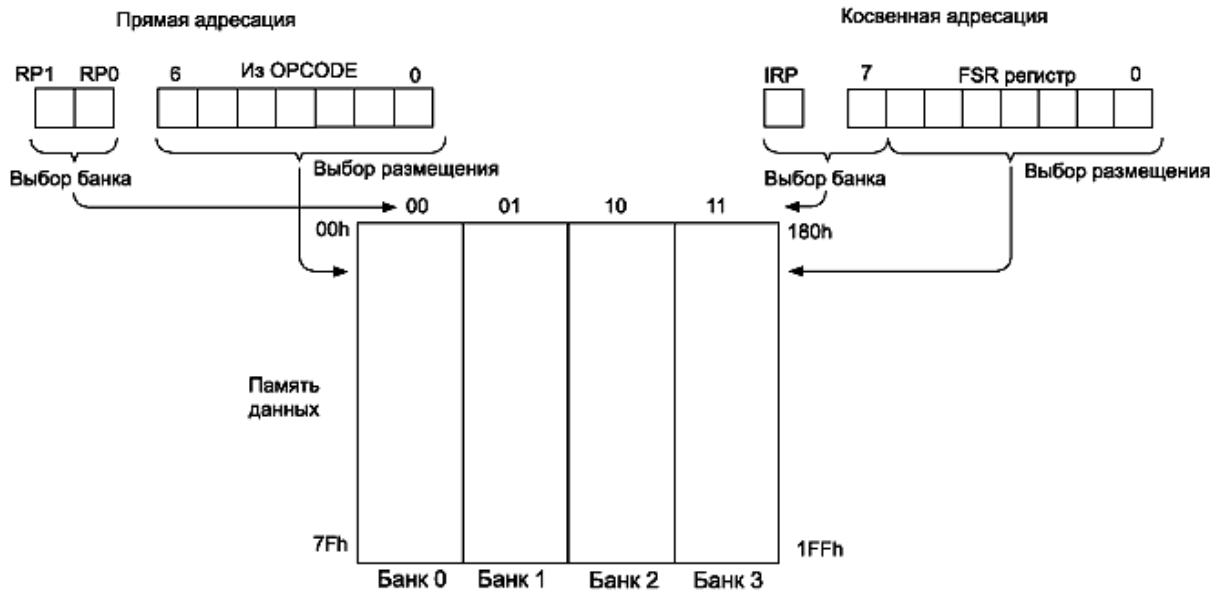


Рисунок 39 – Механизмы прямой и косвенной адресации

Пример использования косвенной адресации для отчистки блока памяти с адресами 20h-2Fh:

```
      BCF STATUS, IRP          ; Установить банк 0,1
      MOVLW 0X20               ; Указать первый регистр в ОЗУ
      MOVWF FSR
NEXT:
      CLRF INDF                ; Очистить регистр
      INCF FSR, F              ; Увеличить адрес
      BTFSS FSR, 4             ; Завершить?
      GOTO NEXT                ; Нет, продолжить очистку

CONTINUE:
                               ;Да
```

Пример переключения между памяти данных для прямой адресации:

```
      CLRF STATUS              ; Очистка регистра STATUS (Банк 0)
;
      BSF STATUS,RP0           ; Банк 1
      ...
      BCF STATUS,RP0           ; Банк 0
      ...
      MOVLW 0x60               ; Установить RP0 и RP1 в STATUS регистре
      XORWF STATUS,F           ; (Банк 3)
      ...
      BCF STATUS,RP0           ; Банк 2
      ...
      BCF STATUS,RP1           ; Банк 0
```

Регистры общего назначения (GPR).

Регистры общего назначения размещаются в разных банках памяти данных. Эти регистры не инициализируются при сбросе по включению питания и имеют неизвестное значение, а при всех остальных сбросах микроконтроллера не изменяют своего значения.

Обращение к регистрам общего назначения может быть выполнено прямой или косвенной адресацией (через регистры FSR и INDF). В микроконтроллере существуют регистры общего назначения, адресуемые к одной и той же ячейке ОЗУ, независимо от текущего банка памяти данных.

При адресации к старшей области памяти данных (последние 16 ячеек) в любом банке происходит «зеркальное» обращение только к регистрам с адресами 70h-7Fh.

Регистры специального назначения (SFR).

Регистры специального назначения используются для управления ядром и периферийными модулями микроконтроллера. Эти регистры реализованы как статическое ОЗУ. Описание регистров SFR, управляющих периферийными модулями, смотрите в соответствующем разделе.

Регистры специального назначения размещены в различных банках памяти данных, а некоторые из регистров отображаются во всех банках.

Переключение рабочего банка памяти выполняется настройкой битов RP1-RP0 регистра STATUS. При сбросе по включению питания и других видах сброса микроконтроллера в некоторые регистры специального назначения записывается определенное значение. Существуют регистры SFR, которые содержат неизвестное значение при сбросе по включению питания, а при других видах сброса не изменяются.

Обращение к регистрам специального назначения может быть выполнено прямой или косвенной адресацией.

5.6 Система прерываний микроконтроллера

Микроконтроллер PIC16F877 имеет 14 источников прерываний. Регистр INTCON содержит флаги отдельных прерываний, биты разрешения этих прерываний и бит глобального разрешения прерываний. Структурная схема логики прерываний показана на рис. 40.

Если бит GIE (INTCON<7>) установлен в «1», разрешены все немаскированные прерывания. Если GIE=0, то все прерывания запрещены. Каждое прерывание в отдельности может быть разрешено или запрещено установкой/сбросом соответствующего бита в регистрах INTCON (Рис. 41), PIE1 (Рис. 42) и PIE2 (Рис. 43). При сбросе микроконтроллера бит GIE сбрасывается в «0».

При возвращении из подпрограммы обработки прерывания, по команде RETFIE, бит GIE аппаратно устанавливается в «1», разрешая все немаскированные прерывания.

В регистре INTCON находятся флаги следующих прерываний: внешнего сигнала INT, изменения уровня сигнала на входах RB7-RB4, переполнения TMR0.

В регистрах PIR1 (Рис. 44), PIR2 (Рис. 45) содержатся флаги прерываний периферийных модулей микроконтроллера, а в регистрах PIE1, PIE2 соответствующие биты разрешения прерываний. В регистре INTCON находится бит разрешения прерываний от периферийных модулей.

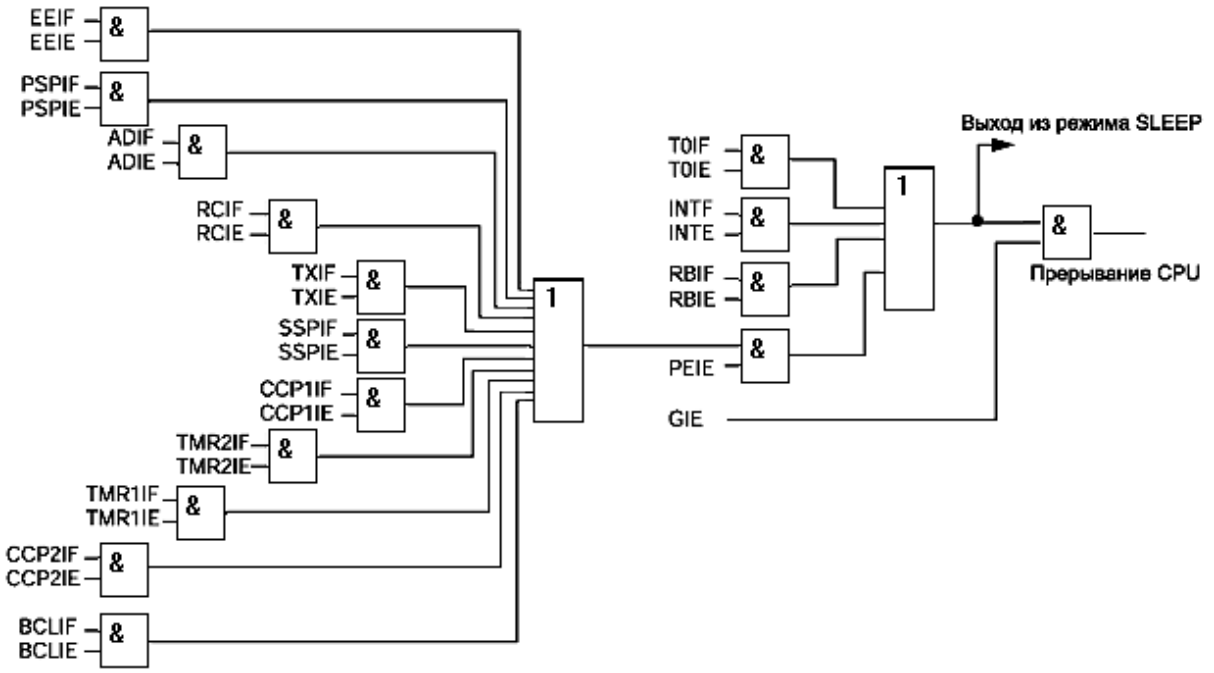


Рисунок 40 – Структурная схема системы прерываний

При переходе на подпрограмму обработки прерываний, бит GIE аппаратно сбрасывается в «0», запрещая прерывания, адрес возврата из подпрограммы обработки прерываний помешается в стек, а в счетчик команд PC загружается вектор прерывания 0004h. Источник прерываний может быть определен проверкой флагов прерываний, которые должны быть сброшены программно перед разрешением прерываний, чтобы избежать повторного вызова.

Флаги прерываний устанавливаются независимо от состояния соответствующих битов маски и бита GIE.

Внешнее прерывание с входа RB0/INT происходит по переднему фронту сигнала, если бит INTEDG (OPTION_REG<6>) установлен в «1»; по заднему фронту сигнала, если бит INTEDG сброшен в «0». Когда активный фронт сигнала появляется на входе RB0/INT бит INTF (INTCON<1>) устанавливается в «1». Прерывание может быть запрещено сбросом бита INTE (INTCON<4>) в «0». Флаг прерывания INTF должен быть сброшен программно в подпрограмме обработки прерываний. Прерывание INT может вывести микроконтроллер из режима SLEEP, если бит INTE=1 до перехода в режим SLEEP. Состояние бита GIE определяет, переходить на подпрограмму обработки прерываний после выхода из режима SLEEP.

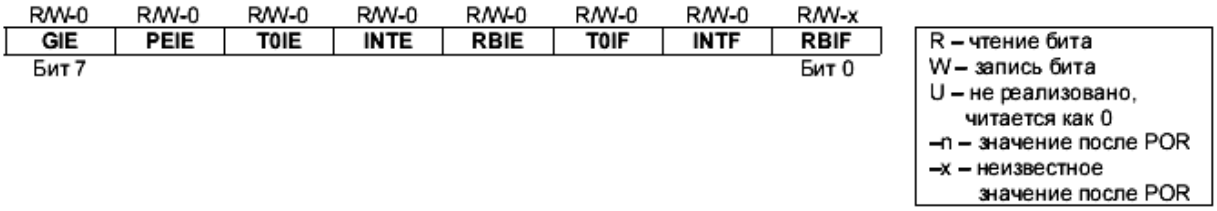


Рисунок 41 – Регистр управления прерываниями INTCON

Назначение битов регистра INTCON:

GIE – Бит общего разрешения прерываний;

PEIE – Бит разрешения прерываний от периферийных модулей;

T0IE – Бит разрешения прерываний по переполнению TMR0;

INTE – Бит разрешения прерываний по внешнему входу INT;

RBIE – Бит разрешения прерываний по изменению уровня на входах RB4-RB7. Обычно используется для обслуживания клавиатурной матрицы.

T0IF – Флаг прерывания по переполнению таймера TMR0 (сбрасывается программно);

INTF – Флаг прерывания по внешнему входу INT (сбрасывается программно);

RBIF – Флаг прерывания по изменению уровня сигнала на входах RB4-RB7 (сбрасывается программно).

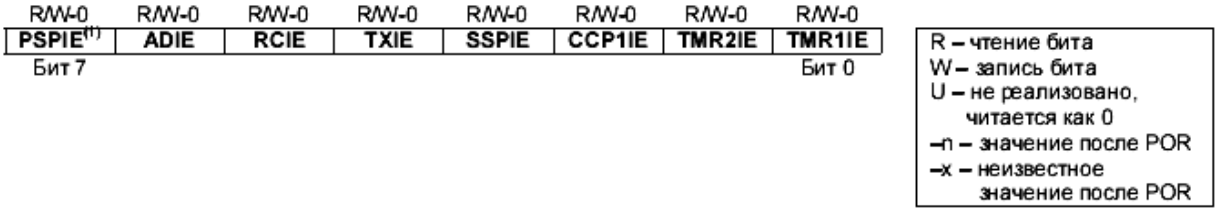


Рисунок 42 – Регистр управления прерываниями PIE1

Назначение битов регистра PIE1:

PSPIE – Разрешение прерываний записи/чтения ведомого параллельного порта;

ADIE – Разрешение прерываний по окончанию преобразования АЦП;

RCIE – Разрешение прерываний от приемника УСАПП;

TXIE – Разрешение прерываний от передатчика УСАПП;

SSPIE – Разрешение прерываний от модуля синхронного последовательного порта;

CCP1IE – Разрешение прерываний от модуля CCP1;

TMR2IE – Разрешение прерываний по переполнению TMR2;

TMR1IE – Разрешение прерываний по переполнению TMR1;

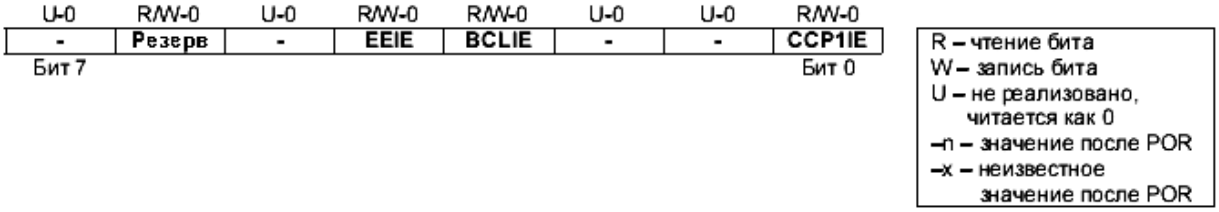


Рисунок 43 – Регистр управления прерываниями PIE2

Назначение битов регистра PIE2:

EEIE – Разрешение прерываний по окончанию записи в EEPROM данных;

BCLIE – Разрешение прерываний по возникновению коллизий на шине;

CCP2IE – Разрешение прерываний от модуля CCP2.

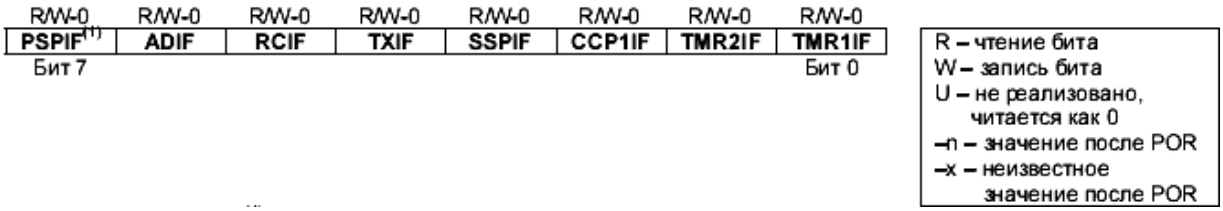


Рисунок 44 – Регистр флагов периферийных модулей PIR1

Назначение битов регистра PIR1:

PSPIF – Флаг прерывания от ведомого параллельного порта после операции чтения/записи;

ADIF – Флаг прерывания от АЦП после завершения преобразования;

RCIF – Флаг прерывания от приемника УСАПП по заполнению буфера приема;

TXIF – Флаг прерывания от передатчика УСАПП по опустошению буфера передачи;

SSPIF – Флаг прерывания от модуля MSSP (сбрасывается программно);

CCP1IF – Флаг прерывания от модуля CCP1:

а) В режиме захвата выполнен захват значения TMR1 (сбрасывается программно);

б) В режиме сравнения значение TMR1 достигло величины, записанной в регистры CCPR1H:CCPR1L (сбрасывается программно);

в) В режиме ШИМ не используется.

TMR2IF – Флаг прерывания по переполнению таймера TMR2 (сбрасывается программно);

TMR1IF – Флаг прерывания по переполнению таймера TMR1 (сбрасывается программно).

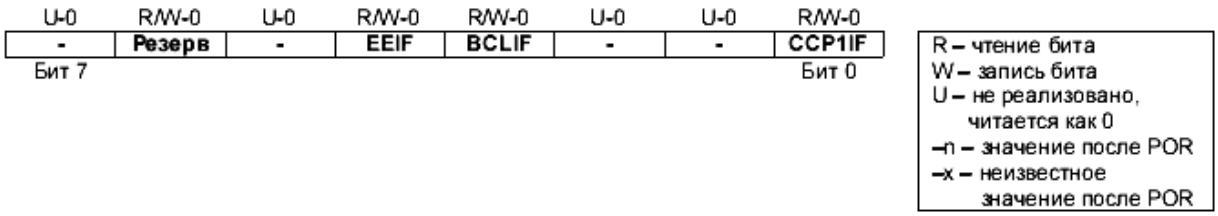


Рисунок 45 – Регистр флагов периферийных модулей PIR2

Назначение битов регистра PIR2:

EEIF – Флаг прерывания по окончанию записи в EEPROM данных (сбрасывается программно);

BCLIF – Флаг прерывания по обнаружению коллизии в режиме ведущего на шине I^2C (сбрасывается программно);

CCP2IF – Флаг прерывания от модуля CCP2:

а) В режиме захвата выполнен захват значения TMR1 (сбрасывается программно);

б) В режиме сравнения значение TMR1 достигло величины, записанной в регистры CCPR2H:CCPR2L (сбрасывается программно);

в) В режиме ШИМ не используется.

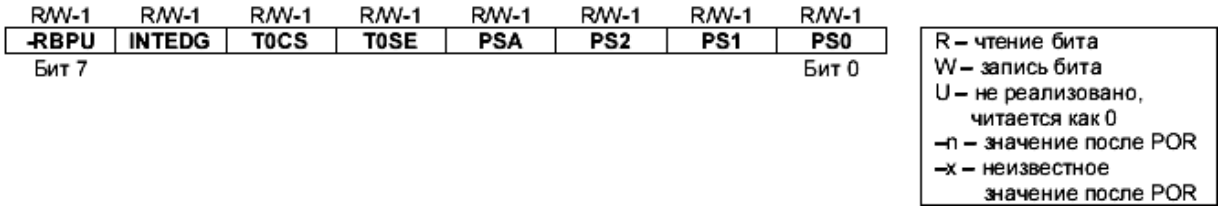


Рисунок 46 – Регистр настроек OPTION_REG

Назначение битов регистра OPTION_REG:

-RBPU – Бит включения внутренних подтягивающих резисторов на входах PORTB. 0 – подтягивающие резисторы включены;

INTEDG – Выбор активного сигнала на внешнем входе прерывания INT:

0 – Прерывание по заднему фронту сигнала;

1 – Прерывание по переднему фронту сигнала.

T0CS – Выбор тактового сигнала для таймера TMR0:

0 – Внутренний сигнал CLKOUT;

1 – Внешний сигнал с вывода RA4/T0CKI.

T0SE – Выбор фронта приращения TMR0 при внешнем тактовом сигнале:

0 – Приращение по переднему фронту;

1 – Приращение по заднему фронту.

PSA – Выбор включения предделителя:

0 – Предделитель включен перед TMR0;

1 – Предделитель включен перед сторожевым таймером.

PSA2, PSA1, PSA0 – выбор коэффициента деления предделителя:

Значение:	Для TMR0:	Для WDT:
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Если предделитель подключен к WDT, то его коэффициент деления для модуля TMR0 равен 1:1.

При переходе на подпрограмму обработки прерываний в стеке аппаратно сохраняется только адрес возврата. Как правило, дополнительно

необходимо сохранять ключевые регистры (например W, STATUS), что выполняется программно.

Операция сохранения значения регистров обычно обозначается PUSH, а восстановление значения регистров обозначается POP. Обратите внимание, что PUSH, POP не являются мнемоникой команд, а лишь обозначают действие, которое может быть выполнено последовательностью команд. Следует отличать эти операции от команд микропроцессора I8051, т. к. микроконтроллеры PICmicro имеют аппаратный стек не доступный из программы. Для упрощения текста программы можно эти сегменты кода программы представить в виде макросов MPASM.

Последовательность операций при сохранении контекста программы:

1) Сохранить регистр W независимо от текущего банка памяти;

2) Сохранить регистр STATUS в банке 0;

3) Выполнить подпрограмму обработки прерываний;

4) Восстановить регистр STATUS и текущий банк памяти данных;

5) Восстановить регистр W.

Если необходимо сохранить и другие регистры, то сохранение нужно выполнять после сохранения регистра STATUS (шаг 2), а восстановление перед восстановлением STATUS (шаг 4).

Можно применять для сохранения контекста программы область памяти данных, которая имеет «зеркальное» отображение во всех банках регистров – адреса 70h-7Fh.

Пример написания макроса сохранения регистров программы:

```
PUSH_MACRO MACRO        ; Макрос сохранения регистров
MOVWF W_TEMP            ; Копировать W во временный регистр независимо
                        ; от текущего банка
SWAPF STATUS,W          ; Обменять полубайты в регистре STATUS
                        ; и записать в W
MOVWF STATUS_TEMP       ; Сохранить STATUS во временном регистре банка 0
ENDM                    ; Конец макроса
;
POP_MACRO MACRO         ; Макрос восстановления регистров
SWAPF STATUS_TEMP,W     ; Обменять полубайты оригинального значения STATUS
                        ; и записать в W (восстановить текущий банк)
MOVWF STATUS            ; Восстановить значение STATUS из регистра W
SWAPF W_TEMP,F          ; Обменять полубайты в регистре W_TEMP и сохранить
                        ; результат в W_TEMP
SWAPF WJTEMP,W          ; Обменять полубайты в регистре W_TEMP и
                        ; восстановить оригинальное значение W без
                        ; воздействия на STATUS
ENDM                    ; Конец макроса
```

Пример инициализации системы прерываний:

```
PIE1_MASK1 EQU B'01101010'    ; Значение для регистра маски прерываний
                              ;
      CLRF STATUS             ; Банк 0
      CLRF INTCON             ; Выключить прерывания и сбросить флаги
      CLRF PIR1               ; Сбросить все флаги
      BSF STATUS,RP0          ; Банк 1
      MOVLW PIE1_MASK1        ; Записать маску прерываний в регистр PIE1
      MOVWF PIE1              ;
      BCF STATUS,RP0          ; Банк 0
      BSF INTCON,GIE          ; Включить прерывания
```

В примере показана инициализация прерываний, где PIE1_MASK – значение, записываемое в регистр маски периферийных прерываний.

Макрокоманды должны быть определены прежде, чем они будут использоваться. Для простоты отладки текста программы макрокоманды рекомендуется помещать в отдельные файлы, включаемые в исходный файл программы, до применения макрокоманды. Рекомендуется включать файлы с макрокомандами в начале исходного файла:

```
      LIST p=pl6F877          ; Список директив
      #INCLUDE <P16F877.INC>  ; Дополнительный файл к микроконтроллеру
      #INCLUDE <MY_STD.MAC>   ; Подключить файл стандартных макрокоманд
      #INCLUDE <APP.MAC>      ; подключить файл специальных макрокоманд
; Определение битов конфигурации для этого приложения
      _CONFIG _XT_OSC & _PWRTE_ON & _BODEN_OFF & _CP_OFF & _WDT_ON
      org 0x00                ; Начало памяти программ
RESET_ADDR                    ; Первая выполняемая инструкция после сброса
end
```

Как указывалось выше, система прерываний микроконтроллера PIC16F877 имеет 14 источников прерываний и всего лишь один вектор входа в подпрограммы прерываний. Такая организация системы прерываний накладывает некоторые особенности при написании программ. Определение источника прерывания назначается программе, которая при обслуживании прерываний должна определять какой узел микроконтроллера нуждается в обработке, путем проверки установленных флагов прерываний.

В нижеприведенном примере представлена типовая структура проверки возникшего прерывания. В этом примере используются макрокоманды для сохранения значения регистров перед выполнением кода обработки прерываний:

```
Org ISR_ADDR                  ;
      PUSH_MACRO              ; Макрокоманда сохранения регистров.
                              ; или другой код
      CLRF STATUS             ; Банк 0
      BTFSC PIR1,TMR1IF       ; Прерывание от TMR1?
      GOTO T1_INT             ; Да
      BTFSC PIR1,ADIF         ; Нет, прерывание от АЦП?
      GOTO AD_INT             ; Да, от АЦП
            ...               ; Нет, проверка других источников
            ...               ; прерываний
      BTFSC INTCON,RBIF       ; Нет, прерывание по изменению сигнала
                              ; на RB7:RB6?
      GOTO PORTB_INT          ; Да.
INT_ERROR_LP1                 ; Нет, процедура восстановления при ошибке
      GOTO INT_ERROR_LPl      ; Здесь должна располагаться процедура
                              ; обработки возникновения неожидаемого
                              ; прерывания
T1_INT                        ; Обработка прерываний от TMR1
            ...
      BCF PIR1,TMR1IF         ; Сброс флага прерывания от TMR1
      GOTO END_ISR            ; Завершение обработки прерываний
AD_INT                        ; Обработка прерываний от АЦП
            ...
      BCF PIR1,ADIF           ; Сброс флага прерывания от АЦП
      GOTO END_ISR            ; Завершение обработки прерываний
PORTB_INT                     ; Обработка прерываний по изменению
            ...               ; сигнала на RB7:RB6
END_ISR
```

```
POP_MACRO               ; Макрокоманда восстановления значения
                        ; регистров или другой код
RETFIE                  ; Возвращение из обработки прерываний,
                        ; разрешение прерываний
```

5.7 Порты ввода/вывода

Универсальные порты ввода/вывода могут рассматриваться как самые простые периферийные модули. Они позволяют микроконтроллерам PIC контролировать работу и управлять другими устройствами. Для большинства каналов портов ввода/вывода регистры TRIS управляют направлением данных на выводе. Бит TRIS<x> управляет направлением данных на канале PORT<x>. Если бит TRIS установлен в «1», то соответствующий канал порта ввода/вывода работает как вход, а если бит TRIS сброшен в «0», то канал ввода/вывода работает как выход. Простой способ запомнить направление канала ввода/вывода и состояние битов регистров TRIS: 1 – напоминает «In» (ввод); «0» – напоминает «Out» (выход).

Регистр PORT – защелка данных, выводимых на порт ввода/вывода. При чтении регистра PORT возвращается состояние выводов порта. Это означает, что необходима некоторая осторожность при выполнении команд со структурой «чтение-модификация-запись» для изменения логического уровня на выходах порта.

Чтение регистра PORT возвращает состояние на выводах порта, а запись выполняется в выходную защелку. Обратите внимание на операции «чтение-модификация-запись» (например BSF и BCF). Сначала происходит чтение состояния выводов порта, изменение полученного значения, а затем выполняется запись в выходную защелку порта.

Расположение выводов микроконтроллера PIC16F877 можно увидеть на рис. 47. Мультиплицирование каналов ввода/вывода с функциями периферийных модулей вносит свои особенности при программировании и настройке микроконтроллера.

Когда периферийный модуль подключен к выводу порта, функциональные возможности канала порта ввода/вывода могут измениться, в соответствии с требованиями периферийного модуля. Например, модуль АЦП, которому настраивают соответствующие каналы как аналоговые входы. Таким образом, выводы портов могут быть мультиплицированы с аналоговыми входами и входом Vref. Для каждого вывода необходимо определить режим его работы (аналоговый вход или цифровой канал ввода/вывода) настройкой управляющих битов в регистре ADCON1 (регистр управления АЦП). Когда вывод работает как аналоговый вход, то чтение состояния этого вывода будет давать результат «0». Следует заметить, что при сбросе микроконтроллера выводы, мультиплицированные с АЦП, настраиваются на аналоговый ввод.

Регистры TRIS управляет направлением каналов ввода/вывода, даже когда он работает в режиме аналогового входа. Пользователь должен гарантировать, что соответствующий бит TRIS установлен в «1», если вывод используется как аналоговый вход.

При включении некоторых периферийных модулей отменяется действие битов TRIS. Поэтому следует избегать команд «чтение-модификация-запись» с регистрами TRIS (например, BSF, BCF, XORWF и т. д.).

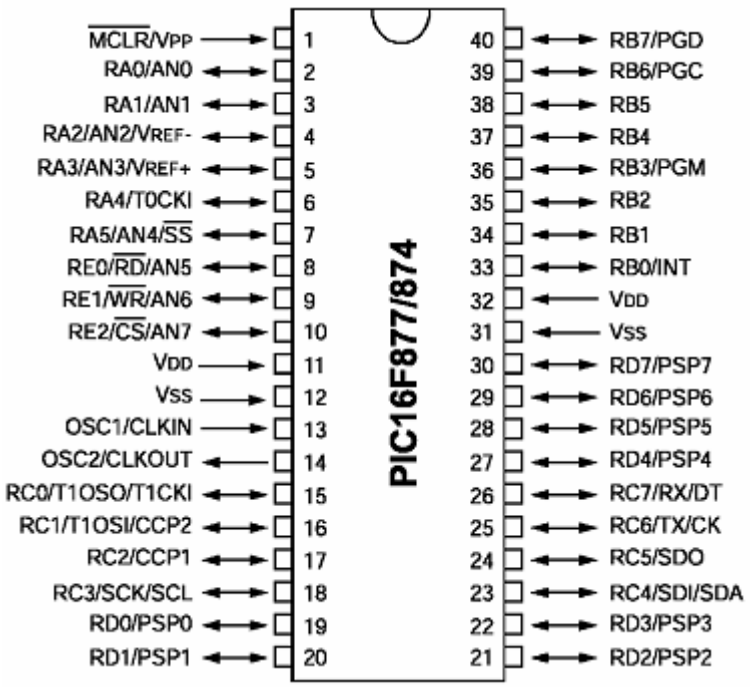


Рисунок 47 – Расположение выводов микроконтроллера PIC16F877

Регистры PORTA и TRISA.

Линия RA4 имеет триггер Шмидта на входе и открытый сток на выходе. Все остальные каналы PORTA имеют ТТЛ буфер на входе и полнофункциональные выходные КМОП буферы. Все выводы имеют биты управления направления данных в регистре TRISA, с помощью которых можно настроить выводы как входы или выходы.

Запись «1» в TRISA переводит соответствующий выходной буфер в Z-состояние. Запись «0» в регистр TRISA определяет соответствующий канал как выход, содержимое защелки PORTA передается на вывод микроконтроллера. Пример настройки каналов порта PORTA:

```
BCF STATUS,RP0    ; Выбрать банк 0
CLRF PORTA        ; Инициализация защелок PORTA
BSF STATUS,RP0    ; Выбрать банк 1
MOVLW 0xCF        ; Значение для инициализации
                  ; направления каналов PORTA
MOVWF TRISA       ; Настроить RA<3:0> как входы.
                  ; настроить RA<5:4> как выходы
                  ; Биты TRISA<7:6> всегда читаются как '0'.
```

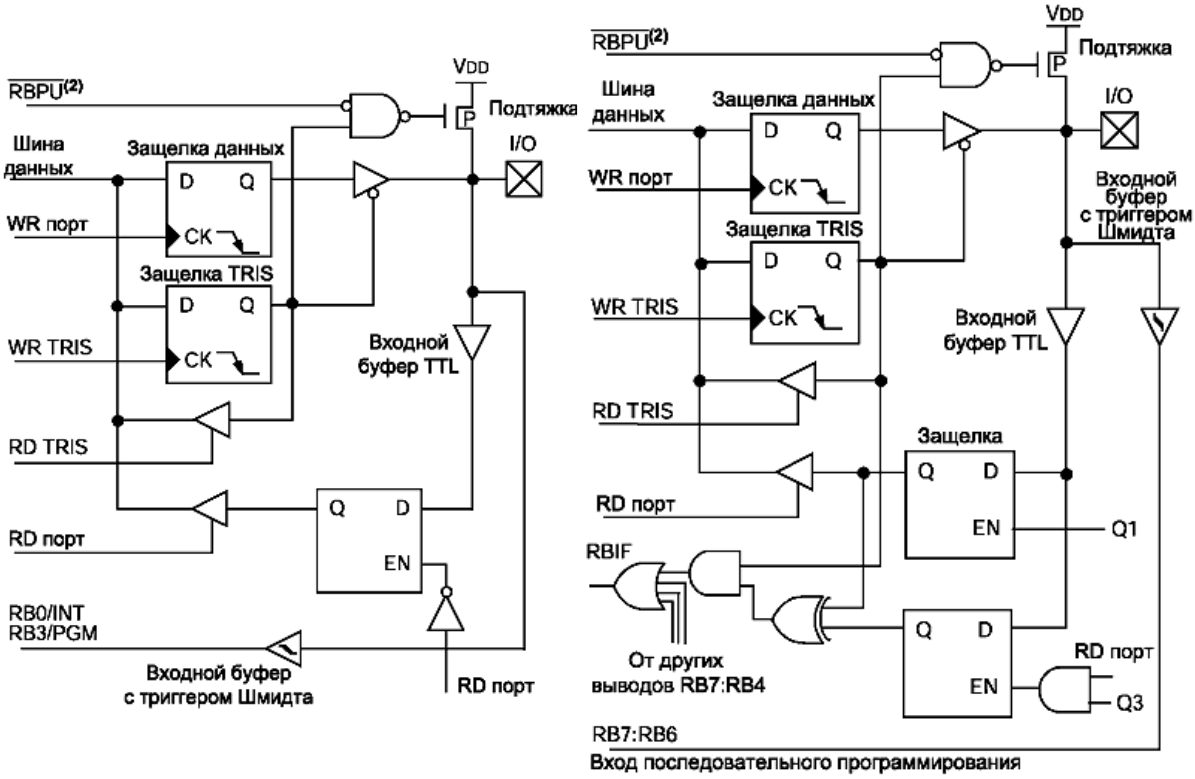
Регистры PORTB и TRISB.

PORTB – 8-разрядный двунаправленный порт ввода/вывода. Биты регистра TRISB определяют направление каналов порта. Установка битва в «1» регистра TRISB переводит выходной буфер в Z-состояние. Запись «0» в регистр TRISB настраивает соответствующий канал как выход, содержимое защелки PORTB передается на вывод микроконтроллера (если выходная защелка подключена к выводу микроконтроллера).

Пример настройки порта PORTB:

```
BCF STATUS,RP0     ; ВЫБРАТЬ БАНК 0
CLRF PORTB         ; ИНИЦИАЛИЗАЦИЯ ЗАЩЕЛОК PORTB
BSF STATUS,RP0     ; ВЫБРАТЬ БАНК 1
MOVLW 0xCF         ; ЗНАЧЕНИЕ ДЛЯ ИНИЦИАЛИЗАЦИИ
                   ; НАПРАВЛЕНИЯ КАНАЛОВ PORTB
MOVWF TRISB        ; НАСТРОИТЬ RB<3:0> КАК ВХОДЫ.
                   ; RB<5:4> КАК ВЫХОДЫ. RB<7:6> КАК ВХОДЫ
```

К каждому выводу PORTB подключен внутренний подтягивающий резистор. Бит -RBPU (регистр OPTION_REG<7>) определяет, подключены (-RBPU=0) или нет (-RBPU=1) подтягивающие резисторы. Подтягивающие резисторы автоматически отключаются, когда каналы порта настраиваются на выход и после сброса по включению питания POR.



а) *б)*

Рисунок 47 – Структурная схема выводов RB0-RB3 (а) и RB4-RB7 (б)

Четыре канала PORTB RB7-RB4 (Рис. 47) настроенные на вход, могут генерировать прерывания по изменению логического уровня сигнала на входе. Если один из каналов RB7-RB4 настроен на выход, то он не может быть источником прерываний. Сигнал на выводах RB7-RB4 сравнивается со значением, сохраненным при последнем чтении PORTB. В

случае несовпадения одного из значений устанавливается флаг RBIF (INTCON<0>) и, если разрешено, генерируется прерывание.

Это прерывание может вывести микроконтроллер из режима SLEEP. В подпрограмме обработки прерываний необходимо сделать следующие действия:

– Выполнить чтение или запись в PORTB, исключив несоответствие;

– Сбросить флаг RBIF в «0».

Несоответствие сохраненного значения с сигналом на входе PORTB всегда устанавливает бит RBIF в «1». Чтение из PORTB прервет условие несоответствия и позволит сбросить флаг RBIF в «0».

Прерывания по изменению сигнала на входах PORTB и программа переключения конфигурации этих каналов позволяет реализовать простой интерфейс обслуживания клавиатуры с выходом из режима SLEEP по нажатию клавиш.

Прерывания по изменению сигнала на входах рекомендуется использовать для определения нажатия клавиш, когда PORTB полностью задействован для реализации клавиатуры. Не рекомендуется опрашивать PORTB при использовании прерываний по изменению входного сигнала.

Вывод RB0/INT может служить для ввода внешнего сигнала прерываний, настраиваемого битом INTEDG регистра OPTION_REG<6>.

Регистры PORTC и TRISC.

PORTC – 8-разрядный двунаправленный порт ввода/вывода. Биты регистра TRISC определяют направление каналов порта. Установка бита в «1» регистра TRISC переводит выходной буфер в Z-состояние. Запись «0» в регистр TRISC настраивает соответствующий канал как выход, содержимое защелки PORTC передается на вывод микроконтроллера (если выходная защелка подключена к выводу микроконтроллера).

Выводы PORTC мультиплицированы с несколькими периферийными модулями. На каналах PORTC присутствует входной буфер с триггером Шмидта.

При использовании периферийных модулей необходимо соответствующим образом настраивать биты регистра TRISC для каждого вывода PORTC. Некоторые периферийные модули отменяют действие битов TRISC, принудительно настраивая вывод на вход или выход. В связи с чем, не рекомендуется использовать команды «чтение-модификация-запись» с регистром TRISC.

Настройка каналов PORTC производится аналогичными способами, указанными выше.

Регистры PORTD и TRISD.

PORTD – 8-разрядный двунаправленный порт ввода/вывода. Биты регистра TRISD определяют направление каналов порта.

Каналы PORTD могут работать как 8-разрядный микропроцессорный порт (ведомый параллельный порт), если бит PSPMODE (TRISE<4>) установлен в «1». В режиме ведомого параллельного порта к входам подключены буферы с уровнями TTL.

Регистры PORTE и TRISE.

PORTE имеет три вывода (RE0/-RD/AN5, RE1/-WR/AN6, RE2/-CS/AN7), индивидуально настраиваемые на вход или выход. Выводы PORTE имеют входной буфер Шмидта.

Каналы PORTE станут управляющими выводами ведомого параллельного порта, когда бит PSPMODE (TRISE<4>) установлен в «1». В этом режиме биты TRISE<2:0> должны быть установлены в «1». В регистре ADCON1 необходимо также настроить выводы PORTE как цифровые каналы ввода/вывода. В режиме ведомого параллельного порта к выводам PORTE подключены входные буферы TTL.

Выводы PORTE мультиплицированы с аналоговыми входами. Когда каналы PORTE настроены как аналоговые входы, биты регистра TRISE не влияют на направление данных PORTE, чтение будет давать результат «0».

После сброса по включению питания выводы настраиваются как аналоговые входы, а чтение дает результат «0».

Особенности выполнения команд «чтение-модификация-запись».

Все операции записи в порт выполняются по принципу «чтение-модификация-запись». Например, команды BCF и BSF считывают значение в регистр ЦПУ, выполняют битовую операцию и записывают результат обратно в регистр. Требуется некоторая осторожность при применении подобных команд к регистрам портов ввода/вывода. Например, команда BSF PORTB,5 считывает все восемь битов PORTB в ЦПУ, изменяет состояние бита 5 и записывает результат в выходные защелки PORTB. Если другой канал PORTB (например, RB0) настроен на вход, то сигнал на выводе будет считан в ЦПУ и записан в защелку данных, поверх предыдущего значения. Пока RB0 настроен как вход, никаких проблем не возникает. Но, если RB0 будет позже настроен как выход, значение в защелке данных может отличаться от требуемого значения.

В нижеуказанном примере показан эффект последовательного выполнения команд «чтение-модификация-запись» с регистром порта ввода/вывода. Начальные установки порта: PORTB<7:4> входы; PORTB<3:0> выходы. Выводы RB7-RB6 имеют внешние подтягивающие резисторы и не подключены к другим цепям в схеме:

```
                    Защелка PORTB     Выводы PORTB
BCF STATUS,RP0
BCF PORTB, 7        ;01PP PPPP        11pp PPPP
BCF PORTB,6         ;10PP PPPP        11PP PPPP
BSF STATUS,RP0      ;
BCF TRISB,7         ;10PP PPPP        11PP PPPP
BCF TRISB,6         ;10PP PPPP        10PP PPPP
```

Обратите внимание. Возможно, пользователь ожидал, что после выполнения программы на выходах PORTB будет значение 00PP PPPP. Однако 2-я команда BCF установила в «1» RB7.

Запись в порт ввода/вывода фактически происходит в конце машинного цикла, а чтение данных выполняется в начале цикла. Поэтому требуется некоторая осторожность при записи в порт ввода/вывода, если перед записью выполняется чтение состояния этого порта.

Последовательность команд должна быть такой, чтобы установилось напряжение на выводе порта прежде, чем будет выполнена команда записи в порт, сопровождаемая чтением состояния выводов (иначе вместо нового значения может быть считано предыдущее). Если возможна описанная ситуация, разделите команды записи инструкциями NOP или любыми другими командами, которые не обращаются к порту ввода/вывода.

При инициализации портов ввода/вывода рекомендуется сначала записать стартовое значение в выходную защелку порта (регистр PORT), а затем настроить направление каналов порта (регистр TRIS). Эта последовательность устраняет возможность ложного уровня на выходе порта, т. к. при включении питания в выходных защелках порта содержится случайное значение.

5.8 Модуль таймера TMR0

TMR0 – таймер/счетчик, имеет следующие особенности:
– 8-разрядный таймер/счетчик;
– Возможность чтения и записи текущего значения счетчика;
– 8-разрядный программируемый предделитель;
– Внутренний или внешний источник тактового сигнала;
– Выбор активного фронта внешнего тактового сигнала;
– Прерывания при переполнении (переход значения от FFh к 00h).

Блок схема модуля TMR0 и общего с WDT предделителя, показана на рис. 48:

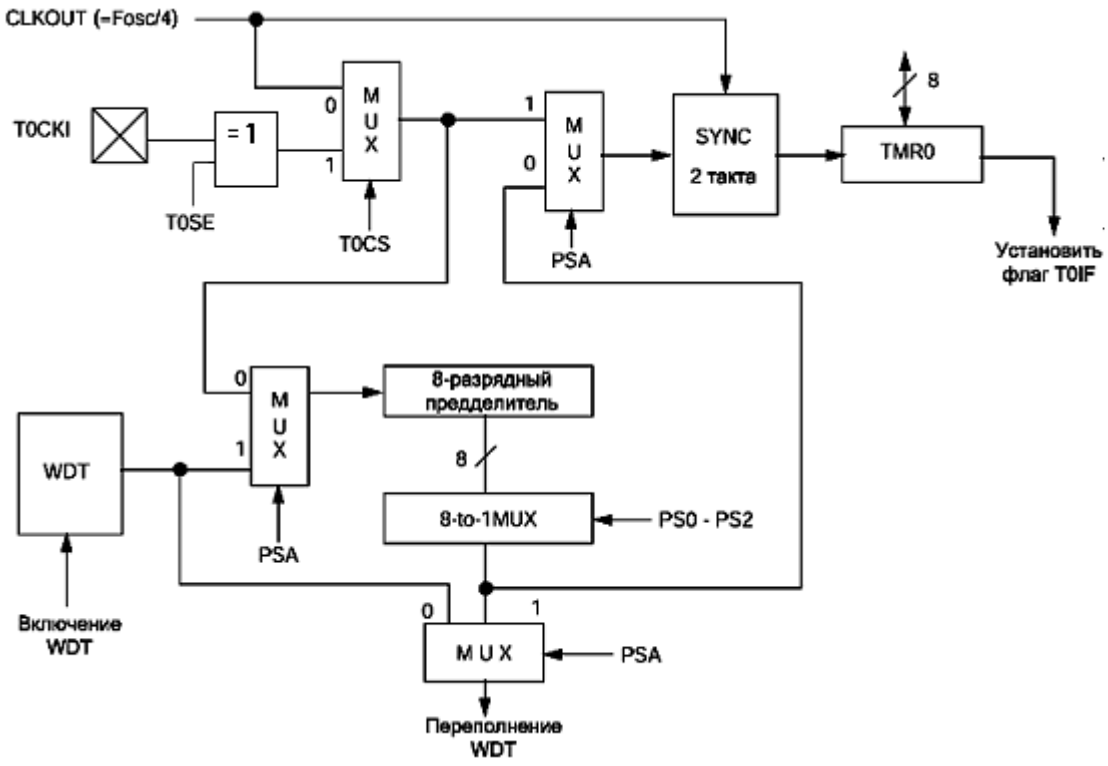


Рисунок 48 – Блок-схема модуля таймера TMR0

Когда бит T0CS сброшен в «0» (OPTION_REG<5>, Рис. 46), TMR0 работает от внутреннего тактового сигнала. Если бит T0CS установлен в «1» (OPTION_REG<5>), TMR0 работает от внешнего источника тактового сигнала с входа RA4/T0CKI. Активный фронт внешнего тактового сигнала выбирается битом T0SE в регистре OPTION_REG<4> (T0SE=0 – активным является передний фронт сигнала). Предделитель может быть включен перед WDT или TMR0, в зависимости от состояния бита PSA (OPTION_REG<3>). Нельзя прочитать или записать новое значение в предделитель.

Любая запись в регистр TMR0 вызовет запрещение приращения таймера TMR0 в течение двух следующих машинных циклов ($2T_{CY}$). Если предделитель включен перед TMR0, то запись в регистр TMR0 вызовет немедленное изменение TMR0 и сброс предделителя. Приращение TMR0 и предделителя запрещено в течение 2-х машинных циклов ($2T_{CY}$), после записи в TMR0. Например, если коэффициент предделителя равен 2, то после операции записи в регистр TMR0 приращение таймера не будет происходить в течение 4 циклов для TMR0. Далее таймер работает в нормальном режиме.

Прерывания от TMR0 возникают при переполнении счетчика, т. е. при переходе его значения от FFh к 00h. При возникновении прерывания устанавливается в «1» бит T0IF (INTCON<2>). Само прерывание может быть разрешено или запрещено установкой/сбросом бита T0IE в регистре INTCON<5>. Флаг прерывания от TMR0 T0IF (INTCON<2>) должен быть сброшен в подпрограмме обработки прерываний. В SLEEP режиме микроконтроллера модуль TMR0 выключен и не может генерировать прерывания.

Переключение предделителя выполняется программным способом, т. е. переключение можно сделать во время выполнения программы.

Для предотвращения случайного сброса микроконтроллера следует выполнять переключение предделителя от TMR0 к WDT как показано в примере ниже, даже если WDT выключен.

В данном примере первая часть изменения регистра OPTION_REG не должна выполняться, если желаемый коэффициент предделителя отличный от 1:1. Если требуется настройка коэффициента предделителя 1:1, то необходимо установить промежуточное значение коэффициента (отличное от 1:1), а затем установить коэффициент предделителя 1:1 в последней части изменения OPTION_REG.

Переключение предделителя от TMR0 к WDT:

```
BSF STATUS,RP0          ;Банк 1
MOVLW Ь'XX0X0XXX'       ; ВЫБРАТЬ ИСТОЧНИК ТАКТОВОГО СИГНАЛА И
MOVWF OPTION_REG        ; КОЭФФИЦИЕНТ ПРЕДДЕЛИТЕЛЯ. ОТЛИЧНЫЙ ОТ 1:1
BCF STATUS,RP0          ; БАНК 0
CLRF TMR0               ; СБРОСИТЬ TMR0 И ПРЕДДЕЛИТЕЛЬ
BSF STATUS,RP0          ; БАНК 1
MOVLW B'XXXX1XXX'       ; ВКЛЮЧИТЬ ПРЕДДЕЛИТЕЛЬ ПЕРЕД WDT.
MOVWF OPTION_REG        ; НО НЕ ВЫБИРАТЬ КОЭФФИЦИЕНТ ДЕЛЕНИЯ
CLRWDT                  ; СБРОСИТЬ WDT И ПРЕДДЕЛИТЕЛЬ
MOVLW B'XXXX1XXX'       ; ВЫБРАТЬ НОВОЕ ЗНАЧЕНИЕ КОЭФФИЦИЕНТА
```

```
      MOVWF OPTION_REG        ; ПРЕДДЕЛИТЕЛЯ
      BCF STATUS,RP0          ; БАНК0
```

Примечания к примеру. Если желаемое значение коэффициента деления отличное от 1:1, то строки 2 и 3 в текст программы не должны включаться. Если требуется настройка коэффициента предделителя 1:1, то необходимо установить промежуточное значение коэффициента (отличное от 1:1) в строках 2 и 3, а затем установить коэффициент предделителя 1:1 в строках 10 и 11.

Пример переключение предделителя от WDT к TMR0:

```
      CLRWDT            ; СБРОСИТЬ WDT И ПРЕДДЕЛИТЕЛЬ
      BSF STATUS,RP0    ;БАНК1
      MOVLW B'XXXX0XXX' ; ВКЛЮЧИТЬ ПРЕДДЕЛИТЕЛЬ ПЕРЕД TMRO И
      MOVWF OPTION_REG  ; ВЫБРАТЬ НОВОЕ ЗНАЧЕНИЕ КОЭФФИЦИЕНТА ДЕЛЕНИЯ
      BCF STATUS,RP0    ; БАНК 0
```

Пример инициализации TMR0 (внутренний источник тактового сигнала):

```
      CLRF TMR0         ; СБРОС TMR0
      CLRF INTCON       ; ВЫКЛЮЧИТЬ ПРЕРЫВАНИЯ И СБРОСИТЬ T0IF
      BSF STATUS,RP0    ; БАНК1
      MOVLW 0XC3        ; ВЫКЛЮЧИТЬ ПОДТЯГИВАЮЩИЕ РЕЗИСТОРЫ НА PORTB.
      MOVWF OPTION_REG  ; ПРЕРЫВАНИЯ ПО ПЕРЕДНЕМУ ФРОНТУ СИГНАЛА НА RB0
; TMR0 ИНКРЕМЕНТИРУЕТСЯ ОТ ВНУТРЕННЕГО ТАКТОВОГО СИГНАЛА
; ПРЕДДЕЛИТЕЛЬ 1:16.
      BCF STATUS,RP0    ; БАНК 0
;**   BSF INTCON,T0IE   ; РАЗРЕШИТЬ ПРЕРЫВАНИЯ ОТ TMR0
;**   BSF INTCON,GIE    ; РАЗРЕШИТЬ ВСЕ ПРЕРЫВАНИЯ

; ЕСЛИ ПРЕРЫВАНИЯ ОТ TMR0 ВЫКЛЮЧЕНЫ, ТО ВЫПОЛНЯЙТЕ ПРОВЕРКУ БИТА
; ПЕРЕПОЛНЕНИЯ.
T0_OVFL_WAIT
      BTFSS INTCON,T0IF
      GOTO T0_OVFL_WAIT
; ПРОИЗОШЛО ПЕРЕПОЛНЕНИЕ TMR0
```

Пример инициализации TMR0 (внешний источник тактового сигнала):

```
      CLRF TMR0         ; СБРОС TMR0
      CLRF INTCON       ; ВЫКЛЮЧИТЬ ПРЕРЫВАНИЯ И СБРОСИТЬ T0IF
      BSF STATUS,RP0    ; БАНК 1
      MOVLW 0X37        ; ВКЛЮЧИТЬ ПОДТЯГИВАЮЩИЕ РЕЗИСТОРЫ НА PORTB.
      MOVWF OPTION_REG  ; ПРЕРЫВАНИЯ ПО ЗАДНЕМУ ФРОНТУ СИГНАЛА НА RB0
; TMR0 ИНКРЕМЕНТИРУЕТСЯ ОТ ВНЕШНЕГО ТАКТОВОГО СИГНАЛА;
; ПРЕДДЕЛИТЕЛЬ 1:256.
      BCF STATUS,RP0    ; БАНК 0
;**   BSF INTCON,T0IE   ; РАЗРЕШИТЬ ПРЕРЫВАНИЯ ОТ TMR0
;**   BSF INTCON, GIE   ; РАЗРЕШИТЬ ВСЕ ПРЕРЫВАНИЯ

; ЕСЛИ ПРЕРЫВАНИЯ ОТ TMR0 ВЫКЛЮЧЕНЫ, ТО ВЫПОЛНЯЙТЕ ПРОВЕРКУ БИТА
; ПЕРЕПОЛНЕНИЯ.
T0_OVFL_WAIT
      BTFSS INTCON, T0IF
      GOTO T0_OVFL_WAIT
; ПРОИЗОШЛО ПЕРЕПОЛНЕНИЕ TMR0
```

5.9 Модуль таймера TMR1

Таймер TMR1 – 16-разрядный таймер/счетчик, состоящий из двух 8-разрядных регистров (TMR1H и TMR1L) доступных для чтения и записи. Счет выполняется в спаренных регистрах (TMR1H-TMR1L), инкрементируя их значение от 0000h до FFFFh, далее считает с 0000h. При переполнении счетчика устанавливается в «1» флаг прерывания TMR1IF в регистре PIR1<0>. Само прерывание можно разрешить или запретить установкой/сбросом бита TMR1IE в регистре P1E1<0>. TMR1 может работать в двух режимах:

– Режим таймера;

– Режим счетчика.

Включение модуля TMR1 осуществляется установкой бита TMR1ON в «1» в регистре T1CON:

Битом TMR1CS (T1CON<1>) выбирается источник тактовых импульсов (Рис. 50). В режиме таймера TMR1 инкрементируется на каждом машинном цикле. Если TMR1 работает с внешним источником тактового сигнала, то приращение происходит по каждому переднему фронту сигнала.

Таймер TMR1 имеет внутренний вход сброса от CPP модуля.

Когда включен генератор тактовых импульсов (T1OSCEN=1), выводы RC1/T1OSI/CCP2 и RC0/T1OSO/T1CKI настроены как входы. Значение битов TRISC<1:0> игнорируется, а чтение данных с этих выводов дает результат '0'.

Управляющие биты TMR1 находятся в регистре T1CON:

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	-	T1CKPS1	T1CKPS0	T1OSCEN	-T1SYNC	TMR1CS	TMR1ON
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

Рисунок 49 – Регистр управления таймером T1CON

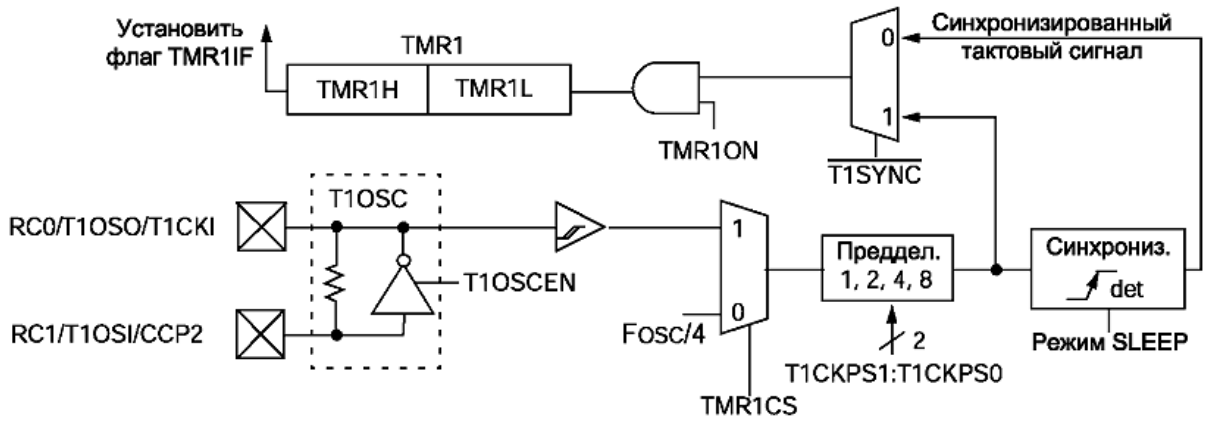


Рисунок 50 – Модуль таймера TMR1

Назначение битов регистра T1CON:

TMR1ON – Бит включения модуля таймера;

TMR1CS – Выбор источника тактового сигнала для приращения таймера (внутренний/внешний);

-T1SYNC – Синхронизация внешнего тактового сигнала;

T1OSCEN – Включение тактового генератора TMR1;

T1CKPS1, T1CKPS0 – Выбор коэффициента деления предделителя TMR1:

Значение:	Коэффициент:
00	1:1
01	1:2
10	1:4
11	1:8

Чтение TMR1H или TMR1L. во время счета в асинхронном режиме, гарантирует получение текущего значения счетчика (реализовано аппаратно). Однако пользователь должен иметь в виду, что чтение 16-разрядного значения выполняется по одному байту. Это накладывает некоторые ограничения, т. к. таймер может переполниться между чтениями байт.

Запись в TMR1 рекомендуется выполнять после остановки таймера. Запись в регистры TMR1 во время приращения таймера может привести к непредсказуемому значению регистра.

Чтение 16-разрядного значения требуется некоторой осторожности, т. к. требуется два цикла чтения для получения всех 16 разрядов.

В нижеуказанном примере представлена рекомендованная последовательность операций чтения 16-разрядного значения TMR1 в асинхронном режиме с решением проблем переполнения. В данном примере таймер не останавливается:

```
; ВЫКЛЮЧИТЬ ВСЕ ПРЕРЫВАНИЯ
      MOVF TMR1H,W      ; ЧТЕНИЕ СТАРШЕГО БАЙТА
      MOVWF TMPH        ;
      MOVF TMR1L,W      ; ЧТЕНИЕ МЛАДШЕГО БАЙТА
      MOVWF TMPL        ;
      MOVF TMR1H,W      ; ЧТЕНИЕ СТАРШЕГО БАЙТА
      SUBWF TMPH,W      ; СРАВНЕНИЕ С ПРЕДЫДУЩИМ ЧТЕНИЕМ
      BTFSC STATUS,Z    ;
      GOTO CONTINUE     ; 16-РАЗРЯДНОЕ ЗНАЧЕНИЕ ПРОЧИТАНО ПРАВИЛЬНО

; ВОЗМОЖНО, МЕЖДУ ЧТЕНИЯМИ БАЙТОВ ПРОИЗОШЛО
; ПЕРЕПОЛНЕНИЕ ТАЙМЕРА
; ПРОЧИТАТЬ ЗНАЧЕНИЯ ЗАНОВО
      MOVF TMR1H,W      ; ЧТЕНИЕ СТАРШЕГО БАЙТА
      MOVWF TMPH        ;
      MOVF TMR1L,W      ; ЧТЕНИЕ МЛАДШЕГО БАЙТА
      MOVWF TMPL        ;
CONTINUE:
; ВКЛЮЧИТЬ ПРЕРЫВАНИЯ (ЕСЛИ НЕОБХОДИМО)
```

Для записи 16-разрядного значения в регистры TMR1, сначала нужно очистить регистр TMR1L, чтобы в запасе было большое число тактов TMR1 прежде, чем произойдет перенос из младшего регистра TMR1L в

TMR1H. Выполнить запись в TMR1H. а затем записать значение в TMR1L. Эта последовательность действий показана в примере:

```
; ВЫКЛЮЧИТЬ ВСЕ ПРЕРЫВАНИЯ
      CLRF TMR1L        ; ОЧИСТИТЬ МЛАДШИЙ БАЙТ
                        ; ДЛЯ ПРЕДОТВРАЩЕНИЯ ПЕРЕНОСА В TMRH
      MOVLW HI_BYTE     ; ЗНАЧЕНИЕ ДЛЯ TMR1H
      MOVWF TMR1H       ; ЗАПИСАТЬ СТАРШИЙ БАЙТ
      MOVLW L0_BYTE     ; ЗНАЧЕНИЕ ДЛЯ TMR1L
      MOVWF TMRIH       ; ЗАПИСАТЬ МЛАДШИЙ БАЙТ
; ВКЛЮЧИТЬ ПРЕРЫВАНИЯ (ЕСЛИ НЕОБХОДИМО)
CONTINUE:
```

Пример инициализации TMR1 от внутреннего тактового сигнала:

```
      CLRF T1CON        ; ВЫКЛЮЧИТЬ TMR1, ВНУТРЕННИЙ ТАКТОВЫЙ СИГНАЛ.
; ГЕНЕРАТОР TMR1 ВЫКЛЮЧЕН, ПРЕДДЕЛИТЕЛЬ =1:1
      CLRF TMR1H        ; ОЧИСТИТЬ СТАРШИЙ БАЙТ РЕГИСТРА TMR1
      CLRF TMR1L        ; ОЧИСТИТЬ МЛАДШИЙ БАЙТ РЕГИСТРА TMR1
      CLRF INTCON       ; ВЫКЛЮЧИТЬ ПРЕРЫВАНИЯ
      BSF STATUS,RP0    ;БАНК 1
      CLRF PIE1         ; ВЫКЛЮЧИТЬ ПЕРИФЕРИЙНЫЕ ПРЕРЫВАНИЯ
      BCF STATUS,RP0    ; БАНК 0
      CLRF PIR1         ; ОЧИСТИТЬ ФЛАГИ ПЕРИФЕРИЙНЫХ ПРЕРЫВАНИЙ
      MOVLW 0X30        ; ВНУТРЕННИЙ ТАКТОВЫЙ СИГНАЛ С ПРЕДДЕЛИТЕЛЕМ 1:8
      MOVWF T1CON       ; TMR1 И ГЕНЕРАТОР TMR1 ВЫКЛЮЧЕНЫ
      BSF TICON,TMR10N  ; ВКЛЮЧИТЬ TMR1

; ПРЕРЫВАНИЯ ОТ TMR1 ВЫКЛЮЧЕНЫ, ПРОВЕРЯЙТЕ БИТ ПЕРЕПОЛНЕНИЯ

T1_OVFL_WAIT
      BTFSS PIR1,TMR1IF
      GOTO T1_OVFL_WAIT

; ПЕРЕПОЛНЕНИЕ TMR1

      BCF PIR1,TMR1IF
```

Пример инициализации TMR1 от внешнего тактового сигнала:

```
      CLRF T1CON        ; ВЫКЛЮЧИТЬ TMR1, ВНУТРЕННИЙ ТАКТОВЫЙ СИГНАЛ.
; ГЕНЕРАТОР TMR1 ВЫКЛЮЧЕН, ПРЕДДЕЛИТЕЛЬ =1:1
      CLRF TMR1H        ; ОЧИСТИТЬ СТАРШИЙ БАЙТ РЕГИСТРА TMR1
      CLRF TMR1L        ; ОЧИСТИТЬ МЛАДШИЙ БАЙТ РЕГИСТРА TMR1
      CLRF INTCON       ; ВЫКЛЮЧИТЬ ПРЕРЫВАНИЯ
      BSF STATUS,RP0    ;БАНК 1
      CLRF PIE1         ; ВЫКЛЮЧИТЬ ПЕРИФЕРИЙНЫЕ ПРЕРЫВАНИЯ
      BCF STATUS,RP0    ; БАНК 0
      CLRF PIR1         ; ОЧИСТИТЬ ФЛАГИ ПЕРИФЕРИЙНЫХ ПРЕРЫВАНИЙ
      MOVLW 0X3E        ; ВНЕШНИЙ НЕ СИНХРОНИЗИРОВАННЫЙ СИГНАЛ С
                        ; ПРЕДДЕЛИТЕЛЕМ 1:8 И ВНЕШНИМ ГЕНЕРАТОРОМ
      MOVWF T1CON       ; TMR1 ВЫКЛЮЧЕН
      BSF TICON,TMR10N  ; ВКЛЮЧИТЬ TMR1

; ПРЕРЫВАНИЯ ОТ TMR1 ВЫКЛЮЧЕНЫ, ПРОВЕРЯЙТЕ БИТ ПЕРЕПОЛНЕНИЯ

T1_OVFL_WAIT
      BTFSS PIR1,TMR1IF
      GOTO T1_OVFL_WAIT

; ПЕРЕПОЛНЕНИЕ TMR1

      BCF PIR1,TMR1IF
```

5.10 Модуль таймера TMR2

Таймер TMR2 – 8-разрядный таймер с программируемым предделителем и выходным делителем, 8-разрядным регистром периода PR2. Таймер TMR2 может быть опорным таймером для CCP модуля в ШИМ режиме. Регистры TMR2 доступны для записи/чтения и очищаются при любом виде сброса.

Входной тактовый сигнал (Fosc/4) поступает через предделитель с программируемым коэффициентом деления (1:1, 1:4 или 1:16), определяемый битами T2CKPS1, T2CKPS0 (T2CON<1:0>).

Таймер TMR2 считает, инкрементируя от 00h до значения в регистре PR2, затем сбрасывается в 00h на следующем машинном цикле. Регистр PR2 доступен для записи и чтения. После сброса значение регистра PR2 равно FFh (Рис. 51).

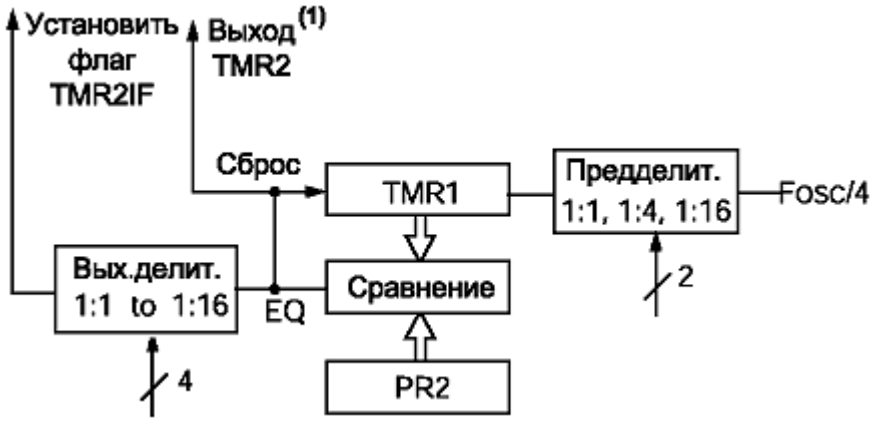


Рисунок 51 – Модуль таймера TMR2

Сигнал переполнения TMR2 проходит через выходной 4-разрядный делитель с программируемым коэффициентом деления (от 1:1 до 1:16 включительно) для установки флага TMR2IF в регистре PIR1<1>.

Для уменьшения энергопотребления таймер TMR2 может быть выключен сбросом бита TMR2ON (T2CON<2>) в «0»:

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

Рисунок 52 – Регистр управления таймером T2CON

Назначение битов регистра:

T2CKPS1, T2CKPS0 – Выбор коэффициента предделителя TMR2:

Значение:	Коэффициент:
00	1:1
01	1:4

1x	1:16

TMR2ON – Бит включения таймера;

TOUTPS3-TOUTPS0 – Биты выбора коэффициента деления выходного делителя TMR2:

Значение:	Коэффициент:
0000	1:1
0001	1:2
…	…
1111	1:16

Пример инициализации таймера TMR2:

```
      CLRF T2C0N         ; Выключить TMR2. предделитель = 1:1,
                         ; выходной делитель =1:1
      CLRF TMR2          ; Очистить регистр TMR2
      CLRF INTCON        ; Выключить прерывания
      BSF STATUS,RP0     ; Банк 1
      CLRF PIE1          ; Выключить периферийные прерывания
      BCF STATUS,RP0     ; Банк 0
      CLRF PIR1          ; Очистить флаги периферийных прерываний
      MOVLW 0X72         ; Предделитель = 1:15, выходной делитель = 1:16,
      MOVWF T2CON        ; TMR2 выключен
      BSF T2CON,TMR20N   ; ВКЛЮЧИТЬ TMR2

; Прерываний OTTMR2 выключены, проверяйте бит переполнения

T2_OVFL_WAIT
      BTFSS PIRI, TMR2IF; Произошло переполнение TMR2?
      GOTO T2_OVFL_WAIT ; Нет, оставаться в цикле

; Переполнение TMR2

      BCF PIR1,TMR2IF
```

6 ТРАНСЛЯТОР АССЕМБЛЕРА МИКРОКОНТРОЛЛЕРОВ PICmicro MPASM

6.1 Правила написания программ

Принципы написания программ на языке ассемблера для микроконтроллеров семейства PICmicro остаются такие же, как и описанные в главах 2 и 3 для микроконтроллеров семейства MCS-51. Существуют только некоторые особенности:

– Необходимо учитывать аппаратные тонкости при программировании микроконтроллеров Microchip, описанные в главе 5 данного издания;

– Микроконтроллер имеет только 35 команд;

– Микроконтроллер имеет только 8 ячеек стека;

– Микроконтроллер имеет прямую, непосредственную и довольно сложный механизм косвенной адресации;

– Синтаксис директив транслятора немного отличается от синтаксиса транслятора ASM51;

– Величины числовых констант записываются следующим способом:

H’9F’ – запись шестнадцатеричного числа;
0x9F – запись шестнадцатеричного числа;
D’10’ – десятичное число;
O’377’ – восьмеричное число;
B’11001101’ – двоичное число;
A’T’ – символ ASCII;
’R’ – символ ASCII.

6.2 Директивы ассемблера MPASM:

Большую часть директив ассемблера ASM51 понимает и транслятор MPASM. Список основных директив транслятора указан в таблице 10. Для более детального описания и получения информации по директивам, обратитесь к руководству пользователя транслятора MPASM.

Таблица 10 – Список основных директив

Директива	**Описание**
BANKISEL	Выбор банка РОН для косвенной адресации
BANKSEL	Выбор банка РОН для прямой адресации
__CONFIG	Установка битов конфигурации
CONSTANT	Определить символьную константу
DA	Сохранение строки в памяти программ
DATA	Сохранение значений или текста в памяти программ

Продолжение таблицы 10

DB	Побайтное сохранение данных в памяти программ
#DEFINE	Определяет замену текста
DT	Определяет таблицу данных
END	Окончание программы
ENDM	Окончание макроса
EQU	Определение константы ассемблера
EXPAND	Включение текста макроса в файл листинга программы
__IDLOCS	Установка значения ID
#INCLUDE	Подключение дополнительного исходного файла
LIST	Список параметров
LOCAL	Объявить локальную переменную макроса
MACRO	Определить макрос
NOEXPAND	Не разворачивать текст макроса
NOLIST	Выключить вывод в файл листинга
ORG	Установить адрес программы
PROCESSOR	Выбор типа микроконтроллера
RADIX	Система счисления по умолчанию
SET	Определение константы
#UNDEFINE	Отменить замену текста

MPASM может использоваться в двух случаях:

– Для генерации абсолютного кода, который может быть загружен непосредственно в микроконтроллер;

– Для генерации объектных файлов, которые связываются с другими компилированными модулями.

Абсолютный код – режим работы программы MPASM по умолчанию.

При компиляции исходного файла в этом режиме, все значения должны быть явно указаны в исходном файле или во включаемых файлах. Если компиляция выполнена без ошибок, то будет создан HEX файл кода программы, который можно использовать для непосредственного программирования микроконтроллера.

Компилятор MPASM, как и ASM-51, так же имеет возможность генерировать объектные модули, которые могут быть связаны друг с другом с использованием линкера MPLINK. Линкер необходим для окончательного формирования исполняемого (абсолютного) кода. Данный метод позволяет многократно использовать отлаженные модули программы. Объектные файлы могут быть сгруппированы в библиотечные файлы с помощью программы MPLIB. Библиотеки могут указываться в качестве параметра во время линковки и, таким образом, в исполняемый код будет включены только необходимые процедуры.

Для набора текста исходной программы может быть использован любой текстовый редактор, как и в трансляторе ASM-51. Таким

редактором, например, как уже было указано ранее, является редактор «Блокнот» ОС Windows.

Типы файлов, связанные с ассемблером MPASM:

*.asm – исходный текст программы на языке ассемблер;

*.lst – листинг программы после трансляции;

*.err – список ошибок, возникших после компиляции;

*.hex – выходной файл кода программы;

*.hxl – файл кода программы отдельно младших байтов кода;

*.hxh – файл кода программы отдельно старших байтов кода;

*.cod – файл для отладчика;

*.o – выходной объектный файл модуля (программы).

Формат файла листинга, генерируемого MPASM следующий: имя файла и версия, дата и время компиляции, номер страницы выводятся в начале каждой страницы.

Первая колонка цифр указывает базовый адрес кода в памяти. Вторая колонка показывает 32-разрядное значение всех символьных переменных созданных директивами SET, EQU, VARIABLE, CONSTANT или CBLOCK. Третья колонка предназначена для машинного кода, выполняемого микроконтроллером. Четвертая колонка содержит номер строки соответствующего исходного файла

Остаток строки зарезервирован для исходного текста, который породил машинный код.

Ошибки, предупреждения и сообщения вставляются между строк исходного кода и относятся к следующей по тексту строке исходного кода.

Таблица символов (SYMBOL TABLE) показывает все символьные переменные, определенные в программе.

Карта использования памяти (MEMORY USAGE MAP) дает представление об использовании памяти в графическом виде. Символ «X» показывает использованный участок, а «-» – отмечает участок памяти не используемый данным объектом. При генерации объектного файла карта памяти не выводится.

7 ПРАКТИКУМ ПО АССЕМБЛЕРУ PICmicro

7.1 Требования к отчетам

К отчетам предъявляются такие же требования, которые указаны в главе 4.1 данного издания.

Для написания программ можно использовать язык ассемблер MPASM Microchip Technology или интегрированную среду разработки и отладки программ MPLab IDE.

Программы решения заданий данной главы необходимо писать в соответствии принципиальной электрической схемы демонстрационно-отладочного стенда PICDEM 2 Plus.

7.2 Практическое задание №1

Тема: Знакомство с ассемблером PICmicro. Программирование цифровых портов ввода/вывода.

Цель: Изучить систему команд микроконтроллера PIC16F877 среднего семейства PICmicro. Получить навыки в программировании портов ввода/вывода.

Таблица 11 – Варианты заданий к самостоятельной работе

Вариант	**Задание**
1	Выполнить бегущий огонь одного светодиода с частотой 1Гц
2	Выполнить бегущий огонь двух светодиодов с частотой 0,5Гц
3	Выполнить бегущую тень одного светодиода с частотой 1Гц
4	Выполнить бегущий огонь 2х светодиодов в шахматном порядке с частотой 0,5Гц
5	Выполнить включение накапливающихся светодиодов с частотой переключения 1,5Гц
6	Выполнить попеременное включение светодиодов D2, D3 и D4, D5 с частотой 1Гц
7	Выполнить включение светодиодов в последовательности D3, D5, D4, D2 с частотой 1,5Гц
8	Выполнить включение светодиодов в последовательности D4, D2, D5, D3 с частотой 0,5Гц
9	Выполнить бегущий огонь одного светодиода с тушением светодиодов между переключениями частотой с 0,5Гц
10	Выполнить бегущий огонь двух светодиодов с тушением светодиодов между переключениями с частотой 1Гц

Продолжение таблицы 11

11	Выполнить бегущую тень одного светодиода с его включением между переключениями с частотой 1,5Гц
12	Выполнить бегущий огонь 2х светодиодов с их выключением между переключениями в шахматном порядке с частотой 0,5Гц
13	Выполнить включение накапливающихся светодиодов с их выключением между переключениями с частотой переключения 1Гц
14	Выполнить попеременное включение светодиодов D2, D3 и D4, D5 с их выключением между переключениями с частотой 1,5Гц
15	Выполнить включение светодиодов в последовательности D3, D5, D4, D2 с их выключением между переключениями с частотой 0,5Гц
16	Выполнить включение светодиодов в последовательности D4, D2, D5, D3 с их выключением между переключениями с частотой 0,5Гц
17	Выполнить бегущий огонь одного светодиода с реверсом на конечном значении с частотой 1,5Гц
18	Выполнить бегущий огонь двух светодиодов с реверсом на конечном значении с частотой 0,5Гц
19	Выполнить бегущую тень одного светодиода с реверсом на конечном значении с частотой 1Гц
20	Выполнить зажигание светодиодов по двоичному закону с частотой смены комбинации 1Гц
21	Выполнить бегущий огонь одного светодиода с двукратным зажиганием в позиции с частотой 0,5Гц
22	Выполнить бегущий огонь двух светодиодов с двукратным зажиганием в позиции с частотой 0,5Гц
23	Выполнить бегущую тень одного светодиода с двукратным погасанием в позиции с частотой 1,5Гц
24	Выполнить бегущую тень двух светодиодов с двукратным погасанием в позиции с частотой 1,5Гц
25	Выполнить попеременное включение светодиодов D2, D3 и D4, D5 с двукратным зажиганием в позиции с частотой 1Гц
26	Выполнить включение светодиодов в последовательности D3, D5, D4, D2 с двукратным зажиганием в позиции с частотой 1,5Гц
27	Выполнить включение светодиодов в последовательности D4, D2, D5, D3 с двукратным зажиганием в позиции с частотой 0,5Гц

Продолжение таблицы 11

28	Выполнить последовательное накапливание, а затем убывание светодиодов в другую сторону с частотой переключений 0,5Гц
29	Выполнить плавное зажигание светодиодов D2 и D3 в течение 3сек., а затем резкое тушение.
30	Выполнить плавное тушение светодиодов D3 и D4 в течение 4сек., а затем их резкое зажигание.

7.3 Практическое задание №2

Тема: Таймеры и система прерываний микроконтроллера PIC16F877.

Цель: Изучить возможности и практически освоить написание программ с использованием таймеров и системы прерываний.

Таблица 12 – Варианты заданий к самостоятельной работе

Вариант	**Задание**
1	Выполнить бегущий огонь на светодиодах используя для задержки таймер и систему прерываний. Частота сдвига составляет 1 Гц. Кнопкой S3 изменять направление, а S2 – останавливать/запускать бегущий огонь.
2	По нажатию кнопки S3 выдать звуковой сигнал с помощью пьезоизлучателя P1 частотой 1000Гц. При повторном нажатии на S3 выключить сигнал. Нажатием кнопки S2 выполнить смену частоты сигнала до 500Гц. Повторное нажатие на S2 выполняет возврат частоты до 1000Гц.
3	Выполнить формирование звукового сигнала пьезоизлучателем P1 частотой 1500Гц. При нажатии на S3 увеличивать частоту сигналов с дискретностью 250Гц. При нажатии на S2 – уменьшать с дискретностью 250Гц.
4	Выдать звуковой сигнал через пьезоизлучатель P1 с частотой 1000Гц длительностью 0,5с, затем сигнал частотой 500Гц длительностью 0,5с. При нажатии на S3 уменьшать длительность на 0,1с. При нажатии на S2 – увеличивать на 0,1с.
5	Выполнить бегущий огонь на светодиодах при помощи нажатия на кнопки. S3 сдвигает огонь влево, S2 – вправо. При нажатии на кнопки S3, S2 выполнить их «озвучку» на пьезоизлучателе частотой 1000Гц длительностью 0,2с для S3 и 500Гц длительностью 0,2с для S2.

Продолжение таблицы 12

6	Выполнить плавное зажигание светодиодов по «треугольному» закону, используя один из таймеров для задания частоты, а другой – длительности (ШИМ). Период импульсов – 2с.
7	Выполнить плавное зажигание светодиодов по «пилообразному» закону, используя один из таймеров для задания частоты, а другой – длительности (ШИМ). Период импульсов – 2с.
8	Выполнить бегущий огонь на светодиодах при помощи нажатия на кнопки. S3 сдвигает огонь влево, S2 – вправо. При зажигании четных светодиодов выдавать короткий звуковой сигнал на пьезоизлучателе P1 частотой 1500Гц, нечетных – 750Гц.
9	Выполнить проигрывание музыкального отрывка с произвольными тонами. Для задания частоты использовать один таймер, длительности – другой. Для увеличения длительности тона разрешается использовать дополнительное деление в регистрах. Число тонов – не менее 5.
10	По нажатию на кнопку S3 выполнить бегущий огонь. При смене светящегося светодиода выполнить звуковой сигнал частотой 1000Гц, длительностью 0,2с. При достижении последнего светодиода выдать звуковой сигнал частотой 750Гц, длительностью 0,5с и изменить направление бегущего огня. При повторном нажатии на S3 потушить светодиоды и выдать двухкратный сигнал частотой 1500Гц и длительностью 0,2с. Пауза между сигналами – 0,2с.
11	Выдать звуковой сигнал частотой 1000Гц длительностью 0,5с, после этого сдвинуть светящийся светодиод. При нажатии на S3 выполнить паузу 5с. Нажатием кнопки S2 можно прервать паузу. Если не нажата ни одна из кнопок – повторить процесс.
12	Выполнить бегущий огонь на светодиодах с интервалом сдвига 5с. Допускается дополнительное деление интервала времени в регистре. При нажатии на S3 должен гореть постоянно старший светодиод, а остальные продолжают бегущий огонь с интервалом в 2с. При повторном нажатии потушить старший светодиод и возвратиться к начальным условиям.
13	Выдавать звуковой сигнал частотой 1000Гц через пьезоизлучатель P1. При нажатии на S2 изменить частоту до 2000Гц. При нажатии на S3 – до 500Гц. При нажатии на обе кнопки – возвратиться к частоте 1000Гц. Светодиоды HL0, HL1, HL2 должны указывать текущую частоту звукового сигнала 500/1000/2000 соответственно.

Продолжение таблицы 12

14	По нажатию S3 выполнить бегущую тень на светодиодах. При зажигании светодиода подать звуковой сигнал частотой 500Гц и длительностью 0,25с. Повторное нажатие S3 останавливает движение. Нажатие S2 увеличивает частоту сигнала до 1000Гц.
15	По нажатию S3 выдать звуковые сигналы 500Гц, 750Гц и 1000Гц длительностью 0,2с. с паузами 0,5с. При нажатии на S2 увеличить длительности сигналов до 1с. Повторное нажатие S3 выключает подачу сигналов.

7.4 Практическое задание №3

Тема: Методы управления шаговыми двигателями.

Цель: Получить навыки в управлении шаговыми двигателями и освоить методы табличного описания функции.

Принципиальная схема подключения шагового двигателя к стенду PICDEM 2 Plus:

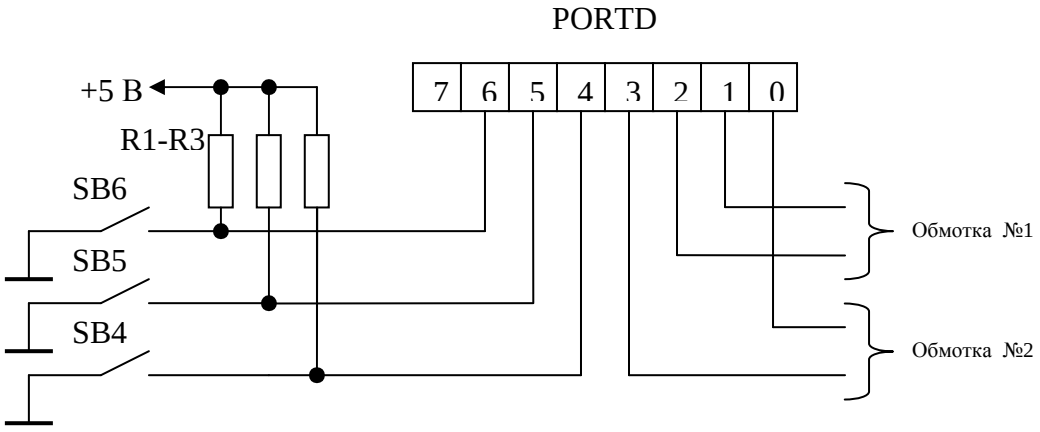


Рисунок 53 – Схема подключения модуля с шаговым двигателем

При управлении шаговым двигателем (ШД), на него необходимо подавать последовательность управляющих импульсов, которые в зависимости от типа двигателя, его схемы включения и режима работы изменяются. Для движения двигателя в одну сторону последовательность прямая. Для движения в обратную сторону – обратная последовательность. Различают несколько режимов работы:

– Полношаговый режим;
– Полушаговый режим;
– Микрошаговый режим.

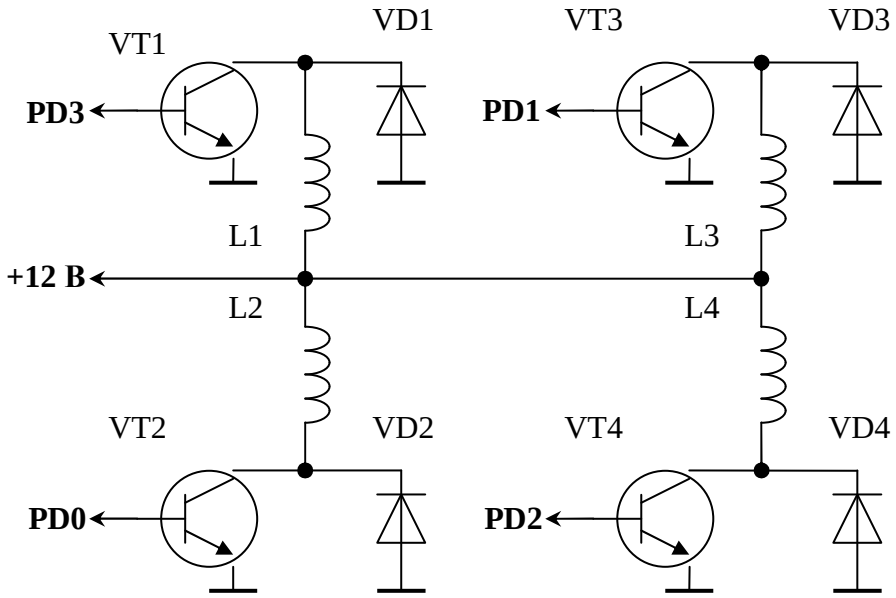


Рисунок 54 – Принципиальная электрическая схема коммутации обмоток ШД

Для осуществления микрошагового режима, кроме управляющей последовательность коммутации обмоток, выполняют еще и ШИМ тока, проходящего через обмотку.

Последовательности импульсов для вращения шагового двигателя с двумя обмотками (Рис. 54), имеющими общие точки, в некоторых режимах приведены ниже:

Последовательность коммутации обмоток в **полношаговом режиме при однофазной коммутации тока**:

№ Шага	PORTD
1	XXXX0001
2	XXXX0010
3	XXXX0100
4	XXXX1000
1	XXXX0001
Выкл.	XXXX0000

Последовательность коммутации обмоток в **полношаговом режиме при двухфазной коммутации тока**:

№ Шага	PORTD
1	XXXX0011
2	XXXX0110
3	XXXX1100
4	XXXX1001
1	XXXX0011
Выкл.	XXXX0000

Последовательность коммутации обмоток в **полушаговом режиме при двухфазной коммутации тока**:

№ Шага	PORTD
1	XXXX0001
2	XXXX0011
3	XXXX0010
4	XXXX0110
5	XXXX0100
6	XXXX1100
7	XXXX1000
8	XXXX1001
1	XXXX0001
Выкл.	XXXX0000

При подаче управляющей последовательности на ШД необходимо не забывать, что двигатель имеет большую электрическую и механическую инерционность по сравнению со скоростью работы микроконтроллера. Это означает, что при смене кода шага необходимо выполнить задержку времени не менее 0,005 сек.

Таблица 13 – Варианты заданий к самостоятельной работе

Вариант	**Задание**
1	Выполнить вращение ШД в полношаговом режиме при однофазной коммутации тока в обмотках. При нажатии на кнопку S3 увеличивать скорость вращения, при нажатии S2 – уменьшать. Максимальное значение скорости вращения указывать светодиодом D4, минимальное – D2, среднее – D3.
2	Выполнить вращение ШД в полношаговом режиме при двухфазной коммутации тока в обмотках. Кнопка S3 выполняет старт/стоп функцию. Кнопка S2 – реверс. Светодиоды информируют о режимах: D2 – вращение в прямом направлении, D3 – стоп, D4 – вращение в обратном направлении.
3	Выполнить вращение ШД в полушаговом режиме при двухфазной коммутации тока в обмотках. По истечении 1 сек. двигатель изменяет направление вращения после остановки в 0,5 сек. Кнопка S3 выполняет функцию старт/стоп. Кнопка S2 изменяет время работы двигателя 1/0,5 с. Время остановки остается неизменным.
4	Выполнить вращение ШД в полношаговом режиме при однофазной коммутации тока в обмотках. Во время работы производится подсчет числа шагов (максимум 2047). При нажатии на S3 выполняется возврат ротора на количество шагов, подсчитанных до нажатия данной кнопки. S2 выполняет запуск двигателя из начального положения. Светодиоды: D4 – достигнут максимальный шаг; D3 – двигатель находится в начальном положении.

Продолжение таблицы 13

5	Выполнить вращение ШД в полушаговом режиме при двухфазной коммутации тока в обмотках. Кнопка S3 – шаг в прямом направлении; S2 – шаг в обратном направлении. Номер шага двигателя выводить на светодиоды D3-D5 в двоичном коде.
6	Выполнить вращение ШД в полушаговом режиме при двухфазной коммутации тока в обмотках. При нажатии на кнопку S3 двигатель плавно запускается в течение 3 сек. При нажатии на S2 выполняет реверс с плавным торможением и разгоном. Повторное нажатие S3 выполняет плавную остановку.
7	Выполнить вращение ШД в полношаговом режиме при однофазной коммутации тока в обмотках. По нажатию кнопки S3 выполнить технологический цикл: разгон в течение 2 сек., работа 5 сек., реверс 4 сек., работа в обратном направлении 5 сек., останов 2 сек. Кнопка S2 – аварийный останов. Светодиоды индицируют состояния: D5 – ускорение; D4 – работа; D3 – замедление.
8	Выполнить вращение ШД в полношаговом режиме при двухфазной коммутации тока в обмотках. Функция изменения скорости – синусоидальная. Кнопка S2 – старт/стоп.
9	Выполнить вращение ШД в полношаговом режиме при однофазной коммутации тока в обмотках. Двигатель выполняет вращение. При нажатии на кнопку S3 он делает 40 шагов в обратном направлении со скоростью в 4 раза меньше основного вращения, а затем продолжает вращение в основном направлении. При нажатии на S2 – выполняет 20 шагов со скоростью в 4 раза меньшей в основном направлении, а затем вращается в обратном направлении.
10	Выполнить вращение ШД в полношаговом режиме при двухфазной коммутации тока в обмотках. При нажатии на кнопку S2 двигатель выполняет шаг в прямом направлении. При достижении 10 шага двигатель начинает вращаться в течение 1 сек. Затем делает останов и медленно выполняет 10 шагов в обратном направлении. Светодиоды D2-D5 указывают на текущий шаг двигателя (D2 – первый… D5 – четвертый).
11	Выполнить вращение ШД в полушаговом режиме при двухфазной коммутации тока в обмотках. Двигатель выполняет 20 шагов в прямом направлении, затем 20 шагов в обратном. Нажатие кнопки S3 добавляет 2 шага, S2 – уменьшает на 2 шага. Светодиод D4 – движение в прямом направлении; D5 – в обратном.

Продолжение таблицы 13

12	Выполнить вращение ШД в полношаговом режиме при однофазной коммутации тока в обмотках. По нажатию на кнопку S3 двигатель медленно выполняет 5 шагов с интервалом 0,25 сек., затем вращается в течение 5 сек., выполняет 5 шагов с интервалом 0,25 сек. в обратном направлении и останавливается. Нажатие кнопки S2 добавляет число медленных шагов на 2.
13	Выполнить вращение ШД в полношаговом режиме при двухфазной коммутации тока в обмотках. Двигатель постоянно находится в режиме реверса. Интервал между шагами – 0,2 сек. Удержание кнопки S3 выполняет вращение двигателя в прямом направлении с высокой скоростью, а удержание S2 – в обратном. Светодиоды индицируют состояния: D3 – медленный шаг в прямом направлении; D4 – большая скорость вращения независимо от направления; D5 – медленный шаг в обратном направлении.
14	Выполнить вращение ШД в полушаговом режиме при двухфазной коммутации тока в обмотках. Нажатие кнопки S3 выполняет 1 шаг двигателя в прямом направлении. При достижении 10 шага, двигатель начинает быстро вращаться в течение 2 сек., затем останавливается. Нажатие S2 выполняет те же действия, только в обратном направлении. Светодиод D3 указывает нажатие S3; D4 – S2.
15	Выполнить вращение ШД в полношаговом режиме при однофазной коммутации тока в обмотках. Двигатель выполняет вращение в течение 1 сек., и паузу в 1 сек. циклически. Нажатие S3 уменьшает паузу на 0,2 сек.; S2 – увеличивает на 0,2 сек. Светодиод D3 указывает, что двигатель работает без пауз; D4 – пауза меньше 2 сек.; D5 – пауза больше 2 сек.

ЛИТЕРАТУРА

1 Микропроцессорные системы: Учебное пособие для ВУЗов / Е. К. Александров, Р. И. Грушвицкий, М. С. Куприянов, О. Е. Мартынов, Д. И. Панфилов, Т. В. Ремизевич, Ю. С. Татаринов, Е. П. Угрюмов, И. И. Шагурин; Под общ. ред. Д. В. Пузанкова. – СПб.: Политехника, 2002. – 935 с.

2 Локазюк В. М. Мікропроцесори та мікроЕОМ у виробничих системах: Посібник. – К.: Академія, 2002. – 368 с.

3 Каспер Э. Программирование на языке Ассемблера для микроконтроллеров семейства i8051. – М.: Горячая линия – Телеком, 2004. – 191 с. – ISBN 5-93517-104-X.

4 Предко, М. Справочник по PIC-микроконтроллерам / М. Предко, пер с англ. – М.: ДМК Пресс, 2002, ООО Издательский дом «Додэка-XXI», 2002. – 512 с.

ПРИЛОЖЕНИЕ А

Система команд микроконтроллера I8051

Микро-ЭВМ рассматриваемого семейства являются типичными микропроцессорными устройствами с архитектурой SISC – со стандартным набором команд. Поэтому их система команд довольно обширна и включает в себя 111 основных команд. Их длина – один, два или три байта, причем большинство из них (94%) – одно- или двухбайтные. Все команды выполняются за один или два машинных цикла, исключение – команды умножения и деления, которые выполняются за четыре машинных. Микро-ЭВМ семейства 8051 используют прямую, непосредственную, косвенную и неявную, адресацию данных

В качестве операндов команд микро-ЭВМ семейства 8051 могут использовать отдельные биты, четырехбитные цифры, байты и двухбайтные слова.

Все эти черты обычны для набора команд любого CISC-процессора и по сравнению с RISC набором команд обеспечивает большую компактность программного кода и увеличение быстродействия при выполнении сложных операций.

Типы команд

Всего микро-ЭВМ выполняют 13 типов команд, они приведены в таблице. Как следует из нее, первый байт команды всегда содержит код операции (КОП), а второй и третий (если они присутствуют в команде) – адреса операндов или их непосредственные значения.

Таблица А.1 – Типы команд I8051

Тип команды	Первый байт D7...D0	Второй байт D7...D0	Третий байт D7...D0
тип 1	коп		
тип 2	коп	#d	
тип 3	коп	ad	
тип 4	коп	bit	
тип 5	коп	rel	
тип 6	коп	a7...a0	
тип 7	коп	ad	#d
тип 8	коп	ad	rel
тип 9	коп	ads	add
тип 10	коп	#d	rel
тип 11	коп	bit	rel
тип 12	коп	ad16h	ad16l
тип 13	коп	#d16h	#d16l

Группы команд

Все команды микро-ЭВМ семейства 8051 можно разбить на пять функциональных групп:

– Пересылки данных;
– Арифметических операций;
– Логических операций;
– Операций над битами;
– Передачи управления.

Обозначения, используемые при описании команд

Rn (n = 0, 1,..., 7) – регистр общего назначения в выбранном банке регистров;

@Ri(i = 0, 1) – регистр общего назначения в выбранном банке регистров, используемый в качестве регистра косвенного адреса;

ad – адрес прямоадресуемого байта;
ads – адрес прямо адресуемого байта-источника;
add – адрес прямо адресуемого байта-получателя;
ad11 – 11-разрядный абсолютный адрес перехода;
ad16 – 16-разрядный абсолютный адрес перехода;
rel – относительный адрес перехода;
#d – непосредственный операнд;
#d16 – непосредственный операнд (2 байта);
bit – адрес прямо адресуемого бита;
/bit – инверсия прямо адресуемого бита;
A – аккумулятор;
PC – счетчик команд;
DPTR – регистр указатель данных;
( ) – содержимое ячейки памяти или регистра,

Команды пересылки данных микроконтроллера 8051

В таблицах указаны тип команды (Т) в соответствии с таблицей А.1, ее длина в байтах (В) и время выполнения в машинных циклах (С)

Таблица А.2 – Команды пересылки данных

Мнемокод	КОП	Т В С	Описание
MOV A, Rn	11101rrr	1 1 1	(A) <– (Rn)
MOV A, ad	11100101	3 2 1	(A) <–(ad)
MOV A, @Ri	1110011i	1 1 1	(A) <– ((Ri))
MOV A, #d	01110100	2 2 1	(A) <– #d
MOV Rn, A	11111rrr	1 1 1	(Rn) <– (A)
MOV Rn, ad	10101rrr	3 2 2	(Rn) <– (ad)
MOV Rn, #d	01111rrr	2 2 1	(Rn) <– #d
MOV ad, A	11110101	3 2 1	(ad) <–(A)
MOV ad, Rn	10001rrr	3 2 2	(ad) <– (Rn)

Продолжение таблицы А.2

MOV add, ads	10000101	9 3 2	(add) <– (ads)
MOV ad, @Ri	1000011i	3 2 2	(ad) <– ((Ri))
MOV ad, #d	01110101	7 3 2	(ad) <– #d
MOV @Ri, A	1111011i	1 1 1	((Ri)) <– (A)
MOV @Ri, ad	0110011i	3 2 2	((Ri)) <– (ad)
MOV @Ri, #d	0111011i	2 2 1	((Ri)) <– #d
MOV DPTR, #d16	10010000	3 3 2	(DPTR) <– #d16
MOVC A, @A+DPTR	10010011	1 1 2	(A) <– ((A)+(DPTR))
MOVC A, @A+pc	10000011	4 1 2	(PC) <– (PC+1), (A) <– ((A)+(PC))
MOVX A,@Ri	11100011	1 1 2	(A) <– ((Ri))
MOVX a, @DPTR	11100000	1 1 2	(A) <– ((DPTR))
MOVX @Ri, A	1111001i	1 1 2	((Ri)) <– (A)
MOVX @DPTR, A	11110000	1 1 2	(DPTR)) <– (A)
PUSH ad	11000000	3 2 2	(SP) <– (SP)+1, ((SP)) <– (ad)
POP ad	11010000	3 2 2	(ad) <– ((SP)), (SP) <– (SP)-1
XCH A, Rn	11001rrr	1 1 1	(A) <-> (Rn)
XCH A, ad	11000101	3 2 1	(A) <–>(ad)
XCH A, @Ri	11000111	1 1 1	(A) <->(((@Ri))
A, @Ri	11010111	1 1 1	(A0-3)<->(((@Ri0-3))

По команде MOV выполняется пересылка данных из второго операнда в первый. Эта команда не имеет доступа ни к внешней памяти данных, ни к памяти программ. Для этих целей предназначены команды M0VX и MOVC соответственно. Первая из них обеспечивает чтение/запись байт из внешней памяти данных, вторая – чтение байт из памяти программ.

По команде XCH выполняется обмен байтами между аккумулятором и ячейкой РПД, а по команде XCHD – обмен младшими.

Команды PUSH и POP предназначены соответственно для записи данных в стек и их чтения из стека. Размер стека ограничен лишь размером резидентной памяти данных.

Группа команд пересылок микроконтроллера имеет следующую особенность – в ней нет специальных команд для работы со специальными регистрами: PSW, таймером, портами ввода-вывода. Доступ к ним, как и к другим регистрам специальных функций, осуществляется заданием соответствующего прямого адреса, т. е. это команды обычных пересылок, в которых вместо адреса можно ставить название соответствующего регистра.

Кроме того, следует отметить, что в микро-ЭВМ аккумулятор имеет два различных имени в зависимости от способа адресации: A – при неявной адресации (например, MOV A,R0) и ACC – при использовании прямого адреса. Первый способ предпочтительнее, однако, не всегда применим.

Команды арифметических операций

Таблица А.3 – Команды арифметических операций

Мнемокод	КОП	Т	В	С	Описание
ADD A, Rn	00101rrr	1	1	1	(A)<-(A)+(Rn)
ADD A, ad	00100101	3	2	1	(A)<-(A)+(ad)
ADD A, @Ri	0010011i	1	1	1	(A)<-(A)+((Ri))
ADD A, #d	00100100	2	2	1	(A)<-(A)+#d
ADDC A, Rn	00111rrr	1	1	1	(A)<-(A)+(Rn)+(C)
ADDC A, ad	00110101	3	2	1	(A)<-(A)+(ad)+(C)
ADDC A, @Ri	0011011i	1	1	1	(A)<-(A)+((Ri))+C)
ADDC A, #d	00110100	2	2	1	(A)<-(A)+#d+(C)
DA A	11010100	1	1	1	Десятичная коррекция аккумулятора
SUBB A, Rn	10011rrr	1	1	1	(A)<-(A)-(Rn)-(C)
SUBB A, ad	10010101	3	2	1	(A)<-(A)-(ad)-(C)
SUBB A, @Ri	1001011i	1	1	1	(A)<-(A)-((Ri))-(C)
SUBB A, #d	10010100	2	2	1	(A)<-(A)-#d-(C)
INC A	00000100	1	1	1	(A)<-(A)+1
INC Rn	00001rrr	1	1	1	(Rn)<-(Rn)+1
INC ad	00000101	3	2	1	(ad)<-(ad)+1
INC @Ri	0000011i	1	1	1	((Ri)<-((Ri))+1
INC DPTR	10100011	1	1	2	(DPTR)<-(DPTR)+1
DEC A	00010100	1	1	1	(A)<-(A)-1
DEC Rn	00011rrr	1	1	1	(Rn)<-(Rn)-1
DEC ad	00010101	3	2	1	(ad)<-(ad)-1
DEC @Ri	0001011i	1	1	1	((Ri))<-((Ri))-1
MUL AB	10100100	1	1	4	(B)(A)<-(A)*(B)
DIV AB	10000100	1	1	4	(A).(B)<-(A)/(B)

По результату выполнения команд ADD, ADDC, SUBB, MUL и DIV устанавливаются флаги PSW.

Флаг C устанавливается при переносе из разряда D7, т. е. в случае, если результат не помещается в восемь разрядов; флаг AC устанавливается при переносе из разряда D3 в командах сложения и вычитания и служит для реализации десятичной арифметики. Этот признак используется командой DA A.

Флаг OV устанавливается при переносе из разряда D6, т. е. в случае, если результат не помещается в семь разрядов и восьмой не может быть интерпретирован как знаковый. Этот признак служит для организации обработки чисел со знаком.

Флаг P устанавливается и сбрасывается аппаратно. Если число единичных бит в аккумуляторе нечетно, то P=1, в противном случае P=0.

Команды логических операций

В этой группе 25 команд, их краткое описание приведено в таблице. Нетрудно видеть, что эти команды позволяют выполнять операции над байтами: логическое И (/\), логическое ИЛИ (\/), исключающее ИЛИ (+), инверсию (NOT), сброс в нулевое значение и сдвиг. Команды, оперирующие отдельными битами, описаны далее.

Таблица А.4 – Команды логических операций

Мнемокод	КОП	Т В С	Описание
ANL A, Rn	01011rrr	1 1 1	(A) <– (A) /\ (Rn)
ANL A, ad	01010101	3 2 1	(A) <– (A) /\ (ad)
ANL A, @Ri	01010111	1 1 1	(A) <– (A) /\ ((Ri))
ANL A, #d	01010100	2 2 1	(A) <– (A) /\ #d
ANL ad, A	01010010	3 2 1	(ad) <– (ad) /\(A)
ANL ad, #d	01010011	7 3 2	(ad) <– (ad) /\ #d
ORL A, Rn	01001rrr	1 1 1	(A) <– (A) \/ (Rn)
ORL A, ad	01000101	3 2 1	(A) <– (A) \/ (ad)
ORL A, @Ri	0100011i	1 1 1	(A) <– (A) \/((Ri)
ORL A, #d	01000100	2 2 1	(A) <– (A) \/ #d
ORL ad, A	01000010	3 2 1	(ad) <– (ad) \/ A
ORL ad, #d	01000011	7 3 2	(ad) <– (ad) \/ #d
XRL A, Rn	01101rrr	1 1 1	(A) <– (A) (+) (Rn)
XRL A, ad	01100101	3 2 1	(A) <– (A) (+) (ad)
XRL A, @Ri	0110011i	1 1 1	(A) <– (A) (+) ((Ri))
XRL A, #d	01100100	2 2 1	(A) <– (A) (+) #d
XRL ad, A	01100010	3 2 1	(ad) <– (ad) (+) A
XRL ad, #d	01100011	7 3 2	(ad) <– (ad) (+) #d
CLR A	11100100	1 1 1	(A) <– 0
CPL A	11110100	1 1 1	(A) <– NOT(A)
SWAP A	11000100	1 1 1	(A0-3) <–> (A4-7)
RL A	00100011	1 1 1	Циклический сдвиг влево
RLC A	00110011	1 1 1	Сдвиг влево через перенос
RR A	00000011	1 1 1	Циклический сдвиг вправо
RRC A	00010011	1 1 1	Сдвиг вправо через перенос

Команды операций над битами

Группа состоит из 12 команд, краткое описание которых приведено в таблице. Эти команды позволяют выполнять операции над отдельными битами: сброс, установку, инверсию бита, а также логические И (/\) и ИЛИ (\/). В качестве «логического» аккумулятора, участвующего во всех операциях с двумя операндами, выступает признак переноса С (разряд D7 PSW), в качестве операндов могут использоваться 128 бит из резидентной

памяти данных и регистры специальных функций, допускающие адресацию отдельных бит.

Таблица А.5 – Команды операций над битами

Мнемокод	КОП	Т В С	Описание
CLR C	11000011	1 1 1	(C) <– 0
CLR bit	11000010	4 2 1	(bit) <– 0
SETB C	11010011	1 1 1	(C) <– 1
SETB bit	11010010	4 2 1	(bit) <– 1
CPL C	10110011	1 1 1	(C) <– NOT(C)
CPL bit	10110010	4 2 1	(bit) <– NOT (bit)
ANL C, bit	10000010	4 2 2	(C)<– (C) /\ (bit)
ANL C, /bit	10110000	4 2 2	(C) <– (C) /\ NOT(bit)
ORL C, bit	01110010	4 2 2	(C) <– (C) \/ (bit)
ORL C, /bit	10100000	4 2 2	(C) <– (C) \/ NOT(bit)
MOV C, bit	10100010	4 2 1	(C) <– (bit)
MOV bit, C	10010010	4 2 2	(bit) <– (C)

Команды передачи управления

Группа представлена командами безусловного и условного переходов, командами вызова подпрограмм и командами возврата из подпрограмм.

Таблица А.6 – Команды передачи управления

Мнемокод	КОП	Т В С	Описание
LJMP ad16	00000010	12 3 2	Длинный безусловный переход по всей памяти
AJMP ad11	00001	6 2 2	Безусловный переход в пределах страницы 2 Кбайт
SJMP rel	10000000	5 2 2	Безусловный переход в пределах страницы 256 байт
JMP @A+DPTR	01110011	1 1 2	Безусловный переход по косвенному адресу
JZ rel	01100000	5 2 2	Переход, если нуль
JNZ rel	01110000	5 2 2	Переход, если не нуль
JC rel	01000000	5 2 2	Переход, если бит переноса установлен
JNC rel	01010000	5 2 2	Переход, если бит переноса не установлен
JB bit, rel	00100000	11 3 2	Переход, если бит установлен
JNB bit, rel	00110000	11 3 2	Переход, если бит не установлен
JBC bit, rel	00010000	11 3 2	Переход, если бит установлен со сбросом бита
DJNZ Rn, rel	11011rrr	5 2 2	Декремент и переход, если не нуль
DJNZ ad, rel	11010101	8 3 2	Декремент и переход, если не нуль

Продолжение таблицы А.6

CJNE: A, ad, rel	10110101	8 3 2	Сравнение аккумулятора с байтом и переход, если не равно
CJNE A, #d, rel	10110100	10 3 2	Сравнение аккумулятора с константой и переход, если неравно
CJNE: Rn, #d, rel	10111rrr	10 3 2	Сравнение регистра с константой и переход, если не равно
CJNE: @Ri, #d, rel	1011011i	10 3 2	Сравнение байта памяти с константой и переход, если не равно
LCALL ad16	00010010	12 3 2	Длинный вызов подпрограммы во всей памяти
ACALL ad11	10001	6 2 2	Вызов подпрограммы в пределах страницы 2 Кбайт
RET	00100010	1 1 2	Возврат подпрограммы
RETI	00110010	1 1 2	Возврат подпрограммы обработки прерывания
NOP	00000000	1 1 1	Пустая операция

Команда безусловного перехода LJMP (long– длинный) осуществляет переход по абсолютному 16-битному адресу, указанному в теле команды, т. е. команда обеспечивает переход в любую точку памяти программ.

Действие команды AJMP (absolute – абсолютный) аналогично команде LJMP, однако в теле команды указаны лишь 11 младших разрядов адреса. Поэтому переход осуществляется в пределах страницы размером 2 Кбайт, при этом надо иметь в виду, что сначала содержимое счетчика команд увеличивается на 2 и, только потом, заменяются 11 разрядов адреса.

В отличие от предыдущих команд, в команде SJMP (short – короткий) указан не абсолютный, а относительный адрес перехода. Величина смещения rel рассматривается как число со знаком, а, следовательно, переход возможен в пределах -128...+127 байт относительно адреса команды, следующей за командой SJMP.

Команда косвенного перехода JMP @A+DPTR позволяет вычислять адрес перехода в процессе выполнения самой программы.

Командами условного перехода можно проверять следующие условия:

JZ – аккумулятор содержит нулевое значение;

JNZ – аккумулятор содержит не нулевое значение;

JC – бит переноса C установлен;

JNC – бит переноса C не установлен;

JB – прямо адресуемый бит равен 1;

JNB – прямо адресуемый бит равен 0;

JBC – прямо адресуемый бит равен 1 и сбрасывается в нулевое значение при выполнении команды.

Все команды условного перехода рассматриваемых ОЭВМ, содержат короткий относительный адрес, т. е. переход может осуществляться в пределах -128... +127 байт относительно следующей команды.

Команда DJNZ предназначена для организации программных циклов. Регистр Rn или байт по адресу ad, указанные в теле команды, содержат счетчик повторений цикла, а смещение rel – относительный адрес перехода к началу цикла. При выполнении команды содержимое счетчика уменьшается на 1 и проверяется на 0. Если значение содержимого счетчика не равно 0, то осуществляется переход на начало цикла, в противном случае выполняется следующая команда.

Действие команд вызова процедур полностью аналогично действию команд безусловного перехода. Единственное отличие состоит в том, что они сохраняют в стеке адрес возврата.

Команда возврата из подпрограммы RET восстанавливает из стека значение содержимого счетчика команд, а команда возврата из процедуры обработки прерывания RETI, кроме того, разрешает прерывание обслуженного уровня. Команды RET и RETI не различают, какой командой LCALL или ACALL была вызвана подпрограмма, т. к. и в том, и в другом случае в стеке сохраняется полный 16-разрядный адрес возврата.

В заключение следует отметить, что большинство трансляторов допускают обобщенную мнемонику JMP – для команд безусловного перехода и CALL – для команд вызова подпрограмм. Конкретный тип команды определяется транслятором, исходя из «длины» перехода или вызова.

ПРИЛОЖЕНИЕ Б

Система команд микроконтроллера PIC16F877

Длина каждой инструкции составляет 14-ти битное слово, разделенное на OPCODE (часть кода команды), которая указывает на тип исполняемой команды, и один или несколько операндов, которые в свою очередь в дальнейшем определяют дальнейшее выполнение инструкции.

Для описания команд семейства PICmicro используют следующие сокращения:

Таблица Б.1 – Используемые сокращения системы команд PICmicro

Поле	Описание
f	Адрес файлового регистра (от 0x00 до 0x7F).
W	Рабочий регистр (аккумулятор).
b	Адрес бита внутри 8-ми битного регистра.
k	Символьное поле, константа или метка.
x	Любое значение(0 или 1). Компилятор сгенерирует код с x=0 – это нужно для совместимости со всеми программными продуктами Microchip.
d	Выбор, где сохранять результат: d=0 (сохранять в W); d=1 (сохранять в f). По умолчанию d=1.
label	Имя метки
TOS	Вершина стека
PC	Счетчик команд (программный счетчик)
PCLATH	Записываемый буфер для старших 5-ти бит PC
GIE	Флаг разрешения глобальных прерываний
WDT	Сторожевой таймер
$\overline{TO}$	бит Таймаута (Time-out bit)
$\overline{PD}$	бит понижения питания (Power-Down bit)
dest	Назначение, либо регистр W или другой регистр указанный в описании команды
[ ]	Опционально, т. е. необязательное использование записи, которая заключена в квадратные скобки
( )	Содержимое
–>	Занести в
< >	Битовое поле в регистре
∈	принадлежит
Курсив	определяется пользователем

Команды подразделяются на байт-ориентированные, бит-ориентированные, символьные и команды управления. На рис. Б.1 указано,

из каких составных частей состоит код команды в зависимости от ее принадлежности к выше определенным группам.

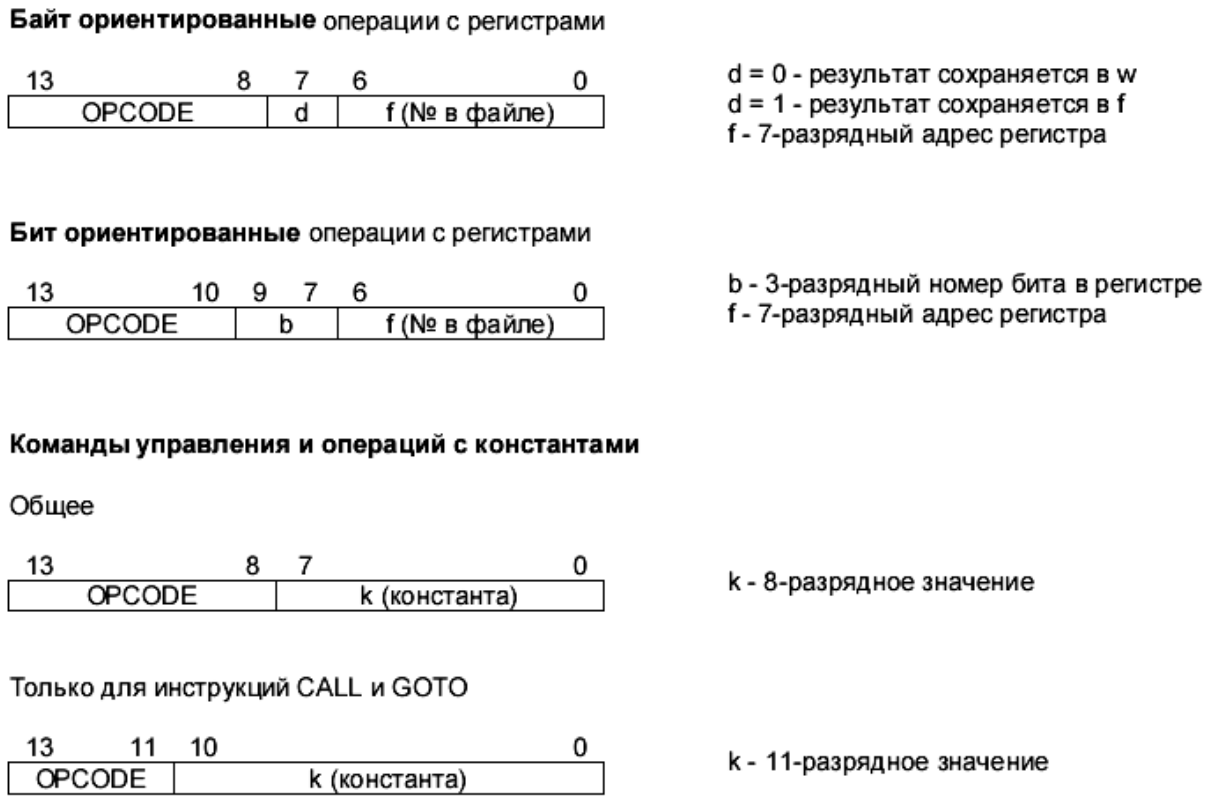


Рисунок Б.1 – Форматы команд

Таблица Б.2 – Система команд PICmicro

Мнемоника команды	**Описание**	**Циклов**	**14-разрядный код**		**Изм. флаги**	**Прим ечеч.**
			Бит 13	**Бит 0**		
1	**2**	**3**	**4**		**5**	**6**
БАЙТ-ОРИЕНТИРОВАННЫЕ КОМАНДЫ						
ADDWF f, d	Сложение W и f	1	00 0111 dfff ffff		C,DC, Z	1.2
ANDWF f, d	Побитное «И» W и f	1	00 0101 dfff ffff		Z	1.2
CLRF f	Очистить f	1	00 0001 lfff ffff		Z	2
CLRW	Очистить W	1	00 0001 0xxx xxxx		Z	
COMF f, d	Инвертировать f	1	00 1001 dfff ffff		Z	1.2
DECF f, d	Вычесть 1 из f	1	00 0011 dfff ffff		Z	1.2
DECFSZ f, d	Вычесть 1 из f и пропустить если 0	1(2)	00 1011 dfff ffff			1.2.3
INCF f, d	Прибавить 1 к f	1	00 1010 dfff ffff		Z	1.2
INCFSZ f, d	Прибавить 1 к f и пропустить если 0	1(2)	00 1111 dfff ffff			1.2.3
IORWF f, d	Побитное «ИЛИ» W и f	1	00 0100 dfff ffff		Z	1.2
MOVF f, d	Переслать f	1	00 1000 dfff ffff		Z	1.2
MOVWF f	Переслать W в f	1	00 0000 1fff ffff			
NOP	Нет операции	1	00 0000 0xx0 0000			
RLF f, d	Циклический сдвиг f влево через перенос	1	00 1101 dfff ffff		C	1.2

Продолжение таблицы Б.2

1	2	3	4	5	6
RRF f, d	Циклический сдвиг f вправо через перенос	1	00 1100 dfff ffff	C	1.2
SUBWF f, d	Вычесть W из f.	1	00 0010 dfff ffff	C,DC, Z	1.2
SWAPF f, d	Поменять местами полубайты в регистре f.	1	00 1110 dfff ffff		1.2
XORWF f, d	Побитное «исключающее ИЛИ» W и f.	1	00 0110 dfff ffff	Z	1.2
БИТ-ОРИЕНТИРОВАННЫЕ КОМАНДЫ					
BCF f, b	Очистить бит b в регистре f.	1	01 00bb bfff ffff		1.2
BSF f, b	Установить бит b в регистре f.	1	01 01bb bfff ffff		1.2
BTFSC f, b	Проверить бит b в регистре f, пропустить если 0.	1(2)	01 10bb bfff ffff		3
BTFSS f, b	Проверить бит b в регистре f, пропустить если 1.	1(2)	01 11bb bfff ffff		3
КОМАНДЫ УПРАВЛЕНИЯ И ОПЕРАЦИЙ С КОНСТАНТАМИ					
ADDLW k	Сложить константу с W	1	11 11 lx kkkk kkkk	C,DC, Z	
ANDLW k	Побитное «И» константы и W	1	11 1001 kkkk kkkk	Z	
CALL k	Вызов подпрограммы	2	10 0kkk kkkk kkkk		
CLRWDT	Очистить WDT	1	00 0000 0110 0100	-TO, -PD	
GOTO k	Безусловный переход	2	10 1kkk kkkk kkkk		
IORLW k	Побитное «ИЛИ» константы и W	1	11 1000 kkkk kkkk	Z	
MOVLW k	Переслать константу в W	1	11 00xx kkkk kkkk		
RETFIE	Возврат из подпрограммы с разрешением прерываний	2	00 0000 0000 1001		
RETLW k	Возврат из подпрограммы с загрузкой константы вW	2	11 01xx kkkk kkkk		
RETURN	Возврат из подпрограммы	2	00 0000 0000 1000		
SLEEP	Перейти в режим SLEEP	1	00 0000 0110 0011	-TO -PD	
SUBLW k	Вычесть W из константы	1	11 110x kkkk kkkk	C,DC, Z	
XORLW k	Побитное «исключающее ИЛИ» константы и W	1	11 1010 kkkk kkkk	Z	

Примечания:

1. Когда регистр ввода вывода используется для модификации самого себя (MOVF PORTB, 1), то для записи будут использоваться значения 0/1 непосредственно с ножек микроконтроллера, а не значение записанное в выходную защелку порта;

2. Если команда оперирует с регистром TMR0 (и при этом d=1), тогда предделитель обнуляется (если он относится к модулю Timer 0);

3. Если изменяется Программный Счетчик (PC) или условие истинно, тогда команда выполняется за 2 командных цикла. Второй командный цикл исполняется как команда NOP.

Навчальне видання

Методичні вказівки

до самостійної роботи
з дисципліни «Мікропроцесорні пристрої»
спеціальності 7.092203
«Електромеханічні системи автоматизації»

усіх форм навчання

(*Російською мовою*)

Укладачі: Наливайко Олександр Михайлович,
Пономарьов Дмитро Сергійович.

Редактор Ініціали Прізвище
Комп’ютерна верстка О. П. Ордіна

Підп. до друку . Формат 60 х 84/16.
Папір офсетний. Ум. друк. арк. Обл.-вид. арк.
Тираж прим. Зам. №

Видавець і виготівник
«Донбаська державна машинобудівна академія»
84313, м. Краматорськ, вул. Шкадінова, 72.
Свідоцтво про внесення суб’єкта видавничої справи
до Державного реєстру
серія ДК №1633 від 24.12.03.