МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«КАЗАНСКИЙ ГОСУДАРСТВЕННЫЙ ЭНЕРГЕТИЧЕСКИЙ УНИВЕРСИТЕТ»

Теория и принципы построения цифровых систем управления

Учебно-методическое пособие по курсовому проектированию

Казань 2025

УДК 621.3:519.8(075.8)

ББК 31.2в631.0я73

Т74 **Теория и принципы построения цифровых систем управления**: учебно-методическое пособие по курсовому проектированию / составители : М.Ф. Садыков . – Казань : КГЭУ 2025. – 92с.

Приведены общие рекомендации, методические указания по работе над курсовым проектом, варианты заданий курсового проектирования.

Предназначено для обучающихся всех форм по образовательной программе направления подготовки 13.03.02 Электроэнергетика и электротехника.

УДК 621.3:519.8(075.8)
ББК 31.2в631.0я73

Рекомендовано к изданию Учебно-методическим советом
Института электротехники и электроники

ОБЩИЕ РЕКОМЕНДАЦИИ ПО РАБОТЕ НАД КУРСОВЫМ ПРОЕКТОМ

Каждый студент выполняет вариант задания, обозначенный последними двумя цифрами его учебного шифра в зачетной книжке. Варианты заданий приведены в Приложении 1 .

Требования к выполнению курсового проекта

Разрабатываемое микроконтроллерное устройство (МКУ) представляет собой цифровую систему управления, включает в себя совокупность аппаратных (*Hard Ware*) и программных средств (*Soft Ware*). Аппаратные средства обеспечивают максимальную производительность или быстродействие, а программные средства – расширение круга задач, решаемых данной системой. Как правило, реализация вычислений программными средствами является менее быстродействующей. Поэтому в процессе проектирования микропроцессорных устройств, в целом необходимо определить оптимальные соотношения и распределение функций между аппаратными и программными средствами с целью получения заданных характеристик системы. При этом в процессе проектирования производительность системы, объем памяти, габариты, потребляемая энергия выступают в качестве критериев выбора конкретной структуры разрабатываемого МКУ. Обобщающим критерием выбора структуры МКУ при решении конкретной задачи, как правило, является стоимостный критерий, сочетающий в себе все перечисления выше.

Курсовое проектирование должно включать следующие этапы проектирования:

1. Изучение теоретического материала по возможностям и особенностям заданного микроконтроллера;
2. Обоснование выбора модулей в составе заданного микроконтроллера, необходимых для конкретного применения;
3. Разработка структуры и алгоритма функционирования разрабатываемого МКУ;
4. Разработка, построение и описание функциональной схемы МКУ;
5. Обоснование выбора и схемотехнического решения аппаратной части МКУ;
6. Описание элементной базы и используемых микросхем;

7. Расчет электрического сопряжения элементов МКУ;

8. Разработка и описание структуры программного обеспечения МКУ, его реализация на языке ассемблера для заданного микроконтроллера;

9. Отладки разработанной программы на языке ассемблера в интегрированной среде разработки.

Требования к оформлению курсового проекта

Курсовой проект включает в себя пояснительную записку и графическую часть.

Пояснительная записка к курсовому проекту должна быть выполнена в соответствии с общими требованиями к оформлению квалификационных работ и должна состоять из следующих обязательных разделов:

– титульного листа;
– содержания;
– введения;
– основных разделов;
– заключения;
– списка используемой литературы; – приложения.

В первом разделе необходимо описать теоретический материал, используемый при разработке МКУ. Разработка программных средств МКУ должна включать обоснования выбора элементной базы, обязательно описание принципиальной электрической схемы МКУ

Объем пояснительной записки составляет 25–30 листов. Описание программного обеспечения МКУ должно содержать структуру, блок-схему, текст программы на языке ассемблера с подробными комментариями и листинг программы.

Основной текст раздела пояснительной записки выполняется шрифтом Times New Roman и размером 14 пт. Все исходные тексты программ набираются тем же шрифтом, но размером 12 пт. Отступ начала абзаца должен составлять 1,25 см; межстрочный интервал – 1.

Название раздела выделяется «полужирным» начертанием шрифта, прописными буквами и начинается с номера раздела. Название подраздела также выделяется «полужирным» и отстоит от текста на расстоянии одного пустого абзаца. Новый раздел начинается с новой страницы.

Графическая часть проекта оформляется на листах стандартных форматов и включает в себя 3 чертежа.

Графическая часть курсового проекта должна содержать

1. Функциональную схему МКУ (формат А4).
2. Электрическую принципиальную схему МКУ с указанием спецификации в соответствии с ГОСТом (формат А3). 3. Блок-схему алгоритма программного обеспечения МКУ (формат А4).

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Структура и функционирование микроконтроллера *MC*68*HC*908*GP*32

Используемые в курсовом проекте микроконтроллеры семейства *68HC08/908* фирмы *Motorola* содержат процессорное ядро *CPU08*, *Flash*память емкостью до 128 Кбайт, ОЗУ данных емкостью от 128 байт до 2 Кбайт. В ряде моделей имеется также ЭСППЗУ емкостью 512 байт или 1 Кбайт. Большинство микроконтроллеров работают при напряжении питания 5,0 В, обеспечивая максимальную тактовую частоту 8 МГц. Некоторые модели работают при пониженном напряжении питания 3,0 В или 2,0 В. Все модели имеют 16-разрядные таймеры, большинство моделей содержит 8- или 10-разрядные АЦП.

Микроконтроллеры реализуют ряд эффективных способов адресации и имеют расширенный набор выполняемых команд, благодаря которым достигается высокая производительность при решении широкого круга практических задач управления различными объектами. Применение *Flash*памяти обеспечивает возможность программирования и репрограммирования микроконтроллеров 68HC908 от персонального компьютера непосредственно в составе реализуемой системы, используя последовательный интерфейс.

В курсовом проекте используется микроконтроллер *MC*68*HC*908*GP*32, который ориентирован на широкий диапазон применения в разнообразной цифровой аппаратуре.

Микроконтроллер *MC*68*HC*908*GP*32 содержит 8-ми разрядный процессор *CPU*08, *Flash*-память емкостью 32 Кбайт, ОЗУ данных емкостью 512 байт и большой набор служебных и периферийных модулей (рис. 1.1).

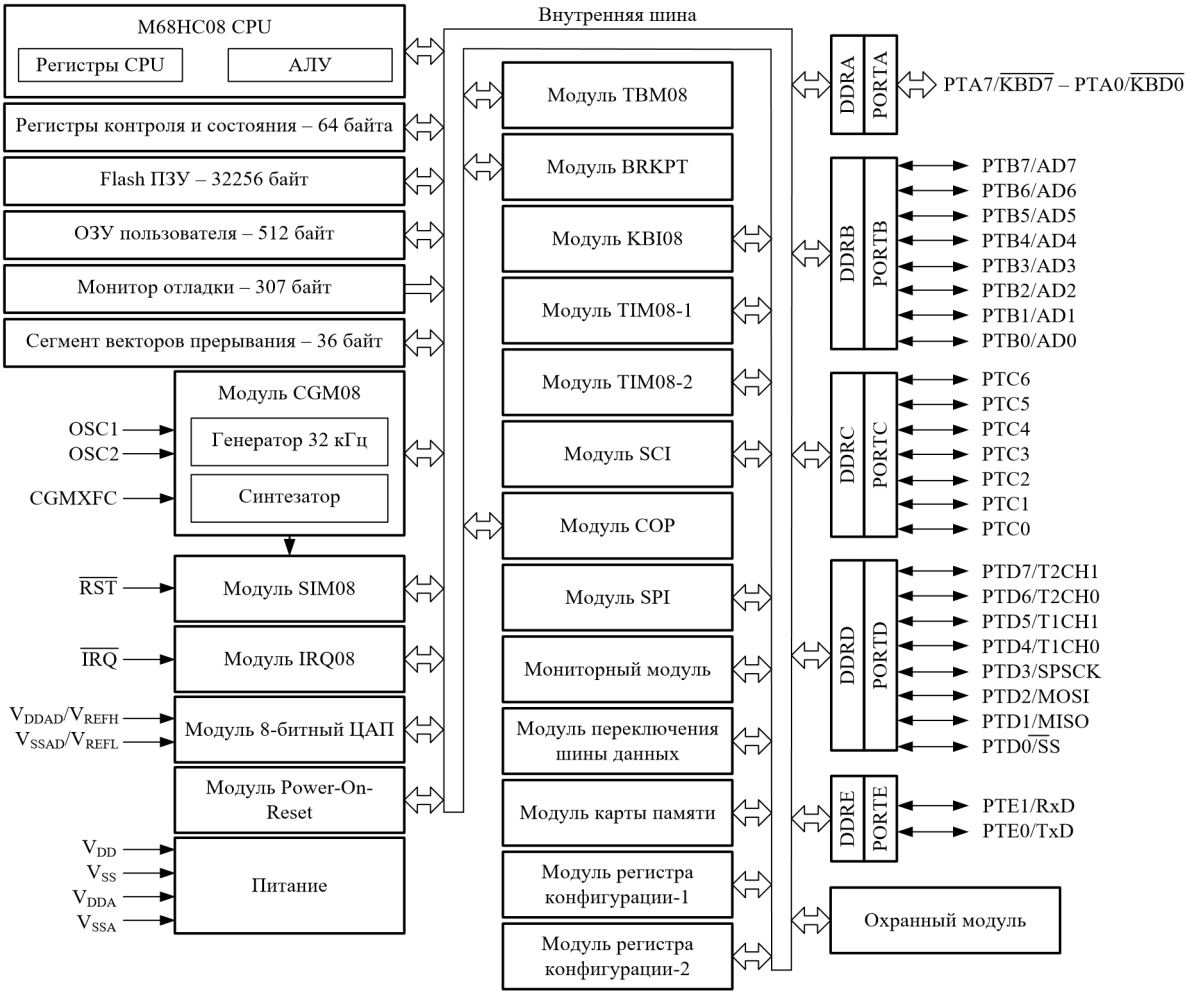


Рисунок 1.1 Структура микроконтроллера *MC*68*HC*908*GP*32

Процессор *CPU*08 выполняет обработку 8-ми разрядных операндов и реализует набор из 90 команд. Он содержит пять программно-доступных регистров: 8-ми разрядные аккумулятор *A* и регистр признаков *CCR,* 16-разрядные индексный регистр *H:X*, указатель стека *SP* и программный счетчик *PC*. Более подробно функционирование процессора и реализация набора команд рассматриваются в лаб. работах №№ 1–4.

В состав служебных модулей входят генератор тактовых импульсов *CGM*08, модуль системной интеграции *SIM*08, модуль контроля напряжения питания *LVI08*, модуль прерывания в контрольной точке *BREAK*08, модуль управления внешним прерыванием *IRQ*08, сторожевой таймер *COP*08, базовый таймер *TBM*08.

Модуль генератора импульсов *CGM*08 генерирует импульсные сигналы, на базе которых модуль системной интеграции *SIM*08 формирует тактовые

импульсы. Выходные сигналы модуля *CGM*08 определяют частоту тактовых импульсов для работы процессора и периферийных модулей.

Модуль системной интеграции *SIM*08 выполняет ряд функций, обеспечивающих совместную работу различных модулей микроконтроллера. Он работает совместно с другими служебными модулями: *CGM*08, *LVI*08, *IRQ*08, *BREAK*08, *COP*08, выполняя формирование тактовых импульсов, запуск микроконтроллера, организацию обслуживания прерываний.

Модуль прерывания в контрольной точке *BREAK*08 обеспечивает останов выполнения программы в заданной контрольной точке и используется в процессе отладки программного обеспечения.

Модуль управления внешним прерыванием *IRQ*08 принимает внешний запрос прерывания, поступающий на вход *IRQ*#, и обеспечивает различные варианты его обслуживания.

Сторожевой таймер *COP*08 осуществляет контроль выполнения текущей программы.

Модуль *LVI*08 вырабатывает сигнал перезапуска микроконтроллера при снижении его напряжения питания ниже порогового уровня.

Модуль базового таймера *TBM*08 обеспечивает периодическое формирование запросов прерывания.

Периферийные модули обеспечивают обмен данными и совместную работу микроконтроллера с другими устройствами, входящими в состав системы управления. Микроконтроллер *MC*68*HC*908*GP*32 содержит следующие периферийные модули:

– пять параллельных портов *A*, *B*, *C*, *D*, *E* для ввода-вывода данных;
– асинхронный последовательный порт *SCI*08;
– синхронный последовательный порт *SPI*08;
– модуль контроля клавиатуры *KBI*08;
– 8-разрядный аналого-цифровой преобразователь *ADC*08; – два таймерных модуля *TIM*08.

Двунаправленные порты *A*, *B*, *C*, *D*, *E* обеспечивают параллельный обмен данными с внешними устройствами. Порты *A, B* имеют по 8 линий ввода-вывода, порт *E* – 2 линии, а порты *C*, *D* – от 5 до 8 линий в зависимости от числа выводов корпуса, в котором смонтирован микроконтроллер.

Выводы параллельных портов *A*, *B*, *D*, *E* совмещены с выводами других периферийных модулей – *KBI*08, *ADC*08, *TIM*08-1, TIM08-2, SPI08, *SC*I08 (рис. 1.1). При работе вышеуказанных модулей соответствующие выводы параллельных портов служат для передачи сигналов, необходимых для функционирования модуля, и не могут использоваться для параллельного ввода-вывода данных.

Последовательные порты *SCI*08, *SPI*08 реализуют соответственно последовательный асинхронный и синхронный обмен данными между микроконтроллером и внешними устройствами.

Таймерный модуль *TIM*08 выполняет широкий набор функций, включая фиксацию времени поступления входных сигналов, выдачу выходных сигналов в заданный момент времени, формирование последовательности импульсов заданной частоты и длительности.

Модуль аналого-цифрового преобразования *ADC*08 производит преобразование значения потенциала, поступающего на один из 8 аналоговых входов, в 8-разрядное двоичное число.

Модуль контроля клавиатуры *KBI*08 обеспечивает формирование запроса прерывания при поступлении сигнала на определенные входы параллельных портов, которые обычно используются для подключения клавиатуры.

Микроконтроллеры семейства 68*HC*08/908 адресуют 64 Кбайт внутренней памяти (адреса $0000-*FFFF)*. Распределение адресного пространства задается картой памяти, вид которой определяется объемом внутренней памяти и набором периферийных устройств, входящим в состав данной модели микроконтроллера. На рис. 1.2 приведено распределение адресного пространства памяти для микроконтроллеров *MC*68*HC*908*GP*32.

В адресном пространстве имеется ряд неиспользуемых позиций, которые соответствуют ячейкам памяти, отсутствующим в данной модели микроконтроллеров. При обращении к этим адресам производится перезапуск микроконтроллера.

\$0000 \$003*F*	Регистры периферийных и служебных модулей (64 байт)
\$0040 \$023*F*	ОЗУ данных (512 байт)
\$0080 \$7*FFF*	Не используется (32 192 байт)
\$8000 \$*FDFF*	*Flash*-память (32 256 байт)
\$*FE*00	Регистр *SBSR* (модуль *BREAK*08)
\$*FE*01	Регистр *SRSR* (указывает причину запуска)
\$*FE*02	Резервировано
\$*FE*03	Регистр *SBFCR* (модуль *BREAK*08)
\$*FE*04	Регистр *INT*1 (запросы прерывания)
\$*FE*05	Регистр *INT*2 (запросы прерывания)
\$*FE*06	Регистр *INT*3 (запросы прерывания)
\$*FE*07	Резервировано
\$*FE*08	Регистр *FLCR* (управление *Flash*-памятью)
\$*FE*09	Регистр *BRKh* (модуль *BREAK*08)
\$*FE*0*A*	Регистр *BRKl* (модуль *BREAK*08)
\$*FE*0*B*	Регистр *BRKSCR* (модуль *BREAK*08)
\$*FE*0*C*	Регистр *LVISR* (модуль *LVI*08)
\$*FE*0*D*	Не используются (19 байт)
\$*FE*1*F*	ПЗУ – монитор отладки (307 байт)
\$*FE*20	Не используются (43 байт)
\$*FE*52	Регистр *FLBPR* (управление *Flash*-памятью)
\$*FE*53	Не используются (93 байт)
\$*FF*7*D*	Вектора запуска и прерываний (36 байт)
\$*FF*7*E*	
\$*FF*7*F*	
\$*FFDB*	
\$*FFDC*	
\$*FFFF*	

Рисунок 1.2 Распределение адресного пространства для микроконтроллера *MC*68*HC*908*GP*32

Младшие 64 позиции адресного пространства (адреса \$000-\$003*F*) занимают регистры служебных и периферийных модулей (табл. 1). Отметим,

что 16-разрядные регистры таймерных модулей *TCNT, TMOD, TCHx* занимают по две позиции адресного пространства: младший байт с суффиксом *l*, старший байт с суффиксом *h*.

Таблица 1.1

Адреса регистров периферийных модулей

Адрес	**Регистр**	**Назначение**
\$0000	*PTA*	Регистры данных портов *A, B, C, D*
\$0001	*PTB*	
\$0002	*PTC*	
\$0003	*PTD*	
\$0004	*DDRA*	Регистры направления передачи портов *A, B, C, D*
\$0005	*DDRB*	
\$0006	*DDRC*	
\$0007	*DDRD*	
\$0008	*PTE*	Регистр данных порта *E*
\$0009-0*B*		Не используются
\$000*C*	*DDRE*	Регистр направления передачи порта *E*
\$000*D*	*PTAPUE*	Регистры управления подтягивающими резисторами портов *A, C, D*
\$000*E*	*PTCPUE*	
\$000*F*	*PTDPUE*	
\$0010	*SPCR*	Регистры синхронного порта *SPI*08
\$0011	*SPSCR*	
\$0012	*SPDR*	
\$0013	*SCC*1	Регистры асинхронного порта *SCI*08
\$0014	*SCC*2	
\$0015	*SCC*3	
\$0016	*SCS*1	

$0017	*SCS*2	
$0018	*SCDR*	
$0019	*SCBR*	
$001*A*	*INTKBSCR*	Регистры модуля *KBI*08
$001*B*	*INTKBIER*	
$001*C*	*TBCR*	Регистр базового таймера *TBM*08
$001*D*	*INTSCR*	Регистр прерываний
$001*E*	*CONFIG*2	Регистры конфигурации
$001*F*	*CONFIG*1	
$0020	*T*1*SC*	Регистры таймерного модуля *TIM*08-1
$0021-22	*T*1*CNTh-l*	
$0023-24	*T*1*MODh-l*	
$0025	*T*1*SC0*	
$0026-27	*T*1*CH0h-l*	

Окончание таблицы 1.1

$0028	*T*1*SC*1	
$0029-2*A*	*T*1*CH1h-l*	
$002*B*	*T*2*SC*	Регистры таймерного модуля *TIM*08-2
$002*C*-2*D*	*T*2*CNTh-l*	
$002*E*-2*F*	*T*2*MODh-l*	
$0030	*T*2*SC0*	
$0031-32	*T*2*CH*0*h-l*	
$0033	*T*2*SC*1	
$0034-35	*T*2*CH*1*h-l*	
$0036	*PCTL*	Регистры модуля генератора импульсов *CGM*08
$0037	*PBWC*	
$0038-39	*PMSh-l*	
$003*A*	*PMRS*	
$003*B*	*PMDS*	
$003*C*	*ADSCR*	Регистры модуля *ADC*08
$003*D*	*ADR*	
$003*E*	*ADCLK*	

В адресном пространстве ОЗУ располагаются ячейки стека, которые адресуются с помощью указателя стека *SP*. При установке микроконтроллера в начальное состояние (запуске) содержимое *SP* принимает значение $00*FF*, адресуя ячейку ОЗУ с данным адресом. В процессе выполнения программы можно установить любое значение указателя стека с помощью команды *TXS* , которая загружает в *SP* содержимое индексного регистра *H:X*, уменьшенное на 1. После записи байта в стек содержимое *SP* уменьшается на 1, адресуя следующую незаполненную ячейку стека. Таким образом, стек заполняется в направлении уменьшения адресов. Адрес вершины стека (последней заполненной ячейки стека) можно загрузить в регистр *H:X* с помощью команды *TSX*.

Микроконтроллер *MC*68*HC*908*GP*32 имеет внутреннюю *Flash*-память, содержимое которой может стираться и записываться при работе в режиме отладки или в процессе выполнения прикладной программы. Допускается до 10000 циклов стирания-программирования, время хранения информации составляет более 10 лет. Необходимое для программирования повышенное напряжение обеспечивается внутренним преобразователем, поэтому не требуется подключение внешнего источника. Специальный механизм защиты позволяет предотвратить случайное стирание содержимого *Flash*-памяти. Наличие байтов секретности позволяет предотвратить несанкционированное считывание информации.

На кристалле микроконтроллера содержится 512 байт статической оперативной памяти, ячейки которой имеют адреса в диапазоне *$*0040–*$*023*F*. Обычно ОЗУ используется для хранения переменных и реализации стека.

Часть адресного пространства занята ячейками служебного ПЗУ, в котором содержится программа-монитор, которая реализует необходимые процедуры при работе микроконтроллера в режиме отладки, обеспечивая возможность контроля его внутреннего состояния. Это масочнопрограммируемое ПЗУ, содержимое которого записывается в процессе изготовления микроконтроллера.

В старших позициях адресного пространства располагаются вектора начального запуска и прерываний.

Тактовые импульсы с частотой *Ft* формируются модулем системной интеграции *SIM*08 на базе импульсных сигналов, которые генерируются модулем *CGM*08. В микроконтроллерах *MC*68*HC*908*GP*32 этот модуль обеспечивает умножение частоты в сотни раз, что позволяет подключать в качестве частотно-задающего элемента дешевые и стабильные кварцевые кристаллы с резонансной частотой *Fq* = 32,768 кГц, широко применяемые в часовой промышленности («часовые» кварцы). Для получения тактовых сигналов заданной частоты *Ft* необходимо выполнить программирование модуля *CGM08*. Однако в лабораторном стенде этого не требуется, т.к. микроконтроллер работает в режиме отладки с фиксированной тактовой частотой *Ft* = 2,4576 МГц.

Запуск микроконтроллера выполняется с помощью модуля системной интеграции *SIM*08. Он автоматически производится в следующих случаях:

– включение напряжения питания *V*п;
– поступление внешнего сигнала сброса *RST*# = 0;
– поступление сигнала сброса от сторожевого таймера *COP*08;
– выборка неправильного кода команды;
– обращение к ячейке памяти, которая отсутствует в адресном пространстве данной модели микроконтроллера;
– поступление от модуля *LVI*08 сигнала о недопустимом снижении напряжения *V*п.

В процессе запуска микроконтроллера выполняются следующие действия:

– запускается ГТИ, который формирует тактовые импульсы с частотой *Ft* = *Fq*, задаваемой кварцевым резонатором;
– в программный счетчик *PC* поступают два байта (старший байт *PCh*, младший байт *PCl*) из ячеек памяти с адресами *$FFFE-FF* (см. рис.1.2), которые задают адрес первой команды, выполняемой микроконтроллером после запуска;

– в указатель стека *SP* заносится число *$FF*, адресующее начальную ячейку стека;

– в регистре признаков *CCR* устанавливается значение маски прерываний *I* = 1, которое запрещает обслуживание любых аппаратных прерываний;

– устанавливается необходимое начальное состояние регистров периферийных модулей.

Микросхема микроконтроллера *MC*68*HC*908*GP*32

При проектировании микропроцессорных устройств на основе микроконтроллеров существенную роль играет понимание назначения выводов микроконтроллера и их использования при схемотехническом проектировании.

Основным управляющим элементом, разрабатываемого МКУ является микросхема микроконтроллера МК *MC*68*HC*908*GP*32, цоколевка которой представлена на рис.1.3 Ниже перечислены назначения её выводов.

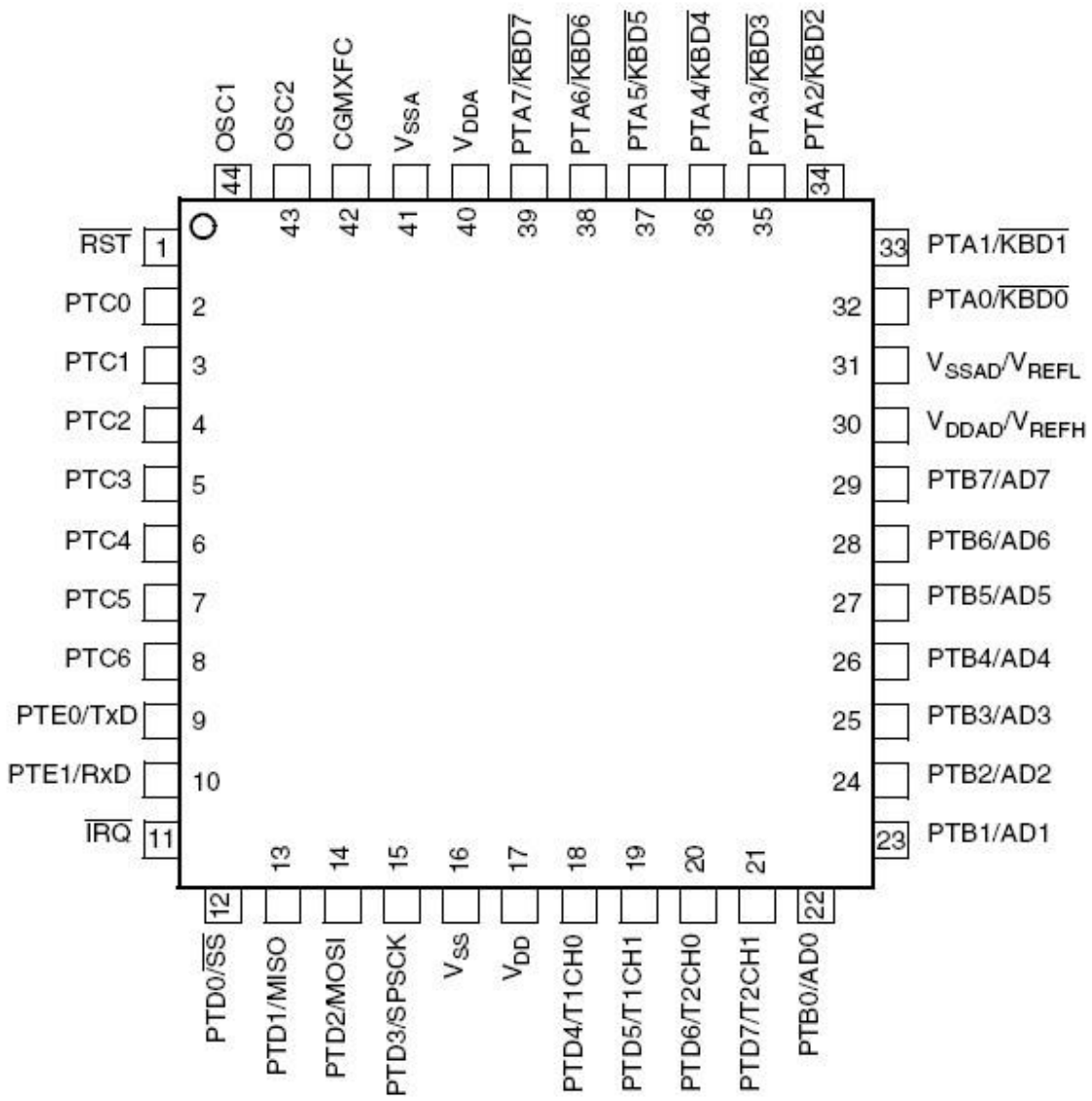


Рисунок 1.3 Вид корпуса и обозначение выводов микроконтроллера *MC*68*HC*908*GP*32

Порт *A* (*PTA*7/*KBD*7…*PTA*0/*KBD*0) представляет собой специальный 8-разрядный порт, который использует все выводы совместно с модулем клавиатурного прерывания (*KBI*). Каждый вывод порта имеет подключенный к источнику питания резистор, который может программно подключаться к выводу, когда он конфигурирован как входной. Все выводы порта могут использоваться как драйверы на ток 10 мА. Регистр данных *PTA* порта *A* содержит триггер-защелку на всех восьми выводах порта.

Порт *B* (*PTB*7/*AD*7…*PTB*0/*AD*0) представляет собой специальный 8разрядный порт, который использует все выводы совместно с модулем аналого-цифрового преобразователя. Как и в других портах, все выводы могут использоваться в качестве драйверов на ток 10 мА. Регистр данных PTB содержит триггер-защелку на восьми выводах порта.

Порт *C* (*PTC*6…*PTC*0) представляет собой специальный 7-разрядный порт, в котором выводы от 0 до 4 служат в качестве мощных драйверов, что дает им возможность непосредственно управлять светодиодами. Каждый вывод порта *C* может быть программно связан с встроенными резисторами, когда этот вывод конфигурирован как входной. Регистр данных *PTC* порта *C* содержит триггер-защелку на восемь выводов порта.

Порт *D* (*PTD*7/*T*2*CH*1, *PTD*6/*T*2*CH*0, *PTD*5/*T*1*CH*1, *PTD*4/*T*2*CH*0, *PTD*3/*SPSCK*, *PTD*2/*MOSI*, *PTD*1/*MISO*, *PTD*0/*SS*) представляет собой специальный 8-разрядный порт, в котором выводы от 4 до 7 используются совместно с модулем таймера/счетчика *TIM*08, а выводы от 0 до 3 – совместно с модулем *SPI*, для организации синхронной последовательной передачи данных. Возможности использования выводов в качестве драйверов такие же, как и у порта B. Каждый вывод порта *D* может быть программно связан с встроенным резистором, когда этот вывод конфигурирован как входной. Регистр данных *PTD* порта *D* содержит триггер-защелку на всех восьми выводах порта.

Порт *E* (*PTE*1/*RxD*, *PTE*0/*TxD*) представляет собой 2-разрядный цифровой порт без встроенных резисторов, используется для организации асинхронной передачи данных *SCI*. Возможности драйвера такие же, как у порта *A*, *B* и *D*. Регистр данных *PTE* порта *E* содержит триггер-защелку на обоих выводах порта.

***V*SS** – земля.

***V*DD** – напряжение питания 5 В.

***V*SSAD/*V*REFL** – земля для аналого-цифрового преобразователя модуля *ADC*.

***V*DDAD/*V*REFH** – напряжение питания для аналого-цифрового преобразователя модуля *ADC*.земля для аналого-цифрового преобразователя модуля *ADC*.

***V*SSA** – земля для модуля тактового генератора (*CGM*).

***V*DDA** – напряжение питания для модуля тактового генератора (*CGM*).

***OSC*1, *OSC*2** – вход и выход встроенного в микросхему инвертирующего усилителя. Используются для подключения навесных элементов кварцевого генератора. Внешний генератор *OSC* предназначен для работы при оптимальной частоте внешнего кварцевого генератора 32,768 кГц и функционирует только в частотном диапазоне от 30 до 100 кГц.

CGMXFC используется для подключения внешнего фильтра при использовании внешнего кварцевого генератора на частоту 32,768 к Гц.

RST сброс микроконтроллера. Сброс происходит в том случае если на нем низкий логический уровень.

IRQ внешнее прерывание.

Микроконтроллеры 68*HC*08/908 принадлежат к семейству интегральных схем *HC*, выпускаемых компанией *Motorola* по технологии «*hight*-speed *CMOS*». *CMOS*, или в русскоязычной терминологии КМОП, – это технология производства цифровых интегральных схем на основе Комплементарных полевых транзисторов со структурой «Металл – Окисел – Полупроводник». Семейство *HC* объединяет цифровые ИС различной степени интеграции: от простых логических элементов, счетчиков, дешифраторов до микроконтроллеров с архитектурой различной сложности. Все элементы, принадлежащие к семейству *HC*, электрически совместимы, поэтому сопряжение микроконтроллеров 68*HC*08/908 с другими элементами семейства *HC* не вызывает затруднений.

Электрические и динамические характеристики микроконтроллера:

– U_{OH} = 4,2 В;
– U_{OL} = 0,4 В;
– I_{OH} = -0,8 мА;
– I_{OL} = 1,6 мА;
– U_{IH} = 3,5 В;
– U_{IL} = 1,0 В;
– I_{IH} = 10 мкА; – I_{IL} = -10 мкА;

где U_{OH} - минимальное выходное напряжение логической 1; U_{OL} - максимально выходное напряжение логического 0; I_{OH} - максимальный выходной ток логической 1; I_{OL} - максимальный выходной ток логического 0; U_{IH} - минимальное входное напряжение логической 1; U_{IL} - максимальное входное напряжение логического 0; I_{IH} - максимальный входной ток логической 1; I_{IL} - максимальный входной ток логического 0;

Работа микроконтроллера задаётся внешним кварцевым резонатором, схема подключения которого приведена на рис. 1.4.

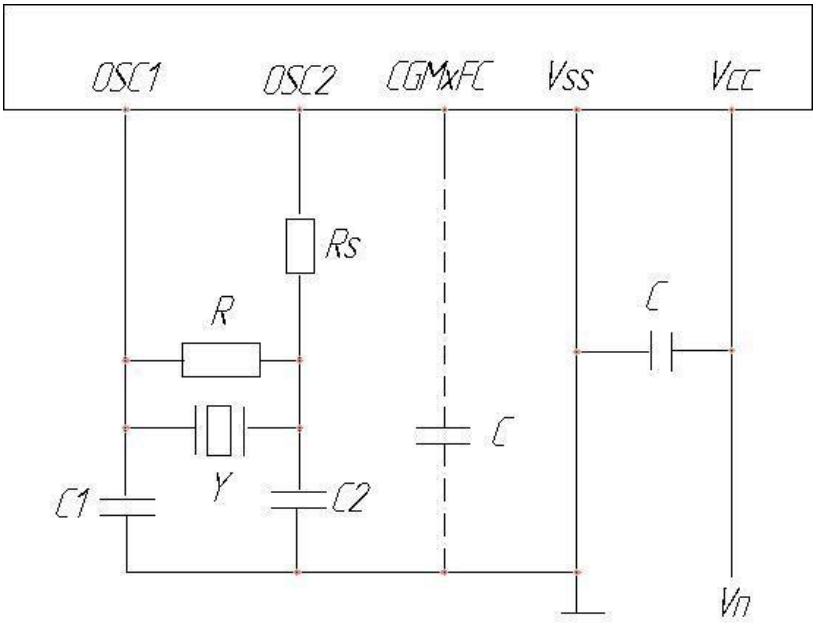


Рисунок 1.4 Схема подключения кварцевого резонатора

В микроконтроллерах серии *GP* используется вариант модуля CGM08, который обеспечивает умножение частоты в сотни раз, что позволяет подключать в качестве частотно-задающего элемента дешевые и стабильные кварцевые кристаллы с резонансной частотой F_q = 32,768 кГц, широко применяемые в часовой промышленности («часовые» кварцы). В паре с этим резонатором работают конденсаторы *C*1 и *C*2 ёмкостью 3 пФ.

Программирование на языке ассемблера 8–ми разрядного микроконтроллера *MC* 68*HC*908*GP*32

В состав современных профессиональных средств написания и отладки программ для микроконтроллеров обычно входят эмуляторы процессоров или отладочные платы, текстовый редактор, компиляторы языка высокого уровня (чаще всего «Си») и ассемблера, редактор связей (компоновщик) и загрузчик программы в отладочную плату. Все программы обычно объединены интегрированной средой разработки программного проекта, которая позволяет поддерживать один или несколько программных проектов.

К достоинствам программирования на языке ассемблера относятся минимальный объем памяти и быстродействие выполнения программ.

К недостаткам - большая трудоемкость составления программ, большая вероятность ошибок и трудность их обнаружения, а также зависимость от типа применяемого МП.

Язык ассемблера является символическим аналогом машинного языка. По этой причине программа, написанная на ассемблере, отражает все особенности архитектуры МП: организацию памяти, способы адресации операндов, правила использования регистров и т.д. Из-за необходимости учёта подобных особенностей ассемблер уникален для каждого типа МП.

Язык ассемблера наиболее широко распространен для программирования МП. В языке ассемблера каждая машинная команда МП обозначается мнемоническим символом, представляющим собой сочетание трех или четырех букв, являющихся первыми буквами полной записи наименования этих команд на английском языке.

Очевидно, что мнемоническое (символическое) кодирование названия и содержания команд легче запоминается, чем ничего не говорящее сочетание нулей и единиц, представляющее собой двоичное кодирование. Мнемокод - последовательность букв, заменяющая полное

слово или фразу, удобную для запоминания. АССЕМБЛЕР - это язык мнемокодов.

Трансляция - замена символов и синтаксиса исходного языка программирования символами и синтаксисом другого языка с сохранением содержания переводимых выражений.

Ассемблер – служебная программа, преобразующая исходную программу, написанную на языке мнемокодов и символических адресов в программу в двоичных кодах. При этом создается объектная программа - программа на машинном языке, получаемая в результате трансляции исходной программы.

Объектный код - команды программы, представленные на машинном языке. Одной команде языка ассемблера соответствует одна машинная команда.

Трансляция с языка ассемблера в машинный код называется ассемблированием.

Язык ассемблера включает два типа операторов:

– команды МП (система команд);

–директивы ассемблера или псевдокоманды ассемблера.

Создание программ на языке ассемблера 8–ми разрядного микроконтроллера *MC 68HC908GP32* выполняется в интегрированной среде разработки прикладного программного обеспечения для встраиваемых микропроцессорных систем *Win IDE ICS08.* Интегрированная среда разработки *Win IDE ICS08* ориентирована на создание законченных прикладных программ для встраиваемых приложений на языке ассемблера. Интегрированная среда разработки *Win IDE ICS08* включает в себя:

– программу управляющей оболочки;

– редактор текста;

– компилятор с языка ассемблера для МК с процессорным ядром *CPU08*;

– линковщик-загрузчик;

– симулятор;

– отладчик в режиме внутрисхемной симуляции (*ICS-Circuit Simulator*); – отладчик реального времени (*Debugger*);

– программатор флэш-памяти МК (*Programmer*).

Пакет интегрированной среды разработки *Win IDE ICS08* имеет достаточно большое число модификаций: *ICS08GPGTZ, ICS08GPZ, ICS08JLZ, ICS08MRZ, ICS08RXZ* и т.д. Каждая такая модификация предназначена для работы с несколькими МК одной серии семейства *HC08*. Например, пакет *ICS08GPZ* обслуживает МК *MC68HC908GP32*.

Операнд, способы адресации операндов

Что же необходимо знать, чтобы научиться писать программы для МПС? Основная функция любого процессора, ради которой он и создается, — это выполнение команд.

Система команд, выполняемых процессором, представляет собой нечто подобное таблице истинности логических элементов или таблице режимов работы более сложных логических микросхем. То есть она включает весь набор операций, выполняемых конкретным процессором, и определяет логику работы процессора и его реакцию на те или иные комбинации внешних событий.

Операнд **–** это данные (коды данных), над которыми выполняется команда (операция) микропроцессора.

Написание программ для микропроцессорной системы — важнейший и часто наиболее трудоемкий этап разработки такой системы. А для создания эффективных программ необходимо иметь хотя бы самое общее представление о системе команд используемого процессора. Систему команд для микроконтроллера *MC68HC908GP32* нами была рассмотрена на предыдущей лекции. Самые компактные и быстрые программы и подпрограммы создаются на **языке ассемблера**, использование которого без знания системы команд абсолютно невозможно, ведь язык ассемблера представляет собой символьную запись цифровых кодов машинного языка, кодов команд процессора. Конечно, для разработки программного обеспечения существуют всевозможные программные средства. Пользоваться ими обычно можно и без знания системы команд процессора. Чаще всего применяются языки программирования высокого уровня, такие как Паскаль и Си. Однако знание системы команд и языка Ассемблер позволяет в несколько раз повысить эффективность некоторых наиболее важных частей программного обеспечения любой микропроцессорной системы — от микроконтроллера до персонального компьютера.

Именно поэтому мы рассмотрим основные типы команд – систему команд, имеющиеся у большинства процессоров, и особенности их применения.

Каждая команда, выбираемая (читаемая) из памяти процессором, определяет алгоритм поведения процессора на ближайшие несколько тактов. Код команды говорит о том, какую операцию предстоит выполнить процессору и с какими ***операндами*** (то есть кодами данных), где взять исходную информацию для выполнения команды и куда поместить результат (если необходимо). Код команды может занимать от одного до нескольких байт, причем процессор узнает о том, сколько байт команды ему надо читать, из первого прочитанного им байта или слова. В процессоре код команды

расшифровывается и преобразуется в набор микроопераций, выполняемых отдельными узлами процессора. Но разработчику микропроцессорных систем это знание не слишком важно, ему важен только результат выполнения той или иной команды.

Большая часть команд процессора работает с кодами данных (*операндами*). Одни команды требуют входных *операндов* (одного или двух), другие выдают выходные *операнды* (чаще один *операнд*). Входные *операнды* называются еще операндами-источниками, а выходные называются операндами-приемниками. Все эти коды *операндов* (входные и выходные) должны где-то располагаться. Они могут находиться во внутренних регистрах процессора (наиболее удобный и быстрый вариант). Они могут располагаться в системной памяти (самый распространенный вариант). Наконец, они могут находиться в устройствах ввода/вывода (наиболее редкий случай). Определение места положения *операндов* производится кодом команды. Причем существуют разные методы, с помощью которых код команды может определить, откуда брать входной *операнд* и куда помещать выходной *операнд*. Эти методы называются ***методами или способами адресации***. Эффективность выбранных *методов адресации* во многом определяет эффективность работы всего процессора в целом.

Количество *методов адресации* в различных процессорах может быть от 4 до 16. Рассмотрим способы *адресации операндов*, используемых на языке ассемблера 8–ми разрядного микроконтроллера *MC 68HC908GP32* семейства *Motorola*.

Для выборки операндов из памяти МК *MC 68HC908GP32* используют следующие способы адресации:

- неявная (*INH – Inherent*);
- непосредственная (*IMM – Immediate*);
- прямая (*DIR – Direct*);
- прямая расширенная (*EXT – Extended*);
- индексная (*IX – Indexed*);
- индексная со смещением 1 байт (*IX1 – Indexed, 8 bit offset*);
- индексная со смещением 2 байта (*IX2 – Indexed, 16 bit offset*);
- индексная с последующим инкрементированием указателя адреса (*IX1+ – Indexed with post incrementer*);
- индексная со смещением 1 байт с последующим инкрементированием указателя адреса (IX+ – Indexed, 8 bit offset with post incrementer);
- индексная по указателю стека со смещением 1 байт (*SP1 – Stack*

pointer, 8 bit offset);

– индексная по указателю стека со смещением 2 байта (*SP2 — Stack pointer, 16 bit offset*);

– относительная (*REL – Relative*).

Область стека. Указатель стека

Указатель стека (*Stack Pointer* - (*SP*)). представляет 16-разрядный регистр, который содержит адрес последнего помещенного в стек байта. Указатель стека декрементируется при каждом помещении в стек и инкрементируется при каждом извлечении из него.

Стек – это область памяти, специально выделяемая для временного хранения данных программы. Отличительной особенностью стека является особая организация обращения к нему со стороны МП.

Запись и чтение данных в стеке осуществляется в соответствии с принципом LIFO (*Last In First Out*) – «последним пришел, первым ушел». Таким образом, информация в стеке размещается в строгой последовательности – ячейка памяти, заполненная последней, считывается первой, а ячейка памяти заполненная первой извлекается последней.

В ячейки стека информация заносится последовательно и извлекается в порядке обратном порядку занесения. Таким образом, стек функционирует как память с последовательным доступом. По мере записи данных в стек он растёт в сторону младших адресов. Эта особенность заложена в алгоритм команд работы со стеком.

Стек предназначен для обработки прерываний и программ. При записи слова данных в ячейку стека значение адреса в указателе стека уменьшается на единицу, а при считывании данных увеличивается на единицу.

Стек выполняется на некоторой выделенной области ОЗУ. В этом случае стек представляет память с последовательным доступом. Обращение и адресация к стеку производится через регистр указателя стека.

Стек на внутренних регистрах МП более быстродействующий, но из-за малого числа регистров, имеющихся в МП, не обеспечивается большая глубина вложения данных. Поэтому стеки большинства МП размещаются в памяти.

Вершиной стека называется адрес его последней загруженной ячейки памяти. Таким образом, указатель стека всегда содержит адрес его вершины. Начало стека в ОЗУ (его дно) определяется программистом путём записи в регистр SP адреса первой ячейки памяти стека. Вершина стека подвижна и её расположение определяется объёмом данных, загруженных в стек. Стек не

имеет ограничений, за исключением тех, которые обусловлены наличием других программ в ОЗУ.

Стек – это зарезервированная область оперативной памяти, используемая для сохранения адресов возврата из подпрограмм и программ обработки прерываний (***ISR***), а также для передачи данных в подпрограммы. Для вызова подпрограммы используются две ячейки памяти стека, для прерывания – пять ячеек. При записи данных в стек значение указателя стека уменьшается, при извлечении их из стека – увеличивается.

Указатель стека может также использоваться как индексный регистр с 8-разрядным или 16-разрядным смещением.

Для микроконтроллеров семейства *HC08* после сброса в указатель стека записывается значение $00FF, назначая область стека в нулевую страницу ОЗУ МК. Это же значение может быть также установлено программно командой ***RSP*** (сброс указателя стека). Нулевую страницу памяти ($0000–$00FF) в этих МК следует использовать для хранения промежуточных данных.

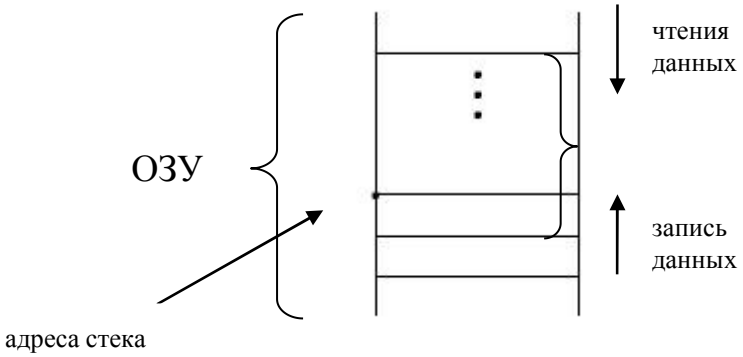
Память для *стека* или ***стек*** (*Stack*) — это часть оперативной памяти, предназначенная для временного хранения данных в режиме *LIFO (Last In — First Out)*.

Особенность *стека* по сравнению с другой оперативной памятью — это заданный и неизменяемый способ адресации.

При записи любого числа (кода) в *стек* число записывается по адресу, определяемому как содержимое *регистра* указателя *стека*(УС или *SP*), предварительно уменьшенное (декрементированное) на единицу .

При чтении из *стека* число читается из адреса, определяемого содержимым указателя *стека*, после чего это содержимое указателя *стека* увеличивается (инкрементируется) на единицу.

Работа со стековой памятью (стеком)



В регистре *SP* – указатель стека, начальный адрес стековой памяти. В *SP* всегда находится **вершина стека**.В МК неявно предусмотренный начальный адрес стека т.е. вершина стека $00FF-начальный адрес.

1) Для задания стандартного адреса $00FF используется команда ***_rsp_***; (SP)←$00FF.

В случае использования нестандартного адреса, используется команды пересылки.

2) Для сохранения текущих значений регистров при передаче управления подпрограмме необходимо выполнить загрузку содержимого регистров РОН: (*A, H, X*) в стековую память. Для этого в системе команд имеются специальные команды:

psha ; стек ← (A)
pshh ; стек ← (H)
pshx ; стек ← (X)

3) Восстановление содержимого регистров РОН выполняется командами:

pula ; (A) ← (стек)
pulh ; (H) ← (стек)
pulx ; (X) ← (стек)

Программно-логическая модель процессорного ядра

Центральный процессор *CPU08* выполняет действия над 8-разрядными операндами. Программно-логическая модель *CPU8* содержит шесть регистров (рис. 1.5):

– 8-разрядный аккумулятор *A*;

– два 8-разрядных индексных регистра *X* и *H*, которые могут быть объединены в 16-разрядный регистр *H:X*;

– указатель стека *SP*; – счетчик адреса *PC*;

– регистр признаков *CCR*.

Рисунок 1.5 Программно-логическая модель центрального процессора HC08

Аккумулятор A представляет собой 8-разрядный регистр, в котором хранятся операнды, результаты арифметических и логических операций.

Индексный регистр *X:H* состоит из двух регистров *H* и *X*, позволяющих осуществлять индексный доступ к внутренней памяти объемом 64 Кб. Регистр *H* содержит старший, а регистр *X* – младший байт адреса. Кроме того, в индексном регистре временно хранятся данные, тем самым осуществляется дополнительная разгрузка аккумулятора. Система команд *CPU8* позволяет загружать данные в индексные регистры и выполняет операции над ними как в двухбайтовом, так и в однобайтовом формате. В последнем случае обращение производится только к младшему байту регистра *X*. После сброса МК старший байт индексного регистра *H* принудительно устанавливается в 0, что обеспечивает выполнение программ, ранее написанных для МК семейства *HC05*. Сброс не изменяет состояние младшего байта индексного регистра *X*.

Указатель стека *SP* – 16-разрядный регистр хранит адрес ***вершины области стека***. Назначение приведено выше.

Счетчик адреса или программный счетчик *PC* – 16-разрядный регистр содержит адрес текущей команды либо адрес операнда, используемого в текущей команде. После выполнения команды содержимое

PC автоматически увеличивается до следующего адреса памяти. После сброса МК программный счетчик автоматически загружается вектором начального запуска, который записан в ячейках памяти ПЗУ с адресами $FFFE (старший байт) и $FFFF (младший байт). Вектор начального запуска является адресом начала прикладной программы управления.

Регистр признаков ССК – 8-разрядный регистр содержит информацию о состоянии ЦП, соответствующем последней выполненной команде. Каждый бит информации в этом регистре называется флагом. Регистр признаков в *CPU8* содержит шесть флагов: переноса *C*, нулевого результата *Z*, отрицательного результата *N*, переполнения *V*, дополнительного переноса *H*, глобальной маски прерывания *I*. Особенностью флагов нулевого результата *Z* и отрицательного результата *N* является установка их после операций пересылки.

C – флаг переноса (*Carry F1ag*) устанавливается в 1, если в результате операции сложения произошло переполнение аккумулятора или в результате операции вычитания произошел заем. Часть логических операций могут устанавливать либо сбрасывать флаг переноса *C*. Процессорное ядро *CPU8* имеет две специальные команды: *CLC* – очистить флаг переноса, *SEC* – установить флаг переноса. Перевод МК в состояние сброса не влияет на состояние флага переноса.

Z – флаг нулевого результата (*Zero F1ag*) устанавливается в 1, если результат арифметической, логической операции или операции пересылки данных равен 0. Сброс МК не влияет на значение бита нулевого результата.

N – флаг отрицательного результата (*Negative Flag*) устанавливается по значению разряда *D7* результата операции пересылки данных, арифметической или логической операции. Сброс МК не влияет на значение флага отрицательного результата. В микропроцессорной технике принят следующий стандарт: когда разряд *D7* является разрядом знака числа $N = D7 = 0$ – число положительное, $N = D7 = 1$ – число отрицательное.

H – флаг дополнительного переноса (*Halt-Carry F1ag*) устанавливается в 1 при возникновении переноса между разрядами *D3* и *D4* аккумулятора при выполнении операций сложения *ADD* и сложения с переносом *ADDC*. Этот признак необходим, если при обработке численных значений используют двоично-десятичный код (*BCD – binary coded decimal*). Команда десятичной коррекции *DAA* использует флаги *H* и *C* для определения подходящего поправочного коэффициента. Сброс МК не влияет на значение бита дополнительного переноса.

V – флаг переполнения устанавливается в 1, если при выполнении арифметической операции получается результат, выходящий за пределы диапазона чисел, которые представлены в дополнительном коде со знаком. Внутренний перенос из разряда *D6* в разряд *D7* называется переполнением.

Результат при этом становится отрицательным. Это событие должно быть распознано и, если необходимо, исправлено.

I – глобальная маска прерывания, ее установка в 1 запрещает все прерывания, кроме программного по команде *SWI*. Если запрос на прерывание появится, когда флаг *I* = 0, то центральный процессор сохранит в стеке содержимое счетчика адреса *PC*, аккумулятора *A*, младшего байта индексного регистра *X* и регистра признаков *CCR*, установит бит *I* в 1 и перейдет на выполнение подпрограммы прерывания. Если запрос на прерывание появится, когда флаг *I* = 1, то запрос на прерывание будет запомнен. Микроконтроллер начнет обрабатывать запрос на прерывание сразу, как только бит *I* будет очищен. Инструкция *RTI* (возврат из прерывания) сбрасывает бит *I* в 0 автоматически. В *CPU08* имеется две команды управляющие флагом *I: CLI* – очистить флаг маски, *SEI* – установить флаг маски. В состоянии сброса флаг *I* устанавливается в 1, что запрещает аппаратные прерывания. Разрешение аппаратных прерываний (установка *I* в 0) может быть выполнено только программной командой *CLI*.

Система команд микропроцессора МК *MC 68HC908GP32* приведена в Приложении 2.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КУРСОВОГО ПРОЕКТА

Обработка данных на микроконтроллере *Motorola MCHC*908*GP*32

К задачам, решаемых микроконтроллерными устройствами, относят: задачи организации ввода/вывода информации, приема, передачи данных; задачи обработки и преобразовании информации, а так же задачи организации временных функций, отсчетов временных интервалов.

Ниже приведены примеры программной реализации типовых алгоритмов обработки данных, рассматриваются примеры программ на языке ассемблера микроконтроллера

Часто при выполнении арифметико-логических операций необходимо определить минимальный (максимальный) элемент массива. Суть алгоритма определения минимального элемента заключается в следующем. Из массива данных выбираются два числа, одно из которых помещается в аккумулятор А, другое в специально отведенную ячейку. Эти числа сравниваются, и меньшее число записывается в аккумулятор. Далее в специально отведенную ячейку, которую использовали ранее, записывается следующий элемент массива. Процедура повторяется. В конечном счете в аккумуляторе будет храниться минимальный элемент массива, который помещается в отдельную ячейку. На

рис. 2.1 представлена блок-схема программы определения минимального элемента списка.

Команда ORG устанавливает начальный адрес программы в памяти МК. При ассемблировании первая команда будет размещаться начиная с адреса $8000. После этого в ячейку памяти ОЗУ с адресом $50, которая выбрана в качестве счетчика циклов, загружается число 5 (число элементов массива). В индексный регистр *H:X* загружается начальный адрес первого элемента массива. В нашем случае – это $82. Далее в аккумулятор записывается число, адрес которого содержится в *H:X*. Содержимое *H:X* увеличивается на единицу, и число, адрес которой указан в ячейке *H:X*, записывается в специально отведенную ячейку $40. После этого содержимое индексного регистра *H:X* уменьшается на 2. Содержимое аккумулятора А сравнивается с числом, содержащимся в ячейке $40. Выполняемое действие будет иметь вид: (А) – ($40). По результату сравнения устанавливаются признаки.

Содержимое А и ячейки памяти после операции не изменяются. Если после операции сравнения бит знака в регистре *CCR* установлен в 0, т. е. число положительное, или содержимое аккумулятора больше содержимого ячейки $40, то процессор переходит по метке *m*2, где в аккумулятор записывается $40. Иначе же процессор переходит по метке *m*3, где очищается ячейка $40, увеличивается содержимое индексного регистра *H:X* на единицу, уменьшается содержимое счетчика циклов на единицу.

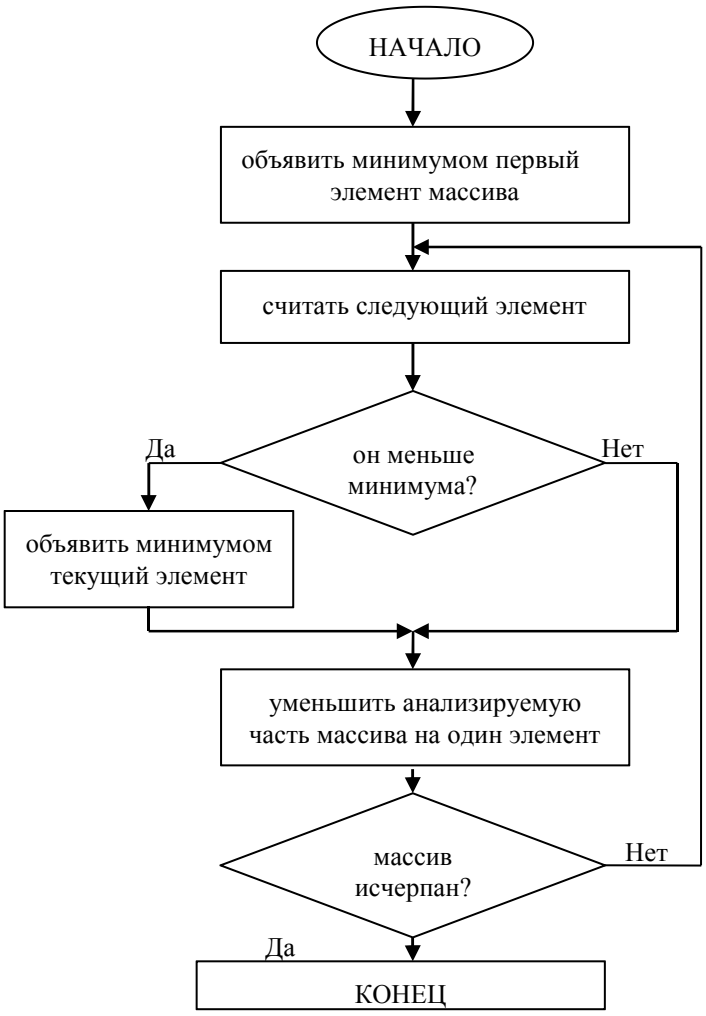


Рисунок 2.1 Блок-схема программы определения минимального элемента массива.

После этого проверяется счетчик циклов на «0». Если содержимое счетчика циклов не равно 0, то необходимо продолжить выполнение программы, перейдя по метке *m*1 (программа опять от метки *m*1 начнет заново выполняться). Если же содержимое счетчика циклов равно 0, то программа выполняется дальше, а именно: происходит запись содержимого аккумулятора в ячейку $51 –в ней и будет храниться минимальное число данного массива чисел.

Процедура *pr2* необходима для образования массива данных, *DB* – для определения байта данных. В поле операндов задаваемые данные записываются через запятую. С ячейки, имеющей адрес $82, резервируются 5 ячеек, в которые записываются соответствующие значения.

Суть программы определения максимального элемента заключается в следующем. Из массива данных выбираются два числа, одно из которых помещается в аккумулятор А, другое – в специально отведенную ячейку. Эти числа сравниваются, и большее число записывается в аккумулятор. Далее в

специально отведенную ячейку, которую использовали ранее, записывается следующий элемент массива. Процедура повторяется. В конечном счете в аккумуляторе будет храниться максимальный элемент массива, который помещается в отдельную ячейку.

Текст программы на языке ассемблера для микроконтроллера *MC*68*HC*908*GP*32:

```
pr1:    org     $8000           ;задание начального адреса программы
        mov     #5,$50          ;организация счетчика циклов с коэффициентом
                                ;пересчета, равным 5
        ldx     #$0082          ;загрузка в H:X начального адреса первого
                                ;элемента массива (H:X)←$82
        lda     ,x              ;запись в А числа, адрес которого находится в H:X
m1:     cmp     1,x             ;сравнение содержимого А с содержимым ячейки
                                ((H:X)+1).      (A) - ((H:X)+1)
        bpl     m2              ;перейти на m2, если N = 0
        lda     1,x
m2:     aix     #1              ;увеличить содержимое H:X на 1
        dec     $50             ;уменьшить содержимое $50 на 1
        bne     m1              ; перейти на m1,если ($50) = 0
        sta     $51             ; ($51)←(A)
        jmp     *
        org     $0082           ;образование массива данных
pr2:    db      $7,$4,$3,$2,$13,$10
        end
```

Ввод информации в микроконтроллерную систему

К задачам ввода относятся как задачи получения от оператора информации о требуемых режимах и настройках системы, так и задачи выделения информации, поступающей в виде сигналов (амплитудных, частотных и других) с датчиков объекта управления.

Часто для переключения каналов используются двоичные ключи. На рис. 2.2 показана схема подключения контакта двоичного датчика ко входу *PTD*0 порта ввода-вывода *PTD MC*68*HC*908*GP*32. Если в разряд *PTD*0 предварительно записать логическую 1, то при разомкнутом ключе К на входе *PTD*0 присутствует напряжение, соответствующее логической 1 (все выводы порта *PTD* внутри микросхемы через резистор подключены к напряжению +5 В), при замкнутом – логическому 0.

Блок-схема программы реализации контроля переключения ключа, подключенного к выводу *PTD*0 порта ввода–вывода *PTD*, представлена на рис. 2.3.

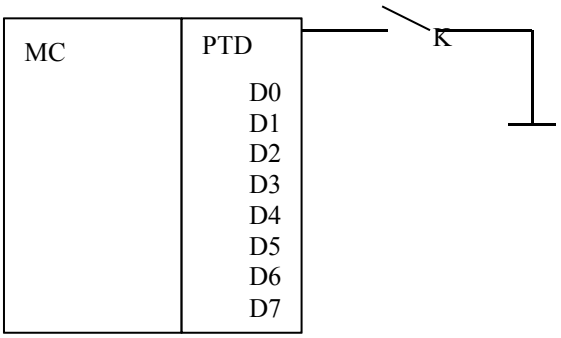


Рисунок 2.2 Подключение двоичного датчика к МКУ

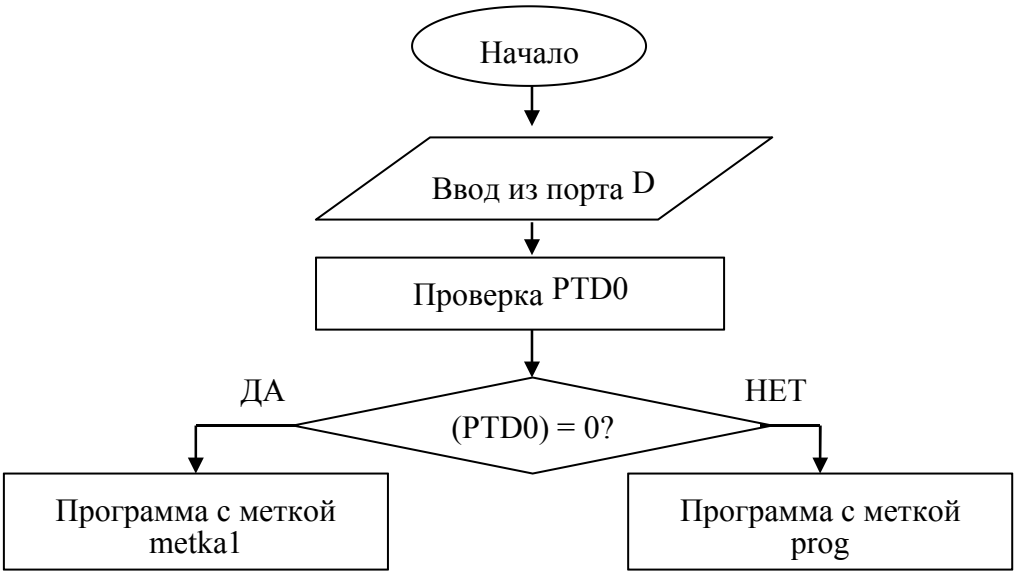


Рисунок 2.3 Блок-схема программы реализации контроля переключения ключа

Программа реализации контроля переключения ключа, подключенного к выводу *PTD0* порта ввода-вывода *PTD*, представлена ниже.

```
ramstart    equ    $0040        ;команды присвоения переменным символьных
romstart    equ    $8000        ;имен
vectorstart equ    $FFDC
PTD         equ    $0003
```

```
DDRD      equ     $0007
          org     Romstart        ;директива установки записи команд в блок Flash-
                                  ;памяти
          mov     $00,DDRD        ;конфигурирование вывода PTD0 порта D на ввод
                                  ;информации
main      bclr  0,PTD,metka1      ;переход по метке при наличии низкого
                                  потенциала на входе вывода порта D
prog:     ...         …           ;программа с меткой prog
metka 1:  ...         …           ;организация цикла опроса на наличие низкого
                                  ;потенциала
          end
```

Для того чтобы программа могла начать работу, должны быть определены через команды присвоения *EQU* все спецификации аппаратных средств, такие как имена регистров и области памяти. С помощью объявлений определяются все переменные.

RamStart EQU $0040

Здесь команда присвоения *EQU* назначает адрес первой ячейки памяти ОЗУ МК, которая будет использована Вами в программе. Обозначенная ячейка $0040 должна соответствовать карте памяти используемого МК.

RomStart EQU $8000

Назначается начальный адрес прикладной программы. Начиная с этого адреса, программа ассемблера разместит коды программы после ассемблирования. Выбранный адрес должен соответствовать карте памяти используемого МК.

VectorStart EQU $*FFDC*

Назначается адрес двух ячеек памяти, в которые необходимо поместить адрес начала прикладной программы. Этот адрес называют вектором начальной загрузки, поскольку при включении питания МК он автоматически загружается в программный счетчик *PC* центрального процессора.

Каждый параллельный порт ввода-вывода имеет регистр данных *PTx* и регистр направления *DDRx*, где *x = A, B, C, D* или *E*. В данной программе используется *PTD*. Поэтому необходимо присвоить абсолютным адресам символьные значения.

PTD EQU $0003
DDRD EQU $0007

Таким образом, в дальнейшем при необходимости обращения к этим ячейкам будут использоваться их символьные имена *PTD* или *DDRD*.

Команда *ORG* сообщает ассемблеру адрес начала оперативной памяти. При ассемблировании первая команда будет размещена в ячейки памяти $8000.

Команда *mov* $00, *DDRD* загружает в регистр направления *DDRD*, определяющий направление передачи данных порта *PTD* код 00_{16}$ (00000000_2). В результате этого все выводы порта *D* конфигурируются как входные.

main – начало главной программы, которая вызывается после аппаратного сброса МК или команды *RESET* отладчика (команды *Power-on* для некоторых отладчиков).

Команда «*brclr 0,PTD,metka1*» является командой условного перехода. Она обеспечивает переход на «*metka1*», при наличии логического нуля на нулевом бите *PTD*, т.е. переход по метке при наличии низкого потенциала на входе вывода порта *D*. В противном случае осуществляется выполнение программы *prog*.

В большинстве встраиваемых систем управления необходимо организовывать связь с оператором для указания параметров работы системы.

Контроллеры технологических объектов работают в реальном масштабе времени, и, следовательно, их функционирование должно определяться событиями, происходящими в объекте управления. Чаще всего событие в объекте управления фиксируется с использованием двоичных датчиков («да – нет»).

Пусть требуется по ходу выполнения управляющей программы приостановить продвижение по программе до тех пор, пока в результате процессов, происходящих в объекте управления, не замкнется контакт К некоторого двоичного датчика (рис. 2.3).

Программа будет практически аналогична рассмотренной выше программе. Из схемы алгоритма видно, что программа должна постоянна опрашивать значение сигнала на входе *PTD0* порта *D* до тех пор, пока оно не станет равным нулю (контакт датчика события замкнут), и в этом случае продолжить выполнение основной управляющей программы.

Блок схема данной программы представлена на рис. 2.4.

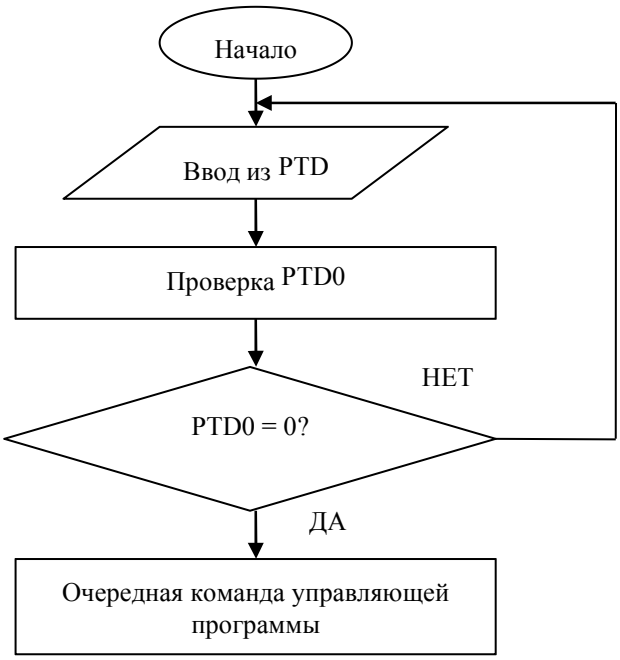


Рисунок 2.4 Блок-схема программы реализации контроля переключения ключа

Текст данной программы представлен ниже

```
ramstart    equ   $0040          ;команды присвоения переменным символьных
romstart    equ   $8000          ;имен
vectorstart equ   $FFDC
PTD         equ   $0003
DDRD        equ   $0007
            org   romStart       ;директива установки записи команд в блок Flash;памяти
            mov   $00, DDRD      ;конфигурирование вывода PTD0 порта D на ввод
                                 ;информации
main:       bclr  0,PTD,metka1   ;переход по метке при наличии низкого потенциала
                                 ;на входе вывода порта D
            bra   main           ;безусловный переход на метку main
metka 1:    ...                  ;организация цикла опроса на наличие низкого
                                 ;потенциала
            end
```

Необходимо рассмотреть задачу определения нажатой клавиши на 12-кнопочной клавиатуре (4 ряда по 3 кнопки), схема подключения которой к микроконтроллеру *MC*68*HC*908*GP*32 показана на рис. 2.5.

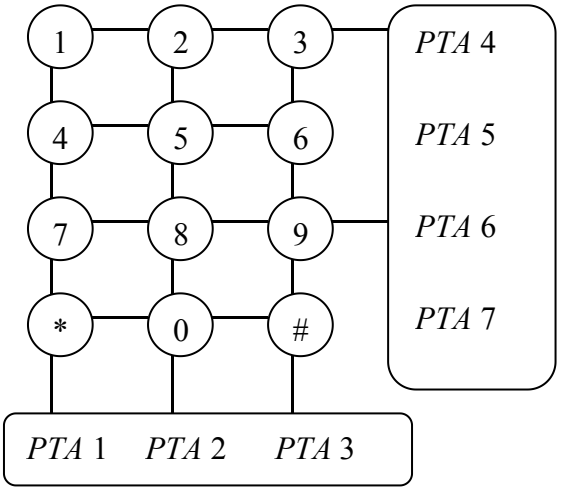


Рисунок 2.5 Схема подключения клавиатуры к выводам *MC*68*HC*908*GP*32

Если контроль состояния клавиатуры осуществляется по опросу, то необходимо сконфигурировать выводы управления строками *PTA4 – PTA7* на ввод данных, а выводы управления столбцами *PTA1 – PTA3* сконфигурировать на вывод данных. Тогда, если кнопки не нажаты, на выводах *PTA4 – PTA7* будет присутствовать высокий потенциал, что соответствует состоянию логической 1.

В процессе опроса клавиатуры на выводы управления столбцами *PTA1–PTA3* следует последовательно подавать низкий потенциал (логический 0) и считывать состояния клавиатуры с выводов управления строками *PTA4 – PTA7*. В случае нажатия клавиши на соответствующем выводе *PTAi* установится состояние логического 0, а на остальных выводах сохранится состояние логической 1.

На рис. 2.6 представлена блок-схема программы опроса 12-кнопочной клавиатуры. Эта программа необходима для определения той клавиши, которая в данный момент замкнута. Рассмотрим подробно выполнение этой программы.

Для того чтобы программа могла начать работу, должны быть определены через команды присвоения *EQU* все спецификации аппаратных средств, такие как имена регистров и области памяти. С помощью объявлений определяются все переменные. Команда *ORG* сообщает ассемблеру адрес начала оперативной памяти. Программа *MAIN* – начало главной программы, которая вызывается после аппаратного сброса МК или команды *RESET* отладчика (команды *Power-on* для некоторых отладчиков).

В регистр *DDRA* порта *PTA* загружается число в двоичном коде %00001110, тем самым первый, второй, третий биты *PTA* устанавливаются на вывод информации, а нулевой, четвертый, пятый, шестой и седьмой – на ввод информации.

Вначале производится сканирование первого столбца клавиатуры. В основной программе обнуляется первый бит *PTA* и устанавливаются высокие уровни на втором и третьем битах. Иначе говоря, на выводах *PTA* в данный момент времени будет следующий код %00001100. Затем через небольшой промежуток времени установившиеся значения на выводах *PTA* записываются в аккумулятор (А). Это значение сравнивается с числами $*ed* (%11101101), $*dd* (%11011101), $*bd* (%10111101), $7*d* (%01111101). Если число, которое хранится в аккумуляторе, равно числу $*ed*, то это означает, что замкнута кнопка на пересечении первой строки и первого столбца. В этом случае осуществляется переход на *metka1*, где в специально отведенную ячейку сохраняется число, соответствующее нажатой клавише. Далее микропроцессор переходит по метке *MAIN* и начинает заново сканировать клавиатуру. Если число, хранящееся в аккумуляторе, не равно $*ed*, то оно сравнивается со вторым числом. При равенстве этих чисел микропроцессор переходит по соответствующей метке, и выполняются операции, описанные выше. Если же содержимое аккумулятора не равно ни второму, ни третьему, ни четвертому числам, то это означает, что на первом столбце матричной клавиатуры нет нажатой клавиши.

Начинает сканироваться второй столбец. В этом случае обнуляется второй бит *PTA* и подаются логические 1 на первый и третий бит *PTA*. В данный момент времени на битах *PTA* будет следующий код: %00001010. Так же, как и при сканировании первого столбца, через некоторый небольшой промежуток времени в аккумулятор записываются установившиеся значения битов *PTA*. Происходит сравнение содержимого аккумулятора со следующими числами: $*eb* (%11101011), $*db* (%11011011), $*bb* (%10111011), $7*b* (%01111011). При равенстве содержимого аккумулятора с одним из этих чисел контроллер переходит по соответствующей метке, где сохраняет соответствующее число, после которого возвращается к метке *MAIN*. В случае же, если оно не равно ни одному из этих чисел, то начинается сканирование третьего столбца.

При сканировании третьего столбца вначале обнуляется третий бит *PTA*, а логические 1 подаются на первый и второй биты. Таким образом, на *PTA* будет следующий код: %00000110. Аналогично сканированию первого и третьего столбцов, через некоторый интервал времени в аккумулятор записываются установившиеся значения битов *PTA*, которое затем сравниваются с числами $*e*7 (%11100111), $*d*7 (%11010111), $*b*7

(%10110111), $77 (%01110111). Если содержимое аккумулятора равно одному из этих чисел, то процессор переходит по соответствующей метке, где сохраняет соответствующее число, после чего возвращается к метке *MAIN*. Если же содержимое аккумулятора не равно ни одному из этих чисел, то начинается сканирование первого столбца, будут повторяться все процедуры, описанные выше.

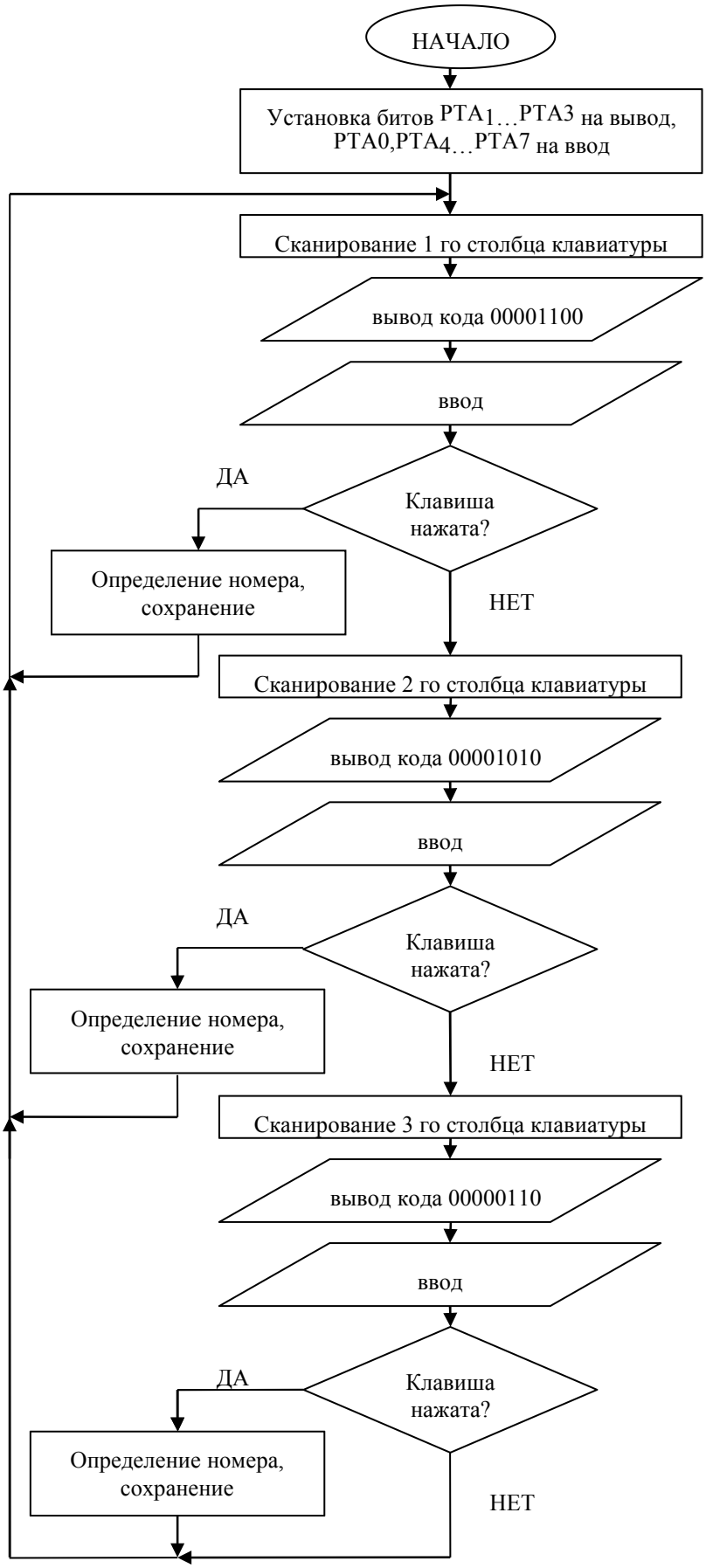

НАЧАЛО
Установка битов PTA1…PTA3 на вывод, PTA0,PTA4…PTA7 на ввод
Сканирование 1 го столбца клавиатуры
вывод кода 00001100
ввод
ДА
Клавиша нажата?
Определение номера, сохранение
НЕТ
Сканирование 2 го столбца клавиатуры
вывод кода 00001010
ввод
ДА
Клавиша нажата?
Определение номера, сохранение
НЕТ
Сканирование 3 го столбца клавиатуры
вывод кода 00000110
ввод
ДА
Клавиша нажата?
Определение номера, сохранение
НЕТ

Рисунок 2.6 Блок-схема программы опроса 12-ти кнопочной клавиатуры

Программа опроса 12-кнопочной клавиатуры представлена ниже

```
romstart:  equ    $8000              ;команды присвоения переменным
ramstart   equ    $0038              ;символьных имен
PTA:       equ    $0000
DDRA:      equ    $0004
PTAPUE     equ    $000D
:
           org    romstart           ;директива установки записи команд в блок
                                     ;Flash-;памяти
           mov    #%00001110,ddra
           mov    #%11110000,ptapue
main:      bclr   1,PTA              ;обнулить первый бит PTA
           bset   2,PTA              ;установить 1 на втором бите PTA
           bset   3,PTA              ;установить 1 на третьем бите PTA
           lda    PTA                ;сохранить в А установившееся значение
                                     ;битов PTA
           cbeqa  #$ed,metka1        ;сравнить содержимое А с константой,
           cbeqa  $dd,metka4         ;перейти по метке если они равны
           cbeqa  #$bd,metka7
           cbeqa  #$7d,metka10
           bset   1,PTA              ;установить 1 на первом бите PTA
           bclr   2,PTA              ;обнулить второй бит PTA
           bset   3,PTA              ;установить 1 на третьем бите PTA
           lda    PTA
           cbeqa  #$eb,metka2
           cbeqa  #$db,metka5
           cbeqa  #bb,metka8
           cbeqa  #$7b,metka11
           bset   1,PTA              ;установить 1 на первом бите PTA
           bset   2,PTA              ;установить 1 на втором бите PTA
```

```
        bclr    3,PTA           ;обнулить третий бит PTA
        lda     PTA
        cbeqa   #$e7,metka3
        cbeqa   #$d7,metka6
        cbeqa   #$b7,metka9
        cbeqa   #$77,metka12
        jmp     main
metka1: mov     #1,$0040        ;записать 1 в $0040
        jmp     main
metka2: mov     #2,$0040        ;записать 2 в $0040
        jmp     main
metka3: mov     #3,$0040        ;записать 3 в $0040
        jmp     main
metka4: mov     #4,$0040        ;записать 4 в $0040
        jmp     main
metka5: mov     #5,$0040        ;записать 5 в $0040
        jmp     main
metka6: mov     #6,$0040        ;записать 6 в $0040
        jmp     main
metka7: mov     #7,$0040        ;записать 7 в $0040
        jmp     main
metka8: mov     #8,$0040        ;записать 8 в $0040
        jmp     main
metka9: mov     #9,$0040        ; записать 9 в $0040
        jmp     main
metka10: mov    #10,$0040       ;записать 10 в $0040
        jmp     main
metka11: mov    #11,$0040       ;записать 11 в $0040
        jmp     main
metka12: mov    #12,$0040       ;записать 12 в $0040
        jmp     main
        end
```

В системах управления одной из типовых задач является задача определения показаний датчиков объекта, представляющих информацию в виде аналогового сигнала – уровня напряжения.

Перед разработчиком в данном случае возникает две задачи: масштабирования сигналов и аналогово-цифрового преобразования.

Первая задача решается применением схемотехнических решений (внешних по отношению к микроконтроллеру узлов масштабирования, выполненных, например, на базе операционных усилителей). Вторая задача решается либо схемотехнически (в случае отсутствия встроенного в микроконтроллер АЦП), либо программно (при наличии данного периферийного модуля, как в нашем случае).

Блок-схема программы для измерения напряжения с использованием АЦП представлена на рис. 2.7.

Для того чтобы программа могла начать работу, должны быть определены через команды присвоения *EQU* все спецификации аппаратных средств, такие как имена регистров и области памяти. С помощью объявлений определяются все переменные. Команда *ORG* сообщает ассемблеру адрес начала оперативной памяти. Программа *MAIN* – начало главной программы, которая вызывается после аппаратного сброса МК или команды *RESET* отладчика (команды *Power-on* для некоторых отладчиков).

Программа начинается с очистки тех битов и регистров, которые понадобятся при работе. Далее происходит инициализация встроенного аналогово-цифрового преобразователя. Загружается 16-разрядное число 40_{16} (%1000000) в регистр управления АЦП *ADCLK*, тем самым устанавливаем коэффициент деления частоты тактирования АЦП = 4. Рассмотрим подробнее биты регистра *ADCLK*.

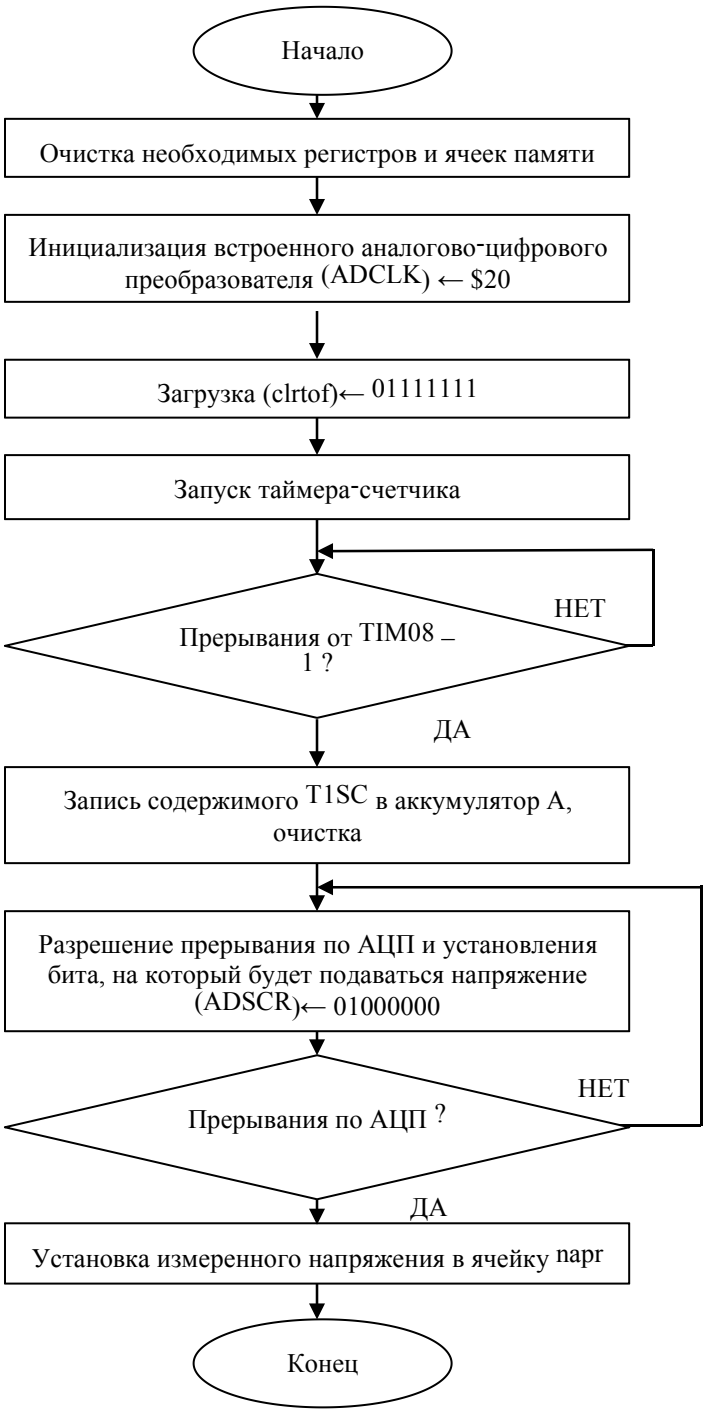


Рисунок 2.7 Блок-схема программы измерения напряжения

Ниже представлена программа для измерения напряжения с использованием АЦП:

```
romstart    equ   $8000
ramstart    equ   $0040
adscr       equ   $003c
adr         equ   $003d
 adclk      equ   $003e
```

```
T1SC      equ  $0020
T1MODh    equ  $0023
T1MODl    equ  $0024
adcin     equ  %01000000
napr      equ  $0042
          org  Romrtart
adc:      mov  #$40,adclk        ;задание коэффициента деления частоты
                                 ;тактирования АЦП, равное 4
          rts
tim1:     mov  #%01111111,clrtof ;запуск таймер-счетчик
          mov  #%01000010,T1SC
          mov  #$FF,T1MODh
          mov  #$FF,T1MODl
          rts
main:     clr  A                 ;очистка используемые ячейки памяти и
          clr  H                 ;регистры
          clr  X
          clr  Clrtof
          bsr  adc               ;переход на подпрограмму adc
          bsr  tim1              ;переход на подпрограмму tim1
          clr  Napr
          cli                    ;очистка глобальную маску прерывания
prer 1:   lda  T1SC              ;записываем содержимое T1SC в A
          and  Clrtof            ;очищаем
          sta  T1SC
          mov  #adcin,adscr      ;разрешаем прерывания по АЦП и
```

```
                                    установление
                                    ;бита,      на
                                    который
                                    будет
                                    подаваться
                                    напряжение
            rti
adcprer:    pshh
            mov   adr,napr          ;записываем
                                    измеренное
                                    напряжение в
                                    ячейку ;napr
            pulh
            rti
            rti
dummy_isr:
            end
```

	7	6	5	4	3	2	1	0
	ADIV2	**ADIV1**	**ADIV0**	**ADICLK**				
Состояние до записи	0	0	0	0	0	0	0	0
после записи	0	1	0	0	0	0	0	0

ADIV2-0: флаги включения предварительного делителя для таймера АЦП. Эти флаги управляют коэффициентом деления предварительного делителя таймера. При состоянии флагов *ADIV2-0* = 0 1 0 коэффициент деления частоты тактирования АЦП = 4.

ADICLK: флаг выбора таймера для АЦП. С помощью этого флага выбирается источник синхронизации для АЦП. В нашем случае *ADICLK* = 0, т.е. для синхронизации выбран внешний таймер *CGMXCLK*.

Таким образом, частота тактирования АЦП = 8МГц/4 = 2МГц. После этого нужно запустить таймер-счетчик. При этом, установить частоту тактирования таймера-счетчика. Для этого нужно выбирать коэффициент деления, равный 4, при помощи записи в биты *PS2–PS0* регистра *T1SC* кода 0 1 0. Частота тактирования в этом случае будет равна 8МГц/4 = 2МГц.

После того как таймер-счетчик остановится, в программе прерывание по таймеру счетчику разрешится АЦП-прерывание. Тут же задается бит *PTB0*, с которого будет поступать напряжение в микроконтроллер. Для этого в регистр состояния и управления АЦП *ADSCR* записывается следующий код: %01000000. Рассмотрим подробнее состояния битов регистра *ADSCR*.

	7	6	5	4	3	2	1	0
	COCO/ IDMAS	**AIEN**	**ADC0**	**ADCH4**	**ADCH3**	**ADCH2**	**ADCH1**	**ADCH0**
Состояние до записи	0	0	0	1	1	1	1	1
после записи	0	1	0	0	0	0	0	0

Когда *COCO* = 0, флаг *COCO* доступен только для чтения. Флаг *COCO* устанавливается при завершении преобразования.

AIEN = 1 – прерывание АЦП разрешено. Прерывание разрешается после завершения АЦП.

ADCO = 0 – выполняется однократное преобразование по выбранному каналу.

ADCH4-0: выбор канала. С помощью этих пяти флагов выбирается вход, информация которого будет преобразована. В нашем случае *ADCH4*-0 = 0 0 0 0 0 соответствует тому, что входом будет нулевой бит порта *B PTB0*. В подпрограмме прерывания по АЦП измеренное напряжение записывается в специально отведенную ячейку, которая получила название *napr*.

Организация временных функций

Временные задержки необходимы для обеспечения требуемого времени индикации, для ожидания окончания временных процессов в дискретных элементах системы, для генерации временных интервалов заданной длительности и пр. Выполнение этих функций реализуется специальными модулями микроконтроллеров – таймерами-счетчиками или процессорами событий. Временные задержки в пределах нескольких тактов можно обеспечить за счет использования команд процессора, не приводящих к изменению данных системы. Задержки большой длительности реализуются за счет вложенных циклов. Длительность задержки можно определить, если учесть количество тактов, необходимое для выполнения каждой команды. Такой способ организации временной задержки называется программным.

Рассмотрим пример реализации режима сравнения таймерного модуля *TIM08*, т.е. пример организации временной задержки длительностью 20 мкс. В данном примере задержки осуществляются благодаря таймеру-счетчику, который устанавливается таким образом, чтобы через каждые 20 мкс он переполнялся. Блок-схема данной программы представлена на рис. 2.8.

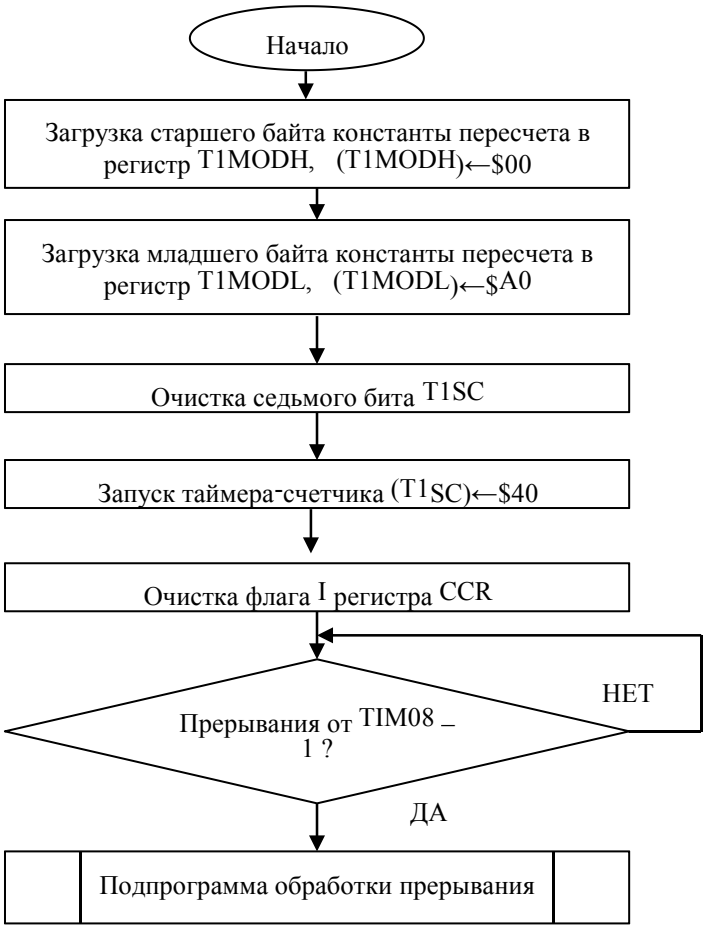


Рисунок 2.8 Блок-схема программы формирования временной задержки 20 мкс в режиме таймера

Текст программы реализации временной задержки длительностью 20 мкс представлен ниже.

```
romstart    equ   $9000      ;назначается начальный адрес прикладной
                              ;программы
ramstart    equ   $0040      ;назначается адрес первой ячейки памяти ОЗУ МК
vectorstart equ   $FFFE      ;назначается адрес вектора начального запуска
vectorTIM1  equ   $FFF6      ;назначается адрес вектора по прерыванию
T1SC        equ   $0020
T1MODH      equ   $0023
T1MODL      equ   $0024
            mov   $00,T1MODH ;записать старший разряд регистра данных
            mov   $A0,T1MODL ;записать младший разряд регистра данных
            bclr  7,T1SC     ;сбросить бит переполнения
            mov   #$40,T1SC  ;запустить таймер с разрешением прерываний
            cli              ;команда разрешения прерываний
```

```
prer:       ...                 ;программа обработки прерываний
                                ;на данном участке располагается подпрограмма
                                ;управления ;прерыванием
            org     vectorTIM1
            dw      prer
            end
```

Начало данной программы аналогично началам рассмотренных ранее программ. Так же происходит присвоение переменным символьных имен.

После этих команд происходит запись констант пересчета в регистр данных. Старший байт константы пересчета записывается в регистр *T1MODH*, младший байт константы пересчета записывается в регистр *T1MODL*. Таким образом 16-разрядный таймер-счетчик будет перебирать числа не от $0000-$*FFFF*, а в диапазоне $0000-$00A0. Затем очищается седьмой бит регистра управления-состояния *T1SC*, т.е. происходит сброс бита переполнения *TOF* регистра *T1SC*. Происходит запуск таймера с разрешением прерываний. В регистр *T1SC* записывается код $40 (%1000000):

	7	6	5	4	3	2	1	0
	TOF	**TOIE**	**TSTOP**	**TRST**	**-**	**PS2**	**PS1**	**PS0**
Состояние до записи		0	0	0	0	0	0	0
после записи		1	0	0	0	0	0	0

TOF – признак переполнения таймера, доступный только для чтения; принимает значение *TOF* = 1, если содержимое *T1CNT* достигает максимального значения, заданного содержимым регистра *T1MOD*.

Бит *TOIE* устанавливается в 1, что разрешает формирование запроса прерывания при переполнении счетчика (признак *TOF* = 1).

Бит *TSTOP* вызывает остановку работы таймера, если *TSTOP* = 1. В нашем случае *TSTOP* = 0, остановки не произойдет.

PS2–PS0 – определяют коэффициент деления частоты переключения счетчика *T1CNT*. В нашем случае *PS2–PS0* = 000, то коэффициент деления частоты $K_d = 1$.

В данный момент запускается таймер-счетчик, который проходит полный цикл от начального состояния кода $0000 до конечного $00A0 за время, определяемое по формуле:

$$t_{\text{вых}} = N \times t_{\text{вх}}\ t_{\hat{a}\hat{u}\tilde{o}} \qquad N \qquad t_{\hat{a}\tilde{o}}$$

где N – количество перебираемых чисел, в нашем случае \$00A0 = 160, $t_{вх}$ – длительность входных импульсов, определяются частотой внутренней шины f = 8 МГц.

$$t_{âõ} \quad \frac{1}{f}$$

Получаем:

$$t_{âûõ} \quad N \quad t_{âõ} \quad N \quad \frac{1}{f} \quad 160 \quad \frac{1}{8} \quad 20^{ìêñ} \quad . \quad 10^{6}$$

После переполнения таймера-счетчика, которое происходит каждые 20 мкс, управление переходит на подпрограмму обработки прерывания. Таким образом, реализуется организация временной задержки 20 мкс.

Теперь рассмотрим формирование временной задержки 0,5 с в режиме таймера. В данном примере временные задержки осуществляются благодаря таймеру-счетчику, который устанавливается таким образом, чтобы через каждые 0,5 с он переполнялся. Блок-схема программы формирования временной задержки представлена на рис. 2.9.

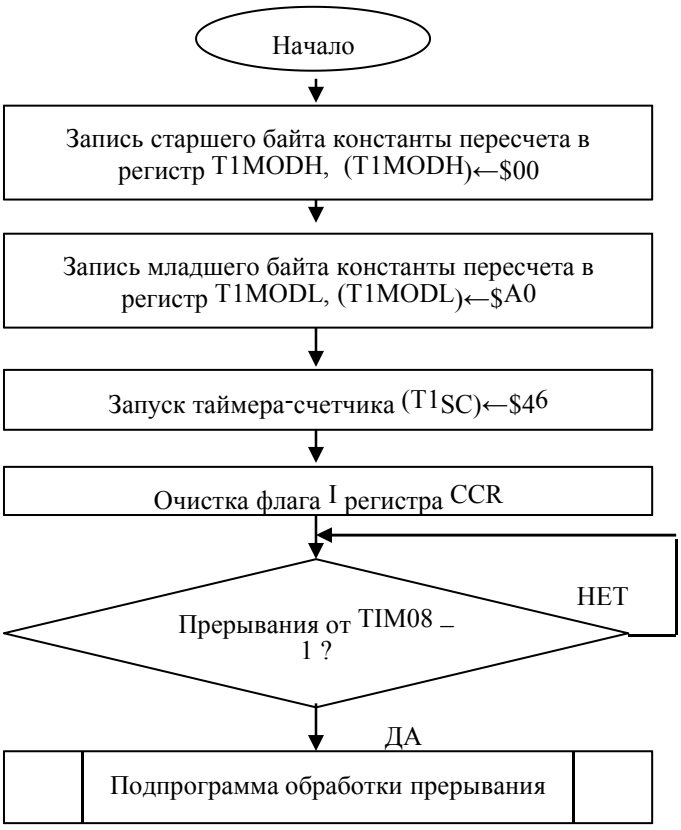


Рисунок 2.9 Блок-схема программы формирования временной задержки 0,5 с

Ниже представлен текст программы реализации временной задержки длительностью 0,5 с.

```
romstart    equ   $9000          ;назначается начальный адрес прикладной
                                 ;программы
ramstart    equ   $0040          ;назначается адрес первой ячейки памяти ОЗУ МК
vectorstart equ   $FFFE          ;назначается адрес вектора начального запуска
vectorTIM1  equ   $FFF6          ;назначается адрес вектора по прерыванию
T1SC        equ   $0020
T1MODH      equ   $0023
T1MODL      equ   $0024
            mov   $E4,T1MODH     ;записать старший разряд регистра данных
            mov   $24,T1MODL     ;записать младший разряд регистра данных
            mov   #$46,T1SC      ;записать в регистр управления код, по
                                 ;необходимым параметрам
            cli                  ;команда разрешения прерываний
prer:       ...                  ;программа обработки прерываний в данном
                                 ;участке располагается подпрограмма управления
                                 ;прерыванием
            org   vectorTIM1
            dw    prer
            end
```

Данная программа практически аналогична предыдущей программе. Рассмотрим отличия этой программы.

Старший байт константы пересчета $*E4* записывается в регистр *T1MODH*, младший байт константы пересчета $24 записывается в регистр *T1MODL*. Таким образом, 16-разрядный таймер-счетчик будет перебирать числа не от $0000 – $*FFFF*, а в диапазоне $0000 – $E424. В регистр *T1SC* записывается код $46 (%1000110).

	7	6	5	4	3	2	1	0
	TOF	**TOIE**	**TSTOP**	**TRST**	**-**	**PS2**	**PS1**	**PS0**
Состояние до записи		0	0	0	0	0	0	0
после записи		1	0	0	0	1	1	0

Бит *TOIE* устанавливается в 1, что разрешает формирование запроса прерывания при переполнении счетчика (признак *TOF* = 1).

Бит *TSTOP* вызывает остановку работы таймера, если *TSTOP* = 1. В нашем случае *TSTOP* = 0, остановки не произойдет.

PS2–PS0 – определяют коэффициент деления частоты переключения счетчика *T1CNT*. В нашем случае *PS2 – PS0* = 110, коэффициент деления частоты K_d = 64. Таким образом, частота входного сигнала будет определяться по формуле:

$$f_{âõ} \quad \frac{f}{64} \quad \frac{8 \cdot 10^6}{64} \quad 125 \quad 10^3 Ãö$$

где f = 8 МГц – частота внутренней шины микроконтроллера.

Таймер-счетчик проходит полный цикл от начального состояния кода \$0000 до конечного \$E424 за время, определяемое по формуле: $t_{âûõ} \quad N \quad t_{âõ}$

где *N* – количество перебираемых чисел, в нашем случае \$E424 = 58404, $t_{вх}$ – длительность входных импульсов, определяются частотой $f_{вх}$ = 125 КГц.

$$t_{âõ} \quad \frac{1}{f_{âõ}}$$

Получаем:

$$t_{âûõ} \quad N \quad t_{âõ} \quad N \quad \frac{1}{f} \quad \frac{1}{58404 \quad 125} \quad \frac{}{10^3} \quad 0.5c$$

âõ

После переполнения таймера-счетчика, которое происходит каждые 0,5 с, управление переходит на подпрограмму обработки прерывания. Таким образом, реализуется организация временной задержки 0,5 с.

Теперь рассмотрим программный способ организации временной задержки 20 мкс. Данный способ организации задержки основывается на реализации циклов. Каждую команду процессор выполняет за определенное время. В результате для получения заданной задержки можно подсчитать, какое количество команд необходимо для ее реализации. Блок-схема программы организации временной задержки длительностью 20 мкс программным способом представлена на рис. 2.10.

Рисунок 2.10 Блок-схема программы организации временной задержки длительностью 20 мкс программным способом

Ниже представлен текст программы организации временной задержки длительностью 20 мкс программным способом.

```
        org     $8000       ;задание начального адреса программы
        mov     #31,$40     ;записать константу пересчета в ячейку $40
m1:     dbnz    $40,m1      ;вычесть 1 из содержимого ячейки $40 и перейти по метке
                            ;m1, если результат не равен 0
        end
```

Представлен программный способ организации временной задержки. В произвольную, пустую ячейку $40 для организации временной задержки 20 мкс записывается константа-пересчета число 31. После этого выполняется команда условного перехода *dbnz*, которая вычитает единицу из содержимого ячейки $40 и проверяет на равенство 0. Если результат не равен 0, то процессор переходит по метке m1, т.е. опять возвращается на эту команду, так продолжается до тех пор, пока содержимое ячейки $40 не станет равным 0. После этого процессор идет на выполнение команды, следующей за командой *dbnz*. Таким образом, в приведенной программе команда *dbnz* будет выполняться 31 раз.

Рассмотрим, каким образом определяется константа пересчета. Общее время выполнения команд (временная задержка) определяется по формуле:

$$t_{çàä} \quad t_{mov} \quad T \quad N \quad t_{dbnz} \quad T$$

$$4 \quad 125 \quad 10^3 \quad N \quad 5 \quad 125 \quad 10^3 \quad 20 ìêñ$$

где N – константа пересчета, t_{mov}, t_{dbnz} – число циклов команд *mov, dbnz* соответственно, эти значения указаны в мнемокоде для микроконтроллера, T – период системной шины.

$$T \quad \frac{1}{f} \quad \frac{1}{8 \quad 6} \quad 125 \quad 10^{9} c$$

$$10$$

f – частота системной шины.

Отсюда можно вычислить константу пересчета N:

$$N \quad t_{çàä} \quad \frac{t_{mov} \quad T}{t_{dbnz} \quad T} \quad 20 \quad 10^{6} \quad 4 \quad 125 \quad 10^{9} \quad 31 \quad 5 \quad 125 \quad 10^{9}$$

Таким образом реализуется программный способ организации задержки.

Рассмотрим пример реализации режима захвата таймерного модуля *TIM08* при положительном переходе сигнала.

Блок-схема программы представлена на рис. 2.11.

Ниже представлена программа реализации режима захвата таймерного модуля *TIM08* при положительном переходе сигнала.

```
romstart    equ   $9000      ;назначается начальный адрес прикладной
                              программы
ramstart    equ   $0040      ;назначается адрес первой ячейки памяти ОЗУ МК
vectorstart equ   $FFFE      ;назначается адрес вектора начального запуска
vectorTIM1  equ   $FFF6      ;назначается адрес вектора по прерыванию
T1SC0       equ   $0025
T1SC        equ   $0020
T1CH0h      equ   $0026
            mov   #$45,T1SC0 ;задание регистра управления каналом
            mov   #$40,T1SC  ;запустить таймер с разрешением прерываний
prer:        ...             ;программа обработки прерываний в данном участке
                             ;располагается подпрограмма обработки прерывания
            org   vectorTIM1
            dw    Prer
            end
```

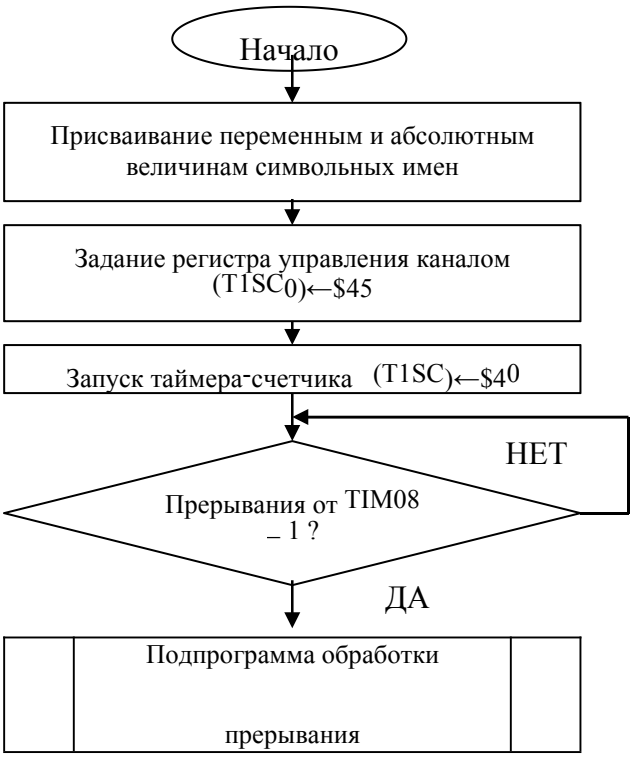


Рисунок 2.11 Блок-схема программы реализации режима захвата таймерного модуля TIM08 при положительном переходе сигнала

Для того чтобы программа могла начать работу, должны быть определены через команды присвоения *EQU* все спецификации аппаратных средств, такие как имена регистров и области памяти. С помощью объявлений определяются все переменные. Назначается начальный адрес прикладной программы, назначается адрес первой ячейки памяти ОЗУ МК, назначается адрес вектора начального запуска, назначается адрес вектора по прерыванию.

Далее при помощи команды *mov* задается регистр управления каналом:

mov #$45, *T1SCO*

Таким образом, в регистр управления каналом *T1SC0* записывается следующий код $45 (%1000101). Рассмотрим подробнее биты регистра *T1SC0*:

	7	6	5	4	3	2	1	0
	CH0F	**CH0IE**	**MS0B**	**MS0A**	**ELS0B**	**ELS0A**	**TOV0**	**CH0MAX**
Состояние до записи		0	0	0	0	0	0	0
после записи		1	0	0	0	1	0	1

После записи в регистр управления кода на нулевом бите *T1SC0 CH0MAX* установится высокий уровень. Этот бит определяет выбор коэффициента заполнения K_m в режиме формирования ШИМ-сигналов. Так как *CH0MAX* = 1, то коэффициент K_m = 1.

Бит *TOV0* задает вариант изменения сигнала на выходе *T1CH0* канала, работающего в режиме совпадения или в режиме форматирования ШИМсигналов. Поскольку *TOV0* = 0, то при переполнении таймера/счетчика произойдет сохранение текущего состояния.

Биты *ELS0B, ELS0A* определяют тип события в режиме захвата или уровень выходного сигнала в режиме совпадения. В нашем случае в эти биты записывается комбинация 0 1, в результате будет происходить захват при положительном перепаде сигнала на выходе *T1CH0*.

Биты *MS0B, MS0A* определяют режим работы канала. В эти биты записываются низкие уровни, что соответствует захвату при положительном перепаде сигнала на входе *T1CH0*.

Бит *CH0IE* = 1, что разрешает формирование запроса прерывания при срабатывании 0-го канала (признак *CH0F* = 1).

Бит *CH0F* доступен только для чтения, поэтому в этот бит ни чего не записывается. Это признак срабатывания 0-го канала принимает значение *CH0F* = 1, если канал формирует выходной сигнал в режиме совпадения.

После задания регистра управления каналом происходит запуск таймера с разрешением прерываний при помощи команды:

mov #$40, *T1SC*

В 8-разрядный регистр управления-состояния *T1SC* записывается код: $40 (%1000000)

	7	6	5	4	3	2	1	0
	TOF	**TOIE**	**TSTOP**	**TRST**	**-**	**PS2**	**PS1**	**PS0**
Состояние до записи		0	0	0	0	0	0	0
после записи		1	0	0	0	0	0	0

TOF – признак переполнения таймера, доступный только для чтения; принимает значение *TOF* = 1, если содержимое *T1CNT* достигает максимального значения, заданного содержимым регистра *T1MOD*.

Бит *TOIE* устанавливается в 1, что разрешает формирование запроса прерывания при переполнении счетчика (признак *TOF* = 1).

Бит *TSTOP* вызывает остановку работы таймера, если *TSTOP* = 1. В нашем случае *TSTOP* = 0, остановки не произойдет.

PS2–PS0 – определяют коэффициент деления частоты переключения счетчика *T1CNT*. В нашем случае *PS2–PS0* = 000, коэффициент деления частоты $K_d = 1$.

При переполнении счетчика происходит формирование запроса на прерывание, управление передается на программу обработки прерывания.

Команда *ORG VectorTIM1* сообщает ассемблеру адрес начала вектора по прерыванию. В последних двух командах программы нет абсолютной необходимости, но при таком подходе вероятность неадекватной работы при возникновении ложных запросов на прерывания снижается. Эти две команды реализуют таблицу векторов прерывания.

Рассмотрим пример реализации режима захвата таймерного модуля *TIM08* при отрицательном переходе сигнала.

Блок-схема программы представлена на рис. 2.12.

Данная программа практически аналогична программе реализации режима захвата таймерного модуля *TIM08* при положительном переходе сигнала. Отличие лишь в задании регистра управления каналом. В данном случае в регистр *T1SC0* записывается код $48 (%1001000). Рассмотрим подробнее каждый бит этого регистра:

	7	6	5	4	3	2	1	0
	CH0F	**CH0IE**	**MS0B**	**MS0A**	**ELS0B**	**ELS0A**	**TOV0**	**CH0MAX**
Состояние до записи		0	0	0	0	0	0	0
после записи		1	0	0	1	0	0	0

В данном случае *CH0MAX* = 0, отсюда следует, что коэффициент заполнения K_m определяется содержимым регистра данных 0-го канала *T1CH0*.

В битах *ELS0B, ELS0A* устанавливается комбинация 1 0, что определяет захват при отрицательном перепаде сигнала на входе *T1CH0*. В этом и заключается отличие от предыдущий программы.

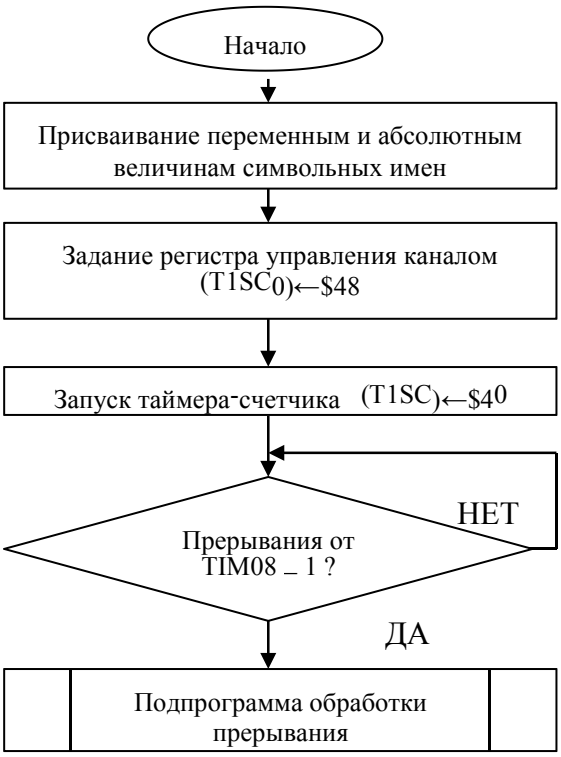


Рисунок 2.12 Блок-схема программы реализации режима захвата таймерного модуля TIM08 при положительном переходе сигнала

Ниже представлена программа реализации режима захвата таймерного модуля *TIM08* при отрицательном переходе сигнала.

```
romstart    equ   $9000      ;назначается начальный адрес прикладной
                              ;программы
ramstart    equ   $0040      ;назначается адрес первой ячейки памяти ОЗУ МК
vectorstart equ   $FFFE      ;назначается адрес вектора начального запуска
vectorTIM1  equ   $FFF6      ;назначается адрес вектора по прерыванию
T1SC0       equ   $0025
T1SC        equ   $0020
T1CH0h      equ   $0026
            mov   #$48,T1SC0 ;задание регистра управления каналом
            mov   #$40,T1SC  ;запустить таймер с разрешением прерываний
prer:        ...             ;программа обработки прерываний в данном участке
                             ;располагается подпрограмма обработки прерывания
            org   vectorTIM1
            dw    Prer
            end
```

В данном случае *CH0MAX* = 0, отсюда следует, что коэффициент заполнения K_m определяется содержимым регистра данных 0-го канала *T1CH0*.

В битах *ELS0B, ELS0A* устанавливается комбинация 1 0, что определяет захват при отрицательном перепаде сигнала на входе *T1CH0.* В этом и заключаются отличии от предыдущий программы.

Вывод информации из микроконтроллерной системы

В номенклатуру задач вывода информации входят как задачи вывода управляющей информации, поступающей в виде различных сигналов (амплитудных, частотных и других), на объекты управления, так и организация вывода информации на пульт оператора.

Рассмотрим программу, выполнение которой приводит к зажиганию светодиода, подключенного к одному из выводов порта ввода–вывода *PTD*.

Ниже представлен текст программы формирования управляющего сигнала на выводе *PTD5* порта вывода *PTD*.

```
Ramstart    equ     $0040       ;команды присвоения переменным символьных
Romstart    equ     $8000       ;имен
vectorstart equ     $FFDC
PTD         equ     $0003
DDRD        equ     $0007
            org     romstart    ;директива установки записи команд в блок Flash-памяти
            mov     $20, DDRD   ;конфигурировать вывод PTD5 порта D на вывод
                                ;информации
loop:       mov                 $00,PTD     ;установить низкий потенциал на выводе
                                для  ;зажигания светодиода
            bra     loop        ;организация цикла для постоянного горения светодиода
            end
```

Начало данной программы аналогично программе реализации контроля переключения ключа, подключенного к выводу порта ввода-вывода *PTD*. Благодаря команде *EQU* присваиваются переменным символьные имена. Команда *ORG* задает начальный адрес программы в памяти МК.

Команда *mov $20, DDRD* необходима для конфигурации вывода *PTD5* порта *D* на вывод информации. В регистр направления и передачи данных *DDRD* загружается код $20 (%100000).

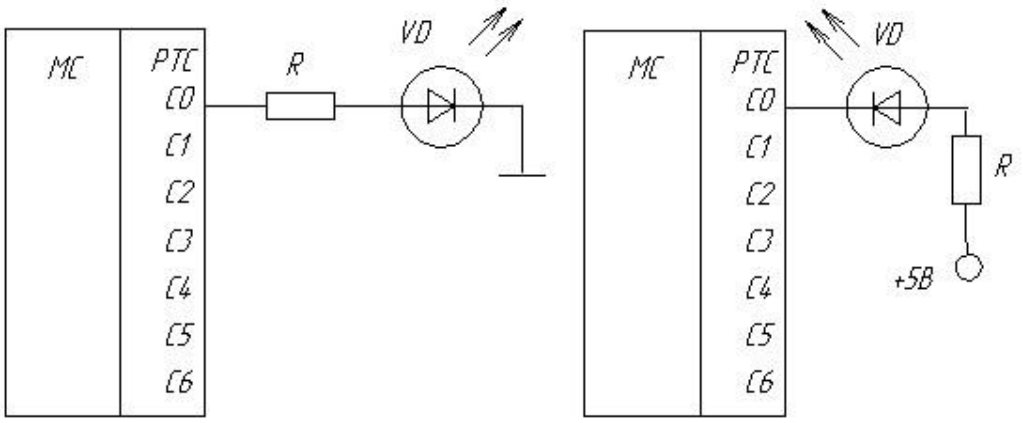
	7	6	5	4	3	2	1	0
	DDRD7	**DDRD6**	**DDRD5**	**DDRD4**	**DDRD3**	**DDRD2**	**DDRD1**	**DDRD0**

Состояние до записи	0	0	0	0	0	0	0	0
после записи	0	0	1	0	0	0	0	0

Таким образом, запрограммировали *PTD5* на вывод информации.

После этого на биты данных порта *D* подается код $00 при помощи команды *mov*, устанавливается низкий уровень на *PTD5* для зажигания светодиода. Далее организуется цикл для постоянного горения светодиода при помощи команды *bra loop*, которая обеспечивает безусловный переход на метку *loop*, таким образом, на *PTD5* будет всегда низкий уровень.

На рис. 2.13 представлены схемы подключения светодиодов к МК.



а) б)

Рисунок 2.13 Подключение светодиода к МК:
а) включение логической «1»; б) включение логическим «0»

Рассмотрим особенности подключения светодиодных индикаторов к микроконтроллеру. На рис. 2.14 и рис. 2.15 представлены схемы подключения светодиодного индикатора с общим катодом и общим анодом к МК.

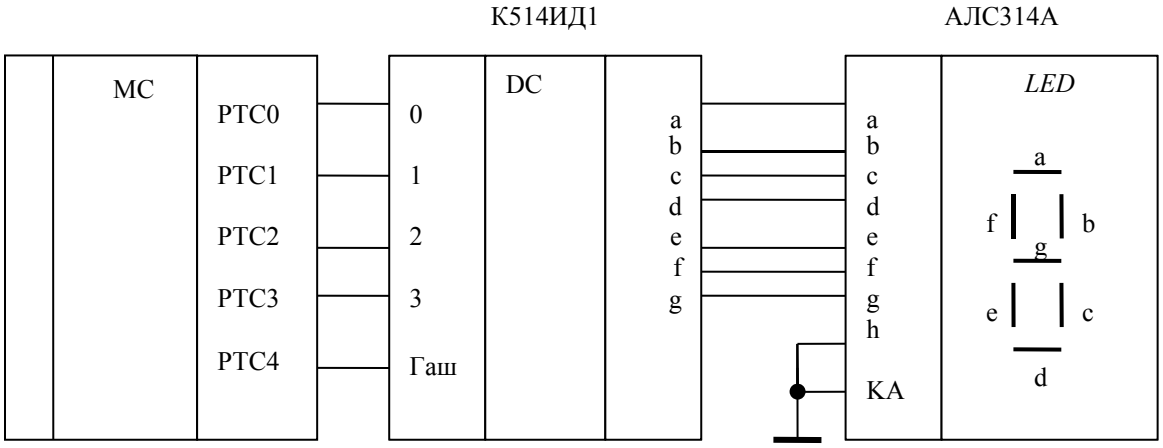


Рисунок 2.14 Подключение светодиодного индикатора с общим катодом к МК

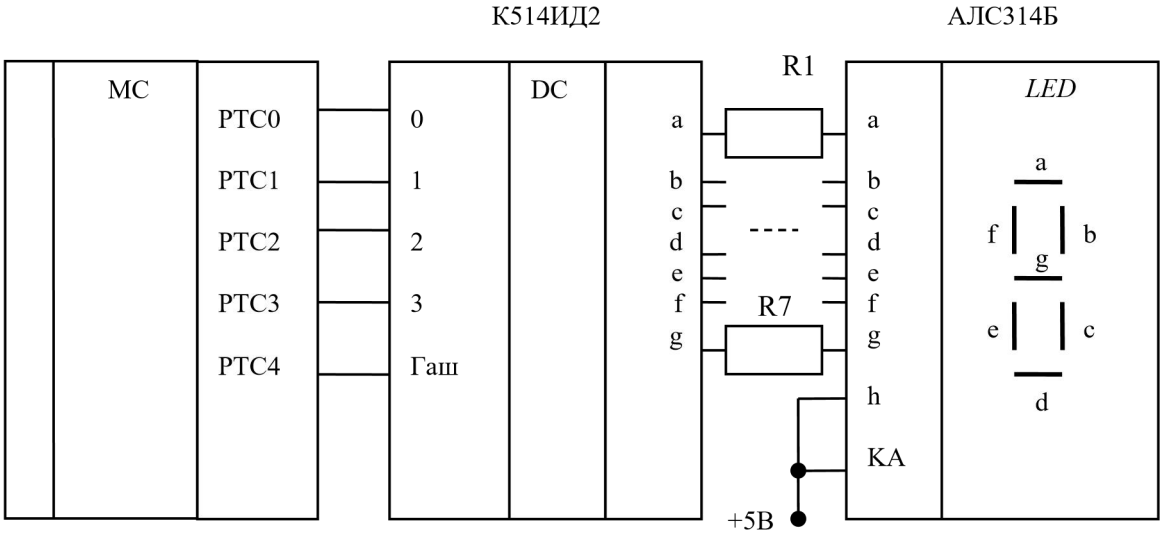


Рисунок 2.15 Подключение светодиодного индикатора с общим анодом к МК

Рассмотрим пример вывода информации на 2-разрядный 7-сегментный индикатор с использованием динамической индикации, которая заключается в следующем. Информация (числа), которые необходимо высветить через выводы порта *PTC PTC0 – PTC3* подаются на дешифратор. Дешифратор необходим для преобразования двоичного кода в код семисегментного индикатора. Через *PTC4* осуществляется управление десятичной точкой индикатора. А через выводы *PTC5, PTC6* подаются управляющие импульсы, которые определяют тот разряд, который будет активным.

Очевидно, что необходимо синхронизировать информационные сигналы с управляющими. На рис. 2.16 приведена блок-схема программы вывода информации на 2-разрядный 7-сегментный индикатор.

Текст программы вывода информации на 2-х разрядный семисегментный индикатор представлен ниже.

```
romstart:     equ   $8000
ramstart:     equ   $0038
vectorstart:  equ   $FFDC
vectorTIM1    equ   $FFF2
T1SC:         equ   $0020
T1MODh:       equ   $0023
T1MODl:       equ   $0024
PTDPUE:       equ   $000A
DDRC:         equ   $0006
PTC:          equ   $0002
count         equ   $0039      ;счетчик
Rezultat      equ   $0053      ;результат, который надо вывести
syncro        equ   $0055      ;ячейка, которая используется для
                               ;синхронизация разрядов при выводе

                               ;информации на 7-сегментный индикатор
syncro+1:     equ   $0056
reserve:      equ   $0057      ;ячейка для временного хранения результатов
                               ;промежуточных операций
odin          equ   $0058      ;ячейка, которая используется для определения
                               ;номера работающего разряда при
                               выводе ;информации на 7-сегментный
                               индикатор
              org   Romstart
main1:        clra             ;очистить используемые регистры и ячейки
                               ;памяти
              clrx
              clr   Count
              clr   rezultat
              clr   syncro
              clr   syncro+1
              clr   reserve
              clr   odin
              mov   #%0100000,syncro    ;1-й разряд
              mov   #%1000000,syncro+1  ;2-й разряд
              mov   #$1,odin
              mov   #%1111111,DDRC      ;установить все биты PTC на вывод
              mov   #$00,PTC            ;обнулить порт C

obnulenie:    mov   #$0,count           ;обнулить счетчик
```

```
flag:       bsr     display
            inc     count           ;увеличить содержимое счетчика на 1
            lda     count
            cmp     odin            ;сравнить содержимое A с odin
            bhi     obnulenie       ;если больше 1, обнулить count
            jmp     flag
display:    lda     #rezultat
            add     count           ;сложить содержимое A с содержимым
                                    ;count
            tax                     ;переслать содержимое A в X
            lda     ,x              ;установить в A содержимое ячейки адрес
                                    ;которой указан в X
            sta     reserve         ;установить содержимое A в reserve
            lda     #syncro         ;установит в A адрес syncro
            add     count           ;сложить содержимое A с содержимым count
            tax
            lda     ,x
            add     reserve         ;сложить содержимое A с содержимым  reserve
            mov     #1250,$59
            sta     PTC             ;установить в PTC
izo:        mov     #%0100110,T1SC  ;запустить т/с
            mov     #$FF,T1MODH
            mov     #$FF,T1MODL
            jmp     *
prer2:      dec     $59             ;уменьшить содержимое ячейки $59 на 1
            bne     Izo
            mov     T1SC,$50        ;выключить т/с
            bclr    7,T1SC
            mov     #0,PTC          ;выключить индикатор
      rts                   rti
end
```

Для того чтобы программа могла начать работу, должны быть определены через команды присвоения *EQU* все спецификации аппаратных средств, такие как имена регистров и области памяти. С помощью объявлений определяются все переменные. Команда *ORG* сообщает ассемблеру адрес начала оперативной памяти. Программа *MAIN* – начало

главной программы, которая вызывается после аппаратного сброса МК или команды *RESET* отладчика (команды *Power-on* для некоторых отладчиков).

Основная программа начинается с очистки необходимых при работе регистров и ячеек памяти. Заметим, что эта процедура является важной и необходимой для обеспечения нормальной работы программы. После данной процедуры в ячейки *syncro, syncro+1, odin* загружаются значения, которые понадобятся при работе. Далее происходит инициализация порта *PTC*. В регистр *DDRC* порта записывается следующее число: %1111111, таким образом, все выводы порта программируются на вывод информации. После этого начинается выбор того разряда 7-сегментного индикатора, на который будет подаваться информация.

Здесь реализована динамическая индикация. Работа индикатора осуществляется следующим образом: разряды индикатора работают поочередно, т.е. в один момент времени может работать только один разряд. Время переключения между разрядами настолько малое, что человеческому глазу оно практически не заметно. Поэтому оператор будет наблюдать постоянно высвеченное значение. По этой причине необходимо синхронизировать управляющие 7-сегментным индикатором сигналы с информационными сигналами.

Происходит очистка счетчика (*count*). Далее происходит безусловный переход к подпрограмме *DISPLAY*, в которой происходит синхронизация управляющих 7-сегментным индикатором сигналов с информационными. После выполнения данной подпрограммы процессор возвращается к команде, следующей за командой *bsr DISPLAY*, происходит увеличение содержимого счетчика на единицу, загрузка содержимого счетчика в аккумулятор A, сравнение содержимого аккумулятора с содержимым ячейки *odin*, т.е. с единицей. Если содержимое счетчика больше единицы, то обнуляем счетчик, в противном случае начинает выполняться подпрограмма *DISPLAY*.

Рассмотрим подробнее выполнение подпрограммы *DISPLAY*. В начале в аккумулятор загружается адрес ячейки *rezultat*. Затем содержимое аккумулятора A складывается с содержимом счетчика count, после чего результат арифметической операции посылается в индексный регистр *H:X*. В аккумулятор загружается число, адрес которого указан в ячейке *H:X*, это число из аккумулятора пересылается в ячейку reserve. Далее в аккумулятор загружается адрес ячейки *syncro*, происходит сложение содержимого аккумулятора с содержимым счетчика count. Результат этой операции посылается в индексный регистр *H:X*. В аккумулятор после этого записывается число, адрес которого указан в регистре *H:X*. В результате происходит сложение содержимого аккумулятора с содержимым ячейки

reserve. Таким образом, в аккумуляторе содержится синхронизированный сигнал, который уже можно посылать на вывод.

Сигнал на индикатор поступает с частотой 100 Гц. Эта частота реализуется при помощи модуля *TIM08* следующим образом. В биты *PS2 PS1 PS0* регистра *T1SC* записывается код 1 1 0, соответственно. Это необходимо для задания коэффициента деления. Частота тактирования = f_{bus}/64. f_{bus} для *MC68HC908GP32* = 8 МГц. Подставляя в формулу, указанную выше, находим, что частота тактирования = 125000 Гц. В самой подпрограмме прерывания необходимо организовать цикл, в котором таймер-счетчик прокручивается 1250 раз. После 2-го раза таймер-счетчик необходимо остановить. Таким образом, частота сигнала, поступающего на индикатор, = 125000/1250 = 100 Гц.

Пример описания принципиальной электрической схемы МКУ

Для обеспечения питания микроконтроллера и задания тактовой частоты используются определенные входные цепи элемента *DD*1. Выводы *VDD*, *VDDA* и *VDDAD* подключаются к источнику постоянного тока 5В, на «землю» подключаются выводы *VSS*, *VSSA* и *VSSAD*.

Резонатор, задающий тактовую частоту, подключается к выводам *OSC*1 и *OSC*2 МК. Конденсатор С1 – конденсатор постоянной емкости. С2 – переменный конденсатор, используемый для адаптации кварцевого резонатора к генератору после определения оптимальной емкости. Номиналы этих конденсаторов лежат в диапазоне 6 – 40 пФ. Резистор *R*1 = 10 – 22 МОм, *R*3 = 1кОм, *QZ* – кварцевый резонатор с *f* = 32 кГц.

Схема внешнего фильтра значительно влияет на стабильность работы МК. Подключается внешний фильтр к выводу *GMXFC* МК. Здесь *R*2 = 10 кОм, *C*3 = 0,033 мкФ, *C*4 = 0,01 мкФ

Цифровой датчик температуры питается от того же источника питания, что и микроконтроллер, то есть от источника постоянного тока +5В. Питание подается на входы *VDDD*, *VDDA*, *SERMODE* термодатчика. Вывод *GND* подключается на «землю».

Термодатчик подключается к выводам *PTD*0 – *PTD*3 порта *D* микроконтроллера.

Вывод *CE* термодатчика подключается к *PTD*0 (*SS*) - вход сигнала выбора ведомого модуля: на этот вход для термодатчика необходимо подать сигнал *SS* =0.

Вывод *SDO* термодатчика подключается к *PTD*1 (*MISO*) - вход данных для микроконтроллера и выход данных для термодатчика.

Вывод *SDI* термодатчика подключается к *PTD*2 (*MOSI*) - выход данных для микроконтроллера и вход данных для термодатчика.

Вывод *SCLK* термодатчика подключается к *PTD*3 (*SPSCK*) - выход синхросигнала ведущего и вход синхросигнала ведомого модуля.

Как видно из вышесказанного, термодатчик является ведомым модулем, а микроконтроллер ведущим модулем.

Весь блок индикации состоит из следующих функциональных узлов: дешифратора, который преобразует входной двоичный код в семисегментный код; трех семисегментных полупроводниковых индикатора *HG*1,*HG*2,*HG*3, три транзисторных ключа для включения соответствующего ключу индикатора. Рассмотрим каждый узел отдельно.

Питается дешифратор *DD*2 от источника постоянного тока +5В, то есть подключаем к общему источнику питания вывод 16 дешифратора. Вывод 8 подключается к «земле». Выводы 6, 2, 1, 7 подключены к выводам порта А МК *PTA*0, *PTA*1, *PTA*2, *PTA*3 соответственно. Вывод гашения не используется, так как полупроводниковая индикация гаснет мгновенно и эффекта остаточного свечения как у ЖК индикации не наблюдается. Сигнал на выводах 9 – 15 низкого уровня (лог.0), это достигается за счет схемы дешифратора с разъединенными катодами сегментов.

Питаются индикаторы от общего источника постоянного тока +5В. Питание каждого индикатора производится через транзисторный ключ, описание которого дается далее, отметим только, что раз на индикаторы с дешифратора подается низкий уровень сигнала, то с транзисторного ключа должен приходить на индикатор сигнал с высоким уровнем (лог.1).

Индикаторы *HL*1 - *HL*3 предназначены для отображения непосредственно значения температуры в десятичном коде. Индикатор *HL*1 отображает сотни, индикатор *HL*2 отображает десятки, а индикатор *HL*3 единицы значения температуры. Питание индикаторов производится от источника постоянного тока +5В через соответствующие каждому индикатору транзисторные ключи. На входы $a - h$ индикаторов сигналы приходят с дешифратора. Как говорилось ранее, эти сигналы имеют низкий логический уровень. Перенагрузки выходов дешифратора нет, так как максимальный нагрузочный ток каждого вывода дешифратора составляет 22мА, а индикатор потребляет 20мА. Для избегания перегрузки входов семисегментных индикаторов включаем в схему токоограничивающие резисторы *R*9 – *R*16. Так как максимальное напряжение на выходе дешифратора может достигать 5В, а нагрузочная способность входов индикаторов 20мА рассчитаем номинал токоограничивающих резисторов:

$$R_{9\ldots16} = \frac{U^{max}}{I_{ном}} = \frac{5\,B}{0{,}02\,A} = 250\,Ом$$

Из справочника выбираем резисторы небольшой мощности, например, МЛТ-0,25-250Ом ± 10%.

Через транзисторные ключи подключаются входы питания семисегментных индикаторов с управляющими выводами микроконтроллера (выводы порта А *PTA*4 – *PTA*7). Делается это в связи с тем, что уровня сигнала с выходов микроконтроллера не достаточно для включения индикаторов, и для увеличения этих уровней до необходимых и используются транзисторные ключи. Благодаря *n-p-n* структуре выбранных транзисторов обеспечивается управление питанием индикаторов сигналом высокого логического уровня (о необходимости этого говорилось выше). К базе транзистора подключается вывод микроконтроллера, коллектор подключен к общему источнику питания +5В, эмиттер подключен ко входу питания семисегментного индикатора. Принцип работы ключа: на базу транзистора подается высокий потенциал, транзистор открывается и питающее напряжение подается на индикатор. Выбираем из справочника маломощный транзистор с *n-p-n* структурой: КТ315. Во избежание перегрузки портов микроконтроллеров и для уменьшения тока базы транзистора используем токоограничивающие резисторы номиналом 1кОм: МЛТ- 0,125 - 1 кОм ± 10%.

Светодиод служит световой сигнализацией режима работы МКУ. Диод подключается таким образом, что его засвечивание происходит высоким логическим уровнем сигнала. Марка светодиода *L* – 1593*SGT*. Технические характеристики:

–номинальное напряжение 5В; – номинальный ток 15мА.

Для избегания перегрузки диода включаем в схему токоограничивающий резистор с номиналом: $R_7 \quad \frac{U_{нно} \quad 5В}{I_{нно} \quad 0{,}015А} \quad 330Ом$

Звуковой сигнал используется для передачи информации на расстоянии без визуального контакта с объектом наблюдения/управления. Используется пятиватный динамик ЗП-1. Положительный электрод которого подключается к выводу *PTC*1 порта *PTC* микроконтроллера, а отрицательный электрод подключается к земле.

Нагреватель подключаем к выводу *PTC*3 порта С.

Непосредственно к порту микроконтроллера подключение нагревателя не представляется возможным в связи со слишком малым сигналом с порта МК. Нагреватель питается от сети с переменным током 220В.

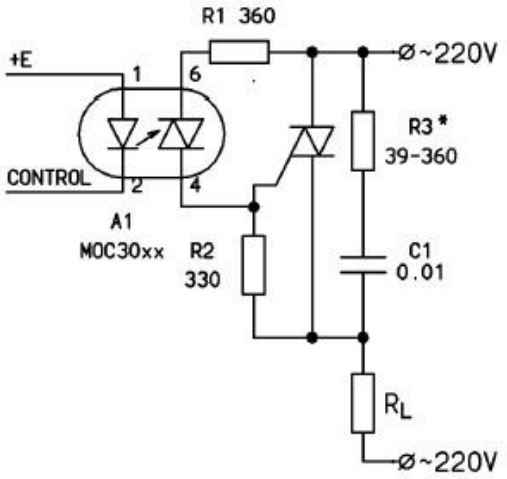


Рисунок 2.16 Схема подключения нагревателя

Схема подключения нагревателя представлена на рисунке 2.16 Здесь управляющим элементом служит оптосимистор, который сигналом с микроконтроллера открывает симистор и ток протекает от сети к нагревателю. Схема включения *RC* – цепочки параллельно симистору улучшает динамические характеристики схемы и гасит помехи создаваемые приборами. Также для уменьшения помех используется оптосимистор *MOC*3032*M* с детектором нуля, то есть оптосимистор открывается только при пересечением синусоиды переменного тока оси абсцисс, то есть нуля. Параметры оптосимистора приведены в табл. 2.1.

Таблица 2.1 - Характеристики оптосимистора *MOC3032M*

ТИП	*MOC*3032*M*
Максимальный ток включения светодиода, мА	10
Наличие детектора нуля (*ZCC*)	+
Максимальноенапряжение на светодиоде, В	3
Максимальноенапряжение симистора в закрытом состоянии, В	250
Максимальный ток утечки симистора, нА	100
Минимальная скорость нарастания сигнала, В/мкс	100
Напряжение изоляции, КВ (1 минута)	4,2

Мощность потребляемая нагревателем составляет 2кВт (см. табл. 2.1). Найдем нагрузочный ток для выбора симистора:

$$I \frac{P}{U} \frac{2000^{Bm}}{220B} 9{,}09A$$

Для того чтобы симистор не вышел из строя при импульсном токе необходимо взять его с большей нагрузочной способностью. Из справочника симисторов фирмы *Philips* выбрали симистор *BT*138*B*-600*F* с параметрами:

– нагрузочный ток: 12А; –

напряжение (макс): 600В;

– ток затвора: 25мА.

Также необходимо в цепь управления перед светодиодом включить в схему токоограничивающий резистор с номиналом:

$$R_4 \frac{U}{I} \frac{3^B}{0{,}01A} 300Ом$$

Для подключения вентилятора к сети и управление им с помощью микроконтроллера используем схему на оптосимисторе, как показано на рисунке 2.17.

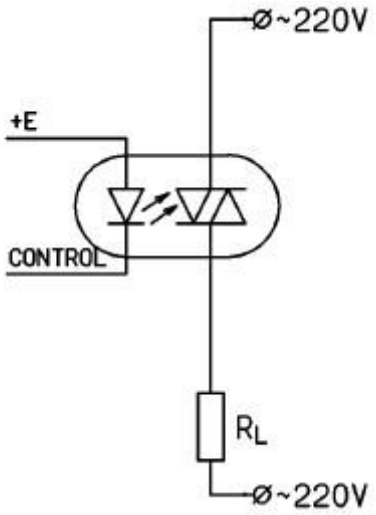


Рисунок 2.17 Схема подключения вентилятора

Для уменьшения помех используем симистор с датчиком нуля, как и в случае с нагревателем. Характеристики используемого оптосимистора приведены в табл. 2.1 . Аналогично с нагревателем включаем в схему токоограничивающий резистор *R*5 с номиналомм порядка 300 Ом.

СПРАВОЧНАЯ ИНФОРМАЦИЯ ДЛЯ ВЫПОЛНЕНИЯ КУРСОВОГО ПРОЕКТА

Модуль таймера/счетчика *TIM08*

Модуль *TIM08*, хотя и носит дословное название «модуль таймерного интерфейса», по сути, является одним из лучших процессоров событий в 8разрядных МК.

Модуль *TIM08* состоит из 16-разрядного таймера-счетчика и некоторого количества связанных с ним полностью идентичных каналов захвата/сравнения. Каждый из каналов в процессе инициализации может быть настроен на один из пяти режимов работы:

– входного захвата;
– небуферированного выходного сравнения;
– буферированного выходного сравнения;
– небуферированной широтно-импульсной модуляции (ШИМ); – буферированной ШИМ.

Каждый канал захвата/сравнения связан с одним из выводов МК. Функция входного или выходного сигнала модуля процессора событий является альтернативной функцией линий порта *Port D*. МК *MC*68*HC*908*GP*32 имеет в своем составе два модуля процессора событий – *TIM1* и *TIM2*.

Каждый из модулей имеет по два канала захвата/сравнения.

16-разрядный таймер-счетчик служит временной базой для модулей захвата/сравнения. Он подсчитывает импульсы тактовой частоты, поступающие на его вход. Все интервалы времени, которые генерируются или измеряются МК, измеряются числом периодов этой тактовой частоты.

В общем случае (модули *TIM08* других МК семейства *HC08*) таймерсчетчик имеет два источника тактирования:

– внутренний генератор, выполненный на основе программируемого делителя частоты шины МК f_{bus};
– внешний генератор, подключаемый к выводу *TxCLK* МК.

Выбор между внутренним и внешним генераторами, а также выбор коэффициента деления программируемого делителя частоты шины $K_{T|Mx}$ определяется комбинацией битов *PS2–PS0* регистра управления таймерасчетчика *TxSC*. Коэффициент деления $K_{T|Mx}$ может принимать семь различных значений: 1, 2, 4, 8, 16, 32, 64. Максимальная частота сигнала

внешнего генератора, подключаемого ко входу *TxCLK*, составляет 4 МГц при условии, что МК работает на предельной частоте внутренней шины f_{bus} – 8 МГц. Процессоры событий *TIM1* и *TIM2* в составе МК *MC68HC908GP32* не имеют выводов *T1CLK* и *T2CLK* в перечне выводов корпуса и, следовательно, могут использовать для тактирования только встроенный генератор.

Таймер-счетчик временной базы допускает программную установку периода работы. Если не предпринимать специальных действий при инициализации процессора событий, то коэффициент счета счетчика временной базы будет равен 2^{16}, т.е. счетчик проходит полный цикл от начального состояния кода $0000 до конечного состояния кода S*FFFF*. Если таймер-счетчик находится в состоянии $*FFFF*, то при поступлении на его вход очередного тактового импульса наступает переполнение таймерасчетчика. Счетчик переходит в состояние $0000, одновременно устанавливается флаг переполнения *TOF*. Переполнение счетчика не оказывает влияния на его работу: при поступлении следующих тактовых импульсов код в счетчике продолжает нарастать. Коэффициент счета таймерасчетчика может быть изменен посредством записи кода желаемого $K_{сч}$ в двухбайтовый регистр периода *TxMOD* (*TxMODH* и *TxMODL* - старший и младший байты этого регистра, x - номер таймерного модуля, для *TIM1* x = 1, для *TIM2* x = 2). Вход сброса счетчика подключен к выходу цифрового компаратора, на один из входов которого поступает код текущего состояния таймера-счетчика, а на другой - код $K_{сч}$, записанный в регистре *TxMOD*. Если эти коды равны, то при поступлении следующего тактового импульса счетчик сбрасывается в «0», и флаг переполнения TOF устанавливается в «1». Диапазон допустимых значений $K_{сч}$ составляет от 1 до (2^{16}-1). Таким образом, дискретность регулирования периода таймера-счетчика, который в режиме ШИМ образует период ШИМ-сигнала, составляет 16 бит.

Предусмотрена возможность пуска и останова таймера-счетчика под управлением программы (бит *TSTOP* в регистре управления таймеромсчетчиком *TxSC*). Кроме того, счетчик и программируемый делитель частоты могут быть одновременно сброшены посредством установки в «1» бита *TRST* в регистре *TxSC*. При этом все триггеры таймера-счетчика установятся в «О», а программируемый делитель частоты будет настроен на режим единичного коэффициента деления частоты внутренней шины МК. Заметьте, что бит TRST не останавливает работу таймера-счетчика, с приходом очередного тактового импульса состояние счетчика станет равным $0001. Сброс таймера-счетчика рекомендуется проводить в следующем порядке:

– остановить таймер-счетчик (бит *TSOP* = 1);
– выполнить операцию сброса таймера-счетчика (бит *TRST* = 1);

– переинициализировать биты *PS2–PS0* регистра *TxSC*, которые определяют источник и частоту тактирования; – разрешить счет таймера-счетчика.

Код таймера-счетчика в процессе счета может быть считан прикладной программой при обращении к регистрам текущего кода *TxCNTH* и *TxCNTL*. При обращении к регистру старшего байта код таймера-счетчика автоматически копируется в указанную регистровую пару. Поэтому, несмотря на то, что операции чтения старшего и младшего байтов разнесены во времени, вы прочитаете состояние таймера-счетчика в момент обращения к регистру старшего байта *TxCNTH*. Такое решение предотвращает получение ложной информации в случае, если частота тактирования таймера-счетчика высока, и по этой причине в моменты обращения к регистрам *TxCNTH* и
TxCNTL состояния счетчика различаются. Однако нельзя допускать ситуацию, при которой после прочтения старшего байта младший прочитан, не будет. Повторное чтение старшего байта не сопровождается защелкиванием текущего кода таймера-счетчика в регистрах *TxCNTH* и *TxCNTL*.

При переполнении таймера-счетчика устанавливается флаг переполнения *TOF* в регистре управления *TxSC* (приложение В) и генерируется запрос на прерывание, если бит разрешения прерывания *TOIE* установлен в «1», т. е. прерывания по переполнению таймера-счетчика разрешены.

Для управления таймером-счетчиком модуля *TIM08* предусмотрены пять регистров специальных функций:

TxSC – регистр управления таймером-счетчиком «x», где «x» - имя модуля процессора событий МК (для TIM1 x=1, для TIM2 x=2);

TxMODH – регистр периода таймера-счетчика (старший байт);

TxMODL – регистр периода таймера-счетчика (младший байт);

TxCNTH – регистр текущего значения таймера-счетчика (старший байт);

TxCNTL – регистр текущего значения таймера-счетчика (младший байт).

Организация памяти и портов ввода/вывода

Микроконтроллеры семейства 68*HC*08/908 адресуют 64 Кбайт внутренней памяти (адреса $0000–*FFFF*). Распределение адресного пространства задается картой памяти, вид которой определяется объемом внутренней памяти и набором периферийных устройств, входящим в состав данной модели микроконтроллера. На рис. 3.1 приведена карта памяти для микроконтроллеров *MC*68*HC*908*GP*32.

В адресном пространстве имеется ряд неиспользуемых позиций, которые соответствуют ячейкам памяти, отсутствующим в данной модели микроконтроллеров. При обращении к этим адресам производится перезапуск микроконтроллера.

Младшие 64 позиции адресного пространства (адреса $000–$003F) занимают регистры служебных и периферийных модулей, табл. 3.1

Отметим, что 16-разрядные регистры таймерных модулей *TCNT*, *TMOD, TCHx* занимают по две позиции адресного пространства: младший байт с суффиксом l, старший байт с суффиксом *h*.

В адресном пространстве ОЗУ располагаются ячейки стека, которые адресуются с помощью указателя стека *SP*. При установке микроконтроллера в начальное состояние (запуске) содержимое *SP* принимает значение $00*FF*, адресуя ячейку ОЗУ с данным адресом. В процессе выполнения программы можно установить любое значение указателя стека с помощью команды *TXS*, которая загружает в *SP* содержимое индексного регистра *H:X*, уменьшенное на 1. После записи байта в стек содержимое *SP* уменьшается на 1, адресуя следующую незаполненную ячейку стека. Таким образом, стек заполняется в направлении уменьшения адресов. Адрес вершины стека (последней заполненной ячейки стека) можно загрузить в регистр *H:X* с помощью команды *TSX*.

Регистры периферийных и служебных модулей (64 байт)
ОЗУ данных (512 байт)
Не используется (32 192 байт)
Flash-память (32 256 байт)
Регистр SBSR (модуль BREAK08)
Регистр SRSR (указывает причину запуска)
Резервировано
Регистр SBFCR (модуль BREAK08)
Регистр INT1 (запросы прерывания)
Регистр INT2 (запросы прерывания)
Регистр INT3 (запросы прерывания)
Резервировано
Регистр FLCR (управление Flash-памятью)
Регистр BRKh (модуль BREAK08)
Регистр BRKl (модуль BREAK08)

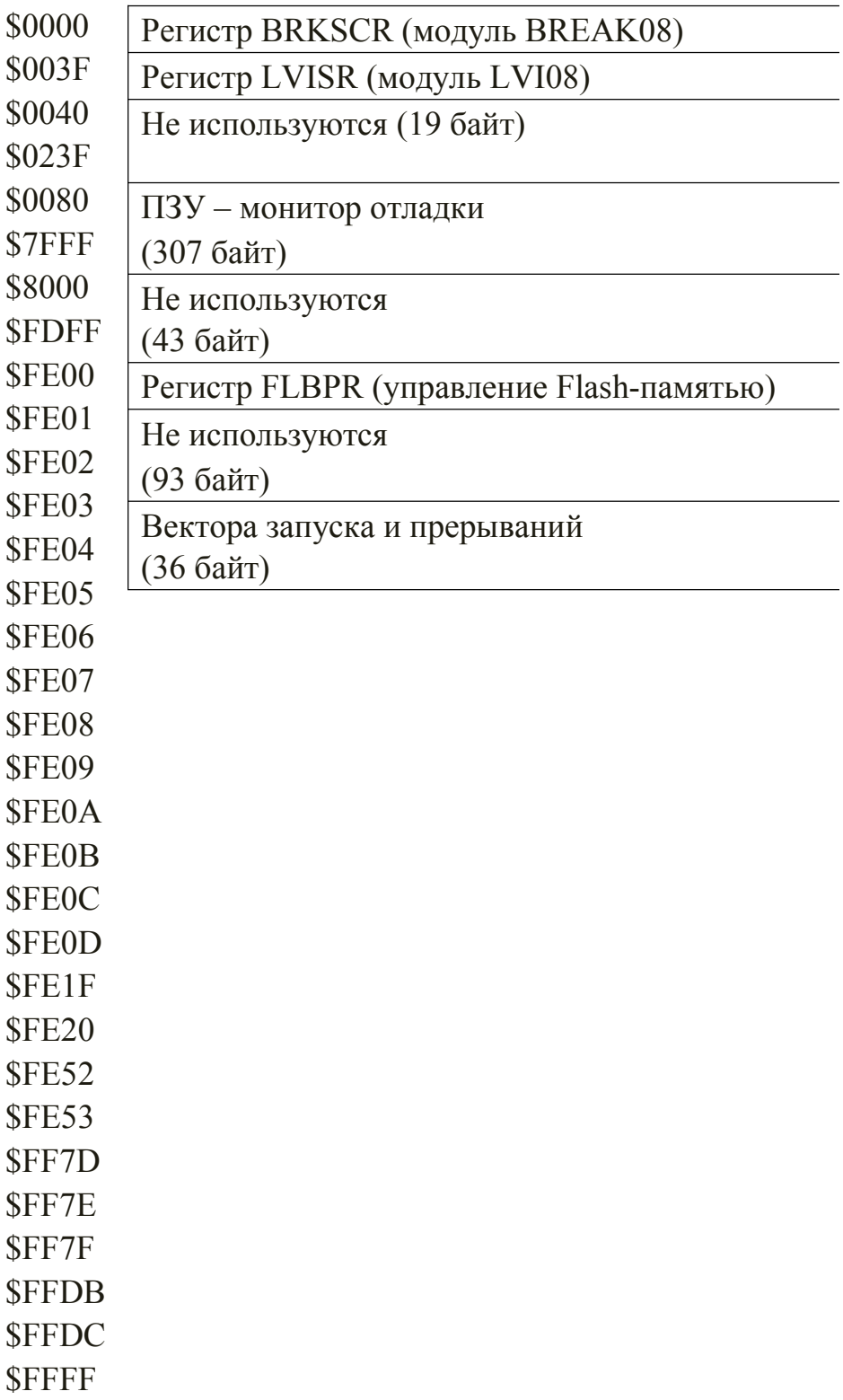


Рисунок 3.1 Карта памяти для микроконтроллера *MC*68*HC*908*GP*32

Таблица 3.3 - Адреса регистров периферийных модулей

Адреса регистров периферийных модулей

Адрес	Регистр	Назначение
$0000	PTA	Регистры данных портов A, B, C, D
$0001	PTB	
$0002	PTC	
$0003	PTD	
$0004	DDRA	Регистры направления передачи портов A, B, C, D
$0005	DDRB	
$0006	DDRC	
$0007	DDRD	
$0008	PTE	Регистр данных порта E
$0009-0B		Не используются
$000C	DDRE	Регистр направления передачи порта E
$000D	PTAPUE	Регистры управления подтягивающими резисторами портов A, C, D
$000E	PTCPUE	
$000F	PTDPUE	
$0010	SPCR	Регистры синхронного порта SPI08
$0011	SPSCR	
$0012	SPDR	
$0013	SCC1	Регистры асинхронного порта SCI08
$0014	SCC2	
$0015	SCC3	
$0016	SCS1	
$0017	SCS2	
$0018	SCDR	
$0019	SCBR	

001A	INTKBSCR	Регистры модуля KBI08
$001B	INTKBIER	
$001C	TBCR	Регистр базового таймера TBM08
$001D	INTSCR	Регистр прерываний
$001E	CONFIG2	Регистры

$001F	CONFIG1	конфигурации
$0020	T1SC	Регистры таймерного модуля TIM08-1
$0021-22	T1CNTh-l	
$0023-24	T1MODh-l	
$0025	T1SC0	
$0026-27	T1CH0h-l	
$0028	T1SC1	
$0029-2A	T1CH1h-l	
$002B	T2SC	Регистры таймерного модуля TIM08-2
$002C-2D	T2CNTh-l	
$002E-2F	T2MODh-l	
$0030	T2SC0	
$0031-32	T2CH0h-l	
$0033	T2SC1	
$0034-35	T2CH1h-l	
$0036	PCTL	Регистры модуля генератора импульсов CGM08
$0037	PBWC	
$0038-39	PMSh-l	
$003A	PMRS	
$003B	PMDS	
$003C	ADSCR	Регистры модуля ADC08
$003D	ADR	
$003E	ADCLK	

Микроконтроллер *MC*68*HC*908*GP*32 имеет внутреннюю *Flash*-память, содержимое которой может стираться и записываться при работе в режиме отладки или в процессе выполнения прикладной программы. Допускается до 10000 циклов стирания–программирования, время хранения информации составляет более 10 лет. Необходимое для программирования повышенное напряжение обеспечивается внутренним преобразователем, поэтому не требуется подключение внешнего источника. Специальный механизм защиты позволяет предотвратить случайное стирание содержимого *Flash*-памяти. Наличие байтов секретности позволяет предотвратить несанкционированное считывание информации.

На кристалле микроконтроллера содержится 512 байт статической оперативной памяти, ячейки которой имеют адреса в диапазоне $0040–$023*F*. Обычно ОЗУ используется для хранения переменных и реализации стека.

Часть адресного пространства занята ячейками служебного ПЗУ, в котором содержится программа-монитор, которая реализует необходимые процедуры при работе микроконтроллера в режиме отладки, обеспечивая возможность контроля его внутреннего состояния. Это

масочнопрограммируемое ПЗУ, содержимое которого записывается в процессе изготовления микроконтроллера.

В старших позициях адресного пространства располагаются вектора начального запуска и прерываний.

Процессоры семейства *68HC08/908* выполняют прерывание текущей программы по одной из следующих причин:

– поступление команды прерывания *SWI*;

– поступление внешнего запроса прерывания на входе *IRQ#*, обслуживание которого реализуется с помощью модуля *IRQ08;*

– поступление запроса прерывания в контрольной точке, который формируется модулем *BREAK08;*

– поступление внутренних запросов прерывания от периферийных устройств, расположенных на кристалле микроконтроллера.

Эти события вызывают прерывание выполнения текущей программы и переход к подпрограмме - обработчику прерывания, который обеспечивает необходимую реакцию микроконтроллера. При поступлении команды *SWI* или соответствующего запроса процессор завершает выполнение текущей команды и выполняет цикл перехода к обработке прерывания, в процессе которого реализуются следующие операции:

– загрузка в стек содержимого регистров процессора в следующей последовательности: *CCR - A - X - PCh - PCl*, где *PCh, PCl* – старший и младший байты содержимого *PC, X* – значение младшего байта индексного регистра *H:X*;

– загрузка в *PC* 16-разрядного вектора прерывания *Ve* – адреса первой команды обработчика прерываний, соответствующего наступившему событию.

Значение вектора *Ve* выбирается из таблицы векторов прерываний, размещенной в конце адресуемого объема памяти.

Обработчик прерывания должен заканчиваться командой возврата из прерывания *RTI*, которая выбирает из стека и восстанавливает содержимое регистров *PC, X, A, CCR*, обеспечивая продолжение выполнения прерванной программы.

Необходимо иметь в виду, что в цикле перехода к обработке прерываний в стеке автоматически сохраняется только младший байт индексного регистра *X*. Чтобы обеспечить сохранение старшего байта индексного регистра H следует в начале обработчика прерываний ввести команду *PSHH* (загрузка в стек содержимого байта *H*), а перед командой *RTI* выполнить команду *PULH,* которая извлекает из стека и восстанавливает содержимое старшего байта *H* индексного регистра.

Для размещения таблицы векторов прерываний в микроконтроллерах семейства *68HC08/908* зарезервировано 64 байт адресного пространства

(адреса $*FFD0-FF*), из которых в микроконтроллере *MC68HC908GP32* используется 36 байт. В табл. 3.2 показано размещение векторов для этого микроконтроллера.

Нижнюю позицию в таблице векторов занимает вектор начального запуска (*RESET*), а перед ним располагается вектор программного прерывания по команде *SWI*. Выше размещаются вектора аппаратных прерываний, которые вызываются поступлением запроса от внешних устройств на вход *IRQ#*, запросов прерывания от таймера и других модулей микроконтроллера.

Таблица 3.2 – Размещение векторов прерывания для микроконтроллера *MC68HC908GP32*

Размещение векторов прерывания для микроконтроллера *MC68HC908GP32*	
Запрос базового таймера TBM08	$FFDC-DD
Запрос модуля ADC08	$FFDE-DF
Запрос модуля KBI08	$FFE0-E1
Запросы модуля SCI08	$FFE2-E3 $FFE4-E5
Запросы модуля SPI08	$FFE6-E7 $FFE8-E9
Переполнение таймера TIM08-2	$FFEA-EB
Запрос канала 1 таймера TIM08-2	$FFEC-ED
Запрос канала 0 таймера TIM08-2	$FFEE-EF
Переполнение таймера TIM08-1	$FFF0-F1
Запрос канала 1 таймера TIM08-1	$FFF2-F3
Запрос канала 0 таймера TIM08-1	$FFF4-F5
Запрос модуля CGM08	$FFF6-F7
Прерывание по внешнему запросу IRQ#	$FFF8-F9
Команда SWI (программное прерывание)	$FFFA-FB
Вектор начального запуска (Reset)	$FFFC-FD $FFFE-FF

Поступление запросов прерывания проверяется микроконтроллером после выполнения каждой команды программы. Если поступило нескольких запросов, то в первую очередь обслуживается запрос с более высоким приоритетом. Запросы имеют фиксированные приоритеты в соответствии с

порядком их расположения в таблице векторов прерываний: чем больше адрес, тем выше приоритет. Таким образом, из аппаратных прерываний наивысший приоритет имеет внешний запрос *IRQ#*, затем запрос генератора тактирующих импульсов *CGM08* и сигналы таймера. Минимальный приоритет имеют запросы, вектора которых размещены в верхней части (табл. 3.2) – запросы *ADC08* и других модулей. МК *MC68HC908GP32* обладает 33 линиями ввода/вывода данных. Эти линии объединены в 8разрядные параллельные порты, которые именуют в соответствии с буквами латинского алфавита: *Port A, Port B, Port C, Port D, Port H.*

Все линии ввода/вывода МК MC68HC908GP32– двунаправленные, т.е. могут использоваться разработчиком как для ввода данных в МК, так и для вывода логических сигналов. Направление передачи линий ввода/вывода настраивается программно путем записи управляющих слов в регистры специальных функций. Возможно изменение направления передачи в ходе выполнения программы посредством перепрограммирования этих регистров. Сигнал сброса устанавливает все линии в режим ввода. Направление передачи каждой линии может быть выбрано разработчиком произвольно, независимо от других линий, принадлежащих к одному и тому же порту ввода/вывода.

Большинство линий ввода/вывода обладают так называемой альтернативной функцией. Эти линии связаны со встроенными в МК периферийными устройствами, они обеспечивают связь периферийных модулей с «внешним миром». Так, линии порта *Port B* используются для подключения к встроенному АЦП измеряемых напряжений, линии других портов служат линиями ввода/вывода последовательных приемопередатчиков. Если соответствующий периферийный модуль МК не используется, то его выводы можно задействовать как обычные линии ввода/вывода. По способу схемного решения выходного драйвера различают два типа линий ввода/ вывода:

– линии с обычной схемотехникой двунаправленной линии
ввода/вывода;

– двунаправленные линии с программно-подключаемыми в режиме ввода подтягивающими резисторами R_{PULLUP}.

Если порт имеет «обычную» схемотехнику, то для его обслуживания предусмотрены два типа регистров:

– *PTx* - регистр данных порта x, где x - имя порта ввода/вывода; – *DDRx* - регистр направления передачи порта x.

Если порт имеет схемотехнику с программно-подключаемым «подтягивающим» резистором, то для обслуживания порта предусмотрены три регистра:

– *PTx* - регистр данных порта x;
– *DDRx* - регистр направления передачи порта x;
– *PTxPUE* - регистр входного сопротивления порта x.

Так, порт *PortA* микроконтроллера *MC68HC908GP32* обслуживается регистрами *PTA*, *DDRA* и *PTAPUE*. В приложении Г, Д, Е приведен формат регистров специальных функций *PTx, DDRx* и *PTxPUE*. Заметим, что формат регистров *PTx* и *DDRx* для портов с различной схемотехникой полностью совпадает.

Детальный формат всех регистров специальных функций портов ввода/вывода приведен на рис. 3.2.

Адрес	Имя регистра	Формат регистров							
$0000	PTA	PTA7	PTA6	PTA5	PTA4	PTA3	PTA2	PTA1	PTA0
$0001	PTB	PTB7	PTB6	PTB5	PTB4	PTB3	PTB2	PTB1	PTB0
$0002	PTC	0	PTC6	PTC5	PTC4	PTC3	PTC2	PTC1	PTC0
$0003	PTD	PTD7	PTD6	PTD5	PTD4	PTD3	PTD2	PTD1	PTD0
$0008	PTE	0	0	0	0	0	0	PTE1	PTE0
$0004	DDRA	DDRA7	DDRA6	DDRA5	DDRA4	DDRA3	DDRA2	DDRA1	DDRA0
$0005	DDRB	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
$0006	DDRC	0	DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0
$0007	DDRD	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0
$000C	DDRE	0	0	0	0	0	0	DDRE1	DDRE0
$000D	PTAPUE	PTA-PUE7	PTA-PUE6	PTA-PUE5	PTA-PUE4	PTA-PUE3	PTA-PUE2	PTA-PUE1	PTA-PUE0
$000E	PTCPUE	0	PTC-PUE6	PTC-PUE5	PTC-PUE4	PTC-PUE3	PTC-PUE2	PTC-PUE1	PTC-PUE0
$000F	PTDPUE	PTD-PUE7	PTD-PUE6	PTD-PUE5	PTD-PUE4	PTD-PUE3	PTD-PUE2	PTD-PUE1	PTD-PUE0

Рисунок 3.2 Регистры для обслуживания портов ввода/вывода МК *MC*68*HC*908*GP*32

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

Основная:

1. Основы микропроцессорной техники / Ю.В. Новиков, П.К. Скоробогатов. – М.: Интернет – Университет Информационных Технологий, 2003. – 440 с.

2. Микропроцессорные системы: Учеб. пособие для вузов / Е.К. Александров, Р.И. Грушвицкий, М.С. Куприянов и др.; под общ. ред. Д.В. Пузанкова. – СПб.: Политехника, 2002. – 935 с.: ил.

3. Современные микроконтроллеры и микропроцессоры Motorola: Справочник. – М.: Горячая линия – Телеком, 2004. – 952 с.: ил.

4. Работа с микроконтроллерами семейства HC(S)08: Учеб. пособие для студентов технических вузов / Х. Крейдл, Г. Куприс, Т.В. Ремизевич и др.; под ред. Д.И. Панфилова. – М.: Изд-во МЭИ, 2005. – 444 с.: ил.

5. Микроконтроллеры. Разработка встраиваемых приложений / А.Е. Васильев – СПб.: БХВ – Петербург, 2008. – 304 с.: ил.

6. Архитектура микропроцессорных систем / Б.В. Костров, В.Н. Ручкин. – М.: Изд-во Диалог–МИФИ, 2007. – 304 с.: табл. 14, ил. 147.

7. Создаем устройства на микроконтроллерах / А.В. Белов– СПб.: Наука и Техника, 2007. – 304 с.: ил.

Дополнительная:

8. Микроконтроллеры для встраиваемых приложений: от общих подходов – к семействам HC05 и HC08 фирмы Motorola / Под ред. И.С. Кирюхина. – М.: ДОДЭКА, 2000. – 272 с.

9. Руководство по микроконтроллерам / М. Предко. – Т.1. – М.: Постмаркет, 2001. – 411 с.

10. Руководство по микроконтроллерам / М. Предко. – Т.1. – М.: Постмаркет, 2001. — 488 с.

11. Самоучитель по микропроцессорной технике / А.В. Белов. – СПб.: Наука и техника, 2003. – 224 с.

12. Микропроцессорные системы бытовой техники / Б.П. Баев – М.: Легкая промышленность и бытовое обслуживание, 2001. – 464 с.

13. Цифровые устройства и микропроцессорные системы: Учебник для техникумов связи / Б.А. Калабеков. – М.: Горячая линия – Телеком, 2000. – 336 с.: ил.

Приложение 1

Задания на

курсовой проект

Спроектировать управляющее устройство на базе микроконтроллера *MC*68*HC*908*GP*32 семейства *Motorola*. Разработать, описать функциональную и принципиальную электрическую схемы устройства (формат А3). Разработать алгоритм функционирования устройства, блоксхему и программу на языке ассемблера микроконтроллера *MC*68*HC*908*GP*32. Описать процесс отладки управляющей программы в интегрированной среде разработки *WinIDE*.

1. Разработать генератор прямоугольных импульсов. Частота импульсов $f_{вых}$ на выходе генератора в герцах 1, 10 задается переключателями и в двоично-десятичном коде отображается на цифровом индикаторе. Длительность импульса 500мс, 50мс соответственно. Кнопка ПУСК запускает генератор. Кнопка СТОП – выключает.

2. Разработать генератор прямоугольных импульсов. Частота импульсов $f_{вых}$ на выходе генератора в герцах от 5, 25 задается переключателями в двоично-десятичном коде и отображается на цифровом индикаторе. Длительность импульса 100мс, 20мс соответственно. Кнопка ПУСК запускает генератор. Кнопка СТОП – выключает.

3. Разработать генератор прямоугольных импульсов. Частота импульсов $f_{вых}$ на выходе генератора в герцах 10, 100 задается переключателями и в двоично-десятичном коде отображается на цифровом индикаторе. Длительность импульса 50мс и 5мс соответственно. Кнопка ПУСК запускает генератор. Кнопка СТОП – выключает.

4. Разработать генератор прямоугольных импульсов. Частота импульсов $f_{вых}$ на выходе генератора в герцах от 25, 50 задается переключателями в двоично-десятичном коде и отображается на цифровом

индикаторе. Длительность импульса 20мс и 10мс соответственно. Кнопка ПУСК запускает генератор. Кнопка СТОП – выключает.

5. Разработать устройство управления:

– при размыкании одного из 4 двоичных ключей включается звуковой сигнал и мигание светодиода (длительность 1.5 с);

– при замыкании всех ключей входной двоичный код вводится с одного из портов ввода, сохраняется в памяти и выводиться на семисегментный индикатор.

Начинает работать по кнопке *ON*, останов – по кнопке *OFF*.

6. Разработать генератор прямоугольных импульсов (меандра) с частотой $f_{вых}$ = 200 Гц, запустить генератор, нажав кнопку ГЕН; остановить генератор кнопкой ОСТ. Вывести на цифровой индикатор $f_{вых}$.

7. Разработать генератор прямоугольных импульсов (меандра) с частотой $f_{вых}$ = 0,7 кГц. Запустить генератор, нажав кнопку ГЕН; остановить генератор кнопкой ОСТ. Вывести на цифровой индикатор $f_{вых}$.

Для генерирования выходной последовательности импульсов использовать программный метод организации временной задержки.

8. Разработать устройство измерения периода входного сигнала (скважность 2). Измерение начинается при замыкании двоичного ключа. Максимальное время измерения 1с. Отобразить на цифровом индикаторе период сигнала в двоично- десятичном коде и выдать звуковой сигнал.

9. Разработать устройство измерения временного интервала между двумя событиями (включить конвейер, фотодатчик). Вывести на цифровой индикатор число деталей, прошедших через конвейер за время его включения. Фотодатчик формирует одиночный импульс в виде логического нуля при прохождении детали по конвейеру.

10. Разработать устройство для выполнения опроса сигнала на входе с интервалом 90 мс, после тридцатого обнаружения сигнала «логического нуля» отобразить на цифровом индикаторе информацию количества опроса сигнала и выдать звуковой сигнал и включить зеленый светодиод. Опрос начинается при нажатии кнопок ПУСК и НУЛЬ.

11. Разработать устройство для выполнения опроса сигнала на входе с интервалом 50 мс, после пятнадцатого обнаружения сигнала «логической единицы» отобразить на цифровом индикаторе информацию количества опроса сигнала и выдать прерывистый звуковой сигнал. Опрос начинается при нажатии кнопок ПУСК и ЕДИН.

12. Разработать устройство управления информацией. В память программ, начиная с адреса $9000, размещены коды символов текстовой строки: *DEAR FRIEND! WELCOME TO KAZAN!* Переписать символы в массив памяти данных. При нажатии кнопок «ВКЛ» и «ВЫВОД» вывести на

цифровой индикатор коды цифр: 2018. Выдать звуковой сигнал, мигание светодиода.

13. Разработать логическое устройство управления. На линии порта в/в (5 разрядов) поступают цифровые сигналы I_1,I_2,I_3,I_4,I_5 от двоичных датчиков. На цифровой индикатор вывести значения двоичных датчиков и значение функции *I1&I3+I4&I2+I1&I5*. Выдать звуковой сигнал.

14. Разработать логическое устройство управления. На линии порта ввода (6 разрядов) поступают цифровые сигналы I_1,I_2,I_3,I_4,I_5 от двоичных датчиков (двоичные ключи). На цифровой индикатор вывести значения двоичных датчиков и значение функции *I1&I2+I3&I4+I5&I6*. Выдать звуковой сигнал.

15. Разработать устройство измерения длительности одиночного импульса, поступающего на вход микроконтроллера. Информацию о длительности импульса (до 9999мкс) вывести на цифровой индикатор. Запуск измерителя производиться нажатием кнопок «ПУСК» и «ИЗМЕРЕНИЕ».

16. Разработать микроконтроллерное устройство для подсчета числа импульсов, поступающих на вход микроконтроллера. Информацию о количестве импульсов (до 255) вывести на цифровой индикатор. Запуск производиться нажатием кнопок «ПУСК» и «СЧЕТ».

17. Разработать устройство для подсчета числа импульсов, поступающих на вход микроконтроллера за заданный промежуток времени t=300 мкс. Информацию о количестве импульсов (до 255) вывести на цифровой индикатор. Запуск производиться нажатием кнопок «ПУСК» и «СЧЕТ».

18. Разработать устройство управления термостатом по следующему алгоритму:

– определить текущее значение tтек. с датчика температуры;

– вычислить разность между фактическим и стандартным значением tстанд (tфакт – tстанд) = Δt;

– если значение разности Δt > t1, то включить световой и звуковой сигналы;

– если Δt <t1 и t1 Δt > t2 , то включить только световой сигнал; – если Δt < 0, включить нагреватель; – если Δt > 0, включить вентилятор.

– вывести на индикацию текущее значение температуры.

19. Разработать устройство для измерения температуры. Сравнить текущее значение температуры Ттек со стандартным значением Т. Отобразить ее значение на семисегментном индикаторе и восьмиразрядной линейке светодиодов со следующей логикой работы:

– светится первый и седьмой разряды, если коды равны;

– светиться нулевой и шестой разряды, если Т2<Ттек-Т<Т1;

– светиться третий и пятый разряды, если T2<T-Ттек<T1; – светиться второй и третий разряды, если Ттек-T>T1; – светиться четвертый разряд, если T-Ттек>T1.

20. Разработать устройство ввода. Ввести с 12-кнопочной клавиатуры десятичные цифры и сохранить их в ОЗУ микроконтроллера. После сохранения каждого числа выдавать звуковой сигнал с помощью пъезодинамика с заданной частотой f = 20 кГц. Найти среднее арифметическое чисел и вывести на цифровой индикатор. Ввод начать после нажатия кнопки «ВВОД», закончить после нажатия кнопки «СТОП».

21. Разработать устройство ввода. Ввести с 12-кнопочной клавиатуры десятичные цифры и сохранить их в ОЗУ микроконтроллера. После сохранения каждого числа выдавать звуковой сигнал с помощью пъезодинамика с заданной частотой f = 20 Гц. Найти среднее арифметическое чисел и вывести на цифровой индикатор. Ввод начать после нажатия кнопки «ВВОД», закончить после нажатия кнопки «СТОП».

22. Разработать устройство - световой автомат для управления составной светодиодной линейкой из восьми светодиодов. Устройство должно обеспечивать поочередное движение огня в двух направлениях. Переключение должно осуществляться при помощи двоичных ключей ВЛЕВО, ВПРАВО с выводом на цифровой индикатор кодов цифр 000 и *FFF* соответственно. Запуск выполняется кнопкой ПУСК.

23. Разработать устройство – световой автомат для управления составной светодиодной линейкой из восьми светодиодов. Устройство должно обеспечивать включение четных светодиодов, выключение всех, включение нечетных и т.д. с выводом на цифровой индикатор кода *F0F*. Запуск начинается кнопкой ПУСК.

24. Разработать устройство для выполнения опроса сигнала на входе с интервалом 15 мс после десятикратного обнаружения сигнала «логической единицы» вывести на семисегментный индикатор информацию количества опроса сигнала и выдать звуковой сигнал.

25. Разработать устройство для выполнения опроса сигнала на входе с интервалом 10 мс после пятикратного обнаружения сигнала «логической единицы» вывести на цифровой индикатор информацию количества опроса сигнала и выдать звуковой сигнал.

Приложение 2

Федеральное государственное бюджетное образовательное учреждение высшего образования

«КАЗАНСКИЙ ГОСУДАРСТВЕННЫЙ ЭНЕРГЕТИЧЕСКИЙ УНИВЕРСИТЕТ»

КГЭУ

(ФГБОУ ВО «КГЭУ»)

Институт электроэнергетики и электроники

(полное название института)

Промышленная электроника и светотехника

(полное название кафедры)

КУРСОВОЙ ПРОЕКТ

по дисциплине

«Микропроцессорные устройства»

Выполнил:

_____(Ф.И.О.)

студент _____курса группы_____

_____

(подпись)

Руководитель работы:

_____

(ФИО, должность)

Работа выполнена и защищена с оценкой _____ Дата защиты_____ _____

(подпись руководителя)

Члены комиссии: _____

(должность) (подпись) (И.О. Фамилия)

_____

(должность) (подпись) (И.О. Фамилия)

_____

(должность) (подпись) (И.О. Фамилия)

Казань, 2020г.

Приложение 3

СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРА *MC68HC908GP32*

Команды пересылки данных

Мнемокод	Код	Операция	Выполняемое действие	Способ адресации	Формат команды (байты)	Число циклов	Влияние на признаки					
							V	H	I	N	Z	C

LDA LDA #opr LDA opr LDA opr LDA ,x LDA opr,x LDA opr,x LDA opr,SP LDA opr,SP	A6 B6 C6 F6 E6 D6	Загрузить в A константу или содержимое ячейки памяти	A (M)	IMM DIR EXT IX IX1 IX2 SP1 SP2	2 2 3 1 2 3 3 4	2 3 4 2 3 4 4 5	0	–	–			–
LDX LDX #opr LDX opr LDX opr LDX ,x LDX opr,x LDX opr,x LDX opr,SP LDX opr,SP	AE BE CE FE EE DE	Загрузить в регистр X константу или содержимое ячейки памяти	X (M)	IMM DIR EXT IX IX1 IX2 SP1 SP2	2 2 3 1 2 3 3 4	2 3 4 2 3 4 4 5	0	–	–			–
LDHX LDX #opr LDX opr	45 55	Загрузить в регистр H:X двухбайтовую константу или содержимое двух ячеек памяти	X (M)	IMM DIR	3 2	3 4	0	–	–			–
STA STA opr STA opr STA ,x STA opr,x STA opr,x STA opr,SP STA opr,SP	B7 C7 F7 E7 D7	Запомнить содержимое A в ячейке памяти	M (A)	DIR EXT IX IX1 IX2 SP1 SP2	2 3 1 2 3 3 4	3 4 2 3 4 4 5	0	–	–			–
STX STX opr STX opr STX ,x STX opr,x STX opr,x STX opr,SP STX opr,SP	BF CF FF EF DF	Запомнить содержимое регистра X в ячейке памяти	M (X)	DIR EXT IX IX1 IX2 SP1 SP2	2 3 1 2 3 3 4	3 4 2 3 4 4 5	0	–	–			–
STHX STHX opr	35	Запомнить содержимое индексного регистра H:X в двух ячейках памяти	(M:M+$01) (H:X)	DIR	2	4	0	–	–			–
MOV MOV opr1,opr2 MOV #opr1,opr2 MOV opr1,x+ MOV x+,opr2	4E 6E 5E 7E	Переслать данные из одной ячейки памяти в другую	(M)opr2 (M)opr1 (H:X) (H:X)+$01 В двух последних	DIR IMMиDIR DIRиIX+ IX+иDIR	3 3 2 2	5 4 4 4	0	–	–			–
TAX	97	Переслать содержимое A в регистр X	X (A)	INH	1	1	–	–	–	–	–	–
TXA	9F	Переслать содержимое регистра X в A	A (X)	INH	1	1	–	–	–	–	–	–
TAP	84	Переслать содержимое A в регистр признаков CCR	CCR (A)	INH	1	1						
TPA	85	Переслать содержимое регистра признаков CCR A в регистр X	A (CCR)	INH	1	1	–	–	–	–	–	–
TSX	95	Переслать увеличенное на 1 содержимое указателя стека SP в индексный регистр H:X	H:X (SP)+$01	INH	1	2	–	–	–	–	–	–
TXS	94	Переслать уменьшенное на 1 содержимое индексного регистра H:X в указатель стека SP	(SP) (H:X)-$01	INH	1	2	–	–	–	–	–	–

Арифметические команды

Мнемокод	Код	Операция	Выполняемое Действие	Способ Адресации	Формат команды (байты)	Число циклов	Влияние на признаки					
							V	H	I	N	Z	C
ADD ADD #opr ADD opr ADD opr ADD ,x ADD opr,x ADD opr,x ADD opr,SP ADD opr,SP	 AB BB CB FB EB DB	Сложить содержимое аккумулятора A с байтом памяти M (или константой). Результат поместить в A.	A (A)+(M)	 IMM DIR EXT IX IX1 IX2 SP1 SP2	 2 2 3 1 2 3 3 4	 2 3 4 2 3 4 4 5			–			
ADC ADC #opr ADC opr ADC opr ADC ,x ADC opr,x ADC opr,x ADC opr,SP ADC opr,SP	 A9 B9 C9 F9 E9 D9	Сложить содержимое аккумулятора A с байтом памяти M (или константой) и значением бита переноса C. Результат поместить в A.	A (A)+(M)+(C)	 IMM DIR EXT IX IX1 IX2 SP1 SP2	 2 2 3 1 2 3 3 4	 2 3 4 2 3 4 4 5			–			
SUB SUB #opr SUB opr SUB opr SUB ,x SUB opr,x SUB opr,x SUB opr,SP SUB opr,SP	 A0 B0 C0 F0 E0 D0	Вычесть байт данных памяти M (или константу) из содержимого аккумулятора A. Результат поместить в A.	A (A)-(M)	 IMM DIR EXT IX IX1 IX2 SP1 SP2	 2 2 3 1 2 3 3 4	 2 3 4 2 3 4 4 5		–	–			
SBC SBC #opr SBC opr SBC opr SBC ,x SBC opr,x SBC opr,x SBC opr,SP SBC opr,SP	 A2 B2 C2 F2 E2 D2	Вычесть байт данных памяти M (или константу) и бит переноса C из содержимого аккумулятора A. Результат поместить в A.	A (A)-(M)-(C)	 IMM DIR EXT IX IX1 IX2 SP1 SP2	 2 2 3 1 2 3 3 4	 2 3 4 2 3 4 4 5		–	–			
MUL	42	Умножить содержимое A на содержимое регистра X. Произведение представлено в двухбайтовом формате. Старший байт произведения содержится в регистре X, младший – в A.	X:A (X)*(A)	INH	1	5	–	0	–	–	–	0
DIV	52	Разделить двухбайтовое делимое на однобайтовый делитель. Старший байт делимого содержится в регистре H, младший – в A. Однобайтовый делитель находится в регистре X. Целое частное помещается в A, остаток от деления – в регистр H.	A (H:A)/(X) H Remainder	INH	1	7	–	–	–	–		
CMP **CMP #opr** **CMP opr** **CMP opr** **CMP ,x** **CMP opr,x** **CMP opr,x** **CMP opr,SP** **CMP opr,SP**	 A1 B1 C1 F1 E1 D1	Сравнить содержимое аккумулятора A с байтом памяти M (или константой). По результату сравнения установить признаки. Содержимое A и ячейки памяти M после операции не изменяется	(A) – (M)	 IMM DIR EXT IX IX1 IX2 SP1 SP2	 2 2 3 1 2 3 3 4	 2 3 4 2 3 4 4 5		–	–			

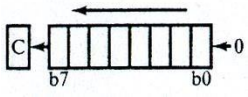
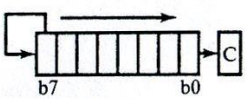
CPX CPX #opr CPX opr CPX opr CPX ,x CPX opr,x CPX opr,x CPX opr,SP CPX opr,SP	 A3 B3 C3 F3 E3 D3	Сравнить содержимое регистра X с байтом памяти M (или константой). По результату сравнения установить признаки. Содержимое регистра X и ячейки памяти M после операции не изменяется	(X) – (M)	IMM DIR EXT IX IX1 IX2 SP1 SP2	2 2 3 1 2 3 3 4	2 3 4 2 3 4 4 5		–	–			
CPHX CPHX #opr CPHX opr	 65 75	Сравнить содержимое индексного регистра H:X с содержимым двух ячеек памяти M:M+1 (или двухбайтовой константой). По результату сравнения установить признаки. Содержимое H:X и ячеек памяти после операции не изменяется	(H:X) – M:M+$0001	IMM DIR	3 2	3 4		–	–			
INC INC opr INCA INCX INC ,x INC opr,x INC opr,SP	 3C 4C 5C 7C 6C	Увеличить на 1 содержимое аккумулятора A, регистра X, ячейки памяти M	(M) – (M)+$01 (A) – (A)+$01 (X) – (X)+$01 (M) – (M)+$01 (M) – (M)+$01 (M) – (M)+$01	DIR INH INH IX IX1 SP1	2 1 1 1 2 3	4 1 1 3 4 5		–	–			
DEC DEC opr DECA DECX DEC ,x DEC opr,x DEC opr,SP	 3A 4A 5A 7A 6A	Уменьшить на 1 содержимое аккумулятора A, регистра X, ячейки памяти M	(M) – (M) (A) – (A) (X) – (X) (M) – (M) (M) – (M) (M) – (M)	DIR INH INH IX IX1 SP1	2 1 1 1 2 3	4 1 1 3 4 5		–	–			
DAA	72	Десятичная коррекция аккумулятора A	$(A)_{10}$	INH	1	2	–	–	–			
AIX AIX #opr	A7	Сложить содержимое регистра H:X с однобайтовой константой, представленной в дополнительном коде со знаком. Результат поместить в H:X	H:X (H:X)+ #opr	IMM	2	2	–	–	–	–	–	–
AIS AIS #opr	AF	Сложить содержимое указателя стека SP и однобайтовой константой, представленной в дополнительном коде со знаком. Результат поместить в SP	SP (SP) +#opr	IMM	2	2	–	–	–	–	–	–

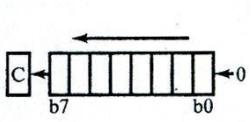
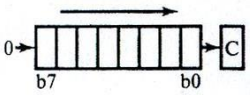
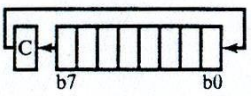
Логические команды

Мнемокод	Код	Операция	Выполняемое Действие	Способ Адресации	Формат команды (байты)	Число циклов	Влияние на признаки					
							V	H	I	N	Z	C
AND AND #opr AND opr AND opr AND ,x AND opr,x AND opr,x AND opr,SP AND opr,SP	 A4 B4 C4 F4 E4 D4	Поразрядное логическое И над содержимым аккумулятора A и байтом данных M. Результат поместить в A	A (A)&(M)	 IMM DIR EXT IX IX1 IX2 SP1 SP2	 2 2 3 1 2 3 3 4	 2 3 4 2 3 4 4 5	0	–	–			–
ORA ORA #opr ORA opr ORA opr ORA ,x ORA opr,x ORA opr,x ORA opr,SP ORA opr,SP	 AA BA CA FA EA DA	Поразрядное логическое ИЛИ над содержимым аккумулятора A и байтом данных M. Результат поместить в A	A (A)˅(M)	 IMM DIR EXT IX IX1 IX2 SP1 SP2	 2 2 3 1 2 3 3 4	 2 3 4 2 3 4 4 5	**0**	–	–			–
EOR EOR #opr EOR opr EOR opr EOR ,x EOR opr,x EOR opr,x EOR opr,SP EOR opr,SP	 A8 B8 C8 F8 E8 D8	Поразрядное исключающее ИЛИ над содержимым аккумулятора A и байтом данных M. Результат поместить в A	A (A) (M)	 IMM DIR EXT IX IX1 IX2 SP1 SP2	 2 2 3 1 2 3 3 4	 2 3 4 2 3 4 4 5	0	–	–			–
COM COM opr COMA COMX COM ,x COM opr,x COM opr,SP	 33 43 53 73 63	Инверсия содержимого аккумулятора A, или регистра X, или ячейки памяти M		 DIR INH INH IX IX1 SP1	 2 1 1 1 2 3	 4 1 1 3 4 5		–	–			
NEG NEG opr NEGA NEGX NEG ,x NEG opr,x NEG opr,SP	 30 40 50 70 60	Получение дополнительного кода содержимого аккумулятора A, или регистра X, или ячейки памяти M	(M) $00 - (M) (A) $00 - (A) (X) $00 - (X) (M) $00 - (M) (M) $00 - (M)	 DIR INH INH IX IX1 SP1	 2 1 1 1 2 3	 4 1 1 3 4 5		–	–			

CLR CLR opr CLRA CLRX CLRH CLR ,x CLR opr,x CLR opr,SP	3F 4F 5F 8C 7F 6F	Очистить (установить в 0) аккумулятор A, или регистр X, или регистр H, или ячейку памяти M	(M) $00 (A) $00 (X) $00 (H) $00 (M) $00 (M) $00 (M) $00	DIR INH INH INH IX IX1 SP1	2 1 1 1 2 3	4 1 1 3 4 5						
BIT BIT #opr BIT opr BIT opr BIT ,x BIT opr,x BIT opr,x BIT opr,SP BIT opr,SP	A5 B5 C5 F5 E5 D5	Поразрядное логическое И над содержимым A и байтом памяти M. Результат операции никуда не записывается. По результату операции устанавливают признаки N и Z	(A)&(M)	IMM DIR EXT IX IX1 IX2 SP1 SP2	2 2 3 1 2 3 3 4	2 3 4 2 3 4 4 5	0	–	–			–
TST TST opr TSTA TSTX TST ,x TST opr,x TST opr,SP	3D 4D 5D 7D 6D	Установить признаки N и Z по содержимому аккумулятора A, или регистра X, или ячейки памяти M. Содержимое последних не изменяется	(A) - $00 (X) - $00 (M) - $00	DIR INH INH IX IX1 SP1	2 1 1 1 2 3	4 1 1 2 3 4	0	–	–			–
NSA	62	Поменять местами полубайты аккумулятора A	(A) (A[3:0]:A[7:4])	INH	1	1	–	–	–	–	–	–

Команды сдвига

Мнемокод	Код	Операция	Выполняемое Действие	Способ адресации	Формат команды (байт)	Число циклов	Влияние на признаки					
							V	H	I	N	Z	C
ASL ASL opr ASLA ASLX ASL ,x ASL opr,x ASL opr,SP	38 48 58 78 68	Сдвиг влево содержимого аккумулятора A, или регистра X,или ячейки памяти M. В бит b0 загружается 0, бит b7 загружается в бит C		DIR INH INH IX IX1 SP1	2 1 1 1 2 3	4 1 1 3 4 5		–	–			
ASR ASR opr ASRA ASRX ASR ,x ASR opr,x ASR opr,SP	37 47 57 77 67	Сдвиг вправо содержимого аккумулятора A, или регистра X, или байта памяти M. Бит b7 не изменяется, бит b0 загружается в бит переноса C		DIR INH INH IX IX1 SP1	2 1 1 1 2 3	4 1 1 3 4 5		–	–			

LSL LSL opr LSLA LSLX LSL ,x LSL opr,x LSL opr,SP		Сдвиг влево содержимого аккумулятора A, или регистра X, или ячейки памяти M. В бит b0 загружается 0, бит b7 загружается в бит переноса C. (Аналог ASL)		DIR INH INH IX IX1 SP1	2 1 1 1 2 3	4 1 1 3 4 5						
LSR LSR opr LSRA LSRX LSR ,x LSR opr,x LSR opr,SP	34 44 54 74 64	Сдвиг вправо содержимого аккумулятора A, или регистра X, или ячейки памяти M. В бит b7 загружается 0, бит b0 загружается в бит переноса C		DIR INH INH IX IX1 SP1	2 1 1 1 2 3	4 1 1 3 4 5		–	–			
ROL ROL opr ROLA ROLX ROL ,x ROL opr,x ROL opr,SP	39 49 59 79 69	Циклический сдвиг влево содержимого аккумулятора A, или регистра X, или ячейки памяти M через бит переноса C		DIR INH INH IX IX1 SP1	2 1 1 1 2 3	4 1 1 3 4 5		–	–			

Команды установки битов

Мнемокод	Код	Операция	Выполняемое действие	Способ адресации	Формат команды (байт)	Число циклов	Влияние на признаки					
							V	H	I	N	Z	C
BSET BSET 0, opr BSET 1, opr BSET 2, opr BSET 3, opr BSET 4, opr BSET 5, opr BSET 6, opr BSET 7, opr	10 12 14 16 18 1A 1C 1E	Установить в 1 разряд с номером n в байте данных. Данные могут располагаться в ячейке ОЗУ или регистре специальных функций. В команде используется только прямая адресация. Диапазон адресов байтов данных opr = $00 ÷ $FF	Mn←1	DIR	2	4	–	–	–	–	–	–

BCLR BCLR 0, opr BCLR 1, opr BCLR 2, opr BCLR 3, opr BCLR 4, opr BCLR 5, opr BCLR 6, opr BCLR 7, opr	11 13 15 17 19 1B 1D 1F	Установить в 0 разряд с номером n в байте данных. Данные могут располагаться в ячейке ОЗУ или регистре специальных функций. В команде используется только прямая адресация. Диапазон адресов байтов данных opr = $00 ÷ $FF	Mn←0	DIR	2	4	–	–	–	–	–	–

Команды установки флагов

Мнемокод	Код	Операция	Выполняемое Действие	Способ адресации	Формат команды (байт)	Число циклов	Влияние на признаки					
							V	H	I	N	Z	C
SEC	99	Установить в 1 флаг переноса C	C←1	INH	1	1	–	–	–	–	–	1
CLC	98	Установить в 0 флаг переноса C	C←0	INH	1	1	–	–	–	–	–	0
SEI	9B	Установить в 1 глобальную маску прерываний I. Запретить прерывания	I←1	INH	1	2	–	–	1	–	–	–
CLI	9A	Установить в 0 глобальную маску прерываний I. Разрешить Прерывания	I←0	INH	1	2	–	–	0	–	–	1

Команды безусловного перехода

Мнемокод	Код	Операция	Выполняемое действие	Способ адресации	Формат команды (байт)	Число циклов	Влияние на признаки					
							V	H	I	N	Z	C

JMP JMP opr JMP opr JMP opr,x JMP opr,x JMP ,x	BC CC EC DC FC	Безусловный переход по адресу, указанному в ячейке памяти (ОЗУ или ПЗУ). Адрес ячейки задан используемым в команде способом адресации	PC←Jump Address	DIR EXT IX2 IX1 IX	2 3 3 2 1	2 3 4 3 2	–	–	–	–	–	–
BRA BRA rel	20	Безусловный переход по адресу, код смещения которого указан во втором байте команды	PC←PC+$0002+rel	REL	2	3	–	–	–	–	–	–
BRN BRN rel	21	Перейти к следующей команде. Операция эквивалентна двум инструкциям NOP. Полезна в режиме отладки в абсолютном коде для замены инструкций условного перехода без изменения абсолютных адресов	PC←PC+$0002	REL	2	3	–	–	–	–	–	–
NOP	9D	Пустая операция. Счетчик команд PC увеличивается на 1. Другие регистры не изменяются	PC←PC+1	INH	1	1	–	–	–	–	–	–

Команды условного перехода

Мнемокод	Код	Операция	Выполняемое действие	Способ адресации	Формат команды (байт)	Число циклов	Влияние на признаки					
							V	H	I	N	Z	C
BCS BCS rel		Перейти по метке, если бит переноса C установлен. Иначе перейти к следующей команде	PC←PC+$0002+rel&(C)	REL	2	3	–	–	–	–	–	–
BCC BCC rel		Перейти по метке, если бит переноса C сброшен. Иначе перейти к следующей команде	PC←PC+$0002+rel& (C)	REL	2	3	–	–	–	–	–	–
BEQ BEQ rel	27	Перейти по метке, если r = m. Иначе перейти к следующей команде.	PC←PC+$0002+rel&(Z)	REL	2	3	–	–	–	–	–	–
BNE BNE rel	26	Перейти по метке, если r ≠ m. Иначе перейти к следующей команде.	PC←PC+$0002+rel&(Z)	REL	2	3	–	–	–	–	–	–
BNCS BNCS rel	29	Перейти по метке, если бит дополнительного переноса H установлен. Иначе перейти к следующей команде	PC←PC+$0002+rel&(H)	REL	2	3	–	–	–	–	–	–
BHCC BHCC rel	28	Перейти по метке, если бит дополнительного переноса H сброшен. Иначе перейти к следующей команде.	PC←PC+$0002+rel&(H)	REL	2	3	–	–	–	–	–	–

CBEQ CBEQ opr,rel CBEQA opr,rel CBEQX opr,rel CBEQ opr,x+ CBEQ x+,rel	31 41 51 61 71	Сравнить содержимое аккумулятора A с содержимым ячейки памяти M (или константой) и перей если они равны	PC←(PC)+$0003 +rel, если (A)-(M)=$00 PC←(PC)+$0003 +rel, если (A)-(#opr)=$00 PC←(PC)+$0003 +rel, если (X)-(#opr)=$00 PC←(PC)+$0003 +rel, если (A)-(M)=$00 PC←(PC)+$0002 +rel,	DIR IMM IMM IX1+ IX+ SP1	3 3 3 3 2 4	5 3 3 5 4 6	–	–	–	–	–	–
DBNZ DBNZA opr,rel DBNZA rel DBNZX rel DBNZ opr,x,rel DBNZ x,rel DBNZ opr,SP,rel	3B 4B 5B 6B7 B	Вычесть 1 из содержимого аккумулятора A, или регистра X, или ячейки памяти M и перейти по метке, если результат не равен 0.	A←(A)-$01, или M←(M)-$01,или X←(X)--$01, PC←(PC)+$0003 +rel, если (result)≠ 0 для DBNZ DIR и IX1. PC←(PC)+$0002 +rel, если (result) ≠0 для DBNZA, DBNZX и IX. PC←(PC)+$0004 +rel, если (result) ≠ 0 для DBNZ SP1	DIR IMM IMM IX1 IX SP	3 2 2 3 2 4	5 3 3 5 4 6	–	–	–	–	–	–

Мнемокод	Код	Операция	Выполняемое действие	Способ адреса-ции	Формат Коман-ды (байт)	Число циклов	Влияние на признаки					
							V	H	I	N	Z	C
BIH BIH rel	2F	Перейти по метке, если на входе IRQ высокий логический уровень. Иначе перейти к следующей команде.	PC←(PC)+$0002+ rel&IRQ	REL	2	3	–	–	–	–	–	–

BIL BIL rel	2E	Перейти по метке, если на входе IRQ низкий логический уровень. Иначе перейти к следующей команде	PC←(PC)+$0002+ rel&IRQ	REL	2	3	–	–	–	–	–	–
BMS BMS rel	2D	Перейти по метке, если флаг маски I переведён на низкий логический уровень. Иначе перейти к следующей команде. Условие I=1.	PC←(PC)+$0002+ rel&(I)	REL	2	3	–	–	–	–	–	–
BMC BMC rel	2C	Перейти по метке, если флаг маски I переведен на низкий логический уровень. Иначе перейти к следующей команде. Условие I=0	PC←(PC)+$0002+ rel&(I)	REL	2	3	–	–	–	–	–	–
BRSET BRSET n,opr,rel	00, 02, … 0C, 0E	Перейти по указанному адресу, если бит n в байте данных установлен. Иначе перейти к следующей команде. Для указания байта данных используется только прямая адресация. Диапазон адресов $00÷$FF	PC←(PC)+$0003+ rel&(Mn)	REL	2	3	–	–	–	–	–	–
BRCLR BRCLRn, opr,rel	01, 03, … 0D, 0F	Перейти по указанному адресу, если бит n в байте данных равен 0. Иначе перейти к следующей команде. Для указания байта данных используется только прямая адресация. Диапазон адресов $00÷$FF	PC←(PC)+$0003+ rel&(Mn)	REL	2	3	–	–	–	–	–	–

Мнемокод	Код	Операция	Выполняемое действие	Способ адреса-ции	Формат коман-ды (байт)	Число циклов	Влияние на признаки					
							V	H	I	N	Z	C
BHS BHS rel	24	Перейти по метке если r≥m. Иначе перейти к следующей команде.	PC←(PC)+$0002+rel&C)	REL	2	3	–	–	–	–	–	–
BLO BLO rel	25	Перейти по метке если r<m. Иначе перейти к следующей команде.	PC←(PC)+$0002+rel&C)	REL	2	3	–	–	–	–	–	–
BLS BLS rel	23	Перейти по метке если r≤m. Иначе перейти к	PC←(PC)+$000	REL	2	3	–	–	–	–	–	–

		следующей команде.	2+rel&(C)vZ)]									
BPL BPL rel	2A	Перейти по метке, если бит знака установлен в 0, т.е число положительно. Иначе перейти к следующей команде.	C←(PC)+$0002 +rel&N)	REL	2	3	–	–	–	–	–	–
BMI BMI rel	2B	Перейти по метке, если бит знака установлен в 1, т.е число отрецательно. Иначе перейти к следующей команде.	PC←(PC)+$000 2+rel&N)	REL	2	3	–	–	–	–	–	–
BGE BGE rel	90	Перейти по метке, если r≥m. Иначе перейти к следующей команде.	⊕ PC←PC+ $0002 +rel V)	REL	2	3	–	–	–	–	–	–
BGT BGT rel	92	Перейти по метке если r>m. Иначе перейти к следующей команде.	PC←(PC)+$000 ⊕ 2+rel& [(Z)·(N V)]	REL	2	3	–	–	–	–	–	–
BLE BLE rel	93	Перейти по метке если r≤m. Иначе перейти к следующей команде.	PC←PC+$0002 ⊕ +rel& [(Z)vN V)]	REL	2	3	–	–	–	–	–	–
BLT BLT rel	91	Перейти по метке если r<m. Иначе перейти к следующей команде.	⊕ PC←PC+$0002 +rel&N V)	REL	2	3	–	–	–	–	–	–

Команды работы со стеком, вызова подпрограмм и обслуживания прерываний

Мнемокод	Операция	Выполняемое действие	Способ адресации	Формат команды	Число циклов	V	H	I	N	Z	C
JSR JSR opr JSR opr JSR opr,x JSR opr,x JSR ,x	Вызов подпрограммы. Адрес подпрограммы хранится в ячейке памяти (ОЗУ или ПЗУ). Адрес ячейки задан используемым в команде способом адресации.	PC=PC+ n, где n = 1, 2, 3 в зависимости от способа адресации. (SP) ←PCL SP = SP – 1 (SP) ←PCH SP = SP – 1 PC ← адрес подпрограммы	DIR EXT IX2 IX1 IX	2 3 3 2 1	4 5 6 5 4	--	--	-	--	--	--
BSR BSR rel	Вызов подпрограммы, записанный по адресу, код смещения которого указан во втором байте команды.	PC = PC + 002, (SP) ←PCL SP = SP – 1 (SP) ←PCH SP = SP – 1 PC =PC + rel	REL	2	4	--	--	-	--	--	--
SWI	Программное прерывание. Счетчик команд загружается вектором программного прерывания из ячеек памяти $FFFC и $FFFD	PC = PC + 1, (SP) ←PCL, SP = SP – 1 (SP) ←PCH, SP ← SP – 1, (SP) ← X, SP = SP – 1, (SP) ← A, SP = SP – 1 (SP) ← CR, SP = SP – 1, I = 1 PCH ← ($FFFC), PCH ← ($FFFD)	REL	1	9	--	--	-	--	--	--

RTS	Возврат из подпрограммы. Адрес возврата загружается из стека в счетчик команд PC	SP = SP + 1 PCH ← (SP) SP = SP + 1 PCL ← (SP)	INH	1	4	- -	- -	-	- -	- -	- -
RTI	Возврат из прерывания. Восстанавливается содержимое регистров CPU и счетчика команд PC	SP = SP + 1 CCR ← (CP) SP = SP + 1 A ← (SP) SP = SP + 1 X ← (SP) SP = SP + 1 PCH ← (SP) SP = SP + 1 PCH ← (SP)	INH	1	7	↕	↕	↕	↕	↕	↕
RSP	Установить регистр – указатель стека в состояние $FF	SP = $FF	INH	1	1	--	- -	-	- -	- -	- -
PSHA	Загрузить аккумулятор A в стек	Push (A) SP = SP - $0001	INH	1	2	--	--	-	--	--	--
PSHH	Загрузить старший байт индексного регистра H в стек	Push (H) SP = SP - $0001	INH	1	2	- -	- -	-	- -	- -	- -
PSHX	Загрузить младший байт индексного регистра H в сек	Push (X) SP = SP - $0001	INH	1	2	- -	- -	-	- -	- -	- -
PULA	Восстановить аккумулятор A из стека	SP = (SP) + $0001 Pull(A)	INH	1	2	--	--	-	--	--	--
PULH	Восстановить старший байт индексного регистра H из стека	SP = (SP) + $0001 Pull(H)	INH	1	2	- -	- -	-	- -	- -	- -
PULX	Восстановить младший байт индексного регистра H из стека	SP = (SP) + $0001 Pull(X)	INH	1	2	- -	- -	-	- -	- -	- -

В таблице приняты следующие обозначения:

A – аккумулятор;
H:X – индексный регистр;
M – ячейка памяти;
PC – программный счетчик;
CCR – регистр флагов;
SP – стек;
IMM – непосредственная адресация;
DIR – прямая адресация (8 – ми разрядный адрес операнда);
EXT – прямая адресация (16 – ти разрядный адрес операнда);
IX – индексная адресация;
IX1 – индексная адресация с 8 – ми разрядным смещением;
IX2 – индексная адресация с 16 – ти разрядным смещением; *SP1* – индексная адресация по указателю стека с 8 – ми разрядным смещением;
SP2 – индексная адресация по указателю стека с 16 – ти разрядным смещением;
REL – относительная адресация;

IX+ – индексная адресация с пост – инкрементом;

IX1+ – индексная адресация со смещением и пост - инкрементом.