Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

ИНТЕЛЛЕКТУАЛЬНЫЕ ЭЛЕКТРОННЫЕ СИСТЕМЫ БЕЗОПАСНОСТИ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

В двух частях

Часть 2

В. М. Логин, О. Ч. Ролич

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве пособия для специальности 1-39 03 01 «Электронные системы безопасности»

Минск БГУИР 2020

УДК 004.056(076.5)
ББК 32.972.5я73
И73

Р е ц е н з е н т ы:

кафедра технологий и организации технического сервиса учреждения образования «Белорусский государственный аграрный технический университет» (протокол №1 от 29.08.2019);

доцент кафедры робототехнических систем Белорусского национального технического университета кандидат технических наук, доцент Р. В. Новичихин

И73 **Интеллектуальные** электронные системы безопасности. Лабораторный практикум. В 2 ч. Ч. 2 : Программирование микроконтроллеров : пособие / В. М. Логин, О. Ч. Ролич. – Минск : БГУИР, 2020. – 72 с. : ил.

ISBN 978-985-543-559-5 (ч. 2).

Излагается теоретический материал по 8-разрядным микроконтроллерам семейства MC68HC11 фирмы Motorola и приводится описание четырех лабораторных работ. Первая работа посвящена изучению методов адресации и команд пересылки данных, вторая – изучению арифметических команд. В третьей лабораторной работе рассматриваются логические команды, команды работы с битовыми полями и команды сдвигов. В четвертой лабораторной работе изучаются команды передачи управления и специальные команды.

Часть 1-я издана в БГУИР в 2014 году (авторы: В. М. Логин, И. Н. Цырельчук, О. Ч. Ролич).

УДК 004.056(076.5)
ББК 32.972.5я73

ISBN 978-985-543-559-5 (ч. 2)
ISBN 978-985-543-024-8

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современных условиях многофункциональные интеллектуальные системы безопасности, построенные на IP- и IT-технологиях, становятся наиболее востребованными, вытесняя традиционные системы. Такие высокотехнологичные решения, как «Умный дом», «Безопасный город», «Безопасный транспорт», принцип «все в одном», предназначены для самых взыскательных заказчиков и объектов жизненно важной инфраструктуры.

Такие инновационные системы используют уникальные технологии видеоаналитики, включая захват и распознавание лиц и номерных знаков автомобильных средств, мониторинг транспортных потоков, специализированные решения для контроля за кассовыми терминалами и банкоматами. Значительная их часть использует беспроводные каналы связи, в том числе спутниковые и каналы мобильной связи.

В настоящее время широко применяются технические решения компании Motorola, такие как телефоны серий Iridium и Satellite, которые могут служить как спутниковым, так и мобильным сотовым телефоном, работающим через наземную сеть. В крупном городе за счет использования сменных картриджей, разработанных для основных стандартов сотовой связи, телефонный аппарат Motorola можно использовать как сотовый. За пределами сотового покрытия антенна телефона соединяется со спутниковой группировкой, гарантируя глобальное покрытие и устойчивый сигнал. Основой таких многофункциональных телефонов является микроконтроллер.

Данный курс лабораторных работ предназначен для получения начальных практических навыков работы с микроконтроллерами семейства MC68HC11 фирмы Motorola и ставит своей целью помочь студентам развить навыки программирования микроконтроллеров, необходимые для успешного усвоения теоретических сведений параллельных курсов, связанных с цифровыми устройствами, а также для решения практических задач в ходе дипломного проектирования. Пособие составлено так, что совершенствование прикладных учебных программ не вызывает необходимости внесения изменений в его текст. Курс лабораторных работ предполагается проводить с использованием симулятора-отладчика Micro-IDE фирмы-производителя BiPOM Electronics. Перед выполнением лабораторных работ студентам необходимо ознакомиться с описанием микроконтроллеров семейства MC68HC11 и программы-отладчика Micro-IDE.

Затем необходимо произвести стандартную последовательность действий:

1) создание программы в редакторе;

2) ассемблирование программы и исправление ошибок;

3) запуск программы на выполнение;

4) отладка программы.

В разделе «Запуск и начальная настройка среды» иллюстрируются основные приемы выполнения этих действий.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ И МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Микроконтроллеры фирмы Motorola

Важной особенностью микроконтроллеров (МК) фирмы Motorola является их высокое качество и надежность. Эта особенность, а также наличие широкого набора программных и аппаратных средств поддержки разработки, доступных как от фирмы Motorola, так и от множества других фирм; подробная справочная литература и информация о применениях, в том числе доступная по сети Интернет; возможность получения бесплатных технических консультаций и программного обеспечения, другие преимущества использования продукции, ставшей фактически промышленным стандартом, обеспечивают фирме Motorola нахождение в первой пятерке по продажам микроконтроллеров в мире в течение ряда последних лет.

Микроконтроллеры фирмы Motorola имеют недостаток – малое количество разновидностей однократно программируемых кристаллов, что ограничивает их применение в мелкосерийном производстве.

Каждый из выпускаемых фирмой Motorola микроконтроллеров относится к одной из относительно крупных групп или семейств, краткая характеристика которых приведена далее.

Семейство HC05 содержит около 180 модификаций микроконтроллеров. Поскольку оно в немалой степени формировалось крупными потребителями фирмы Motorola, заказывавшими разработку микроконтроллеров нужной конфигурации под свою конкретную продукцию, то его иногда называют семейством «заказных» микроконтроллеров.

Областями применения микроконтроллеров семейства HC05 являются самые разнообразные устройства связи, автомобильной и бытовой электроники, промышленного управления, компьютерной периферии.

Все микроконтроллеры этого семейства имеют одинаковое 8-разрядное процессорное ядро, основанное на популярной процессорной архитектуре 6800, и отличаются набором периферийных функций. Это означает, что применение любого микроконтроллера дает пользователю возможность использовать приобретенный опыт при создании новых устройств как с применением других микроконтроллеров из семейства HC05, так и на основе более производительного, но программно-совместимого семейства HC08.

В состав микроконтроллеров семейства HC05 входят программируемые запоминающие устройства (ПЗУ) всех типов, оперативно запоминающие устройства (ОЗУ), таймеры, аналого-цифровые преобразователи (АЦП), широтно-импульсный модулятор (ШИМ), контроллеры жидкокристаллических индикаторов (ЖКИ) и других дисплеев, последовательные интерфейсы и многие другие устройства. Все представители семейства HC05 имеют версии с пониженным питанием и расширенным температурным диапазоном, и выпускаются в самых разнообразных корпусах.

Семейство HC08 – следующий шаг в развитии заказных микроконтролле-

ров фирмы Motorola для массовых приложений. Оно характеризуется повышенной в 5–10 раз производительностью процессорного ядра, совместимого по системе команд с центральным процессором (ЦПУ) HC05. Семейство HC08 поддерживает эффективные дополнительные команды и методы адресации, а также такие новые функции, как прямой доступ к памяти, технология нечеткой логики и элементы цифровой обработки сигналов.

При этом полностью статическое процессорное ядро оптимизировано для работы с пониженным напряжением питания и позволяет гибко управлять потреблением с помощью встроенного синтезатора тактовой частоты. HC08 является первым 8-разрядным семейством с определяемой пользователем архитектурой на базе набора стандартных модулей, что значительно ускоряет цикл разработки нового заказного микроконтроллера.

Семейство HC08 включает в себя различные типы ПЗУ и ОЗУ, таймеры, последовательные интерфейсы, АЦП, контроллер ЖКИ, контроллер прямого доступа к памяти (ПДП), силовые и высоковольтные ключи и т. д. В настоящее время в состав семейства входят около 20 моделей.

Семейство MC68HC11 содержит около 40 универсальных высокопроизводительных микроконтроллеров. Процессорное ядро этих микроконтроллеров отличается от семейства HC05 повышенной производительностью, более эффективной архитектурой, системой команд, наличием дополнительных методов адресации и возможностью адресовать больший объем внешней памяти. Микроконтроллеры семейства HC11 содержат встроенную память различных типов и конфигураций.

Периферийные функции микроконтроллеров фирмы Motorola представлены многофункциональными таймерами, АЦП (до 12 каналов и 10 разрядов), встроенным сопроцессором, ускоряющим выполнение умножения и деления, ШИМ и ЦАП; последовательными интерфейсами, контроллером ПДП, синтезатором тактовой частоты и др. Как и в других семействах, имеется большое разнообразие корпусов, а также версии с пониженным напряжением питания и расширенным температурным диапазоном.

Наиболее характерные особенности ЦПУ:

- два 8-битовых или один 16-битовый аккумулятор;
- два 16-битовых индексных регистра;
- два программно-управляемых режима пониженного энергопотребления;
- операции умножения 88 и деления 16/16;
- внутренняя тактовая частота до 4 МГц.

ЦПУ некоторых моделей семейства содержит встроенный математический сопроцессор, выполняющий 16-битовые операции умножения и деления в 10 раз быстрее, чем процессор. Существуют версии МК с программно-управляемым значением тактовой частоты на основе ФАПЧ, что позволяет гибко управлять энергопотреблением в зависимости от сложности вычислительных задач. ЦПУ семейства HC11 поддерживает следующие режимы адресации: неявная, непосредственная, прямая, расширенная, индексная и относительная.

Система команд представлена следующими группами:

- команды пересылки данных, связанные с аккумуляторами;
- команды пересылки для стека и индексных регистров;
- команды переходов по условиям и состояниям битов;
- арифметические команды;
- логические команды;
- команды работы с битами;
- специальные команды.

МК семейства HC11 имеют в своем составе все типы внутренней памяти: ПЗУ (программируемое или масочное), EEPROM, ОЗУ объемом до 2 Кбайт. Характерно, что все МК семейства HC11 адресуют внешнюю память, причем есть версии с немультиплексированными магистралями данных и адресами (например, подсемейство HC11F1), а также версии с расширенным до 256 Кбайт…1 Мбайта адресным пространством с помощью программируемых выборок внешней памяти (HC11Kx).

Микроконтроллеры фирмы Motorola

Более подробно рассмотрим общую структуру микроконтроллера MC68HC908GP32 как «классического» представителя описанной линейки семейств микроконтроллеров.

Данный МК содержит 8-разрядный процессор CPU08, Flash-память емкостью 32 Кбайта, ОЗУ данных емкостью 512 байт и большой набор служебных и периферийных модулей (рис. 1.1). Процессор CPU08 выполняет обработку 8-разрядных операндов и реализует набор из 90 команд. Он содержит пять программно-доступных регистров: 8-разрядный аккумулятор A и регистр признаков CCR, 16-разрядный индексный регистр H:X, указатель стека SP и программный счетчик PC.

В состав служебных модулей входят:

- генератор тактовых импульсов CGM08;
- модуль системной интеграции SIM08;
- модуль контроля напряжения питания LVI08;
- модуль прерывания в контрольной точке BREAK08;
- модуль управления внешним прерыванием IRQ08;
- сторожевой таймер COP08;
- базовый таймер TBM08.

Модуль генератора импульсов CGM08 генерирует импульсные сигналы, на базе которых модуль системной интеграции SIM08 формирует тактовые импульсы. Выходные сигналы модуля CGM08 определяют частоту тактовых импульсов для работы процессора и периферийных модулей.

Модуль системной интеграции SIM08 выполняет ряд функций, обеспечивающих совместную работу различных модулей микроконтроллера. Он работает совместно с другими служебными модулями: CGM08, LVI08, IRQ08, BREAK08, COP08, выполняя формирование тактовых импульсов, запуск микроконтроллера, организацию обслуживания прерываний.

Модуль прерывания в контрольной точке BREAK08 обеспечивает останов выполнения программы в заданной контрольной точке и используется в процессе отладки программного обеспечения.

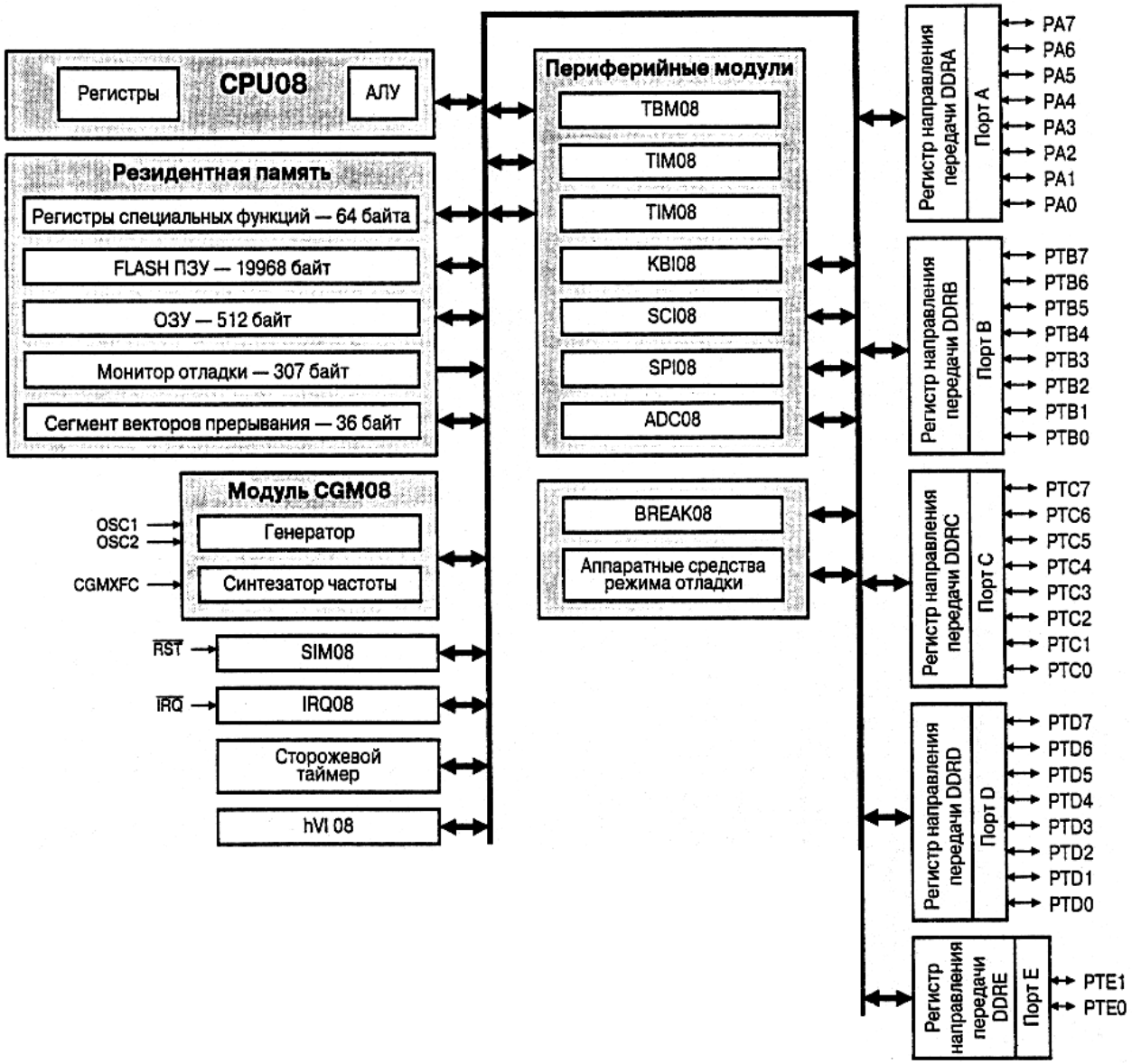

Рис. 1.1

Модуль управления внешним прерыванием IRQ08 принимает внешний запрос прерывания, поступающий на вход IRQ#, и обеспечивает различные варианты его обслуживания.

Сторожевой таймер COP08 осуществляет контроль выполнения текущей программы.

Модуль LVI08 вырабатывает сигнал перезапуска микроконтроллера при снижении его напряжения питания ниже порогового уровня.

Модуль базового таймера TBM08 обеспечивает периодическое формирование запросов прерывания.

Периферийные модули осуществляют обмен данными и совместную работу микроконтроллера с другими устройствами, входящими в состав системы управления. Микроконтроллер MC68HC908GP32 содержит следующие периферийные модули:

- пять параллельных портов A, B, C, D, E для ввода-вывода данных;
- асинхронный последовательный порт SCI08;
- синхронный последовательный порт SPI08;
- модуль контроля клавиатуры KBI08;
- 8-разрядный аналого-цифровой преобразователь ADC08;
- два таймерных модуля TIM08.

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

Выводы параллельных портов A, B, D, E совмещены с выводами других периферийных модулей – KBI08, ADC08, TIM08-1, TIM08-2, SPI08, SCI08 (см. 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 SCI08, SPI08 реализуют соответственно последовательный асинхронный и синхронный обмен данными между микроконтроллером и внешними устройствами.

Таймерный модуль TIM08 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 ADC08 производит преобразование значения потенциала, поступающего на один из восьми аналоговых входов, в 8-разрядное двоичное число.

Модуль контроля клавиатуры KBI08 обеспечивает формирование запроса прерывания при поступлении сигнала на определенные входы параллельных портов, обычно используемые для подключения клавиатуры.

Организация и адресация памяти

Микроконтроллеры семейства 68HC08/908 адресуют 64 Кбайта внутренней памяти (адреса $0000…FFFF). Распределение адресного пространства задается картой памяти, вид которой определяется объемом внутренней памяти и набором периферийных устройств, входящих в состав данной модели микроконтроллера. Далее приведена карта памяти для микроконтроллеров MC68HC908GP32. 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 $003F	Регистры периферийных и служебных модулей (64 байта)
$0040 $023F	ОЗУ данных (512 байт)
$0080 $7FFF	Не используется (32 192 байта)
$8000 $FDFF	Flash-память (32 256 байт)
$FE00	Регистр SBSR (модуль BREAK08)
$FE01	Регистр SRSR (указывает причину запуска)
$FE02	Резервировано
$FE03	Регистр SBFCR (модуль BREAK08)
$FE04	Регистр INT1 (запросы прерывания)
$FE05	Регистр INT2 (запросы прерывания)
$FE06	Регистр INT3 (запросы прерывания)
$FE07	Резервировано
$FE08	Регистр FLCR (управление Flash-памятью)
$FE09	Регистр BRKh (модуль BREAK08)
$FE0A	Регистр BRKl (модуль BREAK08)
$FE0B	Регистр BRKSCR (модуль BREAK08)
$FE0C	Регистр LVISR (модуль LVI08)
$FE0D $FE1F	Не используются (19 байт)
$FE20 $FE52	ПЗУ – монитор отладки (307 байт)
$FE53 $FF7D	Не используются (43 байта)
$FF7E	Регистр FLBPR (управление Flash-памятью)
$FF7F $FFDB	Не используются (93 байта)
$FFDC $FFFF	Векторы запуска и прерываний (36 байт)

Младшие 64 позиции адресного пространства (адреса $000…$003F) занимают регистры служебных и периферийных модулей (табл. 1.1). 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 (запуск) содержимое SP принимает значение $00FF, 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.

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
$0009-0B		Не используются
$000C	DDRE	Регистр направления передачи порта E
$000D	PTAPUE	Регистры управления портами A, C, D
$000E	PTCPUE	
$000F	PTDPUE	
$0010	SPCR	Регистры синхронного порта SPI08
$0011	SPSCR	
$0012	SPDR	

Продолжение табл. 1.1

Адрес	Регистр	Назначение
$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 конфигурации
$001F	CONFIG1	
$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	

Окончание табл. 1.1

Адрес	Регистр	Назначение
$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 MC68HC908GP32 имеет внутреннюю Flash-память, содержимое которой может стираться и записываться при работе в режиме отладки или в процессе выполнения прикладной программы. Допускается до 10 000 циклов стирания программирования, время хранения информации составляет более 10 лет. Необходимое для программирования повышенное напряжение обеспечивается внутренним преобразователем, поэтому подключение внешнего источника не требуется. 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…$023F. Обычно ОЗУ используется для хранения переменных и реализации стека.

Часть адресного пространства занята ячейками служебного ПЗУ, в котором содержится программа-монитор, реализующая необходимые процедуры при

работе микроконтроллера в режиме отладки и обеспечивающая при этом возможность контроля его внутреннего состояния. Это масочно-программируемое ПЗУ, содержимое которого записывается в процессе изготовления микроконтроллера.

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

Генерация тактовых импульсов и запуск микроконтроллера

Тактовые импульсы с частотой F_t формируются модулем системной интеграции SIM08 на базе импульсных сигналов, которые генерируются модулем CGM08. В микроконтроллерах MC68HC908GP32 этот модуль 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). Для получения тактовых сигналов заданной частоты F_t необходимо выполнить программирование модуля CGM08.

Запуск микроконтроллера выполняется с помощью модуля системной интеграции SIM08. Он автоматически производится в следующих случаях:

1) включение напряжения питания $V_п$;

2) поступление внешнего сигнала сброса RST# = 0;

3) поступление сигнала сброса от сторожевого таймера COP08;

4) выборка неправильного кода команды;

5) обращение к ячейке памяти, которая отсутствует в адресном пространстве данной модели микроконтроллера;

6) поступление от модуля LVI08 сигнала о недопустимом снижении напряжения $V_п$.

В процессе запуска микроконтроллера выполняются следующие действия:

- запускается генератор тактовых импульсов (ГТИ), который формирует тактовые импульсы с частотой $F_t = F_q$, задаваемой кварцевым резонатором;
- в программный счетчик PC поступают два байта (старший байт PCh, младший байт PCl) из ячеек памяти с адресами $FFFE…FF, 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.

Режимы функционирования

Микроконтроллеры MC68HC908GP32 могут функционировать в одном из следующих режимов:

- рабочий;
- ожидания;
- останова;
- отладки.

Режим ожидания (Wait mode) реализуется после поступления команды WAIT. При этом прекращается работа процессора, но все остальные модули микропроцессора продолжают функционировать. Потребление мощности в этом режиме снижается в несколько раз. Возврат из режима ожидания в рабочий происходит при поступлении сигнала сброса или запроса прерывания (внешнего IRQ# или от периферийных модулей). При поступлении сигнала сброса производится процедура запуска микроконтроллера, описанная ранее. Если возврат выполняется по сигналу сброса, то в программный счетчик PC загружается вектор начального запуска, если по запросу прерывания, то в PC загружается соответствующий вектор прерывания.

Режим останова (Stop mode) реализуется после поступления команды STOP. В этом случае прекращается работа процессора и большинства других модулей микроконтроллера. Продолжают работать модули BREAK08, IRQ08, KBI08. Функционирование модуля контроля напряжения питания LVI08 и базового таймера TBM08 в режиме останова можно разрешить при записи соответствующего содержимого в регистры конфигурации микроконтроллера. Потребляемый ток в режиме останова снижается до единиц микроампер, если запрещена работа всех модулей микроконтроллера.

Выход из режима останова происходит при поступлении внешних сигналов сброса RST# и прерывания IRQ#, а также по запросам внутренних модулей LVI08, BREAK08, KBI08, TBM08, если разрешена работа соответствующего модуля в режиме останова. При этом в PC загружается вектор запуска или соответствующего прерывания. При поступлении команд WAIT и STOP в регистре признаков CCR устанавливается значение маски прерывания I = 0, чтобы разрешить последующий выход из режима ожидания или останова по запросам прерывания.

Режим отладки. Реализация этого режима является специфической особенностью микроконтроллеров семейства 68HC08/908. Микроконтроллер подключается к персональному компьютеру и работает под управлением внешних команд. Эти команды позволяют контролировать и модифицировать текущее состояние микроконтроллера, выполнять чтение и запись содержимого любой ячейки адресуемой памяти, производить стирание и запись содержимого внутренней Flash-памяти. Данный режим обеспечивает выполнение основных операций, реализуемых в процессе отладки программ.

Функционирование микроконтроллера в этом режиме обеспечивается с помощью специальной программы-монитора, которая записывается в его масочно-программируемое ПЗУ в процессе изготовления. Ввод микроконтроллера в режим отладки осуществляется при подаче на его выводы определенной комбинации потенциалов. На вход IRQ# подается повышенный потенциал

$V_h = V_p$ 2,5 В. Вывод PTA0 порта A служит для последовательного обмена данными между микроконтроллером и персональным компьютером.

В режиме отладки обеспечивается возможность записи содержимого Flash-памяти. Таким образом, можно выполнить программирование микроконтроллера, работающего непосредственно в составе системы управления (ISP – In-System Programming). При этом запись или модификация рабочей программы производится с помощью персонального компьютера без использования специальных программаторов и не требует удаления микроконтроллера из реализованной системы управления.

Программно-логическая модель центрального процессора

Центральный процессор CPU08 выполняет действия над 8-разрядными операндами. Программно-логическая модель CPU8 содержит шесть регистров (рис. 1.2):

- 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.

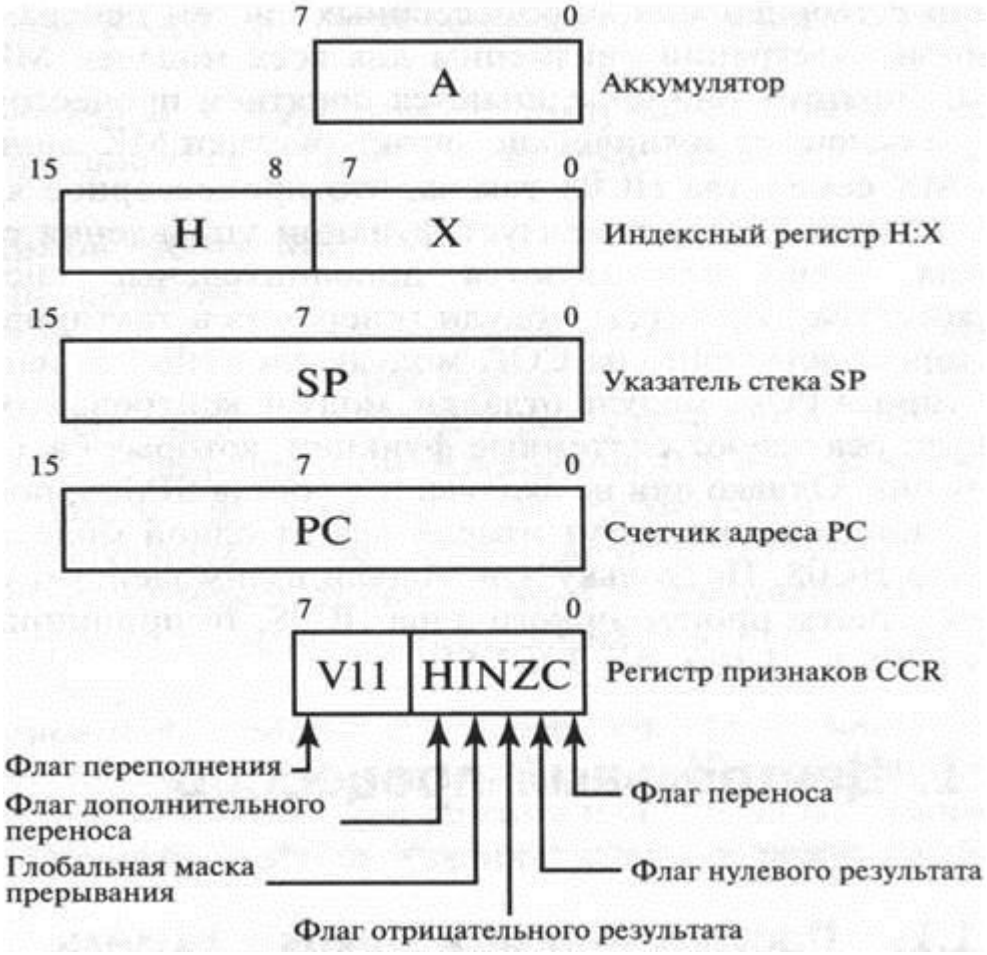

Рис. 1.2

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 Кбайта. 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 2-байтовом, так и в 1-байтовом 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 – это 16-разрядный регистр, который хранит адрес вершины области стека. 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- или 16-разрядным смещением. Чтобы обеспечить совместимость МК семейства HC08 с МК семейства HC05, 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). Следует отметить, что резервирование нулевой страницы памяти для области стека во многих приложениях на МК семейства HC08 нецелесообразно. Нулевую страницу памяти ($0000…$00FF) в этих МК следует использовать для хранения промежуточных данных. Фрагмент кода прикладной программы и время обращения к данным в этом случае будут минимальными. Указатель стека SP можно установить в любое значение из диапазона от $0000 до $FFFF.

Счетчик адреса, или программный счетчик, PC – это 16-разрядный регистр, который 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 (см. рис. 1.2).

Регистр признаков CCR – это 8-разрядный регистр, который содержит информацию о состоянии центрального процессора (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, допол-

нительного переноса H, глобальной маски прерывания I. Особенностью флагов нулевого результата Z и отрицательного результата N является установка их после операций пересылки.

Способы адресации в микроконтроллерах HC08

Рассмотрев программно-логическую модель центрального процессора HC08, отметим следующие его особенности:

- регистры центрального процессора не имеют собственных двоичных адресов в адресном пространстве МК. Следовательно, команды обращения к регистрам должны быть отличны по мнемонике от команд обращения к ячейкам памяти. Например, команда CLRA устанавливает в 0 все биты регистра аккумулятора, команда CLRX – регистра X, команда CLR mm обнуляет ячейку памяти с адресом mm;
- ПЗУ и ОЗУ данных в МК семейства HC08 имеют объединенное адресное пространство. Следовательно, одни и те же команды могут быть использованы для выборки данных из ячеек, принадлежащих к разным типам памяти, и последующего выполнения операций над ними. Например, команда LDA $0023 загружает в аккумулятор центрального процессора данные из ячейки ОЗУ с адресом $23, а команда LDA $81FF – код, хранящийся в ячейке ПЗУ с адресом $81FF.

Центральный процессор HC908 является типичным представителем микропроцессоров с CISC-архитектурой (CISC – Complicated Instruction Set Computer). Одна из отличительных особенностей таких процессоров состоит в том, что длина кода команды в байтах неодинакова для разных команд. Команды процессора HC08 могут иметь 1-, 2- и 3-байтовый (редко 4-байтовый) формат. В первом байте команды помещается код операции, определяющий действие, которое будет совершено над одним или двумя операндами. Во втором и третьем байтах команды помещаются сами операнды или их адреса или задаются данные для вычисления этих адресов. Правило, по которому центральный процессор определяет адрес одного или двух операндов, принято называть способом адресации. Чем больше способов адресации в системе команд центрального процессора, тем более гибко он может обращаться к данным. Последнее увеличивает производительность центрального процессора при исполнении конкретной задачи управления. Кроме того, разнообразие способов адресации позволяет создавать эффективные компиляторы с языков высокого уровня.

Для выборки операндов из памяти МК семейства HC08 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).

Неявная адресация (Inherent). Команды с неявной адресацией содержат адрес операнда в коде команды. Например, INCA – команда увеличения на 1 содержимого аккумулятора A, ASLX – команда сдвига влево регистра X, SEC – команда установки бита переноса C. Все команды с неявной адресацией имеют длину 1 байт и состоят только из кода операции. Эти команды обращаются обычно к регистрам центрального процессора (A, X, H:X) или к отдельным битам регистра признаков (бит переноса C, бит маски прерывания I) и регистров управления режимами работы центрального процессора (команды STOP, WAIT).

Непосредственная адресация (Immediate). Операнд в командах с непосредственной адресацией следует сразу после кода операции и выражается 8-разрядным или 16-разрядным числом, которое не является адресом. Этот режим адресации используется, чтобы, например, загрузить некоторое значение в 8-разрядный или 16-разрядный регистр центрального процессора. Указанные в коде команды данные не могут быть изменены в ходе выполнения программы, так как они расположены непосредственно в памяти программ, т. е. в ПЗУ. Условное обозначение операнда в мнемонике команды – #opr.

Прямая адресация (Direct). Команды с прямой адресацией имеют длину 2 байта. После кода операции команды следует 8-разрядный адрес операнда и, следовательно, могут быть адресованы 256 ячеек памяти. Поскольку старший байт адреса всегда интерпретируется как $00, этот способ адресации обеспечивает возможность обращаться к адресам, расположенным в интервале от $0000 до $00FF. Этот диапазон называют нулевой страницей памяти. В МК семейства HC08 нулевая страница памяти отведена под регистры управления периферийными модулями, однако часть ячеек нулевой страницы в МК любой модели обязательно будет принадлежать ОЗУ данных. Именно в эти ячейки ОЗУ выгодно помещать часто используемые переменные и программные счетчики. Команды с прямой адресацией типа Direct всегда на 1 байт короче команд с прямой расширенной адресацией типа Extended, поэтому объем программы будет меньше. Кроме того, команды с адресацией типа Direct выполняются на один цикл быстрее. Эти два преимущества могут дать большой выигрыш как по быстродействию программы управления, так и по объему ее кодового представления. Условное обозначение адреса в мнемонике команды – opr.

Прямая расширенная адресация (Extended). При прямой расширенной адресации открывается доступ к полному интервалу адресного пространства МК HC08 объемом 64 Кбайта, так как за кодом операции следует уже не 8-разрядный, а 16-разрядный адрес. Поэтому все команды с прямой расширенной адресацией содержат 3 байта: первый байт задает код операции, второй – является старшим байтом (MSB) и третий – младшим байтом (LSB) адреса. Этот способ адресации должен быть выбран, когда необходимо использовать адресное пространство, выходящее за рамки нулевой страницы (от $0000 до $00FF). При этом выбор типа адресации (прямая или прямая расширенная) не является задачей программиста, так как программа Ассемблер выбирает его автоматически в зависимости от численного значения адреса.

Индексная адресация (Indexed) особенно подходит для доступа к блокам памяти или таблицам, в которых адреса расположены один за другим и извлечение или загрузка происходят последовательно шаг за шагом. Принцип индексной адресации заключается в том, что команда воспринимает содержимое индексного регистра H:X как базовый адрес операнда. Если команда не содержит беззнаковой константы, то этот адрес и служит адресом ячейки памяти, в которой хранится искомый операнд. Если же в формате команды указана константа размером в 1 или 2 байта, которая называется смещением, то адрес операнда вычисляется как сумма смещения и базового адреса. Таким образом, смещение, добавленное к содержимому индексного регистра H:X перед исполнением действия над операндом, является текущим адресом операнда. При переходе от одного цикла исполнения программного фрагмента к другому индексный регистр может увеличиваться или уменьшаться программистом. Тогда одинаковые смысловые действия будут производиться над последовательно расположенными в памяти операндами.

Команды с индексной адресацией без смещения – это 1-байтовые команды, адрес операнда которых должен быть предварительно занесен в индексный регистр H:X. Старший байт (MSB) находится в регистре H, младший байт (LSB) – в регистре X. Содержимое индексного регистра представляет собой конечный, или исполнительный, адрес. Этот способ адресации может использоваться для передачи указателя на таблицу или для хранения адреса, который указывает на часто используемую ячейку памяти или порт ввода и вывода.

Команды с индексной адресацией и 8-разрядным смещением (Indexed, 8 bit offset) – это 2-байтовые команды, в которых за кодом операции располагается 8-разрядный адрес смещения. Центральный процессор вычисляет конечный адрес, добавляя 8-разрядное смещение к содержимому индексного регистра H:X. Оба адреса интерпретируются как положительные числа. При этом способе адресации легко адресовать i-й элемент в таблице из n элементов. Таблица может начинаться в любой точке внутри адресного интервала размером 64 Кбайта, потому что в качестве указателя начала таблицы используется 16-разрядный регистр H:X. Чтобы определить i-й элемент, используется смещение, которое не превышает 256 байт.

Если таблица содержит большее число элементов, можно использовать индексную адресацию с 16-разрядным смещением.

Команды индексной адресации с 16-разрядным смещением (Indexed, 16 bit offset) – это 3-байтовые команды, в которых за кодом операции следуют 2 байта 16-разрядного адреса. Смещение также прибавляется к содержимому индексного регистра H:X, чтобы определять исполнительный адрес. Первым байтом после кода операции является старший байт 16-разрядного кода смещения (MSB), а вторым байтом – младший байт (LSB). Этот способ адресации может использоваться для доступа к таблицам, в которых имеется более 256 элементов.

Индексная адресация с последующим инкрементированием (Indexed with post incrementer) используется только в командах mov и cbeq. Адрес одного из операндов этих команд находится в 2-байтовом индексном регистре H:X. После выполнения команды содержимое индексного регистра H:X увеличивается на 1.

Индексная адресация со смещением 1 байт и последующим инкрементированием (Indexed, 8 bit offset with post incrementer) используется только в команде CBEQ. В определении адреса одного из операндов команды участвует индексный регистр H:X, который содержит код базового адреса. Центральный процессор вычисляет адрес операнда путем сложения содержимого индексного регистра H:X с байтом кода смещения, который указан во втором байте команды. После сложения адрес операнда представляется в 2-байтовом формате. После выполнения команды содержимое индексного регистра H:X увеличивается на 1.

Индексная адресация по указателю стека SP. Простой доступ к данным, расположенным в области памяти стека, осуществляется при индексной адресации по указателю стека. При этом способе адресации для вычисления исполнительного адреса операнда центральный процессор добавляет положительное 16-разрядное значение указателя стека SP к положительному смещению, которое указано в коде команды. Смещение может быть 8- или 16-разрядным. Если прерывания блокированы, указатель стека SP свободен от выполнения основных функций и может использоваться в качестве второго индексного регистра в дополнение к регистру H:X.

Все команды с адресацией по указателю имеют 2-байтовый код операции, в качестве первого байта кода операции используется число $9E. Следовательно, эти команды исполняются на один цикл дольше, чем команды с индексной адресацией по регистру H:X.

При использовании индексной адресации по указателю стека с 8-разрядным смещением (Stack pointer, 8 bit offset) исполнительный адрес операнда вычисляется как сумма содержимого указателя стека SP и 8-разрядного смещения. Все команды с такой адресацией имеют 3-байтовый формат.

Относительная адресация (Relative) используется только в командах условных переходов, которые применяются для организации ветвления программ. Команды условных переходов имеют 2-байтовый формат. Первый байт содержит код операции, второй – смещение адреса следующей команды отно-

сительно адреса текущей команды в целочисленном формате со знаком. Коды смещения могут лежать в диапазоне от –128 до +127. Если условие, заданное типом используемой команды условного перехода, выполняется, то адрес следующей команды центральный процессор вычисляет путем сложения текущего адреса с кодом смещения. Если условие не выполняется, то МК переходит к выполнению следующей команды.

Выполнение и защита лабораторных работ

Каждый студент получает индивидуальное задание по его номеру, указанному преподавателем в журнале проведения лабораторных работ.

Защита лабораторной работы осуществляется студентом лично в присутствии преподавателя и состоит из двух частей:

1. Демонстрация хода выполнения программного кода, написанного студентом.

2. Защита студентом заранее подготовленного отчета с дополнительными вопросами как по теоретической, так и по практической части изученного студентом материала.

Отчет по лабораторной работе должен содержать:

1. Цель работы.
2. Задание.
3. Основные теоретические сведения.
4. Полученный результат.
5. Выводы по работе.

2. ЗАПУСК И НАЧАЛЬНАЯ НАСТРОЙКА СРЕДЫ

Загрузка среды осуществляется с запуска файла ide.exe. Для создания нового проекта необходимо выбрать команду «New project» в меню «Project» (рис. 2.1), затем заполнить следующие поля:

- Project name – имя проекта (введите имя проекта);
- Location – расположение (введите название папки, где проект будет расположен с таким же именем, как и имя проекта);
- Toolkit – комплект инструментов (оставьте по умолчанию SM11 Assembler for 68HC11);
- Debugger – отладчик (выберите 6811 Simulator).

Остальные поля заполняются автоматически (рис. 2.2).

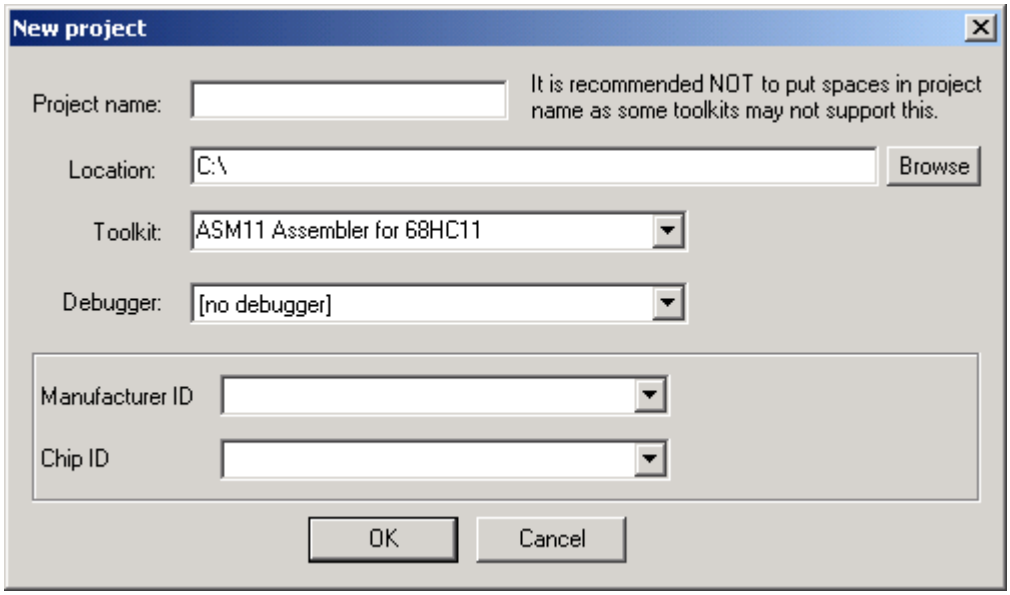

Рис. 2.1

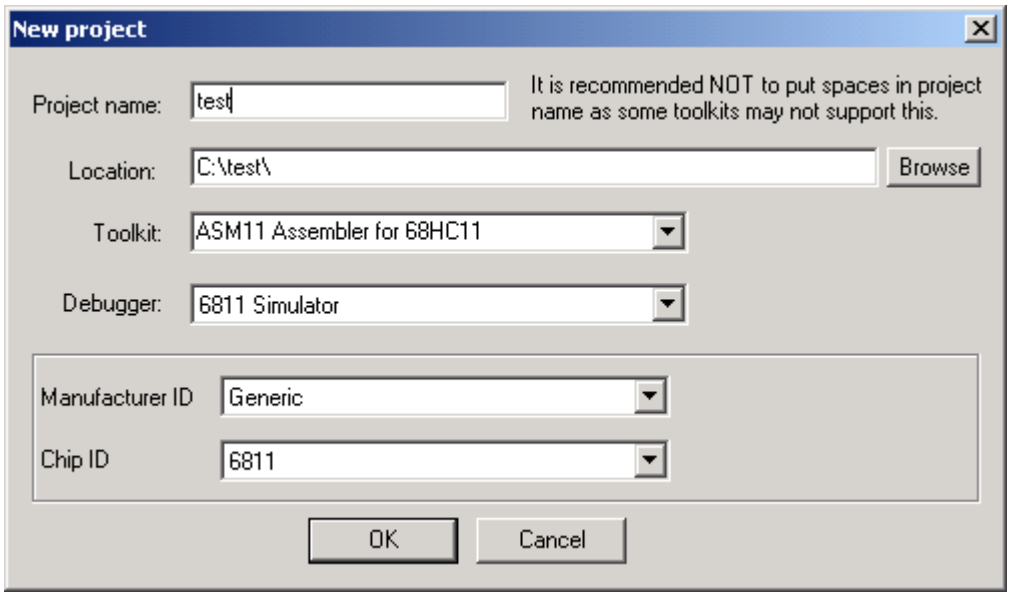

Рис. 2.2

Затем задайте параметры настройки проекта и нажмите OK. После этого будет создан новый пустой проект (рис. 2.3).

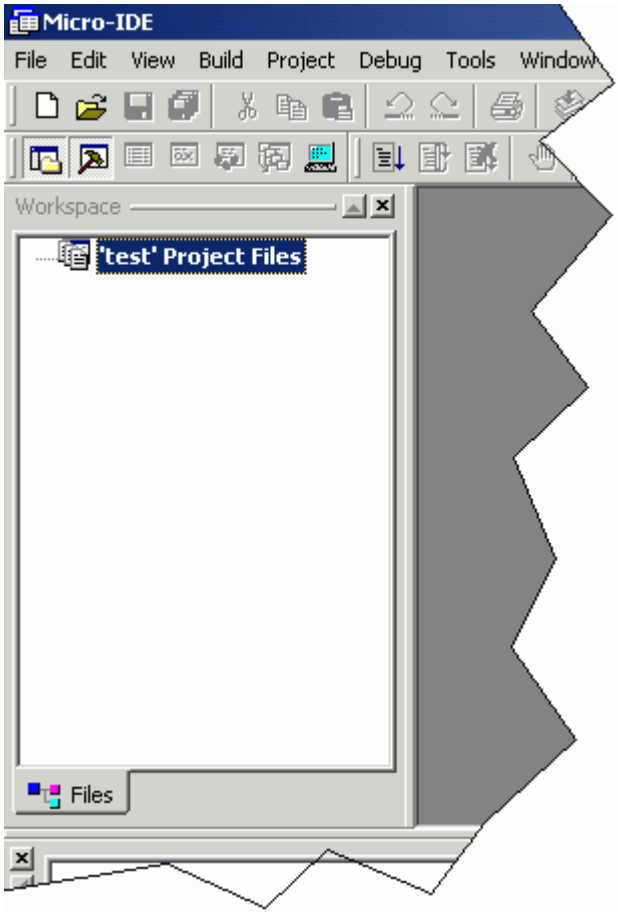

Рис. 2.3

Добавьте файлы в проект. Для этого необходимо выделить проект в диалоговом окне «Workspace» и выбрать команду «New» в меню «File». После чего появится диалоговое окно (рис. 2.4).

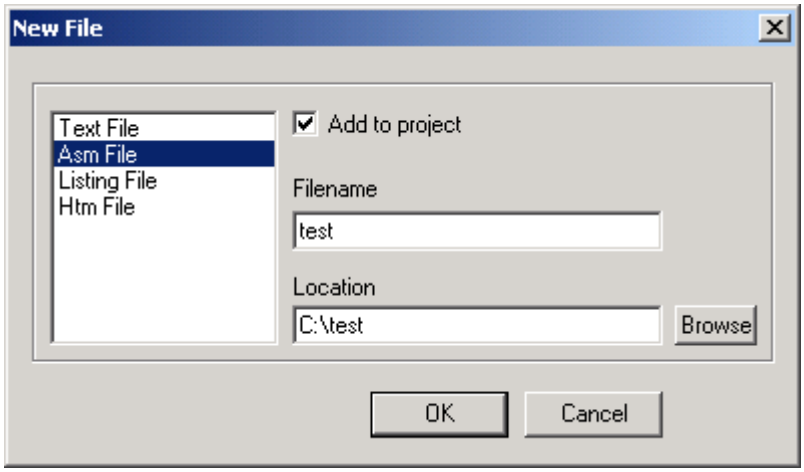

Рис. 2.4

В данном окне выберите тип файла «Asm File» и задайте имя файла (должно совпадать с именем вашего проекта). Далее нажмите OK.

Появится рабочая область созданного файла, в которой необходимо написать код программы (рис. 2.5).

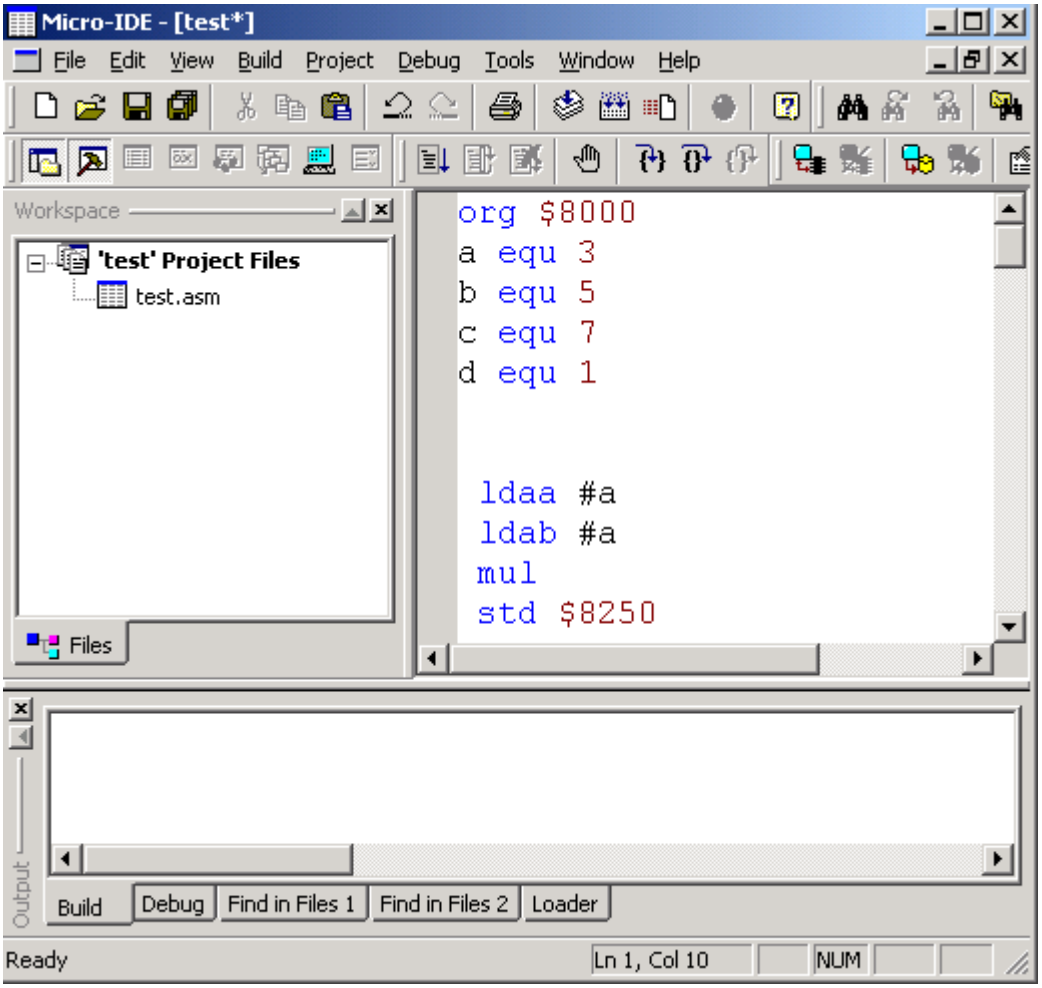

Рис. 2.5

После написания кода необходимо откомпилировать проект. Для этого выберите команду «Assemble» в меню «Build». Если в проекте имеются ошибки, то в диалоговом окне «Output» будет выведена соответствующая информативная строка (рис. 2.6).

Для того чтобы просмотреть, какие именно ошибки были допущены, необходимо выбрать команду «Open» в меню «File» и файл с именем проекта и расширением *.lst (рис. 2.7). Откроется данный файл. В нем будут описаны все ошибки, допущенные в коде (рис. 2.8).

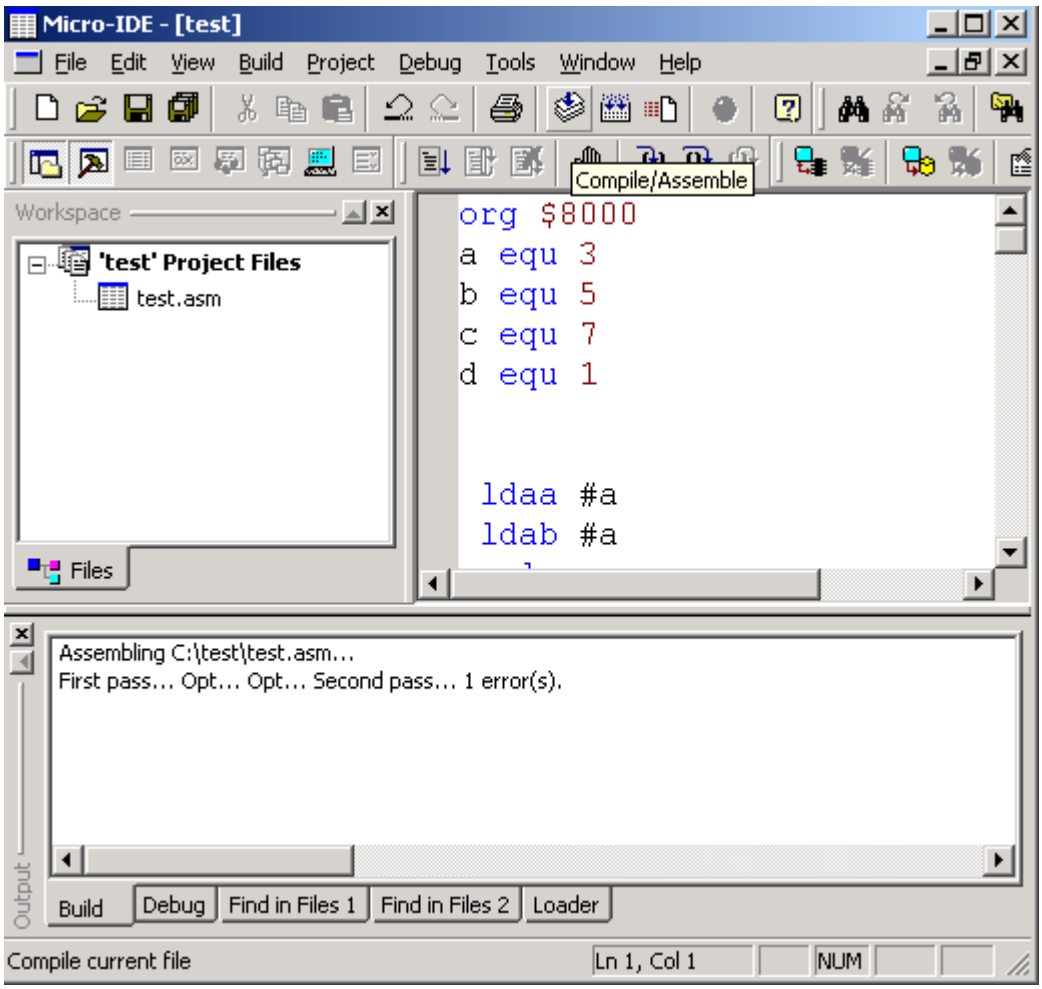
Micro-IDE - [test]
File Edit View Build Project Debug Tools Window Help
Compile/Assemble
Workspace
'test' Project Files
test.asm
Files
org $8000
a equ 3
b equ 5
c equ 7
d equ 1
ldaa #a
ldab #a
Assembling C:\test\test.asm...
First pass... Opt... Opt... Second pass... 1 error(s).
Output
Build
Debug
Find in Files 1
Find in Files 2
Loader
Compile current file
Ln 1, Col 1
NUM

Рис. 2.6

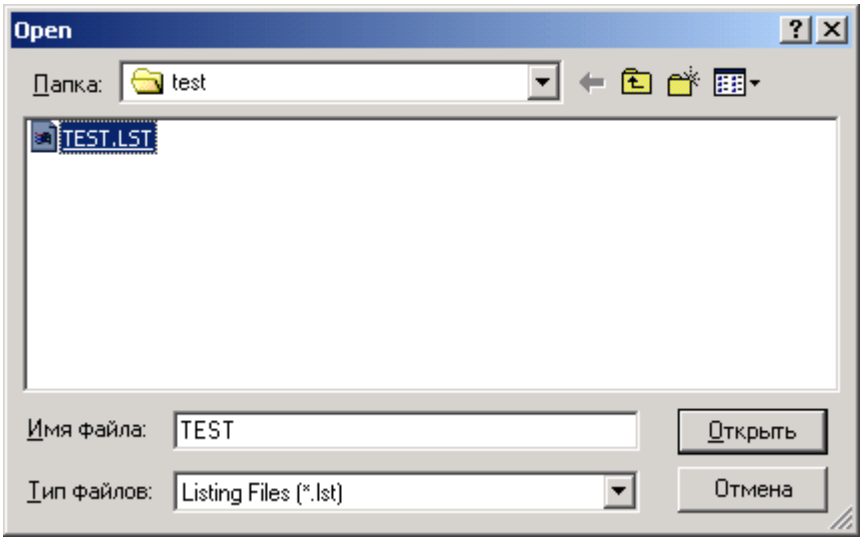
Open
Папка:
test
TEST.LST
Имя файла:
TEST
Открыть
Тип файлов:
Listing Files (*.lst)
Отмена

Рис. 2.7

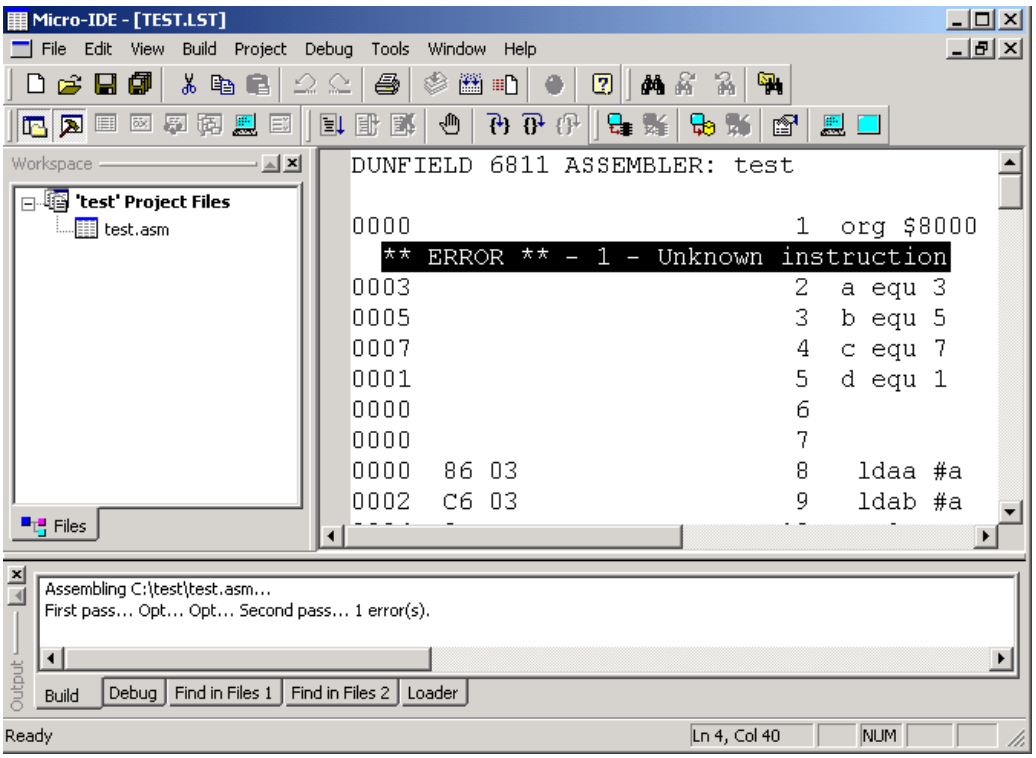

Рис. 2.8

Далее необходимо закрыть файл с расширением *.lst, перейти к исходному коду, исправить все ошибки и снова откомпилировать проект.

После того как все ошибки будут исправлены, а проект будет сохранен, запустите его на исполнение. Для этого выберите команду «Build» в меню «Build».

Для отладки программы и просмотра состояния регистров необходимо выполнить следующие настройки:

1. Выбрать команду «Settings» в меню «Project» и в закладке «General» выставить флаг «Automatically start with assembly listing» (рис. 2.9).

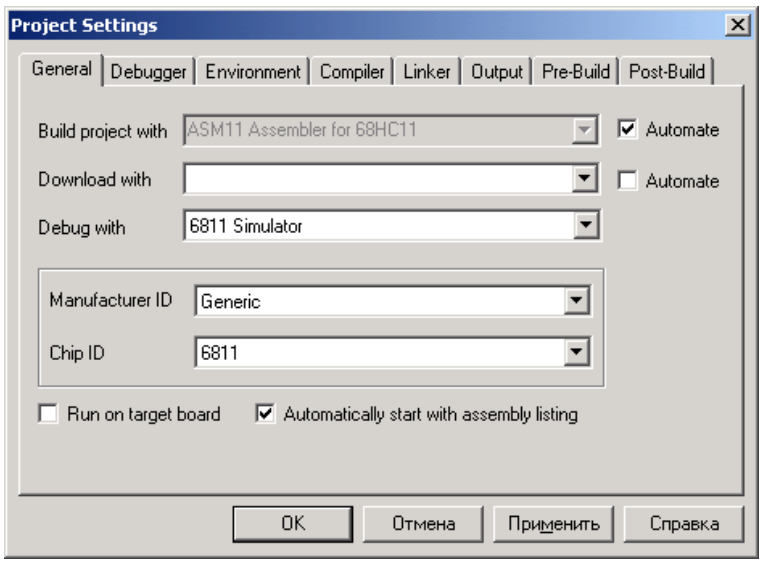

Рис. 2.9

2. В закладке «Output» выбрать файл с именем проекта и расширением *.hex (рис. 2.10).

3. Нажать OK.

После этого приступайте к отладке программы. Для этого нажмите F10 для появления окна отладчика (рис. 2.11).

Нажмите F10 несколько раз подряд до появления зеленой полосы (рис. 2.12).

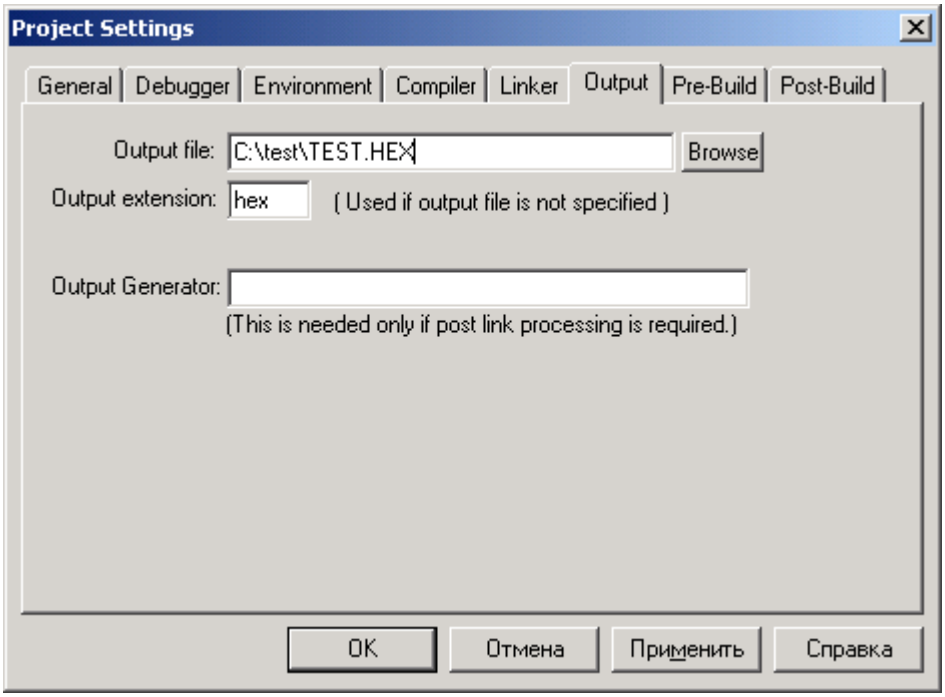

Рис. 2.10

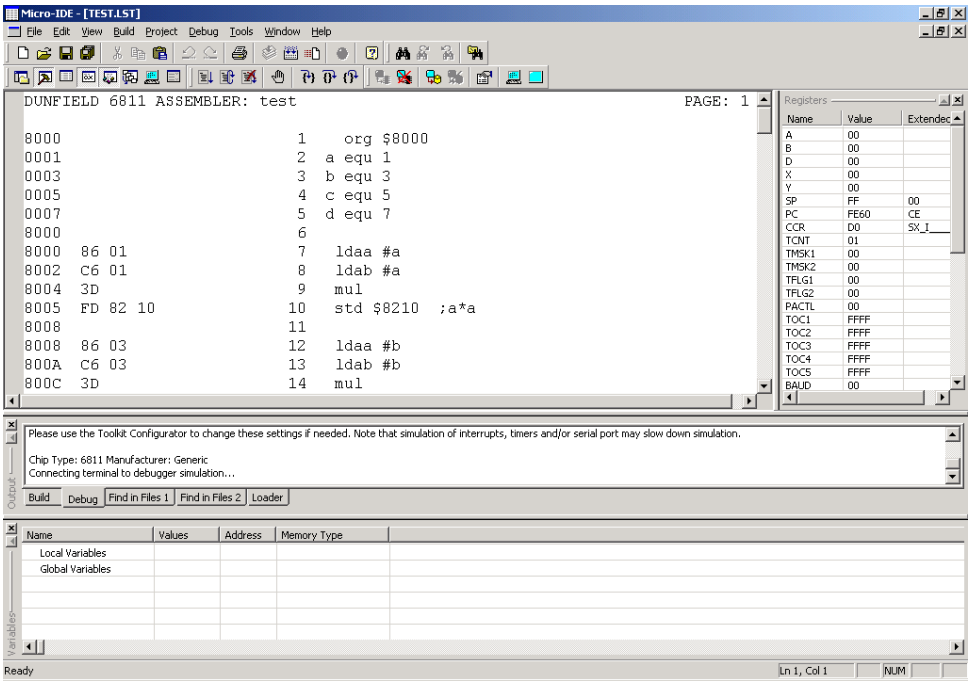

Рис. 2.11

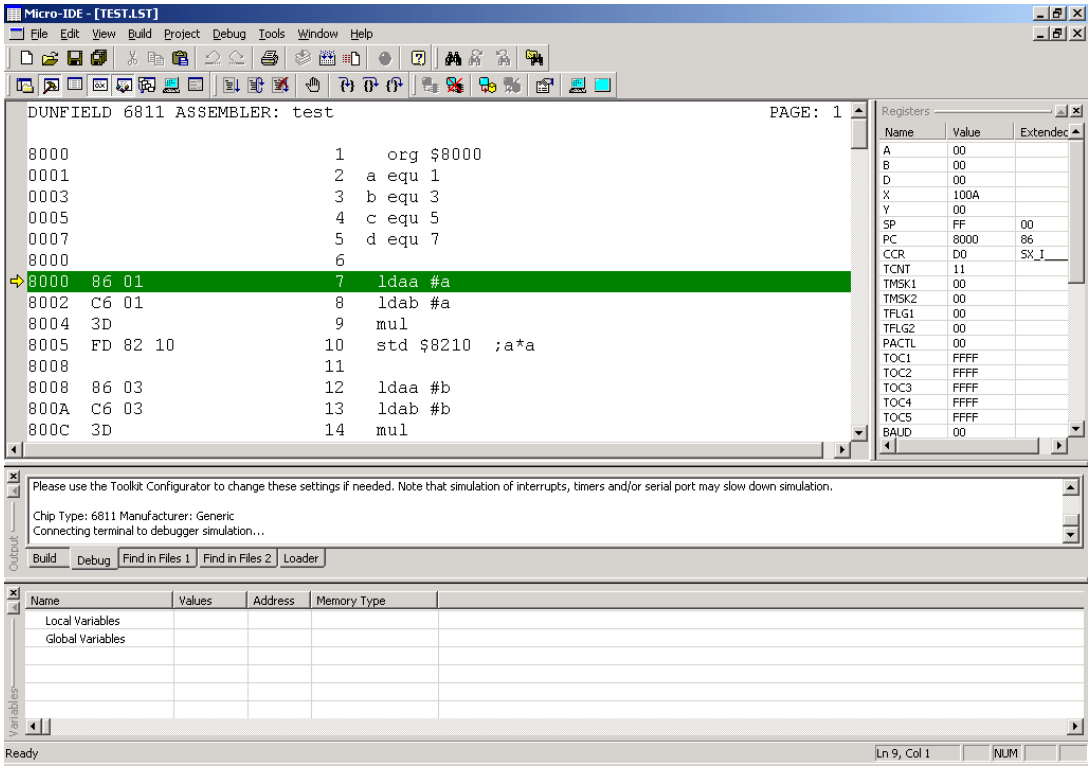

Рис. 2.12

Теперь можно вручную отладить код, нажимая F10, при этом отладчик будет переходить от строки к строке по порядку, причем в окне «Registers» будет отражено состояние регистров в текущий момент. Можно установить контрольную точку на любой из строк кода, нажав правую кнопку мыши и выбрав команду «Toggle Breakpoint» (рис. 2.13).

При нажатии F5 отладчик «прыгнет» на данную контрольную точку, причем состояние регистров изменится в соответствии с командой, находящейся в строке, которая является предыдущей к строке с контрольной точкой (рис. 2.14).

Для того чтобы просмотреть состояние памяти, необходимо выбрать команду «Memory» в меню «View». При этом появится окно «Memory» (рис. 2.15).

Значения регистров отображены в шестнадцатеричной системе счисления. Для просмотра значения в десятеричной системе счисления наведите курсор на значение необходимого регистра и щелкните правой кнопкой мыши (рис. 2.16).

Для того чтобы вернуться к коду, необходимо нажать Shift+F5, при этом будут закрыты все окна отладчика, а затем закрыть файл с расширением *.lst.

Остальные возможности работы с программой можно прочесть в «Help for Micro-IDE (6811 Dev System)».

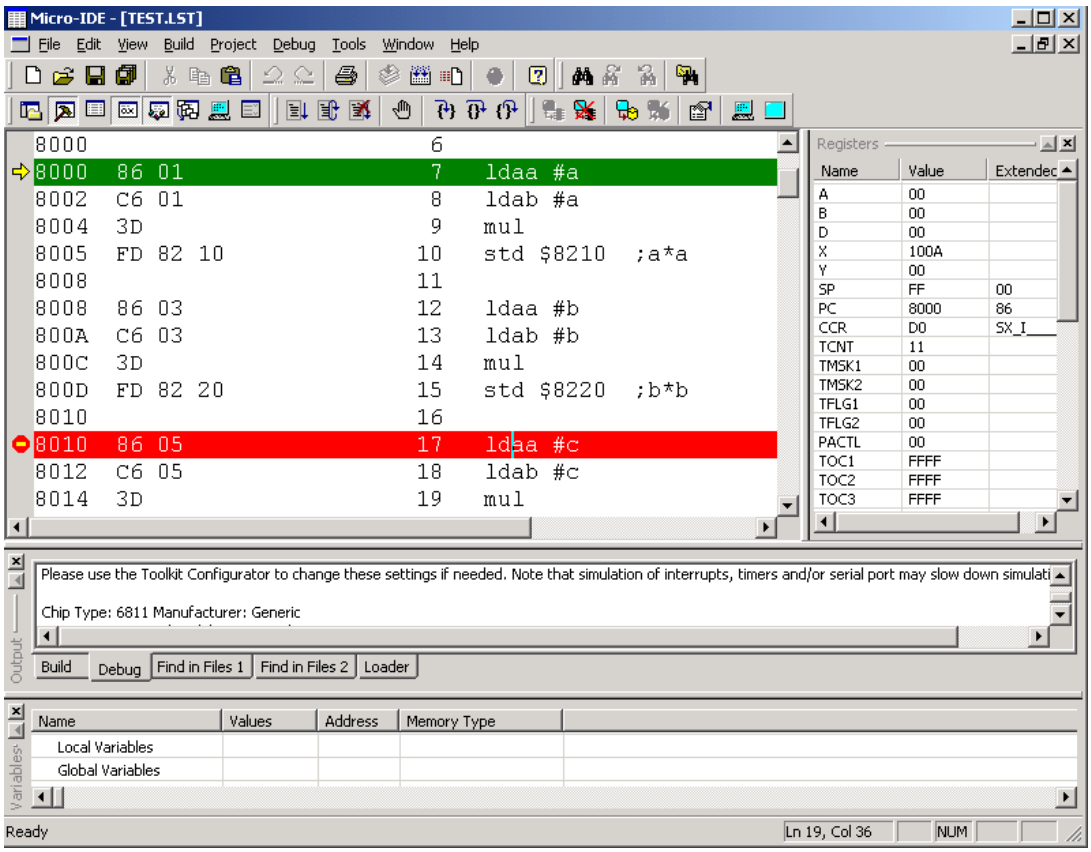

Рис. 2.13

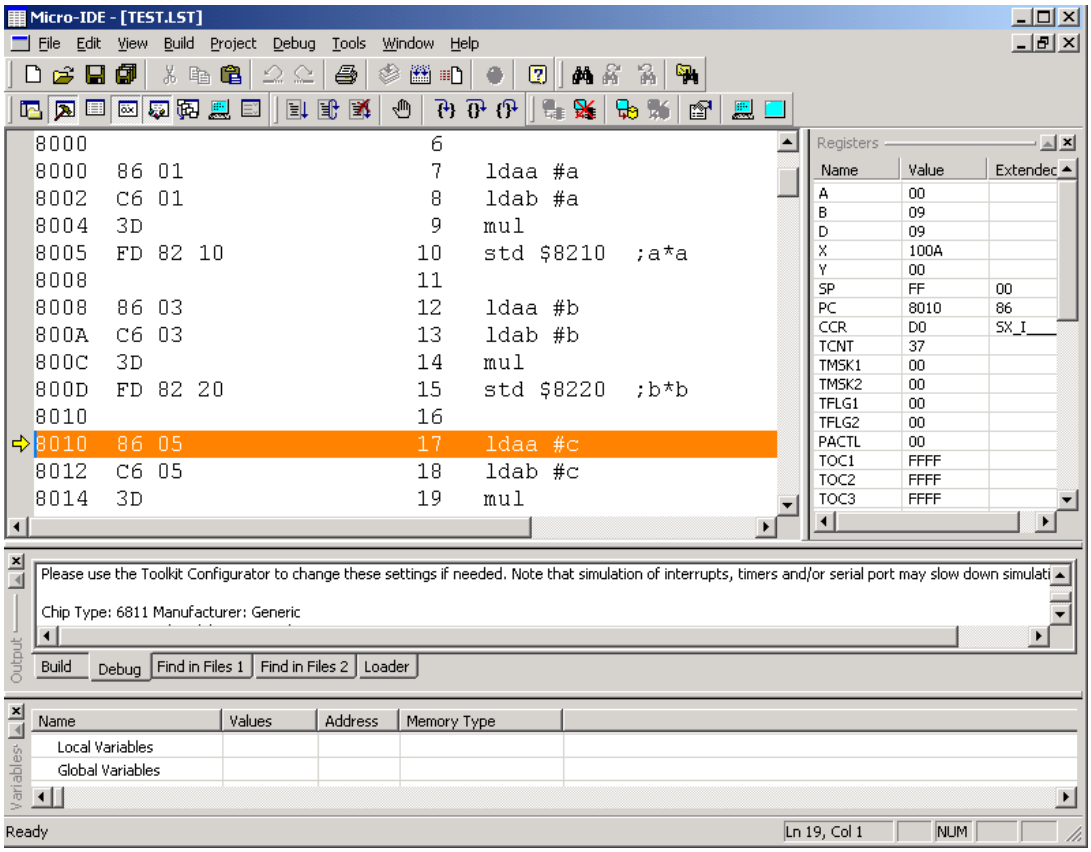

Рис. 2.14

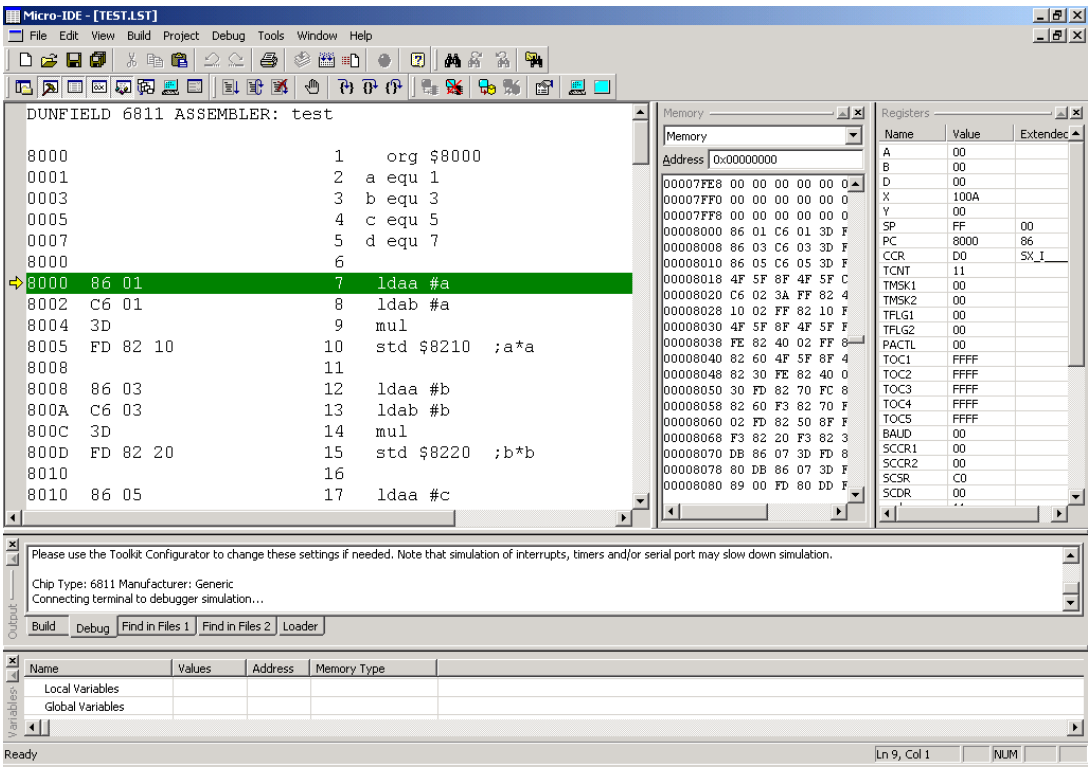

Рис. 2.15

Рис. 2.16

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

ЛАБОРАТОРНАЯ РАБОТА №1
МЕТОДЫ АДРЕСАЦИИ. КОМАНДЫ ПЕРЕСЫЛКИ ДАННЫХ

Цель работы. Разработать программу с использованием следующих групп команд:

- методы адресации;
- команд пересылки данных.

Теоретическая часть

1.1. Методы адресации

Микроконтроллеры семейства MC68HC11 выполняют обработку 8-, 16-разрядных операндов и реализуют набор из 108 команд. Они содержат два 8-разрядных аккумулятора A и B, которые при выполнении ряда команд используются как 16-разрядный регистр D, два 16-разрядных индексных регистра X и Y, регистр условий CCR и 16-разрядные регистр-указатель стека SP и программный счетчик PC.

Регистр CCR (табл. 1.1) содержит значения признаков переноса C, переполнения V, нулевого результата Z, знака N, запрещения прерывания I, переноса между тетрадами H.

Таблица 1.1

Формат содержимого регистра условий CCR

Бит	7	6	5	4	3	2	1	0
Признак	S	X	H	I	N	Z	V	C

Микроконтроллеры семейства MC68HC11 имеют следующие типы адресации: неявная, непосредственная, прямая, расширенная, индексная и относительная.

Неявная адресация применяется в том случае, когда в качестве операндов используются либо регистры (например COMA, CLI), либо фиксированная ячейка памяти (SWI). Другими словами можно сказать, что неявная адресация не требует отдельного битового поля для указания операнда. В большинстве случаев такие команды 1-байтовые.

```
43          COMA
53          COMB
```

Исключение составляют команды, взаимодействующие с регистром Y:

```
18  35      TYS
18  3A      ABY
```

В случае использования ***непосредственной адресации*** операнд (или один из операндов) включен непосредственно в код команды. Длина таких команд может составлять от двух до четырех байтов. При записи команд, использующих данную адресацию, операнд предваряется символом «решетка» (#).

```
86    03          LDAA    #3
CE    80    00    LDX     #32768
18    8C    56 78 CPY     #$5678
```

Прямая адресация используется для доступа к данным, расположенным в первых 256 байтах памяти. При этом младший байт адреса операнда расположен непосредственно за кодом команды. Применение этой группы команд позволяет сократить объем программы, а также время выполнения на выборке операнда из памяти.

```
96    3F          LDAA
63    DA    FF    GRAB    $FF
```

Использование ***расширенной адресации*** позволяет осуществить доступ к любой ячейке памяти в пределах адресного пространства контроллера. При этом два байта, следующие непосредственно за кодом команды, представляют собой абсолютный адрес операнда.

```
B6    40    00    LDAA    $4000
7E    78    12    JMP     $7812
```

Как правило, программа Ассемблер автоматически выбирает наиболее оптимальный из двух вышеописанных методов адресации.

Для доступа к массивам данных удобно использовать ***индексную адресацию***. В микроконтроллерах семейства MC68HC11 используется так называемая *индексная адресация с 8-разрядным смещением*. При этом в индексный регистр X или Y заносится 16-разрядный адрес, а следующий за кодом команды байт содержит 8-разрядное смещение. Абсолютный адрес при этом вычисляется простым суммированием содержимого индексного регистра с байтом смещения.

```
A6    07          LDAA    $07,X
18    AD    00    JSR     0,Y
```

Команды работы со стеком также принято относить к командам с индексной адресацией.

```
32                PULA
37                PSHB
```

Эти команды используют *индексную адресацию без смещения*.

Относительная адресация используется в командах передачи управления. При этом абсолютный адрес перехода вычисляется путем сложения содержимого программного счетчика со смещением, представляющим собой 8-разрядное знаковое число. Таким образом, используя относительную адресацию, можно осуществить переход на адрес, лежащий в пределах от –128 до +127 относительно адреса, следующего за командой перехода.

```
8D    00          BSR     *+$2
24    FF          BCC     *–125
```

Заметим, что для наглядности здесь использован символ «звездочка» (*), который заменяется программой Ассемблер на адрес текущей команды. Программы, использующие только относительную и неявную адресацию, принято называть *позиционно-независимыми программами*. Это объясняется тем, что при перемещении кода из одной области памяти в другую работоспособность программы сохраняется.

1.2. Команды пересылки данных

Простейшими командами являются команды пересылки данных. Список этих команд приведен в табл. 1.2.

Таблица 1.2

Команды пересылки данных

TSTA TSTB TST*	CLRA CLRB CLR*	TAB TBA TAP	PSHA PULA PSHB
LDAA** LDAB** LDD** LDX** LDY** LDS**	STAA*** STAB*** STD*** STX*** STY*** STS***	TPA TSX TXS TSY TYS XGDX XGDY	PULB PSHX PULX PSHY PULY

Примечания:

* – команды, использующие расширенную и индексную адресацию;

** – команды, использующие непосредственную, прямую, расширенную и индексную адресацию;

*** – команды, использующие прямую, расширенную и индексную адресацию.

Команды TSTA, TSTB и TST служат для установки регистра статуса согласно содержимому регистров A, B или ячейки памяти соответственно. Далее результат может быть использован в командах условного перехода. Занесите в регистр A значение $00 и выполните в пошаговом режиме команду TSTA.

TSTA, TSTB, TST (opr)

S	X	H	I	N	Z	V	C
-	-	-	-	?	?	0	0

LDAA (opr), LDAB (opr),
LDD (opr), LDS (opr),
LDX (opr), LDY (opr)

S	X	H	I	N	Z	V	C
-	-	-	-	?	?	0	-

Теперь посмотрите на содержимое регистра статуса: должен быть установлен флаг нуля Z и сброшены флаги отрицательного результата M, переноса C и переполнения V. Проведите подобный опыт при других значениях регистра A, обращая внимание на различное состояние регистра статуса.

Рассмотрим команды загрузки в регистр содержимого ячейки памяти:

```
org   $8000
ldab  $56        ;   загрузить в регистр B содержимое ячейки $56,
                 ;   используя прямую  адресацию
ldy   $c800      ;   загрузить в регистр Y данные, расположенные по адресу
                 ;   $c800 (предыдущую команду)
ldx   #$1f00     ;   установить регистр X
ldaa  $03,x      ;   считать информацию
```

CLRA, CLRB, CLR (opr)

S	X	H	I	N	Z	V	C
-	-	-	-	0	1	0	0

Работа команд очистки регистров A, B и ячейки памяти может быть проиллюстрирована на примере следующей простой программы:

```
org   $8000
clrb             ;   очистить регистр B
ldx   #$1f00     ;   установить регистр X
clr   $04,x      ;   очистить
```

STAA (opr), STAB (opr),
STD (opr), STS (opr),
STX (opr), STY (opr)

S	X	H	I	N	Z	V	C
-	-	-	-	?	?	0	-

Теперь рассмотрим работу команд модификации ячеек памяти. Для этого введем следующую программу:

```
org   $8000
ldd   #$AA55     ;   установить в регистре D значение $AA55
ldx   #$1f00     ;   установить регистр X
clr   $04,x      ;   очистить
staa  $04,x      ;   записать
stab  $04,x      ;   записать
ldaa  $03,x      ;   считать информацию
staa  $04,x      ;   записать
```

В результате выполнения команды TAB значение аккумулятора A будет присвоено аккумулятору B. Команда TBA имеет противоположный эффект. Следует отметить, что регистр статуса принимает состояние, подобное выполнению команд STAA, STAB.

TAB, TBA

S	X	H	I	N	Z	V	C
-	-	-	-	?	?	0	-

Команда TPA осуществляет перенос содержимого регистра CCR в аккумулятор A. Это целесообразно, если после выполнения какой-либо подпрограммы необходимо сохранить состояние регистра статуса (см. также далее описание команды TAP).

TPA, TSX, TSY, TXS, TYS, XGDX, XGDY

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

Группа команд работы с регистром стека имеет одну особенность: регистр стека получает на единицу меньшее значение при переносе числа из индексного регистра, при обратной пересылке происходит увеличение индексного регистра. Рассмотрим эти команды подробнее:

```
org   $8000
ldx   #$220      ; занести в регистр X адрес $220
xgdx             ; обмен содержимого регистров X и D
clrb             ; очистить младший байт регистра D
xgdx             ; X = $200
txs              ; SP = $1ff
tsy              ; Y = $200
```

Обмен содержимого индексного регистра и регистра D, как правило, используется при арифметических операциях (так как арифметические команды работы с регистром D более развиты) или в случае необходимости 8-разрядного доступа к содержимому индексного регистра, что может быть полезно, например, для организации кольцевого буфера.

TAP

S	X	H	I	N	Z	V	C
?	?*	?	?	?	?	?	?

* - значение может быть изменено только из 1 в 0.

Команда TAP осуществляет перенос значения регистра A в соответствующие биты регистра статуса CCR. При этом содержимое регистра A остается неизменным. Флаг X, служащий для маскирования прерывания XIRQ, в результате выполнения этой команды может быть сброшен, но он не может быть установлен, если до выполнения команды флаг был сброшен.

```
org   $8000
ldaa  #$47       ;   занести в регистр A новое содержимое регистра статуса
tap              ;   установить новое значение регистра статуса: заметьте,
                 ;   что флаг X не будет установлен
```

Команды работы со стеком, как правило, используются в подпрограммах для того, чтобы сохранить значение одного или более регистров.

PSHA, PSHB, PSHX, PSHY, PULA, PULB, PULX, PULY

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

Алгоритм работы команд PSH таков:

1) в ячейку памяти, на которую указывает регистр SP, записывается младший байт регистра-операнда;

2) значение регистра SP уменьшается на 1, указывая на следующую свободную ячейку в области стека;

3) в случае 2-байтового операнда последовательность (1–2) повторяется со старшим байтом операнда.

Команды группы PUL выполняют данную последовательность в обратном порядке, увеличивая значение регистра SP.

Следующая программа демонстрирует, каким образом можно сохранить неизменными все внутренние регистры микроконтроллера (рекомендуется также обратить внимание на содержимое стека):

```
org   $8000
psha          ; последовательно сохранить регистры A, B, X, Y, CCR в стеке
pshb
pshx
pshy
tpa
psha
ldaa  #$20    ; выполнить какие-либо действия, в результате которых изменяется
ldx   $12     ; содержимое регистров
ldy   $1f03
clrb
xgdy
pula          ; восстановить регистры CCR, Y, X, B, A
tap
puly
pulx
pulb
pula
```

Следует отметить, что из-за особенностей эмуляции при выполнении программы в пошаговом режиме содержимое ячеек памяти, расположенных ниже указателя, не сохраняется.

Практическая часть

1.3. Задания

1. Напишите программу, заполняющую ячейки $8200…$8205 значением $55, используя индексную адресацию.

2. Перепишите регистр A в регистр B таким образом, чтобы значение регистра флагов осталось неизменным.

3. Занесите $AA и $55 в регистры A и B соответственно. Перенести значение этих регистров в регистр X таким образом, чтобы в регистре X оказалось значение $55AA.

4. Заполните 10 ячеек стека значением ячеек памяти, начиная с $8000.

5. Произведите обмен регистров X и Y тремя различными способами.

6. Занесите в регистр X число $1F0. Используя только рассмотренные в этой лабораторной работе команды, уменьшить это число на 3.

7. Произведите обмен содержимого младшего байта регистра X с регистром A.

8. Измените порядок следования байтов в регистре X, не используя команду XGDX.

9. Занесите значение регистра стека в регистр D.

10. Измените порядок следования байтов в регистре Y, используя только неявную адресацию.

11. Сохраните текущее значение регистра стека в стеке.

12. Установите регистр флагов в соответствии с содержимым младшего байта регистра SP.

13. Перепишите содержимое регистра A в регистры B, X и Y.

14. Сохраните все регистры ОЭВМ в ячейках памяти $8100 ... $8108. При этом содержимое данных ячеек памяти должно соответствовать значению регистров при входе в программу.

В прил. 1 представлена система команд, в прил. 2 – пример программы.

Примечание. При написании программ в случае необходимости следует предварительно записать значения в ячейки памяти в соответствии с заданием.

1.4. Контрольные вопросы

1. Какие методы адресации вам известны? Дайте краткую характеристику каждого из них.

2. Какие методы адресации могут быть использованы в командах LDAA, STAA?

3. На какие флаги влияет выполнение команды TSTA?

4. Как формируется абсолютный адрес перехода в командах, использующих индексную адресацию?

5. Укажите на неточности (если они есть) в написании команд:

```
ldaa   #20
staa   #$50
ldab   #$500
tax
xgdy
```

6. Какие из изученных в данной лабораторной работе команд влияют на содержимое регистра SP?

7. Что такое позиционно-независимая программа?

8. Какие методы адресации используют приведенные ниже команды:

```
ldaa   #20
staa   $20
psha
coma
pulb
```

9. Каковы значения регистров X и D в результате выполнения программы:

```
ldaa   #30
ldx    #$4020
tab
psha
psha
xgdx
pulx
```

10. Какие особенности имеет команда TAP?

11. Какое применение находит команда XGDX?

12. Каково значение регистра SP в результате выполнения программы:

```
ldx   #$200
txs
pshx
pula
```

13. Как формируется абсолютный адрес перехода в командах, использующих относительную адресацию?

14. Какая логическая ошибка допущена в данном фрагменте программы:

```
ldx    #$20
pula
ldaa   0,x
staa   5,x
ldaa   3,x
staa   $22
psha
```

15. Каково значение регистра Y в результате выполнения программы:

```
ldx    #$4644
stx    $20
ldaa   #$20
tab
std    $21
ldy    $20
```

ЛАБОРАТОРНАЯ РАБОТА №2
АРИФМЕТИЧЕСКИЕ КОМАНДЫ

Цель работы. Разработать программу с использованием следующих групп команд:

- сложения;
- вычитания;
- умножения;
- деления;
- десятичной коррекции.

Теоретическая часть

2.1. Арифметические команды

Список арифметических команд приведен в табл. 2.1.

Таблица 2.1

Арифметические команды

INCA	DECA	NEGA	CMPA*	SUBA*	ADDA*	ADCA*	DAA
INCB	DECB	NEGB	CMPB*	SUBB*	ADDB*	ADCB*	MUL
INC**	DEC**	NEG**	CPD*	SUBD*	ADDD*		
INX	DEX		CPX*	SBCA*	ABA		FDIV
INY	DEY		CPY*	SBCB*	ABX		
INS	DES		CBA	SBA	ABY		IDIV

Примечания:
* – команды, использующие непосредственную, прямую, расширенную и индексную адресацию;
** – команды, использующие расширенную и индексную адресацию.

INCA, INCB, INC (opr)
DECA, DECB, DEC (opr)

S	X	H	I	N	Z	V	C
-	-	-	-	?	?	?	-

INS, DES

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

INX, INY, DEX, DEY

S	X	H	I	N	Z	V	C
-	-	-	-	-	?	-	-

Приведем примеры использования этих команд в порядке увеличения сложности.

Команды инкремента и декремента являются простейшими арифметическими операциями и служат соответственно для увеличения и уменьшения на единицу значения регистров общего назначения микроконтроллера или ячейки памяти.

В зависимости от типа операнда значение регистра статуса после выполнения команд может принимать различные значения. При работе с 8-разрядным операндом команды инкремента и декремента влияют на флаги отрицательного результата N, нуля Z и переполнения V. В случае если опе-

рандом является указатель стека, значение регистра статуса остается неизменным. При операциях с индексными регистрами команды инкремента и декремента влияют только на флаг нуля Z.

Как правило, команды INC и DEC используются для организации циклов. Тот факт, что эти команды не изменяют флаг переноса, используется при арифметических операциях над многобайтными числами.

Следующий простой пример иллюстрирует работу этих команд:

```
org   $8000
ldaa  #$10      ;   поместить в регистр A значение $10
inca            ;   увеличить на 1
tab             ;   поместить в регистр B
decb            ;   уменьшить B на 1
std   $10       ;   сохранить регистры A и B в ячейках $10 и $11
ldx   $10       ;   загрузить в регистр X
inx             ;   инкрементировать регистр X
des             ;   указателя стека
```

NEGA, NEGB, NEG (opr)

S	X	H	I	N	Z	V	C
-	-	-	-	?	?	?	?

Команда NEG замещает операнд его двоичным дополнением. Другими словами можно сказать, что результатом операции является изменение знака числа, представленного в дополнительном коде.

Продемонстрируем на примере эмуляцию команды INC через NEG и DEC:

```
org   $8000
nega            ; изменить знак числа
deca            ; увеличить на 1
nega            ; изменить знак числа
```

CMPA (opr), CMPB (opr), CPX (opr), CPY (opr), CPD (opr), CBA

S	X	H	I	N	Z	V	C
-	-	-	-	?	?	?	?

Команды сравнения используются при сравнении значения регистра со значением ячейки памяти или регистра. Фактически происходит операция вычитания ячейки памяти, указанной в качестве операнда, или регистра B (в случае команды CBA) из соответствующего регистра МК. Команды не оказывают влияния на операнды, изменяется лишь регистр статуса. В дальнейшем результат обычно используется командами перехода.

```
org   $8000
ldaa  #$10      ; инициализация регистров A и B
ldab  #$50
cba             ; сравнение регистров A и B
stab  $01
cmpb  $01       ; сравнение содержимого регистра B с ячейкой $01
```

ABA,
ADCA (opr), ADCB (opr),
ADDA (opr), ADDB (opr)

S	X	H	I	N	Z	V	C
-	-	?	-	?	?	?	?

ADDD (opr), SUBD (opr),
SBA,
SBCA (opr), SBCB (opr),
SUBA (opr), SUBB (opr)

S	X	H	I	N	Z	V	C
-	-	-	-	?	?	?	?

ABX, ABY

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

При выполнении команд сложения происходит суммирование содержимого регистра-приемника с непосредственно заданным значением, ячейкой памяти или другим регистром. В командах ADC к результату дополнительно прибавляется значение флага переноса. Результат сложения аккумуляторов командой ABA заносится в регистр A, результат сложения регистра B с индексным регистром – в соответствующий индексный регистр.

При выполнении команд вычитания происходит вычитание из регистра-приемника второго операнда (в случае SBA происходит вычитание регистра B из регистра A). Команды SBC дополнительно вычитают из регистра-приемника значение флага переноса.

Команды, учитывающие флаг переноса, как правило, используются при операциях над многобайтными числами. Ниже приводится пример сложения и вычитания двух 3-байтовых чисел, расположенных в ячейках $0...$2 и $3...$5 соответственно.

```
org   $8000
ldx   #01
ldd   0,x          ; сложение двух младших байтов
addd  3,x
std   0,x
dex                ; переход к третьему байту
ldaa  0,x
adca  3,x          ; сложение с учетом переноса
staa  0,x          ; записать результат
ldx   #$2
ldaa  0,x          ; вычитание с использованием SBA
ldab  3,x
sba
dex                ; переход к следующему байту
ldaa  0,x
sbca  3,x
staa  0,x
dex                ; переход к последнему байту
ldaa  0,x
sbca  3,x
staa  0,x          ; результат получен, записать последний байт
```

Двоично-десятичная коррекция после сложения командами ABA, ADDA и ADCA обеспечивает суммирование двух чисел, представленных в двоично-

DAA

S	X	H	I	N	Z	V	C
-	-	-	-	?	?	?*	?

*-значение не определено

десятичном формате. При этом флаг переноса используется в качестве старшего бита, обеспечивая получение корректного двоично-десятичного значения.

Фактически команда DAA после команд сложения действует следующим образом:

1) если содержимое младшей тетрады аккумулятора больше 9 или флаг полупереноса H установлен в «1», то к аккумулятору добавляется число 6;

2) если содержимое старшей тетрады аккумулятора стало после этого более 9 или установлен флаг переноса, то число 6 добавляется и к старшей тетраде аккумулятора.

```
org    $8000
ldaa   #$99        ; 99 в двоично-десятичном коде
ldab   #$20
aba                ; результат равен B9
daa                ; коррекция до двоично-десятичного значения: C = 1, A = $19
tab
ldaa    #0         ; использование clra недопустимо, так как будет сброшен
                   ; флаг переноса
adca    #0         ; D = $0119
```

MUL

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	?

Команда умножения производит беззнаковое умножение двух чисел, представленных в 8-разрядных аккумуляторах. Результат помещается в 16-разрядный аккумулятор D. Флаг переноса при этом устанавливается таким образом, что при выполнении команды ADCA #0 происходит округление старшего байта.

```
org    $8000
ldaa   #$10
ldab   #$68
mul                ;       $10 * $68 = $0680
adca   #0          ;       A = $7
```

IDIV

S	X	H	I	N	Z	V	C
-	-	-	-	-	?	0	?

Команда IDIV производит целочисленное деление аккумулятора D на индексный регистр X. После выполнения в регистр X заносится частное, а в регистр D – остаток от деления. При выполнении команды IDIV делимое обычно больше делителя.

FDIV

S	X	H	I	N	Z	V	C
-	-	-	-	-	?	?	?

Команда FDFV производит операцию дробного деления тех же аргументов. Фактически FDIV может быть представлена как умножение регистра D на 2^{16} с последующим выполнением команды IDIV, поэто-

му при выполнении этой команды делитель обычно больше делимого. Эти две команды очень редко используются на практике.

```
org   $8000
ldd   #1020      ;     D = 1020 ($3fc)
ldx   #512       ;     X = 512 ($200)
idiv             ;     D = 1 ($1), X = 508 ($1fc)
fdiv             ;     D = 129 ($81), X = 4 ($4)
```

Практическая часть

2.2. Задания

1. Напишите программу суммирования двух 16-разрядных чисел, представленных в BCD-формате, с учетом возможного переполнения.

2. Напишите программу суммирования регистров МК по следующей формуле: D = A + B + lo(X) + hi(X) + lo(Y) + hi(Y), где lo и hi – младший и старший байты соответствующих регистров.

3. Напишите программу вычитания содержимого регистров X и Y из регистра D.

4. Напишите программу сравнения ячеек памяти $0 и $1. Регистр A должен быть равен единице, если ячейки памяти равны.

5. Вычислите произведение двух ячеек памяти. Содержимое всех регистров должно остаться неизменным.

6. Напишите программу, позволяющую вычислить адрес элемента, находящегося в двухмерном массиве размерностью 3 × 3. Массив располагается по адресу $8100. Индекс задается регистрами A и B, где A – номер строки, B – номер столбца массива.

7. Напишите программу, которая преобразует число, заданное в регистре A, в восьмеричное представление этого числа в ASCII-коде.

8. Напишите программу, которая преобразует число, заданное в регистре A, в десятичное представление этого числа в ASCII-коде.

9. Просуммируйте содержимое двух ячеек памяти. Содержимое всех регистров должно остаться неизменным.

10. Вычислите разность содержимого регистров X и Y.

11. Вычислите произведение регистров X и Y.

12. Используя только команды TAB, SUBA, STAB, LDAB, DECA и XGDX, занесите в регистр A значение $FF.

13. Вычислите частное от деления содержимого индексного регистра X на содержимое индексного регистра Y. При этом все остальные регистры необходимо сохранить в начальных условиях.

14. Напишите программу сравнения 16-разрядных чисел, расположенных в ячейках памяти $0 и $2. Регистр A должен быть равен нулю, если ячейки памяти не равны.

2.3. Контрольные вопросы

1. Какие команды сложения вы знаете?
2. Какие методы адресации используют команды ABA, ADDA, ABY?
3. Какие команды вычитания вам известны?
4. Каким образом используется бит переноса в операции вычитания?
5. Над какими операндами могут выполняться команды INC, DEC?
6. Объясните отличие в выполнении команд ADD и ADC.
7. Где располагаются результаты команды FDIV и что они собой представляют?
8. Что может служить операндом команды ADCA?
9. Какой флаг устанавливается, если результат операции сложения превышает $FF?
10. Объясните, по какому принципу устанавливаются флаги переноса, нуля и переполнения в регистре статуса CCR при выполнении арифметических команд сложения и вычитания.
11. Объясните логику работы команд сложения/вычитания с учетом переноса/заема при обработке многобайтовых чисел.
12. Объясните логику работы команды DAA.
13. Чем отличаются команды FDIV и IDIV?

ЛАБОРАТОРНАЯ РАБОТА №3
ЛОГИЧЕСКИЕ КОМАНДЫ. КОМАНДЫ РАБОТЫ С БИТОВЫМИ ПОЛЯМИ. КОМАНДЫ СДВИГОВ

Цель работы. Разработать программу с использованием следующих групп команд:

- логические команды (операции «НЕ», «И», «ИЛИ», «исключающее ИЛИ»);
- команды работы с битовыми полями (установка и сброс битов);
- команды сдвигов (арифметический, логический и циклический сдвиги).

Теоретическая часть

3.1. Логические команды

Логические команды включают в себя действия булевой алгебры над аккумулятором или, в случае команды COM, – над ячейкой памяти, заданной при помощи расширенной или индексной адресации. Команды BITA и BITB по принципу работы схожи с командами ANDA и ANDB, но не изменяют содержимого аккумулятора (сравните CMPA и SUBA). Команды приведены в табл. 3.1.

Таблица 3.1

Логические команды

COMA COMB COM	ANDA ANDB	BITA BITB	ORAA ORAB	EORA EORB

COMA, COMB,
COM (opr)

S	X	H	I	N	Z	V	C
-	-	-	-	?	?	0	1

ANDA (opr), ANDB (opr),
BITA (opr), BITB (opr),
ORAA (opr), ORAB (opr),
EORA (opr), EORB (opr)

S	X	H	I	N	Z	V	C
-	-	-	-	?	?	0	-

Обычно логические команды применяются для выборочной установки, обнуления, дополнения и тестирования битов, что часто используется при работе с периферийными устройствами. Следующий пример показывает, каким образом можно перенести содержимое старшего и младшего битов порта C в старший и младший биты порта B. При этом младший бит порта C переносится с инверсией. (Программа написана таким образом, чтобы задействовать максимальное число логических команд, но не оптимизирована на скорость выполнения.)

```
org   $8000
ldaa  $1f03          ;     считать состояние порта C
ldab  $1f04          ;     считать состояние порта B
```

```
andb  #%01111110      ;     выделить неизменную часть
oraa  #%01111110      ;     установить все неиспользуемые биты  в «1»
coma                  ;     инвертировать значение аккумулятора
eora  #%10000000      ;     инвертировать значение старшего бита
aba                   ;     совместить результат (заметьте, что мы
                      ;     заблаговременно маскировали неиспользуемые
                      ;     биты, чтобы сейчас совместить два числа простым
                      ;     суммированием)
staa  $1f04           ;     вывести результат
```

3.2. Команды работы с битовыми полями

Команды работы с битовыми полями позволяют изменять указанные биты приемника (ячейки памяти или регистра статуса CCR), оставляя незадействованные биты нетронутыми. Список команд приведен в табл. 3.2.

Таблица 3.2

Команды работы с битовыми полями

SEC	SEI	SEV	BSET*
CLC	CLI	CLV	BCLR*

Примечание: * – команды, использующие прямую или индексную адресацию в качестве первого параметра и непосредственную – в качестве второго.

Команды, представленные в первых трех столбцах таблицы, устанавливают (SE?) или сбрасывают (CL?) отдельные флаги в регистре статуса, на которые указывает третья буква в мнемонике команды (C – флаг переноса, I – маскирование прерываний, V – флаг переполнения).

Приведем простой пример, показывающий один из способов занесения числа 1 в аккумулятор:

```
org   $8000
clra              ;  очистить аккумулятор
sec               ; установить флаг переноса
adca  #0          ;  прибавить его к аккумулятору
```

В реализации отладчика имеется одна особенность – если трассируемая команда запрещает прерывания, то выполнение программы будет продолжаться до тех пор, пока либо прерывания не будут разрешены, либо не произойдет прерывание по неправильному коду команды. В частности, это относится к командам SEI и SWI (эта команда будет рассмотрена в лабораторной работе №4).

BCLR (opr), BSET (opr)

S	X	H	I	N	Z	V	C
-	-	-	-	?	?	0	-

Также следует отметить, что если выполнение программы затянется, то отладчик выдаст сообщение об истечении времени ожидания ответа.

Почти в каждой программе требуется возмож-

ность манипуляции отдельными битами ячейки памяти. Так, блок регистров представляет собой по большей части битовые поля.

Команды BCLR и BSET в качестве первого операнда получают ячейку памяти, в которой соответственно сбрасываются или устанавливаются биты, указанные в маске, заданной непосредственно вторым параметром.

Приведем пример, демонстрирующий работу этих команд:

```
org    $8000
ldx    #$1f00          ;     настроить регистр X
bset   4,x,#$AA        ;     установить в «1» через один бит,  оставив
                        ;     незадействованные биты в прежнем состоянии
bclr   4,x,#$55        ;     установить в «0» остальные биты
```

3.3. Команды сдвигов

Список команд сдвигов представлен в табл. 3.3. Эти команды позволяют адресоваться к аккумулятору или ячейке памяти.

Команды сдвигов обычно подразделяют на три группы:

- арифметические сдвиги;
- логические сдвиги;
- циклические сдвиги.

Таблица 3.3

Команды сдвигов

ASIA/ LSLA ASLB/ LSLB ASLD/ LSLD ASL*/ LSL*	ASRA ASRB ASR*	LSRA LSRB LSRD LSR*	ROLA ROLB ROL*	RORA RORB ROR*

Примечание: * – команды, использующие расширенную или индексную адресацию.

ASLA, ASLB, ASL (opr),
ASLD, ASRA, ASRB,
ASR (opr)

S	X	H	I	N	Z	V	C
-	-	-	-	?	?	?	?

При выполнении арифметического сдвига сохраняется знак первоначального операнда. При выполнении команды ASR происходит расширение знакового разряда. Это позволяет использовать команду для деления знакового числа на 2^N. Однако для нечетных чисел деление не всегда является корректным (разница в результате может составлять единицу). При выполнении команды арифметического сдвига влево каждый раз при смене знакового бита устанавливается флаг V, а освободившиеся разряды заполняются нулями. Таким образом, становится возможным при помощи команд ASR производить знаковое умножение числа на 2^N.

Флаг переноса устанавливается в соответствии с отбрасываемым битом. Приведем пример, демонстрирующий работу этих команд:

```
org   $8000
ldaa  #%00101001
ldx   #$1f00
staa  4,x        ;   вывести содержимое аккумулятора
asl   4,x        ;   умножить на два
asl   4,x        ;   еще раз умножить на два  (происходит переполнение)
asr   4,x        ;   разделить на два
asr   4,x        ;   разделить на два
```

Логические сдвиги производят сдвиг содержимого аккумулятора или ячейки памяти влево (LSL) или вправо (LSR). При этом освободившиеся разряды всегда заполняются нулями. Команды групп ASL и LSL выполняют в точности одинаковые действия и имеют одинаковые коды операций, поэтому покажем лишь отличие команд ASR от команд LSR:

LSLA, LSLB, LSL (opr), LSLD, LSRA, LSRB, LSR (opr), LSRD

S	X	H	I	N	Z	V	C
-	-	-	-	?	?	?	?

```
org   $8000
ldaa  #%10101010
staa  $1f04      ; установить в «1» через один бит
asr   $1f04      ; арифметический сдвиг вправо:
asr   $1f04      ; старший бит сохраняется
lsr   $1f04      ; логический сдвиг:
lsr   $1f04      ; старший бит заполняется «0»
```

ROLA,ROLB, ROL (opr), RORA,RORB, ROR (opr)

S	X	H	I	N	Z	V	C
-	-	-	-	?	?	?	?

Команды циклического сдвига позволяют осуществить операцию логического сдвига над многобайтовыми числами. Отличие этих операций от операций логического сдвига состоит в том, что освободившийся разряд заполняется не нулем, а состоянием флага переноса C. Рассмотрим пример использования этих команд для сдвига 4-байтового числа, расположенного в ячейках 0...3, влево:

```
org   $8000
ldx   #0
lsl   3,x
rol   2,x
rol   1,x
rol   0,x
```

Практическая часть

3.4. Задания

1. Напишите программу, осуществляющую сдвиг влево трех ячеек памяти таким образом, чтобы выдвигаемый из старшей ячейки памяти бит становился на место младшего бита в младшей ячейке.

2. Произведите операцию «логическое ИЛИ» над регистрами X и Y.

3. Напишите программу, производящую обмен старшей и младшей тетрады аккумулятора A.

4. Напишите программу, создающую зеркальное отображение битовой карты регистра A в регистре B.

5. Реализуйте подсчет установленных в регистре A битов с занесением суммы в регистр B.

6. Напишите программу умножения двух двоично-десятичных 8-разрядных чисел.

7. Произведите операцию «логическое И» над регистрами X и Y.

8. Напишите тремя способами установку битов 2 и 3 в ячейке памяти $10.

9. Напишите программу, копирующую содержимое регистров A и B в регистр X таким образом, чтобы старшая тетрада регистра A и старшая тетрада регистра B составляли старший байт регистра X, а младшие тетрады – младший.

10. Напишите программу, позволяющую инвертировать те биты регистра A, которые сброшены в регистре B.

11. Напишите программу, сбрасывающую биты в регистре A, если соответствующие биты регистров A и B установлены. Остальные биты должны оставаться в исходном состоянии.

12. Напишите программу, которая в четные биты регистра X записывает биты регистра A, а в нечетные – регистра B.

13. Напишите программу, копирующую регистр A в регистр B с обратным порядком следования битов, инвертируя нечетные биты.

14. Напишите программу, заполняющую ячейки памяти $0 ... $7 соответствующими битами регистра A, т. е., например, если бит 0 в регистре A сброшен, то в ячейку $0 записывается нуль, если установлен – $FF.

15. Установите четвертый и пятый биты в регистре A с помощью команды BSET.

3.5. Контрольные вопросы

1. Каков результат выполнения приведенной программы?

```
sec
clra
adda   #0
```

2. Какие методы адресации применимы к командам циклического сдвига?

3. Расскажите о командах работы с битами регистра CCR.

4. Какие особенности работы с отладчиком следует учитывать при отладке программ, запрещающих прерывания?

5. В чем разность команд ASR и LSR?

6. Можно ли использовать команду ROLA вместо команд ASLA, LSLA?

7. Какие логические команды вы знаете?

8. Каким образом реализуется команда ASRD (сдвиг регистра D на один байт вправо)?

9. На какие группы можно подразделить команды сдвигов?

10. Дайте определение команд логического сдвига.

11. Чем отличается команда COM от команды NEG?

12. Каким образом можно сэмулировать команду COM, пользуясь командами, изученными в данной лабораторной работе?

13. Какие параметры имеет команда BSET?

14. Чем отличается команда ORAA от команды EORA?

ЛАБОРАТОРНАЯ РАБОТА №4
КОМАНДЫ ПЕРЕДАЧИ УПРАВЛЕНИЯ. СПЕЦИАЛЬНЫЕ КОМАНДЫ

Цель работы. Рассмотреть команды передачи управления, служащие для ветвления программы за счет изменения регистра программного счетчика PC, и специальные команды STOP и WAI, служащие для организации эффективной работы микроконтроллера в системах, критичных по параметрам потребляемой мощности.

Теоретическая часть

4.1. Команды передачи управления

Команды передачи управления можно разделить на четыре группы:

- безусловного перехода (JMP, BRA, BRN, NOP);
- работы с подпрограммами (JSR, BSR, RTS);
- условного перехода (BEQ, BNE, BMI, BPL, BCS/BLO, BCC/BHS, BVS, BVC, BGT, BGE, BLT, BLE, BLS, BHI, BRSET, BRCLR);
- работы с прерываниями (SWI, RTI).

Список команд передачи управления представлен в табл. 4.1. Все они не оказывают влияния на состояние регистра статуса.

Таблица 4.1

Команды передачи управления

JMP*	BEQ**	BCS/BLO**	BOT**	BLS**	JSR*****	SWI***
BRA**	BNE**	BCC/BHS**	BGE**	BHI**	BSR**	RTI***
BRN**	BMI**	BVS**	BLT**	BRSET****	RTS***	
NOP***	BPL**	BVC**	BLE**	BRCLR****		

Примечания:
* – команды, использующие расширенную и индексную адресацию;
** – команды, использующие относительную адресацию;
*** – команды, использующие неявную адресацию;
**** – команды, использующие смешанную адресацию: первый операнд использует либо прямую, либо индексную адресацию, второй – относительную;
***** – команды, использующие прямую, расширенную и индексную адресацию.

Команды безусловного перехода служат для передачи управления другому участку программы независимо от состояния регистра статуса микроконтроллера и содержимого ячеек памяти. Рассмотрим работу этих команд более подробно на следующем примере:

JMP, BRA, BRN, NOP

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

```
        org   $8000
        ldab  #$02        ;     выбор варианта ветвления программы
        ldx   #ways       ;     занести в регистр X адрес таблицы переходов
p0      ldy   0,x         ;     считать значение в регистр Y
        jmp   0,y         ;     вызвать подпрограмму по адресу, находящемуся в
                          ;     регистре Y
p1      nop               ;     задержка в два такта
        inx                     увеличить регистр X на 2 для выборки следую-
                                щего
                          ;     адреса из таблицы переходов
        bra   p0          ;     перейти на метку p0
p2      bra   p1          ;     перейти на метку p1
p3      brn   *           ;     задержка в три такта
ways    fdb   p1,p2,p3
```

Выполните программу в пошаговом режиме. Обратите внимание, что команда «jmp 0,x» выполнится два раза, при этом в первом случае переход будет осуществлен на метку p2, а во втором – p3.

JSR, BSR, RTS

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

Команды работы с подпрограммами позволяют выделять часто используемую последовательность действий в подпрограмму. При переходе к подпрограмме (JSR, BSR) в стеке сохраняется адрес следующей за текущей команды и регистр PC изменяется по правилам команд безусловного перехода. При выходе из подпрограммы по команде RTS происходит выборка из стека адреса возврата. Работу этих команд можно проследить на примере программы, размещающей адрес своей второй команды в ASCII-формате в ячейках $0...3:

```
            org   $8000
            bsr   p1          ;     переход на следующую команду
p1          pulx              ;     получить в регистр X адрес p1 ($8002)
            xgdx              ;     расписать адрес в формате ASCII
            ldy   #0          ;     в ячейках $0 ... $3
            bsr   bin2ascii   ;     вызвать подпрограмму, сохраняющую
                              ;     регистр
                              ;     A в формате ASCII в ячейках (y), (y+1)
            tba               ;     переписать регистр B в регистр A
            iny               ;     и расписать его в ячейках $2, $3
            iny
            bsr   bin2ascii
            bra   *
bin2ascii   pshb              ;     сохранить в стеке регистр B
            tab               ;     скопировать в него регистр A
            lsra              ;     выделить в A старшую тетраду
            lsra
```

```
            lsra
            lsra
            bsr   hex2ascii   ;     преобразовать число 0 ... f в ASCII-код
            staa  0,y
            tba               ;     повторить для младшей тетрады
            anda  #$f
            bsr   hex2ascii
            staa  l,y
            pulb
            rts
hex2ascii   adda  #0          ;     преобразовать числа из диапазона
            daa
            adda  #$f0
            adca  #$40
            rts
```

Команды условного перехода служат для передачи управления в зависимости от состояния регистра CCR или значения ячейки памяти (BRSET и BRCLR).

Иногда команды условного перехода, выполняющие передачу управления в зависимости от состояния регистра статуса, подразделяют на три группы (табл. 4.2):

- знаковые;
- беззнаковые;
- простые.

Таблица 4.2

Команды условного перехода

Условие	Логическая функция	Мнемоника	Противоположное действие		Тип
r>m	Z+(N⊕V)=0	BGT	r≤m	BLE	Знаковый
r≥m	N⊕V=0	BGE	r<m	BLT	Знаковый
r=m	Z=l	BEQ	r≠m	BNE	Знаковый
r≤m	Z+(N⊕V)=1	BLE	r>m	BGT	Знаковый
r<m	N⊕V=1	BLT	r≥m	BGE	Знаковый
r>m	C+Z=0	BHI	r≤m	BLS	Беззнаковый
r≥m	C=0	BCC/BHS	r<m	BCS/BLO	Беззнаковый
r=m	Z=l	BEQ	r≠m	BNE	Беззнаковый
r≤m	C+Z=1	BLS	r>m	BHI	Беззнаковый
r<m	C=l	BCS/BLO	r≥m	BCC/BHS	Беззнаковый
Перенос	C=l	BCS/BLO	Нет переполнения	BCC/BHS	Простой
Отрицательный	N=1	BMI	Положительный	BPL	Простой

Окончание табл. 4.2

Условие	Логическая функция	Мнемоника	Противоположное действие		Тип
Пере-полнение	V=l	BVS	Нет пере-полнения	BVC	Простой
r=0	Z=l	BEQ	$r \neq 0$	BNE	Простой

Покажем один из способов организации циклов с помощью команд условного перехода на примере сложения двух 4-байтовых чисел:

```
        org   $8000
        ldx   #$3       ;   конечный адрес первого числа
        ldy   #$7       ;   конечный адрес второго числа
        ldab  #$4       ;   размер числа
        clc             ;   сброс флага переноса
loop    ldaa  0,x       ;   сложить два байта
        adca  0,y
        staa  0,x
        dex             ;   перейти к следующему байту
        dey
        decb            ;   уменьшить число обрабатываемых байтов
        bne   loop      ;   на 1 и повторить, если не 0
```

Другой пример показывает использование команд переходов при проверке правильности даты, записанной в виде BCD-числа в ячейках $0...$3 в формате ДДММГГГГ (т. е. в ячейке $0 хранится день, в ячейке $1 – месяц, а в ячейках $2 и $3 – столетие и год в столетии соответственно):

```
        org   $8000
        ldx   #0        ;   установить указатель на начало даты
        ldd   0,x       ;   загрузить в A и B день и месяц
        tstb            ;   если день или месяц равен 0 или
        beq   invalid   ;   месяц больше 13, то дата некорректна
        tsta
        beq   invalid
        cmpb  #$13
        bhs   invalid
        cmpb  #$2       ;   если это не февраль, то переход
        bne   lab2      ;   на выборку числа дней из таблицы
        ldab  3,x       ;   проверка високосного года
        bsr   bcd2bin
        andb  $03       ;   получение остатка от деления на 4
        bne   lab2      ;   год невисокосный, если не делится нацело
                        ;   на 4
```

```
            ldab  #$29
            bra   lab1
lab2        bsr   bcd2bin     ;     преобразование месяца в двоичный код
            decb
            ldy   #dpm        ;     настроить указатель на список дней в
                              ;     месяце
            aby
            ldab  0,y         ;     занести в B число дней в месяце
lab1        cba               ;     если число дней больше вычисленного,
            bhi   invalid     ;     то дата некорректна
valid       clc               ;     год может быть любым, поэтому проверка
            bra   done        ;     не производится и возвращается признак
                              ;     успешной проверки
invalid     sec               ;     установка флага ошибки
done        bra   *
bcd2bin     psha              ;     преобразование числа в формате BCD,
            tba               ;     заданного в регистре B, в двоичный
            anda  #$0F        ;     формат
            lsrb              ;     выделение старшей тетрады
            lsrb
            lsrb
            lsrb
            lslb              ;     умножение на 10 и сложение
            aba               ;     с регистром A
            lslb
            lslb
            aba
            tab
            pula
            rts
dpm         fcb   $31,$28,$31,$30,$31,$30,$31,$31,$30,$31,$30,$31
```

BRSET (opr),
BRCLR (opr)

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

Команды условного перехода, выполняющие передачу управления в зависимости от значения ячейки памяти, используются в циклах ожидания изменения какого-либо регистра управления, опросе внешних линий данных или работе с другими битовыми данными. Простой пример показывает, каким образом можно обеспечить отображение информации:

```
            org   $8000
            ldx   #$1f00      ;     установить регистр X на базовый адрес
            brset 3,x,#$01,p1 ;     если младший бит установлен, тогда
                              ;     перейти к программе ожидания сброса
p0          bsr   display     ;     отобразить информацию
```

```
            brclr  3,x,#$01,*  ;      ждать установки бита
                               ;      * – адрес текущей команды
p1          bsr    display
            brset  3,x,#$01,*  ;      ждать сброса бита
            bra    p0          ;      следующий цикл
display     ldaa   3,x         ;      отобразить состояние
            staa   4,x
            rts
```

SWI

S	X	H	I	N	Z	V	C
-	-	-	1	-	-	-	-

RTI

S	X	H	I	N	Z	V	C
?	?*	?	?	?	?	?	?

* - значение может быть изменено только из 1 в 0.

Команды работы с прерываниями предназначены для входа или выхода из прерывания.

Иногда бывает необходимо выполнить программную реализацию прерывания. Конечно, это можно реализовать через подпрограмму, в начале которой выполняется сохранение регистров в стеке, однако в данном случае снижается эффективность работы программы в целом. Для выполнения этой задачи служит команда генерации программного прерывания SWI. Она выполняет последовательное сохранение в стеке регистров PC+1, Y, X, D, CCR, запрещает маскируемые прерывания (устанавливает бит I в регистре CCR) и передает управление на подпрограмму, адрес которой находится в таблице векторов прерывания. Адрес обработчика прерывания SWI должен быть расположен по адресу $fff6, если работа происходит в нормальном режиме работы, или по адресу $bff6, если в специальном. В режиме bootstrap по адресам $bf40 ... $bfff находится bootstrap ПЗУ, в котором векторы прерываний указывают на ячейки памяти внутреннего ОЗУ. Таким образом, в исследуемых в данной лабораторной работе экспериментах для задания окончательного адреса перехода мы будем использовать адреса $f4 ... $f6, так как в них будет расположена команда безусловного перехода JMP.

Для возврата из прерывания применяется команда RTI. Она восстанавливает значения регистров из запомненных в стеке, тем самым осуществляется возврат к прерванной программе с сохранением состояния регистров.

Небольшая программа демонстрирует работу этих двух команд:

```
            org    $00f4       ;      установка вектора обработчика
            jmp    ih          ;      прерывания SWI
            org    $8000
gen_int     ldaa   #$55
            swi                ;      вызов прерывания
            coma
            swi
            coma
            swi
            coma
```

```
            swi
done        bra   *
ih          ldx   #$1f04      ;     обработчик прерывания осуществляет
            staa  0,x         ;     отображение числа из регистра A
            ldy   #$500       ;     и задержку ≈10 мс
delay       dey
            bne   delay
            rti
```

Попробуйте выполнить эту программу в пошаговом режиме (как было указано в лабораторной работе №3, обработчик прерывания будет выполняться за один шаг) и с установкой точек останова на метке ih. Посмотрите содержимое области данных выше указателя стека (стекового фрейма) и убедитесь, что все регистры процессора были сохранены.

4.2. Специальные команды

Как отмечалось ранее, к специальным командам относятся команды, переводящие контроллер в режим низкого потребления энергии.

Команда WAI переводит контроллер в режим ожидания первого немаскированного прерывания. При этом сохранение регистров происходит в момент выполнения команды WAI, а не в момент обнаружения прерывания.

Команда STOP выполняет остановку всех внутренних генераторов микроконтроллера и перевод системы в режим минимального энергопотребления. В случае если установлен бит S регистра CCR, то команда STOP выполняется как команда NOP. Восстановление системы из режима минимального энергопотребления может произойти при появлении прерываний от RESET, XIRQ или немаскированного прерывания IRQ. В случае когда установлен бит X регистра CCR, маскирующий прерывание XIRQ, и происходит это прерывание, выполнение программы происходит со следующей за STOP командой. В некоторых масках семейства MC68HC11 была допущена ошибка, приводящая к неправильному интерпретированию кода команды в некоторых особых случаях, поэтому фирма Motorola рекомендует перед командой STOP помещать команду NOP, таким образом исключая эту ошибку.

Практическая часть

4.3. Задания

1. Напишите программу, осуществляющую сложение двух 4-байтовых чисел, представленных в формате BCD.

2. Реализуйте перевод 2-байтового числа в формате BCD в двоичный формат.

3. Напишите программу, копирующую блок данных, расположенных по ад-

ресам $8200 ... $8220, в соответствующие ячейки $0000 ... $0020. При этом данные перезаписываются только в том случае, если бит 3 в соответствующей ячейке памяти сброшен.

4. Напишите программу подсчета суммы 8-битовых беззнаковых чисел, расположенных в ячейках $8200 ... $82ff. Результат поместите в регистр Y.

5. Произведите сортировку по возрастанию чисел, расположенных в ячейках $8200 ... $82ff.

6. Напишите программу, подсчитывающую количество установленных битов в ячейках памяти $8200 ... $821f. Результат поместите в регистр X.

7. Произведите сортировку по убыванию чисел, расположенных в ячейках $8200 ... $82ff.

8. Напишите программу, производящую подсчет количества нечетных чисел в ячейках $8200 ... $82ff.

9. Напишите программу преобразования двоичных чисел, расположенных в ячейках $8200 ... $821f, в BCD-формат. Результат разместите в ячейках $8220 ... $825f.

10. Напишите программу подсчета суммы 8-битовых знаковых чисел, расположенных в ячейках $8200 ... $82ff.

11. Произведите операцию «логическое ИЛИ» между битом 2 и битом 5 для ячеек памяти, расположенных по адресам $8200 ... $821f, результат при этом должен быть записан в бит 3 соответствующей ячейки.

12. Произведите обмен старших тетрад ячеек, расположенных в блоках $8200 ... $821f и $8220 ... $823f.

13. Перестройте массив данных размером 256 байт в обратном порядке, т. е. первый байт меняется местами с последним, второй с предпоследним и т. д.

14. Напишите программу, осуществляющую сдвиг массива данных размером 64 байта на 4 бита влево.

15. Напишите программу, зеркально перестраивающую биты в массиве данных размером 256 байт, т. е. нулевой бит первого элемента массива становится последним битом последнего элемента, первый бит первого элемента – шестым битом последнего элемента и т. д.

4.4. Контрольные вопросы

1. Каково различие между командами JMP и BRA?

2. Объясните различие между командами WAI и STOP.

3. Каким образом можно реализовать переход к подпрограмме, не используя команды BSR и JSR?

4. Произойдет ли вызов программного прерывания при установленном флаге I в регистре CCR?

5. Какие команды относятся к знаковым командам условного перехода?

6. Какие виды переходов вам известны?

7. Каково назначение команд BLE, BSR, BCS, BRCLR?

8. Сколько и какие операнды используются командами условного перехода по состоянию бита?

9. Реализуйте (примерно) команды BRCLR и BRSET через другие команды.

10. Каков результат выполнения приведенного фрагмента программы?

```
        ldaa   #34
        ldab   #$34
        cba
        bmi    p2
        beq    p3
        bra    done
p2      ldaa   #45
        bra    done
p3      ldaa   #$23
done    clrb
```

11. Можно ли выполнить переход, аналогичный переходу по команде BCS, используя команды BNE и BLE?

12. Каким образом можно осуществить корректный выход из подпрограммы, не используя команду RTS?

13. Какие команды относятся к беззнаковым командам условного перехода?

14. Для какой цели используются команды WAI и STOP?

15. Каково назначение команд BLE, RTI, JSR, BEQ?

ПРИЛОЖЕНИЕ 1

СИСТЕМА КОМАНД

Таблица П.1

Команды пересылки

Мнемокод	Команда	Операция
LDA(A, B) (opr) LD(D, S, X, Y) (opr)	Загрузка A или B Загрузка D, SP, X или Y	(M) → A или B (M) → Dh, SPh, Xh илиYh (M+1) →Dl, SPl, Xlили Yl
STA(A, B) (opr) ST(D, S, X, Y) (opr)	Запись A или B в память Запись D, SP, X или Y в память	A или B → M Dh, SPh, Xh или Yh → (M) Dl, SPl, Xl или Yl → (M+1)
PSH(A, B) PSH(X, Y)	Запись A или B в стек Запись X или Y в стек	A → SP, (SP – 1) → SP Xl или Yl → SP, (SP – 1) → SP Xh или Yh → SP, (SP – 2) → SP
PUL(A, B) PUL(X, Y)	Загрузка A или B из стека Загрузка X или Y из стека	SP+1 → SP, (SP) →A или B SP+1 → SP, (SP) →Xh или Yh SP+1 → SP, (SP) →Xl или Yl
TAB TAP TBA TPA TS(X, Y) T(X, Y)S XGD(X, Y)	Пересылка A в B Пересылка A в CCR Пересылка B в A Пересылка CCR в A Пересылка SP в X или Y Пересылка X или Y в SP Обмен D с X или Y	A → B A → CCR B → A CCR → A SP → (X, Y) (X, Y) → SP D ↔ (X, Y)
CLR(A, B) CLR (opr)	Запись 0 в A или B Запись 0 в M	$00 → (A, B) $00 → M

Таблица П.2

Команды битовых операций и изменения признаков

Мнемокод	Команда	Операция
BCLR (opr) #im8	Установка битов в «0»	Bi → 0
BSET (opr) #im8	Установка битов в «1»	Bi → 1
CLC	Установка признака C = 0	0 → C
CLI	Установка признака I = 0	0 → I
CLV	Установка признака V = 0	0 → V
SEC	Установка признака C = 1	1 → C
SLI	Установка признака I = 1	1 → I
SEV	Установка признака V = 1	1 → V

Таблица П.3

Команды арифметических операций и сравнения

Мнемокод	Команда	Операция
ADD(A, B, D) (opr)	Сложение A, B или D с (M)	(A, B, D) + (M), A, B, D
ADC(A, B) (opr)	Сложение A или B с (M) с учетом переноса	(A, B) + (M) + C → A, B,
ABA	Сложение A с B	A + B → A
AB(X, Y)	Сложение X или Y с B	(X, Y) + B → X, Y
DAA	Десятичная коррекция сложения	
SUB(A, B, D) (opr)	Вычитание (M) из A, B или D	(A, B, D) – (M) A, B, D
SUBC(A, B) (opr)	Вычитание (M) из A или B с учетом заема	(A, B) – (M) – C → A, B
MUL	Умножение A на B	A × B → D
IDIV	Деление D на X (целое)	D/X → X, r → D
FDIV	Деление D на X (дробное)	D/X → X, r → D
NEG (opr)	Изменение знака (M)	0 – (M) → (M)
NEG(A, B)	Изменение знака A или B	0 – A, B → A, B
CBA	Сравнение A с B	A – B
CMP(A, B) (opr)	Сравнение A или B с (M)	(A, B) – (M)
CP(D, X, Y)	Сравнение D, X или Y с (M)	(D, X, Y) – (M)
TST (opr)	Тестирование (M)	(M) – 0
TST(A, B)	Тестирование A или B	(A, B) – 0
INC (opr)	Инкремент (M)	(M) + 1 → (M)
INC (A, B, X, Y, S)	Инкремент A, B, X, Y или SP	(A, B, X, Y, SP) + 1 → (A, B, X, Y, SP)
DEC (opr)	Декремент (M)	(M) – 1 → (M)
DEC (A, B, X, Y, S)	Декремент A, B, X, Y или SP	(A, B, X, Y, SP) – 1 → (A, B, X, Y, SP)

Таблица П.4

Команды логических операций и сдвигов

Мнемокод	Команда	Операция
AND(A, B) (opr)	«Логическое И» содержимого A или B с (M)	$(A, B) \wedge (M) \rightarrow A, B$
COM (opr)	Инверсия (M)	$(M) \rightarrow M$
COM(A, B)	Инверсия A или B	$(A, B) \rightarrow A, B$
ORA(A, B) (opr)	«Логическое ИЛИ» содержимого A или B с (M)	$(A, B) \vee (M) \rightarrow A, B$
EOR(A, B) (opr)	«Исключающее ИЛИ» содержимого A или B с (M)	$(A, B) \oplus (M) \rightarrow A, B$
BIT(A, B) (opr)	Побитовое тестирование A или B с (M)	$(A, B) \wedge (M)$
ASL (opr), LSL (opr)	Арифметический (логический) сдвиг влево (M)	
ASL(A, B, D), LSL(A, B, D)	Арифметический (логический) сдвиг влево A, B или D	
ASR (opr)	Арифметический сдвиг вправо (M)	
ASR(A, B)	Арифметический сдвиг вправо A или B	
LSR (opr)	Логический сдвиг вправо (M)	
LSR(A, B, D)	Логический сдвиг вправо A, B или D	
ROL(opr)	Циклический сдвиг влево (M)	
ROL(A, B)	Циклический сдвиг влево A или B	
ROR (opr)	Циклический сдвиг вправо (M)	
ROR(A, B)	Циклический сдвиг вправо	

Таблица П.5

Установка значений признаков

Команды	Признаки				
	H	N	Z	V	C
ABA, ADCA, ADCB, ADDA, ADDB	+	+	+	+	+
ADDD, ASL (LSL), ASLA (LSLA) ASLB (LSLB), ASLD (LSLD), ASR, ASRA, ASRB, CBA, CMPA.CMPB, CPD, CPX, CPY, DAA, NEG, NEGA, NEGB, ROL, ROLA, ROLB, ROR, RORA,RORB, SBA, SBCA, SBCB, SUBA, SUBB, SUBD,	–	+	+	+	+
LSR, LSRA, LSRB, LSRD	–	0	+	+	+
ANDA, ANDB, BCLR, BSET, BITA, BITB, EORA, EORB, LDAA, LDAB, LDD, LDS, LDX, LDY, ORAA, GRAB, STAA, STAB, STD, STS, STX, STY, TAB, TBA	–	+	+	0	–
TST, TSTA,TSTB	–	+	+	0	0
CLR,CLRA, CLRB	–	0	1	0	0
COM, COMA, COMB	–	+	+	0	1
DFC, DECA DECB, INC, INCA, INCB	–	+	+	+	–
DEX, DEY, INX, INY	–	–	+	–	–
MUL	–	–	–	–	+
FDIV	–	–	+	+	+
IDIV	–	–	+	0	+
CLC	–	–	–	–	0
CLV	–	–	–	0	–
SEC	–	–	–	–	1
SEV	–	–	–	1	–

Примечания:

«+» – установка значения по результату операции;

«–» – значение остается неизменным;

0, 1 – установка соответствующих значений признаков.

Таблица П.6

Команды управления программой и процессором

Мнемокод	Команда	Операция
JMP (opr) Bcc r8	Безусловный переход Условное ветвление	EA → PC PC + 2 + r8 → PC, при cc
BRA r8 BRN r8	Безусловное ветвление Отсутствие ветвления	PC + 2 + r8 → PC PC + 2 → PC
BRCLR #im8, (opr), r8 BRSET #im8 (opr), r8	Ветвление при bn = 0 Ветвление при bn = 0	PC + 2 + r8 → PC, при bn = 0 PC + 2 + r8 → PC, при bn = l
JSR (opr)	Переход к подпрограмме	PC + 2 или 3 → PC, PCl → (SP), SP – 1 → SP PCh → (SP), SP – 1 → SP EA → PC
BSR r8	Ветвление в подпрограмме	PC + 2 или 3 → PC, PCl → (SP), SP – 1 → SP PCh → (SP), SP – 1 → SP PC + r8 → PC
RTS	Возврат из подпрограммы	SP + 1 → SP, (SP) → PCh SP + 1 → SP, (SP) → PCl
SWI	Программное прерывание	PC + 1 → PC, PCl → (SP), SP – 1 → SP, PCh → (SP), SP – 1 → SP, Yl → (SP), SP – 1 → SP, Yh → (SP), SP – 1 →SP, Xl → (SP), SP – 1 → SP, Xh → (SP), SP – 1 → SP, A → (SP), SP – 1 → SP, B → (SP), SP – 1 → SP, CCR → (SP), SP – 1 → SP, V → PC, 1 → I
RTI	Возврат из прерывания	SP + 1 → SP, (SP) → CCR, SP + 1 → SP, (SP) → B, SP + 1 → SP, (SP) → A, SP + 1 → SP, (SP) → Xh, SP + 1 → SP, (SP) → Xl, SP + 1 → SP, (SP) → Yh, SP + 1 → SP, (SP) → Yl, SP + 1 → SP, (SP) → PCh SP + 1 → SP, (SP) → PCl
NOP	Отсутствие операций	PC + 1 → PC

Окончание табл. П.6

Мнемокод	Команда	Операция
WAI	Переход в режим ожидания до прерывания	PC + 1 → PC, PCl → (SP), SP – 1 → SP, PCh → (SP), SP – 1 → SP, Yl → (SP), SP – 1 → SP, Yh → (SP), SP – 1 → SP, Xl → (SP), SP – 1 → SP, Xh → (SP), SP – 1 → SP, A → (SP), SP – 1 → SP, B → (SP), SP – 1 → SP, CCR → (SP), SP – 1 → SP
STOP	Переход в режим остановки	Остановка генератора тактовых импульсов
TEST	Тестирование (выполняется на заводе-изготовителе в специальном режиме тестирования)	

Таблица П.7

Мнемокоды и условия выполнения команд условных ветвлений

Мнемокод	Проверяемое условие	Значение
BNE	Не равно (ненулевой результат)	Z = 0
BEQ	Равно (нулевой результат)	Z = 1
BHI	Выше	(Z ∨ C) = 0
BLS	Ниже или равно	(Z ∨ C) = 1
BHS (BCC)	Выше или равно (нет переноса)	C = 0
BLO (BCS)	Ниже (есть перенос)	C = 1
BPL	Положительный результат	N = 0
BMI	Отрицательный результат	N = 1
BCE	Больше или равно	N ⊕ V = 0
BLT	Меньше	N ⊕ V = 1
BOT	Больше	Z ∨ (N ⊕ V) = 1
BLE	Меньше или равно	Z ∨ (N ⊕ V) = 1

ПРИЛОЖЕНИЕ 2

ПРИМЕР ПРОГРАММЫ

В приложении приведен пример программы, которая реализует следующую математическую функцию:

$$F = \frac{a^2 + b^2 + c^2}{a + 2} \cdot d,$$

где a, b, c, d – переменные, принимающие значения 0...255.

Ответ формируется в следующих регистрах:

X – старшая часть;

Y – младшая часть.

```
org $8000    ; адрес размещения
             ; программы в памяти
a equ 3      ; инициализация
b equ 5      ; переменных
c equ 7
d equ 9

 ldaa #a
 ldab #a
 mul
 std $8210   ; a*a

 ldaa #b
 ldab #b
 mul
 std $8220   ; b*b

 ldaa #c
 ldab #c
 mul
 std $8230   ; c*c

 clra
 clrb
 xgdx
 clra
 clrb        ; x = 0, d = 0

 ldx #a
 ldab #2
 abx
 stx $8240   ; x = a + 2

 ldd $8210
 idiv
 stx $8210
 std $8250   ; остаток от a*a/a + 2
 clra
 clrb
 xgdx
 clra
 clrb        ; x = 0, d = 0

 ldd $8220
 ldx $8240
 idiv
 stx $8220
 std $8260   ; остаток от b*b/a + 2
 clra
 clrb
 xgdx
 clra
 clrb        ; x = 0, d = 0

 ldd $8230
 ldx $8240
 idiv
 stx $8230
 std $8270   ; остаток от c*c/a + 2
```

```
ldd $8250       ; 1-й остаток в d
addd $8260      ; 1-й + 2-й
                ; остаток
addd $8270      ; 1-й + 2-й + 3-й
                ; остаток
ldx $8240
idiv
std $8250       ; сохраняем пос-
                ; ледний остаток
xgdx
addd $8210
addd $8220
addd $8230

std tw
ldaa #d
mul
std m2
ldab tw
ldaa #d
mul
addb m2
adca #0
std m1

ldx m1
ldy m2

stx $8290       ; старшая часть
                ; результата
sty $8280       ; младшая
                ; часть результата
ldd $8250
ldaa #d
mul
std $8260       ; младшая часть
                ; от c*d
ldd $8250
ldab #d
mul
std $8270       ; старшая часть
                ; от c*d
pshb
pula
clrb
addd $8260
std $8270       ; c*d

ldx $8240
idiv
std $8210       ; остаток от
                ; всех остатков
stx $8220       ; добавить к
                ; результату
ldd $8280
clra
addd $8220      ; прибавка
                ; точности
pshb
psha
pulb
clra
addd $8290
xgdx            ; формируем
pula            ; результат
clrb
xgdy
xgdy
psha
xgdx
pshb
psha
clra
psha
pulx
puly

clra
clrb

loop:           ; бесконечный
 bra loop       ; цикл
 rts

tw dw 0
m1 db 0
m2 db 0
m3 db 0
```

ЛИТЕРАТУРА

1. Белевич, А. Ю. Микроконтроллеры семейства 68300 фирмы Motorola / А. Ю. Белевич, О. В. Сердюков, Г. Ю. Костин // Chip News. – 1996. – №2. – С. 32–36.

2. Бродин, В. Б. Технология проектирования микропроцессорных контроллеров / В. Б. Бродин // Электроника и компоненты. – 1997. – №1. – С. 8–9 ; №2. – С. 7–9.

3. Бродин, В. Б. Эмуляторы 8-разрядных ОЭВМ к IBM PC // Библиотека информационной технологии : сб. статей / В. Б. Бродин, А. В. Калинин ; под ред. Г. Р. Громова. – М. : ИнфоАрт, 1991. – Вып. 3. – С. 222–229.

4. Кобахидзе, Ш. Средства разработки и отладки для однокристальных микроконтроллеров / Ш. Кобахидзе, А. Тамазов // Chip News. – 1996. – №2. – С. 37–43.

5. Корнеев, В. Современные микропроцессоры / В. Корнеев. – СПб. : БХВ-Петербург, 2003. – 3-е изд. – 448 с.

6. Коффрон, Дж. Технические средства микропроцессорных систем : практический курс / Дж. Коффрон ; пер. с англ. – М. : Мир, 1983. – 344 с.

7. Куприянов, М. С. Коммуникационные контроллеры фирмы Motorola / М. С. Куприянов, О. Е. Мартынов, Д. И. Панфилов. – СПб. : БХВ-Петербург, 2001. – 560 с.

8. Левенталь, Л. Введение в микропроцессоры: программное обеспечение, аппаратные средства, программирование / Л. Левенталь ; пер. с англ. – М. : Энергоатомиздат, 1983. – 464 с.

9. Морисита, И. Аппаратные средства микроЭВМ / И. Морисита ; пер. с яп. – М. : Мир, 1988. – 280 с.

10. Рафикузаман, М. Микропроцессоры и машинное проектирование микропроцессорных систем. В 2 кн. Кн. 1 / М. Рафикузаман ; пер. с англ. – М. : Мир, 1988. – 312 с.

11. Ремизевич, Т. В. Микроконтроллеры для встраиваемых приложений: от общих подходов – к семействам HC05 и HC08 фирмы Motorola : справочник / Т. В. Ремизевич. – М. : ДОДЭКА, 2000. – 272 с.

12. Токхайм, Р. Микропроцессоры: курс и упражнения / Р. Токхайм ; под ред. В. Н. Грасевича ; пер. с англ. – М. : Энергоатомиздат, 1987. – 336 с.

13. Фрир, Дж. Построение вычислительных средств на базе перспективных микропроцессоров / Дж. Фрир ; пер. с англ. – М. : Мир, 1990. – 413 с.

14. Шагурин, И. И. Микропроцессоры и микроконтроллеры фирмы Motorola : справ. пособие / И. И. Шагурин. – М. : Радио и связь, 1998. – 560 с.

15. Программно-аппаратные комплексы для проектирования и отладки систем на базе микроконтроллеров Motorola / И. И. Шагурин [и др.] // Chip News. – 1998. – №1. – С. 22–27.

16. Шагурин, И. И. Современные микроконтроллеры и микропроцессоры Motorola : справочник / И. И. Шагурин. – М. : Горячая линия – Телеком, 2004. – 952 с.

Св. план 2020, поз. 3

Учебное издание

ИНТЕЛЛЕКТУАЛЬНЫЕ ЭЛЕКТРОННЫЕ СИСТЕМЫ БЕЗОПАСНОСТИ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

В двух частях

Часть 2

Логин Владимир Михайлович
Ролич Олег Чеславович

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ

ПОСОБИЕ

Редактор *М. А. Зайцева*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *В. М. Задоля*

Подписано в печать 04.04.2020. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 4,3. Уч.-изд. л. 4,4. Тираж 50 экз. Заказ 378.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014, №2/113 от 07.04.2014,
№3/615 от 07.04.2014.
Ул. П. Бровки, 6, 220013, г. Минск