МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ГРАЖДАНСКОЙ АВИАЦИИ

Е.Б. Биктеева,
К.Н. Матюхин

ПРОГРАММИРУЕМЫЕ МИКРОЭЛЕКТРОННЫЕ УСТРОЙСТВА

Учебно-методическое пособие
по выполнению лабораторных работ

для студентов
специальности 25.05.03
всех форм обучения

Москва
2019

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ (МГТУ ГА)»

Кафедра технической эксплуатации радиоэлектронного оборудования воздушного транспорта

Е.Б. Биктеева, К.Н. Матюхин

ПРОГРАММИРУЕМЫЕ МИКРОЭЛЕКТРОННЫЕ УСТРОЙСТВА

Учебно-методическое пособие
по выполнению лабораторных работ

для студентов
специальности 25.05.03
всех форм обучения

Москва
2019

ББК 6Ф.03
Б 60

Рецензент:
Прохоров А.В. – д-р техн. наук, профессор

Биктеева Е.Б.
Б 60 Программируемые микроэлектронные устройства: учебно-методическое пособие по выполнению лабораторных работ./ Е.Б. Биктеева, К.Н. Матюхин. – Воронеж: ООО «МИР», 2019. – 48 с.

Данное учебно-методическое пособие издается в соответствии с рабочей программой учебной дисциплины «Программируемые микроэлектронные устройства» по учебному плану для студентов специальности 25.05.03 всех форм обучения.

В учебно-методическом пособии приведены лабораторные работы практического цикла изучения дисциплины «Программируемые микроэлектронные устройства». Каждая лабораторная работа включает основные теоретические сведения, организационно-методические указания по подготовке и проведению работы, требования к отчету и контрольные вопросы.

Рассмотрено и одобрено на заседании кафедры 12.09.2019 г. и методического совета 12.09.2019 г.

В авторской редакции.

Подписано в печать 21.10.2019 г.
Формат 60x84/16 Печ.л. 3 Усл. печ. л. 2,79
Заказ 553/1633 Тираж 80 экз.

Московский государственный технический университет ГА
125993 Москва, Кронштадтский бульвар, д.20

Отпечатано ООО «МИР»
394033, г. Воронеж, Ленинский пр-т 119А, лит. Я, оф. 215
Тел.: 8 (958) 649-53-31 Email: 89586495331@mail.ru

Содержание

	Стр.
Лабораторная работа №1. Архитектура МК MC68HC12. Знакомство с технологией отладки программы в среде CodeWarrior Development Studio.	4
Лабораторная работа №2. Способы адресации операндов. Команда загрузки пересылки данных	19
Лабораторная работа №3. Арифметические команды. Команды сложения и вычитания.	27
Лабораторная работа №4. Логические команды, команды сдвигов, команды битового процессора.	35
Лабораторная работа №5. Команды условных и безусловных переходов. Работа с массивами.	42

Лабораторная работа №1.

Архитектура МК MC68HC12. Знакомство с технологией отладки программы в среде CodeWarrior Development Studio.

Цель работы - В данной лабораторной работе изучается архитектура МК MC9S12C128 программно логическая модель центрального процессора HCS, распределение адресного пространства МК MC9S12C128.

Также изучается простейший формат записи программы на языке Ассемблера, технология преобразования исходного текста программы в машинные коды для загрузи в МК, технология отладки разработанной программы с использованием программно-логической модели (симулятора) IDE CodeWarrior Development Studio for S12 (X).

Общие сведения о микроконтроллерах

Микроконтроллер (англ. Micro Controller Unit, MCU) — микросхема, предназначенная для управления электронными устройствами. Типичный микроконтроллер сочетает на одном кристалле функции процессора и периферийных устройств, содержит ОЗУ и (или) ПЗУ. По сути, это однокристальный компьютер, способный выполнять простые задачи.

- Использование в современном микроконтроллере достаточного мощного вычислительного устройства с широкими возможностями, построенного на одной микросхеме вместо целого набора, значительно снижает размеры, энергопотребление и стоимость построенных на его базе устройств.

а) б)

Рис. 1. Микроконтроллеры в схематической плате (а), 8 битные микроконтроллеры ATMEGA128 (б)

Микроконтроллер MC9S12C128

В данном лабораторном практикуме Вами будет детально изучаться однокристальный 16-разрядный микроконтроллер семейства HCS12: MC9S12C128.

Основные технические характеристики этого МК:

- 16-разрядное процессорное ядро HCS12.
- Напряжение питания 2.97..5.5 В. В лабораторном макете составляет 5 В.
- Развитая система тактирования. Тактирование микроконтроллера возможно как от внешних источников, так и от внутренних. В лабораторном стенде используется элемент, относящийся к первому типу: кварцевый резонатор. Частота тактового сигнала, который он генерирует, составляет 4 МГц. При этом частота внутренней шины микроконтроллера f_{BUS} уменьшается вдвое и равна 2 МГц. Максимальное её значение для этой модели микроконтроллера составляет 20 МГц;
- Резидентная память программ (ПЗУ). Её объем равен 128K = 131072 ячеек (128 КБ). Память выполнена по технологии flash с эмуляцией EEPROM, число циклов записи/стирания составляет не менее 100000.
- Резидентная память данных (ОЗУ). Объём составляет 4 КБ.
- Общее число выводов микроконтроллера – 40. Тип корпуса – QFP (Quad Flat Package).

На кристалле микроконтроллера имеются следующие периферийные модули:

- Порты ввода/вывода. Всего доступно 9 портов
- Модуль таймера TIM с 16-разрядным счетчиком временной базы и восемью каналами IC/OC/PWM.
- Встроенный аналого-цифровой преобразователь ATD. Число каналов оцифровки равно 8, разрядность – 10 бит.
- Последовательные интерфейсы: синхронный SPI и асинхронный SCI, а также модуль CAN со скоростью до 1 Мбит/с.
- Модуль обработки внешних прерываний INT.
- Сторожевой таймер COP.
- Модуль неразрушающей отладки BDM с однопроводным интерфейсом связи ПК.

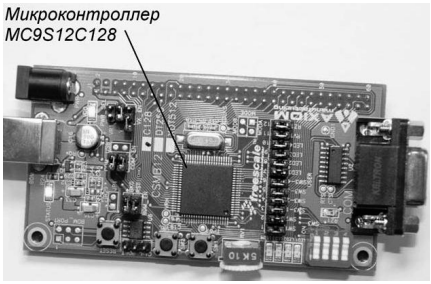

Рис. 2. Микроконтроллер MC9S12C128 на плате CSMB12C128

Описание лабораторной работы

Сначала в разъём платформы ELVIS II вставляется плата, CSMB12C128 при этом подключение к питанию самой платформы не осуществляется. Также дополнительно необходимо соединить специальным кабелем плату CSMB12C128 с разъемом USB вашего ПК. Соединение производится так, как показано на рисунке.

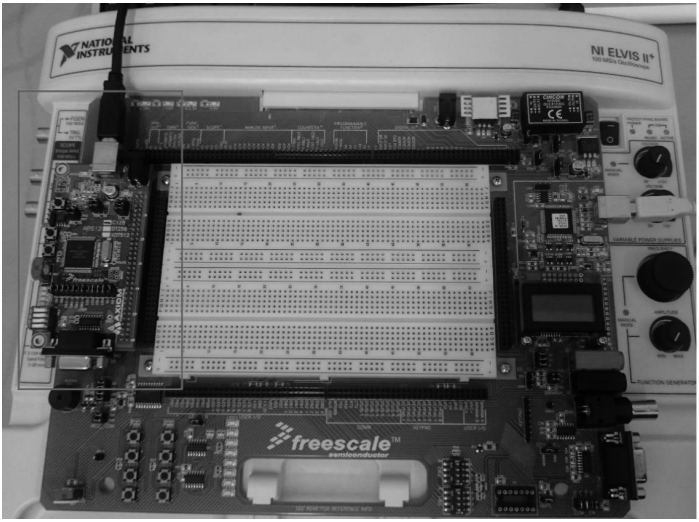

Рис. 3. Плата CSMB12C128 на платформе ELVIS II.

Включаем персональный компьютер, открываем файл “main” типа Assembly Source, расположенный в ПК по адресу: *C:/ ProgramFiles(x86) / Freescale / CWS12v5.1 / bin / Plugins / Support / HC12wizard / DataBase / Sources.*

Выбранный вами файл откроется программой Freescale CodeWarrior.

Далее внутри открывшейся программы закройте окно “main.asm”. Для создания нового проекта необходимо выбрать в диалоговом меню File строку Startup Dialog. Если вы всё сделали правильно, то должно появиться окно, в котором надо будет выбрать “Create New Project” (Рис. 4).

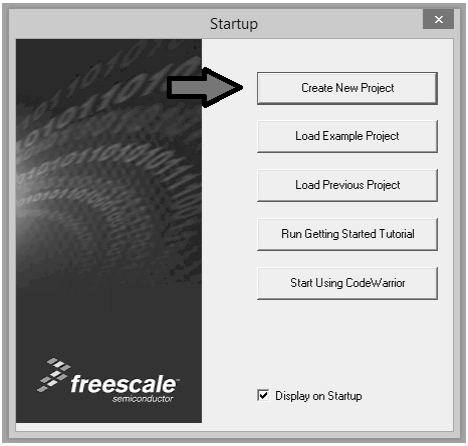

Рис. 4. Окно выбора начального действия.

В следующем окне необходимо выбрать модель МК, для которой вы будете создавать проект. Для этого выбираем HCS12, далее группу HCS12C Family и модель MC9S12C128. Режим программной симуляции нужно выбрать “Full Chip Simulation” (Рис. 5).

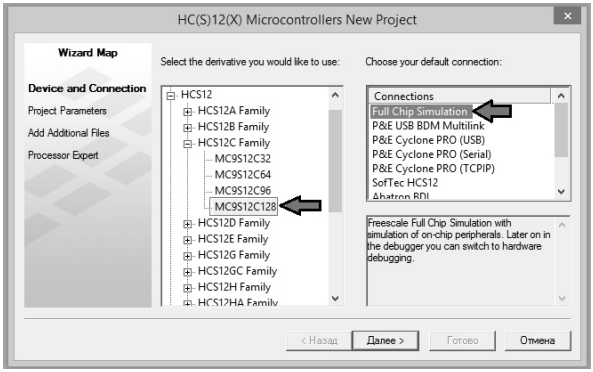

Рис. 5. Первое окно мастера создания проекта.

Нажимаем кнопку “Далее” и выбираем язык программирования МК. Для этого снимаем галочку напротив языка “C” и ставим её напротив языка “Absolute Assembly”. Далее проекту присваивается название и место расположения его файлов, после этого нажимаем кнопку “Готово” (Рис. 6).

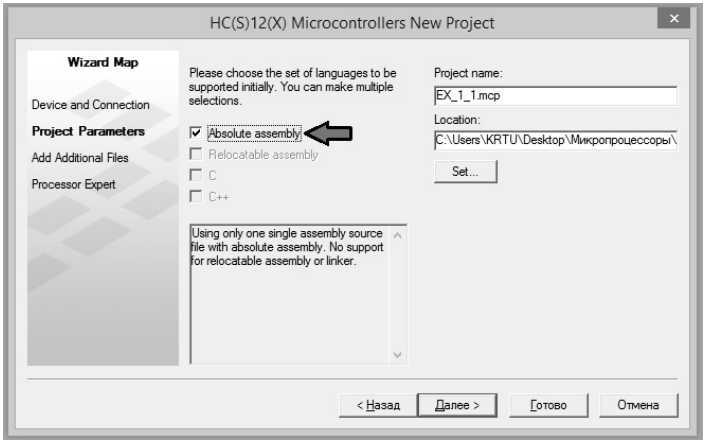

Рис. 6. Второе окно мастера создания проекта.

Таким образом, будет создан файл EX_1_1, и появиться окно с его именем, в нём нужно два раза кликнуть на файл “main.asm” в подкаталоге “Sources” (Рис. 7)

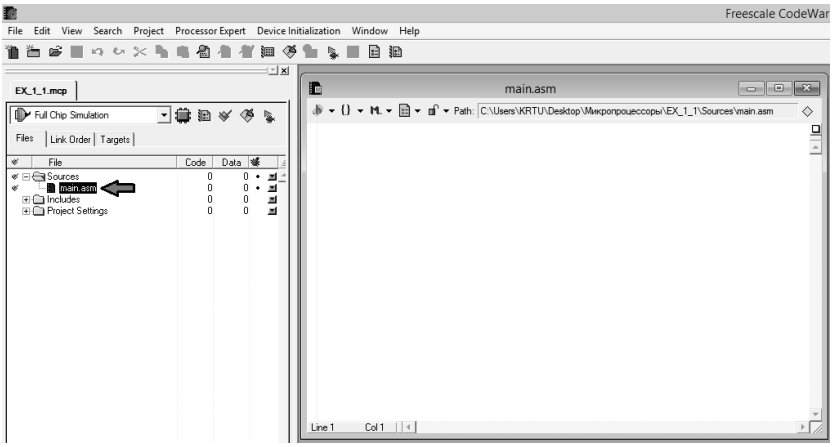

Рис. 7. Основное окно IDE CodeWarrior Development Studio for HC(S)12.

В открывшемся окне файла “main.asm” необходимо удалить все предварительные записи, и далее ввести в него текст своей программы EX_1_1.

Практическая часть.

В ходе выполнения всех последующих лабораторных работ настоящего практикума, посвящённых изучению системы команд и приобретению навыков программирования на языке Ассемблера для МК семейства HCS12 компании Freescale Semiconductor, Вы должны выполнять стандартную последовательность действий:

1. Ввод исходного текста программы на языке Ассемблера с использованием редактора пакета интегрированной среды разработки программного обеспечения (ПО) для микропроцессорной системы WinIDE CodeWarrior;

2. Ассемблирование исходного текста прикладной программы с целью получения машинного кода для загрузки его в память МК;

3. Загрузка машинного кода созданной Вами программой в память МК;

4. Ввод в память МК исходных данных для выполнения программы;

5. Выполнение программы, анализ результатов работы программы.

Описанная последовательность действий именуется процессом разработки отладки прикладной программы управления встраиваемой микропроцессорной системы. На данном занятии необходимо получить практические навыки управления программным пакетом «Интегрированная среда разработки (IDE) CodeWarrior Development Studio for S12(X)» для МК модели MC9S12C128 семейства HC12 на примере простейшей программы EX_1_1.

Программа EX_1_1 производит пересылку содержимого 5 ячеек ПЗУ в ОЗУ МК. Начальный адрес ПЗУ, где расположены данные для пересылки – E100h (или $E100), следовательно, исходный массив *array1* расположенный в ячейках памяти с адресами $E100, $E101, $E102, $E103 и $E104. Адрес области ОЗУ, куда будут производить пересылку – 60h (или $0060). Следовательно, конечный массив *array2* расположится в ячейках памяти с адресами $60, $61, $62, $63 и $64. Исходный текст программы EX_1_1 представлен ниже.

Примечание. Система команд МК семейства HCS12 подробно изучается в следующих лабораторных работах. Данный пример приведён исключительно для демонстрации возможностей симулятора по вводу и исполнению программы, поэтому выбранный способ решения задачи на данном этапе не обсуждается. Необходимые пояснения по способу решения даны в комментариях.

Далее осуществляется ввод программы с пояснениями. Следует иметь ввиду, что текст, расположенный после символа «;» называется комментарием, и при ассемблировании программы не учитывается. Желательно использовать комментарии для пояснения выполняемых Вами смысловых действий прикладной программы.

Пример 1.1.

Команда	Комментарий
ORG $E000	;Команда ORG задаёт начальный адрес Вашей программы в памяти МК. При ассемблировании первая команда будет размещена в ячейке памяти с адресом $E000.
ex_1_1: movb #$08,$11	;Установка начального адреса области регистров специальных функций в $0800. Таким образом, освобождается диапазон адресов $0000..$07FF под ОЗУ.
ldx #$E100	;Индексный регистр *X* загружается начальным адресом первого массива *array1*.
movb #5,$70	;Ячейка памяти ОЗУ с адресом $70 выбрана в качестве счетчика циклов. Предварительно рассчитали, что конечный массив не займет эту ячейку, поэтому ее можно использовать.
m1: ldaa ,x	;Загружаем байт памяти из ячейки первого массива в аккумулятор А.
staa $1F60,x	;Пересылаем данные из аккумулятора А в ;память МК по адресу, вычисленному по ;формуле: (X) + $1F60. $E100 + $1F60 = $10060. Старший разряд 1 уходит за пределы 16-разрядного формата регистра адреса, поэтому автоматически отбрасывается, полученное значение адреса равно $0060.
inx	;Увеличиваем содержимое индексного регистра X на 1.
dec $70	;Уменьшаем счётчик циклов на 1.
bne m1	;Проверяем счётчик циклов на 0. Если не 0, то продолжить исполнение программы, перейдя по метке m1. Иначе переход на следующую команду.

jmp *	;Если все байты массива перемещены, то исполнять пустой цикл (бесконечный переход по адресу текущей команды). Такое решение необходимо, чтобы при отладке в автоматическом режиме исключить исполнение произвольного кода из памяти МК, который Вы не загружали.
ORG $FFFE DC.W $E000	;Задать адрес вектора начальной загрузки. При включении МК этот адрес ($E000) автоматически загрузится в счётчик PC центрального процессора. Конец текста ассемблирования.

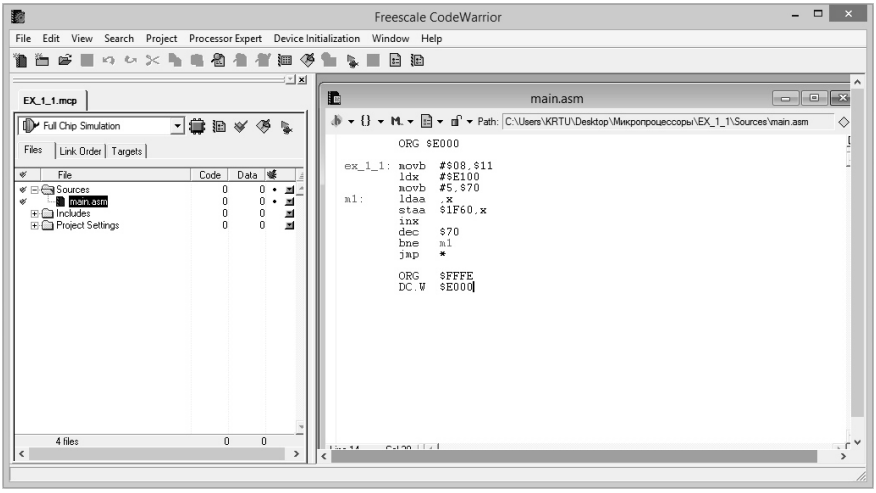

Рис. 8. Основное окно IDE CodeWarrior Development Studio for HC(S)12

В результате должен получиться листинг, изображенный на рис. 8.

После ввода программы нажмите кнопку (make) ассемблирования исходного текста программы, которое находится в окне main.asm. Ассемблирование – это процесс получения машинного кода программы из исходного текста. Если синтаксис во введённой вами программе правильный, то ассемблирование пройдёт успешно, о чём появится запись на экране.

Обратите внимание, что это не означает, что ваша программа правильно реализует задуманный вами алгоритм. Эта запись означает лишь, что отсутствуют ошибки при вводе текста программы. Если в исходном тексте присутствуют ошибки, то в верхней части экрана появится окно с указанием типа ошибок. Иногда желательно ввести ошибки в исходный текст, например, заменив команду «mov» на «miv», а затем исправив метку ex_1_1 на ex_1_20. Тогда можно наблюдать процесс диагностирования ошибок средой разработки.

В случае успешного ассемблирования, следующим шагом будет проверка правильности исполнения вашей программой задуманного алгоритма. Для этого воспользуется программа симулятора, которая без подключения МК средствами только персонального компьютера промоделирует исполнения программы микроконтроллером.

Для того чтобы созданная вами программа была исполнена, код этой программы необходимо загрузить в память МК. Поскольку МК в нашем случае виртуальный (настоящей ИС МК нет, ПК изображает преобразование информации в МК), то загрузка выполняется автоматически в процессе запуска программы симулятора.

После того, как вы добьётесь успешного ассемблирования вашего программного фрагмента, нужно нажать кнопку  (debug). Таким образом, код загрузится в симулятор. Для того чтобы исполнение программы началось с указанного ранее в ней адреса, необходимо произвести сброс МК. Делается это с помощью перехода в главном меню отладчика “HCS12 FCS” → “Reset”. В результате окно должно выглядеть как показано на рис. 9.

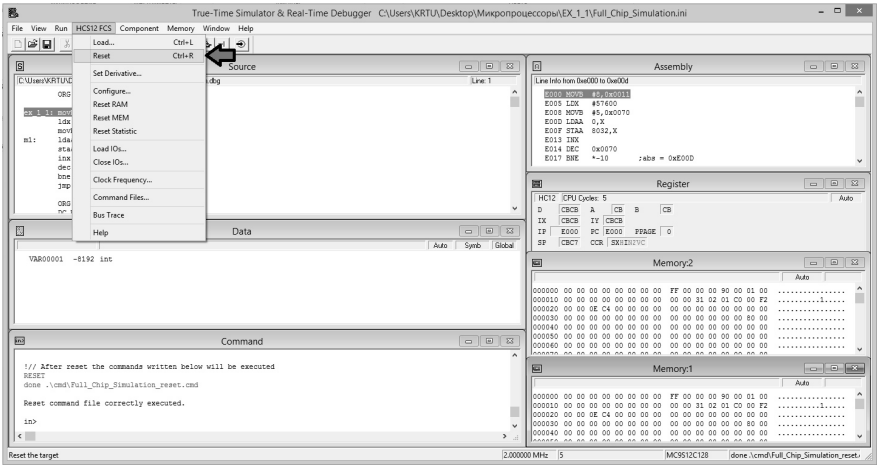

Рис. 9 Окно отладчика IDE CodeWarrior Development Studio for HC(S)12

Каждое малое окно внутри основного регулируется по размерам и местоположению, поэтому вы можете создать собственную конфигурацию интерфейса отладчика. Кроме того, вами могут быть открыты дополнительные окна. В начальной версии интерфейса будет присутствовать только одно окно памяти ***Memory.*** Для отладки примера данной лабораторной работы и многих последующих удобно иметь два окна памяти. Поэтому в пункте меню "C*omponent*" найдите компонент ***Memory*** и откройте его. На экране отладчика появится второе окно ***Memory.*** Сохраните эту конфигурацию отладчика.

В окне ***Source*** отображается исходный текст вашей программы. Первая строка отображается синим цветом и стрелкой. Таким образом, отладчик указывает строку программы, которая соответствует текущему состоянию программы счётчика PC центрального процессора. Убедитесь в этом, взглянув на окно ***Register.*** В этом окне отображается состояние всех регистров центрального процессора.

Для наблюдения за ходом исполнения программы вам потребуется два окна памяти ***Memory1*** и ***Memory2.*** В окне ***Memory1*** разместите свой исходный массив. Для этого установите начальный адрес зоны просмотра в окне, равный $E100. Используйте для этого контекстное меню, которое вызывается нажатием правой кнопки мыши на зону столбца адреса в окне ***Memory1.*** Для этого необходимо в окне "Display address" снять галочку «hex format» и ввести "$E100" (начальный адрес зоны просмотра), как изображено на рис. 10.

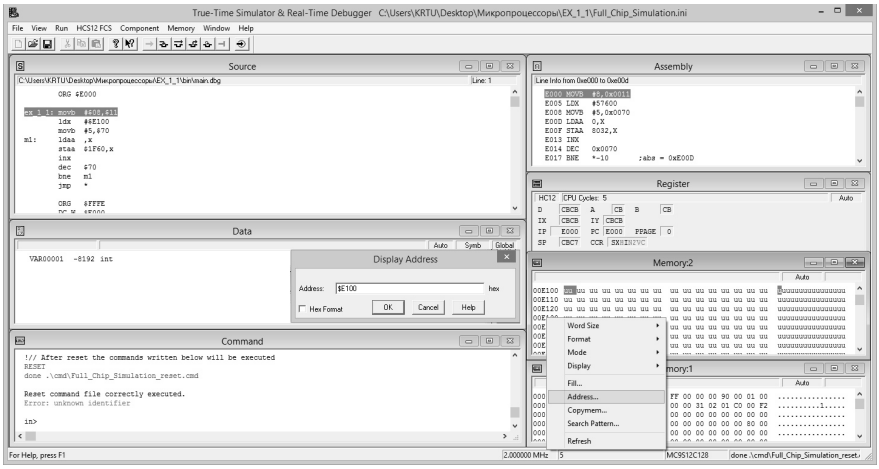

Рис. 10 Окно отладчика IDE CodeWarrior Development Studio for HC(S)12

Далее в окне ***Memory1*** заполните первые 5 ячеек памяти произвольными кодами, например: 31, 32, 33, 34, 35 (можно другими).

В окне ***Memory2*** вы будете наблюдать формирование конечного массива.

Сделайте активным окно ***Memory2,*** нажав на него курсором. Установите начальный адрес зоны просмотра в окне, равный $040. Вы увидите, что часть ячеек памяти помечена символом

“--”. Это означает, что физически ячейки памяти с указанным адресом в МК не существует. Другие ячейки памяти помечены символом “uu”. Это означает, что ячейка в МК присутствует, но она не инициализирована, то есть при включении МК в неё не записывалась информация, либо под управлением программы, либо под управлением оператора, то есть вами. При попытке обращения таким ячейкам памяти из отлаживаемой программы, отладчик будет автоматически останавливаться, и сообщать об этом звуковым сигналом. Вас это не должно волновать, поскольку по ходу исполнения программы в эти ячейки памяти будет записываться информация. Если информация читалась из этих ячеек, то необходимо было бы принять действия по их инициализации, то есть занесению данных в эти ячейки памяти.

Далее следует приступить к исполнению программы. Для этого воспользуемся режимом запуска по шагам - кнопка  (Step over “F10”). Нажмите на эту кнопку один раз (одно нажатие является одним шагом).

Наблюдайте перемещение курсора в окне исходного текста программы и изменения регистра IX центрального процессора. После исполнения второго шага наблюдайте в окне ***Memory2*** изменение содержимого ячейки памяти с адресом $70. Исполняйте далее программу по шагам. Сначала в аккумулятор A (окно ***Register***) загрузится код $31 из ячейки памяти с адресом $E100, а затем этот код будет записан в ОЗУ по адресу $060. Далее содержимого индексного регистра центрального процессора HX увеличится на 1, а код в ячейке памяти счётчика циклов $50 уменьшится на 1. Команда условного перехода “bne m1” определит, что не все пять циклов программы выполнены и осуществит изменение счётчика адреса PC таким образом, что он снова выполнил команду с меткой “m1”. Выполните по шагам все пять циклов программы, наблюдая заполнение ячеек памяти ОЗУ в окне ***Memory2.***

Далее следует повторить исполнение вашей программы в другом режиме.

Для этого сначала запишем в ячейки окна ***Memory2*** нулевые значения, тогда можно будет снова наблюдать заполнение ОЗУ в ходе выполнения программы. Установите счётчик адреса PC в начальное состояние $E000. Для ускорения процесса отладки следует установить контрольную точку в строке условного перехода “bne m1”. Для этого отметим её курсором, затем воспользуемся быстрым меню, нажав на правую кнопку мыши. Выберем опцию “Set breakpoint”. В строке установится красный маркер. Возвратим программу к начальным условиям, нажав на кнопку  (Reset target). Запустим программу, но уже в автоматическом режиме, для этого следует нажать кнопку  . Программа исполнит несколько команд и остановится перед исполнением оператора в строке контрольной точки. Убедитесь в этом в заполнении всего лишь одной ячейки ОЗУ (в окне ***Memory2*** по адресу $060). Далее выполните один шаг программы, вследствие чего осуществиться переход программы к оператору по метке “m1”. Наблюдайте далее исполнение одного цикла программы, при каждом запуске в автоматическом режиме “GO”. Обратите внимание, каждая остановка по контрольной точке сопровождается соответствующей информации в окне ***Command.***

Далее следует запустим программу в автоматическом режиме, удалив контрольную точку (для этого в выпадающем меню следует нажать “Delete breakpoint”), очистим ОЗУ, установим PC в начальное значение (для этого нажмём кнопку кнопку  (Reset target)). Нажмите кнопку автоматического запуска  . На некоторое время поле ***Memory*** станет просто белым –

программа исполняется, но результат исполнения не отображается. Затем произойдёт автоматическая остановка программы, на последнем операторе с индикацией в окне ***Command*** причиной остановки. Исполнение программы прервал сторожевой таймер. По умолчанию в данном МК он включен, но время его срабатывания достаточно большое, поэтому он не будет препятствовать исполнению тестовых примеров. Обратите внимание, ваша программа выполнила все необходимые действия и “зациклилась” на операторе “jmp *”.

Именно этот эффект предусматривался при написании исходного текста программы. Проведите эксперимент: закройте программу отладчика, исключите строку “jmp *” из текста исходной программы и повторите отладку. Наблюдайте отличия исполнения и отображения конечного состояния на экране отладчика. Полученный эффект должен убедить Вас в необходимости постановки строки “jmp *” в тестовых программных примерах.

Задания для самостоятельной работы

Измените текст исходной программы таким образом, чтобы коды программы были записаны в ПЗУ, начиная с адреса $E200, оба массива располагались в ОЗУ МК, а число элементов массива составляло бы 10. Произведите отладку полученной программы.

Ещё раз внимательно ознакомьтесь с режимами работы отладчика/симулятора. Опробуйте интересующие вас режимы при отладке этого примера.

Контрольные вопросы

1. Основные технические характеристики микроконтроллера MC9S12C128?
2. Определение микроконтроллера?
3. Продемонстрируйте технологию загрузки в симулятор отлаживаемой программы?
4. Продемонстрируйте способ занесения кодов в назначенную область памяти?
5. Продемонстрируйте способы изменения содержимого регистров центрального процессора под управлением оператора?
6. Продемонстрируйте пример отладки с установкой контрольной точки?
7. Продемонстрируйте технологию загрузки в симулятор отлаживаемой программы.

8. Продемонстрируйте способ занесения кодов в назначенную область памяти.

9. Продемонстрируйте способы изменения содержимого регистров центрального процессора под управлением оператора.

10. Продемонстрируйте пример отладки с установкой контрольной точки.

Лабораторная работа №2.

Способы адресации операндов. Команда загрузки пересылки данных

Цель работы - В данной лабораторной работе детально изучаются способы адресации операндов, используемы центральным процессором HCS12. Рассматривается группа команд загрузки и пересылки данных, приобретаются навыки использования этих команд для составления типовых фрагментов программ на языке Ассемблера.

Описание лабораторной работы

Для успешного выполнения данной работы и всех последующих работ необходимо освоить материал первого занятия в активном режиме, то есть уметь самостоятельно, без получения подсказки о каждом следующем действии или нажатой клавише, производите ввод и отладку тестируемого примера с использованием отладчика/симулятора.

В процессе подготовке к лабораторно работе №2 рекомендуется ознакомиться с группой команд пересылки и загрузки.

Практическая часть

Пример 2.1. Исходные данные находятся в ОЗУ по адресам $90…$94. Произвольные значения чисел должны быть занесены в эти ячейки под управлением программы. Затем эти числа должны быть перемещены в другую область ОЗУ по адресам $A0…$A4.

Для начала решим задачу самыми простым методом используя непосредственную и прямую адресацию.

Команда	Комментарий
RamStart EQU $0000	;Назначается адрес для первой ячейки ОЗУ.
RomStart EQU $E000	;Назначается начальный адрес прикладной программы.
StartVector EQU $FFFE	;Назначается адрес для вектора начального запуска.
INITRG EQU $0011	;Назначается адрес регистра, отвечающего за начальный адрес области регистров специальных функций. ;С помощью команд EQU именам присваиваются значения. По сути это всё-таки

	псевдокоманды, т.к. они не выполняют реальных действий в программе.
ORG RomStart	
EX_2_1_1: LDAA #$08 STAA INITRG	;Установка начального адреса области регистров ;специальных функций в $0800.
LDAA #$40	;Загрузить в аккумулятор А число $40. Использована ;непосредственная адресация.
STAA $90 LDAA #$29 STAA $91 LDAA #29 STAA $90 LDAA#%01010011 STAA $93 LDAA #$AC	;Переместить число из А в ячейку памяти с адресом $90. Использована прямая адресация.
STAA $94	;Загрузка ячеек ОЗУ начальными значениями закончена. Приступим к перемещению данных из одной области памяти в другую.
LDAB $90 STAB $A0 LDAB $91 STAB $A1 LDAB $92 STAB $A2 LDAB $93 STAB $A3 LDAB $94 STAB $A4	;Загрузить данные из ячейки $90 в аккумулятор В. Загрузить ;данные из В в ячейку памяти с адресом $A0.
JMP *	;Бесконечный переход по адресу текущей команды.
ORG StartVector DC.W EX_2_1_1	

У вас должно получиться тоже, что на рис. 1

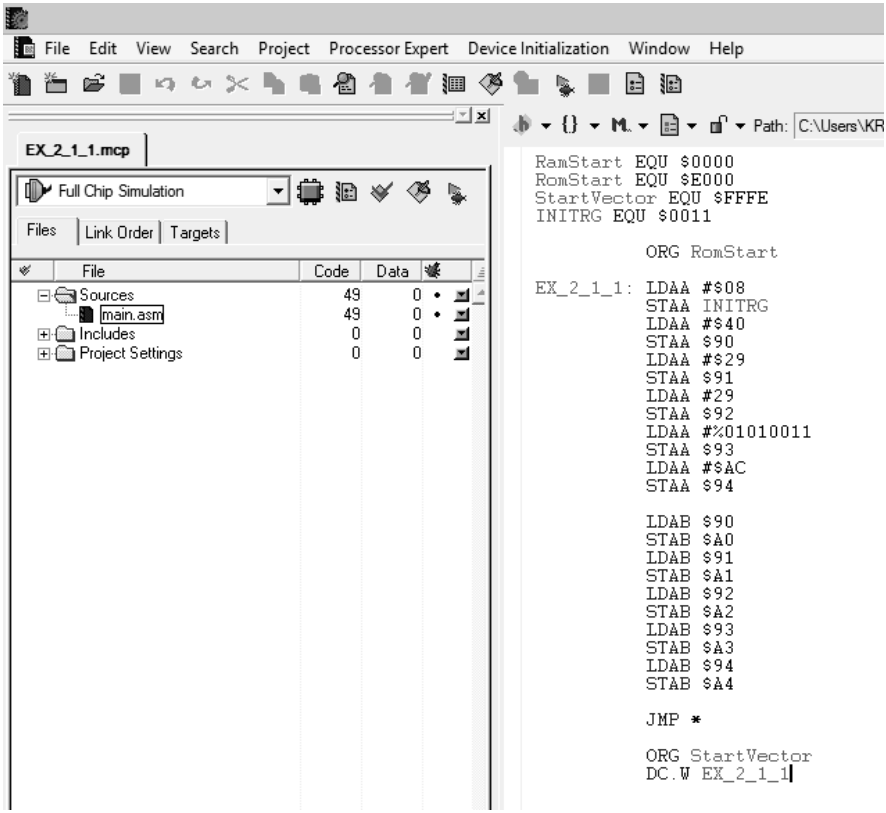

Рис. 1.

После ввода программы нажмите кнопку  (make) ассемблирования исходного текста программы, которое находится в окне main.asm. Ассемблирование – это процесс получения машинного кода программы из исходного текста. Если синтаксис во введённой вами программе правильный, то ассемблирование пройдёт успешно, о чём появится запись на экране. Обратите внимание, что это не означает, что ваша программа правильно реализует задуманный вами алгоритм. Эта запись означает лишь, что отсутствуют ошибки при вводе текста программы. Если в исходном тексте присутствуют ошибки, то в верхней части экрана появится окно с указанием типа ошибок. Затем нужно нажать кнопку  (debug) и проверить работоспособность программы. Таким образом, код загрузится в симулятор. Для того чтобы исполнение программы началось с указанного ранее в ней адреса, необходимо произвести сброс МК. Делается это с помощью перехода в

главном меню отладчика “HCS12 FCS” → “Reset”. После запускаем программу в автоматическом режиме для этого следует нажать кнопку , спустя некоторое время стоит остановить кнопкой выполнение и убедиться в получении верного результата, в окне ***Memory*** будет отображено то же что на рис.2

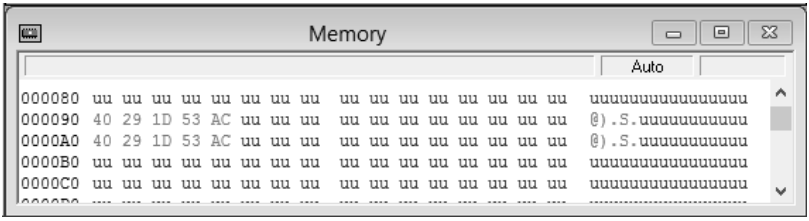

Рис.2.

Пример 2.1.2 Данный пример демонстрирует использование достаточно большого числа команд. Приведем другой способ решения данной задачи, используя другие способы адресации и команды.

Команда	**Комментарий**
RamStart EQU $0000	;Назначается адрес для первой ячейки ОЗУ.
RomStart EQU $E000	;Назначается начальный адрес прикладной программы.
StartVector EQU $FFFE	;Назначается адрес для вектора начального запуска.
INITRG EQU $0011	;Назначается адрес регистра, отвечающего за начальный адрес области регистров специальных функций.
ORG RomStart	
EX_2_1_2: MOVB #$08, INITRG	;Установка начального адреса области регистров специальных функций в $0800.
LDD #$4028	;Загрузить в двухбайтовый аккумулятор D число $40 и $28. Первое помещается в

	A, второе в B.
STD $90	;Записать содержимое аккумулятора D в две ячейки с адресами $90 и $91. Число из A пойдёт в первую ячейку, число из B – во вторую.
LDD #2BDE STD $92 LDAA #125 STAA $94	

MOVB $90, $A0
 	
MOVB $91, $A1
MOVB $92, $A2
MOVB $93, $A3
MOVB $94, $A4
	;Используя команду MOVB с прямой адресацией переместим данные из одной области памяти в другую.
	Перенести данные из ячейки с адресом $90 в ячейку с адресом $A0.

JMP *	;Бесконечный пустой цикл.
ORG StartVector DC.W EX_2_1_2	

Выполняем аналогичные действия что и для первого примера.

Если размер массива довольно велик, то целесообразно организовать циклическое выполнение копирования элементов. О том, как это сделать, показано в примере ниже,

Пример 2.2. Исходные данные находятся в ОЗУ по адресам $0110…$0114. Произвольные значения чисел должны быть занесены в эти ячейки под управлением программы. Затем эти числа должны быть перемещены в другую область ОЗУ по адресам $0150…$0154.

По сравнению с предыдущим примером, адреса массивов теперь находятся за пределами первой страницы памяти ($0000..$00FF). Поэтому нельзя использовать команды с прямой адресацией (когда адрес составляет один байт), а только индексную или прямую расширенную.

Команда	Комментарий
RamStart EQU $0060 RomStart EQU $E000 StartVector EQU $FFFE INITRG EQU $0011	
ORG RomStart	
EX_2_2_1: MOVB #$08, INITRG	;Установка начального адреса области регистров ;специальных функций в $0800.
LDD #$4028	;Загрузить в двухбайтовый аккумулятор D число $40 и $28. Первое помещается в A, второе в B.
STD $0110	;Записать содержимое аккумулятора D в две ячейки с адресами $0110 и $0111. Число из A пойдёт в первую ячейку, число из B – во вторую.
LDD #0604 STD $0112 LDAA #%0110 STAA $0114	;Загрузка в аккумулятор A числа 6.
MOVB #5, $50 LDX #$0110	;Ячейка с адресом $50 выбрана в качестве счетчика циклов. Начальное значение – 5. Загрузка в индексный регистр числа X адреса первого элемента исходного массива.
TRFR: LDAA ,X	;Загрузка байта данных в аккумулятор A из ячейки памяти, на которую указывает регистр X.
STAA $20, X	;Сохранение содержимого аккумулятора A в ячейке памяти с ;адресом (X)+$20. Использована индексная адресация со смещением в 9 бит.
INX	;Увеличение значения регистра X на 1.
DEC $50	;Уменьшение счетчика циклов на 1.
BNE TRFR	;Если счетчик циклов не равен 0, то перейти по метке TRFR ;для перемещения следующего элемента.

	Иначе закончить выполнение программы.
JMP *	
ORG StartVector DC.W EX_2_2_1	

Пример 2.3. Исходные данные находятся в ОЗУ по адресам $0110…$0114. Произвольные значения чисел должны быть занесены в эти ячейки под управлением программы. Затем эти числа должны быть перемещены в обратном порядке в другую область ОЗУ по адресам $0130…$0134. Т.е. из ячейки с адресом число $0110 должно попасть в $0134, из $0111 в $0133 и т.д.

Команда	Комментарий
RamStart EQU $0060 RomStart EQU $E000 StartVector EQU $FFFE INITRG EQU $0011	
ORG RomStart	
EX_2_3_1: MOVB #$08, INITRG MOVW #$5A21, $0110	;Загрузить в ячейки памяти с адресами $0110 и $0111 два ;байта данных $5A и $21 соответственно.
MOVW #$0406, $0112 MOVB #$FE, $0114	
LDY #$05	;Индексный регистр Yбудет выполнять роль счетчика циклов, а также участвовать в формировании адреса ячейки ;нового массива.
LDX #$00	;Очищаем индексный регистр.
LDAA $0110, X	;Загрузка в аккумулятор А значения ячейки памяти с адресом $0110+(X). Использован индексный тип адресации со смещением в 9 бит.
STAA $014F, Y	;Сохранение данных аккумулятора А в ячейку с адресом $014F +(Y)/
INX	;Увеличение значения регистра X на 1.
DEY	;Уменьшение значения регистра Y на 1.

BNE TRFR	;Переход по метке TRFR, если (Y)≠0.
JMP *	
ORG StartVector DC.W EX_2_3_1	

Задания для самостоятельной работы

1. Изменить порядок следования байт в регистре D.
2. Занести данные $1F и $AE в аккумулятор A и индексный регистр X соответственно. Перенести значение этих регистров в индексный регистр X таким образом, чтобы в регистре оказалось значение $1FAE.
3. Заполнить 10 ячеек стека значением памяти, начиная с адреса $0100. Начальный адрес указателя стека $03FF.
4. Написать программу, заполняющую ячейки памяти по адресам $0100..$010F от 1 до 16. Реализация решения должна происходить в цикле.
5. Написать программу, перемещающую по 2 байта данных. Исходные данные находятся по адресам $0100..$010F. Эти числа должны быть перемещены по адресам $0130..$014F.
6. Изменить порядок следования элементов массива, располагающегося в области памяти по адресам $0110..$011B. То есть данные ячейки $0110 должны оказаться в ячейке $011B, $0111 - $011A и т.д.

Контрольные вопросы

1. Какие важнейшие функции выполняют команды пересылки данных?
2. Определение ОЗУ?
3. Достоинства и недостатки статических и динамических ОЗУ?
4. На какие варианты подразделяются ОЗУ?
5. Для чего используется адресация в микропроцессорах?
6. Назовите основные виды адресации?
7. Для чего используется адресация в микропроцессорах?

Лабораторная работа №3.

Арифметические команды. Команды сложения и вычитания.

Цель работы - В данной лабораторной работе изучаются арифметические команды центрального процессора HCS12. Рассматриваются примеры выполнения арифметических действий над кодами, приобретаются навыки использования команд арифметических действий для составления из них типовых фрагментов программ на языке Ассемблера

Описание лабораторной работы

Для успешного выполнения данной работы рекомендуется изучить программно-логическую модель центрального процессора HCS12, способы адресации операндов, используемые этим процессором, а также получить вводные сведения о программировании на языке Ассемблера.

Практическая часть

В процессе подготовки к лабораторной работе №3 рекомендуется повторить команды загрузки и пересылки данных, а также способы адресации. Подробно ознакомиться с группой арифметических команд. На основе полученных теоретических сведений самостоятельно отладить примеры, которые рассматриваются ниже.

Пример 3.1. По адресам $60, $61, $62 хранятся три числа. Необходимо сложить первое со вторым и из полученной суммы вычесть третье число. После этих действий результат уменьшить на 2 и результат поместить в память по адресу $70. Числа представлены в прямом коде.

Для начала решим задачу самыми простым методом используя непосредственную и прямую адресацию.

Команда	Комментарий
RamStart EQU $0000	;Назначается адрес для первой ячейки ОЗУ.
RomStart EQU $E000	;Назначается начальный адрес прикладной программы.
StartVector EQU $FFFE	;Назначается адрес для вектора начального запуска.

INITRG EQU $0011	;Назначается адрес регистра, отвечающего за начальный адрес области регистров специальных функций. ;С помощью команд EQU именам присваиваются значения. По сути это всё-таки псевдокоманды, т.к. они не выполняют реальных действий в программе.
ORG RomStart	
EX_3_1: MOVB #$08, INITRG	
LDAA $60	;Загрузить в аккумулятор А первое число.
ADDA $61	;Прибавить к значению аккумулятора А второе число. ;Используется прямая адресация.
SUBA $62	;Вычитание из полученной суммы значения ячейки памяти с адресом $62
SUBA #2	;Уменьшение результата на 2. Непосредственная адресация.
STAA $70	;Поместить результат по адресу $70.
JMP *	;Бесконечный переход по адресу текущей команды.
ORG StartVector DC.W EX_3_1	

Пример 3.2. Сложить два 16-разрядных числа без проверки разряда переполнения в старшем байте. Данные хранятся в ячейках памяти ОЗУ МК с адресами $60, $61 и $62, $63. Результат сложения должен быть помещен в память по адресам $70 и $71. Числа представлены в прямом коде.

Сначала рассмотрим вариант решения, в котором используются арифметические команды, оперирующие только 8-разрядными числами.

Команда	Комментарий
RamStart EQU $0000	;Назначается адрес для первой ячейки ОЗУ.
RomStart EQU $E000	;Назначается начальный адрес прикладной программы.
StartVector EQU $FFFE	;Назначается адрес для вектора начального запуска.
INITRG EQU $0011	;Назначается адрес регистра, отвечающего за начальный адрес области регистров специальных функций. ;С помощью команд EQU именам присваиваются значения. По сути это всё-таки псевдокоманды, т.к. они не выполняют реальных действий в программе.
ORG RomStart	
EX_3_2_1: MOVB #$08, INITRG	
LDAB $61	;Загрузить в аккумулятор B первое число.
ADDB $63	;Прибавить к значению аккумулятора младший байт второго числа. Разряд переноса запоминается флагом C в регистре кода состояния.
STAB $71	;Поместить младший байт результата по адресу $71. Значение флага C при этом не изменяется.
LDAB $60	;Загрузка в аккумулятор B старшего байта первого числа.
ADCB $62	;Прибавление к (B) старшего байта второго числа и разряда переноса от предыдущего сложения. Новое значение разряда переноса запоминается в флаге C, но оно нам уже не важно.
STAB $70	;Запоминание старшего байта результата в ячейке с адресом ;$70
JMP *	;Бесконечный переход по адресу текущей

	команды.
ORG StartVector DC.W EX_3_2_1	

Изменим программу, вводя в нее арифметические команды, оперирующие с 16-разрядными числами.

Команда	**Комментарий**
RamStart EQU $0000	;Назначается адрес для первой ячейки ОЗУ.
RomStart EQU $E000	;Назначается начальный адрес прикладной программы.
StartVector EQU $FFFE	;Назначается адрес для вектора начального запуска.
INITRG EQU $0011	;Назначается адрес регистра, отвечающего за начальный адрес области регистров специальных функций. ;С помощью команд EQU именам присваиваются значения. По сути это всё-таки псевдокоманды, т.к. они не выполняют реальных действий в программе.
ORG RomStart	
EX_3_2_2: MOVB #$08, INITRG	
LDD $60	;Загрузить в аккумулятор D первое число. Значение ячейки с адресом $60 помещается в аккумулятор A, значение ячейки с ;адресом $61 – в B.
ADDD $62	;Прибавить к значению аккумулятора D второе число.
STD $70	;Запоминание результата в ячейках с адресами $70 и $71.
JMP *	;Бесконечный переход по адресу текущей команды.
ORG StartVector DC.W EX_3_2_2	

Пример 3.3. Сложить два 32-разрядных числа без проверки разряда переполнения в старшем байье. Данные хранятся в ячейках памяти ОЗУ МК с адресами $60, $61, $62, $63 и $70, $71, $72, $73 соответственно. Результат сложения должен быть помещён в память по адресам $80, $81, $82, $83.
Числа представлены в прямом коде.

В варианте решения, приведенном ниже, включены пользовательские символьные константы.

Команда	Комментарий
RamStart EQU $0000	;Назначается адрес для первой ячейки ОЗУ.
RomStart EQU $E000	;Назначается начальный адрес прикладной программы.
StartVector EQU $FFFE	;Назначается адрес для вектора начального запуска.
INITRG EQU $11	;Назначается адрес регистра, отвечающего за начальный адрес области регистров специальных функций. С помощью команд EQU именам присваиваются значения. По сути это всё-таки псевдокоманды, т.к. они не выполняют реальных действий в программе.
NUM1 EQU $60	;Адрес старшего байта первого числа.
NUM2 EQU $70	;Адрес старшего байта второго числа.
RES EQU $80	;Адрес старшего байта результата.
ORG RomStart	
EX_3_3: MOVB #$08, INITRG	
LDD NUM1+2	;Загрузить в аккумулятор D два младших байта первого числа.
ADDD NUM2+2	;Прибавить к значению аккумулятора D два младших байта второго числа. Разряд переноса запоминается флагом C.
STD RES+2	;Сохрааание результата в ячейках с адресами $82 и $83. Значение флага C остается неизменным.

LDD NUM1	;Загрузка в аккумулятор D двух старших байтов первого числа. C=const.
ADCB #$00	;Прибавление к значению аккумулятора B числа 0 с учетом переноса. Фактически прибавится только значение флага C, оставшееся от предыдущего сложения.
ADDD NUM2	;Добавление в аккумулятор D двух старших байтов второго ;числа.
STD RES	;Сохранение результата в ячейки памяти $80 и $81.
JMP *	;Бесконечный переход по адресу текущей команды.
ORG StartVector DC.W EX_3_3	

Пример 3.4. Вычесть два 16-разрядных числа без проверки разряда переполнения в старшем байте. Данные хранятся в ячейках памяти ОЗУ МК с адресами $60, $61 и $62, $63. Результат вычитания должен быть помещен в память по адресам $70 и $71. Числа представлены в прямом коде.

Команда	**Комментарий**
RamStart EQU $0000	;Назначается адрес для первой ячейки ОЗУ.
RomStart EQU $E000	;Назначается начальный адрес прикладной программы.
StartVector EQU $FFFE	;Назначается адрес для вектора начального запуска.
INITRG EQU $0011	;Назначается адрес регистра, отвечающего за начальный адрес области регистров специальных функций. С помощью команд EQU именам присваиваются значения. По сути это всё-таки псевдокоманды, т.к. они не выполняют реальных действий в программе.
ORG RomStart	

EX_3_4: MOVB #$08, INITRG	
LDD $60	;Загрузить в аккумулятор D первое число. Значение ячейки с адресом $60 помещается в аккумулятор A, значение ячейки с адресом $61 - в B.
SUBD $62	;Вычесть из значения аккумулятора D второе число.
STD $70	;Запоминание результата в ячейках с адресами $70 и $71.
JMP *	;Бесконечный переход по адресу текущей команды.
ORG StartVector DC.W EX_3_4	

Анализ состояния бита переноса/заёма после выполнения последней операции может понадобиться для того , чтобы понять, в каком коде представлен результат: в прямом или дополнительном. В процессе отладки выполните несколько численных примеров. Сначала сделайте значение первого числа больше второго. Затем сделайте второе число больше первого и выполните программу вычитания двух чисел. Что вы видите в конечных ячейках памяти? Объясните полученный результат, применив знания из теории дополнительных кодов.

Остальные команды данной группы Вы изучите, ответив на контрольные вопросы и выполнив задания для самостоятельной работы.

Задания для самостоятельной работы

1. Написать программу вычитания содержимого индексного регистра X из двух ячеек памяти ОЗУ с адресами $0100, $0101.
2. Вычислить произведение регистров A и B. Результат поместите в две ячейки памяти с адресами $0120, $0121.
3. Написать программу суммирования содержимого индексного регистра X с 16-битным числом, находящимся в двух ячейках памяти по адресам $0110, $0111. Результат представить в трёх ячейках памяти с адресами $0123, $0124, $0124.
4. Написать программу деления содержимого аккумулятора D на значение на значение индексного регистра X. Частное от деления представить в двух ячейках памяти с адресами $0100, $0101. Остаток от деления в ячейке $0102.

5. Написать программу сложения двух 64-разрядных. Первое число находится в области памяти $0100…$0107, а второе в $0110…$0117. Результат разместить по адресам $0120…$0127.

6. Написать программу суммирования 16-разрядных чисел, представленных в двоично-десятичном формате. Результат разместить в трёх ячейках памяти.

7. Найти произведение индексного регистра X и аккумулятора A. Результат поместить в три ячейки памяти.

8. Написать программу деления 24-разрядного числа на 8-разрядное число. Частное поместить в две ячейки памяти.

9. Написать программу преобразования содержимого аккумулятора B в двоичный вид. Результат представить в 8 ячейках памяти.

10. Написать программу преобразования содержимого аккумулятора A в двоично-десятичное представление этого числа. Результат представить в трёх ячейках памяти. Перечислите основные арифметические операции, осуществляемые в микроконтроллере?

Контрольные вопросы

1. Что такое регистры общего назначения?
2. Что такое стек?
3. Назначение указателя стека?
4. Как осуществляется запись данных в стек?
5. Как осуществляется извлечение данных из стека?

Лабораторная работа №4.

Логические команды, команды сдвигов, команды битового процессора.

Цель работы - В данной лабораторной работе детально изучаются логические команды центрального процессора. Их дополняют шесть операций сдвигов. Рассматриваются примеры выполнения логических действий над кодами, приобретаются навыки использования логических команд и команд сдвигов для составления типовых фрагментов программ на языке Ассемблера.

Описание лабораторной работы

В процессе подготовки к лабораторной работе №4 рекомендуется повторить предыдущий материал. Также желательно подробно ознакомиться с группой логических команд, операций сдвигов, команд битового процессора.
На основе полученных теоретических сведений самостоятельно отладить примеры, которые рассматриваются ниже в практической части

Практическая часть

Пример 4.1. Провести операцию «логическое И» над содержимым регистров X и У. Результат поместить в ОЗУ МК.

Команда	Комментарий
RamStart EQU $0000	;Назначается адрес для первой ячейки ОЗУ.
RomStart EQU $E000	;Назначается начальный адрес прикладной программы.
StartVector EQU $FFFE	;Назначается адрес для вектора начального запуска.
INITRG EQU $0011	
RES EQU $D0	;Адрес ячейки результата
ORG RomStart	
EX_4_1: MOVB #$08, INITRG	
XGDY	;Обменять содержимое регистров Y и D.

STX REX	;Сохранение значения регистра X в ячейки памяти с адресами RES и RES+1(временно).
ANDA RES	;Логическое «И» между аккумулятором A и ячейкой памяти RES.
ANDB RES+1	;Логическое «И» между аккумулятором B и ячейкой памяти RES+1.
STD RES	;Сохранение результата.
JMP *	;Бесконечный переход по адресу текущей команды.
ORG StartVector DC.W EX_4_1	

Пример 4.2. Произвести один циклический сдвиг влево содержимого двух ячеек памяти ByteH и ByteL, находящихся в ОЗУ МК.

Команда	**Комментарий**
RamStart EQU $0000	;Назначается адрес для первой ячейки ОЗУ.
RomStart EQU $E000	;Назначается начальный адрес прикладной программы.
StartVector EQU $FFFE	;Назначается адрес для вектора начального запуска.
INITRG EQU $0011	
ByteH EQU RamStart	;Старший байт данных располагается по адресу начала ОЗУ МК.
ByteL EQU RamStart+1	;Младший байт данных.
ORG RomStart	
EX_4_2_2: MOVB #$08, INITRG	
LSL ByteL	;Логический сдвиг влево содержимого ячейки памяти ByteL. Старший разряд помещается в бит

	С регистра CCR, а младший становится равным 0.
ROL ByteL	;Циклический сдвиг влево содержимого ячейки памяти ByteH. В младший ее разряд помещается значение бита С от предыдущей операции.
BCC quit	;Если С = 0, то сдвиг завершён, и переходим на пустой цикл. Иначе делаем младший разряд в ячейке ByteL равным 1.
LDAB #1	;Загружаем единицу в аккумулятор В.
ORAB ByteL	;Логическое ИЛИ между числом 1 и ячейкой ByteL.
STAB ByteL	;Сохранение в ячейку памяти ByteL окончательного значения.
quit JMP quit	;Бесконечный переход по адресу текущей команды.
ORG StartVector DC.W EX_4_2_2	

Другой вариант решения:

Команда	**Комментарий**
RamStart EQU $0000	;Назначается адрес для первой ячейки ОЗУ.
RomStart EQU $E000	;Назначается начальный адрес прикладной программы.
StartVector EQU $FFFE	;Назначается адрес для вектора начального запуска.
INITRG EQU $0011	
ByteH EQU RamStart	;Старший байт данных располагается по адресу начала ОЗУ МК.
ByteL EQU RamStart+1	;Младший байт данных.
ORG RomStart	
EX_4_2_2: MOVB #$08, INITRG	

LDD ByteH	;Загрузка в аккумулятор D двух ячеек памяти ByteH и ByteL.
LSLD	;Логический сдвиг влево содержимого аккумулятора D. Старший разряд помещается в бит C регистра CCR, а младший становится равным нулю.
BCC save	;Если C = 0, то сдвиг завершен, и переходи на сохранение результата.
BSET ByteL, $01	;Иначе устанавливаем в 1. Младший разряд в ячейке ByteL.
save STD ByteH	;Сохраняем результат в ячейках памяти ByteH и ByteL.
JMP *	;Бесконечный переход по адресу текущей команды.
ORG StartVector DC.W EX_4_2_2	

Пример 4.3. Установить в 1 в значения 2-го и 5-го битов двух ячеек памяти ByteH и ByteL. Установить в 0 разряд 7 и инвертировать разряд 4 в них же. Остальные биты оставить неизменными. Младший разряд числа именуем нулевым.

Для принудительной установки в единицу определенных битов числа можно использовать команды поразрядного логического ИЛИ (начинаются на OR). При этом биты, подлежащие изменению, указываются в команде с помощью однобайтовой маски. Единица в ней означает, что разряд необходимо установить в единицу независимо от его прежнего значения, ноль – оставить неизменным.

Для выборочного принудительного сброса в ноль разрядов числа можно использовать команды поразрядного логического И (начинаются на AND). При этом биты, подлежащие модификации, отмечаются в маске единицами, а которые требуется оставить неизменными – нулями.

Для инвертирования определенных разрядов пользуются командами исключающего ИЛИ (начинаются на EOR). Биты, подлежащие изменению, тоже указываются при помощи маски.

Также для манипуляции отдельными разрядами в числе можно пользоваться командами битового процессора. С помощью инструкции BSET в

байте данных маскированные разряды устанавливаются в единицу. Команда BCLR, наоборот, сбрасывает определенные биты в ноль.

Команда	Комментарий
RamStart EQU $0000	;Назначается адрес для первой ячейки ОЗУ.
RomStart EQU $E000	;Назначается начальный адрес прикладной программы.
StartVector EQU $FFFE	;Назначается адрес для вектора начального запуска.
ByteH EQU RamStart+1	;Старший байт данных располагается по адресу начала ОЗУ МК.
ByteL EQU RamStart+1	;Младший байт данных.
Mask1 EQU %00100100	;Маска, выделяющая разряды 2 и 5 числа.
Mask0 EQU %10000000	;Маска, выделяющая разряд 7 числа.
MaskINV EQU %00010000	;Маска, выделяющая разряд 4 числа.
ORG RomStart	
EX_4_3: MOVB #$08, $11	
LDD ByteH	;Загрузка в аккумулятор D двух ячеек памяти ByteH и ByteL.
ORAA #Mask1	;Логическое ИЛИ над содержимым регистра A и маской Mask1.
ORAB #Mask1	; Логическое ИЛИ над содержимым регистра B и маской Mask1.
EORA #MaskINV	;Исключающее ИЛИ над содержимым регистра A и маской Mask3.
EORB #MaskINV	;Исключающее ИЛИ над содержимым регистра B и маской Mask3.
STD ByteH	;Сохранение промежуточного результата.
BCLR ByteH,Mask0	;Очищение бита 7 в ячейке памяти с адресом ByteH.
BCLR ByteL,Mask0	;Очищение бита 7 в ячейке памяти с адресом ByteL.

JMP *	;Бесконечный переход по адресу текущей команды.
ORG StartVector DC.W EX_4_3	

Задания для самостоятельной работы

1. Напишите программу, осуществляющую сдвиг влево 10 ячеек памяти таким образом, чтобы выдвигаемый из старшей ячейки памяти бит становился на место младшего бита в младшей ячейке.

2. Напишите программу, осуществляющую сдвиг вправо 12 ячеек памяти таким образом, чтобы выдвигаемый из младшей ячейки памяти бит становился на место старшего бита в старшей ячейке.

3. Напишите программу, осуществляющую умножение 2-байтового числа на 2^n, при этом n < 6 хранится в ячейке памяти ОЗУ.

4. Напишите программу, осуществляющую деление 2-байтового числа на 2^n, при этом n < 8 хранится в ячейке памяти ОЗУ.

5. Написать программу, заполняющую ячейки памяти $0100…$0107 соответствующими битами регистра A. То есть, например, если бит 0 в регистре A сброшен, то в ячейку $0100 записывается 0, если установлен - $01 (или $FF).

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

7. Написать программу, копирующую содержимое ячеек памяти DATA1 и DATA2 в регистр D таким образом, что старшие нибблы (полубайты) DATA1 и DATA2 составляли старший байт регистра D, а младшие нибблы – младший.

8. Написать программы сбрасывающую биты в регистре A, если соответствующие биты ячеек памяти DATA1 и DATA2 установлены. Остальные биты должны оставаться в исходном состоянии.

9. Написать программу, позволяющую инвертировать те биты ячейки памяти DATA1, которые сброшены в ячейки памяти DATA2. Остальные биты должны оставаться в исходном состоянии.

Контрольные вопросы

1. Перечислите основные логические операции, осуществляемые в микроконтроллере?
2. Для чего предназначен стек?
3. Какие основные операции выполняют логические команды ?
4. Что включает в себя система команд процессора?
5. Какие способы существуют программирования?
6. Назовите основные режимы адресации ?
7. Что такое регистры общего назначения?

Лабораторная работа №5.

Команды условных и безусловных переходов. Работа с массивами.

Цель работы - В данной лабораторной работе изучается группа команд для управления ходом вычислительного процесса. Эта группа объединяет команды условных и безусловных переходов. Рассматриваются примеры организации ветвления программ, приобретаются навыки использования команд условных переходов для составления типовых фрагментов программ на языке ассемблера. Рассматриваются примеры сортировки чисел, приобретаются навыки составления алгоритмов и их реализации на основе команд по анализу кодовых массивов по тем или иным признакам.

Описание лабораторной работы

В процессе подготовки к лабораторной работе №5 рекомендуется повторить предыдущий материал. Подробно ознакомиться с группой команд условных и безусловных переходов. На основе полученных теоретических сведений самостоятельно отладить примеры, которые рассматриваются ниже в практической части.

Практическая часть

Пример 5.1. Сравнить информацию, хранящуюся в ячейках памяти DATA1 и DATA2.Установить в ячейки Result:

- Бит 0 в 1, если коды в исходных ячейках равны друг другу.
- Бит 1 в 1, если 0 < DATA1 - DATA2 < 20.
- Бит 2 в 1, если 0 < DATA2 – DATA1 < 20.
- Бит 3 в 1, если DATA1 - DATA2 ≥ 20.
- Бит 4 в 1, если DATA2 – DATA1 ≥ 20.

RAMStart	EQU $0000
ROMStart	EQU $E000
StartVector	EQU $FFFE

INITRG	EQU $0011
Data1	EQU $RAMstart+$10
Data2	EQU $RAMstart+$11
Result	EQU $RAMstart+$20
ORG ROMstart	

EX_5_1	MOVB	# $08, INITRG	
	CLR	Result	Очистка ячейки результата.
	LDAB	Data2	Загрузка в аккумулятор *B* данных из ячейки Data2.
	CMPB	Data1	Сравнение Data2 и Data1
	BNE	CHK4	Если не равны, то перейти по метке CHK5.
	INC	Result	Иначе увеличить на 1 ячейку Result.
	JMP	quit	И закончить исполнение программы.
CHK4	SUBB	Data1	Вычесть из Data2 значение ячейки памяти Data1.
	BCS	CHK3	Если результат отрицательный, то перейти на CHK4.
	CMPB	#20	Иначе сравнить полученную разность с числом 20.
	BCS	RES2	Если меньше 20, то перейти на RES2.
	BEET	Result	Иначе Data2 – Data1 ≥ 20, значит, надо установить бит 4 результирующей ячейки в 1.
	JMP	quit	
RES2	BSET	Result, $04	Если 0 < Data2 – Data1 < 20, то

			установить бит 2 результата в 1
	JMP	quit	
CHK3	LDAA	Data1	Загрузка в аккумулятор *A* данных из ячейки Data1.
	SUBA	Data2	Вычитание из Data1 и значение ячейки памяти Data2.
	CMPA	#20	Сравнение полученной разности с числом 20.
	BCS	RES1	Если меньше 20, то переход к метке RES1.
	BSET	Result, $08	Иначе Data1 – Data2 ≥ 20, значит надо установить бит 3 результирующей ячейки в 1.
	JMP	quit	
RES1	BSET	Result, $02	Если 0 < Data1 – Data2 <20, то установить бит 1 результата в 1.
quit	JMP	quit	
	ORG	StartVector	
	DC.W	EX_5_1	

Пример 5.2 Дан массив однобайтовых чисел без знака. Начальный адрес – $110, количество элементов – 15. Переписать массив, начиная с адреса $130, заменив числа, у которых младший бит равен нулю, старший – единице, кодом $A9. Определить количество таких чисел.

RAMStart	EQU $0000	
ROMStart	EQU $E000	
StartVector	EQU $FFFE	
INITRG	EQU $0011	

Array1	EQU $0110	Адрес первого элемента исходного массива.
Array2	EQU $0130	Адрес первого элемента конечного массива.
N	EQU $0150	Ячейка памяти, в которую помещается количество элементов, подвергшихся замене.
CNT	EQU RAMstart	Ячейка памяти для счета циклов.
EXC	EQU $A9	Код, на который необходимо заменить элементы массива, удовлетворяющие заданному условию.
	ORG ROMstart	
EX_5_2	MOVG # $08, INITRG	
	CLR N	Очистка ячейки с количеством измененных элементов
	MOVB # $15, CNT	Загрузка счетчиков циклов начальным значением
	LDX #Array1	Загрузка в регистр *X* адреса первого элемента исходного массива.
CHK	LDAB , X	Загрузка в аккумулятор *B* значения ячейки с адресом, находящимся в *X*.
	ANDB # $81	Выделение старшего и младшего битов.
	CMPB # $80	Сравнение с желаемым значением.
	BNE INIT	Если не равны, перейти по метке INIT.
	INC N	Иначе увеличить счетчик заменённых элементов на 1.
	LDAB , EXC	Загрузить код для замены.

	BRA SAVE	Безусловный переход на SAVE.
INIT	LDAB , X	Загрузка в аккумулятор *A* элемента исходного массива.
	STAB $20, X	Сохранение нового массива.
	INX	Увеличение адреса, хранящегося в регистре *X*, на 1.
	DEC CNT	Уменьшение счетчика циклов.
	BNE CHK	Если обработаны не все элементы, перейти по метке CHK.
	JMP *	
	ORG StartVector	
	DC.W EX_5_2	

Предложенный способ решения имеет ряд особенностей.

Во-первых, для проверки разрядов чисел мы воспользовались универсальным методом, который не зависит от местонахождения в памяти чисел первого массива. Метод заключается в выделении с помощью маски и команды «AND» значимых разрядов числа, т.е. тех, которые необходимо проверить по условию задачи. Элемент массива, ранее находившийся в аккумуляторе, при этом перестает соответствовать значению, записанному в ячейке памяти. Далее значение аккумулятора сравнивается с эталонной константой, которая гарантирует совпадение выделенных разрядов с заданными значениями. Обратите внимание, для переноса элемента из первого массива во второй, элемент необходимо заново восстановить в аккумулятор. Чтобы этого не приходилось делать, для анализа условий можно использовать команды битового процессора.

Во-вторых, мы воспользовались тем фактом, что все элементы первого массива и все элементы второго массива находятся в пределах одной страницы памяти. Поэтому возможно использование команды "INX’’ вместо “LEAX 1,X”.

Задания для самостоятельной работы

1. Дан массив однобайтовых чисел со знаком. Начальный адрес – $200, количество элементов массива – 27. Начиная с адреса $100, переписать только положительные четные числа, определить количество отрицательных чисел.
2. Дан массив однобайтовых чисел со знаком. Начальный адрес – $120, код последнего элемента – $E1. Найти среднее арифметическое максимального и минимального положительных чисел и количество отрицательных чисел.
3. Дан массив однобайтовых чисел со знаком. Начальный адрес – $E200, код последнего элемента – $E1. Найти среднее арифметическое максимального и минимального положительных чисел и количество отрицательных чисел.
4. Произвести сортировку по возрастанию чисел, расположенных в ячейках $100..$120.
5. Произвести сортировку по убыванию чисел, расположенных в ячейках $50..$70.
6. Дан массив двухбайтовых чисел в прямом коде со знаком. Начальный адрес – $120, код последнего элемента – $1A24. Начиная с адреса $150 переписать только положительные четные числа, определить количество отрицательных чисел.
7. Дан массив двухбайтовых чисел без знака. Начальный адрес – $E300, количество элементов массива – 11. Определить максимальное четное число.
8. Дан массив однобайтовых чисел со знаком. Начальный адрес – $80, код последнего элемента – $12. Определить минимальное положительное нечетное число и его адрес.
9. Дан массив двухбайтовых чисел без знака. Начальный адрес – $100, код последнего элемента – $3724. Начиная с адреса $120записать только четные числа, определить количество нечетных чисел.
10. Дан массив двухбайтовых чисел без знака. Начальный адрес – $120, количество элементов массива – 8. Определить максимальное число и его адрес.

Контрольные вопросы

1. Дайте определение слову команда?
2. Для чего предназначены команды переходов ?
3. Дайте определение условным и безусловным переходам?
4. Что такое подпрограмма?

Литература

1. Барретт С. Ф., Пак Д. Дж. Встраиваемые системы. Проектирование приложений на микроконтроллерах семейства 68HC12 / HCS12 с применением языка C. — М.: Издательский дом «ДМКпресс», 2007. — 640 с

2. Ремизевич Т., Доброхотов Д. (2009) Лабораторный практикум «Шестнадцатиразрядные микроконтроллеры семейства HCS12»