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

Кафедра электронных вычислительных средств

ПРОЕКТИРОВАНИЕ ЭВС НА ОДНОКРИСТАЛЬНЫХ МИКРОКОНТРОЛЛЕРАХ

Методическое пособие
для студентов специальности 1-40 02 02
«Электронные вычислительные средства»
дневной формы обучения

В 3-х частях

Часть 3

М. В. Качинский, В. Б. Клюс, А. Б. Давыдов

ПРОГРАММИРОВАНИЕ 8-РАЗРЯДНЫХ МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА M68HC11

Минск БГУИР 2009

УДК 004.31(075.8)
ББК 32.973.26-02я73
П79

Р е ц е н з е н т –
доцент кафедры электронных вычислительных машин БГУИР,
кандидат технических наук А. А. Петровский

П79 **Проектирование** ЭВС на однокристальных микроконтроллерах : метод. пособие для студ. спец. 1-40 02 02 «Электронные вычислительные средства» днев. формы обуч. В 3 ч. Ч. 3 : Программирование 8-разрядных микроконтроллеров семейства M68HC11 / М. В. Качинский, В. Б. Клюс, А. Б. Давыдов. – Минск : БГУИР, 2009. – 42 с. : ил.
ISBN 978-985-488-421-9 (ч. 3)

В третьей части пособия рассматриваются основы ассемблера для 8-разрядных микроконтроллеров семейства M68HC11, программный эмулятор (симулятор) микроконтроллера MC68HC11, приводятся примеры программ на ассемблере. Даны также описания лабораторных работ по изучению симулятора Sim68w и программированию микроконтроллера MC68HC11.

УДК 004.31(075.8)
ББК 32.973.26-02я73

Части 1-я и 2-я изданы в БГУИР в 2000 и 2002 гг.

ISBN 978-985-488-421-9 (ч. 3)
ISBN 978-985-488-420-2

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В процессе разработки и отладки программного обеспечения микропроцессорных систем используются следующие программные средства:

– ассемблеры, компиляторы языков высокого уровня;
– симуляторы (программные эмуляторы);
– отладчики, редакторы связей (компоновщики, загрузчики).

В современных комплексах проектирования/отладки систем эти средства обычно работают совместно в составе интегрированной среды (оболочки) программирования. При этом программирование производится с помощью кросс-средств, инсталлированных на инструментальном компьютере. В качестве инструментальных компьютеров используются персональные компьютеры или рабочие станции. Операционными системами этих компьютеров служат различные версии Windows и UNIX.

При программировании систем на базе микроконтроллеров очень часто применяется язык ассемблера, так как его использование обеспечивает существенное уменьшение объема памяти программ и времени выполнения программных модулей (до 20 – 50 %). Упрощенные (демонстрационные) версии ассемблеров для микроконтроллеров фирмы Motorola предоставляются бесплатно рядом фирм и распространяются по сети Интернет. Эти версии обычно имеют ограничения на объем транслируемых программ (до нескольких сотен или тысяч строк), а также не обеспечивают многие сервисные возможности. Ассемблер с широким набором функциональных возможностей, включая макросы (макроассемблер), поставляется самим разработчиком – фирмой Motorola.

Симуляторы представляют собой программно-логическую модель микроконтроллера на персональном компьютере. Симуляторы позволяют загрузить файл с кодом разработанной программы в память эмулируемого микроконтроллера и выполнить любой фрагмент этой программы, наблюдая за изменением состояния любого программно-доступного ресурса микроконтроллера. Симулятор имитирует работу не только процессорного ядра микроконтроллера, но и его периферийных модулей. Симулятор можно рассматривать как виртуальный микроконтроллер, алгоритм функционирования которого полностью совпадает с алгоритмом работы реального микроконтроллера (микросхемы), но при этом имеются широкие дополнительные возможности по вмешательству в процесс выполнения тестируемой программы.

Основной недостаток симуляторов заключается в невозможности подключения реальных источников входной информации и формирования реальных выходных сигналов в системе. В результате с помощью симулятора можно проверить только правильность выполнения микроконтроллером программы, но нельзя проверить работоспособность аппаратной части проектируемой системы. Поэтому симуляторы используют на начальном этапе проектирования и отладки программного обеспечения микропроцессорной системы.

Симуляторы микроконтроллеров обычно не поставляются в виде отдельных средств разработки и отладки программного обеспечения, а входят в состав

отладчиков. Отладчики являются основным инструментом разработчика программного обеспечения, без которого невозможно получить «работоспособные» программы. Отладчик реализует различные режимы выполнения программы – пошаговый или с точками останова, позволяет производить просмотр и модификацию содержимого регистров и ячеек памяти, дизассемблирование команд программы. Отладчик может работать с программой на уровне объектного кода или в символическом виде, с использованием введенных программистом имен и меток. Последний вариант является наиболее удобным средством отладки, так как позволяет представлять и воспринимать информацию в наиболее наглядной и удобной для человека форме. Кроме симулятора, отладчик обычно содержит компоновщик/загрузчик объектного кода. Для отображения состояния микроконтроллера на экране компьютера отладчик использует многооконный графический интерфейс. Некоторые отладчики не только могут работать с симуляторами, но и могут реализовывать интерфейс со схемными эмуляторами, т. е. с отладочными средствами, построенными на базе реальных микроконтроллеров (отладочные платы и системы).

В данном пособии описывается свободно распространяемый кросс-ассемблер для 8-разрядных микропроцессоров и микроконтроллеров фирмы Motorola [5, 6].

1. ОСНОВЫ АССЕМБЛЕРА ДЛЯ МИКРОКОНТРОЛЛЕРА MC68HC11

1.1. Общие сведения об ассемблере для микроконтроллера MC68HC11

Ассемблер – это программа, которая обрабатывает операторы исходной программы на языке ассемблера и переводит их в исполняемый объектный код на машинном языке. Исходный текст программы не должен содержать непечатаемых управляющих символов. Поэтому он может быть подготовлен с использованием любого текстового редактора, который формирует выходной текст в ASCII формате.

Ассемблер для микроконтроллера MC68HC11 является кросс-ассемблером. *Кросс-ассемблер* позволяет для исходной программы, написанной на одном компьютере (главном), получить исполняемый код для другого компьютера (целевого). После этого исполняемый код может быть загружен и выполнен на целевом компьютере. В нашем случае главным является персональный компьютер, а целевым – микроконтроллер MC68HC11. Для выполнения лабораторных работ вместо микроконтроллера MC68HC11 используется его программный эмулятор Sim68w (sim68w.exe).

Язык ассемблера использует набор мнемонических обозначений: мнемоники машинных команд и директив ассемблера, символические имена, знаки операций и специальные символы. Язык ассемблера понимает мнемоники всех машинных команд, входящих в систему команд микроконтроллера MC68HC11. Директивы ассемблера определяют дополнительные действия, которые должен выполнить ассемблер во время трансляции программы. Эти директивы никогда не транслируются в машинные команды.

Ассемблер для микроконтроллера MC68HC11 является двухпроходным. Во время первого прохода исходный текст просматривается с целью составления таблицы символических имен, используемых в программе. При втором проходе с использованием составленной таблицы символических имен создается исполняемый объектный код. Объектный код создается в специальном S-формате фирмы Motorola (Motorola S Record format). Кроме того, при втором проходе создается листинг программы. Обработка исходного текста программы осуществляется последовательно, оператор за оператором: следующий оператор исходного текста читается только после завершения обработки текущего оператора. Ошибки, найденные ассемблером во время трансляции, вставляются в листинг программы перед строкой, в которой они обнаружены.

При рассмотрении правил оформления исходной программы на языке ассемблера будем использовать различные типы шрифтов для разной информации, а также специальные обозначения.

Примеры программ и символические имена оформляются `специальным шрифтом с равной шириной символов`. В описаниях синтаксиса директив ассемблера круглые скобки ( ) указывают необязательный параметр. Имена директив даются **ПРОПИСНЫМИ (ЗАГЛАВНЫМИ, БОЛЬШИМИ) БУКВАМИ ПОЛУЖИРНЫМ ШРИФТОМ**, а параметры – *строчными (малыми) буквами курсивом*.

Имена параметров заключаются в угловые скобки < >. Части синтаксиса, которые отмечены полужирным шрифтом, должны вводиться, как показано ниже. Части синтаксиса, которые отмечены курсивом, описывают тип информации, которая должна быть введена. Однако при этом все элементы, не стоящие внутри угловых скобок, должны вводиться, как показано ниже. Например, директива

(<*метка*>) **FDB** <*выражение*>(,<*выражение*>,...,<*выражение*>) (<*комментарии*>)

имеет один обязательный параметр (первый), который требуется всегда. Остальные параметры являются необязательными. Однако если эти параметры используются, то они обязательно должны быть разделены запятыми.

1.2. Кодирование исходной программы на языке ассемблера для микроконтроллера MC68HC11

Программа, написанная на языке ассемблера, состоит из последовательности исходных операторов. В свою очередь каждый исходный оператор состоит из последовательности ASCII-символов, закачивающейся управляющим символом CR (возврат каретки, шестнадцатеричный код 0D). Набор символов, которые распознает ассемблер, является подмножеством стандарта ASCII. Коды ASCII-символов приведены в табл. 1.

Таблица 1

Разряды 4–6		0	1	2	3	4	5	6	7
Разряды 0–3	0	NUL	DLE	SP	0	@	P	`	p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	“	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	BEL	ETB	‘	7	G	W	g	w
	8	BS	CAN	(	8	H	X	h	x
	9	HT	EM	)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[	k	{
	C	FF	FS	,	<	L	\	l	|
	D	CR	GS	-	=	M	]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	S1	US	/	?	O	_	o	DEL

Исходный оператор может содержать до четырех упорядоченных полей: поле *метки* (или «*» для строки комментариев), поле *операции* (мнемоника машинной команды или директива ассемблера), поле *операндов* и поле *комментариев*. Общий синтаксис для исходных операторов следующий:

(*метка*)(:) *мнемоника* (*список операндов*) (;)(*комментарии*)

Поле метки

Поле метки является первым полем исходного оператора. Поле метки может принимать одну из следующих форм:

1. Звездочка (*) как первый символ в поле метки указывает, что остальная часть исходного оператора является комментарием. При этом вся строка воспринимается ассемблером как комментарии. Комментарии игнорируются ассемблером и включаются в листинг только для программиста.

2. Пробел или символ табуляции как первый символ указывает, что поле метки пустое. В этом случае строка не имеет метки и не является строкой комментариев.

3. Символьный знак в первой позиции указывает, что данная строка (исходный оператор) имеет метку. В качестве символьного знака могут использоваться прописные (заглавные, большие) и строчные (малые) буквы английского алфавита (A…Z, a…z), десятичные цифры (0…9), специальные символы – точка (.), знак доллара ($) и символ подчеркивания (_). В качестве метки используется символическое имя (symbol). Символическое имя может содержать до 15 знаков, первый из которых должен быть буквой или специальным символом – точкой (.) или подчеркиванием (_). Все знаки являются значимыми, причем прописные и строчные буквы различаются.

Конкретное символическое имя может стоять в поле метки только один раз. *Если символическое имя встречается в поле метки более одного раза, то каждая ссылка на такое символическое имя помечается как ошибка*.

За исключением некоторых директив метке присваивается текущее значение программного счетчика. При этом метка указывает на первый байт команды или данных, которым она предшествует. Значение, присваиваемое метке, является абсолютным. Метка может заканчиваться двоеточием (:). Двоеточие после метки необязательно. Двоеточие (если оно используется) не является частью метки, а только отделяет метку от остальной части строки.

Метка может стоять одна в пустой строке. Ассемблер интерпретирует это как то, что метке присваивается текущее значение программного счетчика. При этом метка указывает на команду в следующей строке.

Поле операции

Поле операции располагается после поля метки и должно отделяться от него по крайней мере одним пробелом или символом табуляции. Поле операции должно содержать допустимую мнемонику машинной команды или директиву ассемблера. Перед проверкой мнемоники на допустимость прописные буквы преобразуются в строчные. Например, записи «nop», «NOP» и «NoP» интерпретируются как одна и та же мнемоника.

Поле операнда

Поле операнда, если оно необходимо, должно следовать за полем операции и отделяться от него по крайней мере одним пробелом или символом табуляции. Интерпретация поля операнда зависит от содержания поля операции. Поле

операнда может содержать символическое имя, выражение или комбинацию символических имен и выражений, разделенных запятыми.

Поле операнда машинных команд используется для задания операнда команды с помощью соответствующего способа адресации. Для микроконтроллера MC68HC11 используются следующие форматы поля операнда, приведенные в табл. 2.

Таблица 2

Формат поля операнда	Способ адресации
Операнд отсутствует	Неявный (регистровый)
<выражение>	Прямой, расширенный или относительный
#<выражение>	Непосредственный
<выражение>, X	Индексный, с использованием регистра X
<выражение>, Y	Индексный, с использованием регистра Y
<выражение> <выражение>	Установить или очистить бит(ы)
<выражение> <выражение> <выражение>	Проверить бит(ы) и перейти

Операнды команд битовых операций отделяются друг от друга пробелами, что позволяет использовать в таких командах индексную адресацию.

Выражение представляет собой комбинацию символических имен, констант, знаков математических операций и скобок. Выражение применяется для задания значения, которое будет использоваться в качестве операнда. Выражения составляются из символических имен, констант или символа «*» (означает текущее значение программного счетчика), соединяемых друг с другом одной из математических операций: +, –, *, /, %, &, |, ^. В данном ассемблере используются такие же операции, как и в языке программирования С:

+ – сложение;
– – вычитание;
* – умножение;
/ – деление;
% – остаток от деления;
& – поразрядное И;
| – поразрядное ИЛИ;
^ – поразрядное ИСКЛЮЧАЮЩЕЕ ИЛИ.

Выражения вычисляются слева направо. Для изменения порядка выполнения операций необходимо использовать скобки. Значение выражения вычисляется с использованием 16-разрядной двоичной целочисленной арифметики в дополнительном коде. При этом каждое символическое имя ассоциируется с 16-разрядным целым значением, которое заменяет это символическое имя во время вычисления выражения. Звездочка (*) используется в выражении как символическое имя, которое ассоциируется с текущим значением программного счетчика (*текущее значение программного счетчика всегда равно адресу первого байта машинной команды, не путать с регистром процессора с таким же названием*).

Константы представляют собой значения, которые не изменяются во время выполнения программы. Константы могут быть заданы в одном из пяти форматов: десятичном, шестнадцатеричном, двоичном, восьмеричном или в виде ASCII-кода. Формат указывается с помощью следующих префиксов:

$ – шестнадцатеричный;
% – двоичный;
@ – восьмеричный;
` – ASCII-код.

Константа без префикса интерпретируется как десятичная. Ассемблер преобразует константу, заданную в любом формате, в двоичный машинный код и отображает в листинге программы в шестнадцатеричном виде.

Десятичная константа представляет собой строку десятичных цифр. Значение десятичной константы должно находиться в диапазоне 0 – 65 535. Следующий пример показывает правильное и неправильное задание десятичных констант:

Правильно	**Неправильно**	**Пояснения**
12	123456	Содержит более пяти десятичных цифр
12345	12,3	Недопустимый символ

Шестнадцатеричная константа содержит максимум четыре символа из набора десятичных цифр (0 – 9) и прописных букв английского алфавита (A – F), которым предшествует знак доллара ($). Шестнадцатеричная константа должна находиться в диапазоне $0000 – $FFFF. Следующий пример показывает правильное и неправильное задание шестнадцатеричных констант:

Правильно	**Неправильно**	**Пояснения**
$12	ABCD	Отсутствует префикс $
$ABCD	$G2A	Недопустимый символ
$001F	$2F018	Содержит более четырех символов

Восьмеричная константа содержит максимум шесть цифр (0 – 7), которым предшествует знак (@). Восьмеричная константа должна находиться в диапазоне @0 – @177777. Следующий пример показывает правильное и неправильное задание восьмеричных констант:

Правильно	**Неправильно**	**Пояснения**
@17634	@2317234	Содержит более шести символов
@377	@277272	Выходит за допустимый диапазон
@177600	@23914	Недопустимый символ

Одиночный ASCII-символ может использоваться в качестве константы в выражении. ASCII-константе предшествует одинарная кавычка (`). Любой символ, кроме одинарной кавычки, может использоваться в качестве символьной

константы. Следующий пример показывает правильное и неправильное задание символьных констант:

Правильно	**Неправильно**	**Пояснения**
`*	`VALID	Содержит более одного символа

Случай, когда символьная константа содержит более одного символа, не рассматривается ассемблером как ошибка: ассемблер формирует ASCII-код первого символа и игнорирует остальные символы.

Поле комментариев

Последнее поле в исходном операторе – это поле комментариев. Поле не является обязательным, и оно включается в листинг программы с целью документирования. Поле комментариев отделяется от поля операнда (или поля операции, если операнды отсутствуют) по крайней мере одним пробелом или символом табуляции. Поле комментариев может содержать любой печатаемый ASCII-символ.

Комментарии могут начинаться символом точка с запятой (;). В этом случае комментарии могут начинаться с любой позиции.

1.3. Директивы ассемблера для микроконтроллера MC68HC11

В описаниях директив ассемблера используются следующие обозначения:

<комментарии>	поле комментариев исходного оператора;
<метка>	метка исходного оператора;
<выражение>	выражение ассемблера;
<выр>	выражение ассемблера;
<строка>	строка ASCII-символов;
<ограничитель>	ограничитель строки.

BSZ – BLOCK STORAGE OF ZEROS

(<*метка*>) **BSZ** <*выражение*> (<*комментарии*>)

По этой директиве ассемблер выделяет в памяти блок байтов. Каждый байт инициализируется нулевым значением. Количество выделяемых байтов задается выражением в поле операнда. Если выражение содержит символическое имя, которое не определено или определяется позже, либо значение выражения равно нулю, ассемблер генерирует ошибку.

EQU – EQUATE SYMBOL TO A VALUE

<*метка*> **EQU** <*выражение*> (<*комментарии*>)

Данная директива назначает метке значение выражения, стоящего в поле операнда. Директива EQU является альтернативным способом назначения значения метке (*напомним, что обычно метке присваивается текущее значение*

программного счетчика). Метка не может быть переопределена где-нибудь в программе. Выражение не может содержать символических имен, которые не определены или определяются позже. Если выражение содержит такие символические имена, то ассемблер генерирует ошибку.

FCB – FORM CONSTANT BYTE

(<*метка*>) **FCB** <*выр*>(,<*выр*>,…,<*выр*>) (<*комментарии*>)

Эта директива может иметь один или несколько операндов, разделенных запятыми. Значение каждого операнда задается с помощью восьми разрядов и хранится в виде одного байта исполняемого объектного кода программы. Если в директиве задано несколько операндов, то они хранятся в следующих друг за другом байтах памяти. Операнд может быть числовой или символьной константой, символическим именем либо выражением. Если в директиве задается множество операндов, то один или несколько операндов могут отсутствовать, что указывается с помощью двух следующих друг за другом запятых. Таким операндам присваиваются нулевые значения, которые хранятся в соответствующих байтах объектного кода. Если при вычислении значения операнда все старшие восемь разрядов получаются не равными 1 или 0, то ассемблер генерирует ошибку.

FCC – FORM CONSTANT CHARACTER STRING

(<*метка*>) **FCC** <*ограничитель*><*строка*><*ограничитель*> (<*комментарии*>)

Эта директива используется для задания ASCII-строки в последовательных байтах памяти. Адрес байта памяти, в котором размещается ASCII-код первого символа строки, определяется текущим значением программного счетчика. Метка соответствует первому байту строки, т. е. значение метки равно адресу первого байта строки. Строка может содержать любой печатаемый ASCII-символ. Строка определяется между двумя одинаковыми ограничителями. В качестве ограничителя может использоваться любой печатаемый ASCII-символ. Первый не пустой символ после FCC-директивы рассматривается как ограничитель.

Например, директива

```
LABEL1    FCC    ,ABC,
```

размещает ASCII-строку `ABC` (английские буквы) по адресу `LABEL1`.

FDB – FORM DOUBLE BYTE CONSTANT

(<*метка*>) **FDB** <*выр*>(,<*выр*>,…,<*выр*>) (<комментарии>)

Эта директива может иметь один или несколько операндов, разделенных запятыми. 16-разрядное значение, соответствующее каждому операнду, хранится в двух соседних байтах исполняемого объектного кода программы, причем сначала (т. е. по меньшему адресу) располагается старший байт операнда. Адрес байта памяти, в котором размещается старший байт первого операнда, определяется текущим значением программного счетчика. Метка соответствует

старшему байту первого операнда, т. е. значение метки равно адресу старшего байта первого операнда. Если в директиве задано несколько операндов, то они хранятся в следующих друг за другом байтах памяти. Операнд может быть числовой или символьной константой, символическим именем либо выражением. Если в директиве задается множество операндов, то один или несколько операндов могут отсутствовать, что указывается с помощью двух следующих друг за другом запятых. Таким операндам присваиваются нулевые значения, которые хранятся в соответствующих байтах объектного кода.

FILL – FILL MEMORY

(<*метка*>) **FILL** <*выражение*>,<*выражение*>

По этой директиве ассемблер выделяет в памяти блок байтов. Все байты блока инициализируются одним и тем же значением – некоторой константой. Значение константы задается первым выражением директивы. Второе выражение определяет размер выделяемого блока памяти, т. е. общее количество инициализируемых константой байтов памяти. Значение первого выражения должно находиться в диапазоне 0 – 255. Выражения не могут содержать символических имен, которые не определены или определяются позже.

ORG – SET PROGRAM COUNTER TO ORIGIN

ORG <*выражение*> (<*комментарии*>)

Данная директива задает новое значение программного счетчика, которое определяется выражением в поле операнда. Результат трансляции последовательности операторов, расположенных в исходной программе после директивы ORG, размещается ассемблером в памяти начиная с нового значения программного счетчика. Если в исходной программе не содержится ни одной директивы ORG, то программный счетчик инициализируется нулевым значением (считается, что исходное значение программного счетчика равно нулю). Выражение не может содержать символических имен, которые не определены или определяются позже.

RMB – RESERVE MEMORY BYTES

(<*метка*>) **RMB** <*выражение*> (<*комментарии*>)

Данная директива увеличивает программный счетчик на значение, которое определяется выражением в поле операнда. Директива RMB резервирует блок памяти, размер которого в байтах равен значению выражения. Выделенный блок памяти не инициализируется никаким значением. Выражение не может содержать символических имен, которые не определены или определяются позже. Эта директива обычно применяется для резервирования временной рабочей области памяти или таблицы, которая будет использоваться в дальнейшем.

ZMB – ZERO MEMORY BYTES (эквивалентна директиве BSZ)

(<*метка*>) **ZMB** <*выражение*> (<*комментарии*>)

По этой директиве ассемблер выделяет в памяти блок байтов. Каждый байт инициализируется нулевым значением. Количество выделяемых байтов задается выражением в поле операнда. Если выражение содержит символическое имя, которое не определено или определяется позже, либо значение выражения равно нулю, ассемблер генерирует ошибку.

1.4. Пример оформления исходной программы на ассемблере для микроконтроллера MC68HC11

Требуется определить длину строки в коде ASCII, которая размещается в памяти начиная с адреса $0041. Конец строки отмечен символом «точка» (шестнадцатеричный код 2E). Результат следует поместить в ячейку с адресом $0040.

Исходный текст программы содержит 29 строк.

```
<1>  * Программа определяет длину строки в коде ASCII.
<2>  * Строка расположена в памяти по адресу $0041.
<3>  * Конец строки отмечен символом "точка".
<4>  * Результат сохраняется в памяти по адресу $0040.
<5>
<6>  * Объявляем адреса начала областей данных и программы
<7>  DataAddr  EQU  $0000     адрес начала области данных
<8>  CodeAddr  EQU  $F000     адрес начала области программы
<9>
<10> * Область данных
<11>           ORG  DataAddr+$0040
<12> Length    FCB  0
<13> String    FCC  "EXAMPLE."
<14>
<15> * Область программы
<16>           ORG  CodeAddr
<17> START:    clrb           длина строки = 0
<18>           ldaa #'.       задание символа "." для сравнения
<19>           ldx  #String   задание указателя начала строки
<20> CHPER:    cmpa 0,X       текущий символ - точка?
<21>           beq  DONE      да, идти к метке DONE
<22>           incb           нет, увеличить длину строки на 1
<23>           inx            увеличиваем указатель строки на 1
<24>           bra  CHPER     переход к проверке следующего символа
<25> DONE:     stab Length    сохраняем длину строки
<26>
<27> * Задаем стартовый адрес программы
<28>           ORG  $FFFE
<29>           FDB  START
```

Строки 7 – 8 определяют адреса, по которым в памяти микроконтроллера будут располагаться области данных и программы.

Строки 11 – 13 определяют область данных программы. Строка 12 определяет ячейку памяти, в которой будет сохраняться длина заданной ASCII-строки. Адрес этой ячейки памяти указывается с помощью директивы ORG в строке 11 программы. Строка в коде ASCII задается в строке 13 программы.

Строка 16 определяет адрес первой команды программы.

Строки 17 – 25 содержат команды программы.

Строка 17 обнуляет аккумулятор B, который используется в качестве счетчика числа символов в заданной ASCII-строке (выполняет инициализацию счетчика символов).

В строке 18 символ «точка» определяется в качестве символа для сравнения: ASCII-код символа «точка» загружается в аккумулятор A.

В строке 19 задается указатель на начало ASCII-строки: адрес первого символа строки загружается в индексный регистр X.

В строках 20 – 24 организован цикл для подсчета числа символов в ASCII-строке. В строках 20 – 21 ASCII-код текущего символа сравнивается с кодом символа «точка». Если текущий символ не является точкой, то выполняются команды, находящиеся в строках 22 – 23, по которым увеличиваются на 1 счетчик числа символов и указатель текущего символа. Затем строка 24 осуществляет переход в начало цикла.

Когда встречается символ «точка», команды, стоящие в строках 22 – 24, пропускаются, и подсчитанная длина ASCII-строки в строке 25 программы сохраняется в ячейке памяти.

Строка 25 помещает подсчитанное число символов в ASCII-строке в память по заданному адресу.

Строки 28 – 29 определяют стартовый адрес программы (адрес первой выполняемой команды программы) путем задания вектора прерывания установки начального состояния микроконтроллера.

Строки 1 – 4, 6, 10, 15 и 27 являются строками комментариев.

Строки 5, 9, 14 и 26 пустые и используются для улучшения читабельности программы.

Ниже приводится листинг программы (программа после ассемблирования) для данного примера.

```
0001                    * Программа определяет длину строки в коде ASCII.
0002                    * Строка расположена в памяти по адресу $0041.
0003                    * Конец строки отмечен символом "точка".
0004                    * Результат сохраняется в памяти по адресу $0040.
0005
0006                    * Объявляем адреса начала областей данных и программы
0007 0000               DataAddr    EQU   $0000   адрес начала области данных
0008 f000               CodeAddr    EQU   $F000   адрес начала области программы
0009
0010                    * Область данных
0011 0040                       ORG  DataAddr+$0040
0012 0040 00            Length  FCB  0
0013 0041 45 58 41 4d   String  FCC  "EXAMPLE."
```

```
          50 4c 45 2e
0014
0015                    * Область программы
0016 f000                       ORG  CodeAddr
0017 f000 5f            START:  clrb         длина строки = 0
0018 f001 86 2e                 ldaa #'.     задание символа "." для сравнения
0019 f003 ce 00 41              ldx  #String задание указателя начала строки
0020 f006 a1 00         CHPER:  cmpa 0,X     текущий символ - точка?
0021 f008 27 04                 beq  DONE    да, идти к метке DONE
0022 f00a 5c                    incb         нет, увеличить длину строки на 1
0023 f00b 08                    inx          увеличиваем указатель строки на 1
0024 f00c 20 f8                 bra  CHPER   переход к проверке следующего симво-
ла
0025 f00e d7 40         DONE:   stab Length  сохраняем длину строки
0026
0027                    * задаем стартовый адрес программы
0028 fffe                       ORG  $FFFE
0029 fffe f0 00                 FDB  START
```

Каждая строка листинга содержит номер строки, адрес и байты объектного кода, полученные в результате ассемблирования соответствующей строки исходной программы, а также саму строку исходной программы. Адрес и байты объектного кода задаются в шестнадцатеричном коде.

Исполнимый объектный код программы в S-формате, который может быть загружен и выполнен на целевом компьютере – микроконтроллере MC68HC11 или его программном эмуляторе Sim68w, имеет следующий вид:

```
S10C0040004558414D504C452E79
S113F0005F862ECE0041A10027045C0820F8D7407B
S105FFFEF0000D
S9030000FC
```

S-формат представляет программу и данные объектного кода в печатаемом (ASCII) формате, что позволяет просматривать объектный код с помощью стандартных средств (в том числе и во время его загрузки в целевую систему).

Объектный код в S-формате состоит из отдельных S-записей. S-запись представляет собой строку символов, состоящую из нескольких полей, которые задают тип записи, длину записи, адрес памяти, код/данные и контрольную сумму. Каждый байт записи кодируется с помощью двух шестнадцатеричных цифр (двухсимвольного шестнадцатеричного числа): первый символ задает четыре старших бита, второй – четыре младших бита.

S-запись имеет следующий формат:

TYPE RECORD LENGTH ADDRESS CODE/DATA CHECKSUM

Назначение полей S-записи приведены в табл. 3.

Таблица 3

Поле	Количество символов	Назначение
Type	2	Тип S-записи – S1 или S9
Record length	2	Количество пар символов в записи, исключая тип и длину записи
Address	4	Два байта адреса, по которому содержимое поля код/данные загружается в память
Code/data	0 – 2n	От 0 до n байт исполнимого кода, загружаемых данных или служебной информации
Checksum	2	Проинвертированное значение младшего байта суммы значений, представленных парами символов, начиная с поля длины записи

Каждая запись должна заканчиваться управляющим символом CR (возврат каретки, шестнадцатеричный код 0D).

В ассемблере используется два типа записи:

• S1 – запись содержит поле код/данные и два байта адреса, по которому содержимое этого поля загружается в память;

• S9 – последняя запись объектного кода. Такая запись не содержит поля код/данные. Поле адреса содержит нули.

Например, для нашей программы первая S-запись расшифровывается следующим образом:

S1 – запись типа S1, которая содержит код/данные;

0C – шестнадцатеричное значение 0C (десятичное 12), которое указывает, что запись содержит 12 пар символов (12 байт);

0040 – двухбайтный адрес $0040, по которому данные загружаются в память;

следующие 9 пар символов (байт) представляют собой загружаемые в память данные;

79 – контрольная сумма первой записи.

2. СИМУЛЯТОР (ПРОГРАММНЫЙ ЭМУЛЯТОР) МИКРОКОНТРОЛЛЕРА MC68HC11

Важнейшей составляющей системы, основанной на микроконтроллере, является программное обеспечение, позволяющее на жестко заданной аппаратной конфигурации выполнять любые задачи по обработке информации. Разработка программного обеспечения в таком случае состоит из ряда этапов: определение источников/получателей информации и необходимых протоколов передачи данных; разработка программного обеспечения и получение листинга программы; кодирование программы в машинных кодах микроконтроллера; запись программы в ПЗУ (РПЗУ) микроконтроллера; отладка программы.

Так как от написания первого варианта программы до получения отлаженного, работоспособного исполняемого кода проходит зачастую большой промежуток времени, то многократная проверка работоспособности программы на аппаратной части системы серьезно затрудняет эту задачу. Самым простым и правильным решением в этом случае является использование *симулятора* (*программного эмулятора*) микроконтроллера – программного средства, имитирующего работу микроконтроллера и предоставляющего все необходимые инструменты для разработчика программного обеспечения (написания исходного текста, компиляции его в объектный файл, отладки).

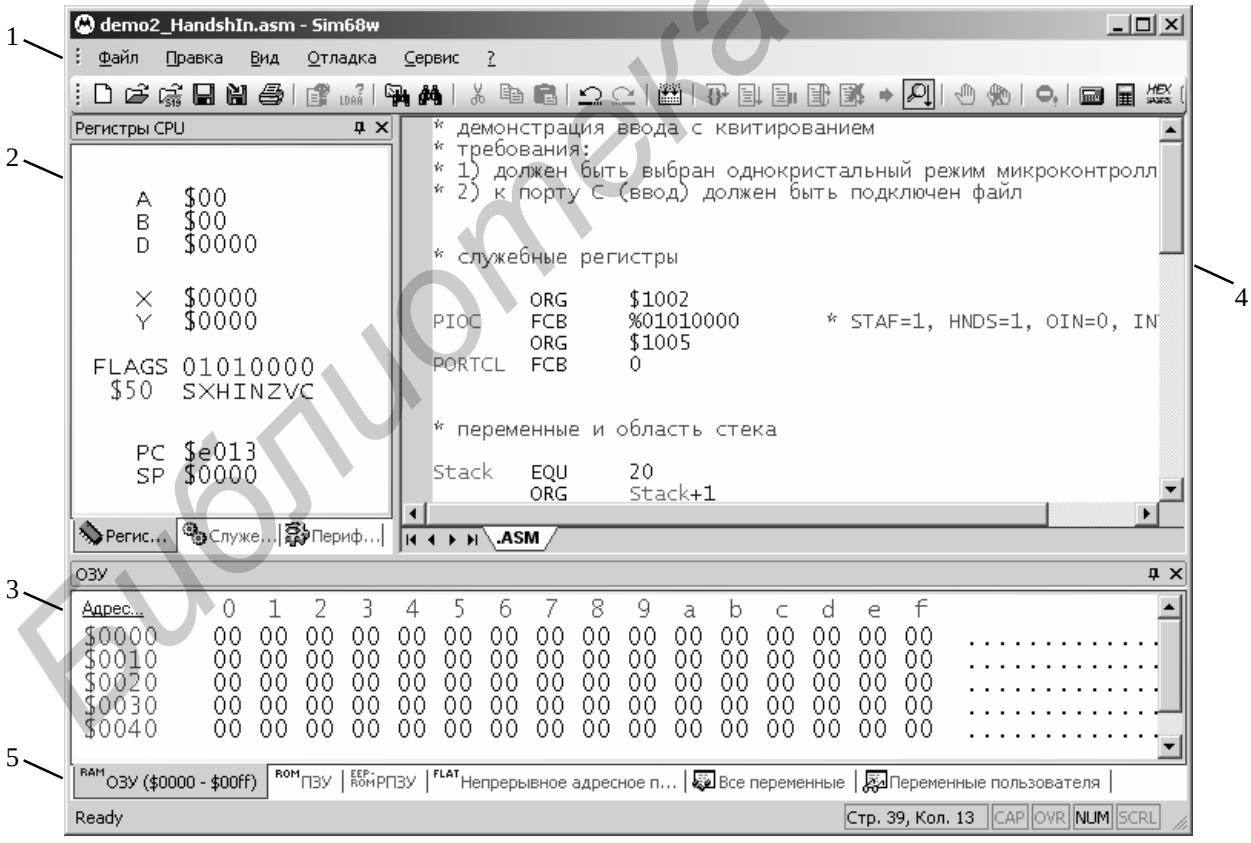

Рис. 1. Главное окно симулятора

Главное окно симулятора Sim68w микроконтроллера MC68HC11 (рис. 1) в режиме набора исходного текста и отладки программы включает следующие элементы:

1 – панель инструментов со всеми функциями, выполняемыми эмулятором;

2 – панель регистров, состоящая из трех вкладок (регистры CPU, служебные регистры, периферийные устройства);

3 – панель для работы с памятью микроконтроллера с возможностью выбора типа памяти и перемещения к ячейке с любым адресом;

4 – рабочее окно для набора и отладки программы;

5 – строка состояния.

Приступая к работе с симулятором, необходимо обратить внимание на три четко разделенные области главного окна: 2 (регистры), 3 (память) и 4 (рабочая область). Основным окном при наборе и отладке является рабочая область, имеющая три дочерние окна с вкладками, обозначенными, как «.ASM», «.S19», «.LST», «Прерывания» и др. Перемещение между одновременно открытыми окнами осуществляется щелчком мыши по соответствующей вкладке.

Исходный текст программы (ассемблерный файл) набирается, открывается и сохраняется из окна «.ASM» (окно исходного текста). Листинг программы, формируемый в процессе компиляции, отображается в окне «.LST» (окно листинга). В начале работы окно листинга скрыто. Выполняемый объектный код программы отображается после компиляции в окне «.S19». Отображаемая в нем информация определяется дизассемблированным содержимым памяти микроконтроллера. При внесении изменений в области памяти, отведенной для машинного кода программы, дизассемблирование производится повторно с запросом на подтверждение.

Окно исходного текста представляет из себя обычный текстовый редактор с функциями копирования/вставки и выделения фрагмента текста. Кроме того, редактор осуществляет выделение элементов синтаксиса исходного текста различным цветом:

черный	–	команды ассемблера (LDA, STA и т. д.), директивы ассемблера (FCB и др.);
зеленый	–	символические имена, имена регистров, имена меток, неопознанные выражения (возможно, содержащие ошибки);
темно-синий	–	числовые константы и выражения;
ярко-синий	–	строковые константы, заключенные в кавычки;
темно-красный	–	комментарии, которым предшествует символ «*».

Для простоты перемещения по исходному тексту в строке состояния 5 отображается номер строки и колонки, в которых находится курсор в данный момент. Для открытия существующего файла с исходным текстом программы используется кнопка «Открыть» на панели 1, продублированная в главном меню. Для сохранения файла исходного текста используются кнопки «Сохранить» и «Сохранить как…».

В остальном работа с текстовым редактором окна «.ASM» аналогична работе с текстовым редактором Блокнот или редактором исходных текстов среды VisualC++/Delphi.

Когда исходный текст пользовательской программы набран (или открыт из файла), можно компилировать его в машинные коды микроконтроллера. Перед компиляцией файл с исходным текстом должен быть сохранен на диск. Это требуется для того, чтобы внешний компилятор имел доступ к исходному тексту программы в виде файла.

Для компиляции исходного текста программы и получения листинга используется кнопка «Компилировать» (продублированная в меню «Отладка» и соответствующая клавише F7).

После успешной компиляции исходного текста программы созданный компилятором файл объектного кода загружается в память микроконтроллера и дизассемблируется. При этом объектный файл и файл с листингом программы становятся доступными в той же папке, где был сохранен файл исходного текста. В случае необходимости быстрого доступа непосредственно к этим файлам можно воспользоваться сервисной функцией «Проводник», которая открывает путь к папке с последним сохраненным файлом *.asm. Сразу после компиляции будет активировано окно объектного кода «.S19» – окно с дизассемблированным содержимым памяти (рис. 2).

BRK	Адрес	Данные	E	Мнемоника	Диапазон дизассемблирования...
•	$e000	b6 10 02	4	LDAA $1002	
•	$e003	84 7f	2	ANDA #$7f	
•	$e005	b7 10 02	4	STAA $1002	
•	$e008	de 15	4	LDX $15	
•	$e00a	b6 10 05	4	LDAA $1005	
•	$e00d	a7 00	4	STAA $00, X	
•	$e00f	08	3	INX	
•	$e010	df 15	4	STX $15	
•	$e012	3b	12	RTI	
⇨	$e013	0e	2	CLI	INH
•	$e014	8e 00 00	3	LDS #$0000	
•	$e017	cc 00 17	3	LDD #$0017	
●	$e01a	dd 15	4	STD $15	
•	$e01c	3e	11	WAI	
•	$e01d	20 fd	3	BRA $e01c	

.S19 | LST | Параллельный I/O

Рис. 2. Окно дизассемблированного объектного кода

При наличии ошибок пользователю открывается окно «.LST» с листингом (рис. 3), в котором строки с ошибками выделены красным, а строки с предупреждениями – желтым цветом.

Отладка программы может осуществляться либо в окне листинга, либо в окне объектного кода, либо с переключением между ними по мере необходимости. Для того чтобы иметь возможность видеть все символические имена, метки и директивы ассемблера, при отладке используется окно листинга. В случае ко-

гда необходимо проконтролировать выполнение программы непосредственно в памяти микроконтроллера, с возможностью выполнения команды, отсутствующей в листинге (например команды за пределами скомпилированной программы), а также для реализации возможности самомодификации программы, используется окно объектного кода. Оно удобно еще и тем, что в ходе отладки можно, например, вручную изменить значение операнда, заданного в команде, и без перекомпиляции выполнить модифицированную команду.

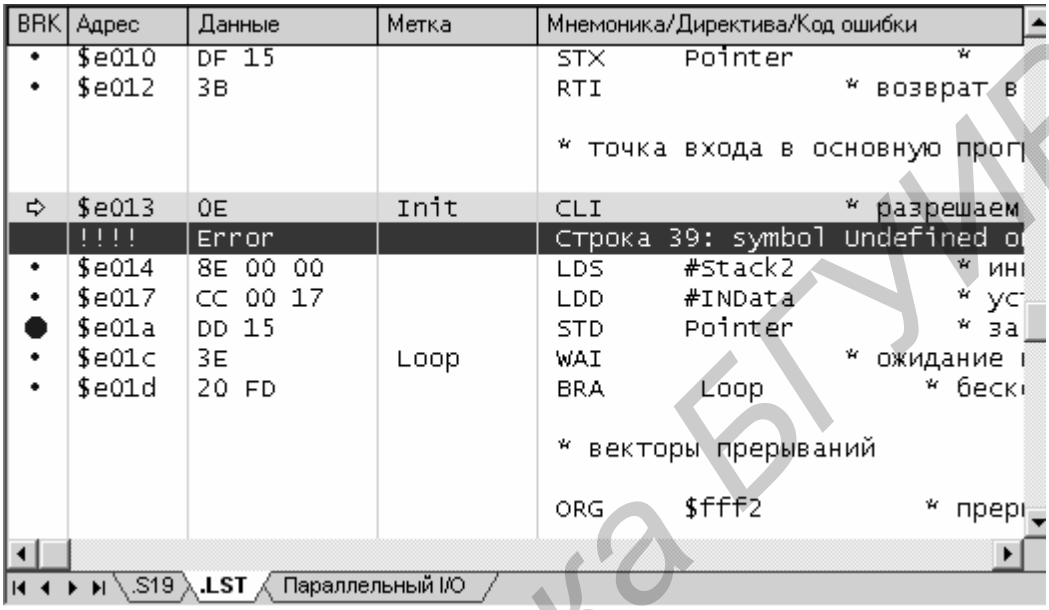

Рис. 3. Окно листинга с ошибками программы

В окне листинга отображается следующая информация:

BRK – точки останова, текущая команда и строки, на которые можно установить точку останова;

Адрес – адрес, которому соответствует строка листинга;

Данные – непосредственные данные или машинный код команды в памяти в виде последовательности байтов;

Метка – метка, сопоставленная данной строке;

Мнемоника/ Директива/ Код ошибки – мнемоническое обозначение команды, директивы ассемблера либо строка с описанием ошибки (предупреждения), выделенная цветом.

Желтая стрелка в колонке «BRK» указывает на строку, соответствующую адресу в счетчике команд (регистр PC), т. е. на следующую выполняемую команду.

Точку останова можно установить на любой строке, отмеченной в колонке «BRK» маленькой синей точкой, щелкнув мышью по синей точке либо выделив строку левой кнопкой мыши (появится синяя рамка вокруг строки) и нажав кнопку «Установить/убрать точку останова» на панели инструментов (клавиша ПРОБЕЛ или F4). При этом в колонке «BRK» синяя точка будет заменена

на красный кружок, указывая, что точка останова установлена на строку. Удаление точки останова производится тем же образом, что и установка, т. е. нажатием на красный кружок мышью либо выделением строки и нажатием кнопки «Установить/убрать точку останова». Удаление сразу всех точек останова производится нажатием кнопки «Убрать все точки останова»  (клавиша CTRL+F4).

Колонки в окне объектного кода (см. рис. 2) имеют тот же смысл, что и в окне листинга, с той разницей, что в них отсутствуют директивы ассемблера, символические имена и метки, а также непосредственно задаваемые данные.

Принцип работы с точками останова аналогичен принципу работы с ними в окне листинга. Необходимо отметить, что установка или снятие точки останова в окне дизассемблирования приведет к автоматической установке/снятию точки останова в окне листинга, и наоборот, т. е. как отмечалось выше, можно вести отладку и по окну листинга, и по окну дизассемблирования, переключаясь между ними по мере необходимости.

Границы, по которым расположена программа в памяти (и по которым проводится дизассемблирование), задаются с помощью вектора прерываний по адресу $FFFE либо могут быть изменены вручную, нажатием на ссылку «Диапазон дизассемблирования» и вводом в появившихся текстовых полях стартового и конечного адресов диапазона дизассемблирования.

Внесение изменений в исходный текст программы в окне «.ASM» и последующее переключение на окно листинга (или дизассемблирования) приведет к появлению сообщения о том, что исходный текст был изменен, для учета этих изменений нужно перекомпилировать файл исходного текста с дальнейшим вопросом: перекомпилировать его прямо сейчас или нет? В случае отрицательного ответа отладка программы будет продолжаться без учета изменений в исходном тексте. Если один или более байт в области программы в ходе отладки был изменен (вручную или в ходе выполнения программы), симулятор сообщит об этом и проведет повторное дизассемблирование программы. При этом не будет проведено никакой перекомпиляции, т. е. изменений в окне листинга не будет. Данная возможность необходима для проектирования программы с возможностью внесения изменений в нее прямо в процессе отладки без перекомпиляции (например, если необходимо изменить операнд, непосредственно заданный в команде).

Альтернативным вариантом получения объектного кода и листинга является прямая загрузка файла с расширением `.s19` с диска, используя соответствующую команду меню или кнопку на панели инструментов. Результат в окне 4 (см. рис. 1) будет тем же, что и при компиляции – симулятор загрузит объектный файл, дизассемблирует его и отобразит в окне 4. Кроме того, если в той же папке, где находится файл `.s19`, есть файл листинга (`.lst`) с тем же именем, что у файла `.s19`, то будет загружен и листинг. В этом случае отладку можно будет вести как по окну дизассемблирования, так и по окну листинга.

Скомпилировав исходный текст и получив объектный файл, можно приступать к отладке программы. Запуск программы на выполнение производится в трех режимах: *автоматическом*, *пошаговом* и *с точками останова*. Запуск в пошаговом режиме осуществляется нажатием на кнопку «Выполнить одну команду»  (клавиша F8). После выполнения текущей команды желтая стрелка-указатель в колонке BRK (см. рис. 3) переместится к следующей команде (в случае перехода – к команде, на которую был сделан переход), и выполнение программы снова становится возможным в любом режиме. При выполнении последней команды из списка дизассемблированных команд или при переходе на адрес, лежащий за пределами дизассемблированной программы, симулятор выдаст сообщение «Выполнение программы завершено».

В случае запуска программы в автоматическом режиме команды выполняются последовательно одна за другой с соответствующим перемещением желтой стрелки-указателя. Если это необходимо, отладка может производиться и без визуального перемещения курсора по строкам программы. Выключить/включить данную опцию можно с помощью команды «Режим визуализации выполнения программы»  . В случае выхода за пределы программы симулятор сообщит об этом, предложит перейти в начало программы и остановит ее выполнение. Запуск программы в автоматическом режиме, начиная с текущей команды, осуществляется нажатием кнопки «Выполнять программу далее»  (клавиша F9). Пауза (остановка) выполнения программы возможна с помощью кнопки «Приостановить выполнение программы»  (клавиша F10). При этом выполнение программы останавливается, управление передается пользователю и становится возможным дальнейший запуск программы в любом режиме. Переход на начало программы (рестарт программы) можно осуществить вручную с помощью кнопки «Перезапуск программы»  (клавиша F11).

Режим выполнения программы с точками останова ничем не отличается от автоматического режима, описанного выше, но если на текущей команде установлена точка останова, то выполнение программы приостанавливается и управление передается пользователю.

В случае когда необходимо перейти к текущей команде (например, окно было прокручено вниз или вверх и текущей строки не видно), существует кнопка «Показать текущую строку программы»  (клавиша F5).

Для проверки и изменения текущих значений регистров микроконтроллера используется окно регистров 2 (см. рис. 1), состоящее из трех дочерних окон, переход к которым осуществляется открытием соответствующей вкладки. Для каждого регистра приводится его имя (отображено синим цветом), текущее хранимое значение (черным цветом), адрес отображения на память (зеленым цветом в квадратных скобках) и значение каждого разряда с именем этого разряда (ниже имени регистра, в два столбика сверху вниз). Возле каждого разряда приводится его номер в круглых скобках (0 – для младшего разряда; 7 или 15 – для старшего). Если открыта вкладка «Регистры CPU», то для каждого регистра не приводится адрес отображения на память; кроме того, регистр флагов FLAGS

(другое имя – CCR) разбит на именованные разряды (флаги). Для записи любого значения в регистр (или в отдельный разряд регистра) достаточно нажать один раз курсором мыши на имени или значении регистра. В появившемся диалоговом окне (рис. 4) отображается описание того значения, которое подлежит изменению, предыдущего и нового значений.

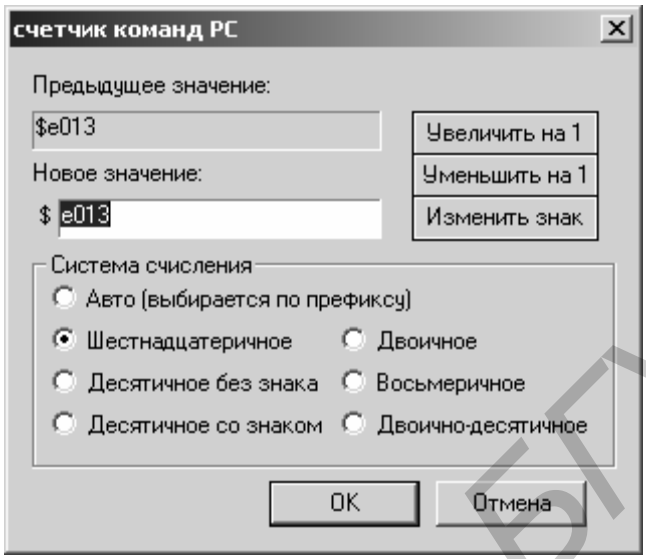

Рис. 4. Окно для изменения значений

После того как содержимое регистра изменилось в ходе выполнения программы или было модифицировано вручную, его значение отображается красным цветом до тех пор, пока не будет выполнена следующая команда.

Панель работы с памятью 3 (см. рис. 1) содержит таблицу со значением каждого байта памяти в шестнадцатеричной системе счисления, вид которой выбирается из выпадающего списка. Для удобства и наглядности каждая строка содержит 16 байтов. Слева от каждой строки зеленым цветом отображается адрес первого байта строки и над каждым столбцом байтов показана последняя значащая цифра адреса. Для перехода к любому адресу памяти можно использовать полосу прокрутки, а для точного перехода – нажать на ссылку «Адрес…» и ввести адрес в текстовое поле. Модификация любой ячейки памяти производится щелчком мыши по изображению ячейки и вводом нового значения. В случае изменения значения ячейки памяти оно изображается красным цветом, пока не будет выполнена следующая команда пользовательской программы.

Сервисные функции эмулятора – это калькуляторы (стандартный калькулятор Windows и шестнадцатеричный калькулятор), шестнадцатеричный редактор файлов, программы Блокнот и Проводник. Эти функции доступны в любой момент и выведены на панель инструментов.

3. ПРИМЕРЫ ПРОГРАММ ДЛЯ МИКРОКОНТРОЛЛЕРА MC68HC11

Задача 1. Сложение *N*-байтных двоичных чисел

```
*****************************************************
* Программа суммирует два N-байтных (N <= 16) числа,
* расположенных в памяти по адресам $0040 и $0050,
* и сохраняет результат в памяти по адресу $0060
* Количество N байт числа задано по адресу $003f
*****************************************************
          ORG  $003f
N         FCB  5
          ORG  $0040
x1        FCB  $11,$22,$33,$44,$55
          ORG  $0050
          FCB  $66,$77,$88,$99,$AA
          ORG  $0060
          FCB  0,0,0,0,0

          ORG  $E000
START:    ldx  #x1
          ldab N
          abx
          dex             *в X адрес младшего байта 1-го числа
          clc             *очищаем флаг переноса C
*суммируем числа в цикле побайтно
LOOP:     ldaa 0,X        *в аккумулятор A байт 1-го числа
          adca 16,X       *прибавляем к аккумулятору A
                          *соответствующий байт 2-го числа
          staa 32,X
          dex             *в индексный регистр X адрес следующего
                          *байта 1-го числа
          decb
          bne  LOOP       *зацикливаем

          ORG  $FFFE
          FDB  START
```

Задача 2. Сложение десятичных чисел

```
*****************************************************
* Программа суммирует два десятичных числа,
* состоящих из 6 десятичных цифр
* и расположенных в памяти по адресам $0040 и $0043,
* и сохраняет результат в памяти по адресу $0046
*****************************************************
          ORG  $0040
x1        FCB  $11,$29,$33
x2        FCB  $66,$29,$99
sum       FCB  0,0,0,0,0

          ORG  $E000
```

```
START:    ldaa x1+2       *в аккумулятор A две младшие цифры
                          *1-го числа
          adda x2+2       *прибавляем к аккумулятору A две младшие
                          * цифры 2-го числа
          daa             *десятичная коррекция
          staa sum+2      *сохраняем две младшие цифры суммы
          ldaa x1+1       *в аккумулятор A две средние цифры
                          *1-го числа
          adca x2+1       *прибавляем с переносом к аккумулятору A
                          *две средние цифры 2-го числа
          daa             *десятичная коррекция
          staa sum+1      *сохраняем две средние цифры суммы
          ldaa x1         *в аккумулятор A две старшие цифры
                          *1-го числа
          adca x2         *прибавляем с переносом к аккумулятору A
                          *две старшие цифры 2-го числа
          daa             *десятичная коррекция
          staa sum        *сохраняем две старшие цифры суммы

          ORG  $FFFE
          FDB  START
```

Задача 3. Умножение двоичных чисел

```
*******************************************************
* Программа умножения целых двоичных чисел
* без знака формата 16×8=24,
* расположенных в памяти по адресам $0040 и $0042.
* Результат сохраняется в памяти по адресу $0043
*******************************************************
DataAddr  EQU  $0040
CodeAddr  EQU  $e000

          ORG  DataAddr
x1        fcb  $11,$22
x2        fcb  $aa
y1        bsz  3

          ORG  CodeAddr
Start:    clr  y1
          ldaa x1+1
          ldab x2
          mul
          std  y1+1
          ldaa x1
          ldab x2
          mul
          addd y1
          std  y1

          ORG  $fffe
          FDB  Start
```

Задача 4. Преобразование целых десятичных чисел в двоичные

```
*****************************************************
* Программа преобразования двухразрядного целого
* десятичного числа без знака, представленного
* в двоично-десятичном коде, в эквивалентное
* однобайтное двоичное число
*****************************************************
DataAddr  EQU  $0040
CodeAddr  EQU  $e000

          ORG  DataAddr
x1        fcb  $25
y1        bsz  1

          ORG  CodeAddr
Start:    ldaa x1
          lsra
          lsra
          lsra
          lsra
          ldab #10
          mul
          ldaa x1
          anda #$0f
          aba
          staa y1

          ORG  $fffe
          FDB  Start
```

Задача 5. Преобразование целых двоичных чисел в десятичные

```
*****************************************************
* Программа преобразования однобайтного целого
* двоичного числа без знака в эквивалентное
* двоично-десятичное число
*****************************************************
DataAddr  EQU  $0040
CodeAddr  EQU  $e000

          ORG  DataAddr
x1        fcb  $6f
y1        bsz  2

          ORG  CodeAddr
Start:    clra
          ldab x1
          ldx  #10
          idiv
          stab y1+1
          xgdx
```

```
          ldx  #10
          idiv
          lslb
          lslb
          lslb
          lslb
          addb y1+1
          stab y1+1
          xgdx
          stab y1

          ORG  $fffe
          FDB  Start
```

Задача 6. Преобразование дробных десятичных чисел в двоичные

```
*****************************************************
* Программа преобразования четырехразрядной
* десятичной дроби без знака, представленной
* в двоично-десятичном коде, в эквивалентную
* двухбайтную двоичную дробь
*****************************************************
DataAddr  EQU  $0040
CodeAddr  EQU  $e000

          ORG  DataAddr
x1        fdb  $5678
Rez       bsz  2

          ORG  CodeAddr
Start:    clr  Rez
          ldaa x1+1
          anda #$0f
          staa Rez+1
          ldaa x1+1
          lsra
          lsra
          lsra
          lsra
          ldab #10
          mul
          addd Rez
          std  Rez
          ldaa x1
          anda #$0f
          ldab #100
          mul
          addd Rez
          std  Rez
          ldaa x1
          lsra
          lsra
```

```
        lsra
        lsra
        ldab #250       *чтобы умножить на 1000,
        mul             *необходимо сперва умножить на 250,
        lsld            *а затем дважды сдвинуть результат
        lsld            *влево (каждый сдвиг эквивалентен
                        *умножению на 2)
        addd Rez
        ldx  #10000
        fdiv
        stx  Rez

        ORG  $fffe
        FDB  Start
```

Задача 7. Преобразование дробных двоичных чисел в десятичные

```
*****************************************************
* Программа преобразования двухбайтной
* двоичной дроби без знака в эквивалентную
* четырехразрядную десятичную дробь
*****************************************************
DataAddr  EQU  $0040
CodeAddr  EQU  $e000
          ORG  DataAddr
x1        fcb  $a9,$87
y1        bsz  3
Rez       bsz  2

          ORG  CodeAddr
Start:    clr  y1
          ldaa x1+1       *умножение на 10
          ldab #10
          mul
          std  y1+1
          ldaa x1
          ldab #10
          mul
          addd y1
          std  y1
          ldaa y1         *выделение и запись 1-й цифры
          lsla
          lsla
          lsla
          lsla
          staa Rez
          ldd  y1+1       *подготовка к следующему шагу
          std  x1         *конец 1-го шаг
          clr  y1
          ldaa x1+1       *умножение на 10
          ldab #10
          mul
```

```
std  y1+1
ldaa x1
ldab #10
mul
addd y1
std  y1
ldaa y1             *выделение и запись 2-й цифры
adda Rez
staa Rez
ldd  y1+1           *подготовка к следующему шагу
std  x1             *конец 2-го шага
clr  y1
ldaa x1+1           *умножение на 10
ldab #10
mul
std  y1+1
ldaa x1
ldab #10
mul
addd y1
std  y1
ldaa y1             *выделение и запись 3-й цифры
lsla
lsla
lsla
lsla
staa Rez+1
ldd  y1+1           *подготовка к следующему шагу
std  x1             *конец 3-го шага
clr  y1
ldaa x1+1           *умножение на 10
ldab #10
mul
std  y1+1
ldaa x1
ldab #10
mul
addd y1
adda Rez+1          *выделение и запись 4-й цифры
staa Rez+1          *конец 4-го шага

ORG  $fffe
FDB  Start
```

4. ОПИСАНИЕ ЛАБОРАТОРНЫХ РАБОТ

ЛАБОРАТОРНАЯ РАБОТА №1
Программный эмулятор (симулятор) микроконтроллера MC68HC11

Цель работы: изучить назначение и структуру симулятора Sim68w микроконтроллера MC68HC11. Получить практические навыки по созданию и отладке программ на языке ассемблера для микроконтроллера MC68HC11.

1. ПРОГРАММЫ ДЛЯ ИЗУЧЕНИЯ РАБОТЫ С СИМУЛЯТОРОМ

Программа 1

```
*****************************************************
* Программа суммирует два однобайтных числа,
* расположенных в памяти по адресам $0040 и $0041,
* и сохраняет результат в памяти по адресу $0042
*****************************************************
          ORG  $0040
x1        FCB  $12
x2        FCB  $34
sum       FCB  0

          ORG  $E000
START:    ldaa x1
          adda x2
          staa sum

          ORG  $FFFE
          FDB  START
```

Программа 2

```
*****************************************************
* Программа суммирует два 16-разрядных числа,
* расположенных в памяти по адресам $0040 и $0042,
* и сохраняет результат в памяти по адресу $0044
*****************************************************
          ORG  $0040
x1        FDB  $1234
x2        FDB  $56f8
sum       FDB  0

          ORG  $E000
START:    ldd  x1
          addd x2
          std  sum

          ORG  $FFFE
          FDB  START
```

Программа 3

```
*****************************************************
* Программа суммирует два трехбайтных числа,
* расположенных в памяти по адресам $0040 и $0043,
* и сохраняет результат в памяти по адресу $0046
*****************************************************
          ORG  $0040
x1        FCB  $11,$22,$33
x2        FCB  $66,$77,$ff
sum       FCB  0,0,0,0,0

          ORG  $E000
START:    ldaa x1+2       *в аккумулятор A младший байт 1-го числа
          adda x2+2       *прибавляем к аккумулятору A младший байт
                          *2-го числа
          staa sum+2      *сохраняем младший байт суммы
          ldaa x1+1       *в аккумулятор A средний байт 1-го числа
          adca x2+1       *прибавляем с переносом к аккумулятору A
                          *средний байт 2-го числа
          staa sum+1      *сохраняем средний байт суммы
          ldaa x1         *в аккумулятор A старший байт 1-го числа
          adca x2         *прибавляем с переносом к аккумулятору A
                          *старший байт 2-го числа
          staa sum        *сохраняем старший байт суммы

          ORG  $FFFE
          FDB  START
```

2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Изучить порядок работы с симулятором Sim68w микроконтроллера MC68HC11.

2. Для каждой из приведенных выше программ:

– подготовить на языке ассемблера исходный текст программы;

– сохранить исходный текст программы в файле с расширением `.asm`;

– выполнить компиляцию исходного текста программы, исправить обнаруженные ошибки и получить два выходных файла: файл объектного кода с расширением `.s19` и файл листинга с расширением `.lst`;

– выполнить отладку программы в трех режимах: пошаговом, с точками останова и автоматическом, контролируя на каждом шаге (точке останова) изменение состояния регистров микроконтроллера и ячеек памяти.

– по результатам отладки внести необходимые изменения в программу.

3. Оформить отчет.

3. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы.

2. Краткие сведения о структуре и порядке работы с симулятором Sim68w микроконтроллера MC68HC11.

3. Листинги программ.
4. Результаты отладки программ.
5. Выводы по работе.

ЛАБОРАТОРНАЯ РАБОТА №2
Программирование арифметических операций с фиксированной запятой

Цель работы: получить практические навыки программирования арифметических выражений с фиксированной запятой для микроконтроллера MC68HC11.

1. КОМАНДЫ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ МИКРОКОНТРОЛЛЕРА MC68HC11

Команды арифметических операций в большинстве случаев выполняют действия над операндами, один из которых (M) располагается в памяти, а второй – в аккумуляторе A, B или D, куда помещается затем результат. При операциях сложения и вычитания второй операнд (M) адресуется любым способом, кроме относительного. Команды ADDD, SUBD осуществляют сложение и вычитание 16-разрядных операндов. Один из них располагается в аккумуляторе D, а второй (M) выбирается из двух рядом расположенных ячеек памяти (в команде задается адрес старшего байта). Команда десятичной коррекции результата DAA выполняется после команд сложения ABA, ADDA, ADDB, ADCA, ADCB, если операндами служили упакованные двоично-десятичные числа.

Команда умножения MUL выполняется над 8-разрядными операндами без знака, расположенными в регистрах A и B. 16-разрядное произведение размещается в аккумуляторе D. Команда деления выполняется над двумя 16-разрядными операндами без знака, размещенными в регистрах D и X: регистр D содержит делимое, а регистр X – делитель. Частное от деления располагается в регистре X, остаток – в регистре D. В микроконтроллере имеется два типа команд деления. Команда целочисленного деления IDIV используется, когда делимое больше делителя. Если же делимое меньше делителя, то используется команда дробного деления FDIV, которая сдвигает делимое на 16 разрядов влево, а затем делит его на делитель. При этом в регистре X получается двоичное число, представляющее дробную часть результата. Выполняя последовательное деление остатка с помощью команды FDIV, можно получать дробную часть результата с требуемой точностью.

2. ОСОБЕННОСТИ ВЫПОЛНЕНИЯ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ НАД БЕЗЗНАКОВЫМИ ЧИСЛАМИ

При *сложении двоичных беззнаковых чисел* может возникнуть ошибка переполнения. Признаком такой ошибки является значение *флага переноса*, равное C = 1 после сложения слагаемых. При *вычитании беззнаковых чисел* разность может стать отрицательным числом, т. к. множество беззнаковых чисел не замкнуто относительно операции вычитания. Признаком этого нарушения

также является значение *флага переноса* C = 1 после вычитания вычитаемого из уменьшаемого вследствие заема из несуществующего старшего разряда. Кроме того, может возникнуть ситуация деления на 0. В этом случае частное устанавливается в $FFFF, а значение остатка не определено. Признаком этого нарушения также является значение *флага переноса* C = 1 после выполнения команды деления. Если возникают данные ситуации, то результат не может быть использован в дальнейших вычислениях. Поэтому такие ситуации должны фиксироваться как ошибки, например, путем установки в 1 определенных разрядов в соответствующей ячейке памяти.

3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. По заданию преподавателя составить исходный текст программы на языке ассемблера.
2. Сохранить исходный текст программы в файл с расширением `.asm`.
3. Выполнить компиляцию исходного текста программы, исправить обнаруженные ошибки и получить два выходных файла: файл объектного кода с расширением `.s19` и файл листинга с расширением `.lst`.
4. Выполнить отладку программы в трех режимах: пошаговом, с точками останова и автоматическом, контролируя на каждом шаге (точке останова) изменение состояния регистров микроконтроллера и ячеек памяти.
5. По результатам отладки внести необходимые изменения в программу.
6. Оформить отчет.

4. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы.
2. Краткие сведения по командам арифметических операций микроконтроллера MC68HC11 и особенностям выполнения арифметических операций над беззнаковыми числами.
3. Листинг программы.
4. Результаты отладки программы.
5. Выводы по работе.

ЛАБОРАТОРНАЯ РАБОТА №3
Программирование операций с массивами

Цель работы: получить практические навыки программирования операций с массивами для микроконтроллера MC68HC11.

1. ОСОБЕННОСТИ ВЫПОЛНЕНИЯ ОПЕРАЦИЙ С МАССИВАМИ

Массив является наиболее простой структурой данных. *Одномерный массив* представляет собой конечное множество простых данных одного типа – множество однородных элементов. Все элементы массива связаны отношением непосредственного следования, что позволяет их пронумеровать. Номер (ин-

декс) позволяет однозначно идентифицировать любой элемент массива. Массивы сохраняют свою структуру постоянной в течение всего времени существования. В силу этого их называют статическими структурами. Достоинствами этих структур являются смежность элементов и как следствие непрерывность области памяти, отводимой под структуры. Недостатком является постоянство отношений между элементами, которое задается при создании структуры и не подлежит изменению в процессе ее обработки.

Обработка массивов осуществляется с помощью программных циклов. Числом повторений цикла управляют счетчики, а указатели или индексы показывают, какой именно элемент данных обрабатывается при данном проходе цикла.

Циклическая программа состоит из четырех различных блоков:

1) блок задания начальных значений переменным, счетчикам и указателям данных. Указатели представляют собой адреса данных;

2) блок обработки, который фактически выполняет требуемые вычисления;

3) блок управления циклом, в котором изменяются значения счетчиков и указателей перед выполнением следующей операции, а также осуществляется проверка числа выполнений цикла;

4) заключительный блок выполняет запоминание результата.

Собственно суммирование выполняется в блоке 2. Однако наличие всех остальных блоков также существенно для обеспечения правильного выполнения этого блока. Первый и четвертый блоки выполняются только один раз, поэтому большая часть машинного времени затрачивается на выполнение второго и третьего блоков. Программа сможет выполняться существенно быстрее только в том случае, если программист сумеет уменьшить время работы второго и третьего блоков. Влияние первого и четвертого блоков на время выполнения программы незначительно.

Рассмотрим пример суммирования ряда чисел. Ряд чисел можно рассматривать как одномерный массив, поэтому задача состоит в том, чтобы осуществить суммирование элементов массива. Эти числа могут представлять собой совокупности входных сигналов определенного типа, находящихся под управлением системы; число изделий, проданных в некоторой торговой операции; число очков, набранных в игре, или число сообщений, принятых за некоторый интервал времени. Предположим, что длина суммируемого ряда (массива) хранится в ячейке $0041 и что сам ряд (массив) размещен начиная с ячейки $0042. Предположим также, что сумма не превышает 255 и для ее хранения достаточно одной 8-разрядной ячейки памяти с адресом $0040.

```
<1>  * Программа определяет сумму элементов масива.
<2>  * Массив расположен в памяти по адресу $0042.
<3>  * Длина массива задана в ячейке $0041.
<4>  * Результат сохраняется в памяти по адресу $0040.
<5>
<6>  * Объявляем адреса начала областей данных и программы
```

```
<7>  DataAddr  EQU  $0000     *адрес начала области данных
<8>  CodeAddr  EQU  $F000     *адрес начала области программы
<9>
<10> * Область данных
<11>           ORG  DataAddr+$0040
<12> Summa     FCB  0
<13> Length    FCB  3
<14> Massive   FCB  $35,$72,$1D
<15>
<16> * Область программы
<17>           ORG  CodeAddr
<18> START:    clra            *сумма = 0
<19>           ldab Length     *счетчик = длина массива
<20>           ldx  #Massive   *задание указателя начала массива
<21> SUMD:     adda 0,X        *сумма = сумма + тек. элемент массива
<22>           inx             *увеличиваем указатель массива на 1
<23>           decb            *счетчик = счетчик - 1
<24>           bne  SUMD       *переход к суммир. след. элемента
<25>           staa Summa      *сохраняем сумму элементов массива
<26>
<27> * Задаем стартовый адрес программы
<28>           ORG  $FFFE
<29>           FDB  START
```

Программа использует оба аккумулятора и индексный регистр X. В аккумуляторе A содержится сумма, в аккумуляторе B – счетчик, а в индексном регистре X – указатель массива, т. е. адрес того элемента данных, который сейчас прибавляется к сумме.

Команда `clra` (строка 18) обнуляет аккумулятор A.

Команда `ldab Length` (строка 19) загружает в аккумулятор B содержимое ячейки памяти с адресом $0041.

Команда `ldx #Massive` (строка 20) загружает в 16-разрядный индексный регистр X адрес первого элемента массива, т. е. значение $0042.

Команда `adda 0,X` (строка 21) осуществляет сложение содержимого ячейки памяти, адрес которой находится в индексном регистре X, и содержимого аккумулятора A.

Команда `inx` (строка 22) прибавляет единицу к содержимому 16-разрядного индексного регистра X. Эта команда увеличивает на единицу указатель массива так, что в нем оказывается следующий, больший на единицу адрес.

По команде `decb` (строка 23) из содержимого аккумулятора B вычитается единица. В аккумуляторе B содержится число повторений цикла, которое осталось выполнить.

Команда `bne SUMD` (строка 24) в случае равенства нулю признака Z вызывает передачу управления команде, помеченной меткой `SUMD`. В команде используется относительная адресация. Значение смещения, записанное в дополнительном коде длиной 8 битов, представляет собой расстояние от команды, на-

ходящейся непосредственно за командой перехода, до команды, которой передается управление. Команда `bne` приводит к выполнению следующих действий:

(PC) = SUMD, если признак Z = 0;

(PC) = (PC) + 2, если признак Z = 1.

Команда `staa Summa` (строка 25) сохраняет содержимое аккумулятора A в ячейке памяти с адресом $0040.

2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. По заданию преподавателя составить исходный текст программы на языке ассемблера.
2. Сохранить исходный текст программы в файл с расширением `.asm`.
3. Выполнить компиляцию исходного текста программы, исправить обнаруженные ошибки и получить два выходных файла: файл объектного кода с расширением `.s19` и файл листинга с расширением `.lst`.
4. Выполнить отладку программы в трех режимах: пошаговом, с точками останова и автоматическом, контролируя на каждом шаге (точке останова) изменение состояния регистров микроконтроллера и ячеек памяти.
5. По результатам отладки внести необходимые изменения в программу.
6. Оформить отчет.

3. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы.
2. Краткие сведения об особенностях выполнения операций с массивами.
3. Листинг программы.
4. Результаты отладки программы.
5. Выводы по работе.

ЛАБОРАТОРНАЯ РАБОТА №4
Программирование арифметических операций над числами с плавающей запятой

Цель работы: получить практические навыки программирования арифметических операций над числами с плавающей запятой для микроконтроллера MC68HC11.

1. ОСОБЕННОСТИ ВЫПОЛНЕНИЯ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ НАД ЧИСЛАМИ С ПЛАВАЮЩЕЙ ЗАПЯТОЙ

Расширение областей использования микропроцессоров требует существенного увеличения диапазона обрабатываемых данных. В этих условиях применение арифметики с фиксированной запятой приводит к резкому увеличению разрядности чисел, росту затрат памяти и времени выполнения программ, что снижает в целом производительность и экономичность микропроцессорных систем. Кроме того, с увеличением разрядности чисел с фиксированной запятой и

усложнением алгоритмов обработки возрастает неопределенность прогнозирования вычислений и затрудняется их предварительное масштабирование, призванное гарантировать корректность и требуемую точность результатов. Эти проблемы решает арифметика с плавающей запятой, поскольку она обеспечивает раздельное представление диапазона и точности чисел (мантиссы и порядка) и реализует их автоматическое масштабирование в процессе вычислений.

Существует большое разнообразие представления чисел с плавающей запятой, определяемое различными разрядностью и размещением порядка, мантиссы и их знаков относительно друг друга в формате числа. В данной лабораторной работе используется представление двоичных чисел с плавающей запятой со смещенным порядком и мантиссой в дополнительном или прямом коде. Числа имеют два формата: трехбайтный обычной точности и четырехбайтный повышенной точности.

При использовании трехбайтного формата обычной точности числа хранятся в памяти в виде последовательности трех байтов, размещенных в порядке возрастания адресов памяти. Первый байт в старшем разряде содержит знак мантиссы $S_М$ и смещенный порядок, а два остальных байта – старший и младший байты мантиссы. Смещение порядка равно $64_{10} = 40_{16}$.

7	6	5	4	3	2	1	0
$S_М$	Порядок + 40_{16}						
Старший байт мантиссы							
Младший байт мантиссы							

При использовании четырехбайтного формата повышенной точности числа хранятся в памяти в виде последовательности четырех байтов, размещенных в порядке возрастания адресов памяти. Первый байт содержит смещенный порядок, а три остальных байта – старший, средний и младший байты мантиссы. Смещение порядка равно $128_{10} = 80_{16}$. Знак мантиссы $S_М$ указывается в старшем байте мантиссы.

7	6	5	4	3	2	1	0
Порядок + 80_{16}							
$S_М$	Старший байт мантиссы						
Средний байт мантиссы							
Младший байт мантиссы							

Условимся нулевое число с плавающей запятой изображать в виде нулевой мантиссы и нулевого смещенного порядка.

Сложение чисел с плавающей запятой выполняется в три этапа:

1) выравнивание порядков;

2) алгебраическое сложение мантисс с равными порядками по правилам арифметики с фиксированной запятой;

3) нормализация мантиссы суммы.

Для *выравнивания порядков* находится больший порядок, определяется разность порядков Δp, затем мантисса числа с меньшим порядком сдвигается вправо на Δp разрядов.

При сложении мантисс возможно *переполнение мантиссы суммы* (денормализация мантиссы влево). В этом случае необходимо устранить переполнение путем сдвига мантиссы вправо на один двоичный разряд и соответствующим увеличением порядка суммы на единицу, т. е. выполнить *нормализацию мантиссы суммы вправо*.

При сложении близких по величине чисел с разными знаками мантисса суммы по абсолютному значению может стать меньше 0,5 (денормализация мантиссы вправо). В этом случае необходимо сдвигать мантиссу влево и соответственно уменьшать порядок на единицу до тех пор, пока мантисса не станет равной или большей 0,5, т. е. выполнить *нормализацию мантиссы суммы влево*.

При выполнении операции сложения чисел с плавающей запятой возможно появления четырех особых ситуаций, которые должны выявляться программными средствами. При выравнивании порядков возможен случай, когда разность порядков превышает разрядность значащей части мантиссы. При этом мантисса меньшего числа полностью выходит за рамки разрядной сетки, т. е. становится равной нулю. В этом случае результат равен большему слагаемому, и нет необходимости выполнять сложение. При сложении равных чисел с разными знаками мантисса суммы получается равной нулю. Такая мантисса не может быть нормализована, и факт ее появления используется для обнуления порядка суммы. При нормализации мантиссы суммы вправо возможно *переполнение порядка суммы* (смещенный порядок становится больше максимально возможного значения), а при нормализации влево – *исчезновение порядка суммы* (смещенный порядок становится меньше нуля). Появление таких ошибок свидетельствует о неправильном выборе формата с точки зрения представления диапазона чисел в конкретной задаче. Такие ошибки должны выявляться, и в случае их появления необходимо прекратить процесс вычислений.

Умножение чисел с плавающей запятой выполняется в три этапа:

1) определяется порядок произведения путем сложения порядков сомножителей;

2) находится мантисса произведения путем перемножения мантисс сомножителей по правилам арифметики с фиксированной запятой;

3) производится, если необходимо, нормализация мантиссы произведения влево.

Поскольку минимальные значения нормализованных мантисс сомножителей равны 0,5, минимальное значение мантиссы произведения составляет 0,25, и нормализация результата требует не более одного сдвига мантиссы влево. При сложении порядков сомножителей и нормализации мантиссы произведения возможны *переполнение* и *исчезновение порядка*, что необходимо выявлять программными средствами. При вычислении порядка произведения в случае смещенных порядков сомножителей сумма последних отличается от правильного смещенного порядка произведения на величину смещения. Поэтому после

сложения порядков сомножителей из полученной суммы необходимо вычесть величину смещения.

Деление чисел с плавающей запятой выполняется в три этапа:

1) определяется порядок частного путем вычитания порядка делителя из порядка делимого;

2) находится мантисса частного путем деления мантиссы делимого на мантиссу делителя по правилам арифметики с фиксированной запятой;

3) производится, если необходимо, нормализация мантиссы частного вправо.

При делении мантисса частного может получиться больше единицы, но меньше двойки, поэтому нормализация результата требует не более одного сдвига мантиссы вправо. Как и в случае умножения чисел при делении возможны *переполнение* и *исчезновение порядка частного*, что необходимо выявлять программными средствами. Кроме того, следует фиксировать условие деления на нулевой делитель. При вычислении порядка частного в случае смещенных порядков разность последних меньше правильного смещенного порядка частного на величину смещения. Поэтому после вычитания порядков к полученной разности необходимо добавить величину смещения.

2. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. По заданию преподавателя составить исходный текст программы на языке ассемблера.
2. Сохранить исходный текст программы в файл с расширением `.asm`.
3. Выполнить компиляцию исходного текста программы, исправить обнаруженные ошибки и получить два выходных файла: файл объектного кода с расширением `.s19` и файл листинга с расширением `.lst`.
4. Выполнить отладку программы в трех режимах: пошаговом, с точками останова и автоматическом, контролируя на каждом шаге (точке останова) изменение состояния регистров микроконтроллера и ячеек памяти.
5. По результатам отладки внести необходимые изменения в программу.
6. Оформить отчет.

3. СОДЕРЖАНИЕ ОТЧЕТА

1. Цель работы.
2. Краткие сведения об особенностях выполнения операций над числами с плавающей запятой.
3. Листинг программы.
4. Результаты отладки программы.
5. Выводы по работе.

ЛИТЕРАТУРА

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

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

3. Левенталь, Л. Введение в микропроцессоры / Л. Левенталь ; пер. с англ. – М. : Энергоатомиздат, 1983. – 464 с.

4. Motorola MC68HC11 Reference Manual. Rev 3. – Motorola, 1991.

5. Motorola Freeware PC-Compatible 8-Bit Cross Assemblers User's Manual. M68FCASS/AD1. – Motorola, 1990.

6. Motorola Freeware 8-Bit Cross Assemblers User's Manual [Электронный ресурс]. – 1989. – Режим доступа : http://www.ele.uri.edu/Courses/ele205/.

Св. план 2009, поз. 18

Учебное издание

ПРОЕКТИРОВАНИЕ ЭВС НА ОДНОКРИСТАЛЬНЫХ МИКРОКОНТРОЛЛЕРАХ

Методическое пособие
для студентов специальности 1-40 02 02
«Электронные вычислительные средства»
дневной формы обучения

В 3-х частях

Часть 3

ПРОГРАММИРОВАНИЕ 8-РАЗРЯДНЫХ МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА M68HC11

Качинский Михаил Вячеславович
Клюс Владимир Борисович
Давыдов Александр Борисович

Редактор Е. Н. Батурчик
Корректор Л. А. Шичко
Компьютерная верстка Е. С. Чайковская

Подписано в печать 20.05.2009. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 2,44.
Уч.-изд. л. 2,2. Тираж 100 экз. Заказ 668.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0494175 от 03.04.2009.
220013, Минск, П. Бровки, 6