

Учреждение образования
Белорусский государственный университет информатики и
радиоэлектроники

Кафедра РЭС

В.М. Логин, И.Н. Цырельчук

**Лабораторный практикум по курсу
«Микропроцессорные системы и их применение»**

**8-разрядные микроконтроллеры семейства M68HC11 фирмы
Motorola**

Минск 2006

1. Введение

Данный курс лабораторных работ предназначен для получения начальных практических навыков работы с микроконтроллерами семейства M68HC11 фирмы Motorola. Курс предполагается проводить с использованием симулятора-отладчика Micro-IDE фирмы-производителя BiPOM Electronics. Перед проведением курса необходимо ознакомиться с описанием микроконтроллеров семейства MC68HC11 и программы-отладчика Micro-IDE.

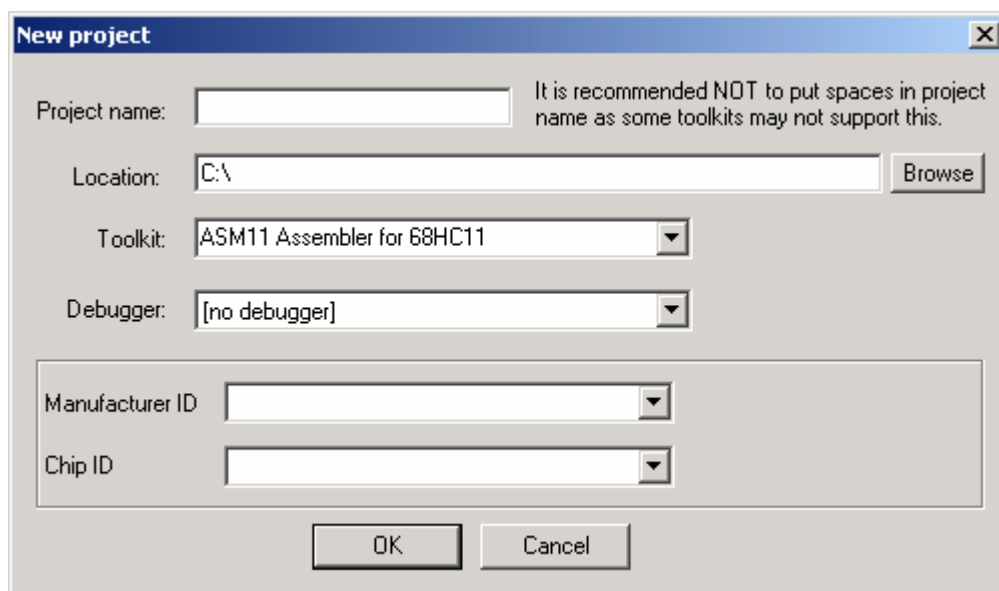
В последующих работах Вам будет необходимо произвести стандартную последовательность действий:

1. создание программы в редакторе;
2. ассемблирование программы и исправление ошибок;
3. запуск программы на выполнение;
4. отладка программы.

В следующей главе иллюстрируются основные приемы выполнения этих действий.

2. Запуск и начальная настройка среды

Загрузка среды осуществляется с запуска файла <ide.exe>. Для создания нового проекта Вам необходимо выбрать команду «New Project» в меню «Project». Вы увидите следующее диалоговое окно:



Вам необходимо будет заполнить следующие поля:

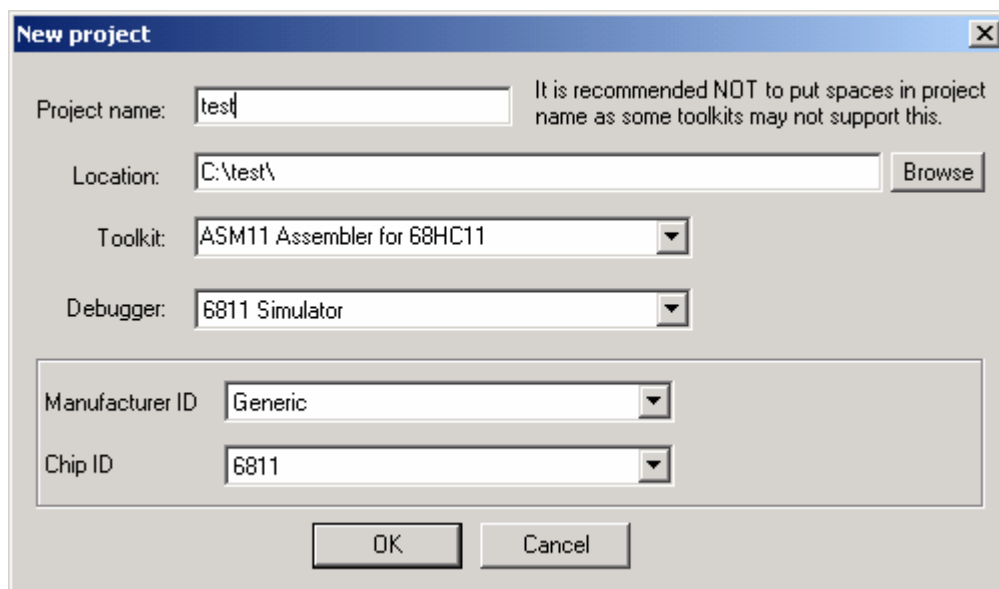
Project name – имя проекта (введите имя проекта);

Location – расположение (введите название папки где проект будет расположен с таким же именем, как и имя проекта);

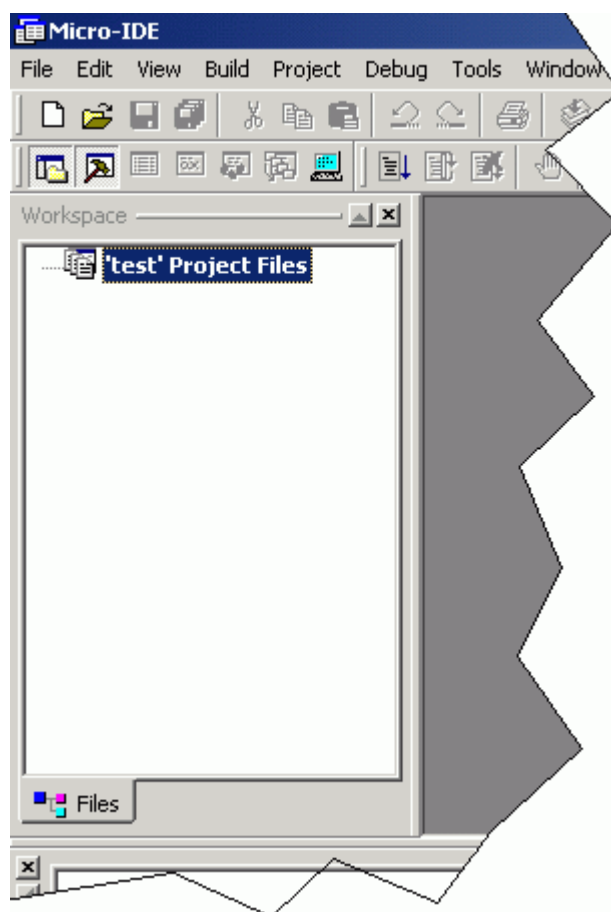
Toolkit – комплект инструментов (оставьте по умолчанию <SM11 Assembler for 68HC11>);

Debugger – отладчик (выберете <6811 Simulator>).

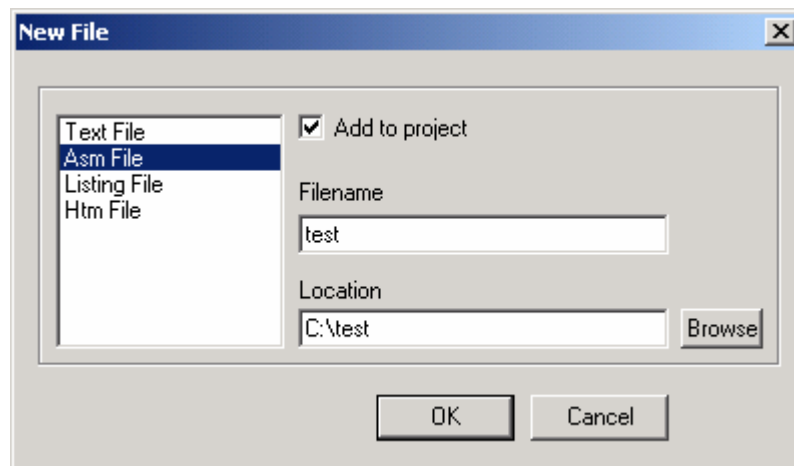
Остальные поля заполняться автоматически, после чего окно будет иметь следующий вид:



После того как Вы задали параметры настройки проекта нажмите <OK>. После этого будет создан новый пустой проект следующего вида:

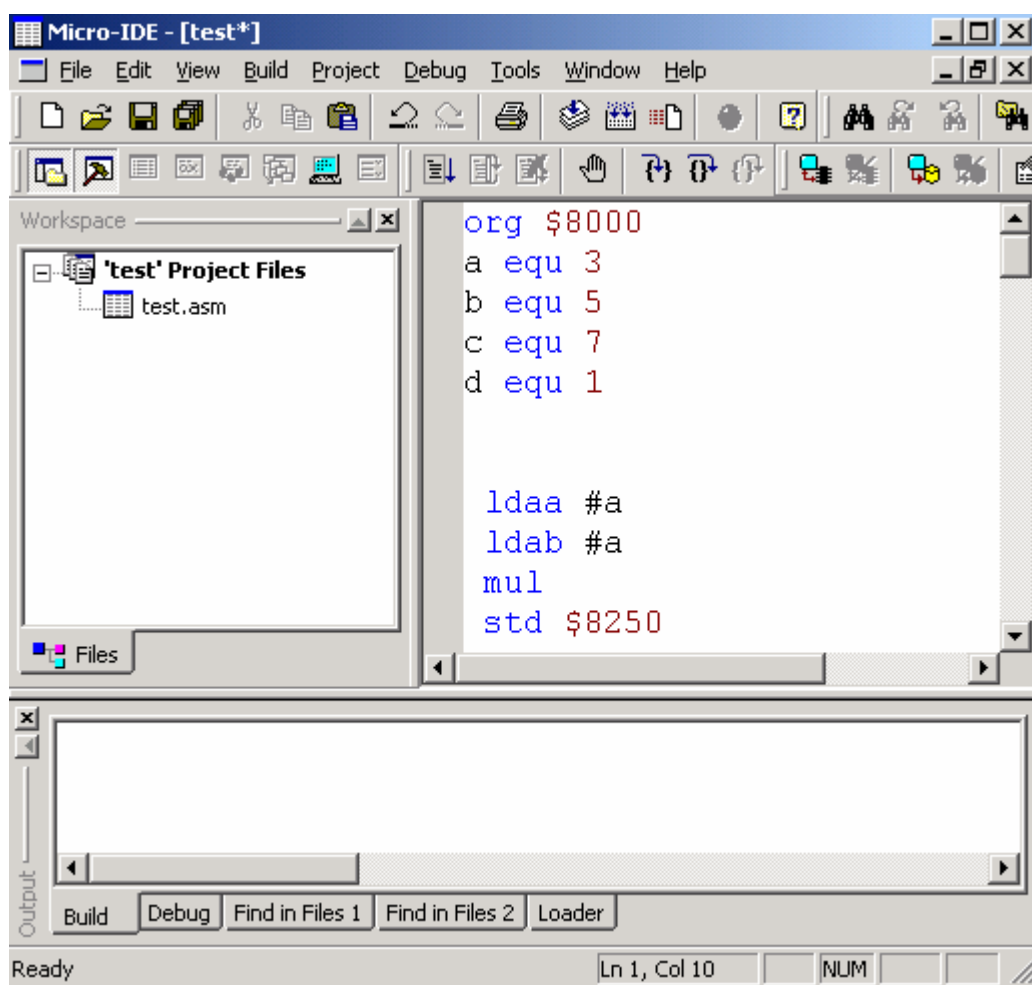


Теперь Вам необходимо добавить файлы к вашему проекту. Для этого необходимо выделить ваш проект в диалоговом окне “Workspace” и выбрать команду “New” в меню “File”. После чего появится диалоговое окно следующего вида:

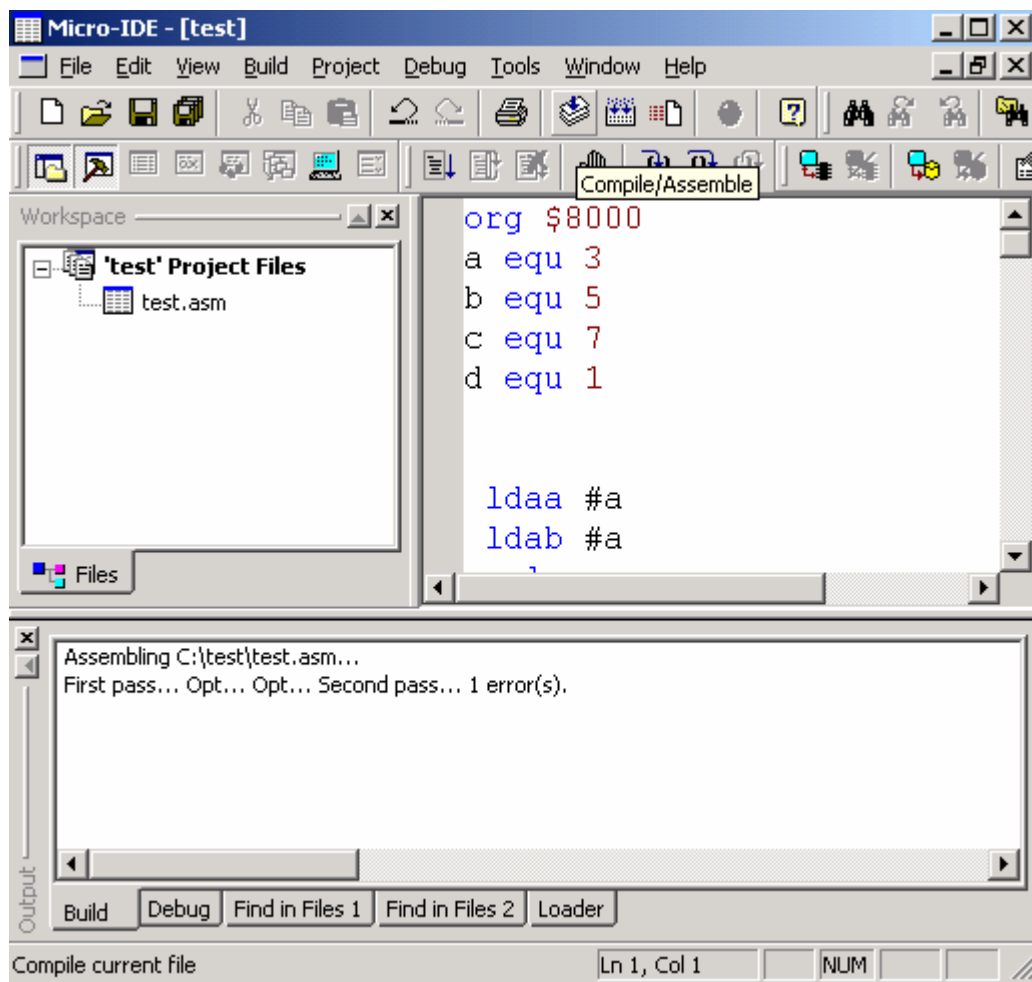


В данном окне Вам необходимо выбрать тип файла “Asm File” и задать имя файла (имя файла должно совпадать с именем вашего проекта). Далее нажмите <OK>.

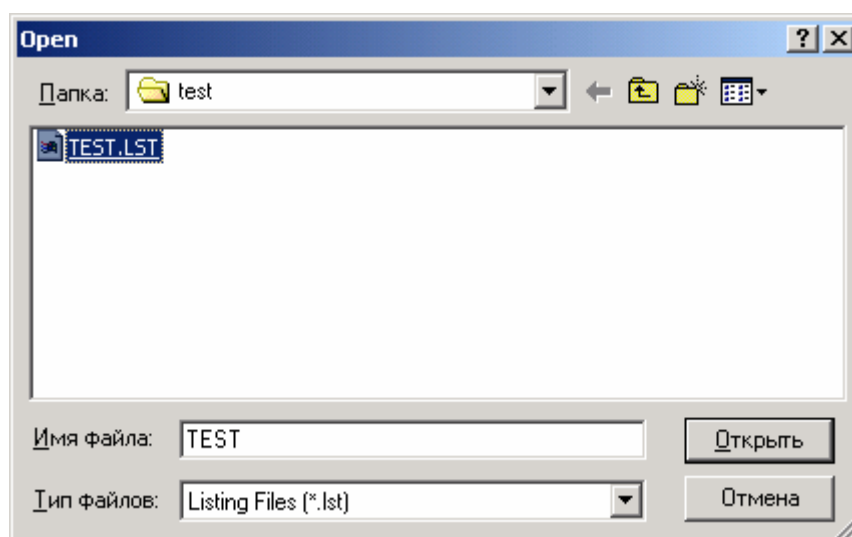
После этого у вас появится рабочая область созданного файла, в которой Вам необходимо будет написать код вашей программы, например:



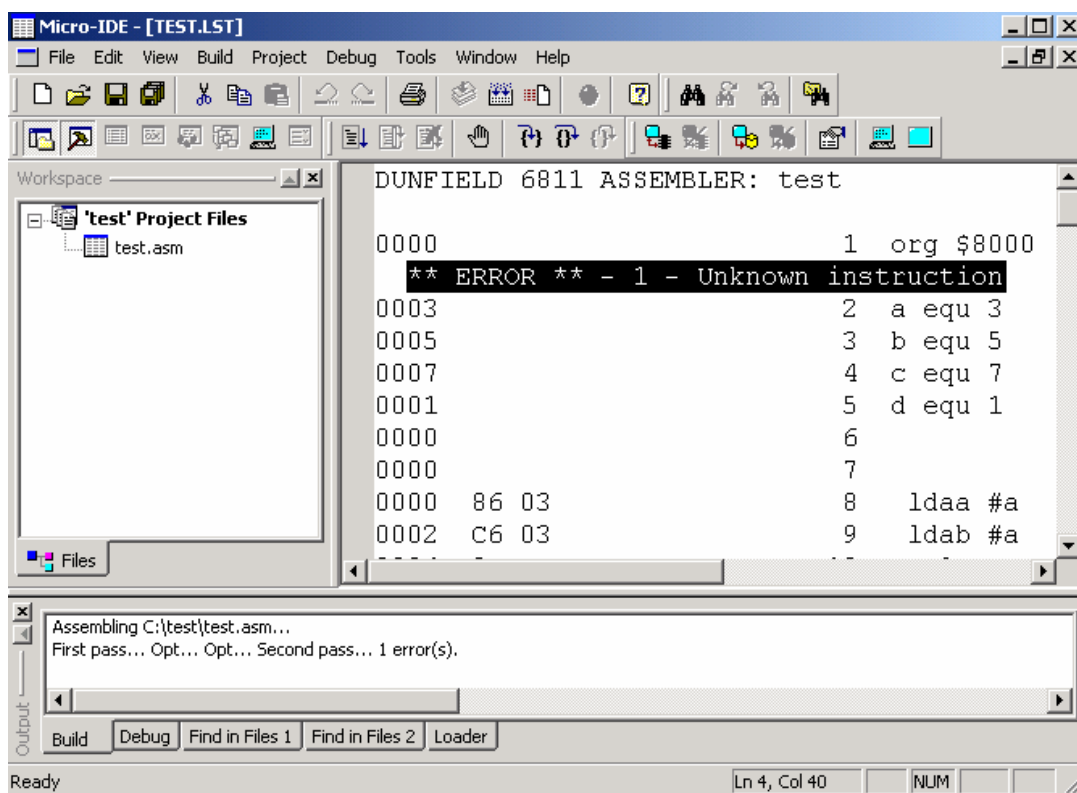
После того как код будет написан Вам необходимо откомпилировать ваш проект. Для этого выберите команду “Assemble” в меню “Build”. Если в проекте имеются ошибки, то в диалоговом окне “Output” будет выведена соответствующая информативная строка, к примеру:



Для того чтобы просмотреть, что за ошибки допущены, необходимо выбрать команду “Open” в меню “File” и выбрать файл с именем проекта и расширением <*.lst>, к примеру:



После этого откроется данный файл, в котором будут описаны все ошибки, допущенные в коде, к примеру:

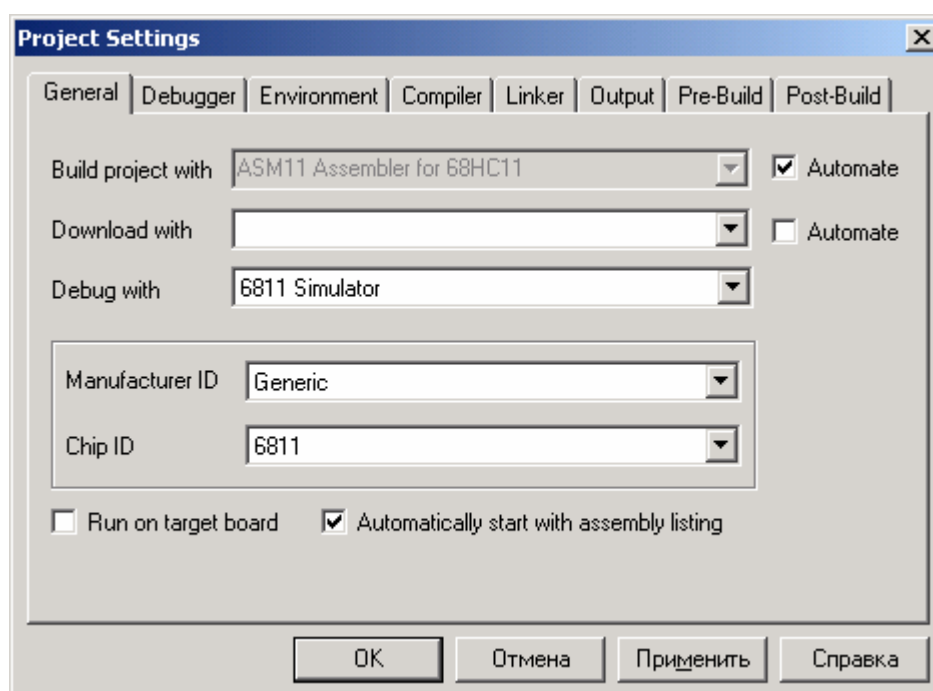


Далее необходимо закрыть файл с расширением <*.lst>, перейти к исходному коду, исправить все ошибки и снова откомпилировать проект.

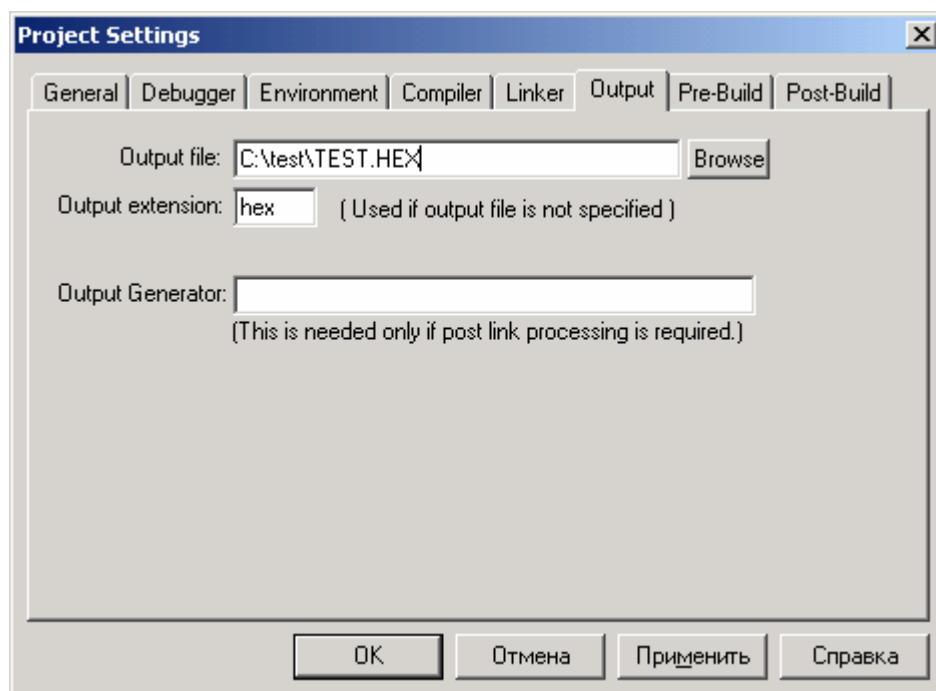
После того как все ошибки будут исправлены и проект будет сохранён необходимо запустить его на исполнение. Для этого выберите команду “Build ” в меню “Build”.

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

1. Выбрать команду “Settings” в меню “ Project” и в закладке “General” выставить флаг “Automatically start with assembly listing”, к примеру:

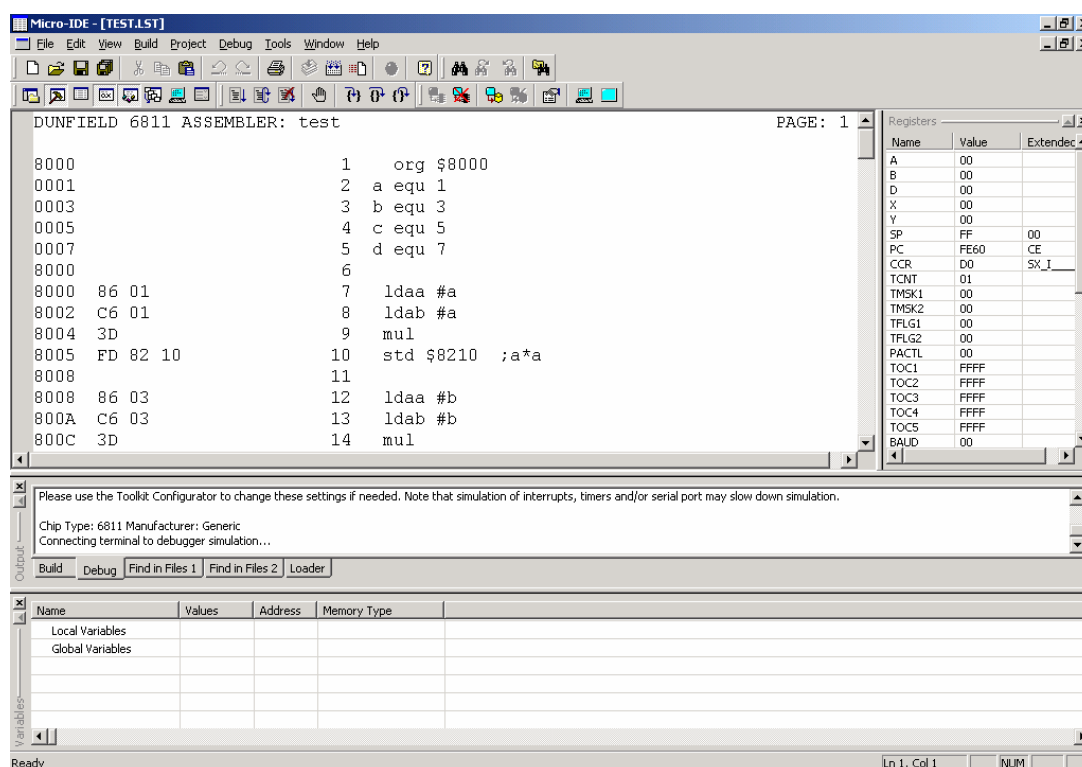


2. В закладке “Output” выбрать файл с именем проекта и расширением <*.hex>, к примеру:

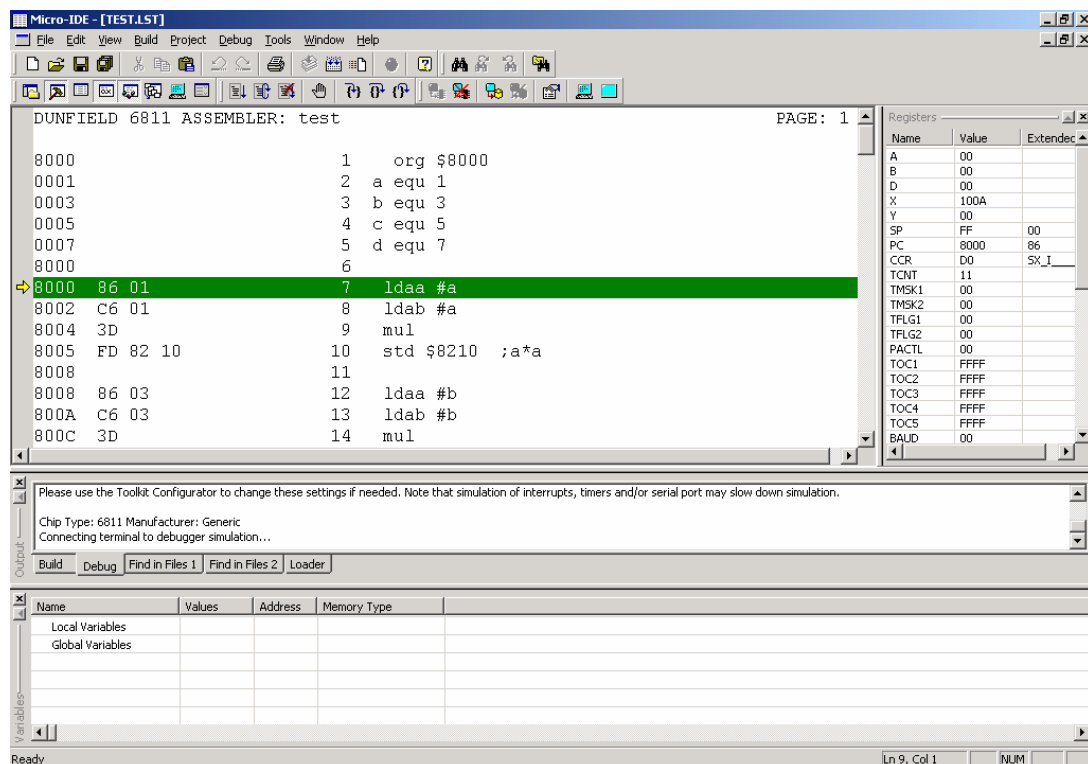


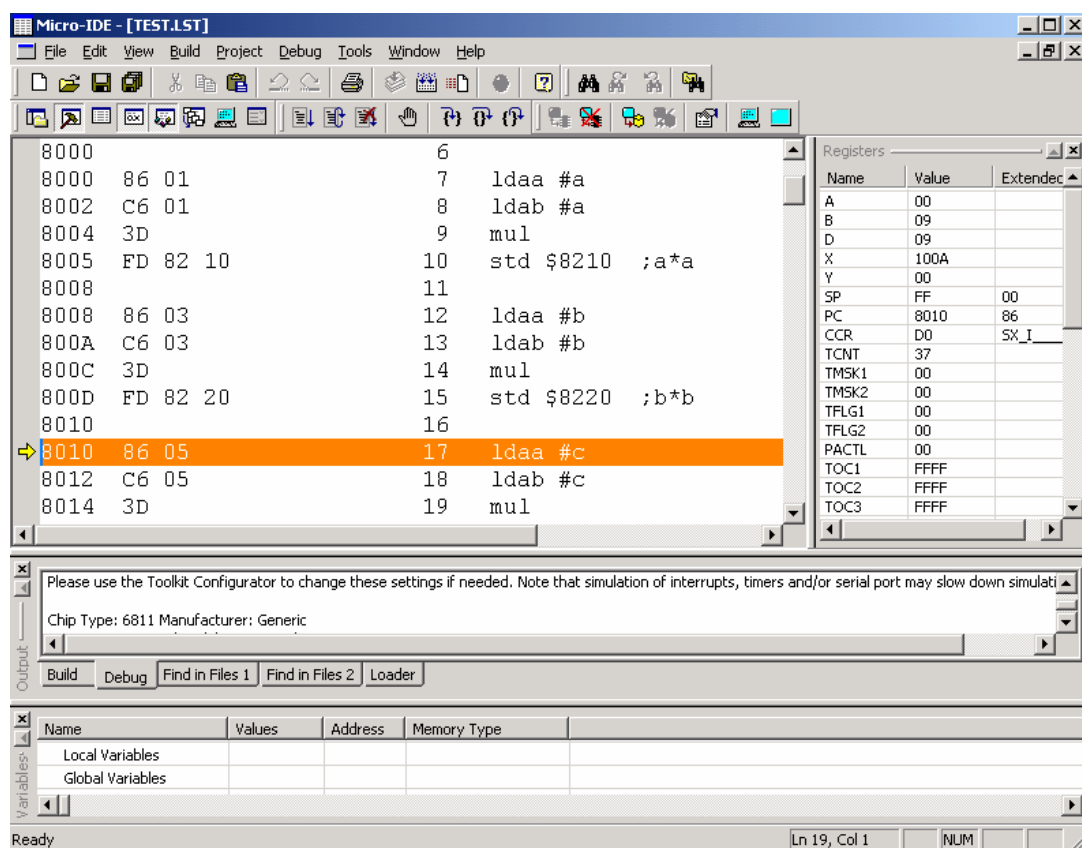
3. Далее нажмите <OK>.

После этого можно приступить к отладке программы. Для этого нажмите <F10> для появления окна отладчика, к примеру:

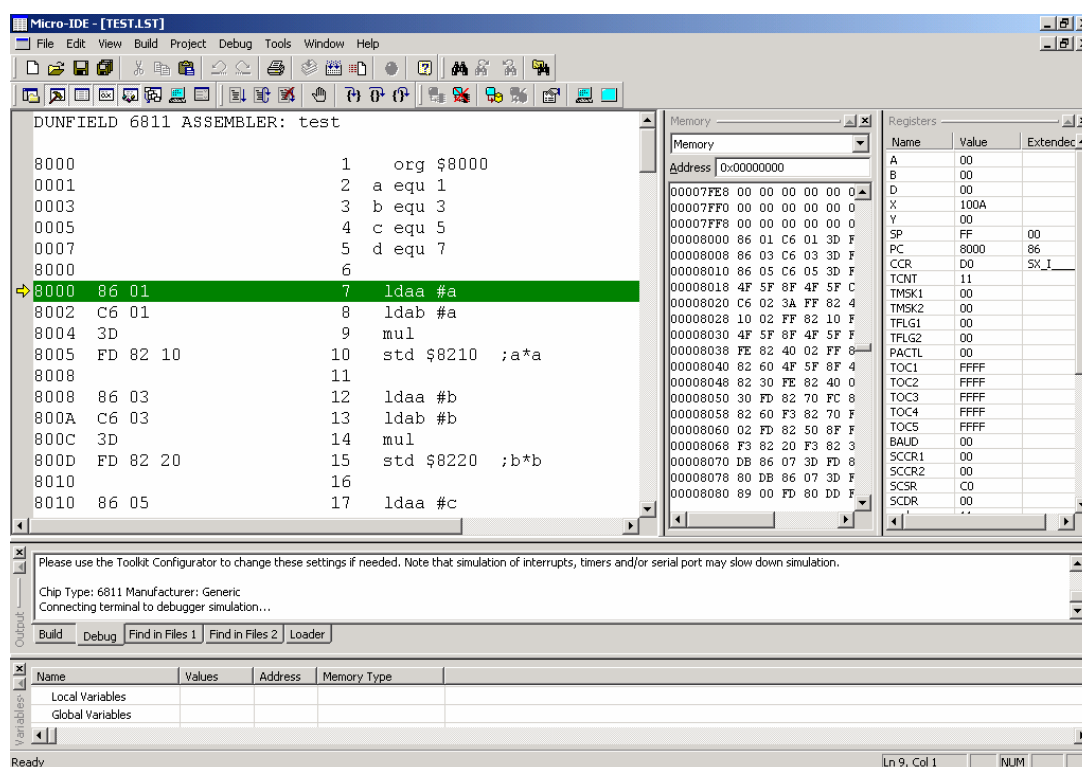


Жмите <F10> несколько раз подряд до появления зелёной полосы, к примеру:

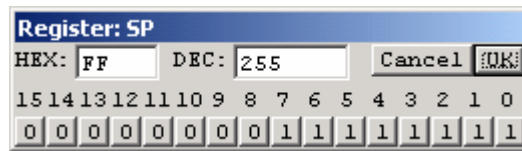




Для возможности просмотра состояния памяти необходимо выбрать команду “Memory” в меню “View”. При этом появится окно “Memory”, к примеру:



Значение регистров отображены в 16-тиричной системе счисления. Для просмотра значения в 10-тиричной системе счисления необходимо навести курсор на значение необходимого регистра и щёлкнуть правой клавишей мыши, к примеру:



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

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

3. Литература

В качестве учебно-методического пособия предлагается справочное пособие «Микропроцессоры и микроконтроллеры фирмы Motorola», автором которого является руководитель учебно-методического центра "Моторола-Микропроцессорные системы" МИФИ, д.т.н., профессор Шагурин Игорь Иванович.

Лабораторная работа №1

Методы адресации. Команды пересылки данных

1. Введение

В этой лабораторной работе изучаются:

- методы адресации,
- группа команд пересылки данных.

2. Методы адресации

Микроконтроллеры семейства M68HC11 имеют следующие типы адресации: неявная, непосредственная, прямая, расширенная, индексная и относительная.

Рассмотрим каждый из видов адресации подробнее.

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

43	COMA
53	COMB

Исключение составляют команды, взаимодействующие с регистром Y:

18	35	TYS
18	3A	ABY

В случае использования **непосредственной адресации** операнд (или один из операндов) включен непосредственно в код команды. Длина таких команд может составлять от двух до четырех байт. При записи команд, использующих непосредственную адресацию операнд предворяется символом "решетка" ('#').

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

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

96	3F	LDA		
63	DA	FF	GRAB	\$FF

Использование **расширенной адресации** позволяет осуществить доступ к любой ячейке памяти в пределах адресного пространства контроллера. При этом два байта,

следующие непосредственно за кодом команды, представляют собой абсолютный адрес операнда.

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

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

Для доступа к массивам данных удобно использовать *индексную адресацию*. В микроконтроллерах семейства M68HC11 используется так называемая *индексная адресация с 8-разрядным смещением*. При этом в индексный регистр X или Y заносится 16-разрядный адрес, а следующий за кодом команды байт содержит 8-разрядное смещение. Абсолютный адрес при этом вычисляется простым суммированием содержимого индексного регистра с байтом смещения.

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

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

32		PULA
37		PSHB

Эти команды используют *индексную адресацию без смещения*.

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

8D	00	BSR	*+\$2
24	FF	BCC	*-125

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

3. Команды пересылки данных

Простейшими командами являются команды пересылки данных. Список этих команд приведен в таблице 1.1. Рассмотрим каждую из команд подробнее на простых примерах.

Команды TSTA, TSTB и TST служат для установки регистра статуса в соответствии с содержимым регистра A, B или ячейки памяти соответственно.

Таблица 1.1. Команды пересылки данных.

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

Примечания:

* - Команды, использующие расширенную и индексную адресацию

** - Команды, использующие непосредственную, прямую, расширенную и индексную адресацию

*** - Команды, использующие прямую, расширенную и индексную адресацию

TSTA, TSTB, TST (opr)

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

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

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

Далее результат может быть использован в командах условного перехода. Занесите в регистр А значение \$00 и выполните в пошаговом режиме команду TSTA. Теперь посмотрите на содержимое регистра статуса: должен быть установлен флаг нуля (Z) и сброшены флаг отрицательного результата (M), переноса (C) и переполнения (V). Проведите подобный опыт при других значениях регистра А, обращая на различное состояние регистра статуса.

Рассмотрим команды загрузки в регистр содержимого ячейки памяти.

org \$8000

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

CLRA, CLRB, CLR (opr)

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

Работа команд очистки регистров А и В и ячейки памяти может быть проиллюстрирована на примере следующей простой программы:

org \$8000

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

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

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

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

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

TAB, TBA

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

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

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

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

Команда TPA осуществляет перенос содержимого регистра CCR в аккумулятор A. Это удобно, если после выполнения какой-либо подпрограммы необходимо сохранить состояние регистра статуса

(см. также TAP).

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

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

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

TAP

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

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

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

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

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

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

Алгоритм работы команд PSH таков:

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

1. в ячейку памяти, на которую указывает регистр SP записывается (младший) байт

регистра-операнда;

- значение регистра SP уменьшается на 1, указывая на следующую свободную ячейку в области стека;
- в случае двухбайтного операнда последовательность (1-2) повторяется со старшим байтом операнда.

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

Следующую программу, демонстрирует каким образом можно сохранить неизменными все внутренние регистры ОЭВМ (рекомендуется так же обратить внимание на содержимое стека):

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


pulx
pulb
pula

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

4. Контрольные вопросы

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

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

3. На какие флаги влияет выполнение команды TSTA?

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

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

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

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

7. Что такое позиционно-независимая программа?

8. Какие методы адресации используют приведенные ниже команды?

ldaa #20
staa \$20
psha
coma
pulb

9. Каково значение регистров X и D в результате выполнения программы:

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

10. Какие особенности имеет команда TAP?

11. Какое применение находит команда XGDX?

12. Каково значение регистра SP в результате выполнения фрагмента программы:

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

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

14. Какая логическая ошибка допущена при написании данного фрагмента программы:

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

15. Каково значение регистра Y в результате выполнения программы:

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

5. Задания

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

2. Перезаписать регистр A в регистр B таким образом, чтобы значение регистра флагов не изменилось.

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

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

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

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

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

8. Изменить порядок следования байт в регистре X, не используя команду XGDH.

9. Занести значение регистра стека в регистр D.

10. Изменить порядок следования байт в регистре Y, используя только неявную адресацию.

11. Сохранить текущее значение регистра стека в стеке.

12. Установить регистр флагов в соответствие с содержимым младшего

байта регистра SP.

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

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

Лабораторная работа №2

Арифметические команды

1. Введение

В данной лабораторной работе изучается работа арифметических команд:

- сложение;
- вычитание;
- умножение;
- деление;
- десятичная коррекция.

2. Арифметические команды

Список арифметических команд приведен в таблице 2.1. Приведем примеры использования этих команд в порядке увеличения сложности.

Таблица 2.1. Арифметические команды.

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

Примечания:

* - Команды, использующие непосредственную, прямую, расширенную и индексную адресацию

** - Команды, использующие расширенную и индексную адресацию

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

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

INS, DES

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

INX, INY, DEX, DEY

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

Команды инкремента и декремента являются простейшими арифметическими операциями и служат соответственно для увеличения и уменьшения на единицу значения регистра ОЭВМ или ячейки памяти.

В зависимости от типа операнда значение регистра статуса после выполнения команд может принимать различные значения. При работе с 8-разрядным операндом команды инкремента и декремента влияют на флаги отрицательного результата (N), нуля (Z) и переполнения (V). В случае если операндом является указатель стека, значение регистра статуса остается неизменным. При операциях с индексными регистрами команды инкремента и декремента влияют только на флаг нуля (Z).

Как правило, команды INC и DEC используются для организации циклов. Тот

факт, что эти команды не изменяют флаг переноса, используется при арифметических операциях над многобайтными числами.

Следующий простой пример иллюстрирует работу этих команд:

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

NEGA, NEGB,
NEG (opr)

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

Команда NEG замещает операнд его двоичным дополнением. Другими словами можно сказать, что результатом операции является изменение знака числа, представленного в дополнительном коде. Пр продемонстрируем на примере эмуляцию команды INC через NEG и DEC:

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

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

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

Команды сравнения используются при сравнении значения регистра со значением ячейки памяти или регистра. Фактически происходит операция вычитания ячейки памяти, указанной в качестве операнда, или регистра B (в случае команды CBA) из соответствующего регистра МК. Команды

не оказывают влияния на операнды, изменяется лишь регистр статуса. В дальнейшем результат обычно используется командами перехода.

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

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

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

При выполнении команд сложения происходит суммирование содержимого регистра-приемника с непосредственно заданным значением, ячейкой памяти или другим регистром. В командах ADC к

результату дополнительно прибавляется значение флага переноса. Результат сложения аккумуляторов командой ABA заносится в регистр A, результат сложения регистра B с индексным регистром - в соответствующий индексный регистр.

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

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

ABX, ABY

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

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

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

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

DAA

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

-значение не определено

Двоично-десятичная коррекция после сложения командами ABA, ADDA и ADCA обеспечивает суммирование двух чисел, представленных в двоично-десятичном формате. При этом флаг переноса используется в качестве старшего бита, обеспечивая

получение корректного двоично-десятичного значения.

Фактически, команда DAA после команд сложения действует следующим образом:

1. если содержимое младшей тетрады аккумулятора больше 9 или флаг полу-переноса H установлен в "1", то к аккумулятору добавляется число 6;
2. если содержимое старшей тетрады аккумулятора стало после этого более 9 или установлен флаг переноса, то число 6 добавляется и к старшей тетраде аккумулятора.

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

MUL

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

Команда умножения производит беззнаковое умножение двух чисел, представленных в восьмиразрядных аккумуляторах. Результат помещается в 16-разрядный аккумулятор D. Флаг

переноса при этом устанавливается таким образом, что при выполнении команды ADCA #0 происходит округление старшего байта.

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

IDIV

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

Команда IDIV производит целочисленное деление аккумулятора D на индексный регистр X. После выполнения в регистр X заносится частное, а в регистр D - остаток от деления. При выполнении команды IDIV делимое обычно больше делителя. Команда FDFV производит операцию дробного деления тех же аргументов. Фактически FDIV может быть представлен как умножение регистра D на 2^{16} с последующим выполнением команды IDIV, поэтому при выполнении

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

FDIV

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

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

3. Контрольные вопросы

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

4. Задания

1. Напишите программу суммирования двух 16-разрядных чисел, представленных в BCD формате, с учётом возможного переполнения.
2. Напишите программу суммирования регистров МК по следующей формуле $D = A + B + lo(X) + hi(X) + lo(Y) + hi(Y)$, где lo и hi соответственно младший и старший байты соответствующих регистров.
3. Напишите программу вычитания содержимого регистров X и Y из регистра D.
4. Напишите программу сравнения ячеек памяти \$0 и \$1. Регистр A должен быть равен единице, если ячейки памяти равны.
5. Вычислите произведение двух ячеек памяти. Содержимое всех регистров должно остаться неизменным.
6. Напишите программу, позволяющую вычислить адрес элемента, находящегося в двухмерном массиве размерностью 3x3. Массив располагается по адресу \$8100. Индекс задается регистрами A и B, где A – номер строки, B – номер столбца массива.
7. Напишите программу, которая преобразует число, заданное в регистре A, в восьмеричное представление этого числа в ASCII коде.
8. Напишите программу, которая преобразует число, заданное в регистре A, в десятичное представление этого числа в ASCII коде.
9. Просуммируйте содержимое двух ячеек памяти. Содержимое всех регистров должно остаться неизменным.

10. Вычислите разность содержимого регистров X и Y.

11. Вычислите произведение регистров X и Y.

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

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

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

Лабораторная работа №3

Логические команды. Команды работы с битовыми полями.

Команды сдвигов

1. Введение

В данной лабораторной работе изучается работа:

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

2. Логические команды

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

Таблица 3.1. Логические команды.

COMA	ANDA	BITA	ORAA	EORA
COMB	ANDB	BITB	ORAB	EORB
COM				

COMA, COMB,
COM (opr)

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

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

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

org \$8000

ldaa \$1f03

ldab \$1f04

andb #%01111110

oraa #%01111110

coma

eora #%10000000

aba

;
; считать состояние порта C
; считать состояние порта B
; выделить неизменную часть
; установить все неиспользуемые биты в "1"
; инвертировать значение аккумулятора
; инвертировать значение старшего бита
; совместить результат (заметьте, что мы
; заблаговременно маскировали неиспользуемые

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

```

; биты, чтобы сейчас совместить два числа простым
; суммированием)
staa $1f04 ; вывести результат

```

3. Команды работы с битовыми полями

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

Таблица 3.2. Команды работы с битовыми полями.

SEC	SEI	SEV	BSET*
CLC	CLI	CLV	BCLR*

Примечания:

* - Команды, использующие прямую или индексную адресацию в качестве первого параметра и непосредственную - в качестве второго.

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

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

```

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

```

В реализации отладчика имеется одна особенность - если трассируемая команда запрещает прерывания, то выполнение программы будет продолжаться до тех пор, пока либо прерывания не будут разрешены, либо не произойдет прерывания по неправильному коду команды. В частности это относится к командам SEI и SWI (эта команда будет рассмотрена в следующей лабораторной работе). Так же следует отметить, что если выполнение программы затянется, то отладчик выдаст сообщение об истечении времени ожидания ответа.

Почти в каждой программе требуется возможность манипуляции отдельными битами ячейки памяти. Так, блок регистров представляет собой по большей части битовые поля.

BCLR (opr), BSET (opr)

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

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

Приведем пример, демонстрирующий работу этих команд:

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

4. Команды сдвигов

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

Команды сдвигов обычно подразделяют на три группы:

- арифметические сдвиги,
- логические сдвиги,
- циклические сдвиги.

Таблица 3.3. Команды сдвигов.

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

Примечания:

* - Команды, использующие расширенную или индексную адресацию.

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

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

При арифметическом сдвиге происходит сохранение знака первоначального операнда при выполнении сдвига. При выполнении команды ASR происходит расширение знакового разряда. Это позволяет использовать команду для деления знакового числа на 2^N . Однако для нечетных чисел деление не

всегда является корректным (разница в результате может составлять 1). При выполнении команды арифметического сдвига влево всякий раз при смене знакового бита устанавливается флаг V, а освободившиеся разряды заполняются 0. Таким образом, становится возможным при помощи команд ASR производить знаковое умножение числа на 2^N .

Флаг переноса устанавливается в соответствии с отбрасываемым битом.

Приведем пример, демонстрирующий работу этих команд:

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

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

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

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

лишь отличие команд ASR от команд LSR:

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

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

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

Команды циклического сдвига позволяют осуществить операцию логического сдвига над многобайтными числами. Отличие этих операций от операций логического сдвига состоит в том, что освободившийся разряд заполняется не нулем, а

состоянием флага переноса C. Рассмотрим пример использования этих команд для сдвига 4-х байтного числа, расположенного в ячейках 0÷3, влево:

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

5. Контрольные вопросы

1. Каков результат выполнения программы:

```
sec
clra
adda #0
```

2. Какие методы адресации применимы к командам циклического сдвига?

3. Расскажите о командах работы с битами регистра CCR.

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

5. В чем разность команд ASR и LSR?

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

7. Какие логические команды Вы знаете?

8. Каким образом реализуется команда ASRD (сдвиг регистра D на 1 байт вправо)?
9. На какие группы можно подразделить команды сдвигов?
10. Дайте определение команд логического сдвига.
11. Чем отличается команда COM от команды NEG?
12. Каким образом можно съэмулировать команду COM, пользуясь командами, изученными в данной лабораторной работе?
13. Какие параметры имеет команда BSET?
14. Чем отличается команда ORAA от команды EORA?

6. Задания

1. Напишите программу, осуществляющую сдвиг влево 3-х ячеек памяти таким образом, чтобы выдвигаемый из старшей ячейки памяти бит становился на место младшего бита в младшей ячейке.
2. Произведите операцию "логическое ИЛИ" над регистрами X и Y.
3. Напишите программу, производящую обмен старшей и младшей тетрады аккумулятора A.
4. Напишите программу, создающую зеркальное отображение битовой карты регистра A в регистре B.
5. Реализовать подсчет установленных в регистре A битов с занесением суммы в регистр B.
6. Напишите программу умножения двух двоично-десятичных 8-разрядных чисел.
7. Произведите операцию "логическое И" над регистрами X и Y.
8. Написать тремя способами установку битов 2 и 3 в ячейке памяти \$10.
9. Написать программу, копирующую содержимое регистров A и B в регистр X таким образом, что старшая тетрада регистра A и старшая тетрада регистра B составляли старший байт регистра X, а младшие тетрады - младший.
10. Написать программу, позволяющую инвертировать те биты регистра A, которые сброшены в регистре B.
11. Написать программу, сбрасывающую биты в регистре A, если соответствующие биты регистров A и B установлены. Остальные биты должны оставаться в исходном состоянии.
12. Написать программу, которая в четные биты регистра X записывает биты регистра A, а в нечетные - регистра B.
13. Написать программу, копирующую регистр A в регистр B с обратным порядком следования бит, инвертируя нечетные биты.
14. Написать программу, заполняющую ячейки памяти \$0÷\$7 соответствующими битами регистра A. Т.е., например, если бит 0 в регистре A сброшен, то в ячейку \$0 записывается ноль, если установлен - \$FF.
15. Установить 4 и 5 биты в регистра A с помощью команды BSET.

Лабораторная работа №4

Команды передачи управления. Специальные команды

1. Введение

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

2. Команды передачи управления

Команды передачи управления можно разделить на 4 группы:

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

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

Таблица 4.1. Команды передачи управления.

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

Примечания:

* - Команды, использующие расширенную и индексную адресацию

** - Команды, использующие относительную адресацию

*** - Команды, использующие неявную адресацию

**** - Команды, использующие смешанную адресацию: первый операнд использует либо прямую, либо индексную адресацию, второй – относительную

***** - Команды, использующие прямую, расширенную и индексную адресацию

JMP, BRA, BRN, NOP

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

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

org \$8000

ldab #\$02

; выбор варианта ветвления программы

	ldx	#ways	;	занести в регистр X адрес таблицы переходов
p0	ldy	0,x	;	считать значение в регистр Y
	jmp	0,y	;	вызвать подпрограмму по адресу, находящемуся в регистре Y
			;	
p1	nop		;	задержка в 2 такта
	inx		;	увеличить регистр X на 2 для выборки следующего адреса из таблицы переходов
			;	
	bra	p0	;	перейти наметку p0
p2	bra	p1	;	перейти на метку p1
p3	brn	*	;	задержка в 3 такта
ways	fdb	p1,p2,p3		

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

JSR, BSR, RTS

Команды работы с подпрограммами позволяют

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

выделять часто используемую последовательность действий в подпрограмму. При переходе к подпрограмме (JSR, BSR) в стеке сохраняется адрес следующей за текущей команды и регистр PC изменяется по правилам

команд безусловного перехода. При выходе из подпрограммы по команде RTS происходит выборка из стека адреса возврата. Работу этих команд можно проследить на примере программы, размещающей адрес своей второй команды в ASCII формате в ячейках \$0÷3:

	org	\$8000		
	bsr	p1	;	переход на следующую команду
p1	pulx		;	получить в регистр X адрес p1 (\$8002)
	xgdx		;	расписать адрес в формате ASCII
	ldy	#0	;	в ячейках \$0÷\$3
	bsr	bin2ascii	;	вызвать подпрограмму, сохраняющую регистр
			;	A в формате ASCII в ячейках (y), (y+1)
	tba		;	переписать регистр B в регистр A
	iny		;	и расписать его в ячейках \$2, \$3
	iny			
	bsr	bin2ascii		
	bra	*		
bin2ascii	pshb		;	сохранить в стеке регистр B
	tab		;	скопировать в него регистр A
	lsra		;	выделить в A старшую тетраду
	lsra			
	lsra			
	bsr	hex2ascii	;	преобразовать число 0-f в ASCII код
	staa	0,y		


```

tba          ;    повторить для младшей тетрады
anda #$f
bsr  hex2ascii
staa  l,y
pulb
rts
hex2ascii    adda #0          ;    преобразование числа из диапазона
daa
adda #$f0
adca #$40
rts

```

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

Иногда команды условного перехода, выполняющие передачу управления в зависимости от состояния регистра статуса, подразделяют на три группы:

- знаковые,
- беззнаковые,
- простые (см. табл. 4.2).

Таблица 4.2. Команды условного перехода.

Условие	Логическая функция	Мнемоника	Противоположное действие		Тип
$r > m$	$Z + (N \oplus V) = 0$	BGT	$r \leq m$	BLE	знаковый
$r \geq m$	$N \odot V = 0$	BGE	$r < m$	BLT	- -
$r = m$	$Z = 1$	BEQ	$r \neq m$	BNE	- -
$r \leq m$	$Z + (N \odot V) = 1$	BLE	$r > m$	BGT	- -
$r < m$	$N \odot V = 1$	BLT	$r \geq m$	BGE	- -
$r > m$	$C + Z = 0$	BHI	$r \leq m$	BLS	беззнак.
$r \geq m$	$C = 0$	BCC/BHS	$r < m$	BCS/BLO	- -
$r = m$	$Z = 1$	BEQ	$r \neq m$	BNE	- -
$r \leq m$	$C + Z = 1$	BLS	$r > m$	BHI	- -
$r < m$	$C = 1$	BCS/BLO	$r \geq m$	BCC/BHS	- -
перенос	$C = 1$	BCS/BLO	нет пер.	BCC/BHS	простой
отрицат.	$N = 1$	BMI	полож.	BPL	- -
переп.	$V = 1$	BVS	нет пер.	BVC	- -
$r = 0$	$Z = 1$	BEQ	$r \neq 0$	BNE	- -

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

```

org  $8000
ldx  #$3          ;    конечный адрес первого числа
ldy  #$7          ;    конечный адрес второго числа
ldab #$4          ;    размер числа

```

	clc	;	сброс флага переноса
loop	ldaa 0,x	;	сложить два байта
	adca 0,y		
	staa 0,x		
	dex	;	перейти к следующему байту
	dey		
	decb	;	уменьшить число обрабатываемых байт
	bne loop	;	на 1 и повторить, если не 0

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

	org	\$8000	
	ldx	#0	;
	ldd	0,x	;
	tstb		;
	beq	invalid	;
	tsta		
	beq	invalid	
	cmpb	#\$13	
	bhs	invalid	
	cmpb	#\$2	;
	bne	lab2	;
	ldab	3,x	;
	bsr	bcd2bin	
	andb	\$03	;
	bne	lab2	;
	ldab	#\$29	
	bra	lab1	
lab2	bsr	bcd2bin	;
	decb		
	ldy	#dpm	;
	aby		
	ldab	0,y	;
lab1	cba		;
	bhi	invalid	;
valid	clc		;
	bra	done	;
			;
invalid	sec		;
done	bra	*	
bcd2bin	psha		;
	tba		;

```

anda  #$0F      ; формат
lsrb                      ; выделить старшей тетрады
lsrb
lsrb
lsrb
lsrb
lsrb      ; умножение на 10 и сложение
aba      ; с регистром A
lsrb
lsrb
aba
tab
pula
rts
dpm      fcb  $31,$28,$31,$30,$31,$30,$31,$31,$30,$31,$30,$31

```

BRSET (opr),
BRCLR (opr)

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

Команды условного перехода, выполняющие передачу управления в зависимости от значения ячейки памяти, используются в циклах ожидания изменения какого-либо регистра управления, опросе внешних линий данных или работе с другими

битовыми данными. Простой пример показывает, каким образом можно обеспечить отображение информации с переключателей на светодиоды по изменению переключателя CO:

```

org  $8000
ldx  #$1f00      ; установить регистр X на базовый адрес
brset 3,x,$01,p1 ; если младший бит установлен, тогда
                    ; перейти к программе ожидания сброса
p0    bsr  display ; отобразить информацию
      brclr 3,x,$01,* ; ждать установки бита
                    ; * - адрес текущей команды
p1    bsr  display
      brset 3,x,$01,* ; ждать сброса бита
      bra  p0        ; следующий цикл
display ldaa 3,x      ; отобразить состояние
      staa 4,x
      rts

```

SWI

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

RTI

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

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

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

Иногда бывает необходимо выполнить программную реализацию прерывания. Конечно, это можно реализовать через подпрограмму, в начале которой выполняется сохранение регистров в стеке, однако это снижает эффективность работы программы в целом. Для выполнения этой задачи служит команда

генерации программного прерывания SWI. Она выполняет последовательное сохранение в стеке регистров PC+1, Y, X, D, CCR, запрещает маскируемые прерывания (устанавливает бит I в регистре CCR) и передает управление на подпрограмму, адрес которой находится в таблице векторов прерывания. Адрес обработчика прерывания SWI должен быть расположен по адресу \$fff6, если работа происходит в нормальном режиме работы, или по адресу \$bff6, если в специальном. В режиме bootstrap по адресам \$bf40÷\$bfff находится bootstrap ПЗУ, в котором вектора прерываний указывают на ячейки памяти внутреннего ОЗУ. Таким образом, в исследуемых в данной лабораторной работе экспериментах для задания окончательного адреса перехода мы будем использовать адреса \$f4÷\$f6, в которых будет расположена команда безусловного перехода JMP.

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

Небольшая программа демонстрирует работу этих двух команд:

```

                org    $00f4      ; установка вектора обработчика
                jmp    ih          ; прерывания SWI
                org    $8000
gen_int         ldaa   #$55
                swi      ; вызов прерывания
                coma
                swi
                coma
                swi
                coma
                swi
done            bra    *
ih              ldx    #$1f04      ; обработчик прерывания осуществляет
                staa   0,x         ; отображение числа из регистра A
                ldy    #$500       ; и задержку ≈10 мс
delay          dey
                bne    delay
                rti

```

Попробуйте выполнить эту программу в пошаговом режиме (как было указано в предыдущей работе, обработчик прерывания будет выполняться за 1 шаг) и с установкой точек останова на метке ih. Посмотрите содержимое области данных выше указателя стека (стекового фрейма) и убедитесь, что все регистры процессора были сохранены.

3. Специальные команды

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

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

Команда STOP выполняет остановку всех внутренних генераторов микроконтроллера и перевод системы в режим минимального энергопотребления. В случае, если установлен бит S регистра CCR, то команда STOP выполняется как команда NOP. Восстановление системы из режима минимального энергопотребления может произойти в случае появления прерываний от RESET, XIRQ или немаскированного прерывания IRQ. В случае, когда установлен бит X регистра CCR, маскирующий прерывание XIRQ, и происходит это прерывание, то выполнение программы происходит со следующей за STOP команды. В некоторых масках семейства M68HC11 была допущена ошибка, приводящая к неправильному интерпретированию кода команды в некоторых особых случаях, поэтому фирма Motorola рекомендует перед командой STOP помещать команду NOP, таким образом, исключая эту ошибку.

4. Контрольные вопросы

1. Каково различие между командами JMP и BRA?
2. Объясните различие между командами WAI и STOP.
3. Каким образом можно реализовать переход к подпрограмме, не используя команд BSR и JSR?
4. Произойдет ли вызов программного прерывания при установленном флаге I в регистре CCR?
5. Какие команды относятся к знаковым командам условного перехода?
6. Какие виды переходов Вам известны?
7. Каково назначение команд BLE, BSR, BCS, BRCLR?
8. Сколько операндов и какие используются командами условного перехода по состоянию бита?
9. Реализуйте (примерно) команды BRCLR и BRSET через другие команды.
10. Каков результат выполнения фрагмента программы:

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

11. Можно ли выполнить переход, аналогичный переходу по команде BCS, ис-

пользуя команды BNE и BLE?

12. Каким образом можно осуществить корректный выход из подпрограммы, не используя команду RTS?
13. Какие команды относятся к беззнаковым командам условного перехода?
14. Для какой цели используются команды WAI и STOP?
15. Каково назначение команд BLE, RTI, JSR, BEQ?

5. Задания

1. Напишите программу, осуществляющую сложение двух 4-х байтных чисел, представленных в формате BCD.
2. Реализуйте перевод двухбайтного числа в формате BCD в двоичный формат.
3. Напишите программу, копирующую блок данных, расположенных по адресам \$8200÷\$8220, в соответствующие ячейки \$0000÷\$0020. При этом данные перезаписываются только в том случае, если бит 3 в соответствующей ячейке памяти сброшен.
4. Написать программу подсчета суммы 8-битных беззнаковых чисел, расположенных в ячейках \$8200÷\$82ff. Результат поместить в регистр Y.
5. Произвести сортировку по возрастанию чисел, расположенных в ячейках \$8200H-\$82ff.
6. Написать программу, подсчитывающую количество установленных битов в ячейках памяти \$8200÷\$821f. Результат поместить в регистр X.
7. Произвести сортировку по убыванию чисел, расположенных в ячейках \$8200÷\$82ff.
8. Написать программу, производящую подсчет количества нечетных чисел в ячейках \$8200÷\$82ff.
9. Написать программу преобразования двоичных чисел, расположенных в ячейках \$8200÷\$821f, в BCD формат. Результат разместить в ячейках \$8220-\$825f.
10. Написать программу подсчета суммы 8-битных знаковых чисел, расположенных в ячейках \$8200÷\$82ff.
11. Произвести операцию "логическое ИЛИ" между битом 2 и битом 5 для ячеек памяти, расположенных по адресам \$8200÷\$821f, результат при этом должен быть записан в бит 3 соответствующей ячейки.
12. Произвести обмен старших тетрад ячеек, расположенных в блоках \$8200÷\$821f и \$8220÷\$823f.
13. Перестроить массив данных размером 256 байт в обратном порядке. Т.е. первый байт меняется местами с последним, второй с предпоследним и т.п.
14. Напишите программу, осуществляющую сдвиг массива данных размером 64 байта на 4 бита влево.
15. Напишите программу, зеркально перестраивающую биты в массиве данных размером 256 байт. Т.е. нулевой бит первого элемента массива становится последним битом последнего элемента, первый бит первого элемента - 6-ым последним элементом и т.д.

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

Приложение

Пример программы реализующую следующую математическую функцию:

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

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

Ответ формируется в следующих регистрах:

X – старшая часть;

Y – младшая часть.

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

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

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

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

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

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

```
ldd $8210
idiv
stx $8210
std $8250      ; остаток от a*a/a+2
```

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

```
ldd $8220
ldx $8240
```



```

idiv
stx $8220
std $8260      ; остаток от b*b/a+2

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

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

ldd $8250      ; 1-ый остаток в d
addd $8260     ; 1-ый + 2-ой остаток
addd $8270     ; 1-ый + 2-ой + 3-ий остаток

ldx $8240
idiv
std $8250      ; сохраняем последний остаток
xgdx
addd $8210
addd $8220
addd $8230

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

ldx m1
ldy m2

stx $8290      ; старшая часть результата
sty $8280      ; младшая часть результата

ldd $8250
ldaa #d
mul
std $8260      ; младшая часть от c*d

ldd $8250
ldab #d
mul
std $8270      ; старшая часть от c*d
pshb
pula
clrb
addd $8260
std $8270      ; c*d

ldx $8240
idiv
std $8210      ; остаток от всех остатков

```

stx \$8220 ; добавить к результату

ldd \$8280

clra

addd \$8220 ; прибавка точности

pshb

psha

pulb

clra

addd \$8290

xgdx ; формируем результат

pula

clrb

xgdy

xgdy

psha

xgdx

pshb

psha

clra

psha

pulx

puly

clra

clrb

loop:

bra loop ; бесконечный цикл

rts

tw dw 0

m1 db 0

m2 db 0

m3 db 0