Московский Государственный Институт Электронной Техники
МГИЭТ (ТУ)

Панфилов Д.И., Плавич М.Л., Аганичев А.С.

8-разрядные микроконтроллеры семейства M68HC11 фирмы Motorola

Лабораторный практикум

1998

Содержание

Лабораторная работа №1

Отладочный комплект.

1. Введение

Данный курс лабораторных работ предназначен для получения начальных практических навыков работы с микроконтроллерами семейства M68HC11 фирмы Motorola. Курс предполагается проводить на отладочном модуле HC11EVB, работающим с микроконтроллером MC68HC11E1. Перед проведением курса необходимо ознакомится с описанием на микроконтроллер MC68HC11E1, на отладочный модуль и на программу-отладчик XDBG11.

В последующих работах Вам будет необходимо произвести стандартную последовательность действий:

1. создание программы в редакторе;
2. ассемблирование программы и исправление ошибок;
3. загрузка программы в память отладочного модуля;
4. отладка программы.

В данной лабораторной работе иллюстрируются основные приемы выполнения этих действий.

2. Общая структура отладочного комплекса

Общая структура отладочного комплекса представлена на рис.1.1. Он включает в себя IBM PC совместимый компьютер, на котором работает программа XDBG11, и отладочный модуль HC11EVB. Питание отладочного модуля может осуществляться как от отдельного источника питания, так и от компьютера (через разъем подключения дисковода 5”25).

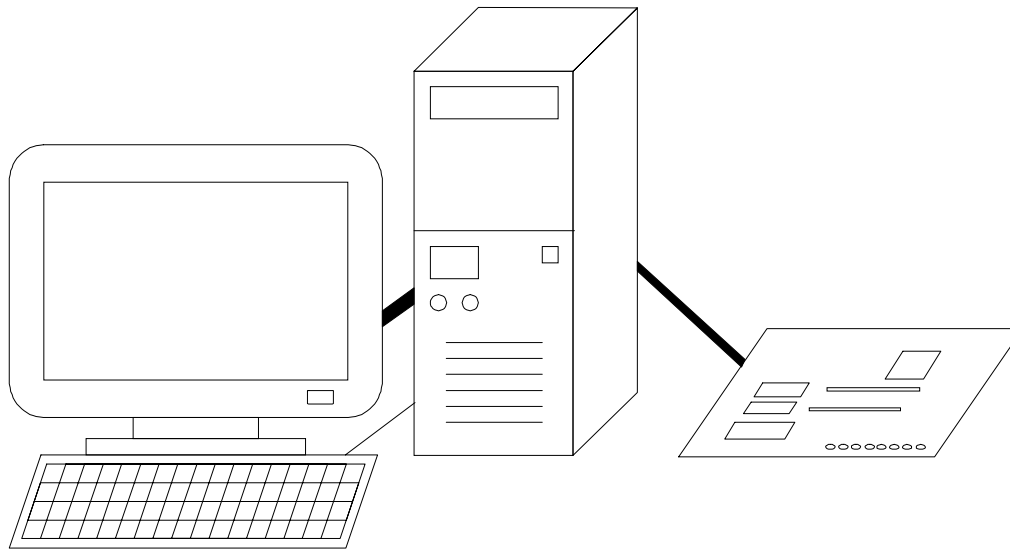


Рис.1.1

Основными частями отладочного модуля являются:

1. микроконтроллер MC68HC11E1;
2. эмулятор портов (PRU) MC68HC24;
3. внешнее ОЗУ (16Кб);
4. система связи с компьютером;
5. блок индикации (8 светодиодов, подключенных к выводам порта B);
6. блок ввода (8 переключателей, подключенных к выводам порта C, кнопка генерации прерывания IRQ и кнопка генерации прерывания PA0);
7. система сброса;

8. два разъема для подключения внешних устройств (на одном из разъемов происходит эмуляция ОЭВМ в однокристальном (single chip) режиме, на втором - в расширенном (expanded) режиме).

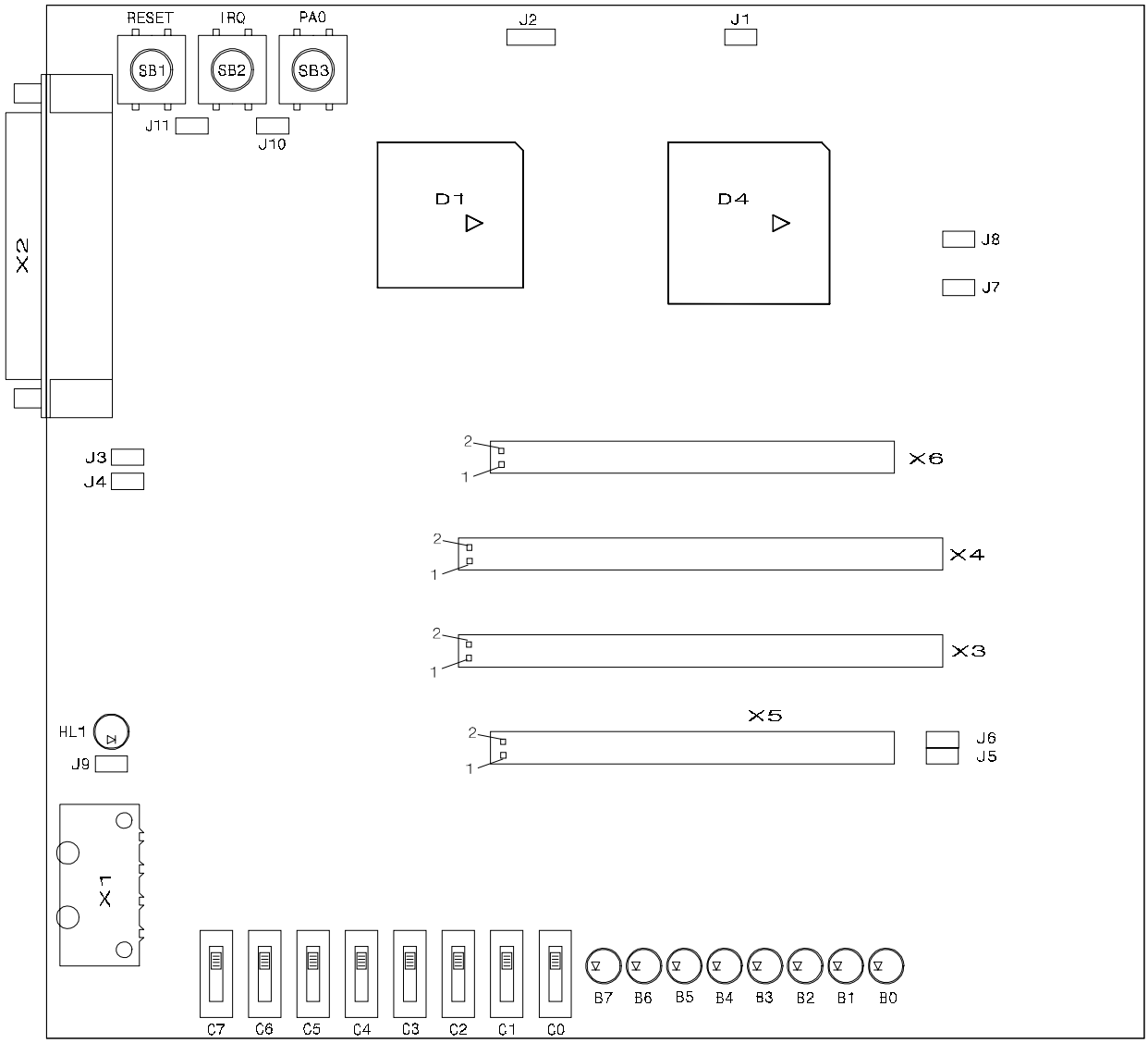


Рис.1.2

Расположение переключателей, перемычек, светодиодов модуля, а также расположение разъемов на плате представлено на рис.1.2. Всего на плате установлено 11 перемычек (J1-J11), описание которых приводится ниже:

- **J1, J2** - управление источником синхронизации микросхемы M68HC11.

Таблица 1.1

Положение **J1**	Положение **J2**	Источник синхронизации M68HC11
1-2 (перемычка установлена)	2-3	кварцевый резонатор 8 МГц, установленный на плате
перемычка снята	1-2	внешний сигнал подаваемый на вывод 7 (“EXTAL”) разъема X3 или X4

- **J3, J4** - подключение блока последовательного интерфейса к M68HC11. Установленная перемычка J3 подключает вывод PD1 (TXD) микросхемы M68HC11 ко входу передатчика последовательного интерфейса. Установленная перемычка J4 подключает выход приемника последовательного интерфейса к выводу PD0 (RXD) M68HC11.
- **J5, J6** - выбор режима работы микросхемы M68HC11 после сброса. Перемычка J5 управляет сигналом MODA M68HC11 (установленная перемычка соответствует “0”, отсутствие перемычки - “1”), перемычка J6 - сигналом MODB.

- **J7, J8** - подача опорных напряжений на внутреннее АЦП микросхемы M68HC11. Если установлена перемычка J7, на вывод VRH HC11 подается напряжение +5В. При установленной перемычке J8, вывод VRL M68HC11 подключается к земле.

Таблица 2.1

J5	J6	состояние MODA	состояние MODB	Режим работы микросхемы M68HC11
установлена	установлена	0	0	специальный режим загрузки
установлена	снята	0	1	однокристальный режим
снята	установлена	1	0	специальный режим теста
снята	снята	1	1	расширенный режим

- **J9** - подача напряжения программирования (+12.25 В) на вывод /XIRQ микросхемы M68HC11. Если перемычка J9 снята, на вход /XIRQ поступает через резистор напряжение +5В (таким образом обеспечивается уровень “1”). Если перемычка J9 установлена (при этом зажигается красный светодиод HL1), на вход /XIRQ подается напряжение программирования Vpp с вывода 4 разъема X1, необходимое для программирования встроенного ПЗУ микросхемы M68HC11.
- **J10** - подключение клавиши “PA0” к выводу PA0 микросхемы M68HC11. При установленной перемычке J10, на вывод PA0 M68HC11 поступает либо “1” (клавиша PA0 не нажата), либо “0” (клавиша PA0 нажата).
- **J11** - разрешение прерывания по линии /IRQ от эмулятора портов MC68HC24. Прерывание разрешается при установленной перемычке J11.

Переключатели SA1-SA8 предназначены для формирования входных воздействий на линии порта C эмулятора портов и соответствуют линиям PC0-PC7 порта (логический “0” формируется в нижнем положении переключателя). Светодиоды HL2-HL9 отображают состояние линий порта B эмулятора портов и соответствуют линиям PB0-PB7 порта (горящий светодиод индицирует наличие логической “1” на соответствующей линии).

3. Запуск и начальная настройка среды

Запуск среды осуществляется вводом в командной строке DOS команды “xdbg11” с последующим нажатием клавиши <Enter>. После запуска пользователю сообщается о загруженном по умолчанию модуле поддержки какого-либо типа контроллеров или предлагается выбрать модуль из имеющихся (смену модуля можно произвести нажатием клавиши <F6>). Далее следует удостовериться, что программа настроена на взаимодействие с платой через правильный коммуникационный порт в меню “Option→Communication”. Теперь программа готова к работе. Для установки связи с платой выберите пункт меню “Debug→Connect” или нажмите клавишу <F4>. В случае ошибки соединения необходимо нажать клавишу “Reset” на плате отладочного модуля и повторить операцию.

4. Редактирование и ассемблирование программы

Создайте новую программу, для чего откройте окно через меню “File→New”, введите текст примера, который будет использован в последующих экспериментах:

```
            cpu   6811          ; определить тип процессора
            org   $c800         ; адрес размещения программы в памяти
                                ; контроллера
            ldaa  #$10          ; загрузить в регистр A 16
11          staa  0,x           ; сохранить содержимое регистра A в
                                ; ячейке памяти, на которую указывает
                                ; регистр X
            inx                 ; увеличить содержимое регистра X
            deca                ; уменьшить содержимое аккумулятора
                                ; на единицу
```

```
        bne   11            ; переход, если содержимое
                            ; аккумулятора не равно 0
```

Сохраните программу, выбрав в меню “File” пункт “Save as...” или нажав клавишу <F2>. Произведите ассемблирование, выбрав в меню “File” пункт “Assembly”. В случае, если при ассемблировании не произошло ошибок, то в рабочем каталоге будет создан файл с указанным Вами именем и расширением S19.

5. Загрузка программы в память контроллера

Для загрузки программы в память отладочного модуля следует воспользоваться командой отладчика “Debug→Download”. После этого откроется окно, в котором предлагается открыть объектный код Вашей программы в формате Motorola S-Records.

6. Просмотр и изменение внутренних регистров

Для просмотра содержимого регистров служит верхняя центральная ячейка окна “Debug window”. Чтобы изменить содержимое регистров, необходимо при помощи клавиши табуляции и стрелок подвести курсор к нужному регистру и затем нажать пробел (в этом случае на экран будет выведено окно, в котором можно будет ввести новое значение) или клавиши ‘c’ или ‘n’ соответственно для получения двоично-сопряженного или числа с противоположным знаком.

7. Просмотр и изменение ячеек памяти

Просмотр содержимого памяти можно осуществить выбрав в меню “Debug→Memory” подпункт “Inspect”, после чего появится окно, в котором Вам будет предложено ввести начальный и конечный адрес для просмотра. Для изменения содержимого регистров необходимо вызвать пункт меню “Debug→Memory→Modify”.

При изменении регистра CONFIG или ячейки EEPROM следует убедиться, что биты защиты в регистре BPROT сброшены. Введите адрес $1034 и нажмите стрелку вверх. При этом в окне данных отобразится содержимое ячейки $1035 (BPROT). После сброса запись в EEPROM и CONFIG запрещена и в регистре BPROT находится число $1f.

Вторым, иногда упускаемым моментом, является то, что записанное в регистр CONFIG значение становится активным (и видимым) лишь после сброса микроконтроллера.

8. Различные режимы выполнения программы

Отладочный комплект обеспечивает работу программы в двух режимах: режим запуска и режим трассировки.

Сначала рассмотрим режим трассировки. В этом режиме при нажатии клавиши <F7> выполняется ровно одна команда, на которую указывает программный счетчик PC. Исполняемая команда не должна запрещать прерывания процессора, иначе выполнение будет происходить до точки программы, в которой прерывания будут разрешены. Тут же следует отметить, что время ответа лимитировано и может произойти разрыв связи (в этом случае можно воспользоваться опцией “Debug→Resynchronize” для восстановления связи без потери данных). Для трассировки программы из приведенного выше примера необходимо предварительно занести в регистр X число $0000, а в регистр PC число $c800.

Если необходимо какой-либо фрагмент текста выполнить целиком (например, это может быть цикл с большим параметром или программа с ветвлением), то можно воспользоваться вторым режимом - выполнение программы до точки останова. Для установки точки останова в окне “Debug window” надо стрелками выберать команду, на которой необходимо поставить точку останова. В нашем случае это будет команда, следующая за командой “bne $c802”. Далее нажмите <Ctrl-F8> и, убедившись что содержимое программного счетчика корректно, нажмите клавишу <F9> (соответствует опции “Debug→Run Program”). Программа остановится сразу после выполнения цикла.

9. Контрольные вопросы

1. Каковы функции опции “Option→Communication”?
2. Каким образом осуществляется просмотр содержимого памяти?
3. Каковы функции кнопки “Trace”?
4. Что представляют из себя данные, выводимые командой “Decode”?
5. Каково назначение точек останова?
6. Какой командой выполняется просмотр содержимого регистров ОЭВМ?
7. Каковы основные режимы выполнения программ?
8. Каким образом осуществляется создание и редактирование файлов в редакторе?
9. Назовите основные части ОМ.
10. Каким образом осуществляется загрузка программы в память ОМ?
11. Какое дополнительное оборудование и как можно подключить к ОМ?
12. Какие особенности имеет ОЭВМ MC68HC11E1?
13. Какие тонкости существуют при модификации регистра CONFIG?
14. Каковы функции директивы “Debug→Registers→Modify”?
15. Каким образом в кросс-ассемблере устанавливается адрес начала программы?

Лабораторная работа №2

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

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

```
96 3F       LDAA  63
DA FF       ORAB  $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 2.1. Рассмотрим каждую из команд подробнее на простых примерах.

Команды TSTA, TSTB и TST служат для установки регистра статуса в соответствии с

Таблица 2.1. 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	-

содержимым регистра A, B или ячейки памяти соответственно. Далее результат может быть использован в командах условного перехода. Занесите в регистр A значение $00 и выполните в пошаговом режиме команду TSTA. 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 A, обращая на различное состояние регистра статуса.

Рассмотрим команды загрузки в регистр содержимого ячейки памяти. Загрузите в память EVB следующую программу:

```
        cpu   6811          ; определение типа микроконтроллера
        org   $c800         ; начальный адрес программы
        ldab  $56           ; загрузить в регистр B содержимое
                            ; ячейки $56, используя прямую
                            ; адресацию
        ldy   $c800         ; загрузить в регистр Y данные,
                            ; расположенные по адресу $c000
                            ; (предыдущую команду)
        ldx   #$1f00        ; установить регистр X на базовый
                            ; адрес PRU MC68HC24
        ldaa  $03,x         ; считать информацию с переключателей
```

Выполните ее в пошаговом режиме, обращая внимание на состояние регистра статуса. Здесь представлены все виды адресации, используемые в этой группе команд.

CLRA, CLRB, CLR (opr)

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

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

```
        cpu   6811          ; определить тип микроконтроллера
        org   $c800         ; начальный адрес программы
        clrb                ; очистить регистр B
        ldx   #$1f00        ; установить регистр X на базовый
                            ; адрес PRU MC68HC24
        clr   $04,x         ; погасить светодиоды
```

Предварительно занесите в порт B микросхемы эмулятора портов (ячейка памяти $1f04) какое-нибудь число, например $AA, затем выполните программу в пошаговом режиме. Обратите внимание, что при завершении программы в регистре B будет находиться значение $00, а светодиоды погаснут.

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

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

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

```
        cpu   6811          ; определить тип микроконтроллера
        org   $c800         ; начальный адрес программы
        ldd   #$AA55        ; установить в регистре D значение
                            ; $AA55
        ldx   #$1f00        ; установить регистр X на базовый
```

```
                                ; адрес PRU MC68HC24
            clr   $04,x         ; погасить светодиоды
            staa  $04,x         ; зажечь все четные светодиоды
            stab  $04,x         ; зажечь все нечетные светодиоды,
                                ; погасив четные
            ldaa  $03,x         ; считать информацию с переключателей
            staa  $04,x         ; отобразить состояние переключателей
                                ; на светодиодах
```

TAB, TBA

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

Работу команд пересылки данных между регистрами A и B легко проследить на простом тесте: введите каждую из команд и выполните в пошаговом режиме. 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.

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

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

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:

```
            cpu   6811
            org   $c800         ; начальный адрес программы
            ldx   #$220         ; занести в регистр X адрес,
                                ; следующий за границами внутреннего
                                ; ОЗУ, плюс некоторое смещение ($20)
            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.

```
            cpu   6811
            org   $c800         ; начальный адрес программы
            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

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

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

1. в ячейку памяти, на которую указывает регистр SP записывается (младший) байт регистра-операнда;
2. значение регистра SP уменьшается на 1, указывая на следующую свободную ячейку в области стека;
3. в случае двухбайтного операнда последовательность (1-2) повторяется со старшим байтом операнда.

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

Загрузите и выполните в пошаговом режиме следующую программу, демонстрирующую каким образом можно сохранить неизменными все внутренние регистры ОЭВМ (рекомендуется так же обратить внимание на содержимое стека):

```
cpu   6811
org   $c800
psha                ; последовательно сохраняем регистры
pshb                ; в стеке: A, B, X, Y, CCR
pshx
pshy
tpa
psha
ldaa  #$20          ; выполняем какие-либо действия,
ldx   $12           ; в результате которых изменяется
ldy   $1f03         ; содержимое регистров
clrb
xgdy
pula                ; восстанавливаем регистры: CCR, Y,
tap                 ; X, B, A
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 $C900÷$C905 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 $C800.
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, не используя непосредственную адресацию.[1]
7. Произвести обмен содержимого младшего байта регистра X с регистром A.
8. Изменить порядок следования байт в регистре X, не используя команду XGDX.
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 $0000÷$0008. При этом содержимое данных ячеек памяти должно соответствовать значению регистров при входе в программу.

[1] -задание повышенной сложности

Лабораторная работа №3

Арифметические команды.

1. Введение

Приведенные в данном разделе эксперименты предназначены для изучения работы арифметических команд: сложения, вычитания, умножения, деления и десятичной коррекции.

2. Арифметические команды

Список арифметических команд приведен в таблице 3.1. Приведем примеры использо-

Таблица 3.1. 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

вания этих команд в порядке увеличения сложности.

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

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).

INS, DES

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

INX, INY, DEX, DEY

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

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:

```
        cpu   6811
        org   $c800
        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, $11 соответственно
        ldx   $10           ; загрузить в регистр X
        inx                 ; инкрементировать регистр X
```

```
        des             ; декремент указателя стека
```

NEGA, NEGB,
NEG (opr)

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

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

```
        cpu   6811
        org   $c800
        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.

```
        cpu   6811
        org   $c800
        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
-	-	-	-	?	?	?	?

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.

ABX, ABY

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

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. В одной из последующих лабораторных работ мы рассмотрим подобный пример, реализованный с помощью цикла.

```
        cpu   6811
        org   $c800
        ldx   #01
        ldd   0,x       ; сложение младших 2 байт
```

```
        addd  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 полупереноса H установлен в “1”, то к аккумулятору добавляется число 6;
2. если содержимое старшей тетрады аккумулятора стало после этого более 9 или установлен флаг переноса, то число 6 добавляется и к старшей тетраде аккумулятора.

```
        cpu   6811
        org   $c800
        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 аккумуляторов. Результат помещается в 16-разрядный аккумулятор D. Флаг переноса при этом устанавливается таким образом, что при выполнении команды ADCA #0 происходит округление старшего байта.

```
        cpu   6811
        org   $c800
        ldaa  #$10
        ldab  #$68
        mul                 ; $10 * $68 = $0680
```

```
        adca  #0          ; A = $7
```

IDIV

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

FDIV

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

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 FDIV 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.

```
        cpu   6811
        org   $c800
        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/заема 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.
2. Напишите программу суммирования регистров MK 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 двухмерного массива размерностью 6×6, расположенного по адресу $c900. Индекс задается регистрами A и B.
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 №4

Логические команды. Команды работы с битовыми полями. Команды сдвигов.

1. Введение

В данной лабораторной работе будут приведены эксперименты, предназначенные для изучения работы логических команд (операции НЕ, И, ИЛИ, исключающее ИЛИ), команд работы с битовыми полями (установка и сброс битов) и 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 4.1. Логические команды.

COMA	ANDA	BITA	ORAA	EORA
COMB	ANDB	BITB	ORAB	EORB
COM				

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 (крайних переключателей) в старший и младший биты порта B (крайних светодиодов). 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).

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	-

```
        cpu   6811
        org   $c800
        ldaa  $1f03         ; считать состояние переключателей
        ldab  $1f04         ; считать состояние светодиодов
        andb  #%01111110    ; выделить неизменную часть
        oraa  #%01111110    ; установить все неиспользуемые биты
                            ; в "1"
        coma                ; инвертировать значение аккумулятора
        eora  #%10000000    ; инвертировать значение старшего
                            ; бита
        aba                 ; совместить результат (заметьте, что
                            ; мы заблаговременно маскировали
                            ; неиспользуемые биты, чтобы сейчас
                            ; совместить два числа простым
                            ; суммированием)
        staa  $1f04         ; высветить результат на светодиодах
```

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 4.2.

Таблица 4.2. 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 (C - флаг переноса, I - маскирование прерываний, V - флаг переполнения). Приведем простой пример, показывающий один из способов занесения числа 1 в аккумулятор:

```
cpu   6811
org   $c800
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	H	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 на светодиодах работу этих команд:

```
cpu   6811
org   $c800
ldx   #$1f00        ; настроить регистр X
bset  4,x,#$AA      ; зажечь через один светодиоды, оставив
                    ; незадействованные биты в прежнем
                    ; состоянии
bclr  4,x,#$55      ; погасить остальные светодиоды
```

Выполните эту программу в пошаговом режиме при разных начальных состояниях светодиодов (ячейки $1f04).

4. Команды сдвигов.

Список команд сдвигов представлен в таблице 4.3. Эти команды позволяют адресоваться к аккумулятору или ячейке памяти.

Команды сдвигов обычно подразделяют на три группы:

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

Таблица 4.3. 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.

Все эксперименты в этой части лабораторной работы будут производиться со светодиодами, так как это наиболее наглядный способ для демонстрации операций сдвига.

```
        cpu   6811
        org   $c800
        ldaa  #%00101001
        ldx   #$1f00
        staa  4,x          ; высветить на светодиодах 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 кода операций, поэтому покажем лишь отличие команд ASR от команд LSR:

```
        cpu   6811
        org   $c800
        ldaa  #%10101010
        staa  $1f04        ; зажечь светодиоды через один
        asr   $1f04        ; арифметический сдвиг вправо:
        asr   $1f04        ; старший бит сохраняется
        lsr   $1f04        ; логический сдвиг:
        lsr   $1f04        ; старший бит заполняется 0
                           ; (светодиод гаснет)
```

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 использова-

ния этих команд для сдвига 4-х байтного числа, расположенного в ячейках 0÷3, влево:

```
cpu   6811
org   $c800
ldx   #0
lsl   3,x
rol   2,x
rol   1,x
rol   0,x
```

5. Контрольные вопросы

1. Каков результат выполнения программы:

```
cpu   6811
org   $c800
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 и5 биты в регистра A с помощью команды BSET.

Лабораторная работа №5

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 5.1. Все команды передачи управления не оказывают влияния на состояние регистра статуса.

Таблица 5.1. Команды передачи управления.

JMP*	BEQ**	BCS/BLO**	BGT**	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:

```
            cpu   6811
            org   $c800          ; начальный адрес программы
            ldab  #$02           ; выбор варианта ветвления программы
            ldx   #ways          ; занести в регистр X адрес таблицы
                                 ; переходов
10          ldy   0,x            ; считать значение в регистр Y
            jmp   0,y            ; вызвать подпрограмму по адресу,
                                 ; находящемуся в регистре Y
11          nop                  ; задержка в 2 такта
            inx                  ; увеличить регистр X на 2 для
            inx                  ; выборки следующего адреса из
                                 ; таблицы переходов
            bra   10             ; перейти наметку 10
12          bra   11             ; перейти на метку 11
13          brn   *              ; задержка в 3 такта
```

```
ways        fdb   l1, l2, l3
```

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 l2, а во втором - l3.

JSR, BSR, RTS

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

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

```
            cpu   6811
            org   $c800
            bsr   l1              ; переход на следующую команду
l1          pulx                  ; получить в регистр X адрес l1 ($c802)
            xgdx                  ; расписать адрес в формате ASCII
            ldy   #0              ; в ячейках $0÷$3
            bsr   bin2ascii       ; вызвать подпрограмму, сохраняющую
                                  ; регистр A в формате ASCII в ячейках
                                  ; (y), (y+1)
            tba                   ; переписать регистр B в регистр A
            iny                   ; и расписать его в ячейках $2, $3
            iny
            bsr   bin2ascii
            bra   *
bin2ascii   pshb                  ; сохранить в стеке регистр B
            tab                   ; скопировать в него регистр A
            lsra                  ; выделить в A старшую тетраду
            lsra
            lsra
            lsra
            bsr   hex2ascii       ; преобразовать число 0-f в ASCII код
            staa  0,y
            tba                   ; повторить для младшей тетрады
            anda  #$f
            bsr   hex2ascii
            staa  1,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 и простые (см. табл. 5.2).

Таблица 5.2. Команды условного перехода.

Условие	Логическая функция	Мнемоника	Противоположное действие		Тип
r>m	Z+(N⊕V)=0	BGT	r⩽m	BLE	знаковый
r⩾m	N⊕V=0	BGE	r<m	BLT	-‖-
r=m	Z=1	BEQ	r≠m	BNE	-‖-
r⩽m	Z+(N⊕V)=1	BLE	r>m	BGT	-‖-
r<m	N⊕V=1	BLT	r⩾m	BGE	-‖-
r>m	C+Z=0	BHI	r⩽m	BLS	беззнак.
r⩾m	C=0	BCC/BHS	r<m	BCS/BLO	-‖-
r=m	Z=1	BEQ	r≠m	BNE	-‖-
r⩽m	C+Z=1	BLS	r>m	BHI	-‖-
r<m	C=1	BCS/BLO	r⩾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≠0	BNE	-‖-

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

```
            cpu   6811
            org   $c800
            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):

```
            cpu   6811
            org   $c800
            ldx   #0            ; установить указатель на начало даты
            ldd   0,x           ; загрузить в A и B день и месяц
            tstb                ; если день или месяц равен 0 или
            beq   invalid       ; месяц больше 13, то дата не корректна
            tsta
            beq   invalid
            cmpb  #$13
            bhs   invalid
            cmpb  #$2           ; если это не февраль, то переход на
            bne   lab2          ; выборку числа дней из таблицы
            ldab  3,x           ; проверка високосного года
            bsr   bcd2bin
```

```
            andb  $03           ; получение остатка от деления на 4
            bne   lab2          ; год невисокосный, если не делится
                                ; нацело на 4
            ldab  #$29
            bra   lab1
lab2        bsr   bcd2bin       ; преобразование месяца в двоичный код
            decb
            ldy   #dpm          ; настроить указатель на список дней
            aby                 ; в месяце
            ldab  0,y           ; занести в B число дней в месяце
lab1        cba                 ; если число дней больше вычисленного,
            bhi   invalid       ; то дата не корректна
valid       clc                 ; год может быть любым, поэтому
            bra   done          ; проверка не производится и
                                ; возвращается признак успешной
                                ; проверки
invalid     sec                 ; установка флага ошибки
done        bra   *
bcd2bin     psha                ; преобразование числа в формате BCD,
            tba                 ; заданного в регистре B в двоичный
            anda  #$0F          ; формат
            lsrb                ; выделение старшей тетрады
            lsrb
            lsrb
            lsrb
            lslb                ; умножение на 10 и сложение
            aba                 ; с регистром A
            lslb
            lslb
            aba
            tab
            pula
            rts
dpm         fcb   $31,$28,$31,$30,$31,$30,$31,$31,$30,$31,$30,$31
```

BRSET (opr), BRCLR (opr)

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

Команды условного перехода, выполняющие передачу управления в зависимости от значения ячейки памяти используются в циклах ожидания изменения какого-либо регистра управления, опросе внешних линий данных или работе с другими битовыми данными. Простой пример показывает каким образом можно обеспечить отображение информации с переключателей на светодиоды по изменению переключателя C0:

```
            cpu   6811
            org   $c800
            ldx   #$1f00        ; установить регистр X на базовый
                                ; адрес PRU MC68HC24
            brset 3,x,#$01,l1   ; если младший бит установлен, тогда
                                ; перейти к программе ожидания сброса
l0          bsr   display       ; отобразить информацию
            brclr 3,x,#$01,*    ; ждать установки бита (включения C0)
                                ; * - адрес текущей команды
l1          bsr   display
            brset 3,x,#$01,*    ; ждать сброса бита (выключения C0)
```

```
            bra   l0              ; следующий цикл
display     ldaa  3,x             ; отобразить состояние переключателей
            staa  4,x             ; на светодиодах
            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:

```
            cpu   6811
            org   $00f4           ; установка вектора обработчика
            jmp   ih              ; прерывания SWI

            org   $c800
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           ; на светодиодах и задержку ≈10мс
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 фрэйма) 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.

5. 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   l2
            beq   l3
            bra   done
l2          ldaa  #45
            bra   done
l3          ldaa  #$23
done        clrb
```

11. Можно ли выполнить переход, аналогичный переходу по команде BCS, используя команды BNE и BLE?
12. Каким образом можно осуществить корректный выход из подпрограммы, не используя команду RTS?
13. Какие команды относятся к беззнаковым командам условного перехода?
14. Для какой цели используются команды WAI и STOP?
15. Каково назначение команд BLE, RTI, JSR, BEQ?

6. 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 $c900÷$c920, 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 $c900÷$c9ff. Результат поместить в регистр Y.
5. Произвести сортировку по возрастанию чисел, расположенных в ячейках $c900÷$c9ff.
6. Произвести подсчет количества отрицательных, положительных чисел и нулей в области памяти $c900÷$c9ff.
7. Написать программу, подсчитывающую количество установленных битов в ячейках памяти $c900÷$c91f. Результат поместить в регистр X.
8. Произвести сортировку по убыванию чисел, расположенных в ячейках $c900÷$c9ff.
9. Написать программу, производящую подсчет количества нечетных чисел в ячейках $c900÷$c9ff.
10. Написать программу преобразования двоичных чисел, расположенных в ячейках $c900÷$c91f, в BCD формат. Результат разместить в ячейках $c920-$c95f.

11. Написать программу подсчета суммы 8-битных знаковых чисел, расположенных в ячейках $c900÷$c9ff.
12. Произвести операцию “логическое ИЛИ” между битом 2 и битом 5 для ячеек памяти, расположенных по адресам $c900÷$c91f, результат при этом должен быть записан в бит 3 соответствующей ячейки.
13. Произвести обмен старших тетрад ячеек, расположенных в блоках $c900÷$c91f и $c920÷$c93f.
14. Перестроить массив данных размером 256 байт в обратном порядке. Т.е. первый байт меняется местами с последним, второй с предпоследним и т.п.
15. Напишите программу, осуществляющую сдвиг массива данных размером 64 байта на 4 бита влево.
16. 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-м последнего элемента и т.д.
17. Напишите программу вычисления факториала числа, заданного в регистре B.

Лабораторная работа №6

Порты параллельного ввода/вывода.

1. Введение

В данной лабораторной работе изучается процесс взаимодействия с портами параллельного ввода/вывода.

2. Порты параллельного ввода/вывода

Микроконтроллер MC68HC11E1 имеет пять параллельных портов ввода/вывода: A, B, C, D и E. Каждая линия ввода/вывода многофункциональна, т.е. за исключением простого параллельного обмена может выполнять дополнительные функции (например, линии порта E могут быть настроены на работу в качестве входов встроенного АЦП; линии порта D могут служить линиями обмена последовательных интерфейсов; линии порта A могут быть подключены к системе таймера и т. д.). При работе в режиме обычного ввода/вывода линии портов B, E, а также линии 0, 1, 2, 4, 5, 6 порта A имеют фиксированные направления передачи данных, в то время как направления обмена линий портов C, D и линий 3 и 7 порта A можно задавать программно. Каждому порту ввода/вывода соответствует внутренний регистр в адресном пространстве микроконтроллера. Для портов A, B, C, D, E эти регистры называются PORTA, PORTB, PORTC, PORTD, PORTE соответственно. Биты регистра соответствуют линиям порта, причем если линия настроена на ввод информации, то при чтении регистра соответствующий бит примет состояние присутствующее на линии на момент чтения; если линия настроена на вывод, то установленное программно состояние бита управляет выходным состоянием соответствующей линии. Для настройки направления обмена линий портов C и D существуют регистры DDRC и DDRD соответственно. Если какой-либо бит регистра направления установлен в “0”, то соответствующая линия настраивается на ввод, и наоборот, состояние “1” программирует линию на вывод. Направление обмена линий 3 и 7 порта A задается битами DDRA3 и DDRA7 регистра PACTL.

Линии портов B и C кроме обычного ввода/вывода поддерживают режим параллельного обмена с сигналами квитирования. Кроме этого, если у микроконтроллера нет внутренней памяти (например, у MC68HC11E1 нет ПЗУ) или ее объем недостаточен, то через эти линии можно подключить внешнюю память. Режим работы с внешней памятью называется *расширенным мультиплексным режимом работы* микроконтроллера (о подробностях его установки см. описание на микроконтроллер). При работе в этом режиме для обеспечения обмена с внешней памятью через порты B и C выводится внутренняя магистраль адреса/данных микроконтроллера. Через порт B выводится старший байт адреса, а через порт C - данные и младший байт адреса. Для отделения данных от адреса порт C должен включаться через регистр, защелкивающий младший байт адреса по сигналу AS микроконтроллера. Вариант схемотехнической реализации внешней магистрали адреса/данных представлен на рис. 6.1. Кроме шины адреса (A0-A15) и шины данных (D0-D7) здесь показано формирование сигналов записи (WR) и чтения (RD).

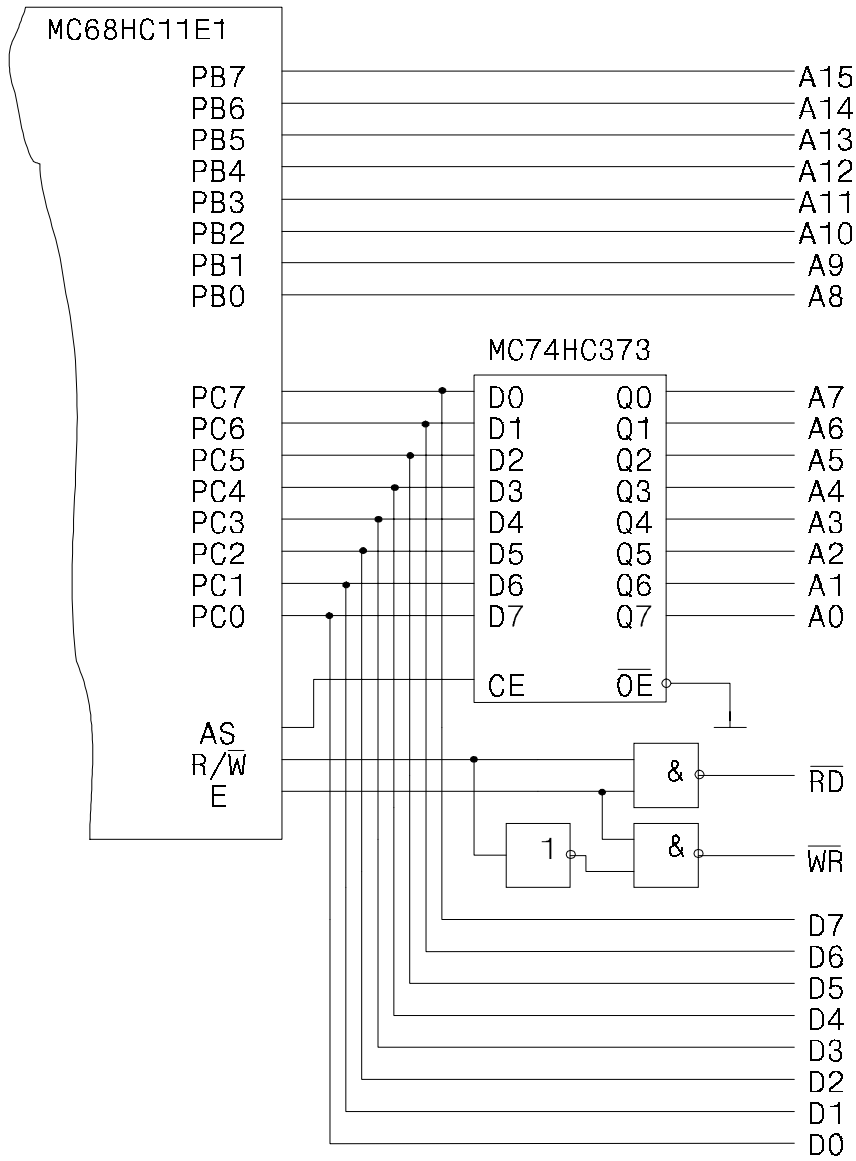


Рис. 6.1 Реализация внешней магистрали адреса/данных микроконтроллера MC68HC11E1.

В отладочном модуле HC11EVB порты B и C также используются для обмена с внешней памятью (ОЗУ емкостью 16 кбайт, занимающее адресное пространство $C000-$FFFF). Для того, чтобы компенсировать “потерю” портов B и C в модуле предусмотрена микросхема-эмулятор MC68HC24, которая имеет два порта, аналогичных по функциональным возможностям (подразумевается режим портов при работе с внутренней памятью) соответствующим портам микроконтроллера. Регистры управления этими портами также находятся внутри эмулятора и располагаются они в другом месте адресного пространства. Новые адреса этих регистров представлены ниже:

```
$xF02       PIOC                ; регистр управления параллельного
                                ; вводом/выводом
$xF03       PORTC               ; регистр данных порта C
$xF04       PORTB               ; регистр данных порта B
$xF05       PORTCL              ; альтернативный регистр порта C
$xF07       DDRC                ; регистр направления порта C
$xF3C       HPRIO               ; регистр высшего приоритета
                                ; I-прерывания
$xF3D       INIT                ; регистр организации памяти
```

Первая тетрада адреса определяется содержимым младших 4-х бит регистра INIT и после сброса принимает значение $01. Работа с регистрами эмулятора не отличается от работы с

аналогичными регистрами микроконтроллера, за исключением того, что те биты этих регистров, которые не задействованы для управления работой портов B и C, не используются. Таким образом, не используются старшая тетрада регистра INIT, и в регистре HPRIO используются только биты 4 (IRV) и 5 (SMOD). Более подробную информацию об эмуляторе см. описание на модуль HC11EVB.

Параллельные порты ввода/вывода B и C кроме режима обычного параллельного ввода/вывода, в котором информация записанная в соответствующий регистр порта выводится непосредственно через линии вывода, а информация присутствующая на линиях ввода может быть считана непосредственно из соответствующего регистра порта, поддерживают два дополнительных режима работы: простой стробируемый ввод/вывод и ввод/вывод с полной поддержкой сигналами квитирования. Рассмотрим особенности этих режимов:

Режим простого стробируемого ввода/вывода выбирается сбросом бита HNDS регистра PIOC (см. рис. 2). В этом режиме запись в порт C сторобируется сигналом по внешней линии STRA контроллера, аналогично порт B стробируется на вывод линией STRB. При обнаружении на линии STRA активного фронта (активный фронт строба A для всех режимов управляется состоянием бита EGA регистра PIOC, при EGA="0" активный фронт - отрицательный, при EGA="1" активный фронт - положительный) текущее состояние на линиях порта C копируется в регистр-защелку порта C PORTCL (адрес $1005) и в регистре PIOC устанавливается флаг стробирования STAF регистра PIOC. Если в регистре PIOC установлен бит разрешения прерываний по стробу A, производится запрос внутреннего прерывания (по IRQ). Сброс флага STAF производится при чтении регистра PIOC (с установленным флагом STAF) и последующим чтении регистра PORTCL, причем запись в регистр PORTCL производится независимо от того, был ли флаг STAF ранее очищен или нет. На линию STRB в режиме простого стробируемого ввода/вывода подается импульс длительностью 2 E цикла (E - внутренняя частота синхронизации контроллера) при каждой записи данных в порт B. Активный уровень импульса определяется состоянием бита INVB регистра PIOC. При INVB="0" активный уровень низкий, при INVB="1" активный уровень высокий.

	7	6	5	4	3	2	1	0	
$1002	STAF	STAI	CWOM	HNDS	OIN	PLS	EGA	INVB	**PIOC**
Состояние после сброса	0	0	0	0	0	-	1	1	

Рис. 6.2 Регистр управления параллельным вводом/выводом PIOC.

Режим параллельного ввода/вывода с полным квитированием установления связи использует порт C и линии STRA и STRB. Существуют два основных режима (ввод и вывод) и дополнительно к ним вариант режима вывода с квитированием установления связи, который допускает работу порта C в трех состояниях. Во всех режимах с полным квитированием установления связи вывод STRA является входом, работающим по фронту, а STRB - выходом.

В ***режиме ввода*** STRA - является линией управления записью в регистр PORTCL из внешней системы, а линия STRB - линия вывода сигнала готовности, управляемая логикой микроконтроллера.

При определении сигнала готовности, внешнее устройство помещает данные на линии порта C, после чего оно подает импульс на линию STRA. По активному фронту на линии STRA данные на входе порта C копируются в регистр PORTCL, устанавливается флаг STAF (не обязательно вызывая прерывание) и сбрасывается сигнал готовности на линии STRB. Сброс сигнала готовности автоматически препятствует стробированию внешними устройствами новых данных в порт C. Чтение порта C (независимо от состояния флага STAF) устанавливает сигнал готовности на линии STRB, сигнализируя о том, что в порт C можно записывать новые данные.

Линия STRB может быть настроена (битом управления PLS регистра PIOC) как импульсный выход (PLS="1", импульсный режим) или статический выход (PLS="0", статический режим).

Протокол режима ввода не оказывает никакого эффекта на использование линий порта С в качестве статических входов или статических выходов. Чтение регистра данных PORTC всегда возвращает логические уровни на выводах порта С (для линий настроенных на ввод). Запись как в регистр PORTC, так и в регистр PORTCL посылает информацию в выходной регистр порта С, не влияя на стробирование ввода с квитированием.

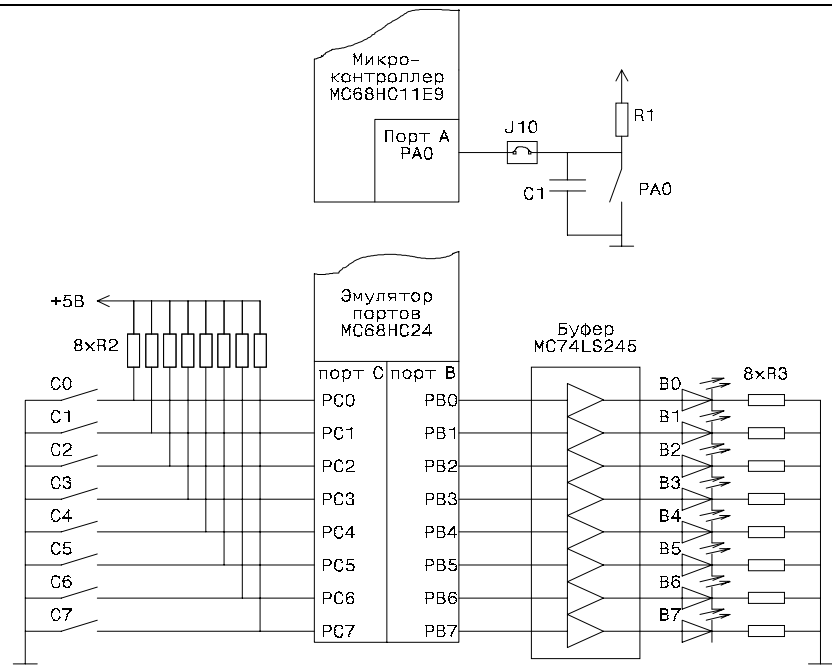
В ***режиме вывода*** на линию STRB выводится сигнал готовности и линия STRA является входной линией подтверждения (работающая по фронту), сообщающая микроконтроллеру о том, что выходные данные приняты внешним устройством. В разных вариантах этого протокола линия STRA используется также и в качестве линии разрешения вывода.

Микроконтроллер помещает данные на выходные линии порта С и подтверждает это действие сигналом на линии STRB. После этого внешнее устройство считывает данные с последующим подтверждением по линии STRA, означающим, что на порт С уже можно выводить новые данные. Активный перепад на линии STRA позволяет сбросить линию STRB и установить флаг STAF. В на установку флага STAF программа выводит через порт С новые данные. Запись данных в регистр PORTCL вызывает появление этих данных на выходных линиях порта С и подачу сигнала подтверждения по линии STRB.

Существуют варианты протокола вывода, которые допускают режим работы регистра С в третьем состоянии, то есть возможно прямое соединение этого 8-разрядного порта с другими устройствами, имеющими трехстабильные выходы без использования других вспомогательных схем.

Во время нахождения линии STRA в неактивном состоянии направление передачи на всех линиях порта С определяется содержимым регистра направления, то есть линии настроенные как входы находятся в высокоимпедансном состоянии. При активизации линии STRA все линии порта С становятся выходами независимо от содержимого регистра направления. Следует заметить, что в рассматриваемом протоколе чтение порта С всегда будет возвращать величину на входе выходных буферов порта, независимо от состояния регистра направления, поскольку эти линии могут не иметь значащих данных в варианте данного протокола с третьим состоянием. Эта особенность делает непрактичным использование некоторых линий порта С для ввода во время протокола вывода, но не мешает им являться статическими выходами.

Для учебных целей на плате отладочного модуля предусмотрены набор переключателей C7-C0 (см. рис.2 технического описания на модуль), подключенных к порту С эмулятора, кнопка PA0, подключенная к линии 0 порта А микроконтроллера и набор светодиодов B7-B0, подключенных к порту В. Схемотехнически это подключение представлено на следующем рисунке:



Представленные ниже эксперименты иллюстрируют возможности простого параллельного ввода вывода через порты B и C. При проведении экспериментов должна быть установлена перемычка J10 для возможности изменения состояния линии 0 порта A с помощью кнопки PA0.

С помощью команды отладчика “Memory Modify” откройте ячейку памяти $1000, соответствующую регистру данных порта A. При этом младший бит регистра должен быть установлен в “1” (т.к. кнопка PA0 не нажата и на линии 0 порта A присутствует высокий уровень). Нажмите кнопку PA0 и удерживая ее вновь откройте ячейку $1000. Убедитесь, что младший бит изменил свое значение на “0”, отобразив таким образом состояние линии 0 порта A при нажатой клавише PA0.

Откройте ячейку памяти $1F04, соответствующую регистру данных порта B. Запишите в нее произвольное число и убедитесь, что оно отобразилось в двоичном виде на светодиодах (горящий светодиод соответствует логической “1”, погашенный - “0”).

Рассмотрим программу, производящую опрос состояния переключателей порта C и запись полученной информации в порт B (на светодиоды) при нажатии на клавишу PA0.

```
            cpu   6811
porta       equ   $00           ; смещение для регистра данных порта A
portb       equ   $1f04          ; регистр данных порта B
portc       equ   $1f03          ; регистр данных порта C
ddrc        equ   $1f07          ; регистр направления порта C

            org   $d000
            ldy   #$1000         ; общее смещение для внутренних
                                 ; регистров
            clr   ddrc           ; настройка линий порта C на ввод
j1          brset porta,y,$01,j1 ; ожидание нажатия на PA0
            ldaa  portc          ; запись в аккумулятор A состояния
                                 ; переключателей
            staa  portb          ; вывод полученных данных в порт B
            bra   j1             ; переход на начало
```

Запустите программу с адреса $D000. Наберите на переключателях какое-нибудь двоичное число (логический “0” соответствует нижнему положению переключателя), нажимая клавишу PA0, убедитесь, что состояние переключателей отображается на светодиодах. Обратите внимание, что в начале программы производится настройка направлений обмена линий

порта С (логический “0”, записанный в определенный бит регистра направления программирует соответствующую ему линию ввода/вывода на ввод). Линия 0 порта А и линии порта В имеют фиксированные направления обмена и не требуют настройки.

Примечание. *При программировании линий порта С на вывод, переключатели, соответствующие этим линиям, должны быть установлены в верхнее положение (состояние логической “1”), иначе микросхема эмулятора может выйти из строя.*

В заключение рассмотрим программу реализующую мигание светодиодов с частотой 1 Гц и скважностью 3 (напомним, что скважностью называется отношение периода сигнала к длительности импульса). Для организации временных задержек воспользуемся тем свойством, что каждая команда микроконтроллера выполняется за фиксированное количество внутренних циклов синхронизации. Количество циклов выполнения для команд микроконтроллеров семейства M68HC11 представлено в соответствующей таблице в описании на микроконтроллер. Длительность цикла для микроконтроллера MC68HC11E1 определяется как 4/f, где f - частота внешнего кварцевого генератора. Для отладочной платы HC11EVB f=8 МГц и, таким образом, длительность одного цикла составляет 0.5 мкс. Следовательно, если команда выполняется за 3 цикла (например команда PSHA), то длительность ее выполнения составит 3 x 0.5= 1.5 мкс.

```
            cpu   6811
portb       equ   $1f04          ; регистр данных порта B

            org   $d000
            clra                ; обнулить счетчик
j1          bne   j2            ; если содержимое счетчика не равно 0,
                                ; то не зажигать светодиоды, иначе
            ldab  #$ff
            stab  portb         ; зажечь светодиоды
j2          ldy   #51282        ; цикл задержки на 1/3 секунды
loop        nop
            nop
            nop
            dey
            bne   loop
            clrb
            stab  portb         ; погасить светодиоды
            inca                ; увеличить содержимое счетчика на 1
            cmpa  #3            ; если его содержимое меньше 3,
            bne   j1            ; то переход на начало,
            clra                ; иначе обнулить счетчик
            bra   j1            ; и переход на начало
```

Рассмотрим работу программы. На аккумуляторе А реализован счетчик, увеличивающий свое значение на 1 через каждые 1/3 секунды. Значение счетчика меняется в диапазоне от 0 до 2, причем светодиоды горят только те 1/3 секунды, когда состояние счетчика равно 0 или одну треть общего периода, составляющего $3 \times 1/3$ с = 1 с. Задержка в 1/3 секунды реализована в цикле loop, который выполняется 51282 раз (это значение записывается в регистр Y перед началом выполнения цикла). Тело цикла состоит из трех команд nop (выполняется в течение 2-х циклов), команды dey (4 цикла) и команды bne (3 цикла). Таким образом каждый проход цикла выполняется за $2 \times 3 + 4 + 3 = 13$ циклов, или 13 x 0.5 мкс = 6.5 мкс. Весь же цикл будет пройден за 51282×6.5 мкс = 333333 мкс = 0.333333 с $\approx$ 1/3 с.

Запустите программу с адреса $D000. Попробуйте модифицировать частоту мигания и скважность.

3. Контрольные вопросы

1. Сколько параллельных портов ввода/вывода имеется у микроконтроллера MC68HC11E1? Перечислите их и те дополнительные функции, которые могут выполнять линии этих портов.
2. Что такое регистр данных параллельного порта? Как согласуется информация, записанная в регистре данных, и состояние соответствующих ему линий ввода/вывода?
3. Перечислите линии с фиксированным направлением обмена и двунаправленные линии. Каким образом задается направление обмена линии?
4. В каких регистрах задается направление портов C и D?
5. В каком регистре задается направление обмена линий 4 и 7 порта A?
6. Что такое расширенный мультиплексный режим работы микроконтроллера? Какие порты ввода/вывода используются для его реализации?
7. Для чего используется внешняя магистраль адреса/данных? Каким образом она реализуется для микроконтроллера MC68HC11E1?
8. Какой объем линейного адресного пространства может адресовать микроконтроллер MC68HC11E1?
9. Объясните назначение микросхемы-эмулятора MC68HC24.
10. В чем для программиста заключается разница при работе с портами B и C микроконтроллера и при работе с соответствующими портами эмулятора?
11. Каким образом вычисляются адреса для регистров эмулятора?
12. Опишите функционирование портов в режиме обычного параллельного ввода/вывода.
13. Что такое параллельный обмен при поддержке сигналов квитирования? Какие параллельные порты MC68HC11E1 поддерживают этот обмен?
14. Что такое режим простого стробируемого ввода/вывода? Как задействованы в этом режиме порты B и C MC68HC11E1?
15. Опишите протокол обмена в режиме полной поддержки сигналами квитирования при вводе информации.
16. Опишите протокол обмена в режиме полной поддержки сигналами квитирования при выводе информации.

4. Задания

1. Написать программу, которая в цикле выводит инвертированное состояние переключателей на светодиоды.
2. Написать программу, которая складывает состояние старшей и младшей тетрады переключателей и выводит результат на светодиоды (в цикле).
3. Написать программу, переводящую двоичное состояние трех младших переключателей в линейный код на светодиоды (т. е. если набрано число двоичное число 101, соответствующее десятичному 5, то должен загореться светодиод B5).
4. Модифицировать предыдущую программу так, чтобы загорался не только светодиод, соответствующий коду на переключателях, но и все светодиоды с номером меньше данного (т. е. для предыдущего примера должны будут загореться светодиоды B0, B1, B2, B3, B4 и B5).
5. Написать программу - “генератор случайных чисел”. Работать программа должна следующим образом: в цикле инкрементируется состояние какой-либо ячейки памяти (с большой скоростью), по нажатию на кнопку PA0 текущее состояние этой ячейки выводится на светодиоды.
6. Написать программу, реализующую эффект “бегущий огонь” на светодиодах (вначале зажигается один светодиод, через некоторое время он гаснет и зажигается соседний и т. д.). Направление и скорость перемещения выберете самостоятельно.

7. Модифицировать программу бегущего огня следующим образом: кратковременно вспыхивает светодиод B7, через 0.5 с кратковременно вспыхивает светодиод B6 и т. д. После вспыхивания светодиода B0 цикл повторяется.
8. Написать программу, которая по нажатию на кнопку PA0 увеличивала бы на 1 двоичное число, отображаемое на светодиодах. Исходное число должно считываться вначале программы с переключателей.
9. Написать программу, которая складывала бы два двоичных 8-разрядных числа и выводила бы результат на светодиоды. Работа программы должна производится следующим образом: первое число устанавливается на светодиодах, после чего нажимается клавиша PA0, далее устанавливается второе число и снова нажимается PA0, после чего выводится результат.
10. Модифицируйте предыдущую программу так, чтобы можно было просмотреть старший бит суммы. Он должен выводится на светодиод B0 (остальные должны быть погашены) по нажатию на PA0.
11. Модифицируйте программу 5. для умножения двух 8-разрядных двоичных чисел. При этом результат - старшие и младшие 8 разрядов, должны чередоваться на светодиодах по нажатию на PA0.
12. Написать программу, которая должна мигать синхронно всеми светодиодами (со скважностью 2), причем частота мигания должна задаваться на младшей тетраде переключателей следующим образом: f=N, где f - частота мигания в герцах, а N - число, установленное на младшей тетраде переключателей. Состояние переключателей должно считываться в цикле в процессе работы программы.
13. Написать программу “бегущий огонь”, скорость перемещения которого определялась бы как в предыдущей программе.
14. Написать программу, в которой изменялась бы яркость свечения светодиодов (яркость свечения всех светодиодов должна быть одинакова), в зависимости от состояния младшей тетрады переключателей. Управление яркостью должно производиться путем управления скважностью мигания. Частота мигания должна быть в пределах 25-100 Гц. Значение скважности однозначно соответствует двоичному числу, установленному на младшей тетраде переключателей.
15. Написать программу, которая в цикле плавно увеличивала бы яркость всех светодиодов до максимума, после чего тушила бы их.
16. Модифицировать предыдущую программу, чтобы после периода плавного увеличения яркости, следовал бы период плавного ее уменьшения.

Лабораторная работа №7

Прерывания.

1. Введение

В данной работе изучается система прерываний микроконтроллера MC68HC11E1.

2. Система прерываний.

Микроконтроллер MC68HC11E1 обладает гибкой системой обработки прерываний и исключительных ситуаций. Система обработки прерываний включает в себя ряд аппаратных источников (порты ввода/вывода, таймер, асинхронный и синхронный последовательные интерфейсы, внешние прерывания) и один программный (SWI). После возникновения запроса на прерывание микроконтроллер завершает выполнение текущей команды и передает управление по адресу, записанному в векторе прерывания, который соответствует источнику, сгенерировавшему запрос. В нормальных режимах работы (однокристальном и расширенном) вектора прерываний располагаются в фиксированной области памяти (см. табл. 7.1) и занимают по 2 байта. В специальных режимах работы (bootstrap и теста) таблица векторов прерывания переносится в область ОЗУ (см. табл. 7.1); каждый вектор здесь занимает 3 байта и в него должна быть записана команда перехода, а не адрес. Это отличие следует иметь ввиду при написании и отладке программ на модуле HC11EVB.

Таблица 7.1. Вектора прерываний для нормальных и специальных режимов работы.

Адрес вектора в нормальном режиме работы	Адрес перехода в специальном режиме работы	Источник прерывания
FFCO, FFC1	-	Зарезервировано
...	...	...
FFD4, FFD5	-	Зарезервировано
FFD6, FFD7	00C4	Последовательный асинхронный интерфейс (SCI)
FFD8, FFD9	00C7	Последовательный синхронный интерфейс (SPI)
FFDA, FFDB	00CA	Аккумулятор пульсаций
FFDC, FFDD	00CD	Переполнение аккумулятора пульсаций
FFDE, FFDF	00D0	Переполнение таймера
FFE0, FFE1	00D3	Выходное сравнение 5
FFE2, FFE3	00D6	Выходное сравнение 4
FFE4, FFE5	00D9	Выходное сравнение 3
FFE6, FFE7	00DC	Выходное сравнение 2
FFE8, FFE9	00DF	Выходное сравнение 1
FFEA, FFEB	00E2	Входной захват 3
FFEC, FFED	00E5	Входной захват 2
FFEE, FFEF	00E8	Входной захват 1
FFF0, FFF1	00EB	Прерывание реального времени
FFF2, FFF3	00EE	Прерывание IRQ
FFF4, FFF5	00F1	Прерывание XIRQ
FFF6, FFF7	00F4	Программное прерывание SWI
FFF8, FFF9	00F7	Неправильный код команды*
FFFA, FFFB	00FA	Система COP*
FFFC, FFFD	00FD	Сбой тактовой частоты*
FFFE, FFFF	BF40	Сброс микроконтроллера (RESET)*

* Группа исключительных ситуаций

	7	6	5	4	3	2	1	0	
$103C	RBOOT*	SMOD*	MDA*	IRV*	PSEL3	PSEL2	PSEL1	PSEL0	HPRIO
Состояние после сброса	-	-	-	-	0	1	0	1	

* Биты старшей тетрады, определяющей режим работы микроконтроллера

Рис. 7.1 Регистр высшего приоритета I-прерваний.

Таблица 7.2. Определение источника с наибольшим приоритетом, в зависимости от состояния битов PSEL3-PSEL0 регистра HPRIO.

PSEL3	PSEL2	PSEL1	PSEL0	Источник с наибольшим приоритетом прерывания
0	0	0	0	Переполнение таймера
0	0	0	1	Переполнение счетчика пульсаций
0	0	1	0	Счетчик пульсаций
0	0	1	1	SPI
0	1	0	0	SCI
0	1	0	1	Резерв (по умолчанию - IRQ)
0	1	1	0	IRQ
0	1	1	1	Прерывание реального времени
1	0	0	0	Входной захват 1
1	0	0	1	Входной захват 2
1	0	1	0	Входной захват 3
1	0	1	1	Выходное сравнение 1
1	1	0	0	Выходное сравнение 2
1	1	0	1	Выходное сравнение 3
1	1	1	0	Выходное сравнение 4
1	1	1	1	Выходное сравнение 5

При одновременном возникновении нескольких запросов обработка производится в соответствии со структурой приоритетов. Из аппаратных прерываний (программное прерывание SWI выполняется как команда и после начала выполнения никакие другие прерывания или исключительные ситуации, кроме RESET не могут прервать ее выполнение) высший приоритет имеет немаскируемое внешнее прерывание XIRQ, разрешаемое сбросом бита X в регистре условий CCR (установка этого бита может производиться только аппаратно - при возникновении прерывания по XIRQ или по сигналу сброса RESET). Остальные прерывания образуют группу I - прерываний, все они могут быть запрещены/разрешены путем соответственно установки/сброса бита I в регистре условий CCR (например командами SEI и CLI). В этой группе высший приоритет имеет прерывание, определяемое состоянием младшей тетрады регистра HPRIO (см. рис. 7.1 и табл. 7.2). Далее приоритет источников прерывания падает в следующем порядке:

прерывание IRQ
прерывание реального времени
входной захват 1
входной захват 2
входной захват 3
выходное сравнение 1
выходное сравнение 2
выходное сравнение 3
выходное сравнение 4

выходное сравнение 5
переполнение таймера
переполнение счетчика пульсаций
счетчик пульсаций
последовательный синхронный интерфейс (SPI)
последовательный асинхронный интерфейс (SCI)

Вызов прерывания влечет за собой запись основных регистров в стек, при этом порядок размещения регистров следующий:

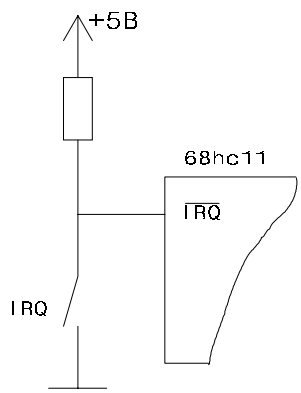
Стек

SP	Младший байт программного счетчика - PCL	←Указатель стека SP до прерывания
SP-1	Старший байт программного счетчика - PCH	
SP-2	Младший байт регистра Y - IYL	
SP-3	Старший байт регистра Y - IYH	
SP-4	Младший байт регистра X - IXL	
SP-5	Старший байт регистра X - IXH	
SP-6	Аккумулятор A - ACCA	
SP-7	Аккумулятор B - ACCB	
SP-8	Регистр условий - CCR	
SP-9		←Указатель стека SP после прерывания

После выполнения подпрограммы обслуживания прерывания состояния основных регистров восстанавливается из стека при выполнении команды возврата RTI.

Рассмотрим более подробно прерывание IRQ. Оно имеет два источника, один из которых внешний, другой - внутренний, от системы обслуживания параллельного ввода/вывода. В данной лабораторной работе мы будем рассматривать только внешнее IRQ - прерывание. При возникновении запроса на прерывание от какого-нибудь из источников происходит генерация прерывания по вектору IRQ (находится по адресу $FFF2 для нормальных режимов работы и $00EE - для специальных).

Вход прерывания IRQ микроконтроллера предназначен для обслуживания запросов на прерывание от внешних устройств. В зависимости от состояния бита IRQE регистра OPTION, обнаружение запроса производится либо по низкому уровню (IRQE="1"), либо по отрицательному перепаду (IRQE="0") на входе IRQ микроконтроллера. Для учебных целей на плате отладочного модуля предусмотрена кнопка IRQ (см. рис. NN технического описания на модуль), схемотехническое подключение которой представлено на следующем рисунке:



Рассмотрим пример обслуживания запроса прерывания от внешнего источника по линии IRQ. В данном примере при нажатии на клавишу IRQ производится инверсия состояния линий порта B (светодиодов).

```
            cpu   6811
portb       equ   $1f04          ; регистр данных порта B
option      equ   $39            ; смещение для регистра конфигурации

            org   $00ee          ; вектор прерывания по IRQ (режим
                                 ; bootstrap)
            jmp   $d100          ; переход на подпрограмму обслуживания
                                 ; прерывания

            org   $d000          ; Начало основной программы
            ldx   #$1000         ; общее смещение для внутренних
                                 ; регистров
            clr   portb          ; погасить светодиоды
            bclr  option,x,$20   ; установить режим обнаружения внешнего
                                 ; прерывания по уровню
            cli                  ; разрешить I-прерывания
            bra   *              ; замкнутый цикл

            org   $d100          ; Начало подпрограммы обслуживания
                                 ; прерывания
            ldaa  #$ff
            eora  portb
            staa  portb          ; инверсия состояния светодиодов
            jsr   dly06          ; задержка 0.6с
            rti                  ; возврат из прерывания

dly06       ldx   #$7f00         ; подпрограмма задержки 0.6с
j1          mul                  ; задержка 30 циклов
            mul
            mul
            dex
            bne   j1
            rts
```

Запустите программу с адреса $D000. Каждое следующее нажатие на клавишу IRQ воспринимается только через 0.6 с, причем, если клавишу оставить нажатой, то светодиоды будут мигать с периодичностью 0.6 с. Это объясняется тем, что после возвращения из подпрограммы обработки прерывания, возникает новый запрос, поскольку клавиша IRQ нажата и поддерживает на соответствующем входе низкий уровень.

Теперь заменим команду

```
            bclr  option,x,$20
```

на

```
            bset  option,x,$20
```

т.е. установим режим обнаружения прерывания ***по перепаду*** и запустим заново программу (вместо этого можно просто изменить соответствующим образом состояние регистра option, с помощью команды отладчика “Debug→Memory→Modify”). При работе модифицированной программы инверсия состояния светодиодов также происходит при нажатии на клавишу IRQ, которое воспринимается через 0.6 с после предыдущего нажатия, но если клавишу оставить нажатой, то мигания светодиодов не будет, т.к. при нажатии клавиши IRQ возникает только один отрицательный перепад.

Для удобства рассмотрения часть прерываний выделены в отдельную группу - группу исключительных ситуаций. К ней относятся: сброс микроконтроллера - RESET, система защиты от сбоев - COP, система обнаружения падения частоты синхронизации и система обработки неправильного кода команды.

Сброс микроконтроллера может быть произведен двумя путями: 1. по перепаду напряжения питания (пока напряжение питания не достигло необходимого уровня, внутри микроконтроллера устанавливается сигнал сброса RESET, который остается установленным еще 4064 тактовых цикла после достижения нормы); 2. подачей низкого уровня на вывод RESET микроконтроллера. После сброса микроконтроллер инициализирует внутренние регистры и начинает выполнять программу пользователя, начиная с адреса, находящегося по вектору $FFFE, $FFFF.

Система защиты от программных сбоев (COP) представляет собой таймер, который по истечении определенного времени передает управление программой по вектору $FFFA, $FFFB (при этом не производится сохранение основных параметров программы в стеке, в отличие от обычных прерываний). Для того, чтобы этого не произошло программа пользователя должна периодически инициализировать таймер COP, причем период между двумя инициализациями не должен превышать заданного значения (периода COP-таймера). Это значение определяется частотой внешней тактовой синхронизации и состоянием битов CR1 и CR0 (два младших бита) регистра OPTION. Следует иметь ввиду, что в нормальных режимах работы эти биты могут быть запрограммированы только в течение первых 64 E циклов работы микроконтроллера после сброса. В специальных режимах работы доступ к этим битам не ограничен (см. описание на микроконтроллер). Для внешней тактовой частоты 8 МГц соответствие между состоянием битов CR1 и CR0 и периодом COP - таймера представлено в табл. 7.3.

Таблица 7.3.

CR1	CR0	Период COP
0	0	16.384 мс
0	1	65.536 мс
1	0	262.14 мс
1	1	1.049 с

Разрешение работы COP производится путем сброса бита NOCOP (бит 2) регистра CONFIG (правила его программирования смотрите в описании на микроконтроллер). Инициализация COP осуществляется последовательной записью байтов $55 и $AA в регистр COPRST (адрес $103A).

Система обнаружения сбоя частоты синхронизации включается установкой бита CME (бит 3) регистра OPTION. Срабатывание системы происходит, если внутренняя частота синхронизации E падает ниже 10-200 кГц (при внешней частоте 8 МГц внутренняя составляет 2 МГц). После срабатывания управление передается по вектору $FFFC, $FFFD, без сохранения состояния контроллера в стеке (как и в случае с COP).

Из трех вышеперечисленных исключительных ситуаций высший приоритет имеет RESET и низший - COP.

Если в процессе выполнения программы встретилась команда с несуществующим кодом (например, $41), то происходит прерывание по вектору $FFF8, $FFF9. По процессу выполнения это прерывание аналогично программному прерыванию SWI, за исключением того, что при переходе на подпрограмму обслуживания прерывания в стек записывается адрес команды, вызвавшей прерывание по неправильному коду, а не адрес следующей команды. Рассмотрим пример программы, которая подсчитывает количество несуществующих команд в программе и выводит его в двоичном виде на светодиоды:

```
            cpu   6811
portb       equ   $1f04          ; регистр данных порта B
tmp         equ   0               ; дополнительная ячейка
```

```
        org   $00f7         ; вектор прерывания по неправильному
                           ; коду команды (режим bootstrap)
        jmp   $d100        ; переход на подпрограмму обслуживания
                           ; прерывания
        org   $d000        ; Начало основной программы
        clr   tmp          ; Обнулить счетчик неправильных команд
        nop
        nop
        nop
        nop
        fcb   $41          ; Код неправильной команды
        nop
        nop
        nop
        fcb   $62          ; Код неправильной команды
        fcb   $5b          ; Код неправильной команды
        nop
        ldaa  tmp
        staa  portb        ; вывести состояние счетчика на
                           ; светодиоды
        bra   *            ; замкнутый цикл

        org   $d100        ; Начало подпрограммы обслуживания
                           ; прерывания
        inc   tmp          ; Инкрементировать счетчик неправильных
                           ; команд
        tsy
        inc   8,y          ; Увеличить на 1 адрес возврата (чтобы
                           ; возврат был на следующую команду, а
                           ; не на вызвавшую прерывание)
        rti                ; возврат из прерывания
```

Примечание. *Программа-отладчик для работы с HC11EVB использует прерывание по неправильному коду команды для организации точек останова и трассировки, поэтому эти функции становятся недоступными при выполнении данной программы. В связи с этим после выполнения программы следует сбросить контроллер и заново восстановить связь.*

В качестве счетчика неправильных команд используется дополнительная ячейка tmp. В начале программы он инициализируется, а затем увеличивается на 1 после каждого следующего обнаружения неправильной команды. В качестве “нормальных команд” используются команды NOP. В конце программы содержимое счетчика выводится на светодиоды. Запустите программу с адреса $D000. На светодиодах должно появится двоичное число 00000011 (т.е. десятичное 3). Проверьте работоспособность программы меняя число неправильных команд.

3. Контрольные вопросы.

1. Назовите все основные источники прерывания микроконтроллера MC68HC11E1.
2. Что такое вектор прерывания?
3. Какие отличия между обработками прерывания в нормальных и специальных режимах работы?
4. Что такое программное прерывание SWI?
5. Как происходит обработка двух и более одновременно поступивших запросов на прерывание?
6. Какое из аппаратных прерываний имеет высший приоритет?
7. Как его можно запретить ? Разрешить?
8. Что такое группа I-прерываний?
9. Опишите структуру приоритетов в этой группе.
10. Какие два источника вызывают прерывание IRQ?
11. В чем отличие обнаружения прерывания по уровню и по перепаду? Каким образом задается метод обнаружения перепада для IRQ?
12. Что такое группа исключительных ситуаций?
13. Какие два сброса микроконтроллера вам известны?
14. Что такое COP? Как его разрешить?
15. Как задается период COP - таймера?
16. Как производится инициализация COP?
17. Как работает система обнаружения сбоя частоты синхронизации? Как ее разрешить?
18. Какое основное отличие существует при обработке исключительных ситуаций по RESET, COP и сбою частоты синхронизации и при обработке обычных прерываний?
19. Что происходит, если при выполнении программы встретилась команда с неправильным кодом?
20. Является ли прерывание по неправильному коду команды аналогом программного прерывания SWI? Объясните почему.
21. Назовите примеры неправильных кодов команды?

4. Задания.

1. Напишите программу увеличивающую на 1 двоичное число на светодиодах при каждом, сколь угодно быстром нажатии на клавишу IRQ.
2. Напишите программу, которая с частотой 1 Гц инкрементирует состояние на светодиодах при нажатой клавише IRQ.
3. Модернизируйте предыдущую программу так, чтобы состояние светодиодов не изменялось при кратковременном нажатии (нажатиях) на IRQ.
4. Напишите программу - генератор случайных чисел. При нажатой кнопке IRQ в некоторой ячейке памяти должен происходить быстрый перебор значений (например, путем инкрементирования содержимого ячейки), и после отпускания это значение должно выводится на светодиоды.
5. Напишите программу - определитель скорость реакции. Через некоторое время после запуска программы (1-3 с) должен загореться старший светодиод. После этого запускается счетчик, останавливающийся после нажатия на кнопку IRQ. После этого на светодиоды в двоичном виде должно выводится его содержимое. Период счета следует сделать 1 мс. Таким образом на светодиодах отобразится двоичное время реакции в микросекундах. Фальшстарт индицируется миганием светодиодов.
6. Напишите программу, которая по нажатии на кнопку PA0 увеличивала бы состояние на светодиодах на 1, а по нажатию на IRQ - уменьшала бы.
7. Напишите программу, выводящую на светодиоды состояние переключателей при нажатии на IRQ.

8. Модернизируйте предыдущую программу так, чтобы при удерживаемой кнопке IRQ происходил циклический сдвиг состояния светодиодов с периодом 0.6 с.
9. Напишите программу, которая определяла бы определяла бы из двух игроков одного - с лучшей реакцией. Через некоторое время после запуска программы (1-3 с) должен загореться старший светодиод B7. Далее один человек нажимает кнопку PA0, а другой IRQ. Если первой была нажата PA0, загорается старшая тетрада светодиодов, если IRQ - младшая. Фальшстарт индицируется миганием светодиода B6 для игрока на PA0, и B1 - для игрока на IRQ.
10. Модернизируйте предыдущую программу так, чтобы на светодиоды выводилось разностное время реакции игроков (на семь младших светодиодов) в микросекундах. Старший светодиод указывает на победителя (зажженный светодиод означает, что победил участник на PA0).
11. Напишите программу, которая зажигала бы все светодиоды, если одновременно нажаты кнопки PA0 и IRQ и гасила бы их в противном случае.
12. Напишите программу состоящую из команд NOP с кодами неправильных команд. При обнаружении неправильной команды программа должна выводить на светодиоды старший и младший байты адреса по которому находится неправильная команда. Первым должен отображаться старший байт. После нажатия на кнопку PA0 - младший, и после следующего нажатия на PA0 - программа должна продолжить выполнение.
13. Модернизируйте предыдущую программу так, чтобы переключение байтов и выход в основную программу производился при нажатии кнопки IRQ.
14. Модернизируйте предыдущую программу так, чтобы переключение байтов и выход в основную программу производился только при одновременном нажатии кнопок IRQ и PA0.
15. Используя обработчик прерывания от программы 11. Модернизируйте любую из программ 1 ... 10, так, чтобы при появлении в ней неправильного кода, он индицировался бы, как в программе 11.
16. Напишите программу, которая просчитывала бы количество всех возможных неправильных команд для микроконтроллера M68HC11.

Лабораторная работа №8

Система таймера.

1. Введение

В данной работе изучается таймер микроконтроллера MC68HC11E1 и функционально связанные с ним устройства.

2. Система таймера

Встроенный таймер обеспечивает работу нескольких внутренних систем микроконтроллера, в число которых входят:

- система обеспечения функции входного захвата;
- система обеспечения функции выходного сравнения;
- прерывания реального времени;
- счетчик пульсаций.

Основу таймера составляет 16-разрядный счетчик, источником синхронизации которого является выход предделителя частоты E (эта частота в 4 раза меньше внешней частоты синхронизации и в отладочном модуле HC11EVB составляет 2 МГц). В предделителе можно программно задать один из четырех коэффициентов деления: 1, 4, 8 или 16. Установка нужного коэффициента производится посредством битов PR1 и PR0 регистра TMSK2 (см. рис. 8.4). Соответствие между состоянием этих битов и коэффициентом деления приведено в табл. 8.1.

Таблица 8.1.

PR1	**PR0**	Коэффициент деления
0	0	1*
0	1	4
1	0	8
1	1	16

* Автоматически устанавливается после сброса.

В нормальных режимах работы микроконтроллера модификация этих битов возможна только в течение первых 64 циклов E после сброса. В специальных режимах работы этого ограничения нет.

Состояние счетчика инкрементируется на 1 с каждым импульсом синхронизации. Это состояние можно прочитать из 16-битного регистра TCNT, представленного в виде двух 8-битных регистров, находящихся по адресу $100E (старший байт) и $100F (младший байт). Регистр TCNT не доступен для модификации (исключение составляет режим теста) и таким образом пользователь может только считывать его состояние. При переходе состояния таймера от $FFFF к $0000 устанавливается флаг переполнения таймера TOF (бит 7) регистра флагов таймера TFLG2 (см. рис. 8.3.). Прерывание по установке этого бита можно разрешить установкой бита TOI регистра маскирования прерываний TMSK2.

Рассмотрим пример использования прерывания по переполнению таймера для реализации программы динамического управления светодиодами - бегущих огней. Запишите в память следующую подпрограмму:

```
            cpu   6811
tflg2       equ   $25           ; регистр флагов таймера 2

            org   $00d0         ; вектор прерывания по переполнению
                                ; таймера
            jmp   to            ; переход на подпрограмму обслуживания
                                ; прерывания
```

```
            org   $d100          ; начало подпрограммы обслуживания
                                  ; прерывания
to          ldy   #$1000
            bset  tflg2,y,#%10000000    ; сброс флага переполнения
                                        ; таймера

            ldaa  $1f04           ; циклический сдвиг влево
            lsla                  ; состояния светодиодов
            bcc   j1
            oraa  #1
j1          staa  $1f04

            rti                   ; возврат из прерывания
```

Запишите в регистр порта B (адрес $1f04) какое-либо число, например, $49. При этом оно должно отобразиться на светодиодах. Запишите по адресу $1024 (регистр TMSK2) число $83 (%10000011), установив таким образом коэффициент деления для таймера 16 (при этом частота переполнений таймера будет составлять 2000000/(65535×16) ≈ 1.9 Гц) и разрешив прерывания по переполнению таймера. При этом на светодиодах будет происходить циклическое перемещение исходного состояния влево. Попробуйте изменить коэффициент предделителя, обратите внимание на изменение скорости переполнения таймера.

Примечание. В предлагаемой лабораторной работе после выполнения каждого примера желательно производить сброс микроконтроллера с последующим восстановлением связи для инициализации состояния внутренних регистров.

3. Функция входной фиксации

Функцию входной фиксации обеспечивают 16-битовые регистры, доступные только для чтения. При обнаружении заданного перепада на входной линии одного из каналов входной фиксации, происходит запись содержимого счетчика таймера в соответствующий регистр входной фиксации. Также при этом устанавливается соответствующий флаг в регистре флагов TFLG1 (первому каналу входной фиксации соответствует бит 2, второму - бит 1, третьему - бит 0 и четвертому - бит 3, см. рис. 8.1.) и происходит генерация прерывания, если установлен соответствующий бит регистра TMSK1 (см. рис. 8.2.).

	7	6	5	4	3	2	1	0	
$1023	OC1F	OC2F	OC3F	OC4F	I4O5F	IC1F	IC2F	IC3F	**TFLG1**
Состояние после сброса	0	0	0	0	0	0	0	0	

Рис. 8.1. Регистр флагов прерывания TFLG1.

	7	6	5	4	3	2	1	0	
$1022	OC1I	OC2I	OC3I	OC4I	I4O5I	IC1I	IC2I	IC3I	**TMSK1**
Состояние после сброса	0	0	0	0	0	0	0	0	

Рис. 8.2. Регистр маскирования прерывания TMSK1.

	7	6	5	4	3	2	1	0	
$1025	TOF	RTIF	PAOVF	PAIF	0	0	0	0	**TFLG2**
Состояние после сброса	0	0	0	0	0	0	0	0	

Рис. 8.3. Регистр флагов прерывания TFLG2.

	7	6	5	4	3	2	1	0	
$1024	TOI	RTII	PAOVI	PAII	0	0	PR1	PR0	**TMSK2**
Состояние после сброса	0	0	0	0	0	0	0	0	

Рис. 8.4. Регистр маскирования прерывания TMSK2.

	7	6	5	4	3	2	1	0	
$1021	EDG4B	EDG4A	EDG1B	EDG1A	EDG2B	EDG2A	EDG3B	EDG3A	**TCTL2**
Состояние после сброса	0	0	0	0	0	0	0	0	

Рис. 8.5. Регистр управления таймером TCTL2.

Входами четырех каналов входной фиксации могут служить линии 0-3 порта А. Первые три линии порта А (PA0-PA2) соответствуют каналам IC3, IC2 и IC1 входной фиксации, линия PA3 , в зависимости от состояния бита I4/O5 регистра PACTL (см. рис. 8.10), может использоваться для обеспечения функции входной фиксации IC4 (I4/O5 = “1”) или выходного сравнения OC5 (I4/O5 = “0”).

Перепад, вызывающий активизацию функции входной фиксации для заданного канала кодируется в регистре управления TCTL2 (см. рис. 8.5). Каждая пара линий EDGxB и EDGxA соответствует каналу входной фиксации x. В табл. 8.2 представлено соответствие между состоянием пары и видом активного перепада на соответствующей линии.

Таблица 8.2

EDGxB	EDGxA	Конфигурация
0	0	Функция вводного захвата запрещена
0	1	Захват только по положительному фронту входного сигнала
1	0	Захват только по отрицательному фронту входного сигнала
1	1	Захват только по любому фронту входного сигнала

Рассмотрим пример программы, использующей функцию входной фиксации для определения быстроты реакции человека. Правила работы с программой следующие: после запуска программы необходимо дождаться зажигания светодиода B0 и затем, как можно быстрее нажать на кнопку PA0. После этого на светодиодах появится (в двоичном коде) время реакции в сотых долях секунды. В случае фальшстарта загорятся все светодиоды.

```
            cpu   6811
porta       equ   $00          ; регистр порта A
tic3        equ   $14          ; регистр входной фиксации 3
tflg1       equ   $23          ; регистр флагов таймера 1
tflg2       equ   $25          ; регистр флагов таймера 2
tctl2       equ   $21          ; регистр управления таймера 2
tmsk2       equ   $24          ; регистр маскирования прерываний от
                               ; таймера 2
tcnt        equ   $0e          ; регистр счетчика таймера
```

```
tmp         equ   $00           ; дополнительная ячейка

            org   $00d0         ; вектор прерывания по переполнению
                                ; таймера
            jmp   to            ; переход на подпрограмму обслуживания
                                ; прерывания

            org   $d000         ; начало основной программы
            ldy   #$1000
            clr   $1f04         ; погасить светодиоды

            ldd   tcnt,y        ; определить псевдослучайную величину
                                ; начальной задержки
            andb  #$1f
            orab  #$40
            stab  tmp

            ldaa  #%10000000
            staa  tmsk2,y       ; разрешить прерывания по переполнению
                                ; таймера, входной коэффициент деления
                                ; для таймера - 1

            cli                 ; разрешить I-прерывания

j1          brclr porta,y,1,j3  ; Задержка на 2...3 секунды с проверкой
                                ; на фальшстарт
            ldab  tmp
            bne   j1

            ldaa  #%00000011
            staa  tmsk2,y       ; установить входной коэффициент
                                ; деления для таймера 16 (следует иметь
                                ; в виду, что установка коэффициента
                                ; деления в нормальных режимах работы
                                ; должна производится в течение 64
                                ; первых циклов E после сброса)

            ldaa  #$1
            staa  tflg1,y       ; сброс флага входного захвата 3
            staa  $1f04         ; зажечь светодиод B0
            inca
            staa  tctl2,y       ; разрешить входной захват 3 по
                                ; отрицательному фронту

            ldd   tcnt,y        ; записать текущее состояние таймера
            std   tmp           ; в ячейке tmp

j2          brset tflg1,y,#%00000001,j4 ; выход, если произошел захват
            ldd   tcnt,y        ; цикл на обнаружения превышения 0.5 с
            inca
            cmpa  tmp
            bne   j2
```

```
j3          ldaa  #$ff          ; зажечь все светодиоды в случае
                                ; фальшстарта или при превышении 0.5 с
            staa  $1f04
            bra   *

j4          ldd   tic3,y        ; расчет задержки
            suba  tmp
            tab
            clra
            ldx   #5
            idiv
            xgdx
            stab  $1f04         ; отобразить задержку на светодиоды
            bra   *

to          ldaa  #$80
            staa  tflg2,y       ; сброс флага переполнения таймера
            dec   tmp           ; уменьшить счетчик на 1
            bne   to_1
            clr   tmsk2,y       ; запретить прерывания по переполнению
                                ; таймера
to_1        rti                 ; возврат
```

Рассмотрим более подробно работу программы. Для осуществления задержки на 2...3 секунды используются прерывания по переполнению таймера, которые происходят с частотой 2000000/65536 ≈ 30.51Гц (коэффициент деления устанавливается равным 1). Следовательно для задержки ≈ на 1 сек требуется возникновение 31 прерывания по переполнению. Исходное число требуемых прерываний определяется следующим образом: после старта программы считывается состояние регистра счетчика таймера (TCNT). Поскольку это значение непрерывно меняется с момента аппаратного сброса, на момент старта программы там находится некоторое псевдослучайное значение. У младшего байта счетчика, записанного в аккумуляторе B, обнуляются старшие три бита, таким образом в аккумуляторе B содержится псевдослучайная величина задержки на 0 ... 32 цикла, или на 0 ... 1.05 секунды. Для смещения этой величины на ≈ 2 секунды, к содержимому B добавляется 64 (%01000000). Содержимое регистра B сохраняется в ячейке tmp, после чего разрешаются прерывания по переполнению таймера. В подпрограмме обслуживания прерывания (to) значение tmp уменьшается на 1 и проверяется на 0. Если значение tmp обнулилось (т.е. произошло заданное количество прерываний), то прерывания по переполнению запрещаются. В цикле ожидания задержки j1 постоянно проверяется не нажата ли кнопка PA0; в этом случае индицируется фальшстарт. Далее после определения и разрешения входного захвата 3 (соответствует линии PA0) зажигается светодиод B0, частота синхронизации таймера уменьшается в 16 раз (т.е. становится равной 2000000/16 = 125000Гц) и запускается цикл ожидания до возникновения входного захвата 3 (по анализу бита 0 регистра TFLG1). Кроме этого, перед началом цикла запоминается старшая тетрада счетчика таймера, которая сравнивается в цикле с текущим значением старшей тетрады счетчика для определения превышения допустимого времени, которое определяется как 16 × 65536/2000000 ≈ 0.524 с (для того, чтобы исключить срабатывание индикатора превышения сразу после входа в цикл текущее значение счетчика увеличивается на 1). После нажатия на клавишу PA0 происходит входная фиксация, вычисляется разность между старшими байтами конечного и начального содержимого счетчика таймера. При этом каждый разряд результата соответствует 256/125000 ≈ 0.002 с. Результат делится на 5 и выводится на светодиоды. Программа запускается с адреса $D000.

4. Функция выходного сравнения

Каждый канал выходного сравнения также имеет связанный с ним 16-битовый регистр, доступный для чтения и записи. При совпадении значения, записанного в этот регистр со значением счетчика таймера устанавливается соответствующий флаг (OCxF) в регистре TFLG1 (рис. 8.1) и происходит изменение состояния соответствующей выходной линии. Это изменение для линий 2 - 5 выходного сравнения кодируется в регистре TCTL1 (см. рис. 8.6). Соответствие между битами OMx и OLx и изменением на соответствующей линии представлено в табл. 8.3.

Таблица 8.3.

OMx	**OLx**	**Действие при совпадении**
0	0	Таймер отключен от выходных линий
0	1	Инверсия состояния соответствующей выходной линии OCx
1	0	Установить в “0” соответствующую линию OCx
1	1	Установить в “1” соответствующую линию OCx

	7	6	5	4	3	2	1	0	
$1020	OM2	OL2	OM3	OL3	OM4	OL4	OM5	OL5	**TCTL1**
Состояние после сброса	0	0	0	0	0	0	0	0	

Рис. 8.6. Регистр управления таймером TCTL1.

	7	6	5	4	3	2	1	0	
$100B	FOC1	FOC2	FOC3	FOC4	FOC5	0	0	0	**CFORC**
Состояние после сброса	0	0	0	0	0	0	0	0	

Рис. 8.7. Регистр форсирования выходного сравнения CFORC.

	7	6	5	4	3	2	1	0	
$100C	OC1M7	OC1M6	OC1M5	OC1M4	OC1M3	0	0	0	**OC1M**
Состояние после сброса	0	0	0	0	0	0	0	0	

Рис. 8.8. Регистр маскирования выходного сравнения 1 OC1M.

	7	6	5	4	3	2	1	0	
$100D	OC1D7	OC1D6	OC1D5	OC1D4	OC1D3	0	0	0	**OC1D**
Состояние после сброса	0	0	0	0	0	0	0	0	

Рис. 8.9. Регистр маскирования выходного сравнения 1 OC1D.

При успешном сравнении также может генерироваться прерывание, если установлен соответствующий бит OCxI в регистре TMSK1 (рис. 8.2). Запрограммированное действие на линии выходного сравнения можно форсировать, если записать “1” в соответствующий бит регистра CFORC (рис. 8.7).

В отличие от остальных каналов выходного сравнения, выходное сравнение 1 может автоматически влиять на любую из пяти выходных линий выходного сравнения (линии 3-7 порта А). При этом задействуются регистры OC1M (рис. 8.8) и OC1D (рис. 8.9). Регистр OC1M определяет какая линия порта А будет подвержена изменению в результате успешного OC1 сравнения. В регистр OC1D записываются данные, передаваемые на соответствующие линии в результате успешного сравнения. В случае одновременного воздействия на одну линию двух выходных сравнений выходное сравнение OC1 имеет высший приоритет.

Рассмотрим пример программы, реализующей управление скважностью мигания светодиодов. Задание скважности производится на старшей тетраде переключателей. Период мигания синхронизирован с переполнением таймера. При этом в момент переполнения таймера (происходит с частотой приблизительно 0.5 с) происходит зажигание светодиода PB3, а в момент совпадения содержимого таймера с содержимым регистра выходного сравнения OC3 (старшая тетрада которого совпадает с состоянием старшей тетрады переключателей, а младшие равны 0) светодиод гаснет. И таким образом чем больше число набрано на старшей тетраде переключателей, тем выше длительность вспышки светодиода (за исключением комбинации $00-в этом случае свечение непрерывное). В данной программе не используются линии выходного сравнения, а работа осуществляется по прерыванию. Запуск программы должен производиться с адреса $D000.

```
            cpu   6811
tmsk1       equ   $22           ; регистр маскирования прерываний от
                                 ; таймера 1
tmsk2       equ   $24           ; регистр маскирования прерываний от
                                 ; таймера 2
tflg1       equ   $23           ; регистр флагов таймера 1
tflg2       equ   $25           ; регистр флагов таймера 2
toc3        equ   $1a           ; регистр выходного сравнения 3

            org   $00d0         ; вектор прерывания по переполнению
                                 ; таймера
            jmp   to            ; переход на подпрограмму обслуживания
                                 ; прерывания
            org   $00d9         ; вектор прерывания по выходному
                                 ; сравнению 3
            jmp   oc3           ; переход на подпрограмму обслуживания
                                 ; прерывания
            org   $d000         ; начало основной программы
            ldy   #$1000
            ldaa  #%00100001
            staa  tmsk1,y       ; разрешить прерывания по выходному
                                 ; сравнению 3
            ldaa  #%10000011
            staa  tmsk2,y       ; разрешить прерывания по переполнению
                                 ; таймера; установить входной
                                 ; коэффициент деления для таймера 16
            cli                 ; разрешить I-прерывания
            bra   *             ; замкнутый цикл
to          bset  tflg2,y,#%10000000    ; сброс флага переполнения
                                 ; таймера
            ldaa  #$ff
            staa  $1f04         ; зажечь светодиоды
            rti                 ; возврат
```

```
oc3         bset  tflg1,y,#%00100000    ; сброс флага выходного
                                        ; сравнения 3
            ldaa  $1f03                 ; ввести новое значение
                                        ; выходного сравнения
            anda  #$f0                  ; с переключателей (старшая
                                        ; тетрада)
            staa  toc3,y
            clra
            staa  toc3+1,y
            clr   $1f04                 ; погасить светодиоды
            rti                         ; возврат
```

5. Прерывания реального времени

Работа системы прерываний реального времени (RTI) управляется состоянием битов RTR1 и RTR0 регистра PACTL (см. рис. 8.10).

	7	6	5	4	3	2	1	0	
$1026	DDRA7	PAEN	PAMOD	PEDGE	DDRA3	I4/O5	RTR1	RTR0	**PACTL**
Состояние после сброса	0	0	0	0	0	0	0	0	

Рис. 8.10. Регистр управления аккумулятором пульсаций PACTL.

В зависимости от состояния этих битов меняется период возникновения прерывания реального времени. Для частоты внешнего кварцевого генератора 8 МГц это соответствие представлено в табл. 8.4.

Таблица 8.4.

RTR1	RTR0	Период RTI
0	0	4.10 мс
0	1	8.19 мс
1	0	16.38 мс
1	1	32.77 мс

После истечении заданного периода RTI устанавливается флаг RTIF в регистре TMSK2, и если установлен флаг RTII регистра TFLG2, то происходит генерация прерывания.

Следующая подпрограмма реализует инверсию состояния светодиодов через каждые 32 цикла RTI. Система прерываний реального времени программируется на вывод запроса с периодичностью в 32.77 мс. После каждого запроса происходит увеличение содержимого дополнительного счетчика на 1, после 32 запросов производится инверсия состояния младшего бита порта B и таким образом светодиоды мигают с частотой 0.5 Гц.

```
            cpu   6811
tmsk2       equ   $24             ; регистр маскирования прерываний от
                                   ; таймера 2
tflg2       equ   $25             ; регистр флагов таймера 2
pactl       equ   $26             ; регистр управления аккумулятором
                                   ; пульсаций
tmp         equ   $00             ; дополнительная ячейка

            org   $00eb           ; вектор прерывания по реальному
                                   ; времени
            jmp   rt              ; переход на подпрограмму обслуживания
                                   ; прерывания
            org   $d000           ; начало основной программы
```

```
            ldy   #$1000
            ldaa  #%00000011
            staa  pactl,y       ; установить период прерывания
                                ; реального времени 32.77 мс
            ldaa  #%01000000
            staa  tmsk2,y       ; разрешить прерывания по системе
                                ; реального времени
            cli                 ; разрешить I-прерывания
            bra   *             ; замкнутый цикл
rt          bset  tflg2,y,#%01000000    ; сброс флага запроса
                                ; прерывания реального времени
            inc   tmp           ; увеличить на 1 содержимое
                                ; дополнительного счетчика
            ldaa  tmp           ; оценка только младших 6 бит
            anda  #$1f
            bne   rt1           ; если равен нулю, то
            ldaa  $1f04         ; инверсия состояния светодиодов
            eora  #%11111111
            staa  $1f04
rt1         rti                 ; возврат
```

Запуск программы производится с адреса $D000.

6. Аккумулятор пульсаций

Аккумулятор пульсаций представляет собой 8-разрядный счетчик доступный для чтения и записи, который может работать в двух режимах (в зависимости от состояния бита PAMOD регистра PACTL): подсчет внешних событий (PAMOD = 1) или управление накоплением частоты E/64 (PAMOD = 0), где E - внутренняя частота синхронизации. Управление аккумулятором пульсаций производится по линии PA7 порта A, причем не имеет значения настроена она на ввод или на вывод (настройка на ввод или вывод осуществляется соответственно сбросом или установкой бита DDRA7 регистра PACTL). Задание вида воздействия, управляющего аккумулятором производится с помощью битов PAMOD и PEDGE регистра PACTL. Зависимость между состоянием этих битов и видом воздействия представлено в табл. 8.5.

Таблица 8.5.

PAMOD	PEDGE	Вид воздействия
0	0	Отрицательный фронт на входе инкрементирует счетчик
0	1	Положительный фронт на входе инкрементирует счетчик
1	0	“0” на входе запрещает счет тактовых импульсов
1	1	“1” на входе запрещает счет тактовых импульсов

Рассмотрим программу управления яркостью горения светодиодов. При этом прерывание по переполнению счетчика пульсаций используется для управления скважностью импульсов поступающих на светодиоды. Для организации длительности импульса используется дополнительный счетчик. Скважность импульсов зависит от состояния младшей тетрады переключателей. Поскольку частота сигнала на светодиоде достаточно велика, то при изменении состояния переключателей меняется яркость горения светодиода PB1 (чем больше число, тем больше яркость). Счетчик пульсаций программируется на синхронизацию от внутренней частоты синхронизации E/64. Запуск программы осуществляется с адреса $D000.

```
            cpu   6811
porta       equ   $00           ; регистр данных порта A
tmsk2       equ   $24           ; регистр маскирования прерываний от
                                ; таймера 2
```

```
tflg2       equ   $25            ; регистр флагов таймера 2
pactl       equ   $26            ; регистр управления счетчиком
                                 ; пульсаций
pacnt       equ   $27            ; регистр накопления аккумулятора
                                 ; пульсаций
tmp         equ   $00            ; дополнительная ячейка

            org   $00cd          ; вектор прерывания по переполнению
                                 ; счетчика импульсов
            jmp   pao            ; переход на подпрограмму обслуживания
                                 ; прерывания

            org   $d000          ; начало основной программы
            ldy   #$1000
            ldaa  #%11100000
            staa  pactl,y        ; разрешить счетчик пульсаций
                                 ; (внутренняя синхронизация, разрешение
                                 ; счета по PA7=1), настроить PA7 на
                                 ; вывод
            ldaa  #%00100000
            staa  tmsk2,y        ; разрешить прерывания по переполнению
                                 ; счетчика пульсаций

            bset  porta,y,#%10000000 ; разрешить накопление счетчика
                                 ; пульсаций

            cli                  ; разрешить I-прерывания
            bra   *              ; замкнутый цикл

pao         bset  tflg2,y,#%00100000 ; сброс флага переполнения
                                 ; счетчика
            ldaa  #$f0
            staa  pacnt,y        ; установить 16 циклов до переполнения
            dec   tmp            ; уменьшить на 1 значение счетчика
                                 ; переполнений
            bne   pao1           ; если равен 0, то
            ldaa  #$10
            staa  tmp            ; инициализировать значение счетчика
            clr   $1f04          ; погасить светодиоды
            rti                  ; и возврат
pao1        ldaa  $1f03          ; иначе ввести состояние переключателей
            anda  #$0f           ; выделить младшую тетраду
            cmpa  tmp            ; сравнить со счетчиком
            bne   pao2           ; если совпадение, то
            ldaa  #$ff
            staa  $1f04          ; зажечь светодиоды
pao2        rti                  ; возврат
```

7. Контрольные вопросы

1. Какие системы микроконтроллера управляются внутренним таймером?
2. Какой элемент лежит в основе таймера?
3. Как производится тактирование счетчика таймера? Как управлять частотой тактирования?
4. Что происходит при переполнении таймера?
5. Опишите функцию входной фиксации.
6. Сколько существует каналов входной фиксации и с какими внешними линиями они связаны?
7. Какие события вызывают активизацию каналов входной фиксации? Как задается вид активного перепада на входе линии входной фиксации?
8. Опишите работу функции выходного сравнения.
9. Сколько существует каналов выходного сравнения и с какими внешними линиями они связаны?
10. Какие действия происходят при совпадении содержимого таймера и регистра выходного сравнения? Как запрограммировать эти действия?
11. Опишите отличие линии выходного сравнения OC1.
12. Какая информация хранится в регистрах OC1M и OC1D?
13. Опишите работу системы прерывания реального времени (RTI)?
14. Как задается период RTI?
15. Опишите работу аккумулятора пульсаций.
16. По какой внешней линии происходит управление аккумулятором пульсаций? Как вы думаете, какие отличия, относительно аккумулятора пульсаций, существуют при программировании этой линии на ввод и на вывод?
17. Как установить вид управляющего воздействия на аккумулятор пульсаций? Перечислите виды управляющего воздействия.
18. Перечислите прерывания, которые могут возникнуть при работе систем, связанных с таймером.
19. Перечислите и охарактеризуйте флаги регистров TFLG1 и TMSK1.
20. Перечислите и охарактеризуйте флаги регистров TFLG2 и TMSK2.

8. Задания

1. Напишите программу, реализующую мигания светодиодов с периодом от 0.1 до 1.6 секунд, период должен задаваться состоянием младшей тетрады переключателей (дискрет для периода 0.1 с).
2. Напишите программу, реализующую функцию входного захвата следующим образом: при нажатии на кнопку PA0 состояние переключателей должно переписываться на светодиоды.
3. Напишите программу, в которой частота мигания светодиодов (светодиоды должны мигать одновременно) увеличивалась бы линейно от 0.5 Гц до 16 Гц за 5 сек, после чего цикл повторяется.
4. Напишите программу, которая позволяла бы менять яркость свечения светодиодов посредством функции выходного сравнения. Яркость должна управляться состоянием младшей тетрады переключателей.
5. Модифицируйте предыдущую программу так, чтобы старшая тетрада переключателей управляла бы частотой мигания светодиодов (аналогично заданию 1), а младшая - яркостью.
6. Напишите программу, заставляющую светиться светодиоды с разной яркостью. При этом яркость свечения должна линейно возрастать от светодиода B7 (погашен) к светодиоду B0 (максимальная яркость).

7. Используя функцию входного захвата реализуйте генератор случайных чисел в диапазоне 0...255. Случайное число должно индицироваться на светодиодах после нажатия кнопки PA0.
8. Модифицируйте программу 3 так, чтобы яркость свечения первой тетрады светодиодов зависела от состояния первой тетрады переключателей, и соответственно яркость свечения второй - от второй тетрады переключателей.
9. Модифицируйте программу 6 так, чтобы одновременно реализовывался эффект “бегущий огонь” (т.е. последовательное перемещение “картинки яркостей” на светодиодах) со сдвигом вправо и скоростью перемещения 0.5 сек.
10. Напишите программу периодически увеличивающую по линейному закону яркость свечения светодиода B0. Период изменения яркости должен составлять 1с. После достижения максимальной яркости светодиод должен гаснуть.
11. Модифицируйте предыдущую программу так, чтобы после периода плавного нарастания яркости следовал бы период ее уменьшения.
12. Модифицируйте предыдущую программу так, чтобы светодиод B1 работал бы в противофазе B0, т.е. периоду увеличения яркости B0 соответствовал бы период уменьшения яркости B1 и наоборот.
13. Модифицируйте программу 7 так, чтобы дополнительно мигал светодиод B1 с частотой 8 Гц.
14. Реализуйте программу бегущего огня (один зажженный светодиод перемещается влево), в которой скорость пробега от правого края к левому менялась бы линейно от 1 сек для перехода от светодиода B0 к B1, до 0.1 сек от B6 к B7.
15. Реализуйте программу бегущего огня (один зажженный светодиод перемещается влево), в которой скорость пробега от правого края к левому менялась бы линейно с каждым новым циклом. Первый цикл должен иметь период 4 сек, второй - 2 сек, третий - 1 сек и т. д. до периода 0.125 сек, после чего приведенный алгоритм повторяется.
16. Модифицируйте программу 8 так, чтобы дополнительно можно было бы управлять яркостью свечения светодиода B1 (как в программе 3).

Лабораторная работа №9

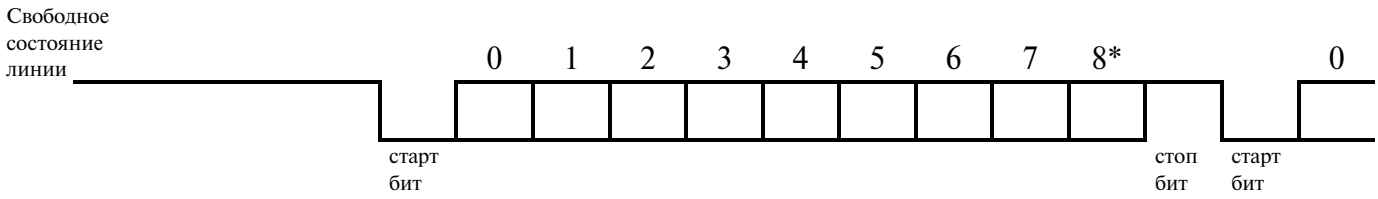
Последовательный асинхронный интерфейс (SCI).

1. Введение

В данной работе рассматривается блок последовательного асинхронного интерфейса и методы взаимодействия с ним.

2. Последовательный асинхронный интерфейс

Последовательный асинхронный интерфейс применяется для обеспечения коммуникации между контроллером и внешними устройствами. Обмен данными производится по двум линиям: одна используется для приема данных, другая для передачи. Формат передаваемых данных представлен на рис. 9.1.



* Наличие 9-го бита в посылке определяется состоянием управляющего бита M.

Рис. 9.1. Формат данных SCI.

Один кадр передаваемых по SCI данных включает в себя:
- старт бит (логический “0”), используемый для индикации начала кадра;
- 8 или 9 битов данных (первым передается младший бит);
- стоп бит.

Для обмена данными между SCI и программой пользователя существует регистр данных SCI SCDR (адрес $102F), физически представляющий собой два регистра: один для чтения (RDR), другой для записи (TDR). Принятый байт данных записывается в регистр RDR, доступный пользователю при чтении регистра SCDR. Передаваемый байт данных помещается в регистр TDR при записи информации в SCDR.

Кроме режима работы, предусматривающего передачу восьми байт в кадре, существует режим передачи девяти байт. Он устанавливается при записи 1 в бит управления M регистра управления SCCR1 (см. рис. 9.2). При этом принятый бит 8 отображается битом R8 регистра SCCR1, а передаваемый бит 8 должен быть записан в бит T8 регистра SCCR1.

	7	6	5	4	3	2	1	0	
$102C	R8	T8	0	M	WAKE	0	0	0	**SCCR1**
Состояние после сброса	-	-	0	0	0	0	0	0	

Рис. 9.2. Регистр управления SCI SCCR1.

	7	6	5	4	3	2	1	0	
$102D	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	**SCCR2**
Состояние после сброса	0	0	0	0	0	0	0	0	

Рис. 9.3. Регистр управления SCI SCCR2.

Разрешение работы SCI производится путем установки битов TE (разрешение работы передатчика) и RE (разрешение работы приемника) регистра управления SCCR2.

Контролировать работу SCI можно по состоянию флагов регистра статуса SCSR (см. рис. 9.4).

	7	6	5	4	3	2	1	0	
$102E	TDRE	TC	RDRF	IDLE	OR	NF	FE	0	**SCSR**
Состояние после сброса	1	1	0	0	0	0	0	0	

Рис. 9.4. Регистр статуса SCI SCSR.

Флаг **TDRE** устанавливается, когда разрешается запись в регистр данных SCDR (т.е. когда обработана предыдущая запись). Очистка данного флага происходит автоматически путем чтения регистра SCSR (с установленным флагом TDRE) и последующей записью в SCDR. Если установлен флаг TIE регистра SCCR2 (см. рис. 9.3), то установка флага TDRE вызывает генерацию прерывания по SCI.

Флаг **TC** индицирует завершение передачи данных по SCI. Сброс его осуществляется по аналогии с TDRE. Разрешить прерывание по этому флагу можно установкой бита TCIE регистра SCCR2.

Флаг **RDRF** предназначен для индикации того, что в регистр данных SCDR поступил новый принятый байт. Сброс флага осуществляется после чтения SCSR (с установленным RDRF) и последующим чтением SCDR. Прерывание по этому флагу разрешается, если установлен бит RIE регистра SCCR2.

Если принят новый байт, а старый еще не считан (RDRF="1"), то возникает ошибка переполнения и устанавливается флаг **OR**. При этом новые полученные данные теряются. Сброс этого флага производится по аналогии с RDRF. Разрешением прерывания по его установке также управляет бит RIE регистра SCCR2.

Если при приеме был зафиксирован большой уровень шума, или произошла ошибка кадрирования, то устанавливаются флаги **NF** и **FE** соответственно. Сброс этих флагов производится по аналогии с RDRF.

Флаг **IDLE** устанавливается для индикации того, что линия находится в свободном состоянии больше длительности передачи одного кадра. Сброс этого флага производится по аналогии с RDRF.

Управление скоростью передачи по SCI осуществляется через регистр BAUD (см. рис. 4). Скорость обмена в *бодах* может быть рассчитана по следующей формуле: $V=E/(16\times N\times M)$, где E - внутренняя тактовая частота микроконтроллера (для модуля HC11EVB E=2 МГц); N - коэффициент деления, задаваемый битами SPC1 и SPC0 регистра BAUD в соответствии с табл. 9.1; M - коэффициент деления, задаваемый битами SCR2, SCR1 и SCR0 регистра BAUD в соответствии с табл. 9.2. Например, для получения серии частот ряда 9600, 4800, 2400, 1200 бод N должно быть равным 13, а M должно быть 1, 2, 4, 8 соответственно.

	7	6	5	4	3	2	1	0	
$102B	TCLR	0	SCP1	SCP0	RCKB	SCR2	SCR1	SCR0	**BAUD**
Состояние после сброса	0	0	0	0	0	-	-	-	

Рис. 9.5. Регистр управления скоростью обмена SCI BAUD.

Таблица 9.1.

SCP1	SCP0	Коэффициент деления N
0	0	1
0	1	3
1	0	4
1	1	13

Таблица 9.2.

SCR2	SCR1	SCR0	Коэффициент деления M
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Для своей работы SCI задействует две линии порта D: PD0 (прием данных) и PD1 (передача данных). В отладочном модуле HC11EVB асинхронный интерфейс используется для обмена с центральной ЭВМ, поэтому и изучение работы SCI наиболее удобно проводить при их взаимодействии. Для выполнения примеров в данной лабораторной работе, после запуска программы в контроллере следует выйти из программы-отладчика и войти в режим терминала. При этом принимаемые ЭВМ байты будут отображаться на дисплее в виде их ASCII эквивалентов, а передаваться будут ASCII значения, соответствующие нажимаемым на клавиатуре символам. После входа в режим терминала следует установить параметры обмена: : 1 старт-бит, 8 бит данных, 1 стоп-бит, скорость обмена и номер COM-порта, к которому подключен отладочный модуль.

Следующая программа реализует режим “эхо” при работе управляющей ЭВМ в режиме терминала. Работа программы состоит в следующем: при нажатии клавиши на клавиатуре ЭВМ ее двоичный ASCII код принимается микроконтроллером и отсылается обратно, при этом принятый ЭВМ код индицируется на дисплее в виде символа, соответствующего нажатой клавише.

```
            cpu   6811
sccr1       equ   $2c           ; регистр управления SCI 1
sccr2       equ   $2d           ; регистр управления SCI 2
scsr        equ   $2e           ; регистр состояния SCI
baud        equ   $2b           ; регистр управления скоростью обмена
                                ; SCI
scdr        equ   $2f           ; регистр данных SCI

            org   $d000         ; начало основной программы
            ldy   #$1000

            sei                 ; запрет I-прерываний

            ldaa  #$7e          ; код команды JMP
            staa  $00c4         ; запись по вектору прерывания SCI
                                ; (режим bootstrap)
            ldx   #$d100        ; адрес подпрограммы обслуживания
                                ; прерывания
```

```
stx   $00c5          ; запись в качестве операнда команды
                     ; JMP

ldaa  #%00110010     ; установить скорость обмена 2400 бод
staa  baud,y
ldaa  #%00000000     ; 1 старт-бит, 8 бит данных, 1 стоп-бит
staa  sccr1,y
ldaa  #%00101100     ; разрешить работу передатчика и
                     ; приемника, разрешить
staa  sccr2,y        ; прерывания от приемника
cli                  ; разрешить I-прерывания
bra   *              ; замкнутый цикл

org   $d100          ; подпрограмма обслуживания прерывания
ldy   #$1000
ldaa  scsr,y         ; чтение регистра scsr (для
                     ; последующего сброса флага rdrf)
ldaa  scdr,y         ; чтение регистра данных (и сброс флага
                     ; rdrf регистра scsr)
staa  scdr,y         ; запись данных в регистр данных
                     ; передатчика
rti                  ; возврат из подпрограммы
```

Поскольку программа-отладчик использует интерфейс SCI, запись вектора прерывания производится непосредственно при выполнении программы. После инициализации SCI работа ведется по прерываниям. Запустите программу с адреса $D000. Выйдите из программы - отладчика и войдите в режим терминала. Выключите режим локального “эха”, если он включен. Установите параметры обмена: 1 старт-бит, 8 бит данных, 1 стоп-бит, скорость обмена 2400 бод и номер COM-порта, к которому подключен отладочный модуль. Нажимая на клавиши ЭВМ убедитесь, что их символьные значения отображаются на дисплее.

При работе в системе с несколькими SCI приемниками (входы которых подсоединены к одной приемной линии) возникает проблема коммуникации, заключающаяся в том, чтобы определить к какому устройству производится доступ. Для решения этой проблемы в SCI предусмотрена возможность перехода в режим “сна”, который характеризуется тем, что прием данных запрещается до возникновения одного из следующих событий, в зависимости от состояния бита WAKE регистра SCCR1:

- линия приема остается в свободном состоянии (логическая “1”) на время передачи одного кадра (WAKE=“0”);

- в принимаемом байте установлен старший бит (WAKE=“1”).

Перевод SCI в режим “сна” осуществляется установкой бита RWU регистра управления SCI SCCR2 (см. рис. 9.3). Этот бит автоматически сбрасывается после обнаружения соответствующего из вышеприведенных событий.

“Пробуждение” SCI по свободной линии может использоваться, например, в следующей ситуации: передатчик оставляет линию свободной на время больше одного кадра, при этом все приемники “пробуждаются” и активизируются. После этого начинается прием блока данных, причем интервал между передачей двух байтов не превышает одного кадра. Каждый приемник по первому байту (байтам) блока определяет к нему ли производится обращение и если нет, то снова возвращается в режим “сна” до начала приема нового блока данных.

При “пробуждении” SCI по установленному старшему биту номер устройства, к которому производится обращение, кодируется в байте с установленным старшим битом. Таким образом первый байт блока передаваемых данных должен быть с установленным старшим битом и указывать на номер микроконтроллера к которому производится обращение. Остальные байты блока должны иметь “0” в старшем разряде. При этом нет необходимости,

чтобы интервал между двумя байтами сообщения был меньше длительности передачи одного кадра. Микроконтроллер после приема байта с установленной “1” в старшем разряде должен определить по этому байту к нему ли производится обращение и если нет, то снова перейти в режим “сна”. Этот режим пробуждения также называется режимом пробуждения по адресному маркеру.

Следующая программа демонстрирует использование режима “сна / пробуждения”. По сути это предыдущая программа реализующая режим “эхо” для терминала ЭВМ, но активизация ее работы происходит только после приема байта с установленным старшим битом и остальными битами соответствующими семи младшим переключателям отладочного модуля.

```
            cpu   6811
sccr1       equ   $2c           ; регистр управления SCI 1
sccr2       equ   $2d           ; регистр управления SCI 2
scsr        equ   $2e           ; регистр состояния SCI
baud        equ   $2b           ; регистр управления скоростью обмена
                                ; SCI
scdr        equ   $2f           ; регистр данных SCI

            org   $d000         ; начало основной программы
            ldy   #$1000
            sei                 ; запрет I-прерываний
            ldaa  #$7e          ; код команды JMP
            staa  $00c4         ; запись по вектору прерывания SCI
                                ; (режим bootstrap)
            ldx   #$d100        ; адрес подпрограммы обслуживания
                                ; прерывания
            stx   $00c5         ; запись в качестве операнда команды
                                ; JMP
            ldaa  #%00110010    ; установить скорость обмена 2400 бод
            staa  baud,y
            ldaa  #%00001000    ; 1 старт-бит, 8 бит данных, 1 стоп-
                                ; бит, “пробуждение” по
            staa  sccr1,y       ; адресному маркеру
            ldaa  #%00101110    ; разрешить работу передатчика и
                                ; приемника, разрешить
            staa  sccr2,y       ; прерывания от приемника, установить
                                ; режим “сна”
            cli                 ; разрешить I-прерывания
            bra   *             ; замкнутый цикл
            org   $d100         ; подпрограмма обслуживания прерывания
            ldy   #$1000
            ldaa  scsr,y        ; чтение регистра scsr (для
                                ; последующего сброса флага rdrf)
            ldaa  scdr,y        ; чтение регистра данных (и сброс флага
                                ; rdrf регистра scsr)
            bita  #$80          ; проверка на установленный старший бит
            beq   j1
            anda  #$7f          ; если установлен, то
            ldab  $1f03         ; сравнить состояние семи мл.
                                ; переключателей и
            cba                 ; семи мл. бит принятого байта
            beq   j_end         ; если совпадают, выход без установки
                                ; режима “сна”
```

```
            bset  sccr2,y,%00000010; иначе установить режим "сна"
            bra   j_end
j1          staa  scdr,y          ; запись данных в регистр данных
                                  ; передатчика
j_end       rti                   ; возврат из подпрограммы
```

Остановимся подробнее на работе программы. В начале программы кроме обычных установок программируется режим пробуждения по адресному маркеру и SCI вводится в режим "сна". После возникновения прерывания от приемника принятый байт проверяется на наличие в старшем разряде "1". Если старший бит установлен, следовательно принят адресный маркер и далее производится проверка на соответствие 7-битного кода маркера и 7-битного кода, установленного на переключателях отладочного модуля. Если они не совпадают, SCI снова вводится в режим "сна", т.е. следующим принятым байтом сможет стать только новый адресный маркер. При совпадении кодов происходит выход из подпрограммы без установки режима "сна", давая таким образом возможность SCI принимать все последующие байты данных, которые снова пересылаются в ЭВМ и отображаются на дисплее.

Установите на переключателях какое-либо двоичное число, например $10. Запустите программу с адреса $D000. Выйдите из программы - отладчика и войдите в режим терминала. Установите параметры обмена: 1 старт-бит, 8 бит данных, 1 стоп-бит, скорость обмена 2400 бод и номер COM-порта, к которому подключен отладочный модуль. Нажимая на цифровые и/или символьные клавиши ЭВМ с ASCII кодом меньше $80 убедитесь, что они не отображаются на дисплее. Введите теперь символ с кодом, соответствующим коду на переключателях и с установленным старшим битом. Для нашего примера это $90 - код, как правило соответствующий русской заглавной "Р". Ввод ASCII символа с произвольным кодом можно производить следующим образом: нажмите клавишу <Alt> на клавиатуре ЭВМ, и удерживая ее наберите на боковой цифровой клавиатуре десятичный код символа (для нашего примера это 144). Передача символа произойдет после отпускания клавиши <Alt>. После ввода маркера убедитесь, что активизировался режим "эхо" и вводимые с клавиатуры ЭВМ символы отображаются на дисплее. Аналогично предыдущему введите новый адресный маркер, но с кодом отличающимся от установленного на переключателях. Убедитесь, что режим "эхо" заблокировался.

3. Контрольные вопросы

1. Опишите формат данных SCI. Что такое кадр данных?
2. В чем разница 8- и 9- битового форматов данных? Как выбрать необходимый формат?
3. Опишите процесс использования регистра данных SCDR. Как этот регистр устроен физически?
4. Как разрешается работа приемника и передатчика?
5. Опишите функцию флага TDRE регистра SCSR и как с ним работать.
6. Опишите функцию флага TC регистра SCSR и как с ним работать.
7. Опишите функцию флага RDRF регистра SCSR и как с ним работать.
8. Опишите функцию флага OR регистра SCSR и как с ним работать.
9. Опишите функцию флагов NF и FE регистра SCSR и как с ними работать.
10. Опишите функцию флага IDLE регистра SCSR и как с ним работать.
11. Как осуществляется установка скорости передачи SCI?
12. Рассчитайте параметры N и M для скорости обмена 15625 бод, учитывая, что внутренняя частота синхронизации равна 2МГц.
13. Опишите функцию битов регистра SCCR2.
14. Какие внешние линии микроконтроллера задействованы для обеспечения работы SCI?
15. Для чего используется режим “сна” SCI?
16. Опишите процесс “пробуждения” SCI по свободной линии.
17. Опишите процесс “пробуждения” SCI по адресному маркеру.

4. Задания

Примечание. Для выполнения некоторых заданий вам потребуются ASCII коды клавиш, представленные в следующей таблице:

		Старшая цифра							
		0	1	2	3	4	5	6	7
Младшая цифра	0	NULL	отключение линии передачи данных	пробел	0	@	P	'	p
	1	начало заголовка	управление устройством 1	!	1	A	Q	a	q
	2	начало текста	управление устройством 2	“	2	B	R	b	r
	3	конец текста	управление устройством 3	#	3	C	S	c	s
	4	конец передачи	управление устройством 4	$	4	D	T	d	t
	5	запрос	ошибочная ситуация	%	5	E	U	e	u
	6	подтверждение	синхронизация	&	6	F	V	f	v
	7	звонок	конец пересылки блока данных	'	7	G	W	g	w
	8	возврат на 1 позицию назад	отмена	(	8	H	X	h	x
	9	горизонтальная табуляция	конец носителя	)	9	I	Y	i	y
	A	перевод строки	подстановка	*	:	J	Z	j	z
	B	вертикальная табуляция	отключение	+	;	K	[	k	{
	C	перевод формата	разделитель файлов	,	<	L	\	l	|
	D	возврат каретки	разделитель групп	-	=	M	]	m	}
	E	нижний регистр	разделитель записей	.	>	N	^	n	~
	F	верхний регистр	разделитель элементов	/	?	O	-	o	забой

1. Напишите программу “эхо” для режима терминала, работающую на скорости 9600 бод.
2. Напишите программу, которая высвечивала бы на светодиодах код нажатой на клавиатуре ЭВМ клавиши.

3. Напишите программу, которая по нажатию на кнопку PA0 отсылала бы на дисплей код, установленный на переключателях модуля.
4. Напишите программу, которая посылала бы на дисплей ASCII сообщение из ОЗУ (например объемом 16 байт), предварительно записанное в него, после нажатия на клавишу PA0.
5. Напишите программу записывающую в 5-разрядный буфер в ОЗУ набор символов с клавиатуры ЭВМ, после чего выводящую одновременно весь буфер на дисплей.
6. Напишите программу “эхо”, которая при вводе заглавных букв печатала бы прописные и наоборот.
7. Напишите программу, которая после ввода произвольного заранее заданного кодового слова зажигала бы на 1 секунду все светодиоды.
8. Напишите программу, выводящую на дисплей последовательно все заглавные буквы латинского алфавита. Каждый новый символ должен выводится через 0.5 сек.
9. Напишите программу, которая после ввода на клавиатуре ЭВМ десятичного числа в диапазоне от 0 до 255 и нажатия на клавишу <Enter> индицировала бы его на светодиодах модуля в двоичном виде.
10. Напишите программу, которая двоичное число, установленное на переключателях модуля отображала бы на дисплее в десятичном виде после нажатия на кнопку PA0.
11. Напишите программу, которая генерировала бы на дисплее слова, состоящие из случайных символов длиной от 3 до 8 позиций (тоже случайная величина), по нажатию на кнопку PA0. Последним символом каждого слова должен быть пробел.
12. Напишите программу, которая реализовывала бы алгоритм задания 2, но только после приема адресного маркера с кодом, установленным на переключателях модуля.
13. Напишите программу, которая после приема правильного адресного маркера (код установлен на светодиодах) выводила бы на дисплей произвольное сообщение, ранее записанное в ОЗУ.
14. Напишите программу, которая двигала бы по строке слева направо произвольный символ. После достижения правой границы строки символ должен исчезнуть, после чего цикл повторяется. Скорость перемещения выберите самостоятельно.
15. Напишите программу, реализующую алгоритм задания 14, с тем отличием, что символ должен двигаться по столбцу снизу вверх.
16. Напишите программу, реализующую следующий игровой алгоритм: После нажатия на кнопку PA0 на дисплее возникает случайный символ латинского алфавита или цифра. Если в течение 1 секунды нажимается клавиша на клавиатуре ЭВМ с соответствующим символьным обозначением, то рядом появляется новый случайный символ и т.д. Если в течение 1 секунду нет нажатия или нажимается неправильная клавиша, то на 1 секунду загораются все светодиоды модуля.
17. Напишите программу, использующую алгоритм задания 14 с тем отличием, что в течение 1 сек символ должен перемещаться по строке слева направо. После достижения правой границы, или если во время перемещения была нажата правильная клавиша, то текущий символ должен исчезнуть, а на левой границе строки появиться новый, после чего цикл повторяется.
18. Напишите программу, реализующую алгоритм задания 15 с тем отличием, что время пробега символа через строку должно уменьшаться с каждым новым символом на 0.05 секунды, причем начальное время должно быть 2 сек. и конечное 0.3 сек. Если символ набран на клавиатуре ЭВМ неправильно, то время прохождения следующего символа должно увеличиваться на 0.1 сек.

Лабораторная работа №10
Работа с EEPROM.

1. Введение

В данной работе изучаются методы взаимодействия с внутренней электрически - программируемой памятью EEPROM.

2. Работа с EEPROM

EEPROM микроконтроллера HC11E1 представляет собой электрически - стираемое/программируемое ПЗУ объемом 512 байт и размещенное по адресам с $B600 по $B7FF. Обычно он используется для хранения данных, которые должны быть сохранены после выключения питания микроконтроллера. Это могут быть, например, оперативные данные, которые не должны быть потеряны при аварии по питанию, коэффициенты или базовые таблицы, требующие периодической корректировки и др.

Механизм записи (программирования) EEPROM управляется регистром PPROG (см. рис. 10.1). EEPROM разрешается (появляется в карте памяти) при установке бита EEON регистра CONFIG (о структуре и особенностях программирования регистра CONFIG см. далее).

При стирании байта из области EEPROM его значение становится $FF. При программировании возможно изменение значения бита только из “1” в “0”. Если какой-либо бит требует установки из “0” в “1”, то перед программированием байта следует его стереть. Если новый байт данных не содержит единиц в позициях уже запрограммированных в “0”, то возможно программирование нового значения без предварительного стирания старого.

Для стирания/программирования EEPROM не требуется подачи внешнего высокого напряжения благодаря наличию в микроконтроллере встроенного генератора накачки. Эффективность работы генератора накачки зависит от частоты внутренней синхронизации микроконтроллера E. Если частота E становится ниже 2 МГц, то эффективность генератора падает, что ведет к увеличению требуемого времени для программирования или стирания EEPROM. При E=2МГц рекомендуемое время составляет 10 мс, это время следует увеличить до 20 мс при уменьшении E до 1...2 МГц. При уменьшении E ниже 1 МГц источником синхронизации для генератора накачки следует выбрать внутренний RC генератор. Для этого необходимо установить бит CSEL (бит 6) регистра OPTION. После установки бита CSEL требуется задержка 10 мс до начала программирования, чтобы RC генератор вошел в установившийся режим.

	7	6	5	4	3	2	1	0	
$103B	ODD	EVEN	0	BYTE	ROW	ERASE	EELAT	EEPGM	**PPROG**
Состояние после сброса	0	0	0	0	0	0	0	0	

Рис. 10.1. Регистр управления EEPROM PPROG.

Определенные области EEPROM могут быть защищены от записи/стирания путем установки в “1” соответствующих битов регистра BPROT (см. рис. 10.2).

	7	6	5	4	3	2	1	0	
$1035	0	0	0	PTCON	BPRT3	BPRT2	BPRT1	BPRT0	**BPROT**
Состояние после сброса	0	0	0	1	1	1	1	1	

Рис. 10.2. Регистр защиты EEPROM BPROT.

Каждый из битов BPRT3...BPRT0 отвечает за определенную область EEPROM, как представлено в следующей таблице:

Бит	Защищаемый блок	Размер блока
BPRT0	$B600-$B61F	32 байта
BPRT1	$B620-$B65F	64 байта
BPRT2	$B660-$B6DF	128 байт
BPRT3	$B6E0-$B7FF	288 байт

При установке всех четырех битов (состояние после сброса) запрещается стирание/программирование всей области EEPROM. В нормальных режимах работы (расширенном и однокристальном) биты регистра BPROT могут быть сброшены в “0” только в течение первых 64 E циклов работы микроконтроллера после сброса. Это ограничение не распространяется на специальные режимы работы. Установить любой бит регистра BPROT в “1” можно в любой момент независимо от режима работы микроконтроллера.

Управление режимом стирания/программирования производится через регистр PPROG (рис. 10.1). Режим стирания EEPROM разрешается установкой бита ERASE регистра PPROG. Стирание ячеек EEPROM осуществляется в одном из следующих режимов:

- побайтное стирание (стирается только конкретный байт);
- построчное стирание (стирается стока из 16-ти байт, начиная с байта, адрес которого имеет $0 в младшей тетраде и кончая байтом с $FF в младшей тетраде адреса: $B600-$B60F, $B610-$B61F, ... ,$B7F0-$B7FF)
- стирание всего EEPROM.

Режим стирания задается битами BYTE и ROW регистра PPROG следующим образом:

BYTE	ROW	Режим стирания
0	0	Стирание всего EEPROM
0	1	Построчное стирание
1	X*	Побайтное стирание

X*-безразличное состояние

В режиме программирования (или чтения) значения битов BYTE и ROW не имеет значение, а бит ERASE должен быть сброшен.

Режим обработки данных для стирания/программирования разрешается при установке бита EELAT регистра PPROG. В режиме чтения этот бит должен быть сброшен.

Непосредственно стирание/программирование выполняется при подачи высокого напряжения от генератора накачки. Высокое напряжение включается при установке бита EEPGM регистра PPROG и выключается при его сбросе.

Чтение EEPROM выполняется при сброшенном бите EELAT и ничем не отличается от чтения других областей памяти.

Для иллюстрации процесса взаимодействия с EEPROM предлагается ряд примеров подпрограмм, осуществляющих его программирование и стирание.

1. Программирование байта.

Входные данные: регистр X - адрес ячейки EEPROM, аккумулятор A - программируемый байт.

```
            cpu   6811
prog        pshb
            ldab  #$02
            stab  $103b          ; установить режим
                                 ; программирования/стирания
            staa  0,X            ; запомнить данные по адресу в EEPROM
            ldab  #$03
```

```
            stab  $103b          ; подать высокое напряжение
            jsr   dly10           ; задержка 10 мс
            clr   $103b           ; выключить высокое напряжение и
                                  ; установить режим чтения
            pulb
            rts

dly10       pshx
            ldx   #$0d10          ; подпрограмма задержки ≈ 10 мс (для
                                  ; частоты E=2МГц)
j1          dex
            bne   j1
            pulx
            rts
```

2. Стирание всего EEPROM (512 байт).

```
bulke       pshb
            ldab  #$06
            stab  $103b           ; установить режим стирания всего
                                  ; EEPROM
            stab  $b600           ; запомнить любые данные по любому
                                  ; адресу в EEPROM
            ldab  #$07
            stab  $103b           ; подать высокое напряжение
            jsr   dly10           ; задержка 10 мс
            clr   $103b           ; выключить высокое напряжение и
                                  ; установить режим чтения
            pulb
            rts
```

3. Стирание строки длиной 16 байт ($B600-$B60F, $B610-$B61F, ... ,$B7F0-$B7FF)

Входные данные: регистр X - любой адрес в стираемой строке EEPROM.

```
rowe        pshb
            ldab  #$0e
            stab  $103b           ; установить режим стирания строки
                                  ; EEPROM
            stab  0,x             ; запомнить любые данные по любому
                                  ; адресу в стираемой строке EEPROM
            ldab  #$0f
            stab  $103b           ; подать высокое напряжение
            jsr   dly10           ; задержка 10 мс
            clr   $103b           ; выключить высокое напряжение и
                                  ; установить режим чтения
            pulb
            rts
```

4. Стирание байта.

Входные данные: регистр X - адрес стираемой ячейки EEPROM.

```
bytee       pshb
            ldab  #$16
            stab  $103b           установить режим стирания байта EEPROM
            stab  0,x             запомнить любые данные по адресу
                                  стираемого байта EEPROM
            ldab  #$17
            stab  $103b           ; подать высокое напряжение
```

```
            jsr   dly10          ; задержка 10 мс
            clr   $103b          ; выключить высокое напряжение и
                                 ; установить режим чтения
            pulb
            rts
```

3. Регистр конфигурации CONFIG

Регистр CONFIG (см. рис. 10.3), определяющий конфигурацию микроконтроллера представляет собой ячейку EEPROM, подчиняющуюся тем же правилам программирования, что и остальные EEPROM ячейки.

	7	6	5	4	3	2	1	0	
$103F	0	0	0	0	NOSEC	NOCOP	ROMON	EEON	**CONFIG**
Состояние после сброса	0	0	0	0	-	-	-	-	

Рис. 10.3 Регистр конфигурации системы CONFIG.

Каждый бит регистра CONFIG отвечает за включение определенной системы в конфигурацию микроконтроллера. Например, сброшенный бит NOCOP разрешает работу системы защиты от программных сбоев COP; установленный бит ROMON разрешает работу встроенного ПЗУ микроконтроллера, т.е. область ПЗУ появляется в карте памяти (в микроконтроллере MC68HC11E1 внутреннего ПЗУ нет и бит ROMON отсутствует); установленный бит EEON разрешает работу EEPROM.

Регистр CONFIG состоит из ячейки EEPROM и статического регистра, состояние битов которого непосредственно определяет конфигурацию системы. Сразу после сброса микроконтроллера содержимое EEPROM ячейки CONFIG переписывается в статический регистр, фиксируя новую конфигурацию. Таким образом изменение содержимого регистра CONFIG приведет к изменению конфигурации только после очередного сброса микроконтроллера.

Следующие подпрограммы показывают как можно программировать/стирать регистр CONFIG.

5. Программирование регистра CONFIG.

Входные данные: аккумулятор A - программируемый байт.

```
progc       pshb
            ldab  #$02
            stab  $103b          ; установить режим
                                 ; программирования/стирания
            staa  $103F          ; запомнить данные по адресу регистра
                                 ; CONFIG
            ldab  #$03
            stab  $103b          ; подать высокое напряжение
            jsr   dly10          ; задержка 10 мс
            clr   $103b          ; выключить высокое напряжение и
                                 ; установить режим чтения
            pulb
            rts
```

6. Стирание регистра CONFIG.

```
bytec       pshb
            ldab  #$16
            stab  $103b          ; установить режим стирания байта
                                 ; EEPROM
            stab  $103F          ; запомнить любые данные по адресу
```

```
                        ; регистра CONFIG
        ldab  #$17
        stab  $103b     ; подать высокое напряжение
        jsr   dly10     ; задержка 10 мс
        clr   $103b     ; выключить высокое напряжение и
                        ; установить режим чтения
        pulb
        rts
```

Рассмотрим практическое применение представленных подпрограмм. Запишите в память микроконтроллера все шесть представленных подпрограмм начиная с адреса $D100. Дополнительно запишите следующую программу:

```
        cpu   6811
        org   $d000     ; начало основной программы
        clr   $1035     ; сбросить регистр bprot, разрешив
                        ; программирование ячеек EEPROM
        ldx   #$B605    ; установить указатель на ячейку EEPROM
        ldaa  #$55      ; установить число для программирования
        jsr   prog      ; выполнить программирование байта
                        ; EEPROM
        bra   *
```

Данная программа выполняет запись числа $55 в ячейку EEPROM с адресом $B605. Перед выполнением программы убедитесь, что программируемая ячейка очищена (т.е. содержит значение $FF), и EEPROM присутствует в карте памяти микроконтроллера, проверив установлен ли младший бит регистра CONFIG. Если нет, установите его, выполнив команду “Memory Modify” отладчика (программа-отладчик автоматически определяет является ли конкретный байт ячейкой EEPROM и выбирает алгоритм его программирования), после чего выполните сброс микроконтроллера и восстановите связь. Выполните программу начиная с адреса $D000. Убедитесь, что в ячейку $B605 записалось число $55. Попробуйте в эту же ячейку аналогичным образом записать число $F0 (не стирая предыдущее значение). После этого в ячейке $B605 будет записано число $50, т.е. переход произошел только от “1” к “0”, но не наоборот.

Замените предыдущую программу на новую:

```
        cpu   6811
        org   $d000     ; начало основной программы
        clr   $1035     ; сбросить регистр bprot, разрешив
                        ; программирование ячеек EEPROM
        ldx   #$B605    ; установить указатель на ячейку EEPROM
        jsr   bytee     ; выполнить стирание байта EEPROM
        bra   *
```

Выполните эту программу начиная с адреса $D000. Убедитесь, что ячейка $B605 стерлась, т.е. ее значение стало $FF.

Следующая программа предназначена для выключения встроенного EEPROM.

```
        cpu   6811
        org   $d000     ; начало основной программы
        clr   $1035     ; сбросить регистр bprot, разрешив
                        ; программирование ячеек EEPROM
        ldaa  $103f     ; записать текущее значение CONFIG в
                        ; аккумулятор
        anda  #$fe      ; сбросить младший бит
        jsr   progc     ; выполнить запись нового значения  в
                        ; CONFIG
        bra   *
```

Выполните программу начиная с адреса $D000. Используя команду отладчика “Memory Modify” убедитесь, что EEPROM все еще присутствует в карте памяти. Нажмите кнопку “Reset” и восстановите связь с модулем. Убедитесь в отсутствии EEPROM в карте памяти.

Для восстановления прежней конфигурации выполните следующую программу:

```
cpu   6811
org   $d000          ; начало основной программы
clr   $1035          ; сбросить регистр bprot, разрешив
                     ; программирование ячеек EEPROM
ldaa  $103f          ; записать текущее значение CONFIG в
                     ; аккумулятор
oraa  #1             ; установить младший бит
jsr   bytec          ; сбросить регистр CONFIG
jsr   progc          ; выполнить запись нового значения  в
                     ; CONFIG
bra   *
```

4. Контрольные вопросы

1. Какой объем имеет EEPROM микроконтроллера MC68HC11E1 и по каким адресам он расположен?
2. Приведите примеры использования EEPROM в реальных устройствах.
3. Если в ячейке EEPROM записано число $96, можно ли в нее записать число $95, не стирая предыдущего значения? Объясните почему.
4. Каким образом осуществляется формирование высокого напряжения для программирования EEPROM в микроконтроллере MC68HC11E1?
5. Объясните для чего используется встроенный RC генератор, разрешаемый установкой бита CSEL регистра OPTION?
6. Можно ли защитить определенные области EEPROM от стирания/записи? Как это сделать?
7. Какую особенность надо учитывать при модификации регистра BPROT, если микроконтроллер работает в расширенном или однокристальном режиме?
8. Каким образом происходит управление стиранием/программированием EEPROM?
9. Перечислите и опишите возможные режимы стирания EEPROM.
10. Опишите назначение пяти младших битов регистра PPROG.
11. Опишите покомандный алгоритм программирования ячейки EEPROM.
12. Опишите покомандный алгоритм стирания байта EEPROM.
13. Опишите покомандный алгоритм стирания строки EEPROM.
14. Опишите покомандный алгоритм стирания всего EEPROM.
15. Для чего используется регистр CONFIG? Опишите назначение трех его младших битов.
16. Объясните, почему модификация регистра CONFIG не приводит к немедленному изменению конфигурации системы?
17. Опишите покомандный алгоритм программирования регистра CONFIG.
18. Опишите покомандный алгоритм стирания регистра CONFIG.

5. Задания

1. Напишите программу, которая записывала бы состояние переключателей в ячейку EEPROM с адресом, указанным в регистре X по нажатию на кнопку PA0.
2. Напишите программу, записывающую бы по прерыванию IRQ состояние счетчика таймера в первые две ячейки EEPROM.
3. Напишите программу, записывающую число $AA в N первых ячеек EEPROM по нажатию кнопки PA0. Число N (от 0 до 255) определяется состоянием переключателей модуля.
4. Напишите программу реализующую алгоритм задания 3 с тем отличием, что при нажатии на кнопку IRQ должно происходить стирание записанных ячеек.
5. Напишите программу, которая в зависимости от состояния двух младших переключателей блокировала бы один из четырех блоков EEPROM ($b600-$b61f, $b620-$b65f, $b660-b6df, $b6e0-b7ff) для записи/стирания.
6. Напишите программу, производящую запись в EEPROM блока данных из ОЗУ длиной 16 байт. Адрес блока в ОЗУ $d200-$d20f, адрес блока в EEPROM $b600-$b60f.
7. Напишите программу, которая позволяла бы реализовывать один из трех вариантов стирания EEPROM, в зависимости от состояния аккумулятора B (B=0 - стирание всего EEPROM, B=1 - стирание строки, B=2 - стирание одного байта). Входной параметр для программы: регистр X - адрес ячейки EEPROM.
8. Напишите программу повторяющую алгоритм задания 7, с тем отличием, что при B=3 выполнялась бы процедура записи байта в EEPROM. Входные параметры для программы: регистр X - адрес ячейки EEPROM, аккумулятор A - записываемые данные.
9. Напишите программу повторяющую алгоритм задания 8, с тем отличием, что при B=4 выполнялась бы модификация регистра CONFIG. Входные параметры для программы: регистр X - адрес ячейки EEPROM, аккумулятор A - записываемые данные.

10. Напишите программу, которая модифицировала бы младший бит регистра CONFIG, в зависимости от состояния переключателя C0 после нажатия кнопки PA0.
11. Напишите программу последовательно записывающую в ячейку $b600 EEPROM числа от 0 до $3f с промежутком между двумя записями в 1 секунду. После записи числа $3f снова записывается 0 и цикл повторяется.
12. Напишите программу, заполняющую первую половину EEPROM числами от $0 до $FF. По нажатии кнопки PA0 содержимое EEPROM должно стираться.
13. Напишите программу осуществляющую перестановку первой и второй строки EEPROM (т.е. строки с адресами $b600-$b60f и $b610 и $b61f).
14. Напишите программу, которая измеряла бы количество доступных для записи ячеек EEPROM без обращения к регистру BPROT. При подсчете постарайтесь обойтись минимальным количеством записей в EEPROM.
15. Напишите программу, которая хранилась бы в EEPROM и позволяла бы сохранить в оставшемся объеме EEPROM произвольный блок данных из ОЗУ допустимой длины и при необходимости восстановить его в исходном месте. Входные параметры для программы: регистр X - начальный адрес блока данных, регистр Y - длина блока. Эти параметры также должны запоминаться для последующего корректного восстановления блока.

Приложение

Некоторые особенности при написании и отладке программ на модуле HC11EVB.

Запуск программы пользователя с помощью программы отладчика осуществляется в специальном режиме работы микроконтроллера bootstrap. В отличие от нормальных режимов работы (однокристального и расширенного) режим bootstrap позволяет изменять содержимое регистра HPRIO, а также в течение всего времени работы производить запись в те регистры и биты, запись в которые в нормальных режимах разрешена только первые 64 E цикла после сброса. Кроме этого в режиме bootstrap таблица переходов векторов прерываний перенесена в область ОЗУ (см. табл. П.1). В отличие от области векторов прерывания для нормальных режимов, здесь следует записывать не адрес перехода, а команду перехода с соответствующим адресом (например команду JMP). Таким образом если программа отлаживается в ОЗУ в режиме bootstrap и требуется, например, обработка прерывания IRQ, то в листинге ассемблера следует записать следующую последовательность команд:

```
org   $00ee
jmp   int
```

, где вместо int может быть любая метка или адрес, соответствующая началу программы обработки прерывания по IRQ. При записи отлаженной программы в ПЗУ (предполагается, что микроконтроллер будет работать в однокристальном режиме) вместо приведенной выше последовательности следует записать команды:

```
org   $fff2
fdb   int
```

Таблица П.1.

адрес	вектор
00C4	SCI
00C7	SPI
00CA	счетчик пульсаций, по входному фронту
00CD	счетчик пульсаций, по переполнению
00D0	переполнение таймера
00D3	выходное сравнение 5
00D6	выходное сравнение 4
00D9	выходное сравнение 3
00DC	выходное сравнение 2
00DF	выходное сравнение 1
00E2	входной захват 3
00E5	входной захват 2
00E8	входной захват 1
00EB	прерывание реального времени
00EE	IRQ
00F1	XIRQ
00F4	SWI
00F7	прерывание по неверному коду команды
00FA	прерывание от системы COP
00FD	тактовый монитор
BF00	сброс

При отладке программы следует иметь в виду, что программа отладчик задействует последовательный интерфейс (SCI) и функцию выходного сравнения OC4 (для организации точек останова), т.е. при запуске программы пользователя имеются определенные установки (отличные от состояния после сброса) в регистрах этих устройств. Кроме этого программа отладчик использует прерывание по коду неправильной команды для организации точек

останова и режима trace. Следует иметь в виду, что в момент запуска программы из программы-отладчика сброшен бит I регистра статуса CCR, и таким образом разрешены связанные с ним прерывания (I-прерывания); непосредственно после сброса бит I устанавливается для запрета прерываний и для его сброса в программе пользователя можно воспользоваться командой CLI. Также после запуска программы в программе-отладчике стек инициализирован значением $C3 (в программе пользователя, предполагающей автономный режим работы значение стека, следует устанавливать в начале программы командой lds).