Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Ильшат Ринатович Мухаметзянов
Должность: директор
Дата подписания: 14.07.2023 09:36:08
Уникальный программный ключ:
aba80b84033c9ef19...

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение высшего образования «Казанский национальный исследовательский технический университет им. А.Н. Туполева-КАИ»
(КНИТУ-КАИ)
Чистопольский филиал «Восток»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

по дисциплине

«МИКРОКОНТРОЛЛЕРЫ»

Индекс по учебному плану: **Б1.В.11**
Направление подготовки: **Информатика и вычислительная техника**
Квалификация: **Бакалавр**
Профиль подготовки: **Автоматизированные системы обработки информации и управления**
Типы задач профессиональной деятельности: **проектный, производственно-технологическая**

Рекомендованы УМК ЧФ КНИТУ-КАИ

Чистополь
2023 г.

Лабораторная работа №1

Программирование на AVR-ассемблере

1. Основы работы с *AVR Studio* 4

Среда *AVR Studio* 4 предназначена для создания и отладки программ для 8–разрядных *RISC*–микроконтроллеров *AVR* фирмы *Atmel*, которые предназначены для встраиваемых приложений (они представляют собой миниатюрные устройства с большим набором периферийных средств).

Управление микроконтроллером осуществляется с помощью программы, записанной в память программ с помощью специального программно-аппаратного комплекса – программатора. Как правило, программа, передаваемая программатором в *AVR*–микроконтроллер, должна иметь формат *Intel hex*. Для получения программы в требуемом формате используются соответствующие компиляторы ассемблера или языков высокого уровня (Си, Паскаль, Бейсик и др.).

Система *AVR Studio* 4 позволяет написать программу на языке ассемблера, отредактировать ее, преобразовать в требуемый формат, отладить и записать в микроконтроллер.

Для запуска *AVR Studio* 4 необходимо в меню **Пуск** найти группу программ ***Atmel AVR Tools*** и выбрать в ней ярлык ***AVR Studio* 4**. На рис. 1 показана программная среда ***AVR Studio* 4** после ее запуска.

Процесс получения программы для микроконтроллера начинается с создания нового проекта. Для этого в меню ***Project*** выбирается команда ***New Project*** (рис. 2) и устанавливаются параметры проекта в диалоговом окне ***Create new Project*** (рис. 3).

В диалоговом окне ***Create new Project*** указываются тип проекта (***Project Type***), имя проекта (***Project Name***), имя первичного файла с текстом программы на языке ассемблера (***Initial File***), размещение проекта (***Location***). При этом устанавливаются флаги создания первичного файла (***Create initial File***) и создания отдельной папки для размещения проекта (***Create Folder***). После установки необходимого минимума параметров проекта становится доступной кнопка ***Next***. После нажатия на эту кнопку необходимо установить параметры отладчика в диалоговом окне ***Select debug platform and device*** (рис. 4).

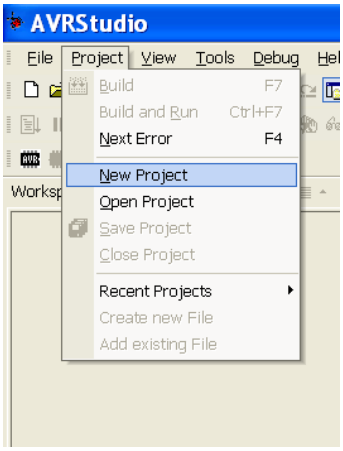

Рис. 2. Команда создания

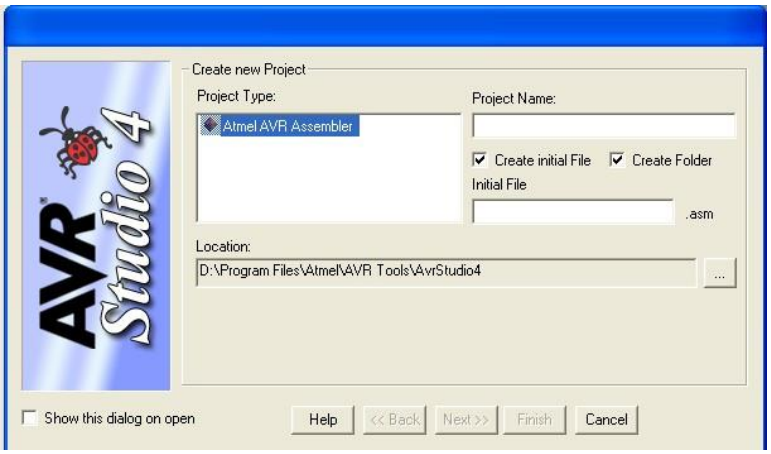

Рис. 3. Диалоговое окно создания

Диалоговое окно ***Select debug platform and device*** содержит две панели. В панели ***Debug Platform*** можно выбрать один интерфейсов внутрисхемной отладки или симулятор (***AVR Simulator***). Панель ***Device*** позволяет указать тип микроконтроллера, для которого отлаживается программа. После выбора типа микроконтроллера и интерфейса отладки становится доступной кнопка ***Finish***, нажатие которой приводит к созданию файлов проекта (рис. 5).

На рис. 5 показан вид *AVR Studio* 4 после создания проекта *prog***1**. В окне *Workspace* указаны файлы проекта (в данном случае проект содержит один файл *prog*1.*asm*), в окне ***Output*** указано, что загружен файл ***ATmega32.xml***, который будет использован при отладке, а в окне редактирования указано содержимое файла *prog*1.*asm* (пока еще пустого).

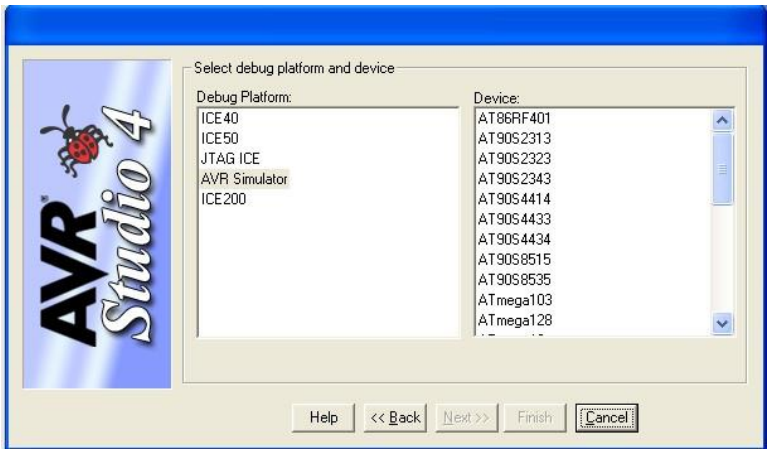

Рис. 4. Диалоговое окно установки параметров отладчика

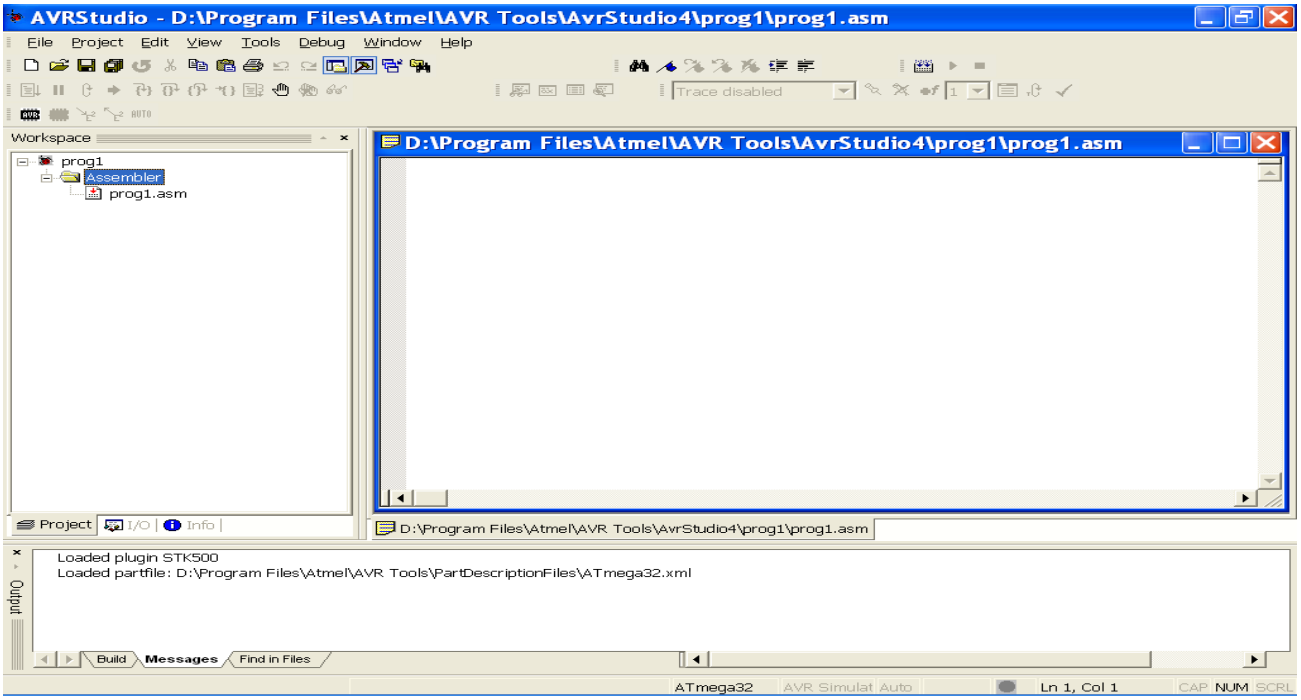

Рис. 5. Вид *AVR Studio* 4 после создания проекта

Теперь необходимо ввести с клавиатуры программу на ассемблере. После ввода программы выполняется ее ассемблирование. Для этого в меню ***Project*** выбирается команда ***Build***. Результат ассемблирования отображается в окне ***Output*** (рис. 6). Информация в окне ***Output*** отражает длину программного кода (***Code***), суммарный объем констант (***Constants***), неиспользуемая часть кода (***Unused***), общая длина программы (***Total***), число ошибок или указание на то, что их нет (***Assembly complete with no errors***) и др.

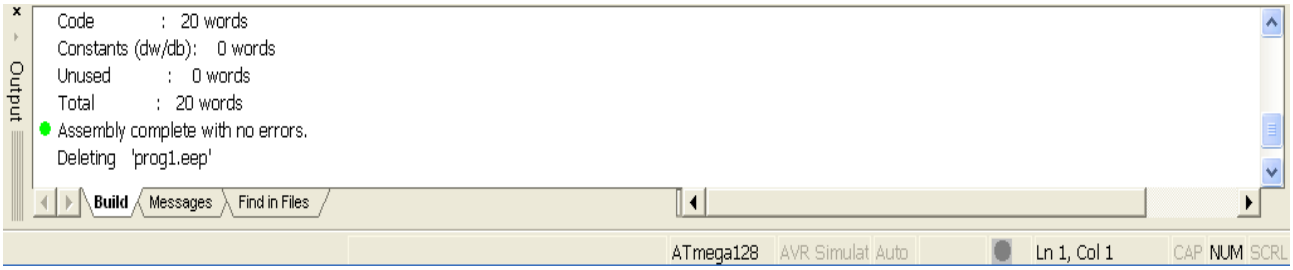

Рис. 6. Результат компиляции в окне

В случае если в окне ***Output*** указываются ошибки необходимо их исправить и повторить ассемблирование еще раз. После того, как все синтаксические ошибки будут исправлены, программу нужно отладить. Для этого используется встроенный отладчик (***Debugger***). Запуск отладчика осуществляется с помощью команды ***Start debugging*** из меню ***Debug***. В процессе отладки отображается состояние различных частей микроконтроллера: регистров общего назначения, памяти данных, портов ввода/вывода и др. Состояние некоторых устройств можно устанавливать, например, щелкнув мышью на одном из разрядов порта ввода/вывода, можно либо сбросить его в ноль, либо установить в единицу.

После того как программа будет отлажена, ее можно записать в микроконтроллер либо непосредственно из системы *AVR Studio* 4, либо с помощью программно–аппаратных средств внешнего программатора.

2. Порядок выполнения лабораторной работы

1. Ознакомиться с описанием *AVR Studio* 4.
2. Создать проект *prog*1.
3. Выбрать для отладки симулятор и устройство *ATmega*32.
4. Ввести следующую программу:

```
.include "m32def.inc"
.cseg
.org    0

                cbi     DDRA,   DDA0
                cbi     DDRA,   DDA1
                sbi     DDRA,   DDA2
                cbi     DDRA,   DDA3
                sbi     PORTA,  PA0
                sbi     PORTA,  PA1
                cbi     PORTA,  PA2
                cbi     PORTA,  PA3
loop:   sbis    PINA, PINA0
                rjmp    end
                sbis    PINA, PINA1
                rjmp    setPA2
setPA3: cbi     PORTA,  PA2
                sbi     PORTA,  PA3
                rjmp    loop
setPA2: sbi     PORTA,  PA2
                cbi     PORTA,  PA3
                rjmp    loop
end:    cbi     PORTA,  PA2
                cbi     PORTA,  PA3
```

5. Ассемблировать программу.

6. Запустить отладчик.
7. Нажимая *F*11, последовательно выполнить все инструкции программы. При этом, устанавливая и/или сбрасывая разряды регистров и портов ввода/вывода микроконтроллера, необходимо определить, что делает данная программа.
8. Ответить на вопросы.

3. Вопросы

1. Микроконтроллер.
2. Архитектура *AVR*–микроконтроллеров.
3. Назначение *AVR*–микроконтроллеров.
4. Программатор.
5. Формат программы для микроконтроллера.
6. Назначение *AVR Studio* 4.
7. Создание проекта в *AVR Studio* 4.
8. Симулятор *AVR Simulator*.
9. Ассемблирование программы в *AVR Studio* 4.
10. Отладка программы в *AVR Studio* 4.

1. Программирование на *AVR*–ассемблере

Программа на ассемблере для *AVR*–микроконтроллеров представляет собой последовательность строк вида:

- **[label:] .directive [operands] [comment]**
- **[label:] instruction [operands] [comment]**
- **comment**
- пустая строка (**empty line**),

где **label** – метка, **directive** – директива (перед директивой ставится точка), **instruction** – инструкция микроконтроллера, **operands** – операнды, **comment** – комментарий. Каждая строка программы не должна превышать 120 символов.

Метка – это идентификатор, содержащий буквы латинского алфавита, цифры и знак подчеркивания.

Директива – это команды управления процессом ассемблирования. Набор директив приведен в таблице 1. В качестве операндов директив используются строки символов.

Таблица 1

Директивы ассемблера

Директива	Описание директивы
BYTE	Выделение байта под переменную в сегменте данных

CSEG	Определение сегмента команд
CSEGSIZE	Определение длины сегмента команд
DB	Выделение байта в памяти программ и *EEPROM*–памяти данных
DEF	Определение символического имени регистра
DEVICE	Определение типа микроконтроллера
DSEG	Определение сегмента данных (память данных)
DW	Выделение двухбайтового слова в памяти программ и *EEPROM*–памяти данных
ENDM, ENDMACRO	Конец макроопределения
EQU	Определение символического имени для выражения
ESEG	Определение сегмента данных в *EEPROM*–памяти
EXIT	Выход из файла (точка прекращения ассемблирования файла)
INCLUDE	Чтение из указанного в директиве файла
LIST	Генерация *list*–файла, содержащего исходную программу, адреса и коды операций
LISTMAC	Включение макроопределений в *list*–файл
NOLIST	Запрещение генерации *list*–файла
ORG	Смещение относительно начала текущего сегмента
SET	Определение символического имени для выражения

Инструкция – это команда микроконтроллера. Список инструкций приведен в приложении (таблицы 4 – 7). При выполнении инструкций устанавливаются в единицу или сбрасываются в ноль разряды (флаги) регистра состояния (*status register*) микроконтроллера *SREG*. Назначение разрядов этого регистра показано в таблице 2. Операндами инструкций являются имена регистров и константы (данные и адреса). При этом *AVR*–ассемблер является нечувствительным к регистру символов (т. е. имена регистров *R*0 и *r*0 идентичны), а некоторые команды допускают использование только определенных регистров.

Таблица 2.2

Разряды регистра состояния

Разряд	***C***	***Z***	***N***	***V***	***S***	***H***	***T***	***I***
Назначение	Перенос	Ноль	Отрицание	Переполнение	Знак	Полуперенос	Бит	Прерывание

Комментарий – это строка, начинающаяся с символа ‘;’. Комментарий при ассемблировании пропускается.

Пустая строка – это последовательность пустых символов (пробелов и табуляций), заканчивающаяся символом перевода строки (вводится после нажатия клавиши *ENTER*).

Выражения, использующиеся в директивах и инструкциях, не должны иметь значения, размер которых превышает 32 бита. Выражения могут состоять из операндов, операторов и функций. В качестве операндов используются:

- метки в сегментах данных и кода;
- переменные, определенные с помощью директивы *SET*;
- константы, определенные с помощью директивы *EQU*;
- целочисленные константы в форматах десятичном (например, 10, 255), шестнадцатеричном (например, 0x0a, $0a, 0xff, $ff), двоичном (например, 0b00001010, 0b11111111), восьмеричном (012, 0377);
- *PC* – текущее значение счетчика команд.

В качестве операторов используются **!** (логическое отрицание), **~** (битовое отрицание), **-** (унарный минус), ***** (умножение), **/** (деление), **+** (сложение), **-** (вычитание), **<<** (сдвиг влево), **>>** (сдвиг вправо), **<** (меньше), **<=** (меньше или равно), **>** (больше), **>=** (больше или равно), **==** (равно), **!=** (не равно), **&** (битовое И), **^** (исключающее битовое ИЛИ), **|** (битовое ИЛИ), **&&** (логическое И), **||** (логическое ИЛИ).

В выражениях могут быть использованы следующие функции:

- **LOW(expression)** – получение младшего байта выражения;
- **HIGH(expression)** – получение старшего байта выражения;
- **BYTE2(expression)** – получение старшего байта выражения;
- **BYTE3(expression)** – получение третьего байта выражения;
- **BYTE4(expression)** – получение четвертого байта выражения;
- **LWRD(expression)** – получение младшего слова выражения;
- **HWRD(expression)** – получение старшего слова выражения;
- **PAGE(expression)** – получение 16..21 битов выражения;
- **EXP2(expression)** – получение 2 в степени, равной значению выражения;
- **LOG2(expression)** – получение целой части двоичного логарифма выражения,

где **expression** – выражение.

Пример 1. Программа создания в памяти данных массива первых 10 чисел Фибоначчи (от 0 до 34).

```
.INCLUDE    “m32def.inc”
.DEVICE     ATmega32          ; Определение микроконтроллера ATMEGA32
.DSEG                         ; Определение сегмента в памяти данных
```

```
table_fib:  .BYTE       10      ; Выделение памяти под 10 однобайтовых значений
        .DEF    fib1=R0         ; (i – 2) – е число Фибоначчи
        .DEF    fib2=R1         ; (i – 1) – е число Фибоначчи
        .DEF    fib=R2                  ; i – е число Фибоначчи
        .DEF    ind=R16         ; индекс в таблице чисел Фибоначчи
        .CSEG                           ; Определение сегмента в памяти программ
        .ORG    0               ; Выполнить смещение в сегменте кода
main:   clr     fib1            ; fib1=0
        clr     fib2            ; fib2=0
        inc     fib2            ; fib2=fib2+1
; Запись в регистровую пару X (R27:R26) адреса таблицы чисел Фибоначчи
        ldi     R26, low(table_fib)
        ldi     R27, high(table_fib)
; Запись первых двух чисел Фибоначчи в таблицу
        st      X+, fib1
        st      X+, fib2
        ldi     ind, 8          ; Инициализация регистра индекса
; Цикл заполнения таблицы
loop:   mov     fib, fib1       ; fib=fib1
        add     fib, fib2       ; fib=fib+fib2
        st      X+, fib                 ; Запись очередного числа Фибоначчи в таблицу
        mov     fib1, fib2      ; fib1=fib2
        mov     fib2, fib       ; fib2=fib
        dec     ind             ; ind=ind–1
        brne    loop            ; Переход на начало цикла, если ind не равен 0
end:    rjmp    end             ; Пустой цикл
```

2. Порядок выполнения лабораторной работы

1. Ознакомиться с программированием на языке ассемблера для **AVR**–микроконтроллера;
2. Ввести, ассемблировать и выполнить с помощью отладчика программу из примера 1 (в системе *AVR Studio* 4).
3. Получить задание из таблицы 3.

4. Написать и отладить программу решения задачи.
5. Ответить на вопросы.

Таблица 3. Задания на лабораторную работу

№	Задание
1	Деление двухбайтовых целых чисел со знаком
2	Деление двухбайтовых вещественных чисел, представленных в виде: <целая часть> <дробная часть>
3	Получение остатка от деления двухбайтовых целых чисел
4	Найти наименьший общий делитель двух двухбайтовых целых чисел
5	Найти минимальный элемент среди 10 двухбайтовых целых чисел в памяти
6	Найти наибольший элемент среди 10 двух байтовых целых чисел в памяти
7	Найти самую короткую серию нулей в массиве из 10 однобайтовых значений
8	Найти самую длинную серию нулей в массиве из 10 однобайтовых значений
9	Определить частоту значений в массиве из 100 однобайтовых чисел
10	Определить позицию наибольшего значения в массиве из 10 однобайтовых чисел

3. Вопросы

1. Структура программы на языке ассемблера для *AVR*–микроконтроллера.
2. Инструкция.
3. Директива.
4. Метка.
5. Операнды.
6. Комментарий.
7. Флаги состояния.
8. Пустая строка.
9. Выражения.
10. Группы инструкций.

Приложение. Инструкции *AVR*-ассемблера

Таблица 4. Арифметические и логические инструкции

Мнемоника	Операнды	Описание	Операция	Флаги
ADD	Rd,Rr	Сложение без переноса	Rd = Rd + Rr	Z,C,N,V,H,S

Мнемоника	Операнды	Описание	Операция	Флаги
ADC	Rd,Rr	Сложение с переносом	Rd = Rd + Rr + C	Z,C,N,V,H,S
ADIW	Rd, K	Сложение регистровой пары со словом	Rd+1:Rd,K	Z,C,N,V,S
SUB	Rd,Rr	Вычитание без переноса	Rd = Rd – Rr	Z,C,N,V,H,S
SUBI	Rd,K8	Вычитание байта	Rd = Rd – K8	Z,C,N,V,H,S
SBC	Rd,Rr	Вычитание с переносом	Rd = Rd – Rr – C	Z,C,N,V,H,S
SBCI	Rd,K8	Вычитание байта с переносом	Rd = Rd – K8–C	Z,C,N,V,H,S
AND	Rd,Rr	Логическое И	Rd = Rd & Rr	Z,N,V,S
ANDI	Rd,K8	Логическое И с байтом	Rd = Rd & K8	Z,N,V,S
OR	Rd,Rr	Логическое ИЛИ	Rd = Rd ∨ Rr	Z,N,V,S
ORI	Rd,K8	Логическое ИЛИ с байтом	Rd = Rd ∨ K8	Z,N,V,S
EOR	Rd,Rr	Логическое исключающее ИЛИ	Rd = Rd ⊕ Rr	Z,N,V,S
COM	Rd	Инверсия	Rd = $FF – Rd	Z,C,N,V,S
NEG	Rd	Дополнение	Rd = $00 – Rd	Z,C,N,V,H,S
SBR	Rd,K8	Установка битов в регистре	Rd = Rd ∨ K8	Z,C,N,V,S
CBR	Rd,K8	Очистка битов в регистре	Rd = Rd & ($FF – K8)	Z,C,N,V,S
INC	Rd	Увеличение на 1	Rd = Rd + 1	Z,N,V,S
DEC	Rd	Уменьшение на 1	Rd = Rd – 1	Z,N,V,S
TST	Rd	Проверка на нулевое или отрицательное значение	Rd = Rd · Rd	Z,C,N,V,S
CLR	Rd	Очистить регистр	Rd = 0	Z,C,N,V,S
SER	Rd	Установить регистр	Rd = $FF	
ADIW	Rdl,K6	Сложение регистровой пары с константой	Rdh:Rdl = Rdh:Rdl + K6	Z,C,N,V,S
SBIW	Rdl,K6	Вычитание константы из регистровой пары	Rdh:Rdl = Rdh:Rdl – K6	Z,C,N,V,S
MUL	Rd,Rr	Беззнаковое умножение	R1:R0 = Rd * Rr	Z,C
MULS	Rd,Rr	Умножение со знаком	R1:R0 = Rd * Rr	Z,C
MULSU	Rd,Rr	Умножение значения со	R1:R0 = Rd * Rr	Z,C

Мнемоника	Операнды	Описание	Операция	Флаги
		знаком на значение без знака		
FMUL	Rd,Rr	Умножение без знака со сдвигом	R1:R0 = (Rd * Rr) << 1	Z,C
FMULS	Rd,Rr	Умножение со знаком со сдвигом	R1:R0 = (Rd *Rr) << 1	Z,C
FMULSU	Rd,Rr	Умножение значения со знаком на значение без знака со сдвигом	R1:R0 = (Rd * Rr) << 1	Z,C

Таблица 5. Инструкции перехода

Мнемоника	Операнды	Описание	Операция	Флаги
RJMP	k	Относительный переход	PC = PC + k +1	
IJMP		Переход по регистру Z	PC = Z	
EIJMP		Расширенный переход по регистру Z	STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND	
JMP	k	Переход по адресу	PC = k	
RCALL	k	Относительный переход к подпрограмме	STACK = PC+1, PC = PC + k + 1	
ICALL		Переход к подпрограмме по регистру Z	STACK = PC+1, PC = Z	
EICALL		Переход к подпрограмме по регистру Z	STACK = PC+1, PC(15:0) = Z, PC(21:16) =EIND	
CALL	k	Переход к подпрограмме	STACK = PC+2, PC = k	
RET		Выход из подпрограммы	PC = STACK	
RETI		Выход из подпрограммы обработки прерывания	PC = STACK	I
CPSE	Rd,Rr	Переход по равенству	if (Rd ==Rr) PC = PC+2 or 3	

Мнемоника	Операнды	Описание	Операция	Флаги
CP	Rd,Rr	Сравнение	Rd – Rr	Z,C,N,V,H,S
CPC	Rd,Rr	Сравнение с переносом	Rd – Rr – C	Z,C,N,V,H,S
CPI	Rd,K8	Сравнение с байтом	Rd – K	Z,C,N,V,H,S
SBRC	Rr,b	Пропуск, если бит в регистре очищен	if(Rr(b)==0) PC = PC+2 or 3	
SBRS	Rr,b	Пропуск, если бит в регистре установлен	if(Rr(b)==1) PC = PC+2 or 3	
SBIC	P,b	Пропуск, если бит в регистре ввода/вывода очищен	if(I/O(P,b)==0) PC = PC+2 or 3	
SBIS	P,b	Пропуск, если бит в регистре ввода/вывода установлен	if(I/O(P,b)==1) PC = PC + 2 or 3	
BRBC	s,k	Переход при очищенном флаге в регистре состояния	if(SREG(s)==0) PC = PC + k + 1	
BRBS	s,k	Переход при установленном флаге в регистре состояния	if(SREG(s)==1) PC = PC + k + 1	
BREQ	k	Переход по равенству	if(Z==1) PC = PC + k+ 1	
BRNE	k	Переход по неравенству	if(Z==0) PC = PC + k+ 1	
BRCS	k	Переход по переносу	if(C==1) PC = PC + k+ 1	
BRCC	k	Переход по отсутствию переноса	if(C==0) PC = PC + k+ 1	
BRSH	k	Переход, если равно или выше	if(C==0) PC = PC + k+1	
BRLO	k	Переход, если ниже	if(C==1) PC = PC + k+ 1	
BRMI	k	Переход, если минус	if(N==1) PC = PC + k+ 1	
BRPL	k	Переход, если плюс	if(N==0) PC = PC + k+ 1	
BRGE	k	Переход, если больше или равно	if(S==0) PC = PC + k+ 1	
BRLT	k	Переход, если меньше	if(S==1) PC = PC + k+ 1	

Мнемоника	Операнды	Описание	Операция	Флаги
BRHS	k	Переход по полупереносу	if(H==1) PC = PC + k+ 1	
BRHC	k	Переход по отсутствию полупереноса	if(H==0) PC = PC + k+ 1	
BRTS	k	Переход при установленном флаге T	if(T==1) PC = PC + k+ 1	
BRTC	k	Переход при очищенном флаге T	if(T==0) PC = PC + k+ 1	
BRVS	k	Переход по переполнению	if(V==1) PC = PC + k+ 1	
BRVC	k	Переход по отсутствию переполнения	if(V==0) PC = PC + k+ 1	
BRIE	k	Переход по разрешению прерывания	if(I==1) PC = PC + k + 1	
BRID	k	Переход по запрещению прерывания	if(I==0) PC = PC + k + 1	

Таблица 6. Инструкции передачи данных

Мнемоника	Операнды	Описание	Операция
MOV	Rd,Rr	Копирование регистра	Rd = Rr
MOVW	Rd,Rr	Копирование регистровой пары	Rd+1:Rd = Rr+1:Rr
LDI	Rd,K8	Запись в регистр байта	Rd = K
LDS	Rd,k	Запись в регистр значения по адресу	Rd = (k)
LD	Rd,X	Запись в регистр значения по адресу в регистре X	Rd = (X)
LD	Rd,X+	См. выше с постинкрементом	Rd = (X), X=X+1
LD	Rd,-X	См. выше с преддекрементом	X=X – 1, Rd = (X)
LD	Rd,Y	Запись в регистр значения по адресу в регистре Y	Rd = (Y)
LD	Rd,Y+	См. выше с постинкрементом	Rd = (Y), Y=Y+1
LD	Rd,-Y	См. выше с преддекрементом	Y=Y – 1, Rd = (Y)
LDD	Rd,Y+q	Запись в регистр значения по адресу	Rd = (Y+q)

Мнемоника	Операнды	Описание	Операция
		в регистре Y со смещением	
LD	Rd,Z	Запись в регистр значения по адресу в регистре Z	Rd = (Z)
LD	Rd,Z+	См. выше с постинкрементом	Rd = (Z), Z=Z+1
LD	Rd,-Z	См. выше с преддекрементом	Z=Z – 1, Rd = (Z)
LDD	Rd,Z+q	Запись в регистр значения по адресу в регистре Z со смещением	Rd = (Z+q)
STS	k,Rr	Запись по адресу значение регистра	(k) = Rr
ST	X,Rr	Запись по адресу в регистре X значение регистра	(X) = Rr
ST	X+,Rr	См. выше с постинкрементом	(X) = Rr, X=X+1
ST	-X,Rr	См. выше с преддекрементом	X=X – 1, (X)=Rr
ST	Y,Rr	Запись по адресу в регистре Y значение регистра	(Y) = Rr
ST	Y+,Rr	См. выше с постинкрементом	(Y) = Rr, Y=Y+1
ST	-Y,Rr	См. выше с преддекрементом	Y=Y – 1, (Y) = Rr
ST	Y+q,Rr	Запись по адресу в регистре Y значения регистра со смещением	(Y+q) = Rr
ST	Z,Rr	Запись по адресу в регистре Z значение регистра	(Z) = Rr
ST	Z+,Rr	См. выше с постинкрементом	(Z) = Rr, Z=Z+1
ST	-Z,Rr	См. выше с преддекрементом	Z=Z – 1, (Z) = Rr
ST	Z+q,Rr	Запись по адресу в регистре Z значения регистра со смещением	(Z+q) = Rr
LPM		Запись в регистр R0 значения из памяти программ по адресу в регистре Z	R0 = (Z)
LPM	Rd,Z	Запись в регистр значения из памяти программ по адресу в регистре Z	Rd = (Z)
LPM	Rd,Z+	См. выше с постинкрементом	Rd = (Z), Z=Z+1

Мнемоника	Операнды	Описание	Операция
ELPM		Расширенная запись в регистр R0 значения из памяти программ по адресу в регистре Z	R0 = (RAMPZ:Z)
ELPM	Rd,Z	Расширенная запись в регистр значения из памяти программ по адресу в регистре Z	Rd = (RAMPZ:Z)
ELPM	Rd,Z+	См. выше с постинкрементом	Rd = (RAMPZ:Z), Z = Z+1
SPM		Запись по адресу памяти программ в регистре Z значения регистровой пары R1:R0	(Z) = R1:R0
ESPM		Расширенная запись по адресу памяти программ в регистре Z значения регистровой пары R1:R0	(RAMPZ:Z) = R1:R0
IN	Rd,P	Чтение из порта	Rd = P
OUT	P,Rr	Запись в порт	P = Rr
PUSH	Rr	Сохранение значения регистра в стеке	STACK = Rr
POP	Rd	Извлечение значения регистра из стека	Rd = STACK

Таблица 7. Битовые инструкции

Мнемоника	Операнды	Описание	Операция	Флаги
LSL	Rd	Логический сдвиг влево	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V,H,S
LSR	Rd	Логический сдвиг вправо	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S
ROL	Rd	Циклический сдвиг влево	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V,H,S
ROR	Rd	Циклический сдвиг вправо	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S

Мнемоника	Операнды	Описание	Операция	Флаги
ASR	Rd	Арифметический сдвиг влево	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S
SWAP	Rd	Обмен полубайтов	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None
BSET	s	Установить флаг в регистре состояния	SREG(s) = 1	SREG(s)
BCLR	s	Очистить флаг в регистре состояния	SREG(s) = 0	SREG(s)
SBI	P,b	Установить флаг в регистре порта ввода/вывода	I/O(P,b) = 1	
CBI	P,b	Очистить флаг в регистре ввода/вывода	I/O(P,b) = 0	
BST	Rr,b	Сохранить бит в разряде T регистра состояния	T = Rr(b)	T
BLD	Rd,b	Записать бит T из регистра состояния	Rd(b) = T	
SEC		Установить флаг C	C =1	C
CLC		Очистить флаг C	C = 0	C
SEN		Установить флаг N	N = 1	N
CLN		Очистить флаг N	N = 0	N
SEZ		Установить флаг Z	Z = 1	Z
CLZ		Очистить флаг Z	Z = 0	Z
SEI		Установить флаг I	I = 1	I
CLI		Очистить флаг I	I = 0	I
SES		Установить флаг S	S = 1	S
CLN		Очистить флаг S	S = 0	S
SEV		Установить флаг V	V = 1	V
CLV		Очистить флаг V	V = 0	V
SET		Установить флаг T	T = 1	T
CLT		Очистить флаг T	T = 0	T
SEH		Установить флаг H	H = 1	H

Мнемоника	Операнды	Описание	Операция	Флаги
CLH		Очистить флаг H	H = 0	H
NOP		Холостая команда		
SLEEP		Переход в спящий режим		
WDR		Разрешить сброс по сторожевому таймеру		
BREAK		Прерывание работы		

В таблицах 2.4 – 2.7 использованы следующие мнемонические обозначения для операндов:

Rd – регистр назначения (*R*0 – *R*31);

Rr – регистр–источник (*R*0 – *R*31);

b – константа от 0 до 7 (номер разряда в регистре общего назначения или регистре порта ввода/вывода);

s – константа от 0 до 7 (номер разряда в регистре состояния);

P – константа от 0 до 31/63 (адрес порта ввода/вывода);

*K*6 – константа от 0 до 63 (непосредственно в инструкции);

*k*8 – константа от 0 до 255;

*K*8 – константа от 0 до 255;

K – константа, значение которой меняется в зависимости от инструкции;

q – константа смещения (непосредственно в инструкции);

Rdl – регистры *R*24, *R*26, *R*28, *R*30 (для инструкций *ADIW* и *SBIW*);

X, *Y*, *Z* – регистровые пары *R*27: *R*26, *R*29: *R*28 и *R*31: *R*30.

Лабораторная работа №2

Программирование встроенных периферийных устройств AVR–микроконтроллера

AVR–микроконтроллер имеет большое количество встроенных периферийных устройств: таймеры–счетчики, устройство последовательной передачи данных *UART*, высокоскоростной последовательный интерфейс *SPI*, многоканальный аналого–цифровой преобразователь, двухпроводной интерфейс и др.

Управление работой этих устройств осуществляется посредством записи в их управляющие регистры соответствующих значений. События, связанные с периферийными устройствами, могут определяться либо по прерыванию, либо по опросу регистров состояния. Рассмотрим программирование периферийных устройств *AVR*-

микроконтроллера на примере 8-разрядного таймера-счетчика *Timer*/*Counter*2 микроконтроллера *ATmega*32.

Для управления таймером-счетчиком *Timer*/*Counter*2 предусмотрен регистр *TCCR*2. Он имеет следующую структуру:

7	6	5	4	3	2	1	0
	***WGM*20**	***COM*21**	***COM*20**	***WGM*21**	***CS*22**	***CS*21**	***CS*20**

Биты *WGM*20, *WGM*21 определяют режим работы таймера–счетчика:

Режим	***WGM*21**	***WGM*20**	Описание
0	0	0	Нормальный счет (*Normal*)
1	0	1	Широтно–импульсная модуляция (ШИМ) с коррекцией фазы (*PWM, Phase Correct*)
2	1	0	Сброс таймера по совпадению (*CTC*)
3	1	1	Быстрая ШИМ (*Fast PWM*)

Биты *COM*20, *COM*21 определяют состояние вывода микроконтроллера, который имеет мнемонику *OC*2, например, при *COM*20 = 0 и *COM*21 = 0 вывод *OC*2 от состояния таймера–счетчика не зависит (*OC*2 отключен).

Биты *CS*20, *CS*22 и *CS*22 управляют частотой счета (*F* – частота тактового генератора):

***CS*22**	***CS*21**	***CS*20**	**Частота счета**	***CS*22**	***CS*21**	***CS*20**	**Частота счета**
0	0	0	нет счета	1	0	0	*F* / 64
0	0	1	*F*	1	0	1	*F* / 128
0	1	0	*F* / 8	1	1	0	*F* / 256
0	1	1	*F* / 32	1	1	1	*F* / 1024

Кроме регистра *TCCR*2 таймер–счетчик *Timer*/*Counter*2 имеет также регистр счетчика *TCNT*2 и регистр сравнения *OCR*2. Регистр *TCNT*2 содержит текущее значение, которое при каждом такте счета увеличивается на 1 до величины 255, после чего сбрасывается в 0. Регистр *OCR*2 содержит значение, при совпадении которого со значением в *TCNT*2 генерируется соответствующее событие.

В микроконтроллере *ATmega*32 предусмотрены следующие события, связанные с таймером–счетчиком *Timer*/*Counter*2:

- совпадение значений в регистрах *TCNT*2 и *OCR*2,
- переполнение регистра *TCNT*2.

Указанные события фиксируются в разрядах *OCF*2 (седьмой разряд) и *TOV*2 (шестой разряд) регистра флагов прерываний таймера–счетчика *TIFR* соответственно. Сбрасываются эти разряды либо аппаратно после выполнения подпрограммы обработки соответствующего прерывания, либо программно записью 1.

Отследить наступление событий таймера–счетчика можно с помощью опроса разрядов в регистре *TIFR*. Также можно определить подпрограммы обработки прерываний по совпадению и/или переполнению. Для этого необходимо разрешить генерацию нужного прерывания с помощью записи 1 в один из разрядов регистра маски прерываний таймера *TIMSK* – *OCIE*2 (разряд 7) и *TOIE*2 (разряд 6), выполнить инициализацию соответствующего вектора прерывания и разрешить глобальное прерывание (бит I) в регистре *SREG*.

Вектора прерываний располагаются в начале памяти программ. Так вектора прерываний по совпадению и переполнению от таймера–счетчика *Timer*/*Counter*2 микроконтроллера *ATmega*32 располагаются по адресам 8 и 10 соответственно. Для инициализации векторов прерываний необходимо разместить по указанным адресам команды перехода (например, *RJMP*) на подпрограммы обработки прерываний, которые должна заканчиваться командой *RETI*.

Регистры периферийных устройств – это порты ввода/ввода, обращение к которым осуществляется по соответствующим адресам с помощью команд *IN* и *OUT*. Например, регистры *TCCR*2, *TCNT*2, *OCR*2, *TIFR*, *TIMSK* имеют адреса 37, 36, 35, 57, 56 соответственно.

Работа микроконтроллера связана с автоматизацией процессов сбора информации и управления. Поэтому в *AVR*–микроконтроллерах предусмотрены цифровые порты ввода/вывода, к которым подключаются внешние устройства (по отношению к самому микроконтроллеру). В *ATmega*32 имеются порты *A, B, C* и *D*. Каждый порт имеет три регистра:

- регистр–защелка вывода,
- регистр направления,
- регистр состояния входов.

Например, для порта *A* – это *PORTA, DDRA, PINA*. Каждый регистр является 8–разрядным. Каждый разряд определяет состояние одного из выводов микроконтроллера.

Направление передачи информации определяется регистром направления: 0 в разряде означает ввод, а 1 – вывод бинарного значения (0 или 1). Вывод значения выполняется с помощью записи в регистр *PORTx*, а ввод – чтения из регистра *PINx*. Например, установить значение 1 на выводе, соответствующем 5–у разряду порта *A* можно с помощью следующих инструкций:

```
in    R16,    DDRA
ori   R16,    32
out   DDRA,   R16
in    R16,    PORTA
ori   R16,    32
out   PORTA,  R16
```

2. Порядок выполнения лабораторной работы

1. Ознакомиться с программированием периферийных устройств *AVR*–микроконтроллера;
2. Написать программу на языке ассемблера генерации меандра с частотой 1000, 2000, 3000, 4000 Гц на выводах портов *A, B, C, D* с использованием таймера-счетчика *Timer*/*Counter*2 на языке ассемблера для микроконтроллера *ATmega*32 с частотой тактового генератора F = 8 МГц;
3. Программу ввести, ассемблировать и отладить с помощью *AVR Studio* 4.
4. Ответить на вопросы.

3. Вопросы

11. Встроенные периферийные устройства *AVR*-микроконтроллера.
12. Управление работой встроенных периферийных устройств *AVR*-микроконтроллера.
13. Управление работой таймера–счетчика.
14. События таймера–счетчика.
15. Обработка событий таймера-счетчика.
16. Вектора прерываний.
17. Обработка прерываний.
18. Регистры периферийных устройств.
19. Цифровые порты ввода-вывода.
20. Управление цифровыми портами ввода-вывода.

Лабораторная работа №3

Управление последовательной памятью

При проектировании различных систем на базе микроконтроллера *AVR* часто необходимо организовать считывание и запись значений в модуль внешней памяти. Так как рассматриваемые микроконтроллеры являются устройствами с ограниченными ресурсами, то при выборе внешней памяти кроме стоимости и объема играют роль также ее быстродействие, число интерфейсных линий, протокол передачи и др. технические характеристики. В настоящее время широкое распространение получила последовательная *flash*-память. В качестве примера рассмотрим микросхемы семейства *AT*45 фирмы *Atmel*.

Их отличительными особенностями являются:

- последовательный интерфейс *SPI*;
- диапазоны напряжений питания
  - 4.5 В ... 5.5 В,
  - 2.7 В ...3.6 В;
- одно напряжение питания для всех операций;
- объем памяти 2, 4, 8 и 16 Мбит;
- страничная организация, объем страницы - 264/528 байта;
- два страничных буфера;
- исполнения для бытовой и промышленной аппаратуры;
- высокая скорость программирования – 7 мс на страницу.

Управление осуществляется командами, обеспечивающими выполнение следующих операций.

1. Группа *A* – операции использующие матрицу основной памяти:

- чтение страницы основной памяти;
- пересылка страницы основной памяти в буфер;
- сравнение данных страницы основной памяти и содержимого буфера;
- стирание и программирование страницы основной памяти содержимым буфера;
- программирование страницы основной памяти содержимым буфера;
- программирование страницы основной памяти;
- автоматическая перезапись страницы.

2. Группа *B* – операции с буферами и регистром состояния:

- чтение содержимого буфера;
- запись данных в буфер;
- чтение регистра состояния.

При выполнении какой-либо операции группы *A* другая операция из этой группы не может быть запущена. При этом может быть запущена операция группы *B*, что позволяет реализовать виртуальный режим непрерывного потока данных – при программировании страницы основной памяти данными из буфера 1, другие данные можно загружать в буфер 2 (или наоборот). В таблице 1 приведено назначение выводов микросхемы памяти.

Таблица 1. Назначение выводов

Вывод	Назначение вывода	Примечания
#CS	Выбор микросхемы	
SCK	Тактовый сигнал последовательного ввода/вывода	Ввод/вывод одного бита на каждый такт
SI	Последовательный ввод	Ввод старшим битом (*MSB*) вперед
SO	Последовательный вывод	Вывод старшим битом (*MSB*) вперед
#WP	Аппаратная защита от записи	***#WP***=0 - запрещает запись на первых 256 страницах
#RESET	Сброс - начальная установка	***#RESET***=0 - прекращает операцию
RDY/#BUSY	Выход сигнала Готов / Занят	Вывод с открытым стоком

Обращение к микросхеме производится при низком уровне сигнала на выводе **#CS**. Все внутренние операции выполняются встроенным автоматом при внутреннем тактировании. Команды подаются по спаду сигнала *#CS*. Стробирование информации (выводы *SI, SO*) ведется импульсами, подаваемыми на вывод *SCK*. Пересылка начинаются со старшего бита (*MSB*).

Чтение страницы основной памяти или буферов.

По окончании загрузки команды данные последовательно поступают на вывод *SO*. При достижении в процессе чтения конца страницы основной памяти (буфера), чтение продолжается с начала той же самой страницы (буфера).

Пересылка страницы основной памяти в буфер 1 или 2. Сравнение данных страницы основной памяти и содержимого буфера 1 или 2.

Пересылка страницы данных из основной памяти в буфер или сравнение данных начинается по фронту сигнала *#CS*. Момент завершения процесса определяется по регистру состояния.

Запись данных в буфер 1 или 2.

Вводимые данные следуют за битами адреса. По достижении конца буфера запись продолжится с начала буфера до фронта сигнала *#CS*.

Программирование содержимым буфера страницы основной памяти без стирания и со стиранием.

Программирование без стирания используется в том случае, когда все биты страницы находятся в состоянии логической 1. При программировании со стиранием очистка страницы выполняется автоматически перед началом программирования. После ввода команды, по фронту сигнала *#CS* начнется цикл программирования или цикл стирания/программирования. Завершение операции определяется через регистр состояния.

Программирование страницы основной памяти через буфер 1 или 2.

По данной команде данные заносятся в буфер и программируются в заданную страницу основной памяти. Запись в буфер начинается сразу после младшего бита адреса первого байта в буфере. По достижении конца буфера запись продолжится с начала буфера. По фронту сигнала *#CS* начнется внутренний цикл стирания/программирования страницы основной памяти.

Автоматическая перезапись страницы через буфер 1 или 2.

Задаваемая этими командами операция применяется в том случае, когда несколько байтов страницы или несколько страниц данных были перепрограммированы в произвольном порядке. По фронту сигнала *#CS* пересылаются данные из страницы основной памяти в буфер и затем переписывает данные из буфера обратно в ту же самую страницу основной памяти.

Регистр состояния.

Информация регистра используется для определения готовности микросхемы, получения результата сравнения страницы основной памяти и буфера, определения типа микросхемы. После передачи в микросхему последнего бита кода команды следующими восемью тактами *SCK* через вывод *SO* принимаются восемь битов регистра статуса. Пока вывод *#CS* находится в низком состоянии и на *SCK* поступают тактовые импульсы на вывод *SO* циклически выводится текущее состояние регистра статуса. Формат регистра состояния показан в таблице 2.

Таблица 2. Формат регистра состояния

7	6	5	4	3	2	1	0
RDY/#BUSY	*COMP*	*N*	*N*	*N*	*X*	*X*	*X*
0 - занят 1 - готов к операции	0 - совпадает 1 - не	код 0 0 0 - объем прибора 2 Мбит код 0 1 1 - объем прибора 4 Мбит код 1 0 0 - объем прибора 8 Мбит					

	совпадает	X - состояние значения не имеет - зарезервированные биты.

Рассмотрим заголовочный файл *at*45.*h*, который можно использовать в программе на языке *C ImageCraft Compilier* для программирования последовательной памяти.

```
#pragma language=extended
// general
#ifdef AT45DB161B
#define AT45_NAME                "AT45DB161B"
#define AT45_PAGES_COUNT         4096
#define AT45_PAGE_SIZE           528
#define AT45_BLOCKS_COUNT 512
#else
#ifdef AT45D081A
#define AT45_NAME                "AT45D081A"
#define AT45_PAGES_COUNT         4096
#define AT45_PAGE_SIZE           264
#define AT45_BLOCKS_COUNT 512
#else
#ifdef AT45DB321
#define AT45_NAME                "AT45DB321"
#define AT45_PAGES_COUNT         8192
#define AT45_PAGE_SIZE           528
#define AT45_BLOCKS_COUNT 1024
#else
#ifdef AT45DB642
#define AT45_NAME                "AT45DB642"
#define AT45_PAGES_COUNT         8192
#define AT45_PAGE_SIZE           1056
#define AT45_BLOCKS_COUNT 1024
#else
#error "Unknown memory type; set in #define directive befofe including this file."
#endif
#endif
#endif
#endif
```

```
// commands
#define CMD_BUFFER_1                                0x00
#define CMD_BUFFER_2                                0x01
#define CMD_BUFFER_1_WRITE                          0x84
#define CMD_BUFFER_2_WRITE                          0x87
#define CMD_BUFFER_1_READ                           0x54
#define CMD_BUFFER_2_READ                           0x56
#define CMD_B1_TO_MM_PAGE_PROG_WITH_ERASE           0x83
#define CMD_B2_TO_MM_PAGE_PROG_WITH_ERASE           0x86
#define CMD_B1_TO_MM_PAGE_PROG_WITHOUT_ERASE        0x88
#define CMD_B2_TO_MM_PAGE_PROG_WITHOUT_ERASE        0x89
#define CMD_MM_PAGE_PROG_THROUGH_B1                 0x82
#define CMD_MM_PAGE_PROG_THROUGH_B2                 0x85
#define CMD_AUTO_PAGE_REWRITE_THROUGH_B1            0x58
#define CMD_AUTO_PAGE_REWRITE_THROUGH_B2            0x59
#define CMD_MM_PAGE_TO_B1_COMP                      0x60
#define CMD_MM_PAGE_TO_B2_COMP                      0x61
#define CMD_MM_PAGE_TO_B1_XFER                      0x53
#define CMD_MM_PAGE_TO_B2_XFER                      0x55
#define CMD_MAIN_MEMORY_PAGE_READ                   0x52
#define CMD_PAGE_ERASE                              0x81
#define CMD_BLOCK_ERASE                             0x50
#define CMD_FULL_READ                               0xE8

// pins
//hw spi: sck=pb7, mosi=pb5, miso=pb6
//at45: cs=pb4, ready=pc4, spi=spi
#define AT45_CS     4 // pb
#define AT45_READY      4 // pc
#define AT45_SCK  7 // pb
#define AT45_MOSI       5 // pb
#define AT45_MISO       6 // pb

// low_level functions
static void delay_45() { ; }
```

```
static void init_45(void)
{
        DDRB |= (1<<AT45_SCK);
        DDRB |= (1<<AT45_MOSI);
        DDRB &= ~(1<<AT45_MISO);
        PORTB |= (1<<AT45_MISO);
        DDRB |= (1<<AT45_CS);
        PORTB |= (1<<AT45_CS); // no cs at start!
        DDRC &= ~(1<<AT45_READY);
        PORTC |= (1<<AT45_READY); // pullup
}

static void init_SPI_45(void)
{
        DDRB |= (1<<AT45_SCK);
        DDRB |= (1<<AT45_MOSI);
        DDRB &= ~(1<<AT45_MISO);
        PORTB |= (1<<AT45_MISO);
        PORTB |= (1<<AT45_SCK);
SPCR = 0x5f;
SPSR = 0;
}

static void close_SPI_45(void)
{
SPCR = 0;
SPSR = 0;
        DDRB |= (1<<AT45_SCK);
        PORTB |= (1<<AT45_SCK);
        DDRB &= ~(1<<AT45_MOSI);
        DDRB &= ~(1<<AT45_MISO);
        PORTB |= (1<<AT45_MISO);
        PORTB |= (1<<AT45_MOSI);
}
```

```
static unsigned char spi(unsigned char x)
{
        SPDR = x;
        while(!(SPSR & 0x80 /*(1<<SPIF)*/)) ;
        return SPDR;
}

static unsigned char read_SPI_45(void)
{
   return spi(0);
}

static void write_SPI_45(unsigned char data)
{
   spi(data);
}

static void set_CS_45(void)
{
        PORTB |= (1<<AT45_CS);
}

static void clear_CS_45(void)
{
        PORTB &= ~(1<<AT45_CS);
}

static void wait_ready_45(void)
{
        while(!(PINC & (1<<AT45_READY)));
}

// high-level functions
// assume, AT45_init is called first
// address      =       { 0..AT45_PAGE_SIZE-1 }
```

```
// page          =       { 0..AT45_PAGES_COUNT-1 }

void AT45_init(void);
void AT45_erase_all(void);
unsigned char AT45_get_status(void);
void AT45_read_buffer_begin(unsigned int address);
unsigned char AT45_read(void) { return read_SPI_45(); }
void AT45_read_buffer_end(void);
void AT45_write_buffer_begin(unsigned int address);
void AT45_write(unsigned char data) { write_SPI_45(data); }
void AT45_write_buffer_end(void);
void AT45_mem_to_buf(unsigned int addr);
void AT45_buf_to_mem(unsigned int addr);

void AT45_init(void)
{
        init_45();
        init_SPI_45();
        clear_CS_45();
        write_SPI_45(0x27);
        write_SPI_45(0x4f);
        write_SPI_45(0x37);
        write_SPI_45(0xa0);
        write_SPI_45(0x17);
        write_SPI_45(0x20);
        wait_ready_45();
        set_CS_45();
        close_SPI_45();
}

void AT45_erase_all(void)
{
        unsigned int i;
        wait_ready_45();
        init_SPI_45();
```

```
        for(i = 0; i < AT45_BLOCKS_COUNT; i++)
        {
                wait_ready_45();
                clear_CS_45();
                write_SPI_45(CMD_BLOCK_ERASE); // ok
                write_SPI_45((char)(i>>4));
                write_SPI_45((char)(i<<4));
                write_SPI_45(0x00);
                set_CS_45();
        }
        close_SPI_45();
        wait_ready_45();
}

unsigned char AT45_get_status(void)
{
        unsigned char data;
        init_SPI_45();
        clear_CS_45();
        write_SPI_45(0x57);
        data = read_SPI_45();
        set_CS_45();
        close_SPI_45();
        return data;
}

void AT45_read_buffer_begin(unsigned int address)
{
        wait_ready_45();
        init_SPI_45();
        clear_CS_45();
        write_SPI_45(CMD_BUFFER_1_READ);
        write_SPI_45(0x00);
        write_SPI_45(0x00);
        write_SPI_45(0x00);
```

```
        write_SPI_45(0x00);
}

void AT45_read_buffer_end(void)
{
        set_CS_45();
        close_SPI_45();
}

void AT45_read_mem_begin(unsigned int addr)
{
        wait_ready_45();
        init_SPI_45();
        clear_CS_45();
        write_SPI_45(CMD_FULL_READ); // ok
        write_SPI_45((char)(addr>>5));
        write_SPI_45((char)(addr<<3));
        write_SPI_45(0x00);
        write_SPI_45(0x00);
        write_SPI_45(0x00);
        write_SPI_45(0x00);
        write_SPI_45(0x00);
}

void AT45_read_mem_end(void)
{
        set_CS_45();
        close_SPI_45();
}

void AT45_write_buffer_begin(unsigned int addr)
{
        wait_ready_45();
        init_SPI_45();
        clear_CS_45();
```

```
        write_SPI_45(CMD_BUFFER_1_WRITE); // ok
        write_SPI_45(0x00);
        write_SPI_45((char)(addr>>8));
        write_SPI_45((char)addr);
    }

    void AT45_write_buffer_end(void)
    {
        set_CS_45();
        close_SPI_45();
    }

    void AT45_buf_to_mem(unsigned int addr)
    {
        wait_ready_45();
        init_SPI_45();
        clear_CS_45();
        write_SPI_45(CMD_B1_TO_MM_PAGE_PROG_WITHOUT_ERASE);     //
ok
        write_SPI_45((char)(addr>>5));
        write_SPI_45((char)(addr<<3));
        write_SPI_45(0x00);
        close_SPI_45();
        set_CS_45();
        wait_ready_45();
    }

    void AT45_mem_to_buf(unsigned int addr)
    {
        wait_ready_45();
        init_SPI_45();
        clear_CS_45();
        write_SPI_45(CMD_MM_PAGE_TO_B1_XFER); // ok
        write_SPI_45((char)(addr>>6));
        write_SPI_45((char)(addr<<2));
```

```
        write_SPI_45(0x00);
        close_SPI_45();
        set_CS_45();
        wait_ready_45();
}


void AT45_mem_0x82(unsigned int addr_page)
{
        wait_ready_45();
        init_SPI_45();
        clear_CS_45();
        write_SPI_45(CMD_MM_PAGE_PROG_THROUGH_B1); // ok
        write_SPI_45((char)(addr_page >> 5));
        write_SPI_45((char)(addr_page << 3));
        write_SPI_45(0x00);
}

void AT45_mem_0x52(unsigned int addr_page, unsigned int addr_byte)
{   unsigned long addr_data;

        addr_data = (unsigned long)addr_page;
        addr_data <<= 11;
        addr_data += (unsigned long)addr_byte;

        wait_ready_45();
        init_SPI_45();
        clear_CS_45();
        write_SPI_45(CMD_MAIN_MEMORY_PAGE_READ); // ok
        write_SPI_45((char)(addr_data >> 16));
        write_SPI_45((char)(addr_data >> 8));
        write_SPI_45((char)addr_data);
        write_SPI_45(0x00);
        write_SPI_45(0x00);
        write_SPI_45(0x00);
```

```
        write_SPI_45(0x00);
    }
```

2. Порядок выполнения лабораторной работы

1. Написать программу управления последовательной памятью *AT*45 фирмы *Atmel* на языке *C ImageCraft Compilier*. Запись и чтение определяются командой, поступающей по последовательному порту *UART*. В режиме записи после команды поступают адрес начальной страницы и число страниц данных, после которых следует записываемая последовательность данных. В режиме чтения после команды также поступают адрес начальной страницы и число страниц данных. После этого требуется считать требуемые данные и передать их по последовательному порту *UART*.
2. Программу отладить с помощью отладчика системы *AVR Studio* 4.
3. Ответить на вопросы.

3. Вопросы

1. Особенности последовательной памяти.
2. Группы команд управления последовательной памятью.
3. Назначение выводов микросхемы памяти.
4. Чтение страницы основной памяти или буферов.
5. Пересылка страницы основной памяти в буфер 1 или 2/сравнение данных страницы основной памяти и содержимого буфера 1 или 2.
6. Запись данных в буфер 1 или 2.
7. Программирование содержимым буфера страницы основной памяти без стирания и со стиранием.
8. Автоматическая перезапись страницы через буфер 1 или 2.
9. Регистр состояния.
10. Описание команд управления памятью на языке *C*.

Лабораторная работа №4

Управляющее устройство на базе микроконтроллера AVR

Одним из основных приложений микроконтроллеров *AVR* является их использование в системах управления. Рассмотрим упрощенный вариант такого устройства.

Устройство управление представляет собой блок на базе микроконтроллера *ATMega*32, к которому подключено четыре управляемых объекта и датчик состояния.

Датчик состояния необходимо опрашивать с определенной частотой. В зависимости от значения с датчика определяется режим работы системы управления.

Управляемые объекты получают управляющие воздействия от контроллера. Сигналы управления формируются в виде последовательности прямоугольных импульсов. При этом единичное значение на входе управляемого устройства обеспечивает его включение, а нулевое – выключение. Управление осуществляется с помощью широтно-импульсной модуляции, когда период прямоугольных импульсов неизменен, а режим работы управляемого объекта определяется длительностью единичного значения.

Работа системы выполняется по жесткой временной программе. Программа состоит из трех составных частей:

1. запуск,
2. рабочий режим,
3. останов.

Запуск осуществляется по команде с пульта управления, который имитируется с помощью кнопки, подключенной к одному из выводов микроконтроллера. Во время запуска необходимо обеспечить заданные состояния управляемых устройств, подавая сочетания управляющих сигналов на их входы, т. е. обеспечить их плавный переход в рабочий режим.

После запуска система переходит в рабочий режим, когда состояние объектов управления определяется значением с датчика состояния.

Останов происходит по команде выключения с пульта управления. При этом необходимо обеспечить плавный переход управляемых объектов в выключенное состояние.

Датчик состояния формирует непрерывные значения. Для их обработки необходимо выполнить их аналого-цифровое преобразование. Аналого-цифровое преобразование выполняется с помощью встроенного преобразователя. Аналого-цифровой преобразователь (АЦП) содержит базовый преобразователь, выполняющий преобразование аналогового сигнала в десятиразрядный двоичный код методом

последовательных приближений, аналоговый мультиплексор для подключения одного из входов микроконтроллера к входу базового преобразователя, регистр управления *ADMUX*, регистр управления-состояния *ADCSR* и шестнадцатиразрядный регистр результата, состоящий из двух восьмиразрядных частей *ADCH* и *ADCL*. Для подачи напряжения питания и опорного напряжения используются отдельные выводы микроконтроллера *AVCC*, *AGND* и *AREF*.

Аналоговые сигналы принимаются на выводы микроконтроллера *ADC*0 – *ADC*7. Величина напряжения аналогового сигнала может находиться в пределах от уровня на шине *AGND* до уровня на шине *AVCC*. Аналоговый мультиплексор подключает один из входов микроконтроллера к входу базового преобразователя. Двоичный код номера подключаемого входа задается комбинацией состояний разрядов *MUX*2, *MUX*1, *MUX*0 регистра *ADMUX*. Преобразование аналогового сигнала в цифровой код выполняется под управлением тактового сигнала в пределах от 50 кГц до 200 кГц. Тактовый сигнал преобразователя формируется в пересчетной схеме путем деления частоты тактового сигнала микроконтроллера. Пересчетная схема работает при единичном состоянии разряда *ADEN* регистра *ADCSR*. Коэффициент деления частоты определяется комбинацией состояний разрядов *ADPS*2, *ADPS*1, *ADPS*0 регистра *ADCSR* в соответствии с таблицей 1.

Таблица 1. Коэффициент деления тактовой частоты АЦП

*ADPS*2	*ADPS*1	*ADPS*0	Коэффициент деления
0	0	0	2
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

Преобразование начинается при установке в единичное состояние разряда *ADSC* регистра *ADCSR*. Разряд *ADSC* сохраняет единичное состояние до завершения преобразования и затем аппаратно переводится в нулевое состояние. Сформированный десятиразрядный код переписывается в регистр результата. При этом устанавливается в единичное состояние разряд *ADIF* регистра *ADCSR* и при единичном состоянии разряда *ADIE* регистра *ADCSR* в блок прерываний поступает запрос прерывания. Разряд *ADIF*

сбрасывается в нулевое состояние аппаратно при переходе микроконтроллера к выполнению подпрограммы обработки прерывания или при выполнении команды установки этого бита в единичное состояние. Чтение результата из соответствующего регистра должно начинаться с чтения младшего байта *ADCH*.

Преобразователь может работать в одиночном и циклическом режимах. Выбор режима определяется состоянием разряда *ADFR* регистра *ADCSR*. При его нулевом значении преобразователь работает в одиночном режиме.

В одиночном режиме преобразование начинается при установке в единичное состояние разряда *ADCS* и выполняется за 14 тактов. Для выполнения следующего преобразования необходимо вновь установить в единичное состояние этот разряд. В циклическом режиме преобразование (после установки в единичное состояние разряда *ADCS*) выполняется за 13 тактов, после чего сразу начинается следующее преобразование. Работа в циклическом режиме прекращается после сброса в нулевое состояние *ADFR*.

Для уменьшения помех, вызываемых работой процессора, предусмотрена возможность выполнения преобразования с переводом контроллера в режим холостого хода.

В некоторых микроконтроллерах имеется возможность подключения к входу базового преобразователя источника эталонного напряжения. Выбор опорного напряжения определяется комбинацией состояний разрядов *REFS*1 и *REFS*0 регистра *ADMUX*. Кроме этого, в некоторых микроконтроллерах с помощью установки в единичное состояние разряда *ADLAR* регистра *ADMUX* можно обеспечить запись результата в старшие десять разрядов регистра результата. Тогда при чтении только *ADCH* получают восьмиразрядный код результата.

Кроме аналого-цифрового преобразование значение с датчика состояния необходимо отфильтровать, а также масштабировать. Фильтрация заключается в выделении полезного значения на фоне помех, которые формируются из-за работы самого микроконтроллера, работы подключенных устройств и др. причин. Одним из самых распространенных методов фильтрации в микропроцессорных системах является усреднение значений за определенный промежуток времени. Масштабирование – это преобразование значения к виду, требуемому для обработки, например, преобразование условных единиц в физические.

Для обеспечения ШИМ-управления объектами, подключенными к управляющему устройству можно воспользоваться встроенными таймерами-счетчиками. При этом весь период сигнала разбивается на дискретные интервалы времени, в течение которых выставляется то или иное значение.

Еще одной проблемой является обеспечение надежной работы пульта управления, имитируемым кнопкой. В данном случае необходимо подавить так называемый дребезг контактов, когда при однократном нажатии на кнопку возникает процесс многократного изменения значения сигнала, поступающего от кнопки. Для его подавления обычно используют временные задержки.

2. Порядок выполнения лабораторной работы

1. Написать программу управления на языке *C CodeVisionAVR*. Режим запуска системы задается в таблице 2. Режим останова определяется следующим образом: выключение объектов управления выполняется в случае их работы не более чем на 10%, а плавный переход к этому значению должен осуществляться с приращением не более чем на -2% за 0.1 с. Рабочий режим задается в таблице 3. Переход из одного состояния в другое должен осуществляться с приращениями не более чем ±5%. Датчик подключен к 3-у каналу АЦП. Управляемые объекты подключены к выводам 0 – 3 порта *B*. Кнопка, имитирующая пульт управления, подключена к выводу 4 порта *B*. Управление объектами должно производиться ШИМ-сигналами с частотами 1000, 2000, 3000, 4000 Гц. Длительность этапов запуска рвана 10 с, 10 с, 20 с, 15 с, 5 с, 20 с, 10 с, 10 с соответственно. Использовать микроконтроллер *ATMega*32 с тактовой частотой 1 МГц.
2. Программу отладить с помощью отладчика системы *AVR Studio* 4.
3. Ответить на вопросы.

3. Вопросы

1. Микроконтроллер в системе управления.
2. Аналого-цифровое преобразование.
3. Регистры аналого-цифрового преобразователя.
4. Выбор входа для аналого-цифрового преобразования.
5. Установка коэффициента деления частоты аналого-цифрового преобразования.
6. Запуск аналого-цифрового преобразования.
7. Режимы работы аналого-цифрового преобразователя.
8. Масштабирование значения от датчика.
9. Фильтрация значения от датчика.
10. Подавление дребезга контактов.

Таблица 2. Запуск системы

	ШИМ-значения (в процентах от периода) сигналов управления объектами (**объект1/объект2/объект3/объект4**) на этапах запуска							
	1	2	3	4	5	6	7	8
	10 /10/10/10	10 /15/15/10	20 /15/15/10	30 /30/40/20	50 /40/45/40	60 /60/60	70 /70/80/90	90 /90/90/90
	5/ 10/15/10	10 /15/15/10	20 /30/30/10	30 /30/40/20	50 /40/45/50	60 /60/60	70 /80/80/90	90 /90/90/90
	10 /15/10/10	20 /15/15/10	20 /35/25/20	30 /40/40/20	50 /40/50/40	60 /60/60	70 /80/85/90	90 /90/90/90
	10 /10/10/10	10 /15/15/10	20 /15/30/10	30 /30/40/20	50 /40/50/40	60 /60/60	70 /70/80/95	90 /90/90/95
	10 /20/15/15	10 /15/25/20	20 /15/25/20	30 /30/40/40	50 /40/40/40	60 /60/60	70 /75/85/90	90 /90/90/90
	5/ 10/15/20	10 /15/15/30	20 /30/30/40	30 /30/40/50	50 /40/50/60	60 /60/60	60 /80/80/90	90 /90/90/90
	15 /15/10/10	15 /15/15/15	20 /15/50/30	30 /30/40/40	50 /40/45/60	60 /60/60	70 /70/90/90	90 /90/90/90
	5/ 5/5/5	10 /10/10/10	20 /15/15/20	30 /40/40/30	50 /40/45/40	60 /60/60	70 /70/80/90	90 /90/90/90
	5/ 15/10/10	15 /15/15/15	20 /15/50/35	30 /30/40/50	40 /40/45/60	60 /60/60	70 /80/90/90	90 /90/90/90
0	15 /15/15/15	15 /15/20/20	20 /15/20/25	30 /20/40/30	40 /40/45/40	60 /60/60	70 /75/80/90	90 /90/90/90

Таблица 3. Рабочий режим

	Значение с датчика	Первый объект	Второй объект	Третий объект	Четвертый объект
	0..255 256…511 512...767 768…1023	100 70 50 20	10 70 50 60	100 75 60 10	100 60 50 10
	0..200 201…500	100 75 60	30 70 50	100 90 50	20 40 50

	501...900 901…1023	10	80	20	80
	0..255 256…511 525...800 801…1023	100 90 50 20	100 70 70 20	20 50 90 100	20 70 50 30
	0..255 256…511 512...1000 1001…1023	20 50 90 100	100 50 60 90	100 75 40 15	80 70 50 20
	0..100 101…500 501...800 801…1023	100 75 40 15	100 70 55 25	20 70 100 50	90 70 50 40
	0..199 200…800 801...1000 1001…1023	20 70 100 50	100 60 50 10	100 70 50 30	100 70 50 20
	0..50 51…767 768...800 801…1023	100 70 50 30	20 40 50 80	100 90 80 70	100 90 80 70
	0..60	100	20	70	70

	61…511 512...767 768…1023	90 80 70	70 50 30	80 90 100	80 90 100
	0..127 128…664 665...767 768…1023	70 80 90 100	80 70 50 20	20 40 50 80	20 40 50 80
0	0..10 11…100 101...1000 1001…1023	100 60 40 10	90 70 50 40	80 70 50 20	100 90 80 70