Направление подготовки: 140400 Электроэнергетика и электротехника

Профиль подготовки: Электрический транспорт

Квалификация (степень) выпускника: магистр

Форма обучения: очная

Дисциплина:

"Информационные и компьютерные технологии в электротехнике"

Комаров В.Г. Лабораторная работа 3 15.09.2026 г.

Методические указания

Тема: Структура программ микроконтроллеров

Цель: *Освоить разработку программ для микроконтроллеров на языке ассемблера на примерах устройств ввода.вывода и их отладку с помощью симулятора.*

Задание: *разработать программу и промоделировать исполнение основных команд для работы устройств ввода/вывода в среде программирования AVR Studio на основе выданного задания.*

Общие сведения

Структура программ для микроконтроллеров на ассемблере тесно связана с архитектурой микроконтроллера и особенностями работы его памяти. Программа размещается в **памяти программ** (обычно Flash-памяти), а для хранения данных используется **память данных** (ОЗУ, EEPROM и т.д.). omgtu.rulibeldoc.bsuir.by

Структура программы на ассемблере для микроконтроллера включает чётко выделенные разделы: вектор сброса, инициализация, основной цикл, обработчики прерываний и раздел данных. Ключевую роль играет таблица векторов, которая связывает аппаратные события с соответствующими подпрограммами. Понимание этой структуры необходимо для эффективной работы с низкоуровневым программированием.

Основные элементы структуры программы

1. Вектор сброса (Reset Vector). Это адрес в памяти программ, с которого начинается выполнение после сброса или подачи питания. Обычно он располагается по нулевому адресу (0x0000). По этому адресу должна находиться команда безусловного перехода, например, rjmp RESET, которая передаст управление инициализационной части программы.

libeldoc.bsuir.by

2. Таблица векторов прерываний. Располагается, как правило, в самом начале

памяти программ (например, у AVR — по адресу 0). Это массив записей, каждая из которых содержит адрес обработчика соответствующего прерывания. В ассемблерном коде это выглядит как таблица косвенных переходов (rjmp), где каждому вектору соответствует метка обработчика (например,

rjmp TIMER0_COMP_vect

). microsin.net
3. Раздел инициализации (init или reset). Здесь настраиваются все периферийные устройства: порты ввода-вывода, таймеры, АЦП, интерфейсы связи (UART, SPI, I^2C). Например, для AVR это настройка DDRB (направление данных для порта B) как выхода и включение подтягивающего резистора на PD2.
4. Основной цикл (main или loop). После настройки периферии программа входит в бесконечный цикл, где в постоянном опросе датчиков или активном ожидании событий выполняет основную задачу.
5. Обработчики прерываний. Это подпрограммы, которые выполняются при возникновении события (нажатие кнопки, завершение АЦП, переполнение таймера). В ассемблерном коде это отдельные метки, на которые делаются переходы из таблицы векторов.
6. Секция объявления данных (Data Section). В этой области размещаются инициализированные данные: константы, массивы, переменные. Ассемблер позволяет задавать их с помощью директив, таких как

.equ

.def

.db

Особенности организации памяти

Микроконтроллеры часто используют **Гарвардскую архитектуру**, при которой память физически разделена на две части: отдельно память программ (Flash) и отдельно память данных (SRAM, EEPROM). Это влияет на то, как программа размещается и организуется в памяти. libeldoc.bsuir.byelar.urfu.ru

- **Таблица векторов прерываний** обычно располагается в начале памяти программ и содержит адреса обработчиков.
- **Стек** размещается в оперативной памяти (ОЗУ) и используется для временного хранения данных при вызовах подпрограмм и обработке прерываний.
- **Регистры общего назначения (РОН)** — это сверхбыстродействующая память внутри процессора, активно используемая при выполнении команд.

microsin.netlibeldoc.bsuir.by

Директивы ассемблера

Помимо команд процессора, в ассемблерном коде используются **директивы** — команды для управления самим ассемблером. Они начинаются с точки и позволяют задавать параметры сборки, размещать данные в памяти, резервировать память и т.д.. omgtu.ru

Комментарии

Для пояснения кода используются комментарии. В ассемблере для однострочных

комментариев применяются символы

;

или

//

, для многострочных —

/* */

. omgtu.ru

Пример шаблона программы на ассемблере AVRASM.

Все программы на микроконтроллерах обычно зацикленные, т.е. имеется какой то главный цикл, который выполняется непрерывно.

Структура программы при этом следующая:

- **Макросы и макроопредения**
- **Сегмент ОЗУ**
- **Точка входа** — ORG 0000
- **Таблица векторов** — и вектора, ведущие в секцию обработчиков прерываний
- **Обработчики прерываний** — тела обработчиков, возврат отсюда только по RETI
- **Инициализация памяти** — а вот уже отсюда начинается активная часть программы
- **Инициализация стека**
- **Инициализация системы ввода-вывода** — программирование и запуск в работу АЦП, компараторов, таймеров, интерфейсов, выставление портов ввода-вывода, разрешение прерываний и т.п..
- **Инициализация внешней периферии** — инициализация дисплеев, внешней памяти, разных аппаратных расширений, что подключены к микроконтроллеру извне.
- **Запуск фоновых процессов** — процессы работающие непрерывно, вне зависимости от условий. Такие как сканирование клавиатуры, обновление экрана и так далее.
- **Главный цикл** — тут уже идет вся управляющая логика программы.
- Сегмент EEPROM

Начинается все с макросов, их пока немного, по ходу добавим.

```
1          .include "m16def.inc"   ; Используем ATMega16
2
3  ;= Start macro.inc ======================================
4          .macro    OUTI
5          LDI    R16,@1
6          .if @0 < 0x40
7          OUT    @0,R16
8          .else
9          STS      @0,R16
10         .endif
11         .endm
12
13         .macro    UOUT
14         .if     @0 < 0x40
15         OUT     @0,@1
16         .else
```

```
17          STS     @0,@1
18          .endif
19          .endm
20 ;= End  macro.inc ====================================
```

В оперативке пока ничего не размечаем.

```
1 ; RAM =============================================
2                   .DSEG
3 ; END RAM =========================================
```

Точка входа и таблица векторов прерываний:

```
1  ; FLASH ============================================
2            .CSEG
3            .ORG $000      ; (RESET)
4            RJMP   Reset
5            .ORG $002
6            RETI             ; (INT0) External Interrupt Request 0
7            .ORG $004
8            RETI             ; (INT1) External Interrupt Request 1
9            .ORG $006
10           RETI          ; (TIMER2 COMP) Timer/Counter2 Compare Match
11           .ORG $008
12           RETI             ; (TIMER2 OVF) Timer/Counter2 Overflow
13           .ORG $00A
14           RETI         ; (TIMER1 CAPT) Timer/Counter1 Capture Event
15           .ORG $00C
16           RETI             ; (TIMER1 COMPA) Timer/Counter1 Compare Match A
17           .ORG $00E
18           RETI             ; (TIMER1 COMPB) Timer/Counter1 Compare Match B
19           .ORG $010
20           RETI             ; (TIMER1 OVF) Timer/Counter1 Overflow
21           .ORG $012
22           RETI             ; (TIMER0 OVF) Timer/Counter0 Overflow
23           .ORG $014
24           RETI             ; (SPI,STC) Serial Transfer Complete
25           .ORG $016
26           RETI                ; (USART,RXC) USART, Rx Complete
27           .ORG $018
28           RETI             ; (USART,UDRE) USART Data Register Empty
29           .ORG $01A
30           RETI             ; (USART,TXC) USART, Tx Complete
31           .ORG $01C
32           RETI         ; (ADC) ADC Conversion Complete
33           .ORG $01E
34           RETI             ; (EE_RDY) EEPROM Ready
35           .ORG $020
36           RETI             ; (ANA_COMP) Analog Comparator
37           .ORG $022
38           RETI             ; (TWI) 2-wire Serial Interface
39           .ORG $024
40           RETI             ; (INT2) External Interrupt Request 2
41           .ORG $026
42           RETI             ; (TIMER0 COMP) Timer/Counter0 Compare Match
43           .ORG $028
44           RETI             ; (SPM_RDY) Store Program Memory Ready
45
46           .ORG   INT_VECTORS_SIZE        ; Конец таблицы прерываний
```

Обработчики пока тоже пусты,добавим пр инаписании конкретных программ:

```
1 ; Interrupts ==========================================
```

```
2 ; End Interrupts ========================================
```

Инициализация памяти

При включении питания в оперативке далеко не всегда все байты равны 0xFF. Они могут быть равны 0xFF, но не обязаны. Равно как и регистры РОН не всегда равны нулю при запуске. Обычно да, все обнулено, но бывают случаи когда после перезапуска и/или включения-выключения питания, микроконтроллер начинет творить не пойми что. Особенно часто это возникает, когда питание выключаешь, а потом, спустя некоторое время, пара минут, не больше, включаешь. А всему виной остаточные значения в регистрах.

Чтобы предотвратить это после каждого включения необходимо делать зануление памяти и очистку всех регистров. Делается это в разделе инициализации, даже до инициализации стека, в виде цикла. Вот примерный вариант соответствующего модуля программы:

```
1  RAM_Flush:      LDI     ZL,Low(SRAM_START)      ; Адрес начала ОЗУ в индекс
2                  LDI     ZH,High(SRAM_START)
3                  CLR     R16                     ; Очищаем R16
4  Flush:          ST      Z+,R16                  ; Сохраняем 0 в ячейку памяти
5                  CPI     ZH,High(RAMEND+1)       ; Достигли конца оперативки?
6                  BRNE    Flush                   ; Нет? Крутимся дальше!
7
8                  CPI     ZL,Low(RAMEND+1)        ; А младший байт достиг конца?
9                  BRNE    Flush
10
11                 CLR     ZL                      ; Очищаем индекс
12                 CLR     ZH
```

Поскольку адрес оперативки двубайтный, то мы вначале смотрим, чтобы старший байт совпал с концом, а потом добиваем оставшиеся 255 байт в младшем байте адреса. Далее очищаем все регистры от первого до последнего. Все, контроллер готов к работе.

```
1                 LDI     ZL, 30          ; Адрес самого старшего регистра
2                 CLR     ZH              ; А тут у нас будет ноль
3                 DEC     ZL              ; Уменьшая адрес
4                 ST      Z, ZH           ; Записываем в регистр 0
5                 BRNE    PC-2            ; Пока не перебрали все не успокоились
```

В принципе значения можно сразу же инициализировать нужными величинами. Но, обычно, всё зануляют.

Инициализация ядра. Память, стек, регистры:

```
1  Reset:          LDI R16,Low(RAMEND)             ; Инициализация стека
2                  OUT SPL,R16                     ; Обязательно!!!
3
4                  LDI R16,High(RAMEND)
5                  OUT SPH,R16
6
7  ; Start coreinit.inc
8  RAM_Flush:      LDI     ZL,Low(SRAM_START)      ; Адрес начала ОЗУ в индекс
9                  LDI     ZH,High(SRAM_START)
10                 CLR     R16                     ; Очищаем R16
11 Flush:          ST      Z+,R16                  ; Сохраняем 0 в ячейку памяти
12                 CPI     ZH,High(RAMEND)         ; Достигли конца оперативки?
13                 BRNE    Flush                   ; Нет? Крутимся дальше!
14
15                 CPI     ZL,Low(RAMEND)          ; А младший байт достиг конца?
```

```
16                  BRNE    Flush
17
18                  CLR     ZL                          ; Очищаем индекс
19                  CLR     ZH
20                  CLR     R0
21                  CLR     R1
22                  CLR     R2
23                  CLR     R3
24                  CLR     R4
25                  CLR     R5
26                  CLR     R6
27                  CLR     R7
28                  CLR     R8
29                  CLR     R9
30                  CLR     R10
31                  CLR     R11
32                  CLR     R12
33                  CLR     R13
34                  CLR     R14
35                  CLR     R15
36                  CLR     R16
37                  CLR     R17
38                  CLR     R18
39                  CLR     R19
40                  CLR     R20
41                  CLR     R21
42                  CLR     R22
43                  CLR     R23
44                  CLR     R24
45                  CLR     R25
46                  CLR     R26
47                  CLR     R27
48                  CLR     R28
49                  CLR     R29
50 ; End coreinit.inc
```

Весь этот листинг является заголовочной частью программы, который желательно спрятать в inc файл и больше не трогать.

Секции внешней и внутренней инициализации периферии пока пусты, как и запуск фоновых программ. Они заполняются в процессе разработки конкретных программ.

```
1  ; Internal Hardware Init  ====================================
2
3  ; End Internal Hardware Init ================================
4
5  ; External Hardware Init  ====================================
6
7  ; End Internal Hardware Init ================================
8
9  ; Run ========================================================
10
11 ; End Run ====================================================
```

А теперь, собственно, сам главный цикл.

```
1 ; Main =======================================================
2 Main:
3
4                   JMP     Main
5 ; End Main ====================================================
```

Все процедуры располагаются в отдельной секции, не смешиваясь с главным циклом. Так удобней, потом можно их по частям вынести в библиотечные файлы и разделить исходник на несколько файлов. Но мы пока это делать не будем. Разделим их просто логически.

```
1 ; Procedure ====================================================
2
3 ; End Procedure ================================================
```

Простой пример структуры программы на ассемблере

```
; Шапка программы (служебная информация для ассемблера)
…

; Секция инициализации микроконтроллера и внешних устройств
reset:
…

; Основной цикл
main:
; Опрос датчиков и обработка событий
…

; Обработчик внешнего прерывания
…

INT0_vect:
; Обработка прерывания
…

; Таблица векторов прерываний (обычно в начале памяти программ)
.org 0x0000
rjmp reset
.org 0x0002
…
```

Устройства ввода/вывода

К универсальным устройствам ввода/вывода дискретных сигналов относятся параллельные и последовательные порты ввода/вывода, а аналоговых сигналов компараторы (аналого-логические преобразователи), аналого-цифровые (АЦП) и цифро-аналоговые (ЦАП) преобразователи. Любой микроконтроллер AVR имеет от одного до семи параллельных портов ввода-вывода (обычно восьмиразрядных). Каждый порт имеет свое имя (от A до G). Каждый разряд такого порта подсоединен к одному из выводов (контактов) микросхемы. Для управления каждым портом ввода-вывода используются три специальных регистра :
PORTx – регистр данных для вывода информации;

DDRx – регистр указания направления передачи информации;

PINx – регистр для ввода информации,

где x = A,…,G указатель порта.

Отдельные разряды регистров также имею свои имена:

для PORTx это Pxn, где n – номер разряда (n=0,…,7).

Аналогично: DDRx – DDxn; и PINx – PINxn.

Любой вывод портов может работать как на вывод так и на ввод, что определяется состоянием соответствующего разряда управляющего регистра DDRx:

если в этом разряде записана логическая единица, то разряд порта работает на выход,

если нуль - то как вход. Чтобы выдать байт информации на внешний вывод МК, нужно:

- в соответствующий разряд DDRx записать логическую единицу,

- а затем записать байт в регистр PORTx.

Чтобы считать информацию с внешнего вывода МК, нужно:

- в соответствующий разряд DDRx записать логический ноль,

- только затем подать сигнал от внешнего устройства, после чего МК читает байт в регистр PINx и соответствующий разряд этого байта соответствует сигналу на контакте порта МК. Схема организации ввода-вывода МК AVR представлена на рисунке 1.

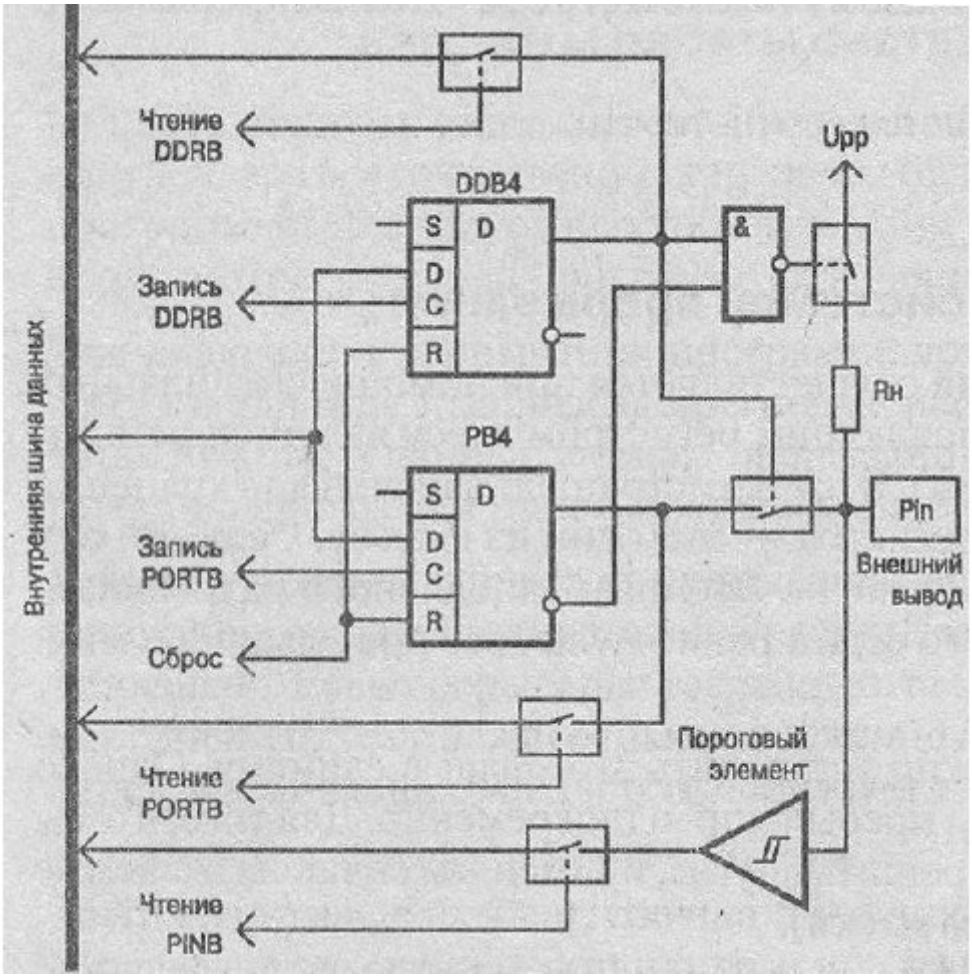

Рисунок 1 – Функциональная схема параллельного порта ввода-вывода МК AVR

Как видно их схемы, в режиме ввода информации возможно подключение контакта МК для ввода к внутреннему нагрузочному резистору (подтяжка к напряжению питания +5 В). Для этого нужно предварительно установить в единицу соответствующий разряд регистра Pxn так как это показано табл. 1.

Таблица 1. Конфигурирование входного порта

DDxn	Pxn	Режим	Резистор	Примечание
0	0	Вход	Отключен	Вывод отключен от схемы
0	1	Вход	Подключен	Вывод является источником тока
1	0	Выход	Отключен	На выходе «0»
1	1	Выход	Отключен	На выходе «1»

Для программирования ввода-вывода микроконтроллеров AVR на ассемблере используются специальные команды, которые работают с регистрами ввода-вывода. Они позволяют загружать данные из порта в регистр и выводить данные из регистра в порт.
Особенности работы с портами:

- Если вывод порта сконфигурирован как выход, то его переключение производится через регистр данных (PORTA, PORTB, PORTC, PORTD).
- Если вывод сконфигурирован как вход, то его опрос следует производить через регистр выводов входа порта (PINA, PINB, PINC, PIND).

Ввод

Пример команды для загрузки данных с порта ввода-вывода PORTB в регистр R16:

in r16, PORTB

Вывод

Пример команды для вывода данных из регистра R16 на порт ввода-вывода PORTB:

out PORTB, r16

Примеры программ

1. Пример программы для ввода и вывода данных через порты ввода-вывода:

```
in r16, PORTB ; Загрузить данные с порта ввода-вывода PORTB в регистр R16
out PORTB, r16 ; Вывести данные из регистра R16 на порт ввода-вывода PORTB
```

2. Пример программы для инициализации порта ввода-вывода и мигания светодиодом:

```
.include <avr/io.h>

.equ LED_PORT, PORTB
.equ LED_DDR,  DDRB
.equ LED_PIN,  PINB
.equ LED_BIT,  PB0

ldi r16, 0xFF   ; Установить все биты порта B как выходы
out LED_DDR, r16

main_loop:
    sbi LED_PORT, LED_BIT   ; Установить бит PB0 в 1 (включить светодиод)
```

```
    _delay_ms(1000)          ; Подождать 1 секунду
    cbi LED_PORT, LED_BIT    ; Установить бит PB0 в 0 (выключить
светодиод)
    _delay_ms(1000)          ; Подождать еще 1 секунду
    rjmp main_loop             ; Бесконечный цикл
```

3. Пример программы для управления шаговым двигателем:

```
.include <avr/io.h>

.equ motor_PORT, PORTD
.equ motor_DDR,  DDRD

.equ STEP_PIN,  PD2
.equ DIR_PIN,   PD3
.equ MS1_PIN,   PD4
.equ MS2_PIN,   PD5
.equ ENABLE_PIN, PD6

.org 0x00
    rjmp reset

reset:
    ldi r16, (1 << STEP_PIN) | (1 << DIR_PIN) | (1 << MS1_PIN) | (1 <<
MS2_PIN) | (1 << ENABLE_PIN)
    out motor_DDR, r16  ; Установить все пины управления драйвером мотора
как выходы

main_loop:
    ; Управление мотором, включая изменение направления вращения, шагов и
режима шага
    ; Здесь можно добавить логику для движения двигателя в нужном
направлении и количестве шагов
    rjmp main_loop
```

Контрольные вопросы

1. Какие устройства ввода/вывода в микроконтроллерах AVR вы знаете и как они работают?
2. Какие есть регистры в портах ввода/вывода и какие функции они выполняют?
3. Как можно в программе указать адрес порта ввода/вывода микроконтроллера?
4. Как и в какой части программы настраивается порт ввода/вывода?
5. Как и зачем используют команды операций с битами при работе с портами ввода/вывода?
6. Как переслать данные в устройства ввода/вывода?

Задание

На основе выданного Вам преподавателем вопроса напишите программу на языке ассемблера, демонстрирующую ответ на заданный вопрос с помощью симулятора AVR-studio.

Ответ оформите в виде отчёта.

Приложение 1

Титульный лист отчёта по лабораторной работе

ФГБОУ ВО

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ "МЭИ"

ИНСТИТУТ ЭЛЕКТРОТЕХНИКИ

КАФЕДРА ЭКАО И ЭТ

Направление подготовки: 13.04.02 Электроэнергетика и электротехника

Профиль подготовки: Теория движения электроподвижного состава и проблема оптимизации тягового оборудования и устройств электроснабжения транспортных систем

Квалификация (степень) выпускника: магистр

Форма обучения: очная

ОТЧЁТ

по лабораторной работе № 3

по дисциплине **ИЭТ; Б1.Б.3 "Информационные и компьютерные технологии в электротехнике"**

Тема: Структура программ микроконтроллеров

Студент_____

Фамилия, и., о. *группа* *подпись*

Научный руководитель_____доцент__к.т.н._Комаров_В.Г.

должность, звание, фамилия, и., о.

Допущен	Принято	Оценка

Москва 2026 г.