Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

Кафедра вычислительной техники

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5**

**”EEPROM”**

**Вариант №7**

|  |  |  |  |
| --- | --- | --- | --- |
| Преподаватель |  | подпись, дата | Яблонский А.П. |
| Студент КИ19-09Б | 031940750  номер зач. книжки | подпись, дата | Кудрявцев Н.М. |

Красноярск 2022 г.

**ВВЕДЕНИЕ**

**Цель работы**: изучить операции чтения/записи данных в EEPROM, научится применять макросы для структурирования кода.

**Задание**: необходимо разработать две программы.

1. Первая программа формирует массив С из лабораторной работы №2 и помещает его в EEPROM.
2. Вторая программа вы вычитывает массив С из EEPROM и выводит на светодиодные инжекторы минимально или максимальное число согласно варианту на 2 лабораторную работу.

Все обращения к памяти выполнить виде макросов.

**Команды:**

Справка по Ассемблеру для Atmel AVR:

<https://dfe.karelia.ru/koi/posob/avrlab/avrasm-rus.htm>

.macro load\_x ; Объявление макроса

ldi xl, low(@0) ; Регистровая пара

ldi xh, high(@0) ; @0 - первый параметр, @1 - второй параметр…

.endmacro

sts min\_memory, min\_element ─ Прямое сохранение **min\_element** в **min\_memory**;

push E\_ADR\_H ─ Занесение регистра **E\_ADR\_H** в стек;

pop E\_ADR\_H ─ Извлечение регистра **E\_ADR\_H** из стека;

SBIC EECR, EEWE ─ Пропустить если бит **EEWE** в порту **EECR** очищен;

SBI EECR, EEWE ─ Установить бит **EEWE** в порту **EECR**;

Посредством регистров адреса **EEARH** и **EEARL** определяется адрес в пространстве адресов EEPROM емкостью 2К / 4К.

* **Bits 7-0 - Данные EEPROM**

В процессе выполнения операции записи в **регистре данных EEDR** содержатся данные, записываемые в EEPROM по адресу, задаваемому **регистром адреса EEAR**. При чтении регистр EEDR содержит данные, считываемые из EEPROM по адресу, определяемому EEAR.

**EECR** (регистр управления):

* **Bits 7 - 4 - Зарезервированные биты**;
* **Bit 3 - Разрешение прерывания по готовности EEPROM;**
* **Bit 2 - Управление разрешением записи EEPROM;**
* **Bit 1 - Разрешение записи EEPROM;**
* **Bit 0 - Разрешение чтения EEPROM.**

**Ход работы:**

ПРОГРАММА №1:

.include "lib.asm"

.def counter = r16 // счетчик

.def alt\_counter = r17 // второй счетчик

.def a\_current = r18 // элемент масива А

.def b\_current = r19 // элемент масива B

.def c\_current = r20 // элемент масива B

.def bool = r21 // логическая переменная

.def mx = r22 // логическая переменная

.def counterWorkSize = r23 // счетчик заполненной зоны в массиве

.set sizeAB = 10 // размер массивов A и B

.set sizeC = 10 // размер массивов C

.set false = 0 // логический ноль

.set true = 1 // логическая еденица

.dseg // указатель на сегмент данных

.org 0x60 // начальный адрес масивов

arr\_a: .byte sizeAB // Выделяем sizeAB байт на массив А

arr\_b: .byte sizeAB // Выделяем sizeAB байт на массив В

.org 0x7b

arr\_c: .byte sizeC // Выделяем sizeС байт на массив С

.org 0x8c

min\_el: .byte 1 // выделим 1 байт под максимальный элемент

.cseg // указатель на сегмент кода

//ldi xl, low(arr\_c) // X

//ldi xh, high(arr\_c)

//ldi yl, low(arr\_b) // Y

//ldi yh, high(arr\_b)

//ldi zl, low(arr\_a) // Z

//ldi zh, high(arr\_a)

START:

load\_x arr\_c

load\_y arr\_b

load\_z arr\_a

ldi a\_current, 4 // переменная для заполнения массива a

ldi counter, sizeAB // Устанавливаем размер счетчика

MASA: // заполняем массив a

st z+, a\_current // сохранение a\_current в масив Вaи инкремент указателя

inc a\_current

dec counter

brne MASA // условия цикл заполнения массива a

// заполнения масив В

ldi b\_current, 16 // переменная для заполнения массива B

ldi counter, sizeAB // Устанавливаем размер счетчика

MASB: // заполняем массив B

st y+, b\_current // сохранение b\_current в масив В и инкремент указателя

dec b\_current

dec counter

brne MASB // условия цикл заполнения массива В

// Заполним массив С разностью массивов А и В

//ldi zl, low(arr\_a) // Z

//ldi zh, high(arr\_a)

load\_z arr\_a

ldi alt\_counter, sizeAB

ldi counterWorkSize, 0

FILL1AB:

ld a\_current, z+ // a\_current=z, z=z+1

ldi bool, false // bool = true

//ldi yl, low(arr\_b) // Y

//ldi yh, high(arr\_b)

load\_y arr\_b

ldi counter, sizeAB

FILL2AB:

ld b\_current, y+ // b\_current=y, y=y+1

cp b\_current, a\_current // Сравнить b\_current с a\_current

brne RAZN1 // Перейти если b\_current не равно a\_current

ldi bool, true // иначе bool == true

inc counterWorkSize

RAZN1:

dec counter

brne FILL2AB // условия цикл нахождения элемента a\_current в В

cpi bool, true // Сравнить bool с false (true)

brne RAZN2 // Перейти если bool не равно false

st x+, a\_current // x = a\_current, x = x + 1

RAZN2:

dec alt\_counter // декремент счетчика2

brne FILL1AB

; Запись в память

SAVE:

ldi r20,low(ramend)

out spl,r20

ldi r20, high(ramend)

out sph,r20

load\_x arr\_c

ld c\_current, x+

mov counter, counterWorkSize

ldi ZL,0

ldi ZH,0

E\_WRITE ZH, ZL, counterWorkSize

inc ZL

LOOP:

E\_WRITE ZH, ZL,c\_current

inc ZL

ld c\_current, x+

dec counter

brne LOOP

WAIT:

rjmp WAIT

ПРОГРАММА №2:

; поиск минимального элемента

.include "lib.asm"

.def count = r16 ; счетчик

.def c\_elem = r20 ; текущий элемент массива C

.def c\_length = r21 ; длинна заполненой части массива C

.def min\_element = r22

.set start\_max = 255

.dseg

.org 0x9c

mas\_c: .byte 16 ; выделим под массив C 20 байт

.org 0x8a

min\_memory: .byte 1

.cseg

START:

LOAD:

load\_x mas\_c

ldi ZL,0

ldi ZH,0

E\_READ ZH, ZL, c\_length

inc ZL

mov count, c\_length

LOOP:

E\_READ ZH, ZL, c\_elem

st x+, c\_elem

inc ZL

dec count

brne LOOP

load\_z mas\_c

ldi min\_element, start\_max

mov count, c\_length

FIND\_MIN:

ld c\_elem, z+

cp min\_element, c\_elem

brlo SAME\_OR\_HIGHER ; перейти если min\_element >= c\_length

mov min\_element, c\_elem

SAME\_OR\_HIGHER:

dec count

brne FIND\_MIN

sts min\_memory, min\_element

ldi r20, 0xff ; настроqка порта В к которому подключены светодиоды

out DDRB, r20

com min\_element

out PORTB, min\_element ; выключить все светодиоды (включаются нулем)

WAIT:

rjmp WAIT

EEPROM ("lib.asm"):

.include "m32adef.inc"

.def E\_ADR\_H = r16 ; это имя регистра для старшего байта адреса в EEPROM

.def E\_ADR\_L = r17 ; а это для младьшего

.def E\_DATA = r18 ; этот регистр будем использовать для данных EEPROM

.cseg

jmp START

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

.macro load\_x

ldi xl, low(@0) ; X

ldi xh, high(@0)

.endmacro

.macro load\_y

ldi yl, low(@0) ; Y

ldi yh, high(@0)

.endmacro

.macro load\_z

ldi zl, low(@0) ; Z

ldi zh, high(@0)

.endmacro

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

.macro E\_WRITE ; запись в EEPROM

push E\_ADR\_H ; сохраним в стеке регистры, чтобы не испортить

push E\_ADR\_L

push E\_DATA

mov E\_ADR\_H,@0 ; подготовим адрес для записи - это число

mov E\_ADR\_L,@1

mov E\_DATA,@2 ; подготовим данные - они в регистре

rcall EEWrite ; и запишем число по адресу в EEPROM

pop E\_DATA ; восстановим регистры адреса и данных

pop E\_ADR\_L

pop E\_ADR\_H

.endmacro

.macro E\_READ ; чтоение из EEPROM

push E\_ADR\_H ; сохраняием регистры

push E\_ADR\_L

push E\_DATA

mov E\_ADR\_H,@0 ; готовим адрес

mov E\_ADR\_L,@1

rcall EERead ; и читам из этого адреса данные

mov @2,E\_DATA ; передаем результат в регистр

pop E\_DATA ; восстанавливаем старые регистры

pop E\_ADR\_L

pop E\_ADR\_H

.endmacro

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

EEWrite:

SBIC EECR,EEWE ; анализ бита готовности EEPROM

; EEPROM Готова?

RJMP EEWrite ; нет еще - EEWE очищен - ждем

OUT EEARH,E\_ADR\_H ; установка адреса EEROM

OUT EEARL,E\_ADR\_L ;

OUT EEDR,E\_DATA ; загрузка данных

SBI EECR,EEMWE ; разрешить запись

SBI EECR,EEWE ; записать данные

RET ; выход из подпрограммы

EERead:

SBIC EECR,EEWE ; Ждем готовности

; иначе читаем

RJMP EERead ;

OUT EEARL, E\_ADR\_L ; загрузить адрес

OUT EEARH, E\_ADR\_H ; в EEPROM который

; нам нужен

SBI EECR,EERE ; начать чтение

IN E\_DATA, EEDR ; забрать результат

RET