МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра ИиСП

Отчет
по лабораторной работе № 4

по дисциплине «Машинно-зависимые языки программирования»
Вариант 22

Выполнил: ст. гр. ПС-11
Ложкин С.А.
Проверил: доцент, доцент кафедры ИиСП Баев А.А.

г. Йошкар-Ола
2023

Цель работы: получить практический опыт по работе с отладкой приложения и его симуляцией

Задания на лабораторную работу:

1. Сделать отладку и симуляцию работы арифметических и логических операций ассемблера AVR, чтобы проверить изменение флагов при различных значениях операндов.
2. Сделать отладку и симуляцию работы условных переходов ассемблера AVR, чтобы проверить наличие переходов и изменение счётчика команд в тех или иных состояниях программы
3. Выполнить работу с массивами и написать задачу в соответствии с вариантом

1. Теоретические сведения

Для отладки программы и его симуляции необходимо наличие файлов симуляции, в которых описана соответствующая конфигурация, в том числе тесты. С их помощью легче отслеживать изменения в программе во время отладки. К примеру, не придётся делать пошаговые переходы во время задержки, или постоянно выставлять новые значения для тестирования.

Наша задача — указать в этом файле переменные, состояние которых мы хотим отслеживать, начало записи логов в указанный файл, затем сами тесты, после них логирование надо остановить.
Необходимо установить через # количество тактов, чтобы входные значения для тестов не записались все один разом, а по очереди на каждом цикле программы.
Запускаем отладку и последующую паузу программы в Atmel Studio через Debug->Start Debugging and Break. Программа запустилась, но находится перед выполнением первой команды. В этот момент надо через Debug->Execute Stimulifile исполнить файл симуляции. Затем нажимаем F5 для исполнения записанной нами симуляции. После логирования произойдёт пауза в программе, её можно завершить. Как итог, мы получаем выходной файл с логами, по которому можно будет понять состояние программы и результат у каждой операции.

2. **Практическая часть**

Проверка математических и логических операций, работа с битами

ADC

Мнемоника	ADC
Операнды	Rd, Rr
Описание	Суммирование с переносом
Операция	Rd = Rd + Rr + C
Флаги	Z, C, N, V, H, S
Циклы	1
Задача:	проверить изменение флагов при различных значениях операндов

Текст программы

```
setup:
	nop
loop:
	; ввод
	in r18, OCR0A
	in r19, OCR0B
	; выполнение операции
	adc r18, r19
	; вывод
	out OCR0A, r18
	out OCR0B, r19
rjmp loop
```

Файл stimuli

```
$log OCR0A
$log OCR0B
$log SREG
$startlog ADC_log_output.stim

OCR0A = 15
OCR0B = 1
```

#7
OCR0A = 128
OCR0B = 0

#7
OCR0A = 127
OCR0B = 1

#7
OCR0A = 0
OCR0B = 0

#7
OCR0A = 255
OCR0B = 1

#7
OCR0A = 0
OCR0B = 0

#7
$stoplog
$break

Выход

#1
OCR0B = 0x01
OCR0A = 0x0f
#2
SREG = 0x20
#1
OCR0A = 0x10

#4
OCR0B = 0x00
OCR0A = 0x80
#2
SREG = 0x14

#5
OCR0B = 0x01
OCR0A = 0x7f
#2
SREG = 0x2c
#1
OCR0A = 0x80

#4
OCR0B = 0x00
OCR0A = 0x00
#2
SREG = 0x02

#5
OCR0B = 0x01
OCR0A = 0xff
#2
SREG = 0x23
#1
OCR0A = 0x00

#4
OCR0B = 0x00
#2
SREG = 0x00
#1
OCR0A = 0x01

Таблица результатов

	OCR0A	**OCR0B**	**OCR0A(res)**	**SREG**	**Флаги**
1.	0x0F	0x01	0x10	0x20	H
2.	0x80	0x00	0x80	0x14	S, N
3.	0x7F	0x01	0x80	0x2C	H, V, N

4.	0x00	0x00	0x00	0x02	Z
5.	0xFF	0x01	0x00	0x23	H, Z, C
6.	0x00	0x00	0x01	0x00	-

MULSU

Мнемоника MULSU
Операнды Rd, Rr
Описание Умножение числа со знаком с числом без знака
Операция R1 : R0 = Rd * Rr
Флаги Z, C
Циклы 2
Задача: проверить изменение флагов при различных значениях операндов

Текст программы

```
setup:
	nop
loop:
	; ввод
	in r18, OCR0A
	in r19, OCR0B
	; выполнение операции
	mulsu r18, r19
	; вывод
	out OCR0A, r1
	out OCR0B, r0
rjmp loop
```

Файл stimuli

```
$log OCR0A
$log OCR0B
$log SREG
$startlog MULSU_log_output.stim

OCR0A = 255
OCR0B = 1
```

```
#8
OCR0A = 128
OCR0B = 1

#8
OCR0A = 127
OCR0B = 2

#8
OCR0A = 255
OCR0B = 255

#8
OCR0A = 0
OCR0B = 0

#8
$stoplog
$break
```

Выход

```
#2
OCR0B = 0x01
OCR0A = 0xff
#3
SREG = 0x01
#2
OCR0B = 0xff

#3
OCR0B = 0x01
OCR0A = 0x80
#4
OCR0A = 0xff
#1
OCR0B = 0x80

#3
OCR0B = 0x02
```

OCR0A = 0x7f
#3
SREG = 0x00
#1
OCR0A = 0x00
#1
OCR0B = 0xfe

#3
OCR0B = 0xff
OCR0A = 0xff
#3
SREG = 0x01
#2
OCR0B = 0x01

#3
OCR0B = 0x00
OCR0A = 0x00
#3
SREG = 0x02

Таблица результатов

	OCR0A	**OCR0B**	**OCR0A(res)**	**OCR0B(res)**	**SREG**	**Флаги**
1.	0xFF	0x01	0xFF	0xFF	0x01	C
2.	0x80	0x01	0xFF	0x80	0x01	C
3.	0x7F	0x02	0x00	0xFE	0x00	-
4.	0xFF	0xFF	0xFF	0x01	0x01	C
5.	0x00	0x00	0x00	0x00	0x02	Z

SBCI

Мнемоника SBCI
Операнды Rd, K8
Описание Вычитание константы с переносом
Операция Rd = Rd - K8 - C
Флаги Z, C, N, V, H, S
Циклы 1
Задача: проверить изменение флагов при различных значениях операндов

Текст программы

```
loop:
    nop
    ; ввожу второй аргумент нативно,
    ; так как команда по своей сути создана,
    ; для того чтобы сразу вычитать значение,
    ; а не записывать его предварительно в регистр

    ; ввод 16
    in r18, OCR0A
    sbci r18,1
    out OCR0A, r18

    ; ввод 128
    in r18, OCR0A
    sbci r18,0
    out OCR0A, r18

    ; ввод 128
    in r18, OCR0A
    sbci r18,1
    out OCR0A, r18

    ; ввод 0
    in r18, OCR0A
    sbci r18,0
    out OCR0A, r18

    ; ввод 256
```

```
        in r18, OCR0A
        sbci r18,1
        out OCR0A, r18

        ; ввод 0
        in r18, OCR0A
        sbci r18,0
        out OCR0A, r18

        ; ввод 127
        in r18, OCR0A
        sbci r18,-1
        out OCR0A, r18
rjmp loop
```

Файл stimuli

```
$log OCR0A
$log OCR0B
$log SREG
$startlog SBCI_log_output.stim

OCR0A = 16
#3
OCR0A = 128
#3
OCR0A = 128
#3
OCR0A = 0
#3
OCR0A = 256
#3
OCR0A = 1
#3
OCR0A = 127

#3
$stoplog
$break
```

Выход

#1
OCR0A = 0x10
#1
SREG = 0x20
#1
OCR0A = 0x0f

#1
OCR0A = 0x80
#1
SREG = 0x14

#3
SREG = 0x38
#1
OCR0A = 0x7f

#1
OCR0A = 0x00
#1
SREG = 0x00

#3
SREG = 0x35
#1
OCR0A = 0xff

#1
OCR0A = 0x01
#1
SREG = 0x00
#1
OCR0A = 0x00

#1
OCR0A = 0x7f

#1
SREG = 0x0d
#1
OCR0A = 0x80

	OCR0A	Константа	OCR0A(res)	SREG	Флаги
1.	0x10	1	0x0f	0x20	H
2.	0x80	0	0x80	0x14	S, N
3.	0x80	1	0x7f	0x38	Z, N, H
4.	0x00	0	0x00	0x00	-
5.	0x00	1	0xFF	0x35	C, N, H, S
6.	0x01	0	0x00	0x00	-
7.	0x7f	-1	0x80	0x0D	V, N, C

MUL

Мнемоника MUL
Операнды Rd, Rr
Описание Умножение чисел без знака
Операция R1 : R0 = Rd * Rr
Флаги Z, C
Циклы 2
Задача: проверить изменение флагов при различных значениях операндов

Текст программы

```
setup:
    nop
loop:
    ; ввод
    in r18, OCR0A
    in r19, OCR0B
    ; выполнение операции
    mul r18, r19
    ; вывод
```

```
        out OCR0A, r1
        out OCR0B, r0
rjmp loop
```

Файл stimuli

```
$log OCR0A
$log OCR0B
$log SREG
$startlog MUL_log_output.stim

OCR0A = 255
OCR0B = 1

#8
OCR0A = 128
OCR0B = 1

#8
OCR0A = 127
OCR0B = 2

#8
OCR0A = 255
OCR0B = 255

#8
OCR0A = 0
OCR0B = 0

#8
$stoplog
$break
```

Выход

```
#2
```

```
OCR0B = 0x01
OCR0A = 0xff
#4
OCR0A = 0x00
#1
OCR0B = 0xff

#3
OCR0B = 0x01
OCR0A = 0x80
#4
OCR0A = 0x00
#1
OCR0B = 0x80

#3
OCR0B = 0x02
OCR0A = 0x7f
#4
OCR0A = 0x00
#1
OCR0B = 0xfe

#3
OCR0B = 0xff
OCR0A = 0xff
#3
SREG = 0x01
#1
OCR0A = 0xfe
#1
OCR0B = 0x01

#3
OCR0B = 0x00
OCR0A = 0x00
#3
SREG = 0x02
```

Таблица результатов

	OCR0A	OCR0B	OCR0A(res)	OCR0B(res)	SREG	Флаги
1.	0xFF	0x01	0x00	0xFF	0x00	-
2.	0x80	0x01	0x00	0x80	0x00	-
3.	0x7F	0x02	0x00	0xFE	0x00	-
4.	0xFF	0xFF	0xFE	0x01	0x01	C
5.	0x00	0x00	0x00	0x00	0x02	Z

SUB

Мнемоника SUB
Операнды Rd, Rr
Описание Вычитание без переноса
Операция Rd = Rd - Rr
Флаги Z, C, N, V, H, S
Циклы 1
Задача: проверить изменение флагов при различных значениях операндов

Текст программы

```
setup:
    nop
loop:
    ; ввод
    in r18, OCR0A
    in r19, OCR0B
    ; выполнение операции
    sub r18, r19
    ; вывод
    out OCR0A, r18
    out OCR0B, r19
rjmp loop
```

Файл stimuli

```
$log OCR0A
$log OCR0B
$log SREG
$startlog SUB_log_output.stim

OCR0A = 16
OCR0B = 1

#7
OCR0A = 128
OCR0B = 0

#7
OCR0A = 128
OCR0B = 1

#7
OCR0A = 0
OCR0B = 0

#7
OCR0A = 256
OCR0B = 1

#7
OCR0A = 127
OCR0B = 0xFF

#7
$stoplog
$break
```

Выход

```
#1
OCR0B = 0x01
OCR0A = 0x10
#2
SREG = 0x20
```

```
#1
OCR0A = 0x0f

#4
OCR0B = 0x00
```

OCR0A = 0x80
#2
SREG = 0x14

#5
OCR0B = 0x01
#2
SREG = 0x38
#1
OCR0A = 0x7f

#4
OCR0B = 0x00
OCR0A = 0x00
#2
SREG = 0x02

#5
OCR0B = 0x01
#2
SREG = 0x35
#1
OCR0A = 0xff

#4
OCR0B = 0xff
OCR0A = 0x7f
#2
SREG = 0x0d
#1
OCR0A = 0x80

Таблица результатов

	OCR0A	**OCR0B**	**OCR0A(res)**	**SREG**	**Флаги**
1.	0x10	0x01	0x0f	0x20	H
2.	0x80	0x00	0x80	0x14	S, N
3.	0x80	0x01	0x7F	0x38	Z, N, H
4.	0x00	0x00	0x00	0x02	Z
5.	0x00	0x01	0xFF	0x35	C, N, H, S
6.	0x7F	0xFF	0x80	0x0D	V, N, C

Общие выводы изменения флагов:

1. Флаг C ставится, когда возникает заём бита при операции
2. Флаг Z ставится, если результат операции равен нулю
3. Флаг V ставится при переполнении во время операции
4. Флаг N ставится, если результат операции отрицательный
5. Флаг H ставится при переполнении полубайта во время операции

6. Флаги выставляются и остаются неизменными до следующей операции, которая может эти флаги изменить

Условные переходы

BRVS

Мнемоника BRVS
Операнды k
Описание Перейти если флаг переполнения установлен
Операция if (V == 1) PC = PC + k + 1
Флаги -
Циклы 1/2
Счётчик команд: PC = PC + k + 1, если флаг V выставлен, иначе PC = PC +1

Текст программы

```
setup:
	nop
loop:
	; ввод
	in r18,OCR0A
	in r19,OCR0B
	; изменения флагов
	add r18,r19
	brvs overflow_is_setted
	; флаг V очищен
	ldi r18,0
	rjmp output
overflow_is_setted:
	ldi r18,1
	; выравниваю, чтобы было одинаковое количество тактов
	; и при V==0 и при V==1
	nop
output:
	out OCR0A,r18
rjmp loop
```

Файл stimuli

```
$log OCR0A
$log OCR0B
$startlog BRVS_log_output.stim

OCR0A = 127
OCR0B = 1

#10
OCR0A =  2
OCR0B = 2

#10
$stoplog
$break
```

Выход

```
#1
OCR0B = 0x01
OCR0A = 0x7f
#7
OCR0A = 0x01
#3
OCR0B = 0x02
OCR0A = 0x02
#7
OCR0A = 0x00
```

Таблица результатов

	OCR0A	**OCR0B**	**OCR0A(res)**
1.	0x7F	0x01	0x01
2.	0x02	0x02	0x00

Вывод

При установленном флаге переполнения программа переходит на установленную в аргументе метку, счётчик команд увеличивается на k + 1.

Если этот флаг не установлен, то выполняется следующая команда, счётчик увеличивается на 1.

BRCC

Мнемоника BRCC
Операнды k
Описание Перейти если перенос очищен
Операция if (C == 0) PC = PC + k + 1
Флаги -
Циклы 1/2
Счётчик команд: PC = PC + k + 1, если флаг не выставлен, иначе PC = PC +1

Текст программы

```
setup:
	nop
loop:
	; ввод
	in r18,OCR0A
	in r19,OCR0B
	; изменения флагов
	add r18,r19
	brcc carry_is_cleared
	; флаг carry установлен
	ldi r18,1
	rjmp output
carry_is_cleared:
	ldi r18,0
	; выравниваю, чтобы было одинаковое количество тактов
	; и при C==0 и при C==1
	nop
output:
	out OCR0A,r18
rjmp loop
```

Файл stimuli

```
$log OCR0A
```

```
$log OCR0B
$startlog BRCC_log_output.stim

ACR0A = 127
ACR0B = 1

#10
ACR0A = 2
ACR0B = 2

#10
$stoplog
$break
```

Выход

```
#1
OCR0B = 0x01
OCR0A = 0x7f
#7
OCR0A = 0x01
#3
OCR0B = 0x02
OCR0A = 0x02
#7
OCR0A = 0x00
```

Таблица результатов

	OCR0A	**OCR0B**	**OCR0A(res)**
1.	0x7F	0x01	0x01
2.	0x02	0x02	0x00

Вывод

При очищенном флаге переноса программа переходит на установленную в аргументе метку, счётчик команд увеличивается на k + 1. Если же этот флаг установлен, то выполняется следующая команда, счётчик увеличивается на 1.

BRPL

Мнемоника BRPL
Операнды k
Описание Перейти если плюс
Операция if (N == 0) PC = PC + k + 1
Флаги -
Циклы 1/2
Счётчик команд: PC = PC + k + 1, если результат не отрицательный (флаг N очищен), иначе PC = PC +1 (результат отрицательный)

Текст программы

```
setup:
	nop
loop:
	; ввод
	in r18,OCR0A
	in r19,OCR0B
	; изменения флагов
	add r18,r19
	brpl negative_is_cleared
	; Флаг N установлен
	ldi r18,1
	rjmp output

negative_is_cleared:
	ldi r18,0
	nop

output:
	out OCR0A,r18
rjmp loop
```

Файл stimuli

```
$log OCR0A
$log OCR0B
$startlog BRPL_log_output.stim
```

```
OCR0A = 0
OCR0B = 0xFF

#10
OCR0A =  2
OCR0B = 2

#10
$stoplog
$break
```

Выход

```
#1
OCR0B = 0xff
#7
OCR0A = 0x01

#3
OCR0B = 0x02
OCR0A = 0x02
#7
OCR0A = 0x00
```

Таблица результатов

	OCR0A	**OCR0B**	**OCR0A(res)**
1.	0x00	0xFF	0x01
2.	0x02	0x02	0x00

Вывод

При очищенном флаге negative программа переходит на установленную в аргументе метку, счётчик команд увеличивается на k + 1. Если же флаг установлен (результат вычислений не отрицательный), то выполняется следующая команда, счётчик увеличивается на 1.

BRBS

Мнемоника BRBS
Операнды s,k
Описание Перейти если флаг в SREG установлен
Операция if (SREG(s) == 1) PC = PC + k + 1
Флаги -
Циклы 1/2
Счётчик команд: PC = PC + k + 1, если в регистре SREG выставлен бит s из аргумента, иначе PC = PC +1

Текст программы

```
setup:
	nop
loop:
	; ввод
	in r18,OCR0A
	in r19,OCR0B
	; изменения флагов
	add r18,r19
	brbs 1,flag_is_setted // проверка на флаг нуля
	; флаг нуля очищен
	ldi r18,0
	rjmp output

flag_is_setted:
	ldi r18,1
	nop

output:
	out OCR0A,r18

rjmp loop
```

Файл stimuli

```
$log OCR0A
$log OCR0B
$startlog BRBS_log_output.stim
```

OCR0A = 0
OCR0B = 0

#10
OCR0A = 2
OCR0B = 2

#10
$stoplog
$break

Выход

#8
OCR0A = 0x01

#3
OCR0B = 0x02
OCR0A = 0x02
#7
OCR0A = 0x00

Таблица результатов

	OCR0A	**OCR0B**	**OCR0A(res)**
1.	0x00	0x00	0x01
2.	0x02	0x02	0x00

Вывод

При установленном флаге, соответствующему номеру бита в аргумете команды BRBS, программа переходит на установленную в аргументе метку, счётчик команд увеличивается на k + 1. Если же этот флаг очщиен, то выполняется следующая команда, счётчик увеличивается на 1.

BRTC

Мнемоника BRTC
Операнды k

Описание Перейти если флаг T установлен
Операция if (T == 0) PC = PC + k + 1
Флаги -
Циклы 1/2
Счётчик команд: PC = PC + k + 1, если флаг не выставлен, иначе PC = PC +1

Текст программы

```
setup:
	nop
loop:
	; ввод
	in r18,OCR0A
	; изменения флагов,
	; записываем 0 бит регистра r18 в SREG на место флага T
	bst r18,0
	brtc flag_is_cleared
	; флаг установлен
	ldi r18,1
	rjmp output

flag_is_cleared:
	ldi r18,0
	nop

output:
	out OCR0A,r18

rjmp loop
```

Файл stimuli

```
$log OCR0A
$startlog BRTC_log_output.stim

OCR0A = 2

#9
OCR0A = 4
```

#9
$stoplog
$break

Выход

#1
OCR0A = 0x02
#6
OCR0A = 0x01

#3
OCR0A = 0x04
#6
OCR0A = 0x00

Таблица результатов

	OCR0A	**OCR0A(res)**
1.	0x02	0x01
2.	0x04	0x00

Вывод

При очищенном флаге T программа переходит на установленную в аргументе метку, счётчик команд увеличивается на k + 1. Если же этот флаг установлен, то выполняется следующая команда, счётчик увеличивается на 1.

Работа с массивами

Разбор дампов памяти Flash и ОЗУ из примера с копированием массива с инверсией

Дампы выглядят так:
Flash
:02001E008E6BE7
:08002000AE607F69D82B0215C8
:00000001FF
S

:0A0100007194519F809627D4FDEA08
:00000001FF

Получим значения из памяти:

F

:02 001E 00	8E 6B	E7
:08 0020 00	AE 60 7F 69 D8 2B 02 15	C8
:00 0000 01		FF

S

:0A 0100 00	71 94 51 9F 80 96 27 D4 FD EA	08
:00 0000 01		FF

Flash: 8E 6B AE 60 7F 69 D8 2B 02 15
Этим значениям соответствует начальное состояние массива

S: 71 94 51 9F 80 96 27 D4 FD EA
Каждое значение является побитовой инверсией соответствующего элемента исходного массива

0x8E = 0b10001110
0x71 = 0b01110001

0x6B = 0b01101011
0x94 = 0b10010100

0xAE = 0b10101110
0x51 = 0b01010001

0x60 = 0b01100000
0x9F = 0b10011111

0x7F = 0b01111111
0x80 = 0b10000000

0x69 = 0b01101001
0x96 = 0b10010110

0xD8 = 0b11011000
0x27 = 0b00100111

0x2B = 0b00101011
0xD4 = 0b11010100

0x02 = 0b00000010
0xFD = 0b11111101

0x15 = 0b00010101
0xEA = 0b11101010

Сортировка пузырьком

Программа написана для знаковых слов

Текст программы

```
.set ARR_SIZE = 10

.def TEMP = r16
.def COUNT_NEGATIVE_NUMS = r17
.def NUM_ITEMS = r18
.def HAS_SWAP = r19
.def LEFT_WORD_L = r20
.def LEFT_WORD_H = r21
.def RIGHT_WORD_L = r22
.def RIGHT_WORD_H = r23

.dseg
arr: .BYTE ARR_SIZE

.cseg

reset:
        rjmp main
main:
        ldi ZH,High(src * 2)
```

```
        ldi ZL,Low(src * 2)
        ldi YH,High(arr)
        ldi YL,Low(arr)
        ldi NUM_ITEMS,ARR_SIZE
arr_copy:
        lpm r0,Z+
        lpm r1,Z+
        st Y+,r0
        st Y+,r1
        subi NUM_ITEMS,1
        brne arr_copy


loop:
        ldi HAS_SWAP,0
        ldi NUM_ITEMS,ARR_SIZE - 1 ; нам не надо доходить ровно до
последнего элемента
        cpi NUM_ITEMS,1 ; выходим, если количество смежных пар < 1
        brlo exit

        ; загрузка начальных значений
        ldi YH,High(arr)
        ldi YL,Low(arr)
        ; загрузка следующего элемента в массиве
        ldi ZH,High(arr + 2)
        ldi ZL,Low(arr + 2)


        load_words:
                ldi COUNT_NEGATIVE_NUMS,0
                ; загрузка данных
                ld LEFT_WORD_L,Y+
                ld LEFT_WORD_H,Y+
                ld RIGHT_WORD_L,Z+
                ld RIGHT_WORD_H,Z+

        ; считаем количество отрицательных чисел
                sbrc LEFT_WORD_H,7
                inc COUNT_NEGATIVE_NUMS
```

```
            sbrc RIGHT_WORD_H,7
            inc COUNT_NEGATIVE_NUMS

            cpi COUNT_NEGATIVE_NUMS,1
            ; переход, если оба >=0 или оба < 0
            brne common_comparison

        ; сравнение при одном числе < 0
            cp RIGHT_WORD_H,LEFT_WORD_H ; сравниваем со знаком
            brlt swap_words
            rjmp continue


        common_comparison:
        ; обычное сравнение (оба >= 0 или оба < 0)
            cp RIGHT_WORD_H,LEFT_WORD_H ; сравниваем со знаком
            brlt swap_words
            brne continue
            cp RIGHT_WORD_L,LEFT_WORD_L ; сравниваем без знака (так как в младшем байте знак числа не учитывается)
            brlo swap_words

        continue:
            ; делаем, пока всех не пройдём все смежные пары
            dec NUM_ITEMS
            brne load_words

            ;  после прохода по всему массиву проверяем, были ли перестановки
            cpi HAS_SWAP,1
            breq loop
            ; если нет, то завершаем
            brne exit

        ; перестановки слов
        swap_words:
            ldi HAS_SWAP,1
            ; перемещаем указатель на текущие элементы
            sbiw Y,2
```

```
            sbiw Z,2
            ; загружаем в ОЗУ, меняя местами
            st Z+,LEFT_WORD_L
            st Z+,LEFT_WORD_H
            st Y+,RIGHT_WORD_L
            st Y+,RIGHT_WORD_H
            rjmp continue

rjmp loop

exit:
        nop
        rjmp exit

.cseg
src: .dw -16384, -12, -1, 10213, -7, 14, -4097, 0, 6544, 3
```

Файл стимуляции

```
$startlog BubbleSort_output.stim


#1718

$stoplog

$memdump BubbleSort_oputput_array_dump.stim 0x0100 20 s

$break
```

Выход дампа памяти

```
:1001000000C0FFEFF4FFF9FFFFFF000003000E0047
:040110009019E52736
:00000001FF
```

Получим значения из него:

```
:10 0100 00        00C0 FFEF F4FF F9FF FFFF 0000 0300 0E00      47
:04 0110 00        9019 E527                                    36
:00 0000 01                                                     FF
```

Значения:

```
00C0 FFEF F4FF F9FF FFFF 0000 0300 0E00
9019 E527
```

Меняем байты:
C000 EFFF FFF4 FFF9 FFFF 0000 0003 000E
1990 27E5

Получаем значения в 10 системе счисления:
0xC000 = -16384
0xEFFF = -4097
0xFFF4 = -12
0xFFF9 = -7
0xFFFF = -1
0x0000 = 0
0x0003 = 3
0x000E = 14
0x1990 = 6544
0x27E5 = 10213

Выводы

В результате выполнения данной лабораторной работы я научился грамотно отлаживать файлы, отслеживать состояние программы, работать с флагами и условными переходами, работать с массивом, в том числе изменять его состояние напрямую через память и применять полученные знания на практике