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

Факультет информатики и
вычислительной техники

Кафедра ИиСП

Отчет
по лабораторной работе № 3

по дисциплине «Машинно-зависимые языки программирования»
Вариант 7

Выполнил: студент группы ПС-11
Щеглов Г.С
Проверил: Баев А.А.

г. Йошкар-Ола
2024

Цель работы: научиться тестировать ассемблерные команды и писать простой алгоритм.

Задания на лабораторную работу:

1. Выполнить проверку различных операций на флаги регистра состояний.
2. Написать программу с использованием условного перехода
3. Написать программу для работы с массивом

1. **Теоретические сведения**

https://proglib.io/p/7-sposobov-sortirovki-massivov-na-primere-s-s-illyustraciyami-2022-04-20

https://www.radiokot.ru/start/mcu_fpga/avr/14/

http://easyelectronics.ru/skazhu-paru-slov-o-optimizacii-koda.html

https://easyelectronics.ru/avr-uchebnyj-kurs-makroassembler.html

https://trolsoft.ru/ru/avr-assembler

2. Практическая часть

Задача 1.

Выполнение для команды SBCI

Мнемоника SBCI

Операнды Rd,K8

Описание Вычитание константы с учётом переноса

Операция Rd = Rd – K8 - C

Флаги Z,C,N,V,H,S

Циклы 1

Текст программы

```
.equ k8 = 0x08
reset:
     rjmp main

main:
     ldi r18, 0xFF
     out  PORTB, r18
loop:
     in r18, PORTB
     sbci r18, k8
     out  PORTB, r18

     rjmp loop
```

Тестовый файл

```
$log PORTB
$log SREG
$startlog Asm_Lab_Math_log_output.stim
#3
PORTB = 255
#5
```

```
PORTB = 247
#5
PORTB = 135
#5
PORTB = 7
#5
$stoplog
$break
```

Выходной файл

#2

PORTB = 0xff

#2

SREG = 0x14

#1

PORTB = 0xf7

#4

SREG = 0x34

#1

PORTB = 0xef

#3

PORTB = 0x87

#1

SREG = 0x38

#1

PORTB = 0x7f

#3

PORTB = 0x07

#1

SREG = 0x35

#1

PORTB = 0xff

Выводы:

Флаг Z всегда равен нулю, остальные флаги представлены в таблице

PORTB	SREG	PORTB’	Флаги
0xFF	0x14	0xF7	S, N
0xF7	0x34	0xEF	H, S, N
0x87	0x38	0x7F	H, S, V
0x07	0x35	0xFF	H, S, N, C

Выполнение для команды ASR

Мнемоника ASR

Операнды Rd

Описание Арифметический сдвиг вправо

Операция Rd(n)=Rd(n+1), n=0,...,6

Флаги Z,C,N,V,S

Циклы 1

Текст программы

```
reset:
    rjmp main
main:
    ldi r18, 0xFF
    out  PORTB, r18
loop:
    in r18, PORTB
    asr r18
    out  PORTB, r18

    rjmp loop
```

Тестовый файл

```
$log PORTB
$log SREG
$startlog Asm_Lab_Math_log_output.stim
#3
PORTB = 255
#5
PORTB = 1
#5
$stoplog
$break
```

Выходной файл

```
#2
PORTB = 0xff
#2
SREG = 0x15
#4
PORTB = 0x01
#1
SREG = 0x1b
#1
PORTB = 0x00
```

Выводы

PORTB	SREG	PORTB’	Флаги
0xFF	0x15	0x01	S, N, C
0x01	0x1B	0x00	S, V, Z, C

Выполнение для команды SBIW

Мнемоника SBIW

Операнды Rdl, K6

Описание Вычитание константы из слова

Операция	Rdh:Rdl = Rdh:Rdl -K6
Флаги	Z,C,N,V,S
Циклы	1

Текст программы

```
.equ k6 = 0x01
reset:
     rjmp main
main:
     ldi r25, 0x00
     out  PORTB, r25
     ldi r24, 0x00
     out  PORTC, r24
loop:
     in r25, PORTB
     in r24, PORTC
     sbiw r25:r24, k6
     out  PORTB, r25
     out PORTC, r24

     rjmp loop
```

Тестовый файл

$log PORTB

$log PORTC

$log SREG

$startlog Asm_Lab_Math_log_output.stim

#5

PORTB = 0

PORTC = 1

#8

PORTB = 0

PORTC = 0

#8

$stoplog

$break

Выходной файл

#5

PORTC = 0x01

#2

SREG = 0x02

#3

PORTC = 0x00

#5

SREG = 0x15

#2

PORTB = 0xff

#1

PORTC = 0x7f

Выводы

Флаг V всегда равен 0, остальные флаги представлены в таблице

PORTB	PORTC	SREG	PORTB’	PORTC’	Флаги
0x00	0x01	0x02	0x00	0x01	Z
0x00	0x00	0x15	0xFF	0x7F	S, N, C

Выполнение для команды SUB

Мнемоника SUB

Операнды Rd, Rr

Описание Вычитание без учёта переноса

Операция Rd = Rd - Rr

Флаги Z,C,N,V,H,S

Циклы 1

Текст программы

```
reset:
     rjmp main
main:
     ldi r25, 0x00
     out  PORTB, r24
     ldi r24, 0x00
     out  PORTC, r24
loop:
     in r25, PORTB
     in r24, PORTC
     sub r25, r24
     out  PORTB, r25
     out PORTC, r24

      rjmp loop
```

Тестовый файл

```
$log PORTB
$log PORTC
$log SREG
$startlog Asm_Lab_Math_log_output.stim
#5
PORTB = 0
PORTC = 0
#7
PORTB = 255
```

```
PORTC = 127
#7
PORTB = 128
PORTC = 127
#7
PORTB = 1
PORTC = 127
#7
$stoplog
$break
```

Выходной файл

```
#2
PORTB = 0xff
#2
PORTC = 0x7f
#1
PORTB = 0x00
PORTC = 0x00
#2
SREG = 0x02
#5
PORTB = 0xff
PORTC = 0x7f
#2
SREG = 0x14
#1
PORTB = 0x80
#6
SREG = 0x38
#1
PORTB = 0x01
#6
SREG = 0x35
#1
PORTB = 0x82
```

Выводы

PORTB	PORTC	SREG	PORTB’	PORTC’	Флаги
0x00	0x00	0x02	0x00	0x00	Z
0xFF	0x7F	0x14	0x81	0x7F	S, N
0x80	0x7F	0x38	0xFE	0x7F	V
0x01	0x7F	0x35	0x82	0x7F	H, S, N, C

Выполнение для команды COM

Мнемоника COM

Операнды Rd

Описание Дополнение до единицы

Операция Rd = $FF - Rd

Флаги Z,C,N,V,S

Циклы 1

Текст программы

```
reset:
        rjmp main
main:
        ldi r18, 0xFF
        out PORTB, r18
loop:
        in r18, PORTB
        com r18
        out    PORTB, r18

        rjmp loop
```

Тестовый файл

```
$log PORTB
$log SREG
$startlog Asm_Lab_Math_log_output.stim
#3
PORTB = 255
#5
PORTB = 0
#5
$stoplog
$break
```

Выходной файл

```
#2
PORTB = 0xff
#2
SREG = 0x03
#1
PORTB = 0x00
#4
SREG = 0x15
#1
PORTB = 0xff
```

Выводы

Флаг V всегда равен 0, остальные флаги представлены в таблице

PORTB	SREG	PORTB’	Флаги
0xFF	0x03	0x00	Z, C
0x00	0x15	0xFF	S, N, C

Задача 2.

Оператор BREQ

Мнемоника BREQ

Операнды k

Описание: Условный относительный переход. Проверяется флаг нулевого значения (Z) регистра статуса SREG и, если бит установлен, выполняется переход относительно значения счётчика команд PC. Если инструкция выполняется непосредственно после выполнения любой из команд CP, CPI, SUB или SUBI переход произойдет тогда, и только тогда, двоичное число (со знаком или без знака), представленное в Rd, равно двоичному числу (со знаком или без знака), представленному в Rr. Данная инструкция выполняет переход в любом направлении относительно счётчика команд (PC - 63 ≤ назначение ≤ PC + 64). Параметр k является смещением относительно значения счётчика PC и представлен в форме дополнения до двух..

Операция if(Z==1) PC = PC + k + 1

Флаги None

Циклы 1/2

Счетчик программ: PC ← PC + k +1 ; PC ← PC +1 если условие не выполнено

Текст программы

```
reset:
        rjmp main
main:
        ldi r24, 0x00
        ldi r25, 0x00
loop:
        in r24, PORTB
        in r25, PORTC
        cp r24, r25
        breq check
        rjmp loop
check:
        out PORTD, r24
        rjmp loop
```

Тестовый файл

```
$log PORTD
$log SREG
$startlog Asm_Lab_Math_log_output.stim
#3
PORTB = 35
PORTC = 200
#6
PORTB = 105
PORTC = 220
#6
PORTB = 5
PORTC = 200
#6
PORTB = 45
```

PORTC = 45

#8

$stoplog

$break

Выходной файл

#5

SREG = 0x35

#6

SREG = 0x20

#6

SREG = 0x35

#6

SREG = 0x02

#3

PORTD = 0x2d

Выводы

	PORTB	PORTC	PORTD
1.	0x23	0xC8	-
2.	0x69	0xDC	-
3.	0x05	0xC8	-
4.	0x2D	0x2D	0x2D

Оператор BRNE

Мнемоника BRNE

Операнды k

Описание: Условный относительный переход. Проверяется флаг нулевого значения (Z) регистра статуса и, если бит очищен, выполняется переход относительно значения счётчика команд PC. Если инструкция выполняется непосредственно после выполнения любой из команд CP, CPI, SUB или SUBI переход произойдет тогда, и только тогда, двоичное число (со знаком или без знака), представленное в Rd, не равно двоичному числу (со знаком или без знака), представленному в Rr. Данная инструкция выполняет переход в любом направлении относительно счётчика команд (PC - 63 ≤ назначение ≤ PC + 64). Параметр k является смещением относительно значения счётчика PC и представлен в форме дополнения до двух..

Операция if(Z==0) PC = PC + k + 1

Флаги None

Циклы 1/2

Счетчик программ: PC <-- PC + k + 1; PC <-- PC + 1, если условие не выполнено

Текст программы

```
reset:
        rjmp main
main:
        ldi r24, 0x00
        ldi r25, 0x00
loop:
        in r24, PORTB
        in r25, PORTC
        cp r24, r25
        brne check
        rjmp loop
```

```
check:
        out PORTD, r24
        rjmp loop
```

Тестовый файл

```
$log PORTD
$log SREG
$startlog Asm_Lab_Math_log_output.stim
#3
PORTB = 35
PORTC = 200
#6
PORTB = 105
PORTC = 220
#6
PORTB = 5
PORTC = 200
#6
PORTB = 45
PORTC = 45
#8
$stoplog
$break
```

Выходной файл

```
#5
SREG = 0x35
#3
```

PORTD = 0x23

#5

SREG = 0x20

#3

PORTD = 0x69

#5

SREG = 0x35

#3

PORTD = 0x05

Выводы

	PORTB	PORTC	PORTD
1.	0x23	0xC8	0x23
2.	0x69	0xDC	0x69
3.	0x05	0xC8	0x05
4.	0x2D	0x2D	-

Оператор CPI

Мнемоника CPI

Операнды Rd, K8

Описание: Команда выполняет сравнение содержимого регистра Rd с константой. Работает со "старшими" регистрами. Содержимое регистра не

изменяется. После этой команды можно выполнять любые условные переходы.

Операция Rd - K

Флаги None

Циклы 1

Счетчик программ: PC <-- PC + 1

Текст программы

```
reset:
        rjmp main
main:
        ldi r24, 0x2D
loop:
        in r24, PORTB
        cpi r24, 0x2D
        brlo less
        rjmp loop
less:
        out PORTD, r24
        rjmp loop
```

Тестовый файл

```
$log PORTD
$log SREG
$startlog Asm_Lab_Math_log_output.stim
#2
PORTB = 32
#7
PORTB = 45
#7
PORTB = 64
```

#7

PORTB = 0x26

#7

$stoplog

$break

Выходной файл

#3

SREG = 0x35

#3

PORTD = 0x20

#4

SREG = 0x02

#10

SREG = 0x20

#5

SREG = 0x35

#3

PORTD = 0x1a

Выводы

	PORTB	PORTD
1.	0x20	0x20
2.	0x2D	-
3.	0x40	-
4.	0x1A	0x1A

Оператор BRVC

Мнемоника BRVC

Операнды k

Описание: Условный относительный переход. Проверяется флаг переполнения (V) регистра статуса SREG и, если бит очищен, выполняется переход относительно значения счётчика команд PC. Данная инструкция выполняет переход в любом направлении относительно счётчика команд (PC - 63 ≤ назначение ≤ PC + 64). Параметр k является смещением относительно значения счётчика PC и представлен в форме дополнения до двух.

Операция if(V==0) PC = PC + k + 1

Флаги None

Циклы 1/2

Счетчик программ: PC <-- PC + k + 1 ; PC <-- PC + 1, если условие не выполнено.

Текст программы

```
reset:
        rjmp main
main:
        ldi r24, 0x2D
        ldi r25, 0x2D
loop:
        in r24, PORTB
        in r25, PORTC
        cp r24, r25
        brvc same
        rjmp loop
same:
        out PORTD, r24
        rjmp loop
```

Тестовый файл

```
$log PORTD
$log SREG
$startlog Asm_Lab_Math_log_output.stim
#3
PORTB = 65
PORTC = 40
#6
PORTB = 145
PORTC = 250
#6
PORTB = 0
PORTC = 200
#6
PORTB = 45
PORTC = 45
#8
$stoplog
$break
```

Выходной файл

```
#5
SREG = 0x20
#3
PORTD = 0x41
#5
SREG = 0x38
```

#6

SREG = 0x35

#3

PORTD = 0x00

#5

SREG = 0x02

Выводы

	PORTB	PORTC	PORTD
1.	0x41	0x28	0x41
2.	0x91	0xFA	-
3.	0x00	0xC8	0x00
4.	0x2D	0x2D	-

Оператор BRLO

Мнемоника BRLO

Операнды k

Описание: Условный относительный переход. Проверяется флаг переноса (C) регистра статуса SREG и, если бит установлен, выполняется переход относительно значения счётчика команд PC. Если инструкция выполняется непосредственно после выполнения любой из команд CP, CPI, SUB или SUBI переход произойдет тогда, и только тогда, двоичное число без

знака, представленное в Rd, меньше двоичного числа без знака, представленного в Rr. Данная инструкция выполняет переход в любом направлении относительно счётчика команд (PC - 63 ≤ назначение ≤ PC + 64). Параметр k является смещением относительно значения счётчика PC и представлен в форме дополнения до двух.

Операция if(C==1) PC = PC + k + 1

Флаги None

Циклы 1/2

Счетчик программ: PC <-- PC + k + 1 ; PC <-- PC + 1, если условие не выполнено.

Текст программы

```
reset:
        rjmp main
main:
        ldi r24, 0x2D
        ldi r25, 0x2D
loop:
        in r24, PORTB
        in r25, PORTC
        cp r24, r25
        brlo same
        rjmp loop
same:
        out PORTD, r24
        rjmp loop
```

Тестовый файл

```
$log PORTD
$log SREG
```

$startlog Asm_Lab_Math_log_output.stim

#3

PORTB = 65

PORTC = 40

#6

PORTB = 145

PORTC = 250

#6

PORTB = 0

PORTC = 200

#6

PORTB = 45

PORTC = 45

#8

$stoplog

$break

Выходной файл

#5

SREG = 0x20

#6

SREG = 0x38

#6

SREG = 0x35

#8

SREG = 0x02

Выводы

	PORTB	PORTC	PORTD
1.	0x41	0x28	-
2.	0x91	0xFA	-
3.	0x0A	0x14	0x0A
4.	0x2D	0x2D	-

Задача 3

Преобразование массива (инверсия порядка бит)

1) Для типа байт

```
.set ARR_SIZE=10
.dseg
arr:    .BYTE ARR_SIZE
.cseg
reset:
    rjmp main
main:
    ldi ZH, High(src*2)  ; Загружаем адрес исходного массива
    ldi ZL, Low(src*2)

    ldi YH, High(arr)    ; Загружаем адрес полученного массива
    ldi YL, Low(arr)

    ldi R18, ARR_SIZE
arr_copy:
    lpm R0, Z+           ; Загружаем байт, переставляем указатель Z
    out OCR0A, R0
    clr R1
    ldi R21, 8           ; Количество бит не отвечащих за знак
reverse_bits:
    lsr R0               ; Сдвигаем вправо, младший попадает в флаг C
    rol R1               ; Сдвигаем влево, загружаем флаг C в младший
бит
    dec R21
    brne reverse_bits    ; Пока не равен 0
    out OCR0B, R1
    st  Y+, R1           ; Сохраняем, переставляем указатель Y
    dec R18
    brne arr_copy
loop:
    nop
    rjmp loop
```

```
.cseg
src: .db 0x8E, 0x6B, 0xAE, 0x60, 0x7F, 0x69, 0xD8, 0x2B, 0x02, 0x15
```

Исходный массив: 8e 6b ae 60 7f 69 d8 2b 02 15
Полученный массив: 71 d6 75 06 fe 96 1b d4 40 a8

Дамп памяти
Asm_Lab_Array_mem_F_log_output
:02001E00199235
:080020002A95A1F70000FECFB4
:00000001FF

Asm_Lab_Array_mem_S_log_output
:0A01000071D67506FE961BD440A8C8
:00000001FF

hex: 71 D6 75 06 FE 96 1B D4 40 A8
dec: 113 214 117 6 254 150 27 212 64 168

2) Для знакового байта

```
.set ARR_SIZE=10
.dseg
arr:    .BYTE ARR_SIZE
.cseg
reset:
    rjmp main
main:
    ldi ZH, High(src*2)  ; Загружаем исходный массив в Z
    ldi ZL, Low(src*2)

    ldi YH, High(arr)    ; Загружаем полученный массив в Y
    ldi YL, Low(arr)

    ldi R18, ARR_SIZE
arr_copy:
    lpm R0, Z+           ; Загружаем байт, переставляем указатель Z
    out OCR0A, R0
    mov R16, R0          ; Копируем значение
    andi R16, 0x80       ; Побитово умножаем
    clr R1               ; Очищаем
    ldi R21, 7           ; Количество бит не отвечащих за знак
reverse_bits:
    lsr R0               ; Сдвигаем вправо, младший попадает в флаг C
```

```
    rol R1              ; Сдвигаем влево, загружаем флаг C в младший
бит
    dec R21
    brne reverse_bits   ; Пока не равен 0
    or R1, R16          ; Восстанавливаем знаковый бит
    out OCR0B, R1
    st Y+, R1           ; Сохраняем, переставляем указатель Y
    dec R18
    brne arr_copy
loop:
    nop
    rjmp loop
.cseg
src: .db 0x8E, 0x6B, 0xAE, 0x60, 0x7F, 0x69, 0xD8, 0x2B, 0x02, 0x15
```

Исходный массив: 8e 6b ae 60 7f 69 d8 2b 02 15
Полученный массив: b8 6b ba 03 7f 4b 8d 6a 20 54

Дамп памяти
Asm_Lab_Array_mem_F_log_output
:02001E00E1F708
:08002000102A18BC19922A9560
:00000001FF

Asm_Lab_Array_mem_S_log_output
:0A010000B86BBA037F4B8D6A2054E0
:00000001FF

hex: B8 6B BA 03 7F 4B 8D 6A 20 54
dec: 184 107 186 3 127 75 141 106 32 84

Для знаковых двойных слов (32 бита):

```
.set ARR_SIZE = 5
.dseg
arr:    .BYTE ARR_SIZE * 4
.cseg
reset:
    rjmp main
main:
    ldi ZH, High(src * 2)  ; Загружаем исходный массив в Z
    ldi ZL, Low(src * 2)

    ldi YH, High(arr)      ; Загружаем полученный массив в Y
    ldi YL, Low(arr)
```

```
    ldi r25, ARR_SIZE

process:
    lpm r0, Z+              ; Загружаем байты начиная с младшего,
переставляем указатель Z
    lpm r1, Z+
    lpm r2, Z+
    lpm r3, Z+

    mov r22, r3
    andi r22, 0x80

    rcall reverse_signed_byte     ; Инверсия порядка бит r0
    mov r16, r0             ; Сохраняем результат в r16
    mov r0, r1
    rcall reverse_byte7
    mov r17, r0
    mov r0, r2
    rcall reverse_byte7
    mov r18, r0
    mov r0, r3
    rcall reverse_byte7
    mov r19, r0
    st Y+, r16              ; Сохраняем, переставляем указатель Y
    st Y+, r17
    st Y+, r18
    st Y+, r19

    dec r25
    brne process    ; Переход к следующему числу
loop:
    nop
    rjmp loop
reverse_byte7:
    clr r21
    ldi r20, 7
    lsr r24            ; Ставим оставшийся бит
    rol r21
reverse_loop7:
    lsr r0                  ; Сдвигаем вправо, младший попадает в флаг
C
    rol r21                 ; Сдвигаем влево, загружаем флаг C в
младший
    dec r20
    brne reverse_loop7
    mov r23, r0
    lsr r23          ; Сохраняем оставшийся бит
    rol r24
    mov r0, r21             ; Возвращаем результат
    ret
```

```
reverse_signed_byte:
    clr r21
    ldi r20, 7
reverse_signed_loop:
    lsr r0
    rol r21
    dec r20
    brne reverse_signed_loop
    or r21, r22
    mov r23, r0
    lsr r23
    rol r24
    mov r0, r21
    ret
.cseg
src: .dd 0x8E6BAE60, 0x7F69D82B, 0x0215FFFF, 0x12345678, 0x0000FFFF
```

Исходный массив: 0x8E6BAE60, 0x7F69D82B, 0x0215FFFF, 0x12345678, 0x0000FFFF

0x8E6BAE60: 10001110 01101011 10101110 01100000
0x7F69D82B: 01111111 01101001 11011000 00101011
0x0215FFFF: 00000010 00010101 11111111 11111111
0x12345678: 00010010 00110100 01010110 01111000
0x0000FFFF: 00000000 00000000 11111111 11111111

Полученный массив: 0x833AEB38, 0x6A0DCB7f, 0x7FFFD420, 0x0F351624, 0x7FFF8000

0x833AEB38: 10000011 00111010 11101011 00111000
0x6A0DCB7F: 01101010 00001101 11001011 01111111
0x7FFFD420: 01111111 11111111 11010100 00100000
0x0F351624: 00001111 00110101 00010110 00100100
0x7FFF8000: 01111111 11111111 10000000 00000000

Дамп памяти
Asm_Lab_Array_mem_F_log_output
:02001E000FD001
:08002000102D022C0CD0202D44
:00000001FF

Asm_Lab_Array_mem_S_log_output
:0A010000833AEB386A0DCB7F7FFFD6
:00000001FF

Выводы: мы научились тестировать ассемблерные команды и написали простой алгоритм инверсии порядка бит.