МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ»

Кафедра ИиСП

Отчет

по лабораторной работе № 3

по дисциплине «Машинно-зависимые языки программирования»

Вариант 2

Выполнил: ст. гр. ПС-14

Сайфутдинов

Проверил: доцент, доцент кафедры

ИиСП Баев А.А.

г. Йошкар-Ола

2024

Цель работы: изучить математические и логические операции; ознакомиться с массивами и работой с ними на ассемблере.

Задачи:

1. Согласно своему варианту разобрать 5 математических и логических операций:
    a. написать программу с использованием данной операции;
    b. проверить в режиме отладки изменение флагов при различных значениях операндов;
2. Согласно своему варианту разобрать 5 команд:
    a. написать программу с использованием данной команды;
    b. проверить в режиме отладки изменение работу программы при различных значениях операндов; отследить переходы;
3. Согласно своему варианту выполнить работу с массивами:
    a. понять принцип задачи ;
    b. написать программу.

Мнемоника: COM

- **Операнды:** Rd
- **Описание:** Побитная инверсия.
- **Операция:** Rd = $FF - Rd
- **Флаги:** Z, C, N, V, S
- **Циклы:** 1

Код программы:

```
reset:
rjmp main

main:
ldi r16, 0xff
out OCR0A, r16
bset 3

loop:
com r16
out OCR0A, r16

 rjmp loop
```

Тестовый файл:

```
$log SREG
$log OCR0A
$startlog
COM_log_output.stim #8
$stoplog
$break
```

Выходной файл:

```
#1
OCR0A = 0xff
#1
SREG = 0x08
#1
SREG = 0x03
#1
OCR0A = 0x00
#3
SREG = 0x15
```

#1
OCR0A = 0xff

OCR0A	SREG	OCR0A'	Флаги
0x55	0x14	0xAA	N=1, S=1
0xAA	0x14	0x55	N=1, S=1
0x00	0x02	0xFF	Z=1

Мнемоника: LSL

- **Операнды: Rd**
- **Описание: Логический сдвиг влево.**
- **Операция: Rd(n+1) = Rd(n), Rd(0) = 0, C = Rd(7)**
- **Флаги: Z, C, N, V, H, S**
- **Циклы: 1**

Код программы:

```
reset:
    rjmp main

main:
    ldi r16, 0x7f
    out OCR0A, r16

loop:
    lsl r16
    out OCR0A, r16

    rjmp loop
```

Тестовый файл:

```
$log SREG

$log OCR0A

$startlog LSL_log_output.stim

#31

$stoplog

$break
```

Выходной файл:

#1
OCR0A = 0x7f
#1
SREG = 0x2c
#1
OCR0A = 0xfe
#3
SREG = 0x35
#1
OCR0A = 0xfc
#4
OCR0A = 0xf8
#4
OCR0A = 0xf0
#3
SREG = 0x15
#1
OCR0A = 0xe0
#4
OCR0A = 0xc0
#4
OCR0A = 0x80
#3
SREG = 0x1b
#1
OCR0A = 0x00

OCR0A	SREG	OCR0A'	Флаги
0x55	0x00	0xAA	-
0xAA	0x00	0x54	-
0x00	0x02	0x00	Z=1

Мнемоника: DEC

- **Операнды: Rd**
- **Описание: Декрементировать значение регистра.**
- **Операция: Rd = Rd - 1**
- **Флаги: Z, N, V, S**
- **Циклы: 1**

Код программы:

```
reset:
    rjmp main

main:
    ldi r16, 0x01
    out OCR0A, r16

loop:
    dec r16
    out OCR0A, r16
    dec r16
    out OCR0A, r16
    ldi r16, 0x80

    rjmp loop
```

Тестовый файл:

```
$log SREG
$log OCR0A
$startlog Dec_log_output.stim
#10
$stoplog
$break
```

Выходной файл:

#1
OCR0A = 0x01
#1
SREG = 0x02
#1
OCR0A = 0x00
#1
SREG = 0x14
#1
OCR0A = 0xff
#4
SREG = 0x18
#1
OCR0A = 0x7f

OCR0A	SREG	OCR0A'	Флаги
0x01	0x02	0x00	Z=1
0x00	0x02	0xFF	Z=1
0xFF	0x14	0xFE	N=1, S=1

Мнемоника: LSR

- **Операнды: Rd**
- **Описание: Логический сдвиг вправо.**
- **Операция: Rd(n) = Rd(n+1), Rd(7) = 0, C = Rd(0)**
- **Флаги: Z, C, N, V, S**
- **Циклы: 1**

Код программы:

```
reset:
    rjmp main

main:
    ldi r16, 0xff
    out OCR0A, r16

loop:
    lsr r16
    out OCR0A, r16

    rjmp loop
```

Тестовый файл:

```
$log SREG
$log OCR0A
$startlog Lsr_log_output.stim
#32
$stoplog
$break
```

Выходной файл:

```
#1
OCR0A = 0xff
#1
SREG = 0x04
#1
SREG = 0x19
#1
OCR0A = 0x7f
#4
OCR0A = 0x3f
#4
OCR0A = 0x1f
#4
OCR0A = 0x0f
```

#4
OCR0A = 0x07
#4
OCR0A = 0x03
#4
OCR0A = 0x01
#3
SREG = 0x1b
#1
OCR0A = 0x00

OCR0A	SREG	OCR0A'	Флаги
0xAA	0x00	0x55	-
0x55	0x00	0x2A	-
0x00	0x02	0x00	Z=1

Мнемоника: ORI

- **Операнды: Rd, K8**
- **Описание: Логическое ИЛИ с константой.**
- **Операция: Rd = Rd V K8**
- **Флаги: Z, N, V, S**
- **Циклы: 1**

Код программы:

```
reset:
    rjmp main

main:
    ldi r16, 0xff
    out OCR0A, r16
    nop

loop:
    ori r16, 0x01
    out OCR0A, r16
    ldi r16, 0
    out OCR0A, r16
    ori r16, 0
```

```
out OCR0A, r16

rjmp loop
```

Тестовый файл:

```
$log SREG
$log OCR0A
$startlog
Ori_log_output.stim
#8
$stoplog
$break
```

Выходной файл:

#1
OCR0A = 0xff
#1
SREG = 0x08
#1
SREG = 0x14
#3
OCR0A = 0x00
#1
SREG = 0x02

OCR0A	SREG	OCR0A	Флаги
0x55	0x00	0x5F	-
0x5F	0x00	0x5F	-
0x00	0x02	0x0F	Z=1

Мнемоника: BRLT

- **Операнды:** k
- **Описание:** Перейти если меньше (со знаком).
- **Операция:** if (S == 1) PC = PC + k + 1
- **Флаги:** None
- **Циклы:** ½

Код программы:

```
reset:
    rjmp main
main:
    ; Загрузка значений в регистры     ldi r18,
0x80  ; Rd = -128 (знаковое число)     ldi r19,
0x01  ; Rr = 1 (знаковое число)

    ; Вывод данных из РОН в IO регистр для отображения
out OCR0A, r18     out OCR0B, r19     nop  loop:
    ; Ввод данных из IO регистров в РОН для обработки
in r18, OCR0A     in r19, OCR0B

    ; Выполнение операции CP (сравнение)
cp r18, r19

    ; Выполнение операции BRLT
brlt less_than

    ; Если не меньше, загружаем 0 в r20
ldi r20, 0x00     rjmp end  less_than:
    ; Если меньше, загружаем 1 в r20
ldi r20, 0x01
 end:
    ; Вывод данных из РОН в IO регистр для отображения
out OCR0C, r20          rjmp loop
```

Код теста:

$log OCR0A $log

OCR0B $log

OCR0C $startlog

BRLT_log_output.

stim

#6

OCR0A = 0x80

OCR0B = 0x01

#8

OCR0A = 0x01

OCR0B = 0x80

#8

OCR0A = 0x7F

OCR0B = 0x7F

$stoplog

$break

Выходной файл:

#3
OCR0A = 0x80 #1
OCR0B = 0x01
#2
OCR0C = 0x01 ; BRLT выполнен, так как -128 < 1
#1
OCR0A = 0x01
OCR0B = 0x80
#2
OCR0C = 0x00 ; BRLT не выполнен, так как 1 > -128
#1
OCR0A = 0x7F OCR0B
= 0x7F
#2
OCR0C = 0x00 ; BRLT не выполнен, так как 127 == 127

OCR0A	**OCR0B**	**OCR0C**	**Результат**
0x80	0x01	0x01	BRLT выполнен
0x01	0x80	0x00	BRLT не выполнен
0x7F	0x7F	0x00	BRLT не выполнен

Мнемоника: SBRS

- **Операнды:** Rr, b
- **Описание:** Пропустить если бит в регистре установлен.
- **Операция:** if (Rr(b) == 1) PC = PC + 2 or 3
- **Флаги:** None • **Циклы:** 1/2/3

Код программы:

```
reset:
    rjmp main
main:
    ; Загрузка значений в регистры
ldi r18, 0x10  ; Установлен 4-й бит
ldi r19, 0x00  ; Бит не установлен

    ; Вывод данных из РОН в IO регистр для отображения
out OCR0A, r18     out OCR0B, r19     nop  loop:
    ; Ввод данных из IO регистров в РОН для обработки
in r18, OCR0A     in r19, OCR0B
```

```
    ; Выполнение операции SBRS
sbrs r18, 4
    ldi r20, 0x00  ; Если бит не установлен, загружаем 0
sbrs r18, 4     rjmp end
    ldi r20, 0x01  ; Если бит установлен, загружаем 1
 end:
    ; Вывод данных из РОН в IO регистр для отображения
out OCR0C, r20          rjmp loop
```

Код теста:

$log OCR0A

$log OCR0B

$log OCR0C

$startlog SBRS_log_output.stim

#6
OCR0A = 0x10

OCR0B = 0x00

#8

OCR0A = 0x00

OCR0B = 0x10

$stoplog

$break

Выходной файл

#3
OCR0A = 0x10 #1
OCR0B = 0x00
#2
OCR0C = 0x01 ; SBRS выполнен, бит установлен
#1
OCR0A = 0x00
OCR0B = 0x10
#2
OCR0C = 0x00 ; SBRS не выполнен, бит не установлен

OCR0A	OCR0B	OCR0C	Результат
0x10	0x00	0x01	SBRS выполнен
0x00	0x10	0x00	SBRS не выполнен

Мнемоника: BRBS

- **Операнды: s, k**
- **Описание: Перейти если флаг в SREG установлен.**
- **Операция: if (SREG(s) == 1) PC = PC + k + 1**
- **Флаги: None**
- **Циклы: 1/2**

Код программы:

```
reset:
    rjmp main

main:
    ; Загрузка значений в регистры
    ldi r18, 0x80  ; Устанавливаем значение, которое вызовет отрицательный результат
    ldi r19, 0x01  ; Добавляем 1, чтобы вызвать перенос

    ; Выполнение операции SUB
    sub r18, r19

    ; Вывод данных из РОН в IO регистр для отображения
    out OCR0A, r18
    nop

loop:
    ; Ввод данных из IO регистров в РОН для обработки
    in r18, OCR0A

    ; Выполнение операции BRBS (проверка флага N)
    brbs 2, negative_flag_set

    ; Если флаг N не установлен, загружаем 0 в r20
    ldi r20, 0x00
    rjmp end

negative_flag_set:
    ; Если флаг N установлен, загружаем 1 в r20
    ldi r20, 0x01

end:
    ; Вывод данных из РОН в IO регистр для отображения
    out OCR0B, r20

       rjmp loop
```

Код теста:

```
$log OCR0A

$log OCR0B

$startlog BRBS_log_output.stim
```

#6

OCR0A = 0x80

#8

OCR0A = 0x7F

$stoplog

$break

Выходной файл:

```
#3
OCR0A = 0x80
#1
OCR0B = 0x01  ; BRBS выполнен, флаг N установлен
#1
OCR0A = 0x7F
#2
OCR0B = 0x00  ; BRBS не выполнен, флаг N не установлен
```

OCR0A	OCR0B	Результат
0x80	0x01	BRBS выполнен
0x7F	0x00	BRBS не выполнен

Мнемоника: BRSH

- **Операнды: k**
- **Описание: Перейти если равно или больше (без знака).**
- **Операция: if (C == 0) PC = PC + k + 1**
- **Флаги: None**
- **Циклы: 1/2**

код программы:

```
reset:
    rjmp main

main:
    ; Загрузка значений в регистры
    ldi r18, 0x20  ; Rd = 32
    ldi r19, 0x10  ; Rr = 16

    ; Вывод данных из РОН в IO регистр для отображения
    out OCR0A, r18
```

```
    out OCR0B, r19
    nop

loop:
    ; Ввод данных из IO регистров в РОН для обработки
    in r18, OCR0A
    in r19, OCR0B

    ; Выполнение операции CP (сравнение)
    cp r18, r19

    ; Выполнение операции BRSH
    brsh greater_or_equal

    ; Если не больше или равно, загружаем 0 в r20
    ldi r20, 0x00
    rjmp end

greater_or_equal:
    ; Если больше или равно, загружаем 1 в r20
    ldi r20, 0x01

end:
    ; Вывод данных из РОН в IO регистр для отображения
    out OCR0C, r20

       rjmp loop
```

код теста:

```
$log OCR0A

$log OCR0B

$log OCR0C

$startlog BRSH_log_output.stim


#6

OCR0A = 0x20

OCR0B = 0x10


#8

OCR0A = 0x10

OCR0B = 0x20


#8

OCR0A = 0x20

OCR0B = 0x20
```

$stoplog

$break

выходной файл:

#3
OCR0A = 0x20
#1
OCR0B = 0x10
#2
OCR0C = 0x01 ; BRSH выполнен, так как 32 >= 16
#1
OCR0A = 0x10
OCR0B = 0x20
#2
OCR0C = 0x00 ; BRSH не выполнен, так как 16 < 32
#1
OCR0A = 0x20
OCR0B = 0x20
#2
OCR0C = 0x01 ; BRSH выполнен, так как 32 == 32

OCR0A	OCR0B	OCR0C	Результат
0x20	0x10	0x01	BRSH выполнен
0x10	0x20	0x00	BRSH не выполнен
0x20	0x20	0x01	BRSH выполнен

Мнемоника: BRCS

- **Операнды: k**
- **Описание: Перейти если перенос установлен.**
- **Операция: if (C == 1) PC = PC + k + 1**
- **Флаги: None**
- **Циклы: 1/2**

код программы:

```
reset:
    rjmp main
```

```
main:
    ; Загрузка значений в регистры
    ldi r18, 0xFF  ; Rd = 255
    ldi r19, 0x01  ; Rr = 1

    ; Вывод данных из РОН в IO регистр для отображения
    out OCR0A, r18
    out OCR0B, r19
    nop

loop:
    ; Ввод данных из IO регистров в РОН для обработки
    in r18, OCR0A
    in r19, OCR0B

    ; Выполнение операции ADD (сложение с переносом)
    add r18, r19

    ; Выполнение операции BRCS
    brcs carry_set

    ; Если перенос не установлен, загружаем 0 в r20
    ldi r20, 0x00
    rjmp end

carry_set:
    ; Если перенос установлен, загружаем 1 в r20
    ldi r20, 0x01

end:
    ; Вывод данных из РОН в IO регистр для отображения
    out OCR0C, r20

       rjmp loop
```

код теста:

$log OCR0A

$log OCR0B

$log OCR0C

$startlog

BRCS_log_output.s

tim

#6

OCR0A = 0xFF

OCR0B = 0x01

#8

```
OCR0A = 0x7F

OCR0B = 0x01


$stoplog

$break
```

выходной файл:

```
#3
OCR0A = 0xFF
#1
OCR0B = 0x01
#2
OCR0C = 0x01  ; BRCS выполнен, перенос установлен
#1
OCR0A = 0x7F
OCR0B = 0x01
#2
OCR0C = 0x00  ; BRCS не выполнен, перенос не установлен
```

OCR0A	**OCR0B**	**OCR0C**	**Результат**
0xFF	0x01	0x01	BRCS выполнен
0x7F	0x01	0x00	BRCS не выполнен

Сортировка выбором

Код:

```
.dseg
array: .dw 10          ; Объявление секции данных с массивом размером 10
.cseg
main:
  ldi ZH, High(src*2)    ; Загружаем верхний байт адреса src в регистр ZH
  ldi ZL, Low(src*2)     ; Загружаем нижний байт адреса src в регистр ZL
```

```
 ldi YH, High(array)    ; Загружаем верхний байт адреса массива в YH
 ldi YL, Low(array)     ; Загружаем нижний байт адреса массива в YL
 ldi r16, 0x0a          ; Устанавливаем счетчик в 10 для копирования элементов

copingarray:
 lpm r17, Z+            ; Загружаем элемент из Flash по адресу Z в r17 и инкрементируем Z
 lpm r18, Z+            ; Загружаем следующий элемент из Flash в r18 и инкрементируем Z
 st Y+, r18             ; Сохраняем r18 в массив и инкрементируем Y
 st Y+, r17             ; Сохраняем r17 в массив и инкрементируем Y
 dec r16                ; Уменьшаем счетчик
 brne copingarray       ; Если не ноль, продолжаем копирование

 ; Подготовка к сортировке
 ldi YH, High(array)    ; Сброс указателя Y на начало массива
 ldi YL, Low(array)
 ldi r16, 0x09          ; Устанавливаем счетчик на 9 (количество сравнений)
 mov r3, r16            ; Сохраняем значение в r3

loopset:
 ldi r16, 0             ; Обнуляем регистры для хранения элементов
 mov r0, r16            ; r0 = 0 для элемента 1
 mov r1, r16            ; r1 = 0 для элемента 2
 mov r16, r3            ; Загружаем количество оставшихся сортируемых элементов
 ld r17, Y+             ; Загружаем первый элемент из массива в r17
 ld r18, Y+             ; Загружаем второй элемент в r18
 ldi r21, 0x01          ; Устанавливаем флаг, чтобы обозначить изменение
 mov r26, r28           ; Копируем значение указателя Y в r26 для обмена
 mov r27, r29           ; Копируем значение указателя X в r27

changesort:
 ld r19, X+             ; Загружаем элемент X в r19
 ld r20, X+             ; Загружаем следующий элемент X в r20
 bclr 0                 ; Обнуляем бит 0, установка флага 0
 cp r19, r17            ; Сравниваем r19 и r17
 brcs lowerelem         ; Если r19 < r17, переходим к lowerelem
 cp r17, r19            ; Сравнение в обратном порядке
```

```
  brcs counterdec        ; Если r17 <= r19, уменьшаем счетчик
  cp r20, r18            ; Сравниваем r20 и r18
  brcs lowerelem         ; Если r20 < r18, переходим к lowerelem
  rjmp counterdec        ; Иначе, уменьшаем счетчик

lowerelem:
  bclr 0                 ; Обнуляем бит 0
  cp r19, r0             ; Сравниваем r19 с элементом 1
  brcs morelower         ; Если r19 < r0, переходим к morelower
  sbrs r21, 0            ; Если изменение не произошло, пропускаем уменьшение флага
  cp r0, r19             ; Сравниваем r0 и r19
  brcs counterdec        ; Если r0 <= r19, уменьшаем счетчик
  cp r20, r1             ; Сравниваем r20 с элементом 2
  brcs morelower         ; Если r20 < r1, переходим к morelower
  sbrs r21, 0            ; Если изменение не произошло, пропускаем уменьшение флага
  rjmp counterdec        ; Иначе, уменьшаем счетчик

morelower:
  mov r0, r19            ; Запоминаем большее значение в r0
  mov r1, r20            ; Запоминаем второе меньшее значение в r1
  mov r2, r26            ; Запоминаем адрес элемента для обмена
  sbrc r21, 0            ; Если флаг изменения установлен, ничего не делаем
  dec r21                ; Уменьшаем флаг изменений

counterdec:
  dec r16                ; Уменьшаем счетчик
  brne changesort        ; Если не ноль, продолжаем сортировку

elementschanges:
  dec r3                 ; Уменьшаем оставшееся количество элементов для сортировки
  sbrs r21, 0            ; Если флаг изменений установлен, ничего не делаем
  rjmp changeplaces      ; Иначе, переходим к обмену местами
  rjmp loopset           ; Возвращаемся к началу цикла

changeplaces:
  subi r28, 0x02         ; Корректируем указатель Y перед сохранением элементов
```

```
    st Y+, r0           ; Сохраняем r0 в массив и инкрементируем Y
    st Y+, r1           ; Сохраняем r1 в массив и инкрементируем Y
    mov r26, r2         ; Восстанавливаем регистр для адреса
    subi r26, 0x02      ; Корректируем указатель X перед обменом элементов
    st X+, r17          ; Сохраняем r17 в память и инкрементируем X
    st X+, r18          ; Сохраняем r18 в память и инкрементируем X
    rjmp loopset        ; Возвращаемся к началу цикла

.cseg
src: .dw 0x1100, 0x1200, 0x1101, 0x0100, 0x0011, 0x0001, 0x0900, 0x812, 0x9320, 0x3456
```

Тестовый файл:

```
$log SREG
$startlog Change_sort_output.stim
$memdump Change_sort_output_Flash.stim 0x007E 20 f
#133
$memdump Change_sort_output_Sram_not_sorted.stim 0x0100 20 s
#1010
$stoplog
$memdump Change_sort_output_Sram_sorted.stim 0x0100 20 s
$break

Change_sort_output:
:02007E0000116F
:1000800000120111000111000100000912082093 63
:020090005634E4
:00000001FF
Change_sort_output_Flash:
:1001000011001200110101000011000109000812 84
:0401100093203456AE
:00000001FF
```

Change_sort_output_Sram_not_sorted:

:1001000000010011010008120900110011011200 84

:0401100034569320AE

:00000001FF

Заключение

Вывод: в ходе работы я разобрал 5 операций и отследил изменения флагов при их выполнении; и 5 команд условного перехода. Научился работать с массивами в ассемблере, написав программу “ Сортировка выбором” .