Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий
институт
Кафедра «Информатика»
кафедра

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Лабораторная работа №6. Программирование на языках ассемблера
тема

Преподаватель _____ А.С. Кузнецов
подпись, дата инициалы, фамилия

Студент КИ18-17/1б 031830504 _____ Е.В. Железкин
номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2019

СОДЕРЖАНИЕ

1 Цель работы

Разработка программ на языках ассемблера.

2 Краткие теоретические сведения

Ассемблер (от англ. assembler — сборщик) — транслятор исходного текста программы, написанной на языке ассемблера, в программу на машинном языке.

3 Задача работы

Требуется: разработать ассемблерную программу, исходный код которой представляет собой программу, разделенную на основную часть и подпрограммы (не менее двух). Результат вычислений выводится на экран.

Вариант 9. Дана матрица размера N x M. Вывести количество строк столбцов, элементы которых монотонно возрастают, монотонно убывают

4 Ход работы

Листинг 1 – содержание файла SP_6_MIPS_32.s, вычислительная система MIPS:

```
# Вариант 9. Дана матрица размера N x M. Вывести количество строк столбцов,
# элементы которых монотонно возрастают, монотонно убывают

# CheckG(a0 - sp, s0 - N, s1 - M) функция, проверяющая строки по условию
    .globl CheckG
CheckG:
    addi $sp, $sp, -4
    sw $ra, ($sp)
  addi $sp, $sp, -4
    sw $s3, ($sp)
    addi $sp, $sp, -4
```

```
    sw $s6, ($sp)
    addi $sp, $sp, -4
    sw $s7, ($sp)
    addi $sp, $sp, -4
    sw $s8, ($sp)

    move $s6, $a0
    move $s7, $a1      # s7 - N
    move $s8, $a2    # s8 - M

#  if (matrix[i *   m + j -  1]   >   matrix[i *  m +   j])
#  if (matrix[s0 * s8 + s1 - 1] (bgt) matrix[s0 * s8 + s1])

#  if (matrix[(i + 1) * m  - j -  1]   <   matrix[(i + 1) * m   - j])
#  if (matrix[(s0 + 1) * s8 - s1 - 1] (blt) matrix[(s0 + 1) * s8 - s1])

  li $t5, 1
  li $s0, 0
  li $s1, 0
  li $t1, 0
  li $t0, 0
  li $s3, 0
  li $t4, 4
  li $v0, 1

# for ($s0 = 0; $s0 < $s7; $s0++)
first_loop_G:
  # for ($s1 = 1; $s1 < $s8; $s1++)
  second_loop_G:
    beq $s1, $s8, second_exit_G

    addi $s1, $s1, 1

    li $t0, 1
    mul $t0, $t0, $s0
    mul $t0, $t0, $s8
    add $t0, $t0, $s1
    sub $t0, $t0, $t5
    mul $t0, $t0, $t4
    add $t0, $t0, $s6
```

```
            lw $t0, ($t0)
            li $t1, 1
            mul $t1, $t1, $s0
            mul $t1, $t1, $s8
            add $t1, $t1, $s1
            mul $t1, $t1, $t4
            add $t1, $t1, $s6
            lw $t1, ($t1)
              bgt $t0, $t1, first_flag_G

            continue_loop_G:  #  if (matrix[(s0 + 1) * s8 - s1 - 1] (blt) matrix[(s0 + 1)
* s8 - s1])

              li $t0, 1
              add $t0, $t0, $s0
              mul $t0, $t0, $s8
              sub $t0, $t0, $s1
              sub $t0, $t0, $t5
              mul $t0, $t0, $t4
              add $t0, $t0, $s6
              lw $t0, ($t0)
              li $t1, 1
              add $t1, $t1, $s0
              mul $t1, $t1, $s8
              sub $t1, $t1, $s1
              mul $t1, $t1, $t4
              add $t1, $t1, $s6
              lw $t1, ($t1)

              blt $t0, $t1, second_flag_G

              j second_loop_G

          second_exit_G:

            bne $t8, $zero, nextop_G
            addi $s3, $s3, 1
            j end_of_loop_G
            nextop_G:
              bne $t9, $zero, end_of_loop_G
```

```
      addi $s3, $s3, 1

      end_of_loop_G:

        addi $s0, $s0, 1
        li $s1, 0
        li $t8, 0
        li $t9, 0

        beq $s0, $s7, first_exit_G
        j first_loop_G

        first_flag_G:
          addi $t8, $t8, 1
          j continue_loop_G

        second_flag_G:
          addi $t9, $t9, 1
          j second_exit_G
first_exit_G:

    move $v0, $s3

    lw $s8, ($sp)
    addi $sp, $sp, 4
    lw $s7, ($sp)
    addi $sp, $sp, 4
    lw $s6, ($sp)
    addi $sp, $sp, 4
    lw $s3, ($sp)
    addi $sp, $sp, 4
    lw $ra, ($sp)
    addi $sp, $sp, 4

    jr $ra



# CheckV(a0 - sp, s0 - N, s1 - M) функция, проверяющая столбцы по условию
    .globl CheckV
```

```
CheckV:
    addi $sp, $sp, -4
    sw $ra, ($sp)
  addi $sp, $sp, -4
    sw $s3, ($sp)
    addi $sp, $sp, -4
    sw $s6, ($sp)
    addi $sp, $sp, -4
    sw $s7, ($sp)
    addi $sp, $sp, -4
    sw $s8, ($sp)

    move $s6, $a0
    move $s7, $a1      # s7 - N
    move $s8, $a2    # s8 - M

#  if (matrix[m  * (j -  1) +  i]   >   matrix[m  * j  +  i])
#  if (matrix[s8 * (s1 - 1) + s0] (bgt) matrix[s8 * s1 + s0])

#  if (matrix[m  * (n  - j  - 1) +  i]   >   matrix[m  * (n  -  j) +  i])
#  if (matrix[s8 * (s7 - s1 - 1) + s0] (bgt) matrix[s8 * (s7 - s1) + s0])

  li $t5, 1
  li $s0, 0
  li $s1, 0
  li $t1, 0
  li $t0, 0
  li $s3, 0
  li $t4, 4
  li $v0, 1

# for ($s0 = 0; $s0 < $s8; $s0++)
first_loop_V:
  # for ($s1 = 1; $s1 < $s7; $s1++)
  second_loop_V:
    beq $s1, $s7, second_exit_V

    addi $s1, $s1, 1 #  if (matrix[s8 * (s1 - 1) + s0] (bgt) matrix[s8 * s1 + s0])

    li $t0, 1
```

```
            li $t6, -1
            add $t6, $t6, $s1
            mul $t0, $t0, $s8
            mul $t0, $t0, $t6
            add $t0, $t0, $s0
            mul $t0, $t0, $t4
            add $t0, $t0, $s6
            lw $t0, ($t0)

            li $t1, 1
            mul $t1, $t1, $s8
            mul $t1, $t1, $s1
            add $t1, $t1, $s0
            mul $t1, $t1, $t4
            add $t1, $t1, $s6
            lw $t1, ($t1)

              bgt $t0, $t1, first_flag_V

            continue_loop_V:  #  if (matrix[s8 * (s7 - s1 - 1) + s0] (bgt) matrix[s8 * (s7
- s1) + s0])

              li $t0, -1
              add $t0, $s7, $t0
              sub $t0, $t0, $s1
              mul $t0, $t0, $s8
              add $t0, $t0, $s0
              mul $t0, $t0, $t4
              add $t0, $t0, $s6
              lw $t0, ($t0)

              li $t1, 0
              add $t1, $t1, $s7
              sub $t1, $t1, $s1
              mul $t1, $t1, $s8
              add $t1, $t1, $s0
              mul $t1, $t1, $t4
              add $t1, $t1, $s6
              lw $t1, ($t1)
```

```
      blt $t0, $t1, second_flag_V

      j second_loop_V

  second_exit_V:

    bne $t8, $zero, nextop_V
    addi $s3, $s3, 1
    j end_of_loop_V
    nextop_V:
      bne $t9, $zero, end_of_loop_V
      addi $s3, $s3, 1

      end_of_loop_V:

        addi $s0, $s0, 1
        li $s1, 0
        li $t8, 0
        li $t9, 0

        beq $s0, $s8, first_exit_V
        j first_loop_V

        first_flag_V:
          addi $t8, $t8, 1
          j continue_loop_V

        second_flag_V:
          addi $t9, $t9, 1
          j second_exit_V
first_exit_V:

    move $v0, $s3

    lw $s8, ($sp)
    addi $sp, $sp, 4
    lw $s7, ($sp)
    addi $sp, $sp, 4
    lw $s6, ($sp)
    addi $sp, $sp, 4
```

```
    lw $s3, ($sp)
    addi $sp, $sp, 4
    lw $ra, ($sp)
    addi $sp, $sp, 4

    jr $ra


# Write matrix(a0 - sp, s0 - N, s1 - M)
    .globl Write
Write:
    addi $sp, $sp, -4
    sw $ra, ($sp)
    addi $sp, $sp, -4
    sw $s6, ($sp)
    addi $sp, $sp, -4
    sw $s7, ($sp)
    addi $sp, $sp, -4
    sw $s8, ($sp)

    move $s6, $a0
    move $s7, $a1
    move $s8, $a2

  li $s0, 0
  li $s1, 0
  li $t1, 4
  li $t0, 0

  # for ($s0 = 0; $s0 < $s7; $s0++)
  first_loop_write:
    # for ($s1 = 0; $s1 < $s8; $s1++)
    second_loop_write:
      beq $s1, $s8, second_exit_write

      li $v0, 1
      mul $t0, $s0, $s8
            add $t0, $t0, $s1
            mul $t0, $t0, $t1
            add $t0, $s6, $t0
```

```
            lw $a0, ($t0)
            syscall
            li $v0, 4
            la $a0, Space
            syscall
      addi $s1, $s1, 1

      j second_loop_write

    second_exit_write:
      addi $s0, $s0, 1
      li $s1, 0

      li $v0, 4
            la $a0, NewLine
            syscall

      beq $s0, $s7, first_exit_write
      j first_loop_write
  first_exit_write:

    lw $s8, ($sp)
    addi $sp, $sp, 4
    lw $s7, ($sp)
    addi $sp, $sp, 4
  lw $s6, ($sp)
    addi $sp, $sp, 4
    lw $ra, ($sp)
    addi $sp, $sp, 4

    jr $ra


# main
    .globl main
main:
  li $v0, 4
    la $a0, FirstMsg
    syscall
```

```
    li $v0, 5
    syscall
    move $s7, $v0              # read s7 - N

    li $v0, 5
    syscall
    move $s8, $v0              # read s8 - M

  li $v0, 4
  la $a0, SecondMsg
  syscall

  li $t0, 4
  mul $t0, $t0, $s7
    mul $t0, $t0, $s8
    sub $sp, $sp, $t0

  li $s0, 0
  li $s1, 0
  li $t1, 4
  li $t0, 0

# for ($s0 = 0; $s0 < $s7; $s0++)
first_loop:
  # for ($s1 = 0; $s1 < $s8; $s1++)
  second_loop:
    beq $s1, $s8, second_exit

    li $v0, 5
        syscall
        move $t2, $v0
    mul $t0, $s0, $s8
    add $t0, $t0, $s1
    mul $t0, $t0, $t1
    add $t0, $t0, $sp
    sw $t2, ($t0)

    addi $s1, $s1, 1

    j second_loop
```

```
  second_exit:
    addi $s0, $s0, 1
    li $s1, 0
    beq $s0, $s7, first_exit
    j first_loop
first_exit:

    move $a0, $sp
    move $a1, $s7
    move $a2, $s8
  jal Write

  move $a0, $sp
    move $a1, $s7
    move $a2, $s8
  jal CheckV
  move $s2, $v0

  move $a0, $sp
    move $a1, $s7
    move $a2, $s8
  jal CheckG
  add $s2, $s2, $v0
    li $v0, 1
  move $a0, $s2
    syscall


  li $v0, 10
    syscall

  .data

FirstMsg:
    .asciiz "Введите размер матрицы(Два целых числа, M x N): "

NewLine:
    .asciiz "\n"
```

```
    SecondMsg:
        .asciiz "Введите матрицу(целые числа, по одному элементу): "


    Space:
        .asciiz " "
```

Листинг 2 – содержание файла SP_6_x86_32.s, вычислительная система x86:

```
# Вариант 9. Дана матрица размера N x M. Вывести количество строк столбцов,
# элементы которых монотонно возрастают, монотонно убывают


        #   4(%ebp) [4]  old %ebp
    #   0(%ebp) [4]
    #  -4(%ebp) [4]  n
    #  -8(%ebp) [4]  m
    # -12(%ebp) [4]  matrix
    # -16(%ebp) [4]  i
    # -20(%ebp) [4]  j
    # -24(%ebp) [4]  flag1
        # -28(%ebp) [4]  flag2
        # -32(%ebp) [4]  temp_res
        # -36(%ebp) [4]  temp M
        # -40(%ebp) [4]  result
        # -44(%ebp) [4]  0 (%esp)

    .globl main
main:

        pushl %ebp
    movl %esp, %ebp
    #pushl %esi
    subl $44, %esp

        pushl $FirstMsg
    calll  printf
    addl  $4, %esp

        leal -8(%ebp), %edx
        pushl %edx
    leal -4(%ebp), %edx
        pushl %edx
    leal FormatPairIn, %edx
    pushl %edx
    calll scanf
        addl $12, %esp

        pushl $SecondMsg
    calll  printf
    addl  $4, %esp

        movl -4(%ebp), %eax
    imull -8(%ebp), %eax
    imull $4, %eax
    movl %eax, (%esp)
    calll malloc
```

```
    movl %eax, -12(%ebp)

        movl $0, %eax
        movl $0, %ebx
        movl $0, %ecx
        movl $0, %edx
        movl $0, -20(%ebp)
        movl $0, -16(%ebp)


# Считывание матрицы
# for (-16(%ebp) = 0; -16(%ebp) < -4(%ebp); -16(%ebp)++)
first_loop_I:
        # for (-20(%ebp) = 0; -20(%ebp) < -8(%ebp); -20(%ebp)++)
        second_loop_I:
                movl -20(%ebp), %eax
                cmp %eax, -8(%ebp)
                je second_exit_I

                movl -16(%ebp), %ecx
                imull -8(%ebp), %ecx
                addl -20(%ebp), %ecx
                imull $4, %ecx
                addl -12(%ebp), %ecx
                pushl %ecx
                pushl $FormatDigitIn
                calll scanf
                addl $8, %esp

                movl -20(%ebp), %eax
                addl $1, %eax
                movl %eax, -20(%ebp)

                jmp second_loop_I

        second_exit_I:
                movl -16(%ebp), %eax
                addl $1, %eax
                movl %eax, -16(%ebp)
                movl -16(%ebp), %eax
                cmp %eax, -4(%ebp)
                je first_exit_I

                movl $0, %ebx
                movl %ebx, -20(%ebp)

                jmp first_loop_I
first_exit_I:

        movl $0, -16(%ebp)
        movl $0, -20(%ebp)

# Вывод матрицы
# for (-16(%ebp) = 0; -16(%ebp) < -4(%ebp); -16(%ebp)++)
        first_loop_W:
                # for (-20(%ebp) = 0; -20(%ebp) < -8(%ebp); -20(%ebp)++)
                second_loop_W:
                        movl -20(%ebp), %eax
                        cmp %eax, -8(%ebp)
                        je second_exit_W
```

```
                        movl -16(%ebp), %ecx
                        imull -8(%ebp), %ecx
                        addl -20(%ebp), %ecx
                        imull $4, %ecx
                        addl -12(%ebp), %ecx
                        movl (%ecx), %edx
                        pushl %edx
                        pushl $FormatDigitOut
                        calll printf
                        addl $8, %esp

                        movl -20(%ebp), %eax
                        addl $1, %eax
                        movl %eax, -20(%ebp)

                        jmp second_loop_W

                second_exit_W:
                        pushl $NewLine
                        calll printf
                        addl $4, %esp

                        movl -16(%ebp), %eax
                        addl $1, %eax
                        movl %eax, -16(%ebp)
                        movl -16(%ebp), %eax
                        cmp %eax, -4(%ebp)
                        je first_exit_W

                        movl $0, %ebx
                        movl %ebx, -20(%ebp)

                        jmp first_loop_W
        first_exit_W:


        movl $0, -16(%ebp)
        movl $0, -20(%ebp)
        movl $0, -24(%ebp)
        movl $0, -28(%ebp)
        movl $0, -32(%ebp)
        movl -8(%ebp), %eax
        subl $1, %eax
        movl %eax, -36(%ebp)
        movl $0, -40(%ebp)

#  Проверка строк

#  if (matrix[i *   m + j -  1]   >   matrix[i *  m +   j])
#  if (matrix[-16(%ebp) * -8(%ebp) + -20(%ebp) - 1] > matrix[-16(%ebp) * -8(%ebp) + -
20(%ebp)])

#  if (matrix[(i + 1) * m  - j -  1]   <   matrix[(i + 1) * m   - j])
#  if (matrix[(-16(%ebp) + 1) * -8(%ebp) - -20(%ebp) - 1] < matrix[(-16(%ebp) + 1) * -
8(%ebp) - -20(%ebp)])

        # for (-16(%ebp) = 0; -16(%ebp) < -4(%ebp); -16(%ebp)++)
        first_loop_G:
                # for (-20(%ebp) = 1; -20(%ebp) < -8(%ebp); -20(%ebp)++)
```

```
second_loop_G:
        movl -20(%ebp), %eax
        cmp %eax, -36(%ebp)
        je second_exit_G

        movl -20(%ebp), %eax
        addl $1, %eax
        movl %eax, -20(%ebp)

        movl -16(%ebp), %eax
        imull -8(%ebp), %eax
        addl -20(%ebp), %eax
        movl %eax, %ebx
        subl $1, %eax
        imull $4, %eax
        imull $4, %ebx

        addl -12(%ebp), %eax
        addl -12(%ebp), %ebx

        movl (%eax), %ecx
        movl (%ebx), %edx

        cmp %ecx, %edx

        jg change_flag_1_G

        continue_loop_G:

                movl -16(%ebp), %eax
                addl $1, %eax
                imull -8(%ebp), %eax
                subl -20(%ebp), %eax

                movl %eax, %ebx
                subl $1, %eax
                imull $4, %eax
                imull $4, %ebx

                addl -12(%ebp), %eax
                addl -12(%ebp), %ebx

                movl (%eax), %ecx
                movl (%ebx), %edx

                cmp %ecx, %edx

                jl change_flag_2_G

                jmp second_loop_G

second_exit_G:

        movl -24(%ebp), %eax
        cmp $0, %eax
        jne nextop_G

        movl -32(%ebp), %eax
        addl $1, %eax
        movl %eax, -32(%ebp)
```

```
                        jmp end_of_loop_G

                        nextop_G:
                                movl -28(%ebp), %eax
                                cmp $0, %eax
                                jne end_of_loop_G

                                movl -32(%ebp), %eax
                                addl $1, %eax
                                movl %eax, -32(%ebp)

                        end_of_loop_G:

                                movl -16(%ebp), %eax
                                addl $1, %eax
                                movl %eax, -16(%ebp)
                                cmp %eax, -4(%ebp)
                                je first_exit_G

                                movl $0, %ebx
                                movl %ebx, -20(%ebp)

                                movl $0, %ebx
                                movl %ebx, -24(%ebp)

                                movl $0, %ebx
                                movl %ebx, -28(%ebp)

                                jmp first_loop_G

                                change_flag_1_G:
                                        movl -24(%ebp), %eax
                                        addl $1, %eax
                                        movl %eax, -24(%ebp)
                                        jmp continue_loop_G

                                change_flag_2_G:
                                        movl -28(%ebp), %eax
                                        addl $1, %eax
                                        movl %eax, -28(%ebp)
                                        jmp second_loop_G

        first_exit_G:


        movl -32(%ebp), %eax
        movl %eax, -40(%ebp)

        movl $0, -16(%ebp)
        movl $0, -20(%ebp)
        movl $0, -24(%ebp)
        movl $0, -28(%ebp)
        movl $0, -32(%ebp)
        movl -4(%ebp), %eax
        subl $1, %eax
        movl %eax, -36(%ebp)

#  Проверка столбцов
```

```
#  if (matrix[m  * (j -  1) +  i]   >   matrix[m  * j  +  i])
#  if (matrix[-8(%ebp) * (-20(%ebp) - 1) + -16(%ebp)] > matrix[-8(%ebp) * -20(%ebp) + -
16(%ebp)])

#  if (matrix[m  * (n  - j  - 1) +  i]   >   matrix[m  * (n  -  j) +  i])
#  if (matrix[-8(%ebp) * (-4(%ebp) - -20(%ebp) - 1) + -16(%ebp)] > matrix[-8(%ebp) * (-
4(%ebp) - -20(%ebp)) + -16(%ebp)])

        # for (-16(%ebp) = 0; -16(%ebp) < $-8(%ebp); -16(%ebp)++)
        first_loop_V:
                # for (-20(%ebp) = 1; -20(%ebp) < -4(%ebp); -20(%ebp)++)
                second_loop_V:
                        movl -20(%ebp), %eax
                        cmp %eax, -36(%ebp)
                        je second_exit_V

                        movl -20(%ebp), %eax
                        addl $1, %eax
                        movl %eax, -20(%ebp)

                        movl -20(%ebp), %eax
                        subl $1, %eax
                        imull -8(%ebp), %eax
                        addl -16(%ebp), %eax
                        imull $4, %eax

                        movl -20(%ebp), %ebx
                        imull -8(%ebp), %ebx
                        addl -16(%ebp), %ebx
                        imull $4, %ebx

                        addl -12(%ebp), %eax
                        addl -12(%ebp), %ebx

                        movl (%eax), %ecx
                        movl (%ebx), %edx

                        cmp %ecx, %edx

                        jg change_flag_1_V
                                                                #  if (matrix[m  * (n  - j  -
1) +  i]   >   matrix[m  * (n  -  j) +  i])
                        continue_loop_V:#  if (matrix[-8(%ebp) * (-4(%ebp) - -20(%ebp)
- 1) + -16(%ebp)] > matrix[-8(%ebp) * (-4(%ebp) - -20(%ebp)) + -16(%ebp)])

                                movl -4(%ebp), %eax
                                subl -20(%ebp), %eax
                                subl $1, %eax
                                imull -8(%ebp), %eax
                                addl -16(%ebp), %eax
                                imull $4, %eax

                                movl -4(%ebp), %ebx
                                subl -20(%ebp), %ebx
                                imull -8(%ebp), %ebx
                                addl -16(%ebp), %ebx
                                imull $4, %ebx

                                addl -12(%ebp), %eax
                                addl -12(%ebp), %ebx
```

```
                movl (%eax), %ecx
                movl (%ebx), %edx

                cmp %ecx, %edx


                jl change_flag_2_V

                jmp second_loop_V

second_exit_V:

        movl -24(%ebp), %eax
        cmp $0, %eax
        jne nextop_V

        movl -32(%ebp), %eax
        addl $1, %eax
        movl %eax, -32(%ebp)

        jmp end_of_loop_V

        nextop_V:
                movl -28(%ebp), %eax
                cmp $0, %eax
                jne end_of_loop_V

                movl -32(%ebp), %eax
                addl $1, %eax
                movl %eax, -32(%ebp)

        end_of_loop_V:

                movl -16(%ebp), %eax
                addl $1, %eax
                movl %eax, -16(%ebp)
                cmp %eax, -8(%ebp)
                je first_exit_V

                movl $0, %ebx
                movl %ebx, -20(%ebp)

                movl $0, %ebx
                movl %ebx, -24(%ebp)

                movl $0, %ebx
                movl %ebx, -28(%ebp)

                jmp first_loop_V

                change_flag_1_V:
                        movl -24(%ebp), %eax
                        addl $1, %eax
                        movl %eax, -24(%ebp)
                        jmp continue_loop_V

                change_flag_2_V:
                        movl -28(%ebp), %eax
                        addl $1, %eax
```

```
                                        movl %eax, -28(%ebp)
                                        jmp second_loop_V

        first_exit_V:

#  Обработка и вывод результатов

        movl -40(%ebp), %eax
        addl -32(%ebp), %eax
        pushl %eax
        pushl $FormatDigitOut
        calll printf
        addl  $8, %esp

        pushl $NewLine
        calll printf
        addl  $4, %esp

        addl $44, %esp
    popl %ebp

        retl



        .data

FirstMsg:
    .asciz "Введите размер матрицы(Два целых числа, M x N):\n"

NewLine:
    .asciz "\n"

SecondMsg:
    .asciz "Введите матрицу(целые числа, по одному элементу): "

Space:
    .asciz " "

FormatPairIn:
    .asciz "%d%d"

FormatDigitIn:
    .asciz "%d"

FormatDigitOut:
    .asciz "%d "

FormatPairOut:
    .asciz "%d %d\n"
```

Листинг 3 – содержание файла SP_6_aarch64.s, вычислительная система ARM(64bit):

```
        // 96(sp) [8] old x19
        // 88(sp) [8] old x30
        // 80(sp) [8] old x21
        // 72(sp) [8] old x20
```

```
        // 64(sp) [8] n
        // 56(sp) [8] m
        // 48(sp) [8] matrix -> x15
    // 40(sp) [8]
        // 32(sp) [8]
        // 24(sp) [8]
        // 16(sp) [8]
        //  8(sp) [8]
        //  0(sp) [8]

        .global main
main:
    str x19, [sp, #-8]!
        str x20, [sp, #-8]!
    str x21, [sp, #-8]!
        str x30, [sp, #-8]!

        adr x0, FirstMsg
    bl printf

        sub sp, sp, #64


    mov x2, fp
    mov x1, x2
    add x1, x1, #-8
        adr x0, FormatPairIn
        bl scanf

        ldr w20, [fp]
        ldr w21, [fp, #-8]
        mul w23, w20, w21
        mov w20, #8
        mul w23, w23, w20
        sub sp, sp, w23

        mov x23, #0
        mov x24, #0

// Ввод матрицы
// for (x23 = 0; x23 < [fp]; x23++)
first_loop_R:
        // for (x24 = 0; x24 < [fp, #-8]; x24++)
        second_loop_R:
                ldr w20, [fp, #-8]
                cmp w24, w20
                beq second_exit_R

                mov w20, w23
                ldr w21, [fp]
                mul w20, w20, w21
                add w20, w20, w24
                mov w21, #8
                mul w20, w20, w21
                mov x15, sp
                add x20, x20, x15

                mov x1, x20
                adr x0, FormatDigitIn
                bl scanf
```

```
                add w24, w24, #1

                b second_loop_R

        second_exit_R:
                add w23, w23, #1
                mov w24, #0
                ldr w20, [fp]
                cmp w23, w20
                beq first_exit_R
                b first_loop_R
first_exit_R:


        mov w23, #0
        mov w24, #0

// Вывод матрицы
// for (x23 = 0; x23 < [fp]; x23++)
first_loop_W:
        // for (x24 = 0; x24 < [fp, #-8]; x24++)
        second_loop_W:
                ldr w20, [fp, #-8]
                cmp w24, w20
                beq second_exit_W

                mov w20, w23
                ldr w21, [fp]
                mul w20, w20, w21
                add w20, w20, w24
                mov w21, #8
                mul w20, w20, w21
                mov x15, sp
                add x20, x20, x15

                ldr x1, [x20]
                adr x0, FormatDigitOut
                bl printf

                add w24, w24, #1

                b second_loop_W

        second_exit_W:
                adr x0, NewLine
                bl printf

                add w23, w23, #1
                mov w24, #0
                ldr w20, [fp]
                cmp w23, w20
                beq first_exit_W
                b first_loop_W
first_exit_W:


        mov w23, #0
        mov w24, #0
        mov w25, #0
```

```
        mov w26, #0
        mov w27, #0
        mov x28, #0


//  Проверка строк
//  if (matrix[m * i + j - 1] > matrix[m * i + j])
//  if (matrix[[sp, #-8] * w23 + w24 - 1] > matrix[[sp, #-8] * w23 + w24])

//  if (matrix[m * i + j - 1] < matrix[m * i + j])
//  if (matrix[[sp, #-8] * w23 + w24 - 1] < matrix[[sp, #-8] * w23 + w24])

        // for (x23 = 0; x23 < [fp]; x23++)
        first_loop_G:
                // for (x24 = 1; x24 < [fp, #-8]; x24++)
                second_loop_G:
                        ldr w19, [fp, #-8]

                        add w24, w24, #1

                        mov w20, w24

                        cmp w19, w20

                        b.eq second_exit_G

                        ldr w19, [fp, #-8]
                        mul w19, w19, w23
                        add w19, w19, w24
                        mov w20, w19
                        sub w19, w19, #1
                        mov w21, #8
                        mul w19, w19, w21
                        mul w20, w20, w21
                        mov x21, sp
                        add x19, x19, x21
                        add x20, x20, x21

                        ldr x21, [x19]
                        ldr x22, [x20]

                        cmp w21, w22

                        b.gt change_flag_1_G

                        continue_loop_G:

                                ldr w19, [fp, #-8]
                                mul w19, w19, w23
                                add w19, w19, w24
                                mov w20, w19
                                sub w19, w19, #1
                                mov w21, #8
                                mul w19, w19, w21
                                mul w20, w20, w21
                                mov x15, sp
                                add x19, x19, x15
                                add x20, x20, x15

                                ldr x21, [x19]
```

```
                                ldr x22, [x20]

                                cmp w21, w22

                                b.lt change_flag_2_G

                                b second_loop_G

                        second_exit_G:

                                mov w20, w25
                                mov w21, #0
                                cmp w20, w21
                                b.ne nextop_G

                                add w27, w27, #1

                                b end_of_loop_G

                                nextop_G:
                                        mov w20, w26
                                        mov w21, #0
                                        cmp w20, w21
                                        b.ne end_of_loop_G

                                        add x27, x27, #1

                                end_of_loop_G:

                                        mov w25, #0
                                        mov w26, #0
                                        mov w24, #0

                                        add w23, w23, #1
                                        ldr w19, [fp]

                                        cmp w19, w23
                                        b.eq first_exit_G

                                        b first_loop_G

                                        change_flag_1_G:
                                                add w25, w25, #1
                                                b continue_loop_G

                                        change_flag_2_G:
                                                add w26, w26, #1
                                                b second_loop_G

                first_exit_G:


                mov w23, #0
                mov w24, #0
                mov w25, #0
                mov w26, #0
                mov x28, #0


//  Проверка столбцов
```

```
//  if (matrix[j * m + i] > matrix[(j - 1) * m + i])
//  if (matrix[[sp, #-8] * w24 + w23] > matrix[[sp, #-8] * (w24 - 1) + w23])

//  if (matrix[j * m + i] < matrix[(j - 1) * m + i])
//  if (matrix[[sp, #-8] * w24 + w23] < matrix[[sp, #-8] * (w24 - 1) + w23])

        // for (x23 = 0; x23 < [fp, #-8]; x23++)
        first_loop_V:
                // for (x24 = 1; x24 < [fp]; x24++)
                second_loop_V:
                        ldr w19, [fp]

                        add w24, w24, #1

                        mov w20, w24

                        cmp w19, w20

                        b.eq second_exit_V

                        ldr w19, [fp, #-8]
                        mul w19, w19, w24
                        add w19, w19, w23
                        mov w20, w24
                        sub w20, w20, #1
                        ldr w21, [fp, #-8]
                        mul w20, w20, w21
                        add w20, w20, w23

                        mov w21, #8
                        mul w19, w19, w21
                        mul w20, w20, w21

                        mov x21, sp
                        add x19, x19, x21
                        add x20, x20, x21

                        ldr x21, [x19]
                        ldr x22, [x20]

                        cmp w21, w22

                        b.gt change_flag_1_V

                        continue_loop_V:

                                ldr w19, [fp, #-8]
                                mul w19, w19, w24
                                add w19, w19, w23
                                mov w20, w24
                                sub w20, w20, #1
                                ldr w21, [fp, #-8]
                                mul w20, w20, w21
                                add w20, w20, w23

                                mov w21, #8
                                mul w19, w19, w21
                                mul w20, w20, w21

                                mov x21, sp
```

```
                        add x19, x19, x21
                        add x20, x20, x21

                        ldr x21, [x19]
                        ldr x22, [x20]

                        cmp w21, w22

                        b.lt change_flag_2_V

                        b second_loop_V

        second_exit_V:

                mov w20, w25
                mov w21, #0
                cmp w20, w21
                b.ne nextop_V

                add w27, w27, #1

                b end_of_loop_V

                nextop_V:
                        mov w20, w26
                        mov w21, #0
                        cmp w20, w21
                        b.ne end_of_loop_V

                        add x27, x27, #1

                end_of_loop_V:

                        add w23, w23, #1
                        ldr w19, [fp, #-8]

                        cmp w19, w23
                        b.eq first_exit_V

                        mov w20, #0
                        mov w25, #0
                        mov w26, #0
                        mov w24, #0

                        b first_loop_V

                        change_flag_1_V:
                                add w25, w25, #1
                                b continue_loop_V

                        change_flag_2_V:
                                add w26, w26, #1
                                b second_loop_V

first_exit_V:

mov x1, x27
adr x0, FormatDigitOut
bl printf
```

```
        ldr w3, [fp]
        ldr w4, [fp, #-8]
        mul w0, w3, w4
        mov w1, #8
        mul w0, w0, w1
        add sp, sp, w0

        add sp, sp, #64

        ldr x30, [sp], #8
        ldr x21, [sp], #8
        ldr x20, [sp], #8
        ldr x19, [sp], #8

    mov x0, #0

    ret



FirstMsg:
    .asciz "Введите размер матрицы(Два целых числа, M x N):\n"

NewLine:
    .asciz "\n"

SecondMsg:
    .asciz "Введите матрицу(целые числа, по одному элементу): "

Space:
    .asciz " "

FormatPairIn:
    .asciz "%d%d"

FormatDigitIn:
    .asciz "%d"

FormatDigitOut:
    .asciz "%d "

FormatPairOut:
    .asciz "%d %d\n"
```

1) Примеры работы:

```
super@DESKTOP-34H6L9A:/mnt/c/Users/super/OneDrive/Stud/SP/SP_6/MIPS$ spim -f SP_6_MIPS_32.s
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Введите размер матрицы(Два целых числа, M x N): 2
2
Введите матрицу(целые числа, по одному элементу): 1
2
3
4
1 2
3 4
4
```

Рисунок 1 – результат работы 1 на системе MIPS

```
super@DESKTOP-34H6L9A:/mnt/c/Users/super/OneDrive/Stud/SP/SP_6/MIPS$spim -f SP_6_MIPS_32.s
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Введите размер матрицы(Два целых числа, M x N): 3
3
Введите матрицу(целые числа, по одному элементу): 1
2
3
4
5
6
7
8
9
1 2 3
4 5 6
7 8 9
6
```

Рисунок 2 – результат работы 2 на системе MIPS

```
super@DESKTOP-34H6L9A:/mnt/c/Users/super/OneDrive/Stud/SP/SP_6/MIPS$ spim -f SP_6_MIPS_32.s
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Введите размер матрицы(Два целых числа, M x N): 3
3
Введите матрицу(целые числа, по одному элементу): 3
3
3
3
1
3
3
3
3
3 3 3
3 1 3
3 3 3
4
```

Рисунок 3 - результат работы 3 на системе MIPS

```
super@DESKTOP-34H6L9A:/mnt/c/Users/super/OneDrive/Stud/SP/SP_6/x86$ qemu-i386 ./1
Введите размер матрицы(Два целых числа, M x N):
2
2
Введите матрицу(целые числа, по одному элементу): 1
2
3
4
1 2
3 4
4
```

Рисунок 4 – результат работы 1 на системе x86

```
super@DESKTOP-34H6L9A:/mnt/c/Users/super/OneDrive/Stud/SP/SP_6/x86$ qemu-i386 ./1
Введите размер матрицы(Два целых числа, M x N):
3
3
Введите матрицу(целые числа, по одному элементу): 1
2
3
4
5
6
7
8
9
1 2 3
4 5 6
7 8 9
6
```

Рисунок 5 – результат работы 2 на системе x86

```
super@DESKTOP-34H6L9A:/mnt/c/Users/super/OneDrive/Stud/SP/SP_6/x86$ qemu-i386 ./1
Введите размер матрицы(Два целых числа, M x N):
3
3
Введите матрицу(целые числа, по одному элементу): 3
3
3
3
1
3
3
3
3
3 3 3
3 1 3
3 3 3
4
```

Рисунок 6 – результат работы 3 на системе x86

```
superzloyuser@DESKTOP-M1A6FS2:/mnt/c/Users/super/OneDrive/Stud/SP/SP_6/ARM$ qemu-aarch64 ./2
Введите размер матрицы(Два целых числа, M x N):
2
2
1
2
3
4
1 2
3 4
4
```

Рисунок 7 – результат работы 1 на системе aarch64

```
superzloyuser@DESKTOP-M1A6FS2:/mnt/c/Users/super/OneDrive/Stud/SP/SP_6/ARM$ qemu-aarch64 ./2
Введите размер матрицы(Два целых числа, M x N):
3
3
1
2
3
4
5
6
7
8
9
1 2 3
4 5 6
7 8 9
6
```

Рисунок 8 – результат работы 2 на системе aarch64

```
superzloyuser@DESKTOP-M1A6FS2:/mnt/c/Users/super/OneDrive/Stud/SP/SP_6/ARM$ qemu-aarch64 ./2
Введите размер матрицы(Два целых числа, M x N):
3
3
3
3
3
3
5
3
3
3
3
3 3 3
3 5 3
3 3 3
4
```

Рисунок 8 – результат работы 3 на системе aarch64

```
superzloyuser@DESKTOP-M1A6FS2:/mnt/c/Users/super/OneDrive/Stud/SP/SP_6/ARM$ qemu-aarch64 ./2
Введите размер матрицы(Два целых числа, M x N):
3
3
1
2
3
3
2
1
4
4
4
1 2 3
3 2 1
4 4 4
5
```

Рисунок 10 – результат работы 4 на системе aarch64

5 Вывод

В ходе данной лабораторной работы были разработаны программы для решения простых задач с матрицами на языках ассемблера для ОС GNU/Linux.