Самилык Анастасия ДЗ №5
Отчет (программа на 8 + программа на 10)

Программа на оценку 8.

Файл main с вызовом подпрограмм.

main.asm* | cin.asm | sum.asm | cout_cor.asm | cout_uncor.asm | macrolib.asm

```
1  include "cin.asm"
2  include "sum.asm"
3  include "cout_cor.asm"
4  include "cout_uncor.asm"
5  global main
6
7
8  main:
9          jal cin                 # Вызов подпрограммы для ввода данных
10         jal sum                 # Вызов подпрограммы для подсчета суммы
11         beq s11 zero corr       # Если переполнения не произошло, вызывается подпрограмма для случая без переполнения
12         jal cout_uncor          # Вызов подпрограммы для случая переполнения
13         corr:   jal cout_cor    # Вызов подпрограммы для случая без переполнения
14
15
16
```

Файл cin с вводом данных и вызовом макроопределения read_int из библиотеки с семинара

main.asm | cin.asm | sum.asm | cout_cor.asm | cout_uncor.asm | macrolib.asm

```
cin:
.include "macrolib.asm"
.data
        len: .asciz "Количество элементов в массиве: "
        number_error: .asciz "Ошибка: количество элементов должно быть в диапазоне от 0 до 10"
        arg: .asciz "Введите число: "

.align 2
        array: .space 64
        arrend:
.text
        la a0, len                          # Помещаем строку len в регистр a0
        li a7, 4                            # Системный вызов №4 - вывести null-terminated string
        ecall
        read_int(s0)                        # Чтение целого числа
        blt  s0, zero, uncorrect_number     # Если число, введеннное пользователем, меньше нуля - переход к uncorrect_number
        addi, s10, s10, 10                  # Помещаем в регистр s10 число 10
        bgt s0, s10, uncorrect_number       # Если число, введеннное пользователем, больше десяти - переход к uncorrect_number
        j correct_number                    # Переход к uncorrect_number

uncorrect_number:
        li a7, 4                            # Системный вызов №4 - вывести null-terminated string
        la a0, number_error                 # Помещаем строку number_error в регистр a0
        ecall                               # Выводим текст ошибки
        li a0, 0                            # exit code
        li a7, 10                           # syscall exit
        ecall

correct_number:
        la t0 array                             # Счетчик
        la s1 arrend
        while:  la a0, arg                      # Помещаем в регистр a0 строку arg
                li a7, 4                        # Системный вызов №4 - вывести null-terminated string
                ecall
                read_int(a0)                    # Чтение целого числа
                sw a0 (t0)                      # Запись введенного числа по адресу t0
                addi t0 t0 4                    # Увеличение адреса на размер слова в байтах

                addi s2, s2, 1                  # Увеличение счетчика, отвечающего за количество введенных элементов
                blt s2, s0, while               # Если счетчик меньше числа элементов, запускаем тело цикла еще раз
        la t0 array
        mv s2, zero

ret
```

Файл sum с суммирование элементов

main.asm | cin.asm | sum.asm | cout_cor.asm | cout_uncor.asm | macrolib.asm

```
1  sum:
2  .data
3          odd: .asciz "Количество нечетных элементов массива: "
4          even: .asciz "Количество четных элементов массива: "
5  .text
6          sum_array:
7                  lw a0 (t0)                    # Загружаем в a0 значение по адресу t0
8                  mv s8, s3                     # Запишем в регистр s8 промежуточное значение суммы
9                  add s3, s3, a0                # Добавим к счетчику суммы текуший элемент
10                 blt s8, zero, other           # Проверки на переполнение
11                 bgt s3, zero, ok
12                 blt a0, zero, ok
13                 j error                       # Если сумма была больше нуля, стала меньше нуля, а текуший элемент > 0 => произошло переполнение
14                 other:  blt s3, zero, ok
15                         bgt a0, zero, ok      # Если сумма была меньше нуля, стала больше нуля, а текуший элемент < 0 => произошло переполнение
16                 error:  addi s11, s11, 1      # Положим в регистр s11 число 1, чтобы позднее понять, что была ошибка
17                         ret
18
19                 ok:
20                 addi t0, t0, 4                # Увеличение адреса на размер слова в байтах
21                 addi s2, s2, 1                # Увеличение счетчика, отвечающего за количество введенных элементов
22                 blt s2, s0, sum_array         # Если счетчик меньше числа элементов, запускаем тело цикла еще раз
23
24 ret
25
```

Файлы с выводом элементов для случая с переполнением и без + использование макроопределения read_int из библиотеки с семинара

```
.macro print_int (%x)
        li a7, 1
        mv a0, %x
        ecall
.end_macro
cout_cor:

.data
        ans_sum: .asciz "Сумма элементов массива: "
        numb_el: .asciz "\nКоличество просуммированных элементов: "
.text

        la a0, ans_sum                          # Помещаем в регистр a0 строку ans_sum
        li a7, 4                                # Системный вызов №4 - вывести null-terminated string
        ecall
        print_int (s3)                          # Вывод целого числа из регистра s3

        la a0, numb_el                          # Помещаем в регистр a0 строку numb_el
        li a7, 4                                # Системный вызов №4 - вывести null-terminated string
        ecall

        print_int (s2)                          # Вывод целого числа из регистра s2
        ecall
        li a7 10                                # Системный вызов №10 — остановка программы
        ecall
```

```
1  .macro print_int (%x)
2          li a7, 1
3          mv a0, %x
4          ecall
5  .end_macro
6  cout_uncor:
7  .data
8          overflow: .asciz "Произошло переполнение!"
9          correct_sum: .asciz "\nСумма до переполнения: "
10         el_number: .asciz "\nКоличество просуммированных элементов: "
11 .text
12         la a0, overflow         # Помещаем в регистр a0 строку overflow
13         li a7, 4                # Системный вызов №4 - вывести null-terminated string
14         ecall
15         la a0, correct_sum      # Помещаем в регистр a0 строку correct_sum
16         ecall
17         print_int (s8)          # Вывод целого числа из регистра s8
18         la a0, el_number        # Помещаем в регистр a0 строку el_number
19         li a7, 4                # Системный вызов №4 - вывести null-terminated string
20         ecall
21         print_int (s2)          # Вывод целого числа из регистра s2
22         li a7 10                # Системный вызов №10 — остановка программы
23         ecall
```

Программа на оценку 10.

В решении 2 файла: один - main, второй - библиотека макроопределений

main.asm | lib.asm

```
.include "lib.asm"
.global main


main:
        cin                     # Вызов макроопределения для ввода данных
        sum                     # Вызов макроопределения для подсчета суммы
        beq sll zero corr       # Если переполнения не произошло, вызывается макроопределения для случая без переполнения
        cout_uncor              # Вызов макроопределения для случая переполнения
        j exit_code
        corr:   cout_cor        # Вызов макроопределения для случая без переполнения
        exit_code: exit

```

lib.asm - файл с такими макросами, как:

cin	- ввод данных в массив
read_int(%x)	- ввод целого числа
sum	- суммирование элементов
print_int (%x)	- вывод целого числа
cout_cor	- вывод для слуяая без переполнения
cout_uncor	- вывод для случая с переполнением
exit	- завершение программы

main.asm | lib.asm

```
.macro cin
        .data
                len: .asciz "Количество элементов в массиве: "
                number_error: .asciz "Ошибка: количество элементов должно быть в диапазоне от 0 до 10"
                arg: .asciz "Введите число: "

        .align 2
                array: .space 64
                arrend:
        .text
                la a0, len                              # Помещаем строку len в регистр a0
                li a7, 4                                # Системный вызов №4 - вывести null-terminated string
                ecall
                read_int(s0)                            # Чтение целого числа
                blt  s0, zero, uncorrect_number         # Если число, введенное пользователем, меньше нуля - переход к uncorrect_number
                addi, s10, s10, 10                      # Помещаем в регистр s10 число 10
                bgt s0, s10, uncorrect_number           # Если число, введенное пользователем, больше десяти - переход к uncorrect_number
                j correct_number                        # Переход к uncorrect_number

        uncorrect_number:
                li a7, 4                                # Системный вызов №4 - вывести null-terminated string
                la a0, number_error                     # Помещаем строку number_error в регистр a0
                ecall                                   # Выводим текст ошибки
                li a0, 0                                # exit code
                li a7, 10                               # syscall exit
                ecall

        correct_number:
                la t0 array                             # Счетчик
                la s1 arrend
                while:  la a0, arg                      # Помещаем в регистр a0 строку arg
                        li a7, 4                        # Системный вызов №4 - вывести null-terminated string
                        ecall
                        read_int(a0)                    # Чтение целого числа
                        sw a0 (t0)                      # Запись введенного числа по адресу t0
                        addi t0 t0 4                    # Увеличение адреса на размер слова в байтах

                        addi s2, s2, 1                  # Увеличение счетчика, отвечающего за количество введенных элементов
                        blt s2, s0, while               # Если счетчик меньше числа элементов, запускаем тело цикла еще раз
                la t0 array
                mv s2, zero

.end_macro
```

```
.macro read_int(%x)
   li a7, 5
   ecall
   mv %x, a0
.end_macro

.macro sum
        .data
                odd: .asciz "Количество нечетных элементов массива: "
                even: .asciz "Количество четных элементов массива: "
        .text
                sum_array:
                        lw a0 (t0)                      # Загружаем в a0 значение по адресу t0
                        mv s8, s3                       # Запишем в регистр s8 промежуточное значение суммы
                        add s3, s3, a0                  # Добавим к счетчику суммы текущий элемент
                        blt s8, zero, other             # Проверки на переполнение
                        bgt s3, zero, ok
                        blt a0, zero, ok
                        j error                         # Если сумма была больше нуля, стала меньше нуля, а текущий элемент > 0 => произошло переполнение
                        other:  blt s3, zero, ok
                                bgt a0, zero, ok        # Если сумма была меньше нуля, стала больше нуля, а текущий элемент < 0 => произошло переполнение
                        error:  addi s11, s11, 1        # Положим в регистр s11 число 1, чтобы позднее понять, что была ошибка
                                j end

                        ok:
                        addi t0, t0, 4                  # Увеличение адреса на размер слова в байтах
                        addi s2, s2, 1                  # Увеличение счетчика, отвечающего за количество введенных элементов
                        blt s2, s0, sum_array           # Если счетчик меньше числа элементов, запускаем тело цикла еще раз
                        end:

.end_macro


.macro print_int ( %x)
        li a7, 1
        mv a0, %x
        ecall
.end_macro
```

```
.macro cout_cor

        .data
                ans_sum:  .asciz "Сумма элементов массива: "
                numb_el:  .asciz "\nКоличество просуммированных элементов: "
        .text

                la a0, ans_sum                          # Помещаем в регистр a0 строку ans_sum
                li a7, 4                                # Системный вызов №4 - вывести null-terminated string
                ecall
                print_int ( s3)                         # Вывод целого числа из регистра s3

                la a0, numb_el                          # Помещаем в регистр a0 строку numb_el
                li a7, 4                                # Системный вызов №4 - вывести null-terminated string
                ecall

                print_int ( s2)                         # Вывод целого числа из регистра s2

.end_macro


.macro cout_uncor
        .data
                overflow:  .asciz "Произошло переполнение!"
                correct_sum:  .asciz "\nСумма до переполнения: "
                el_number:  .asciz "\nКоличество просуммированных элементов: "
        .text
                la a0, overflow                         # Помещаем в регистр a0 строку overflow
                li a7, 4                                # Системный вызов №4 - вывести null-terminated string
                ecall
                la a0, correct_sum                      # Помещаем в регистр a0 строку correct_sum
                ecall
                print_int ( s8)                         # Вывод целого числа из регистра s8
                la a0, el_number                        # Помещаем в регистр a0 строку el_number
                li a7, 4                                # Системный вызов №4 - вывести null-terminated string
                ecall
                print_int ( s2)                         # Вывод целого числа из регистра s2
.end_macro

.macro exit
                li a7 10                                # Системный вызов №10 — остановка программы
                ecall
.end_macro
```