Архитектура вычислительных систем.
ИДЗ 1. Отчёт
Вариант 8
Работа на 10 баллов

Фролов-Буканов Виктор Дмитриевич БПИ-228

14 октября 2023

1 Условие задачи

Сформировать массив B по следующим правилам:

1. если $A_i > 5$, то увеличить элемент на 5,
2. если $A_i < -5$, то уменьшить на 5,
3. остальное обнулить.

2 Тесты, демонстрирующие проверку разработанных программ

Программа с ассемблерным кодом будет проходить 10 тестов, покрывающих всё возможное тестовое покрытие. Содержание 10 тестовых файлов:

1. 5 5 -5 -9 10 -105
2. 5 1 2 -9 0 43
3. 0
4. 5 1 2 3 4 5
5. 3 -76 105 24
6. 11
7. 10 5 -5 -6 9 11 -71 34 43 -12 0
8. 7 -2 4 44 -632 -123 176 11
9. 5 6 7 8 9 10
10. 5 -6 -7 -8 -9 -10

Первое число в тесте - размер вводимого массива (по условию он должен быть в отрезке [1, 10]). Далее, если размер валидный, то следуют элементы массива. Если же размер не валидный, то никаких элементов массива не вводится, так как программа закончит свою работу в соответствии с условием. Данные тесты лежает соответственно в файлах **test_case1.txt ... test_case10.txt**

3 Результаты тестовых прогонов для различных исходных данных

Для тестирования программы я разработал программу на языке C++ (**main.cpp**), запускающую ассемблерный код из файла **main.asm** для данных из каждого тестового файла **test_case1.txt ... test_case10.txt**. Вот код программы:

main.cpp

```
#include <fstream>
#include <iostream>
#include <string>
#include <vector>

const std::string rarsExecutablePath = R"(C:\Users\frolo\rars1_6.jar)";
const std::string assemblyFilePath = R"( D:\asm\ihw1\main.asm)";

int main() {
  std::vector<std::string> vec;

  for (std::size_t i = 0; i < 10; ++i) {
    std::string str = "java -jar " + rarsExecutablePath
        + assemblyFilePath + " < " + R"(D:\asm\ihw1\test_case)"
        + std::to_string(i + 1) + ".txt";
    vec.push_back(str);
  }

  for (std::size_t i = 0; i < vec.size(); ++i) {
    std::string testFilePath = R"(D:\asm\ihw1\test_case)"
        + std::to_string(i + 1) + ".txt";
    std::ifstream fin(testFilePath);
    std::cout << i + 1 << ". File content: ";
    while (!fin.eof()) {
      int num;
      fin >> num;
      std::cout << num << " ";
    }
    std::cout << '\n';
    const char *command = vec[i].c_str();
    system(command);
    std::cout << "\n----------\n";
  }
  return 0;
}
```

Пути к файлам указаны такие, какие они на моем компьютере, поэтому в случае использования этого кода на других машинах, в соответствующих переменных следует указывать пути, которые соответствуют расположению файлов на используемой машине

Результаты тестирования прикреплены ниже в виде скриншотов:

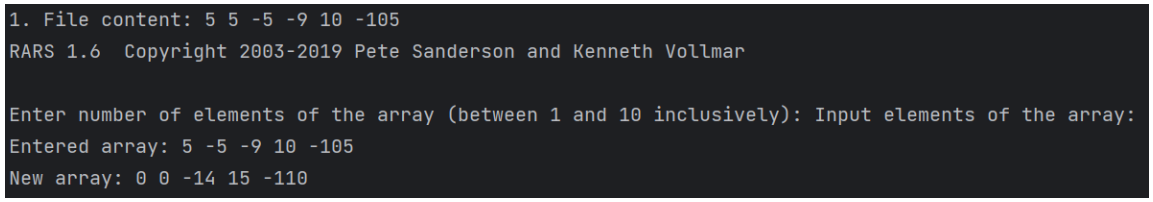

Figure 1: 1 скрин

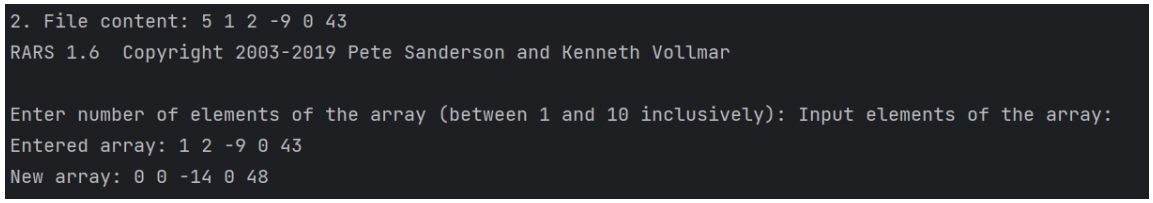

Figure 2: 2 скрин

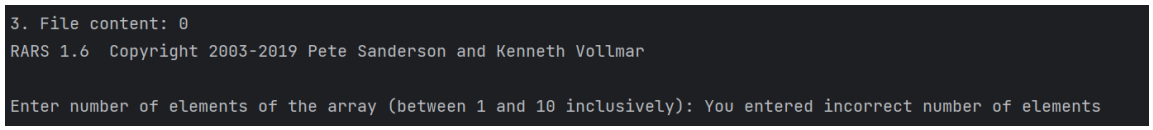

Figure 3: 3 скрин

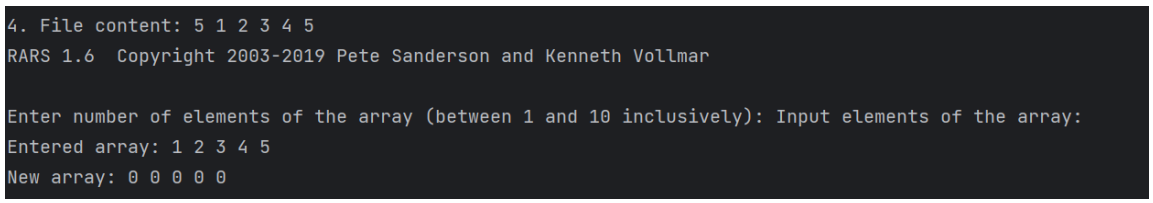

Figure 4: 4 скрин

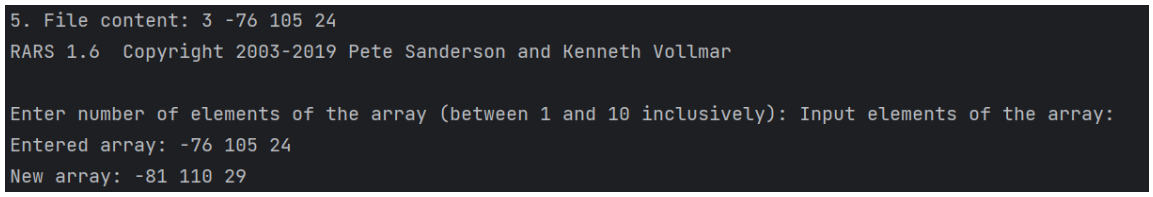

Figure 5: 5 скрин

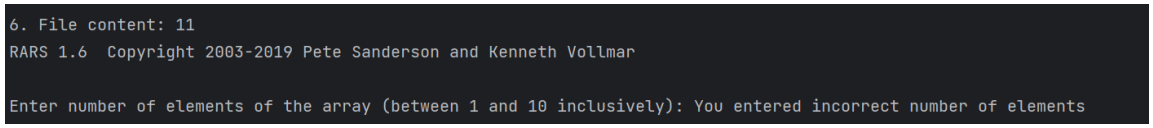

Figure 6: 6 скрин

```
7. File content: 10 5 -5 -6 9 11 -71 34 43 -12 0 0
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

Enter number of elements of the array (between 1 and 10 inclusively): Input elements of the array:
Entered array: 5 -5 -6 9 11 -71 34 43 -12 0
New array: 0 0 -11 14 16 -76 39 48 -17 0
```

Figure 7: 7 скрин

```
8. File content: 7 -2 4 44 -632 -123 176 11
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

Enter number of elements of the array (between 1 and 10 inclusively): Input elements of the array:
Entered array: -2 4 44 -632 -123 176 11
New array: 0 0 49 -637 -128 181 16
```

Figure 8: 8 скрин

```
9. File content: 5 6 7 8 9 10
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

Enter number of elements of the array (between 1 and 10 inclusively): Input elements of the array:
Entered array: 6 7 8 9 10
New array: 11 12 13 14 15
```

Figure 9: 9 скрин

```
10. File content: 5 -6 -7 -8 -9 -10
RARS 1.6  Copyright 2003-2019 Pete Sanderson and Kenneth Vollmar

Enter number of elements of the array (between 1 and 10 inclusively): Input elements of the array:
Entered array: -6 -7 -8 -9 -10
New array: -11 -12 -13 -14 -15
```

Figure 10: 10 скрин

4 Тексты программы на языке ассемблера

Всего моя программа состоит из двух файлов: макробиблиотеки **macrolib.asm** и основной программы **main.asm**, в которой эта макробиблиотека подключается через директиву **.include**. Код основной программы **main.asm**:

main.asm

```
.include "macrolib.asm"
.data
        .align 2
        arr_a: .space 40
        arr_b: .space 40
.text
main:
        print_str("Enter␣number␣of␣elements␣of␣the␣array␣(between␣1␣and
           ␣10␣inclusively):␣")
        read_int_in(s1) # s1 stores the size of the array
        if_less(s1, 1, jump_error)
        if_greater(s1, 10, jump_error)
        print_str("Input␣elements␣of␣the␣array:\n")
        read_array(arr_a, s1)
        print_str("Entered␣array:␣")
        print_array(arr_a, s1)
        print_char('\n')
        print_str("New␣array:␣")
        build_array_b(arr_a, arr_b, s1)
        print_array(arr_b, s1)
        print_char('\n')
        exit
error:
        print_str("You␣entered␣incorrect␣number␣of␣elements")
        exit
```

Код макробиблиотеки **macrolib.asm**:

macrolib.asm

```
# Read array by the label %label with size equal to %size_reg
.macro read_array(%label, %size_reg)
        push(t0)
        push(s0)
        la t0 %label # t0 contains array
        li s0 0 # counter
while:
        bge s0 %size_reg break # if counter >= size of the array ->
           break
        read_int_in(a1)
        sw a1(t0)
        addi t0 t0 4
        addi s0 s0 1
        j while
break:
        pop(t0)
        pop(s0)
.end_macro

# Print array of the size% size_reg by the label %label
.macro print_array(%label, %size_reg)
        push(t0)
        push(s0)
        la t0 %label # t0 contains array
        li s0 0 # counter
while:
```

```
        bge s0 %size_reg break # if counter >= size of the array ->
            break
        lw a0(t0)
        print_int(a0)
        print_char('␣')
        addi s0 s0 1
        addi t0 t0 4
        j while
break:
        pop(t0)
        pop(s0)
.end_macro

# Read an integer from the keyboard in the register %dest apart from a0
    register
.macro read_int_in(%dest)
        push(a0)
        li a7 5
        ecall
        mv %dest a0
        pop(a0)
.end_macro

# Read an integer from the keyboard in the register a0
.macro read_int_in_a0
        li a7 5
        ecall
.end_macro

# Print the integer from the %int_reg register
.macro print_int(%int_reg)
        push(a0)
        li a7 1
        mv a0 %int_reg
        ecall
        pop(a0)
.end_macro

# Print the string, immediately transferred in the macro (string is
    transferred immediately, not by the label or register)
.macro print_str(%str)
.data
        str: .asciz %str
.text
        push(a0)
        la a0 str
        li a7 4
        ecall
        pop(a0)
.end_macro

# Print the character, immediately transferred in the macro (character
    is transferred immediately, not by the label or register)
.macro print_char(%char_imm)
        push(a0)
```

```
        li a7 11
        li a0 %char_imm
        ecall
        pop(a0)
.end_macro

# Analogy of the if-statement in higher level languages (the logic is
    written in the comment below)
.macro if_less(%first_reg, %second_imm, %macro_name) # if (first <
    second) { macro_name() }
        push(a7)
        li a7 %second_imm
        bge %first_reg a7 break # if first >= second -> break (
            condition does not hold)
        %macro_name()
break:
        pop(a7)
.end_macro

# Analogy of the if-statement in higher level languages (the logic is
    written in the comment below)
.macro if_greater(%first_reg, %second_imm, %macro_name) # if (first >
    second) { macro_name() }
        push(a7)
        li a7 %second_imm
        ble %first_reg a7 break  # if first <= second -> break (
            condition does not hold)
        %macro_name()
break:
        pop(a7)
.end_macro

# Build the array b using array a according to the rule from the
    variant 8
.macro build_array_b(%label_a, %label_b, %size_reg)
        push(t0)
        push(t1)
        push(s0)
        la t0  %label_a # t0 contains array a
        la t1 %label_b # t1 contains array b
        li s0 0 # counter
while:
        bge s0, %size_reg, break # if counter >= size of the array ->
            break
        lw a0(t0)
        rule
        sw a1(t1)
        addi s0 s0 1
        addi t1 t1 4
        addi t0 t0 4
        j while
break:
        pop(t0)
        pop(t1)
        pop(s0)
```

```
.end_macro

# The rule which is used to from the array b (see variant 8)
.macro rule #  0  contains the current element of the array a
        push(s2)
        li s2 0 # indicates whether we add 5 or subtract 5(1 if yes, 0
            if no)
        if_less(a0, -5, sub5)
        if_greater(a0, 5, add5)
        if_less(s2, 1, annul) # annul only in the case if we did not
            add or subtract 5. Then s2 == 0 (or s2 < 1)
        pop(s2)
.end_macro

# Add 5 to the number stored in a1 register. This is part of the
    realization of the rule that is used to change array a to build
    array b. a1 will store the number which will be added to array b
.macro add5
        li s2 1 # indicates whether we add 5 or subtract 5(1 if yes, 0
            if no)
        mv a1 a0
        addi a1 a1 5
.end_macro

# Subtract 5 from the number stored in a1 register. This is part of the
     realization of the rule that is used to change array a to build
    array b. a1 will store the number which will be added to array b
.macro sub5
        li s2 1 # indicates whether we add 5 or subtract 5(1 if yes, 0
            if no)
        mv a1 a0
        addi a1 a1  -5
.end_macro

# Annul a1 register. This is part of the realization of the rule that
    is used to change array a to build array b. a1 will store the number
     which will be added to array b
.macro annul
        li a1 0
.end_macro

# Macro for the correct processing of the corectness of the entered
    size
.macro jump_error
        j error
.end_macro

# Saving given register on the stack
.macro push(%x)
        addi    sp sp -4
        sw      %x (sp)
.end_macro

# Popping value from the top of the stack to the given register
.macro pop(%x)
```

```
        lw      %x (sp)
        addi    sp sp 4
.end_macro

# Finishing of the program
.macro exit
        li a7 10
        ecall
.end_macro
```

Код вставился несколько криво, так как съехали комментарии, но всё же его можно прочитать без пробелм

5 Дополнительная информация, подтверждающая выполнение задания в соответствие требованиям на предполагаемую оценку

1. Все комментарии, а также текстовые сообщения, выводимые пользователю в качестве подсказок, написаны на английском языке во избежании конфликтов с кодировками (LaTeX не позволял импортировать код), а также для соответствия предоставленным требованиям

2. В каждом макросе, где уместно, используется сохранение локальных переменных на стек с использованием макросов **push** и **pop** из макробиблиотеки с семинаров. Хоть нам и хватает регистров для корректной работы программы, но чтобы выполнить требование на 6-7 баллов, я добавил сохранение регистров на стек в качестве локальных переменных

3. Так как требование реализовать макросы для ввода и вывода массива (и не только их) противоречит требованию реализовать подпрограммы для тех же нужд, то я выбрал реализацию через макросы, так как это требование на более высокую оценку

4. Перед каждым макросом написан комментарий, поясняющий его функциональность. Неочевидные шаги в ходе реализации макросов я также помечал комментариями. В основной программе я не оставлял комментарии около вызова макросов, так как по названию вполне понятно, за что отвечает каждый их них

5. Макросы **if_less** и **if_greater** реализованы для удобства реализации основной программы (проверки на корректность введенного размера массива), а также для удобства реализации макроса **rule**, который используется в построении массива b

6. Учитывая замечание от самих авторов RARS относительно их макроассемблера ("Макросредства ассемблера не входят ни в какой стандарт и остаются на усмотрение авторов ассемблера"), я не придерживался каких-то строгих конвенций при реализации макросов. Единственное, чтобы не запутаться в ходе реализации макроса, передаю я регистр, непосредственно значение, метку или название другого макроса, я использовал следующий подход:

- если передаётся регистр, то я сопровождал его постфиксом _reg
- если передаётся непосредственное значение, то я сопровождал его постфиксом _imm
- если передаётся метка, то я сопровождал её префиксом label_
- если передаётся имя другого макроса, то я использовал %macro_name

Такой подход упрощает чтение сигнатуры макроса и защищает от ошибок в реализации

7. Реализованные макросы поддерживают многократное использование с различными наборами исходных данных, включая возможность подключения различных исходных и результирующих массивов (имеется в виду подключение из тестовых файлов). Каждый макрос не привязан к конкретному регистру или метке, что обеспечивает унификацию, то есть возможность многократного использования одних и тех же конструкций. Так, макросы с чтением целочисленного числа в регистр, или выводом строки, символа, числа, массива, а также многие другие используются многократно в программе, что и подтверждает разбиение программы на **унифицированные модули** (один из критериев на 10 баллов). Отмечу, что хоть макрос чтения массива в метку и используется единожды, но он является унифицированным модулем в силу своей реализации

8. Автоматизированное тестирование программы реализовано через код на C++, а тестовые данные поступают из файлов **test_case1.txt . . . test_case10.txt**, что соответствует выполнению соответвующего критерия (данные не вводятся вручную). Данные тесты составляют полный набор тестового покрытия, и программа выдаёт на них корректный ответ, что позволяет утвердить о корректности реализации ассемблерной программы