МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №6

по дисциплине «Организация ЭВМ и систем»

ТЕМА: Организация связи Ассемблера с ЯВУ на примере программы построения частотного распределения попаданий псевдослучайных целых чисел в заданные интервалы.

Студент гр. 9382 _____ Бочаров Г.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить работу программы на ЯВУ с ассеблером, написать ассемблерный модуль.

Задание:

На языке высокого уровня (Pascal или C) генерируется массив псевдослучайных целых чисел, изменяющихся в заданном диапазоне и имеющих равномерное распределение. Необходимые датчики псевдослучайных чисел находятся в каталоге Tasks\RAND_GEN (при его отсутствии программу датчика получить у преподавателя).

Далее должен вызываться ассемблерный модуль(модули) для формирования распределения количества попаданий псевдослучайных целых чисел в заданные интервалы. В общем случае интервалы разбиения диапазона изменения псевдослучайных чисел могут иметь различную длину.

Результирующий массив частотного распределения чисел по интервалам, сформированный на ассемблерном уровне, возвращается в программу, реализованную на ЯВУ, и затем сохраняется в файле и выводится на экран средствами ЯВУ.

Исходные данные.

1. Длина массива псевдослучайных целых чисел - NumRanDat (<= 16K, K=1024)

2. Диапазон изменения массива псевдослучайных целых чисел [Xmin, Xmax] , значения могут быть биполярные;

3. Количество интервалов, на которые разбивается диапазон изменения массива псевдослучайных целых чисел - NInt (<=24)

4. Массив левых границ интервалов разбиения LGrInt (должны принадлежать интервалу [Xmin, Xmax]).

Результаты:

1. Текстовый файл, строка которого содержит:

- номер интервала,

- левую границу интервала,

- количество псевдослучайных чисел, попавших в интервал.

Количество строк равно числу интервалов разбиения.

2. График, отражающий распределение чисел по интервалам. (необязательный результат)

В зависимости от номера бригады формирование частотного распределения должно производиться по одному из двух вариантов:

1. Для бригад с нечетным номером: подпрограмма формирования распределения количества попаданий псевдослучайных целых чисел в заданные интервалы реализуется в виде одного ассемблерного модуля, сразу формирующего требуемое

распределение и возвращающего его в головную программу, написанную на ЯВУ;

Ход работы:

В качестве ЯВУ используется C++. Считываются необходимые данные: верхний и нижний порог распределения, кол-во разделителей и массив разделителей. Генерируется массив случайных чисел. После чего вызывается функция, описанная в ассемблерном модуле, котороя пробегает по массиву чисел и определляет к какому промежутку пренадлежит каждое число, формируя массив кол-ва попаданий чисел в заданные промежутки.

Тестирование.

Ввод данных	Вывод данных
Введите размер массива 5 Введите нижний предел 1 Введите верхний предел 7 Сгенерированный массив : 2 5 4 2 6	Результат : Промежуток : [1,3) --> Попаданий : 2 Промежуток : [3,7) --> Попаданий : 3

Введите кол-во разделителей 1 Введите массив разделителей 3	
Введите размер массива 10 Введите нижний предел -5 Введите верхний предел 15 Сгенерированный массив : -2 1 12 10 8 10 1 7 4 -4 Введите кол-во разделителей 3 Введите массив разделителей 3	Результат : Промежуток : [-5,3) --> Попаданий : 4 Промежуток : [3,8) --> Попаданий : 2 Промежуток : [8,14) --> Попаданий : 4 Промежуток : [14,15) --> Попаданий : 0

8 14	

Выводы.

В результате выполнения лабораторной работы был разработан код, состоящий из ассемблерного модуля и остальной части на ЯВУ C++, который выводит частоту попадания псевдо-рандомных чисел в определенные диапозоны.

Приложение.

Текст файла main.cpp

```
#include <iostream>
#include <fstream>
#include <ctime>
#include <random>

int64_t getRandomNum(int64_t min, int64_t max) {
    return min + rand() % (max - min + 1);
}

void fillArray(int64_t size, int64_t *arr, int64_t min, int64_t max) {
    for (int64_t i = 0; i < size; i++)
        arr[i] = getRandomNum(min, max);
}

void initArray(int64_t size, int64_t *arr) {
    for (int64_t i = 0; i < size; i++)
        arr[i] = 0;
}

void readArray(int64_t size, int64_t *arr) {


    for (int i = 1; i < size; i++) {
        std::cin >> arr[i];
        if (arr[i] <= arr[i - 1]) {
            std::cout << "Неверный промежуток --> [" << arr[i - 1] << "," <<
arr[i] << ")" << std::endl;
            std::cout << "Введите заново" << std::endl;
            i--;
        }

        if (arr[i] >= arr[size]) {
            std::cout << "Левая граница больше максимума" << std::endl;
            std::cout << "Введите заново" << std::endl;
            i--;
        }
    }
}
```

```
void printArray(int64_t size, int64_t *arr) {
    for (int64_t i = 0; i < size; i++)
        std::cout << arr[i] << " ";

    std::cout << std::endl;
}

void printBordersAndCounts(int64_t size, int64_t *arr, int64_t *res) {
    int64_t i = 0;
    for (i = 0; i < size - 1; i++) {
        std::cout << "Промежуток : [" << arr[i] << "," << arr[i + 1] << ") --
> Попаданий : " << res[i];
        std::cout << std::endl;
    }
    std::cout << "Промежуток : [" << arr[i] << "," << arr[i + 1] - 1 << "] --
> Попаданий : " << res[i];
    std::cout << std::endl;

}

void printToFile(int64_t size, int64_t *arr, int64_t *res) {


    std::ofstream res_file("res.txt");

    for (int64_t i = 0; i < size; i++) {
        res_file << "Промежуток : [" << arr[i] << "," << arr[i + 1] << ") -->
Попаданий : " << res[i];
        res_file << std::endl;
    }

}

extern "C" void count_int(int64_t *array, int64_t array_size, int64_t *bor-
ders, int64_t borders_size, int64_t *res);

int main() {
    int64_t arr_size = 0;
    int64_t borders_size = 0;
    int64_t Left_lim = 0;
    int64_t Rihgt_lim = 0;

    std::cout << "Введите размер массива" << std::endl;
    std::cin >> arr_size;

    std::cout << "Введите нижний предел" << std::endl;
    std::cin >> Left_lim;

    std::cout << "Введите верхний предел" << std::endl;
    std::cin >> Rihgt_lim;


    int64_t array[arr_size];
    fillArray(arr_size, array, Left_lim, Rihgt_lim);

    std::cout << "Сгенерированный массив : ";
    printArray(arr_size, array);


    std::cout << "Введите кол-во разделителей" << std::endl;
    std::cout << "Максимальное число разделителей для промежутка [" <<
Left_lim << "," << Rihgt_lim << "]" << ":->"
              << Rihgt_lim - Left_lim << std::endl;
```

```
    int k = Rihgt_lim - Left_lim;
    borders_size = k+1;

    while (borders_size > k) {
        std::cin >> borders_size;
        if(borders_size>k)
            std::cout<<"Кол-во прмежутков польше максимального"<<std::endl;
    }

    borders_size++;

    int64_t borders[borders_size + 1];
    initArray(borders_size, borders);

    std::cout << "Введите массив разделителей" << std::endl;

    ////////////////////////////
    borders[0] = Left_lim;

    //borders[0] = Left_lim-1;



    borders[borders_size] = Rihgt_lim + 1;

    readArray(borders_size, borders);


    //borders[borders_size-1]++;

    int64_t res[borders_size];
    initArray(borders_size, res);

    //std::cout << "size=" << sizeof(int64_t) << std::endl;
    count_int(array, arr_size, borders, borders_size, res);


    std::cout << "Результат : " << std::endl;
    printBordersAndCounts(borders_size, borders, res);
    printToFile(borders_size, borders, res);

}
```

Текст файла 1.s

```
.intel_syntax noprefix
.global count_int
.text

# The first six integer or pointer arguments are passed in registers RDI,
RSI, RDX, RCX, R8, R9

# rdi -> array
# rsi -> array_size
# rdx -> borders
# rcx -> borders_size
# r8 -> res
count_int:
        sub rax, rax            # счетчик для пробега по array
```

```
        sub rbx, rbx            # счетчик для borders
        sub r9, r9
    get_num:
        mov r9,[rdi]            # получаем число из array

    check_right:
        cmp r9,[rdx+rbx+8]      # проверяем левее ли число от текущей границы

        jl end                  # если число меньше текущей границы переходим
в конец

        add rbx, 8             # если нет, берем следующую границу
        jmp check_right

    end:
        mov r9, [r8+rbx]
        inc r9
        mov [r8+rbx], r9     # увеличиваем кол-во чисел попавших в промежуток

        add rdi, 8          # берем адрес следующего элемента из array
        inc rax

        sub rbx, rbx
        cmp rax , rsi      #  проверка на конец array
        jl get_num
ret
```