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 распределение попаданий псевдослучайных целых чисел в заданные интервалы.

Студент гр. 0382 _____ Азаров М.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

Цель работы.

Изучить основные концепции связи между языком Ассемблера и ЯВУ(язык высокого уровня). Создать свою программу, которая создает числовое распределение (обязанность кода реализованном на C++) и считает количество попаданий этих чисел в заданные интервалы ((обязанность кода реализованном на MASM).

Задание.

Вариант 1

На языке C программируется ввод с клавиатуры и контроль исходных данных, а также генерируется массив псевдослучайных целых чисел, изменяющихся в заданном диапазоне и имеющих заданный закон распределения. Необходимые датчики псевдослучайных чисел находятся в каталоге RAND_GEN (при его отсутствии получить у преподавателя). Следует привести числа к целому виду с учетом диапазона изменения.

Далее должны вызываться 1 или 2 ассемблерных процедуры для формирования распределения количества попаданий псевдослучайных целых чисел в заданные интервалы. Ассемблерные процедуры должны вызываться как независимо скомпилированные модули. Передача параметров в процедуру должна выполняться через кадр стека.

Результирующий массив частотного распределения чисел по интервалам, сфор-мированный 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)
2. Диапазон изменения массива псевдослучайных целых чисел

[Xmin, Xmax] (м.б. биполярный, например, [-100, 100])

3. Массив псевдослучайных целых чисел {Xi}.
4. Количество интервалов, на которые разбивается диапазон изменения массива псевдослучайных целых чисел - NInt (<=24)
5. Массив левых границ интервалов разбиения LGrInt .

В общем случае интервалы разбиения диапазона изменения псевдослучайных чисел могут иметь различную длину, левые границы могут задаваться в произвольном порядке и иметь произвольные значения. Если Xmin < LGrInt(1), то часть данных не будет участвовать в формировании распределения. Каждый интервал, кроме последнего, следует интерпретировать как [LGrInt(i), LGrInt(i+1)). Если у последнего интервала правая граница меньше Xmax, то часть данных не будет участвовать в формировании распределения.

Результаты:

Текстовая таблица, строка которой содержит:

- номер интервала,
- левую границу интервала,
- количество псевдослучайных чисел, попавших в интервал.

Количество строк должно быть равно числу интервалов разбиения.

Таблица должна выводиться на экран и сохраняться в файле.

Задание на разработку программы выбирается из таблицы 1 в зависимости от номера студента в группе.

Варианты заданий различаются:

1) видом распределения псевдослучайных чисел: равномерное или нормальное (гаусовское);

2) количеством ассемблерных модулей, формирующих требуемое распределение:

- если указан 1 модуль, то он сразу формирует распределение по заданным интервалам и возвращает его в головную программу, написанную на ЯВУ;
- если указаны 2 модуля, то первый из них формирует распределение исходных чисел по интервалам единичной длины и возвращает его в вызывающую программу на ЯВУ как промежуточный результат (это распределение должно выводится на экран для контроля); затем вызывается второй модуль который по этому промежуточному распределению формирует окончательное распределение псевдослучайных целых чисел по интервалам произвольной длины (с заданными границами).

Это распределение возвращается в головную программу и выдается как основной результат в виде текстового файла.

3) условием – может ли число интервалов быть больше-равно (Nint $\geq$ Dx) или меньше (Nint < Dx) диапазона изменения входных чисел;

4) условием – может ли первая левая граница быть больше Xmin (Lg1 > Xmin) или могут ли какие-то левые границы быть меньше Xmin (Lgi $\leq$ Xmin);

5) условием – может ли правая граница последнего интервала быть больше Xmax (ПГпосл > Xmax) или меньше-равна Xmax (ПГпосл > Xmax) .

Таблица 1. $D_x = X_{max} - X_{min}$; Lg1, Lgi –первая или любая левая граница; ПГпосл – правая граница последнего интервала

№	Вид распредения	Число ассем. процедур	Nint $\geq D_x$	Nint $< D_x$	Lgi $\leq X_{min}$	Lg1 $> X_{min}$	ПГпосл $\leq X_{max}$	ПГпосл $> X_{max}$
1	равном.	1	+	-	-	+	-	+

Выполнение работы.

Программа состоит из двух файлов :

- ***main.cpp*** - часть программы написанной на языке C++. Отвечает за ввод дынных и проверку их корректности, создание массива псевдослучайных чисел и вывод результата.
- ***module.asm*** - часть программы написаной на языке MASM. Подсчитывает количество попаданий псевдослучайных целых чисел в заданные интервалы.

Файл **main.cpp:**

Сначала программа обрабатывает ввод данных и проверяет их корректность :

```cpp
int main() {
    setlocale(LC_ALL, "Russian");
    int n, x_min, x_max, n_gr;
    cout << "Введите нужное кол-во псевдослучайных целых чисел:" << endl;
    cin >> n;
    cout << "\nВведите диапазон псевдослучайных целых чисел Xmin и Xmax, через пробел:" << endl;
    cin >> x_min >> x_max;
    cout << "\nВведите кол-во интервалов  Nint:" << endl;
    cin >> n_gr;
    n_gr++; // границ на 1 больше чем интервалов

    // Nint < D_x
    if ((n_gr - 1) < (x_max - x_min)) {
        cout << "\nНекоректные данные: Nint < D_x" << endl;
        return 0;
    }

    cout << "\nВведите " << n_gr <<" границ интервалов, через пробел:" << endl;
    auto borders = new int[n_gr];
    for (int i = 0; i < n_gr; ++i) {
        cin >> borders[i];
    }

    //  Lg1 <= X_min
    if (borders[0] <= x_min) {
        cout <<"\nНекоректные данные: Lg1 <= X_min" << endl;
        return 0;
    }

    // Last_gr <= X_max
    if (borders[n_gr - 1] <= x_max) {
        cout << "\nНекоректные данные: Last_gr <= X_max" << endl;
        return 0;
    }
```

Затем инициализируемый генератор равномерно распределенных псевдослучайных чисел и создается массив псевдослучайных чисел в заданном диапазоне.

```
//инициализация генератора
std::random_device rd;
std::mt19937 gen(rd());
std::uniform_int_distribution<int>  distr(x_min, x_max);


//генерация чисел
auto numbers = new int[n];
for (int i = 0; i < n; ++i) {
    numbers[i] = distr(gen);

    //показать массив сгенерированных чисел
    //cout << numbers[i] << " ";
    //if ((i + 1) % 50 == 0) cout << "\n";
}
```

После создаем необходимые массивы для работы процедуры ***count_distribution()*** из файла **module.asm** и вызываем её**.** Эта процедура подсчитывает кол-во попаданий в интервалы и записывает результат в массив ***res***. (Более подробно рассмотрим эту процедуру , когда перейдем к описанию файла **module.asm**)

```
//посчет попаданий в интергвалы
int n_units = x_max - x_min + 1;
auto units = new int[n_units];
auto result = new int[n_gr - 1];
for (int i = 0; i < n_units; ++i)
    units[i] = 0;
for (int i = 0; i < n_gr - 1; ++i)
    result[i] = 0;

count_distribution(numbers, n, borders, n_gr, units, n_units, x_min, result);
```

Для того чтобы подключить эту процедуру в main.cpp, прописываем:

```
extern "C" void count_distribution(int* numbers, int n, int* intervals, int n_int, int* units, int n_units, int x_min, int* res);
```

И выводим полученные результаты в нужном формате в консоль и в файл.

```
//вывод
ofstream file("res.txt");
auto name_table = "Таблица частотного распределения чисел по интервалам\n";
auto head = "N\tLg[i]\t    Кол-во попаданий";
file << name_table <<head << endl;
cout << "\n" << name_table << head << endl;
for (int i = 0; i < n_gr - 1; i++) {
    auto row = to_string(i) + "\t" + to_string(borders[i]) + "\t\t   " + to_string(result[i]) + "\n";
    file << row;
    cout << row;
}
```

Файл **module.cpp:**

Содержит реализацию одной единственной функции ***count_distribution(),*** которая подсчитывает кол-во попаданий в заданные интервалы. На вход процедура требует :

```
void count_distribution(int* numbers, int n, int* borders, int n_gr, int* units, int n_units, int x_min, int* res)
```

где :

- ***numbers*** - массив псевдо случайных чисел.
- ***n*** - длина ***numbers***
- ***borders*** - массив границ интервалов.
- ***n_gr*** - длина массива ***borders***
- ***inits*** - массив для хранения кол-ва попаданий в конкретные числа. ***count_distribution()*** требует чтоб длина этого массива была равна диапазону распределения псевдослучайных чисел и чтоб массив был инициализирован нулями.
- ***n_units*** - длина ***units***.

- ***x_min*** - нижняя граница диапазона распределения псевдослучайных чисел.
- ***res*** - массив куда будет записан результат работы процедуры. ***count_distribution()*** требует чтоб длина этого массива была равна кол-ву интервалов и массив должен быть инициализирован нулями.

Как работает ***count_distribution():***

1. Сначала мы рассматриваем каждое число из массиве ***numbers*** минус ***x_min,*** как индекс в массиве ***units.*** И увеличиваем значение по этому индексу на один . В результате массив ***units*** будет содержать количество попаданий случайных чисел в каждое число в диапазоне ***x_min*** и ***x_max.***

```
    mov esi, numbers
    mov edi, units
    mov ecx, n

loop_count_dist:
    mov eax, [esi]
    sub eax, x_min
    mov ebx, [edi + 4*eax]
    add ebx, 1
    mov [edi + 4*eax], ebx
    add esi, 4
    loop loop_count_dist
```

2. Затем для каждого интервала берем элементы массива ***units*** , индексы которых входят в этот интервал (границы интервала уменьшаем на x_min) и суммируем значение этих элементов. Это и будет количество попаданий псевдослучайных чисел в текущий интервал. И записываем это значение в ***res***.

```
    mov esi, borders
    mov edi, res
    sub n_gr, 1
    mov ecx, n_gr
    sub n_units,1 ;делаем из кол-ва эл , max индекс

start_loop:
    mov eax, [esi]
    add esi, 4h
    mov ebx, [esi]
    sub ebx, eax ; смещение от левой границы до правой границы (не включ)  текущего инт.
    sub eax, x_min ; левая граница текущ. инт.

    push ecx
    push esi

    mov ecx, ebx
    mov ebx, 0h ; сумма единичных интервалов в диапазоне [ lg[i], rg[i] )
    mov esi, units

start_loop2:
        cmp eax, n_units
        jg end_loop2

        add ebx, [esi + eax*4]
        add eax, 1
        loop start_loop2
end_loop2:

    mov [edi], ebx
    add edi, 4h

    pop esi
    pop ecx

    loop start_loop
```

Тестирование.

```
Введите нужное кол-во псевдослучайных целых чисел:
1000

Введите диапазон псевдослучайных целых чисел Xmin и Xmax, через пробел:
-5 5

Введите кол-во интервалов  Nint:
10

Введите 11 границ интервалов, через пробел:
-4 -3 -2 -1 0 1 2 3 4 5 6


Таблица частотного распределения чисел по интервалам
N       Lg[i]       Кол-во попаданий
0       -4                  93
1       -3                  79
2       -2                  84
3       -1                  98
4       0                   87
5       1                   96
6       2                   102
7       3                   87
8       4                   99
9       5                   79
```

Программа работает коректно.

Выводы.

Было получено понимание связи между языком Ассемблера и ЯВУ.

В ходе выполнения работы была разработана программа выполняющая поставленное задание, а именно, программа генерирует случайные числа в задаваемом диапазоне и подсчитывает кол-во попаданий чисел в задаваемые интервалы и выводит результат в консоль и в файл. Программа реализована в двух файлах реализованных на разных языках (C++ и MASM) и взаимодействующих по соглашению *конвенции C*.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: **main.asm**

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

using namespace std;

extern "C" void count_distribution(int* numbers, int n, int* borders, int n_gr, int* units, int n_units, int x_min, int* res);

int main() {
	setlocale(LC_ALL, "Russian");
	int n, x_min, x_max, n_gr;
	cout << "Введите нужное кол-во псевдослучайных целых чисел:" << endl;
	cin >> n;
	cout << "\nВведите диапазон псевдослучайных целых чисел Xmin и Xmax, через пробел:" << endl;
	cin >> x_min >> x_max;
	cout << "\nВведите кол-во интервалов  Nint:" << endl;
	cin >> n_gr;
	n_gr++; // границ на 1 больше чем интервалов

	// Nint < D_x
	if ((n_gr - 1) < (x_max - x_min)) {
		cout << "\nНекоректные данные: Nint < D_x" << endl;
		return 0;
	}

	cout << "\nВведите " << n_gr <<" границ интервалов, через пробел:" << endl;
	auto borders = new int[n_gr];
	for (int i = 0; i < n_gr; ++i) {
		cin >> borders[i];
	}

	//  Lg1 <= X_min
	if (borders[0] <= x_min) {
		cout <<"\nНекоректные данные: Lg1 <= X_min" << endl;
		return 0;
	}

	// Last_gr <= X_max
	if (borders[n_gr - 1] <= x_max) {
		cout << "\nНекоректные данные: Last_gr <= X_max" << endl;
		return 0;
	}
```

```
            //инициализация генератора
            std::random_device rd;
            std::mt19937 gen(rd());
            std::uniform_int_distribution<int>  distr(x_min, x_max);


            //генерация чисел
            auto numbers = new int[n];
            for (int i = 0; i < n; ++i) {
                  numbers[i] = distr(gen);

                  //показать массив сгенерированных чисел
                  //cout << numbers[i] << " ";
                  //if ((i + 1) % 50 == 0) cout << "\n";
            }

            cout << endl;

            //посчет попаданий в интергвалы
            int n_units = x_max - x_min + 1;
            auto units = new int[n_units];
            auto result = new int[n_gr - 1];
            for (int i = 0; i < n_units; ++i)
                  units[i] = 0;
            for (int i = 0; i < n_gr - 1; ++i)
                  result[i] = 0;

            count_distribution(numbers,   n,   borders,   n_gr,   units,
n_units, x_min, result);

            /*for (int i = 0; i < n_units; ++i) {

                  cout << units[i] << " ";

            }*/

            //выввод
            ofstream file("res.txt");
            auto name_table = "Таблица частотного распределения чисел по
интервалам\n";
            auto head = "N\tLg[i]\t    Кол-во попаданий";
            file << name_table <<head << endl;
            cout << "\n" << name_table << head << endl;
            for (int i = 0; i < n_gr - 1; i++) {
                  auto row = to_string(i) + "\t" + to_string(borders[i])
+ "\t\t   " + to_string(result[i]) + "\n";
                  file << row;
                  cout << row;
            }

            return 0;
      }
```

Название файла: **module.asm**

```
      .MODEL FLAT, C
      .CODE


      PUBLIC C count_distribution
      count_distribution PROC C   numbers: dword, n: dword, borders:
dword, n_gr: dword, units: dword, n_units: dword, x_min: dword, res:
dword
            mov esi, numbers
            mov edi, units
            mov ecx, n

      loop_count_dist:
            mov eax, [esi]
            sub eax, x_min
            mov ebx, [edi + 4*eax]
            add ebx, 1
            mov [edi + 4*eax], ebx
            add esi, 4
            loop loop_count_dist

      ;-----------------------------------------------

            mov esi, borders
            mov edi, res
            sub n_gr, 1
            mov ecx, n_gr
            sub n_units,1 ;делаем из кол-ва эл , мах индекс

      start_loop:
            mov eax, [esi]
            add esi, 4h
            mov ebx, [esi]
            sub ebx, eax ; смещение от левой границы до правой границы
(не включ)  текущего инт.
            sub eax, x_min ; левая граница текущ. инт.

            push ecx
            push esi

            mov ecx, ebx
            mov  ebx,  0h  ;  сумма  единичных  интервалов  в  диапазоне
[ lg[i], rg[i] )
            mov esi, units

            start_loop2:
                  cmp eax, n_units
                  jg end_loop2

                  add ebx, [esi + eax*4]
                  add eax, 1
                  loop start_loop2
            end_loop2:
```

```
        mov [edi], ebx
        add edi, 4h

        pop esi
        pop ecx

        loop start_loop

ret
count_distribution ENDP
END
```