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. 0382 _____ Сергеев Д.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2021

Цель работы.

Изучение работы ЯВУ со вставками из языка Ассемблера, написание работы построения частотного распределения псевдослучайных целых чисел в заданные интервалы.

Задание.

На языке С 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, сформированный на ассемблерном уровне, возвращается в программу, реализованную на ЯВУ, и затем сохраняется в файле и выводится на экран средствами ЯВУ.

16	нормал.	2	+	-	+	-	+	-

Выполнение работы.

Программа реализована с помощью среды разработки Microsoft Visual Studio 2019, первым делом в файле lab6.cpp объявляются функции, реализованный на языке Ассемблера mod1 и mod2, это делается с помощью ключевого слова extern “C”. Далее происходит ввод данных и проверка их на корректность в соответствии с условиями лабораторной работы. Интервалы сортируются с помощью функции qsort(), далее генерируются рандомные числа в соответствии с нормальным Гауссовским распределением. Далее используется

первая ассемблерная процедура mod1(), она распределяет числа по единичным интервалам.

Процедура mod1 принимает на вход массив чисел numbers, кол-во чисел n, результирующий массив res1, и минимальный элемент xmin, с каждой итерацией цикла из числа, полученного из массива numbers вычитается xmin, таким образом определяется его индекс в массиве res1, и затем в массиве res1 по полученном индексу увеличивается число включений.

Вторая ассемблерная процедура mod2() распределяет числа уже по заданным пользователем интервалам.

Процедура mod2 принимает на вход массив res1, полученный из предыдущий процедуры, массив левых границ интервалов, количество интервалов, массив для результатов res2, а также минимальный элемент xmin. Далее с каждой итерацией цикла берётся число xmin+i, находится его место в интервалах, после чего складывается содержимое массива res2 и res1, и помещается обратно в res2.

Полученный результат записывается в файл result.txt.

Исходный программный код смотрите в приложении А.

Тестирование.

Результаты представлены в таблице 1.

Таблица 1 – Результаты тестирования

№	Входные данные	Выходные данные	Комментарий
1	100 1 10 10 0 1 2 3 4 5 6 7 8 9	Lgi 0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9	Всё правильно
2	50 -3 3 6 -3 -2 -1 0 1 2	Lgi 0 -3 1 -2 2 -1 3 0 4 1 5 2	Всё правильно

Выводы.

В ходе работы была изучена работа со вставками на языке Ассемблера в ЯВУ, а также написана программа в соответствии с заданием.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab6.cpp

```
#include <iostream>
#include <stdio.h>
#include <fstream>
#include <string>
#include <random>

using namespace std;

extern "C" void mod1(int* numbers, int n, int* res1, int xmin);
extern "C" void mod2(int* res1, int* intervals, int nint, int* res2, int
xmin);

int comp(const void* a, const void* b) {
	return (*(int*)a - *(int*)b);
}

int main() {
	int n, xmin, xmax,nint;
	cin >> n;
	cin >> xmin >> xmax;
	cin >> nint;
	if (nint < (abs(xmax - xmin)))
	{
		cout << "Nint<Dx :O" << endl;
		return 0;
	}
	if (nint < 1 || nint>24) {
		cout << "nint <= 24" << endl;
		return 0;
	}
	int* intervals = new int[nint + 1];
	int* res2 = new int[nint];
	for (int i = 0; i < nint; i++) {
		cin >> intervals[i];
		res2[i] = 0;
	}
	intervals[nint] = xmax+1;
	qsort(intervals, nint+1, sizeof(int*), comp);
	if (intervals[0] > xmin) {
		cout << "Lg1>Xmin :O" << endl;
		return 0;
	}
	int* res1 = new int[abs(xmax - xmin) + 1];
	for (int i = 0; i < abs(xmax - xmin) + 1; i++)
	{
		res1[i] = 0;
	}
	int* numbers = new int[n];
	random_device rd;
	mt19937 gen(rd());
	normal_distribution<> conc_gen((xmin + xmax) / 2,abs(xmax - xmin) /
4);
```

```
	for (int i = 0; i < n; i++) {
		numbers[i] = int(conc_gen(gen));
		//cout << numbers[i] << ' ';
	}
	cout << endl;
	mod1(numbers, n, res1, xmin);
	/*for (int i = 0; i < abs(xmax - xmin) + 1; i++) {
		cout << xmin + i << ' ' << res1[i] << endl;
	}*/
	mod2(res1, intervals, nint, res2, xmin);

	ofstream file("result.txt");
	file << "№\t\tLgi\t\t#\n";
	cout << "№\t\tLgi\t\t#\n";
	for (int i = 0; i < nint; i++) {
		file << i << "\t\t" << intervals[i] << "\t\t" << res2[i] <<
endl;
		cout << i << "\t\t" << intervals[i] << "\t\t" << res2[i] <<
endl;
	}

	/*for (int i = 0; i < nint; i++) {
		if (i == nint - 1)
		{
			cout << "[" << intervals[i] << ":" << intervals[i + 1] -
1 << "]: " << res2[i] << endl;
			break;
		}
		cout << "[" << intervals[i] << ":" << intervals[i + 1] << "):
" << res2[i] << endl;
	}*/
	return 0;
}
```

Название файла: mod1.asm

```
.586
.MODEL FLAT, C
.CODE
PUBLIC C mod1
mod1 PROC C numbers:dword, n:dword, res1:dword, xmin:dword


push esi
push edi


mov esi,numbers
mov edi,res1
mov ecx,n


cicle:
```

```
            mov eax,[esi]
            sub eax,xmin
            mov ebx,[edi+4*eax]
            inc ebx
            mov [edi+4*eax],ebx
            add esi,4
            loop cicle

      pop edi
      pop esi

      ret
      mod1 ENDP
END
```

Название файла: mod2.asm

```
      .586
      .MODEL FLAT, C
      .CODE
      PUBLIC C mod2
      mod2 PROC C res1:dword, intervals:dword, nint:dword, res2:dword, xmin:
dword

      push esi
      push edi

      mov esi,res2
      mov edi,intervals
      mov ecx,xmin ; так как у нас уже есть распределение по еденичным
интервалам, то начнём с самого первого
      mov eax,0 ; eax - индекс числа в массиве единичных интервалов res1


      Cicle:
      mov ebx,0 ; индекс текущего интервала в массиве левых границ
интервалов

      Poisk_int:
```

```
    cmp ecx, [edi+4*ebx] ; сравниваем текущее число с левой границей интервала
    jl Obrabotka ; если число меньше текущей границы, то приступаем к обработке

    Incs: ;если число больше текущей левой границы, то двигаемся к следующей границе
          inc ebx
          jmp Poisk_int

    Obrabotka:
          dec ebx ; так как текущее ebx - правая граница интервала, в который входит число, то уменьшаем индекс на 1
          push edi
          push ecx
          mov edi,res1
          mov edx, [edi+4*eax] ; получаем количество встреч данного числа
          mov  ecx,[esi+4*ebx]  ;  получаем  текущее  кол-во  попаданий  в интервал
          add ecx,edx ; суммируем числа из двух предыдущих шагов
          mov [esi+4*ebx],ecx ; обновляем число в результирующем массиве
          pop ecx
          pop edi
          inc eax ; увеличивам индекс в массиве единичных интервалов res1
          inc ecx ; увеличиваем текущее рассматриваемое число
          push eax
          mov eax,nint
          cmp ecx,[edi+eax*4] ; сравниваем текущее рассматриваемое число с xmax
          pop eax
          jl Cicle

    pop edi
    pop esi

    ret
    mod2 ENDP
END
```