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. 9382 _____ Юрьев С.Ю.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

Познакомиться с организацией связи Ассемблера с ЯВУ. Применить на практике на примере программы построения частотного распределения попаданий псевдослучайных чисел в заданные интервалы.

Задание.

Вариант 2.

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 целыхчисел - 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:

2. Для бригад с четным номером: подпрограмма формирования распределения количества попаданий псевдослучайных целых чисел в заданные интервалы реализуется в виде двух ассемблерных модулей, первый из которых формирует распределение исходных чисел по интервалам единичной

длины и возвращает его в вызывающую программу на ЯВУ как промежуточный результат. Это распределение должно выводиться в текстовом виде для контроля. Затем вызывается второй ассемблерный 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 и, возможно, графика.

Выполнение работы.

В ходе работы была разработана программа, которая выполняет построение частотного распределения попаданий псевдослучайных чисел в заданные интервалы.

В основной функции main() (файл main.cpp) происходит считывание данных:

* количество чисел в массиве
* минимальный и максимальный элементы массива
* количество левых границ
* сами левые границы

По этим данным строится массив псевдослучайных чисел. Затем вызывается первый ассемблерный модуль, в котором происходит подсчет частот каждого числа в массиве.

И уже вызов второго модуля возвращает нужные нам частоты чисел в интервалах, которые выводятся на экран.

Исходный код см. в приложении А.

Выводы.

Было произведено знакомство с организацией связи Ассемблера с ЯВУ. Было применено написание программы на практике на примере программы построения частотного распределения попаданий псевдослучайных чисел в заданные интервалы.

Приложение А

Исходный код программы

main.cpp:

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

using namespace std;

extern "C" void mod1(int* numbers, int numbers_size, int* result, int xmin);
extern "C" void mod2(int* array, int array_size, int xmin, int* intervals, int intervals_size, int* result);


int main()
{
setlocale(0, "Russian");
srand(time(NULL));
ofstream result("result.txt");

int numbers_size;
int* numbers;
int xmin, xmax;
int intervals_size;
int* intervals;
int* intervals2;
int* mod1_result;
int* mod2_result;

cout << "Enter count of numbers:\n";
cin >> numbers_size;
if (numbers_size > 16 * 1024)
{
     cout << "Count of numbers must be <= 16*1024\n";
     return 0;
}
cout << "Enter xmin and xmax:\n";
cin >> xmin >> xmax;
cout << "Enter count of borders:\n";
cin >> intervals_size;
if (intervals_size > 24)
{
     cout << "Count of intervals must be <= 24\n";
     return 0;
}

numbers = new int[numbers_size];
```

```
    intervals = new int[intervals_size];
    intervals2 = new int[intervals_size];

    int len_asm_mod1_res = abs(xmax - xmin) + 1;
    mod1_result = new int[len_asm_mod1_res];
    for (int i = 0; i < len_asm_mod1_res; i++)
    {
        mod1_result[i] = 0;
    }

    mod2_result = new int[intervals_size + 1];
    for (int i = 0; i < intervals_size + 1; i++)
    {
        mod2_result[i] = 0;
    }

    cout << "Enter all borders:\n";
    for (int i = 0; i < intervals_size; i++)
    {
        cin >> intervals[i];
        intervals2[i] = intervals[i];
    }

    for (int i = 0; i < numbers_size; i++)
    {
        numbers[i] = xmin + rand() % (xmax - xmin + 1);
    }

    cout << "Randomized values\n";
    result << "Randomized values\n";
    for (int i = 0; i < numbers_size; i++)
    {
        cout << numbers[i] << ' ';
        result << numbers[i] << ' ';
    }
    cout << '\n';
    cout << '\n';
    result << '\n';
    result << '\n';

    cout << "The number of repetitions of each individual
number:\n";
    result << "The number of repetitions of each individual
number:\n";

    mod1(numbers, numbers_size, mod1_result, xmin);


    for (int i = 0; i < len_asm_mod1_res; i++)
    {
```

```
          cout << xmin + i << ": " << mod1_result[i] << "\n";
          result << xmin + i << ": " << mod1_result[i] << "\n";
     }
     cout << '\n';
     result << '\n';

     mod2(mod1_result,     numbers_size,     xmin,     intervals,
intervals_size, mod2_result);


     cout << "Result:\n";
     result << "Result:\n";
     cout << "№\tBorder\tCount of numbers" << endl;
     result << "№\tBorder\tCount of numbers" << endl;

     for (int i = 0; i < intervals_size + 1; i++)
     {
          if (i != intervals_size)
          {
               cout << i + 1 << "\t" << intervals2[i] << '\t' <<
mod2_result[i] << endl;
               result << i + 1 << "\t" << intervals2[i] << '\t' <<
mod2_result[i] << endl;
          }
          else
          {
               cout  <<  i  +  1  <<  "\t"  <<  xmax  <<  '\t'  <<
mod2_result[i] << endl;
               result  <<  i  +  1  <<  "\t"  <<  xmax  <<  '\t'  <<
mod2_result[i] << endl;
          }
     }

     delete[] numbers;
     delete[] intervals;
     delete[] intervals2;
     delete[] mod1_result;
     delete[] mod2_result;

     return 0;
     }
```

mod1.asm:

```
.586p
.MODEL FLAT, C
.CODE
PUBLIC C mod1
mod1  PROC  C  numbers:  dword,  numbers_size:  dword,  result:  dword,
xmin: dword
```

```
push esi
push edi

mov edi, numbers
mov ecx, numbers_size
mov esi, result

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

pop edi
pop esi

ret
mod1 ENDP
END
```

mod2.asm:

```
.586p
.MODEL FLAT, C
.CODE
PUBLIC C mod2
mod2 PROC C array: dword, array_size: dword, xmin: dword, intervals: dword, intervals_size: dword, result: dword

push esi
push edi
push ebp

mov edi, array
mov esi, intervals
mov ecx, intervals_size

index_for_intervals:
        mov eax, [esi]
        sub eax, xmin
        mov [esi], eax
        add esi, 4
        loop index_for_intervals
```

```
mov esi, intervals
mov ecx, intervals_size
mov ebx, 0
mov eax, [esi]

for_loop:
        push ecx
        mov ecx, eax
        push esi
        mov esi, result

   for_array:
                cmp ecx, 0
                je end_for
      mov eax, [edi]
      add [esi + 4*ebx], eax
      add edi, 4
      loop for_array

end_for:
   pop esi
   inc ebx
        mov eax, [esi]
        add esi, 4
        sub eax, [esi]
        neg eax
        pop ecx
        loop for_loop

mov esi, result
mov ecx, intervals_size
mov eax, 0

final_interval:
        add eax, [esi]
        add esi, 4
        loop final_interval

mov esi, result
sub eax, array_size
neg eax

add [esi + 4*ebx], eax

pop ebp
```

```
pop edi
pop esi

ret
mod2 ENDP
END
```