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

Студентка гр. 0383 _____ Петровская Е.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2021

Цель работы.

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, 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 {X i }.

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.

Задание.

Вариант 16:

Вид распределения: нормальный; число процедур = 2; Nint > Dx, Lgi <= Xmin; ПГпосл <= Xmax

Выполнение работы.

Для считывания исходных данных, их проверки соответствия условию, генерации псевдослучайных чисел и вывода результатов работы ассемблерных модулей была написана программа на языке C++.

Случайные числа генерируются в заданном диапозоне и сохраняются в массив x[arr_size]. Введенные левые границы интервалов сохраняются в массив left_border[interval_cnt]. Создан был результирующий массив для обоих внешних модулей.

На ассемблере были написаны 2 модуля. В первом реализовано распределение чисел по единичным отрезкам с помощью комманды loop. В новый массив записывается число повторений каждого из сгенерированных чисел. Во втором модуле те же числа распределяются по заданным пользователем интервалам. С помощью вложенных циклов происходит перебор генерированных чисел и поиск интервала, в который они попадают. В случае если число попадает в интервал, в результирующий массив, хранящий счетчики числа повторений, добавляется +1.

Исходный код программы см. в Приложении А

Таблица 1 – Результаты работы программы lab5

Входные данные	Выходные данные	Комментарий
Чисел: 10 Xmin & Xmax: -1 1 Число интервалов: 5 -10 10 1 2 5	Generated numbers: 1 1 0 0 0 0 0 0 0 0 1st module Interval num Left border Nums in the interval 1 0 8 2 1 2 3 2 0 2nd module Interval num Left border Nums in the interval 1 -10 8 2 1 2 3 2 0 4 5 0 5 10 0	Верно
Чисел: 5 Xmin & Xmax: 3 0 Число интервалов: 5 -20 100 -10 1 2	Generated numbers: 0 0 -2 0 -1 1st module Interval num Left border Nums in the interval 1 0 1 2 1 1 3 2 3 4 3 0 5 4 0 6 5 0 2nd module Interval num Left border Nums in the interval 1 -20 0 2 -10 5	Верно

	3 1 0 4 2 0 5 100 0	
Чисел: 7 Xmin & Xmax: -10 1 Число интервалов: 11 -10 -100 -8 -7 -5 -4 0 1 2 3 4	Generated numbers: -5 -3 -4 -6 -2 -7 -6 1st module Interval num Left border Nums in the interval 1 0 0 2 1 0 3 2 1 4 3 2 5 4 1 6 5 1 7 6 1 8 7 1 9 8 0 10 9 0 11 10 0 12 11 0 2nd module Interval num Left border Nums in the interval 1 -100 0 2 -10 0 3 -8 0 4 -7 3 5 -5 1 6 -4 3 7 0 0 8 1 0 9 2 0	Верно

	10 11	3 4	0 0	

Выводы.

В ходе выполнения лабораторной работы была изучена организация связи Ассемблера с ЯВУ на примере программы построения частотного распределение попаданий псевдослучайных целых чисел в заданные интервалы

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab6.cpp

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

extern "C" void module_1(int* x, int arr_size, int* result, int xmin);
extern "C" void module_2(int* x, int arr_size, int* left_border, int interval_cnt, int* result);

int main() {
	int arr_size = 0;
	int xmin = 0, xmax = 0;
	int interval_cnt = 0;

	std::ofstream output("output.txt");

	std::cout << "Input amount of the random numbers: ";
	std::cin >> arr_size;

	if (arr_size > 16 * 1000) {
		std::cout << "Error - max amount of numbers must be <= 16K" << std::endl;
		return 0;
	}


	int* x = new int[arr_size];

	std::cout << "Input Xmin and Xmax: ";
	std::cin >> xmin >> xmax;

	if (xmax <= xmin) {
		std::cout << "Error - bad range input, Xmax must be > Xmin" << std::endl;
		return 0;
	}

	std::cout << "Input number of intervals: ";
	std::cin >> interval_cnt;

	if (interval_cnt > 24 || interval_cnt < 0) {
		std::cout << "Error - number of intervals must be < 24 and not a negaitve" << std::endl;
		return 0;
	}

	if (interval_cnt < xmax - xmin) {
		std::cout << "Error - number of intervals must be >= Dx" << std::endl;
		return 0;
	}
```

```
            int* left_border = new int[interval_cnt];
            for (int i = 0; i < interval_cnt; i++)
                  std::cin >> left_border[i];

            for (int i = 0; i < interval_cnt - 1; i++) {
                  for (int j = i + 1; j < interval_cnt; j++) {
                        if (left_border[j] < left_border[i]) {
                              std::swap(left_border[j], left_border[i]);
                        }
                  }
            }

            if (left_border[0] > xmin) {
                  std::cout << "Error - 1st left border must be < Xmin" <<
std::endl;
                  return 0;
            }

            std::random_device rng;
            std::mt19937 gen(rng()); //to get certin range calculate mean
and stddev
            float mean = float(xmax + xmin) / 2; //mean
            float stddev = float(xmax - xmin) / 6; //standart deviation
            std::normal_distribution<> d(mean, stddev);


            for (int i = 0; i < arr_size; i++) {
                  x[i] = round(d(rng));
            }

            output << "Generated numbers: ";      //write  generated  nums
for reference into output.txt
            std::cout << "Generated numbers: ";
            for (int i = 0; i < arr_size; i++) {
                  output << x[i] << " ";
                  std::cout << x[i] << " ";
            } output << std::endl; std::cout << std::endl;

            ////////////////////////

            int* m1 = new int[xmax - xmin + 1];
            int* result = new int[interval_cnt];
            for (int i = 0; i < interval_cnt; i++) {
                  result[i] = 0;
            }

            for (int i = 0; i < xmax - xmin + 1; i++) {
                  m1[i] = 0;
            }

            ////////////////////////
            module_1(x, arr_size, m1, xmin);
            //first & second modules
            module_2(x, arr_size, left_border, interval_cnt, result);
            ////////////////////////
```

```
          std::cout << "\t\t\t1st module\nInterval num \tLeft border \
tNums in the interval" << std::endl;
          output << "\t\t\t1st module\nInterval num \tLeft border \tNums
in the interval" << std::endl;
          for (int i = 0; i < xmax - xmin + 1; i++) {
               std::cout << "\t" << (i + 1) << "\t" << i << "\t\t" <<
m1[i] << '\n';
               output << "\t" << (i + 1) << "\t" << i << "\t\t" <<
m1[i] << '\n';
          }

          std::cout << "\t\t\t2nd module\nInterval num \tLeft border \
tNums in the interval" << std::endl;
          output << "\t\t\t2nd module\nInterval num \tLeft border \tNums
in the interval" << std::endl;

          for (int i = 0; i < interval_cnt; i++) {
               std::cout << "\t" << i + 1 << "\t" << left_border[i] <<
"\t\t" << result[i] << '\n';
               output << "\t" << i + 1 << "\t" << left_border[i] << "\
t\t" << result[i] << '\n';
          }

          delete[] result;
          delete[] left_border;
          delete[] x;

          return 0;
     }
```

Название файла: module_1.asm

```
     ;------------1st module

     .586p                                        ;directive   for   correct
translation of 32-bit commands (pentium cpu)
     .MODEL FLAT, C
     .CODE
     PUBLIC C module_1

     module_1  PROC  C  x:  dword,  arr_size:  dword,  result:  dword,  xmin:
dword

     PUSH EDI                                     ;save initial vals
     PUSH ESI
     push ebp

     MOV EDI, x
     MOV ESI, result
     MOV ECX, arr_size

     lp:
          MOV EAX, [EDI]                          ;put  into  EAX  addr  of
next element
          SUB EAX, xmin                          ;get its index
          sub eax, 1
```

```
          MOV EBX, [ESI + 4 * EAX]        ;put the addr of result arr +
index into EBX
          INC EBX                                ;curr index + 1
          MOV [ESI + 4 * EAX], EBX        ;put the result into the arr
          ADD EDI, 4                                ;next element
          LOOP lp                                ;loop   (decr   the   ECX
value, if ECX > 0 -> loop)

     pop ebp
     POP ESI
     POP EDI

     RET
     module_1 ENDP
     END
```

Название файла: module_2.asm

```
     ;------------2nd module

     .586p
     .MODEL FLAT, C
     .CODE
     PUBLIC C module_2

     module_2  PROC  C  x:  dword,  arr_size:  dword,  left_border:  dword,
interval_cnt: dword, result: dword

     PUSH EDI
     PUSH ESI
     PUSH EBX
     PUSH ECX
     PUSH EAX
     push ebp

     MOV ECX, arr_size          ;get arr_size into ECX
     MOV ESI, x                      ;get start of the x into ESI
     MOV EDI, left_border  ;get start of the interval arr into EDI
     MOV EAX, 0                 ;index of curr value (x)

     lp_x:                                ;loop for x arr with gen
nums
          MOV EBX, 0                        ;index of start interval
          lp_borders:                            ;loop to search the
inteval for the curr x
               CMP EBX, interval_cnt      ;if in the end of interval arr
- exit loop
               JE lp_b_end
               PUSH EAX                        ;save  EAX  =  index  of
curr val
               MOV EAX, [ESI + EAX * 4]   ;get curr x value into EAX
               CMP EAX, [EDI + EBX * 4]   ;get curr interval left_border
value and CMP with EAX
                                                ;if EAX < curr left
border -> jmp to write
               POP EAX                         ;restore EAX
```

```
                JL lp_b_end                             ;jump   if   <   curr
left border of interval
                JG check_last_right

                INC EBX                                 ;interval index + 1
                JMP lp_borders

                check_last_right:
                     INC EBX
                     CMP EBX, interval_cnt
                     JE trash
                     JMP lp_borders


          lp_b_end:

          DEC EBX                                       ;turn   curr   left
border into right border (basically, get the interval we seek for)
          CMP EBX, 0
          JL trash                                  ;if index < 0 -> trash

          MOV EDI, result                 ;get  start  of  the  result  arr
into EDI
          PUSH EAX                                  ;save   EAX   =   index   of
curr val
          MOV EAX, [EDI + 4 * EBX]        ;get   x   counter   val   in   the
result by the index of the interval
          INC EAX                                       ;x counter + 1
          MOV [EDI + 4 * EBX], EAX        ;update counter
          POP EAX
          MOV EDI, left_border            ;get start of interval arr

          trash:
          INC EAX                    ;index of curr val + 1

     LOOP lp_x

     pop ebp
     POP EAX
     POP ECX
     POP EBX
     POP ESI
     POP EDI

     RET
     module_2 ENDP
     END
```