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. 1383 _____ Кошкин Е.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург
2022

Цель работы.

Получение навыков организации связи Ассемблера с ЯВУ, а именно использование модулей, написанных на Ассемблере, в программе на ЯВУ.

Задание.

Бригада 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.

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.

Из потока ввода считываются NumRanDat, Xmin, Xmax, NInt и проверяются на корректность. Далее считвается массив границ borders, сортируется по возрастанию и также проверяется на корректность. Генерируется массив псевдослучайных чисел array с помощью функции rand. Вызывается функция asmfunc1, создающая массив частотного распределения по каждому из чисел. Вызывается функция asmfunc2, которая создает частотное распределение по интервалам. Далее результат в нужном формате выводится в файл и терминал.

Тестирование.

№	Входные данные	Выходные данные
1	10 0 10 1 5	Array of random numbers: 1 1 1 8 8 9 0 2 9 7 Intermediate rusult after asmfunc1: 1 3 1 0 0 0 0 1 2 2 0 Result: 1 0 5 2 5 5
2	10 -10 10 0	Array of random numbers: 6 10 -6 4 5 -10 3 2 2 -4 Intermediate rusult after asmfunc1: 1 0 0 0 1 0 1 0 0 0 0 0 2 1 1 1 1 0 0 0 1 Result: 1 -10 10
3	10 -20 -10 2	Array of random numbers:

	-15 -19	-16 -15 -16 -18 -11 -18 -20 -13 -20 -15 Intermediate rusult after asmfunc1: 2 0 2 0 2 2 0 1 0 1 0 Result: 1 -20 2 2 -19 4 3 -15 4
4	10 -1 1 3 0 -1 1	Array of random numbers: 0 0 -1 0 1 -1 1 -1 0 0 Intermediate rusult after asmfunc1: 3 5 2 Result: 1 -1 3 2 0 7 3 1 0

Выводы.

Получены навыки организации связи Ассемблера с ЯВУ.

Разработана программа, строящая частотное распределение попаданий псевдослучайных целых чисел в заданные интервалы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

main.asm:

```
#include <stdio.h>

#include <stdlib.h>
#include <time.h>

int cmp (const void * first, const void * second) {
	int * a = (int *) first;
	int * b = (int *) second;

	if (*a > *b)
		return 1;
	else if (*a == *b)
		return 0;
	return -1;
}

void print_array (int * array, int n) {
	for (int i = 0; i < n; i ++) {
		printf ("%d ", array[i]);
	}
	printf ("\n");
}

void asmfunc1 (int *, int n, int *, int Min);
void asmfunc2 (int * borders, int * result, int n, int * tmp, int m, int
Min);

int main () {
int NumRanDat, Xmin, Xmax, NInt;

printf ("Enter NumRanDat, Xmin, Xmax, NInt \n");
scanf ("%d %d %d %d", &NumRanDat, &Xmin, &Xmax, &NInt);

//check input
if (NumRanDat > 16384 || NumRanDat < 0) {
	printf ("Error: NumRanDat in wrong interval\n");
	return 0;
}
```

```
if (Xmin > Xmax) {
        printf ("Error: Xmin > Xmax \n");
        return 0;
}
if (NInt > 24 || NInt < 0) {
        printf ("Error: wrong NInt\n");
        return 0;
}

printf ("Enter borders\n");
int * borders = calloc (NInt, sizeof (int));
for (int i = 0; i < NInt; i++) {
        scanf ("%d", &borders[i]);
}
qsort (borders, NInt, sizeof (int), cmp);
if (borders[0] < Xmin || borders[NInt - 1] > Xmax) {
        printf ("Error: wrong border\n");
        free(borders);
        return 0;
}

int * array = calloc (NumRanDat, sizeof (int));
srand (time(NULL));
for (int i = 0; i < NumRanDat; i++) {
        array[i] = Xmin + rand() % (Xmax - Xmin + 1);
}

printf ("Array of random numbers:\n");
print_array (array, NumRanDat);

int arrange = abs(Xmax - Xmin) + 1;
int * tmp = calloc (arrange, sizeof (int));
int * result = calloc (NInt + 1, sizeof (int));

asmfunc1 (array, NumRanDat, tmp, -Xmin);
printf ("Intermediate rusult after asmfunc1:\n");
print_array (tmp, arrange);

asmfunc2 (borders, result, NInt, tmp, arrange, -Xmin);

printf ("Result:\n");
FILE * file = fopen ("result.txt", "w");
```

```
int i = 1;
int k = 0;
if (result[0] != -1) {
        printf ("%d %d %d\n", 1, Xmin, result[0]);
        fprintf (file, "%d %d %d\n", 1, Xmin, result[0]);
        k = 1;
}
for (int j = 0; j < NInt; j++) {
        printf ("%d %d %d\n", j + k + 1, borders[j], result[i]);
        fprintf (file, "%d %d %d\n", j + k + 1, borders[j], result[i]);
        i++;
}
fclose(file);
free (array);
free(borders);
free (tmp);
return 0;
}
```

asmfunction.s:

```
.global asmfunc1

asmfunc1:
push rbp
mov rbp, rsp
push rdi
push rax

sub rax, rax
mainloop:
mov eax, DWORD PTR [rdi]
add eax, ecx
inc QWORD PTR [rdx + rax*4]
add rdi, 4
dec esi
cmp esi, 0
jne mainloop

pop rax
pop rdi
mov rsp, rbp
pop rbp
```

```
ret

.global asmfunc2
# rcx - tmp
# rsi - result
# rdi - borders
# edx - NInt
# r8d - arrange
# r9d - -Xmin
asmfunc2:
push rbp
mov rbp, rsp
push rdi
push rax

mov rdx, 0
# checking first border
mov eax, DWORD PTR [rdi]
add eax, r9d
cmp eax, 0
jne L1
mov DWORD PTR [rsi], -1;
add rsi, 4
add rdi, 4
L1:
mov eax, DWORD PTR [rdi]
add eax, r9d
L2:
cmp edx, r8d
je finish
mov ebx, DWORD PTR [rcx + rdx*4]
cmp edx, eax
je L3
add DWORD PTR [rsi], ebx
inc edx
jmp L2
L3:
add rsi, 4
add rdi, 4
add DWORD PTR [rsi], ebx
inc rdx
jmp L1
```

```
finish:
pop rax
pop rdi
mov rsp, rbp
pop rbp
ret
```