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. 8383 _____ Ларин А.

Преподаватель _____ Ефремов М.А,

Санкт-Петербург

2019

Цель работы.

Научится совмещать ЯВУ с языком ассемблера. Научится использовать в ЯВУ функции, написанные на языке ассемблера, передавать в них параметры и обрабатывать их. Использовать полученные знания для анализа распределения псевдослучайных чисел.

Основные теоретические положения.

Существуют следующие формы комбинирования программ на языках высокого уровня с ассемблером:

- Использование ассемблерных вставок (встроенный ассемблер, режим inline). Ассемблерные коды в виде команд ассемблера вставляются в текст программы на языке высокого уровня. Компилятор языка распознает их как команды ассемблера и без изменений включает в формируемый им объектный код. Эта форма удобна, если надо вставить небольшой фрагмент.
- Использование внешних процедур и функций. Это более универсальная форма комбинирования. У нее есть ряд преимуществ:

— написание и отладку программ можно производить независимо;
— написанные подпрограммы можно использовать в других проектах;
— облегчаются модификация и сопровождение подпрограмм.

Использование внешних процедур

Для связи посредством внешних процедур создается многофайловая программа. При этом в общем случае возможны два варианта вызова:

- программа на языке высокого уровня вызывает процедуру на языке ассемблера;
- программа на языке ассемблера вызывает процедуру на языке высокого уровня.

Рассмотрим более подробно первый вариант. В программах, написанных на языке ассемблера, используется соглашение передачи параметров stdcall. Однако по сути получение и передача параметров в языке ассемблера производится явно, без помощи транслятора.

При связи процедуры, написанной на языке ассемблера, с языком высокого уровня, необходимо учитывать соглашение по передаче параметров.

Конвенция Pascal заключается в том, что параметры из программы на языке высокого уровня передаются в стеке и возвращаются в регистре AX/EAX, — это способ, принятый в языке PASCAL (а также в BASIC, FORTRAN, ADA, OBERON, MODULA2), — просто поместить параметры в стек в естественном порядке. В этом случае запись

```
some_proc(a,b,c,d);
```

запишется как

```
push a
push b
push c
push d
call some_proc@16
```

Процедура some_proc, во-первых, должна очистить стек по окончании работы (например, командой ret 16) и, во-вторых, параметры, переданные ей, находятся в стеке в обратном порядке:

```
some_proc proc
push ebp
mov ebp,esp ; пролог
mov eax, [ebp+20] ; a
mov ebx, [ebp+16] ; b
mov ecx, [ebp+12] ; c
mov edx, [ebp+8] ; d
…
pop ebp ; эпилог
ret 16
some_proc endp
```

Этот код в точности соответствует полной форме директивы proc.

Однако можно использовать упрощенную форму, которую поддерживают все современные ассемблеры:

```
some_proc proc PASCAL, a:dword, b:dword, c:dword, d:dword
…
ret
some_proc endp
```

Главный недостаток этого подхода — сложность создания функции с изменяемым числом параметров, аналогичных функции языка C printf. Чтобы определить число параметров, переданных printf, процедура должна сначала прочитать первый параметр, но она не знает его расположения в стеке. Эту проблему решает подход, используемый в C, где параметры передаются в обратном порядке.

C используется, в первую очередь, в языках C и C++, а также в PROLOG и других. Параметры помещаются в стек в обратном порядке, и, в противоположность PASCAL-конвенции, удаление параметров из стека выполняет вызывающая процедура.

Запись some_proc(a,b,c,d)

будет выглядеть как

```
push d
push c
push b
push a
call some_proc@16
add esp,16 ; освободить стек
```

Вызванная таким образом процедура может инициализироваться так:

```
some_proc proc
push ebp
mov ebp,esp ; пролог
mov eax, [ebp+8] ; a
mov ebx, [ebp+12] ; b
mov ecx, [ebp+16] ; c
mov edx, [ebp+20] ; d
…
```

```
pop ebp
ret
some_proc endp
```

Трансляторы ассемблера поддерживают и такой формат вызова при помощи полной формы директивы proc с указанием языка C:

```
some_proc proc C, a:dword, b:dword, c:dword, d:dword
…
ret
some_proc endp
```

Регистр EBP используется для хранения параметров, и его нельзя изменять программно при использовании упрощенной формы директивы proc.

Преимущество по сравнению с PASCAL-конвенцией заключается в том, что освобождение стека от параметров в конвенции C возлагается на вызывающую процедуру, что позволяет лучше оптимизировать код программы. Например, если необходимо вызвать несколько функций, принимающих одни и те же параметры подряд, можно не заполнять стек каждый раз заново, и это — одна из причин, по которой компиляторы с языка C создают более компактный и быстрый код по сравнению с компиляторами с других языков.

Смешанные конвенции

Существует конвенция передачи параметров STDCALL, отличающаяся и от C, и от PASCAL-конвенций, которая применяется для всех системных функций Win32 API. Здесь параметры помещаются в стек в обратном порядке, как в C, но процедуры должны очищать стек сами, как в PASCAL.

Еще одно отличие от C-конвенции – это быстрое или регистровое соглашение FASTCALL. В этом случае параметры в функции также передаются по возможности через регистры. Например, при вызове функции с шестью параметрами

```
some_proc(a,b,c,d,e,f);
```

первые три параметра передаются соответственно в EAX, EDX, ECX, а только начиная с четвертого, параметры помещают в стек в обычном обратном порядке:

```
mov a, eax
mov b, edx
mov c, ecx
mov d, [ebp+8]
mov e, [ebp+12]
mov f, [ebp+16]
```

В случае если стек был задействован, освобождение его возлагается на вызываемую процедуру.

В случае быстрого вызова транслятор Си добавляет к имени значок @ спереди, что искажает имена при обращении к ним в ассемблерном модуле.

Возврат результата из процедуры

Чтобы возвратить результат в программу на С из процедуры на ассемблере, перед возвратом управления в вызываемой процедуре (на языке ассемблера) необходимо поместить результат в соответствующий регистр

Задание

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:

1. Для бригад с нечетным номером: подпрограмма формирования распределения количества попаданий псевдослучайных целых чисел в заданные интервалы реализуется в виде одного ассемблерного модуля, сразу формирующего требуемое распределение и возвращающего его в головную программу, написанную на ЯВУ;

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

Программа реализована с использованием ЯВУ C++. Функционал ЯВУ используется для объявления переменных, инициализации массивов ввода/вывода информации из стандартного потока.

Программа начинается со считывания данных из потока ввода. Далее производится заполнение массива чисел случайными числами, сгенерированными функцией dnk_normal. Имеющиеся данные передаются в функцию, реализованную на языке ассемблера.

void MAS_FUNC(int n, int ni, int xmi, int xma, int* nums, int* lB, int* res);

Функция на вссемблере пробегает массив чисел, проверяя их на принадлежность интервалу. При назождении числа в интервале соответствующая ячейка результирующего массива увеличивается на 1. После прохода всего массива программы возвращает управление в программу на ЯВУ. Там происходит форматный вывод данных в виде таблицы

Тестирование.

1.

Enter amount of elements:

5

Enter amount of intervals:

2

Enter lower border:

-5

Enter upper border:

5

Enter 1 left borders:

1

NInt lGrInt Num

0 -5 4

1 1 1

2.

Enter amount of elements:

10000

Enter amount of intervals:

10

Enter lower border:

-10

Enter upper border:

10

Enter 9 left borders:

-8 -6 -4 -2 0 2 4 6 8

NInt lGrInt Num

0 -10 2247

1 -8 656

2 -6 740

3 -4 776

4 -2 791

5 0 771

6 2 753

7 4 690

8 6 589

9 8 1987

3.

Enter amount of elements:

10000

Enter amount of intervals:

10

Enter lower border:

-50

Enter upper border:

-30

Enter 9 left borders:

-48 -46 -44 -42 -40 -38 -36 -34 -32

NInt lGrInt Num

0 -50 2247

1 -48 656

2 -46 740

3 -44 776

4 -42 791

5 -40 771

6 -38 753

7 -36 690

8 -34 589

9 -32 1987

Выводы.

В результате работы были разобраны некоторые концепции совмещения языка ассемблера с ЯВУ. Были изучены Способу передачи аргументов в функцию и возвращение значений. Была написана рабочая программа для исследования распределения псевдослучайных величин.

ПРИЛОЖЕНИЕ

EVM_6.CPP

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include<iostream>
#include"RANDOM.H"


extern "C"
{
  void MAS_FUNC(int n, int ni, int xmi, int xma, int* nums,
int* lB, int* res);
}

//using namespace std;
int main()
{
  dnk_randomize();
  int n = 0;
  int ni = 0;
  int xMin = 0;
  int xMax = 0;

  std::cout << "Enter amount of elements:\n";
  std::cin >> n;
  if (n > 16 * 1024) {
    std::cout << "amount of elements shold be no bigger than "
<< 16 * 1024; exit(1);
  }
  std::cout << "Enter amount of intervals:\n";
  std::cin >> ni;
  if (ni > 24) {
    std::cout << "amount of intervals shold be no bigger than
" << 24; exit(1);
  }
  std::cout << "Enter lower border:\n";
  std::cin >> xMin;
  std::cout << "Enter upper border:\n";
  std::cin >> xMax;

  int *nums = new int[n];
  int *lB = new int[ni];
  int *res = new int[ni];
  for (int i = 0; i < ni; i++)res[i] = 0;

  std::cout << "Enter "<<ni-1<<" left borders:\n";
```

```
      for (int i = 0; i < ni-1; i++) {
        //check for inside borders
        std::cin >> lB[i];
        if (lB[i] > xMax || lB[i] < xMin) {
          std::cout << "Left borders shoult be within min-max
interval"; exit(1);
        }
      }
      lB[ni - 1] = xMax;

      for (int i = 0; i < n; i++) {
        nums[i]    =    round(dnk_normal((double)((xMin+xMax)/2),
(double)((xMax-xMin)/2)));
        // std::cout << nums[i] << " ";
        if (nums[i] > xMax)nums[i] = xMax-1;
        if (nums[i] < xMin)nums[i] = xMin+1;
      }
      std::cout << std::endl;


      MAS_FUNC(n, ni, xMin, xMax,nums, lB,res);

      printf("NInt\tlGrInt\tNum\n");
      for (int i = 0; i < ni; i++) {
        printf("%d\t%d\t%d\n", i, i ? lB[i - 1] : xMin, res[i]);
      }

    }
```

INC.ASM

```
    .686
    .MODEL FLAT, C
    .STACK
    .DATA
    ;-----------Local data-----------------------------
    .CODE
    ;-----------External usage-------------------------
    ;-----------Function definitions-------------------
    MAS_FUNC    PROC    C    n:dword,    ni:dword,    xmi:dword,
xma:dword,nums:dword, bordArr:dword, inclArr:dword
    mov ecx,0
    mov ebx,[nums]
    mov esi,[bordArr]
    mov edi,[inclArr]
    lp:
    nop
    mov eax,[ebx]
```

```
push ebx;

mov ebx,0;edge num
lp1:
mov edx,ebx;
shl edx,2;
;mov ebx,[esi+edx]
cmp eax,[esi+edx];more then next border
jg ink
jmp ent
ink:
inc ebx
jmp lp1
ent:

add edx,edi
mov eax,[edx]
inc eax
mov [edx],eax;element lies in edx section

pop ebx;

add ebx,4
inc ecx
cmp ecx,n
jl lp

ret
MAS_FUNC ENDP

END
```