Министерство образования Республики Беларусь

Учреждение Образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Лабораторная работа №6

по дисциплине «Программирование на языке ассемблера»

на тему «Интерфейс с языками высокого уровня. Работа с математическим сопроцессором»

вариант №8

|  |  |  |
| --- | --- | --- |
| Выполнил:  студент гр. 250504  Солодков М. Д |  | Проверил:  Туровец Н.О. |

Минск 2023

**Цель работы:** Ознакомиться с вариантами внедрения ассемблерной процедуры в программу, написанную на языке программирования C\C++, изучить архитек-туру математического сопроцессора и команды работы с ним.

**Теоретические сведения:**

Написание программы полностью на языке ассемблера допустимо только для небольших программ. На практике используют совмещенные варианты со-здания программ, которые требуют сочетания ассемблера и более высоких язы-ков программирования:

-- основная часть программы пишется на языке высокого уровня, а на ассем-блере пишутся отдельные процедуры, которые должны осуществлять управле-ние нижнего уровня и(или) иметь высокую производительность;

-- ассемблерная программа использует библиотечные средства языков высо-кого уровня.

В данной лабораторной работе выполняется создание основной програм-мы на языке С\С++, а часть связанная с вычислениями на математическом со-процессоре лежит на ассемблерной процедуре.

Для выполнения работы требуется рассмотреть следующие элементы языка ассемблера и операционной системы:

1. *Соглашения об объединении программных модулей.*

Связь ассемблерных модулей с языками высокого уровня требует следу-ющих соглашений, которые сильно зависят от применяемых компиляторов и операционной системы:

-- Согласование вызовов.

Вызов процедуры и возврат из нее в головную программу должны быть согласованы друг с другом.

В DOS вызываемая процедура может находиться:

- в том же сегменте, что и команда вызова, при этом вызов называется близ-ким или внутрисегментным (NEAR), адрес возврата занимает слово и возврат из процедуры должен быть тоже близким (RETN),

- в другом сегменте, тогда вызов называется дальним или межсегментным (FAR), адрес возврата занимает двойное слово и возврат из процедуры должен быть тоже дальним (RETF).

Поэтому при объединении программных модулей, написанных на языках С и ассемблера, эти модули должны использовать одну и ту же модель памяти.

В Windows используется односегментная модель памяти FLAT, в которой все вызовы по типу являются близкими и согласование вызовов упрощается.

-- Согласование имен.

Согласование имен требуется для того, чтобы компоновщик мог собрать исполняемый модуль. Проблемы согласования имен следующие:

- автоматическое добавление в конце имени процедуры строки @N, где N – ко-личество передаваемых в стек параметров.

- автоматическое добавление символа «\_» (подчеркивание) перед именем (например, MASM генерирует подчеркивание автоматически, а TASM этого не делает).

- согласование заглавных и прописных букв (язык C автоматически различает регистр, а для TASM нужно использовать ключ /ml, чтобы различать пропис-ные и заглавные буквы).

2. *Встроенный ассемблер.*

Встроенный ассемблер – вставка ассемблерного кода непосредственно в код программы на языке высокого уровня. Использование встроенного ассем-блера позволяет создавать программы более быстро, используя небольшие фрагменты кода без выполнения выше изложенных требований по сборке про-екта.

Любую ассемблерную команду можно записать в виде:

аsm код\_операции операнды ;

-- asm – оператор встроенной команды ассемблера (для компиляторов C++ от Microsoft используется ключевое слово \_asm);

-- код\_операции – команду языка ассемблера (например, mov);

-- операнды – операнды команды (например, ax, bx).

Если с помощью одного слова asm необходимо задать много ассемблер-ных команд, то они заключаются в фигурные скобки. Комментарии можно за-писывать только в форме, принятой в языке С++.

В программе на языке С++, использующей ассемблерные команды, ино-гда необходимо задать директиву #pragma inline – эта директива сообщает компилятору, что программа содержит внутренний ассемблерный код, что важ-но при оптимизации программы.

3. *Работа с математическим сопроцессором.*

В процессорах Intel операции с плавающей запятой выполняет специаль-ный математический сопроцессор (FPU), который имеет собственные регистры и собственный набор команд.

Команды математического сопроцессора делят на следующие группы:

-- команды управления сопроцессором:

FINCSTP – увеличить указатель вершины стека (если ТОР было равно 7, то оно обнуляется).

FDECSTP – уменьшить указатель вершины стека (если ТОР было равно 0, то оно устанавливается в 7).

FFREE операнд – освободить регистр данных ST(n).

FINIT – инициализировать FPU.

FCLEX – обнулить флаги исключений.

FSTCW приемник – сохранить регистр CR в приемник (16-битная переме-няя).

FLDCW источник – загрузить регистр CR из источника (16-битная пере-менная)

FSAVE приемник – сохранить состояние FPU в область памяти размером 94 или 108 байт, в зависимости от разрядности операндов и инициализирует FPU аналогично команде FINIT.

FRSTOR источник – восстановить состояние FPU.

FWAIT (WAIT) – ожидание готовности сопроцессора (эту команду можно указывать в критических ситуациях после команд FPU, чтобы убедиться, что возможные исключения будут обработаны).

FNOP – отсутствие операции.

-- команды пересылки данных:

FLD источник – загрузить вещественное число в стек – помещает содер-жимое источника (32-х, 64-х или 80-ми битная переменная или ST(n)) и уменьшает ТОР на 1. Команда FLD ST(0) делает копию вершины стека.

FST приемник – скопировать вещественное число из стека – копирует ST(0) в приемник (32- или 64-битную переменную или пустой ST(n)).

FSTP приемник – считать вещественное число из стека – копирует ST(0) в приемник (32-, 64- или 80-битную переменную или пустой ST(n)), а затем выталкивает число из стека (помечает ST(0) как пустой и увеличивает ТОР на один).

FILD источник – загрузить целое число в стек – преобразовывает целое число со знаком из источника (16-, 32- или 64-битная переменная) в веществен-ный формат, помещает на вершину стека и уменьшает ТОР на 1.

FIST приемник – скопировать целое число из стека – преобразовывает число из вершины стека в целое со знаком и записывает его в приемник (16- или 32-битная переменная).

**Код программы (.exe)**

**Файл «main.cpp»**

#include <iostream>

extern "C" void \_cdecl func(float\* arr, float\* res);

#define ARR\_SIZE 10

void print\_arr(float\* arr) {

for (int i = 0; i < ARR\_SIZE; i++) {

std::cout << arr[i] << " ";

}

}

int main(void) {

float arr[ARR\_SIZE];

float res[ARR\_SIZE];

for (int i = 0; i < ARR\_SIZE; i++) {

std::cout << "Enter " << i + 1 << " element:";

std::cin >> arr[i];

}

func(arr, res);

print\_arr(arr);

std::cout << std::endl;

print\_arr(res);

}

**Файл «lr6.asm»**

.model flat, c

.code

func PROC

push ebp

mov ebp, esp

arr equ [ebp + 8]

res equ [ebp + 12]

mov esi, arr

mov edi, res

mov ebx, 0

mov ecx, 0

finit

main\_cycle:

cmp ecx, 10

jge exit

fld dword ptr [esi + ebx] ; загрузка Xi

test ecx, 1

jnz even\_index

; нечетный индекс

fsin

fstp dword ptr [edi + ebx]

jmp next\_iteration

even\_index:

; четный индекс

fcos

fstp dword ptr [edi + ebx]

next\_iteration:

add ebx, 4

inc ecx

jmp main\_cycle

exit:

pop ebp

ret

func ENDP

end

**Вывод программы**

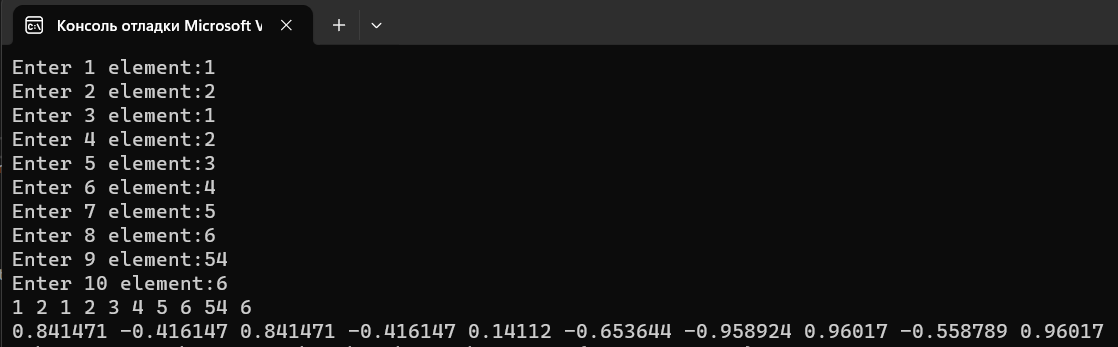
****

Рисунок 1 – Результат работы программы