МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

# Кафедра проектирования информационно-компьютерных систем

Отчет по предмету

Программирование на языке ассемблера

Лабораторная работа №6

**«ИНТЕРФЕЙС С ЯЗЫКАМИ ВЫСКОГО УРОВНЯ. РАБОТА С МАТЕМАТИЧЕСКИМ СОПРОЦЕССОРОМ.»**

|  |  |
| --- | --- |
| **Выполнил:**  Студент группы 150501  Смоленский Н.О. | **Проверил:**  Туровец Н.О. |

Минск 2022

**Цель работы**: ознакомиться с вариантами внедрения процедуры в программу, написанную на языке программирования С\С++, изучить архитектуру 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 создания программ, которые требуют сочетания ассемблера и более высоких языков программирования:

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

-- ассемблерная программа использует библиотечные средства языков высокого уровня.

1. *Соглашения об объединении программных модулей.*

Связь ассемблерных модулей с языками высокого уровня требует следующих 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 близким 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 – количество 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 прописные и заглавные буквы).

- автоматическое добавление к концу имени перегружаемой функции в С++ некоторой строки для того, чтобы эти функции различались при компоновке – для исключения этой проблемы нужно использовать модификатор extern "С".

Чтобы обеспечить доступ к глобальным переменным при объединении модулей, необходимо выполнить следующие требования:

- если процедура на языке ассемблера вызывается из программы на языке C\С++, то такая процедура в языке ассемблера должна быть описана как  
PUBLIC;

- если переменная объявлена в программе на языке ассемблера, то в программе на ассемблере она должна иметь атрибут PUBLIC, а в программе на С\С++ – extern;

- если переменная объявлена в программе на C\С++, то в программе на ассемблере она должна иметь атрибут EXTRN.

-- Согласование параметров.

Язык С использует следующие правила формирования параметров:

-- параметры помещаются в стек в порядке обратном их записи в списке параметров;

-- удаление параметров из стека выполняет вызывающая программа.  
При этом запись вызова функции:

some\_proc(a, b, c, d, e)

превращается в следующий ассемблерный код:

push e

push d

push с

push b

push a

call some\_proc

add sp,10 ; освободить стек от параметров

2. *Встроенный ассемблер.*

Встроенный ассемблер – вставка ассемблерного кода непосредственно в  
код программы на языке высокого уровня. Использование встроенного ассемблера позволяет создавать программы более быстро, используя небольшие фрагменты кода без выполнения вышеизложенных требований по сборке проекта.

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 ассемблерных команд, то они заключаются в фигурные скобки. Комментарии можно записывать 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 важно при оптимизации программы.

В командах встроенного ассемблера можно свободно использовать переменные из языка высокого уровня, так как они автоматически преобразуются в соответствующие выражения.

3. *Работа с математическим сопроцессором.*

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

Сопроцессор может выполнять операции с разными типами данных, в том числе и с данными с плавающей запятой.

Сопроцессор выполняет все вычисления в 80-ти битном расширенном формате, а 32-х и 64-х битные числа используются для обмена данными с основным процессором и памятью.

В математическом сопроцессоре есть следующие регистры:

-- регистры данных (R0 – R7) – не доступны по именам, а рассматриваются как стек, вершина которого называется ST(0) или просто ST, а следующие элементы – ST(1), ST(2) и т.д. до ST(7).

-- регистр состояний SR

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| B | C3 | TOP | TOP | TOP | C2 | C1 | C0 | ES | SF | PE | UE | OE | ZE | DE | IE |

Важные флаги регистра состояний:

- С3 – С0 – результат выполнения предыдущей команды, используются для условных переходов;

- ТОР – номер регистра данных, который в настоящий момент является вершиной стека.

- ES – общий флаг ошибки;

- SF – ошибка стека;

- UE – флаг антипереполнения;

- ОЕ – флаг переполнения;

- ZE – флаг деления на ноль;

- IE – флаг недопустимой операции.

-- регистр управления CR

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  |  |  | IC | RC | RC | PC | PC |  |  | PM | UM | OM | ZM | DM | IM |

Важные флаги регистра состояний:

- RC – управление округлением (0 – к ближайшему числу, 1 – к отрицательной бесконечности, 2 – к положительной бесконечности, 3 – к нулю).

- PC – управление точностью результатов команд (FADD, FSUB, FSUBR, FMUL, FDIV, FDIVR и FSQRT): 0 – одинарная точность (32-х битные числа), 2 – двойная точность (64-х битные), 3 – расширенная точность (80-ти битные).

- биты 0 – 5 – маскируют соответствующие исключения – если маскирующий бит установлен, то исключения не происходит, а результат вызвавшей его команды определяется правилами для каждого исключения специально.

-- регистр тегов TW – описывает текущее состояние каждого регистра данных (биты 15 - 14 описывают регистр R7, 13 - 12 — R6 и т.д.): 00 – регистр содержит число, 01 – ноль, 10 – нечисло (бесконечность, денормализованное число, неподдерживаемое число), 11 – регистр пуст.

-- регистры FIP и FDP содержат адрес последней выполненной команды и адрес ее операнда соответственно (используются в обработчиках исключений для анализа вызвавшей его команды).

Особенностями использования математического сопроцессора являются:  
 - необходимость инициализации с помощью команды FINIT перед использованием;  
 - параллельная работа процессора Intel 8086 и сопроцессора требуют дополнительной синхронизации, т.к. оба процессора подключены к общей системной шине, например, при работе с памятью:

FIST I ; скопировать число в память I

FWAIT ; ожидать готовности сопроцессора

MOV AX,I ; загрузить данные в центральный процессор

Команда FWAIT приостанавливает работу центрального процессора, который может загрузить данные в регистр AX быстрее, чем нужные данные скопируются из сопроцессора. В современных процессорах такие операции синхронизации обычно выполняются автоматически.

**Код программы:**

#pragma inline

#include "stdio.h"

#include "locale.h"

#define N 10

float array[N];

void input();

void output();

void inlineAsm();

int main()

{

setlocale(LC\_ALL, "Russian");

printf("Введите 10 элементов массива: \n");

input();

printf("\nМассив: \n");

output();

inlineAsm();

printf("\nПреобразованный массив: \n");

output();

return 0;

}

void input()

{

int flag;

for (int i = 0; i < N; ++i)

{

do

{

rewind(stdin);

flag = scanf\_s("%f", &array[i]);

if (flag != 1) printf("Должно быть введено число!\n");

} while (flag != 1);

}

}

void output()

{

for (int i = 0; i < N; ++i)

{

printf("%.3f ", array[i]);

}

printf("\n");

}

void inlineAsm()

{

\_asm

{

finit

mov ecx, N

lea ebx, array

fldz

Calculate\_loop:

fld [ebx]

fcom

fstsw ax

sahf

ja Greater

Less:

fmul [ebx]

fstp [ebx]

jmp End

Greater:

fmul [ebx]

fmul [ebx]

fstp [ebx]

End:

add ebx, 4

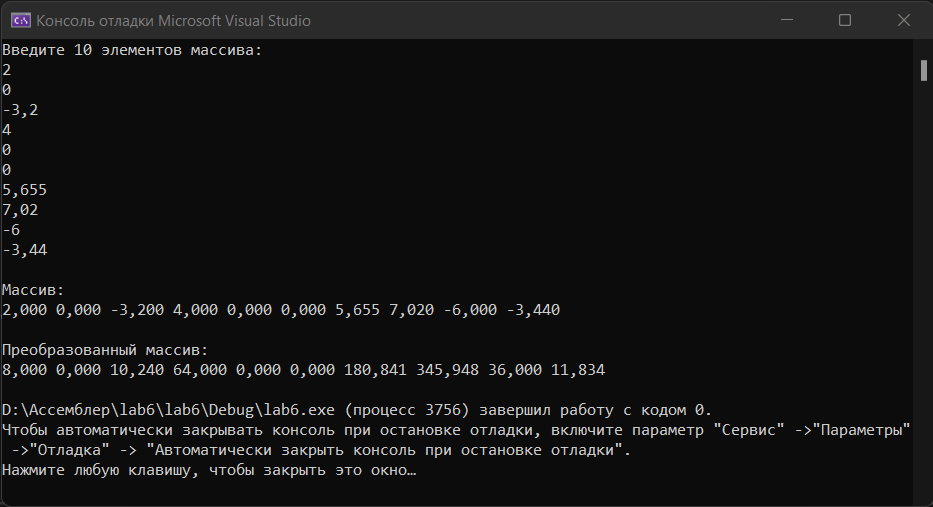
loop Calculate\_loop

fwait

}

}

**Результат работы:**

****

**Вывод**

В ходе выполнения лабораторной работы было проведено ознакомление с различными вариантами внедрения процедуры в программу, написанную на языке программирования С, изучены архитектура математического сопроцессора и команды работы с ним.