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 «Конструирование программ и языки программирования»

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

вариант №6

|  |  |  |
| --- | --- | --- |
| Выполнил  студент гр. 250505  Зорин А.Ю. |  | Проверил  Туровец Н.О. |

Минск 2023

**Цель работы:** Ознакомиться с вариантами внедрения ассемблерной процедуры в программу, написанную на языке программирования C\C++, 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.

В данной лабораторной работе выполняется создание основной программы на языке С\С++, а часть связанная с вычислениями на математическом сопроцессоре 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 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.

Для стыковки ассемблерных процедур с головной программой следует знать правила передачи 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 входе в ассемблерную процедуру в стеке будут сохранены регистры SI и DI и размещены локальные переменные х и у. Доступ к этим данным организуется с помощью адресации по базе с использованием регистра ВР.

По команде RET автоматически генерируются команды восстановления регистров SI, DI, ВР, SP и затем только выполняется возврат в вызывающую программу.

В ассемблерной процедуре можно свободно использовать регистры AX, BX, CX, DX. Остальные регистры должны быть сохранены (например, в стеке), а затем восстановлены.

Возвращаемое из процедуры значение обычно передается в регистре АХ. Если возвращаемый результат не умещается в одном регистре, то такие данные передаются через DX:AX, а если результат число с плавающей запятой, то через ST(0).

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 выше изложенных 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, иногда 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 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

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

Важные флаги регистра управления:

- 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 58 для анализа вызвавшей его команды).

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 переменяя).

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

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 содержимое 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 вещественный 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).

FISTP приемник – считать целое число из стека – преобразовывает число из вершины стека в целое со знаком и записывает его в (16-, 32- или 64-битная переменная), а затем выталкивает число из стека.

FBLD источник – загрузить десятичное число в стек – преобразовывает BCD число из источника (80-битная переменная в памяти), помещает в вершину стека и уменьшает ТОР на 1.

FBSTP приемник – считать десятичное число из стека – преобразовывает число из вершины стека в 80-битное упакованное десятичное, записывает его в приемник (80-битная переменная) и выталкивает это число из стека.

FXCH приемник – обменять местами два регистра стека – обмен местами содержимого регистра ST(0) и источника (регистр ST(n)), если операнд не указан, обменивается содержимое ST(0) и ST(1).

-- команды базовой арифметики:

FADD приемник,источник – сложение вещественных чисел:

a) FADD источник – когда источником является 32- или 64-битная переменная, а приемником – ST(0);

b) FADD ST(0),ST(n), FADD ST(n),ST(0) – когда источник и приемник заданы явно в виде регистров FPU;

c) FADD (без операндов) – эквивалентно FADD ST(0),ST(1).

FADDP приемник,источник – сложение с выталкиванием из стека:

a) FADDP ST(n),ST(0) – когда источник и приемник заданы явно в виде регистров FPU;

b) FADDP (без операндов) – эквивалентно FADDP ST(1),ST(0).

FIADD источник – сложение целых чисел, когда источником является 16- или 32-битная переменная, содержащая целое число, а приемником – ST(0).

Варианты задания операндов для ниже перечисленных команд аналогичны выше перечисленным ситуациям задания операндов для команд сложения (с учетом выполняемых действий).

FSUB приемник,источник – вычитание вещественных чисел.

FSUBP приемник,источник – вычитание с выталкиванием из стека:

FISUB источник – вычитание целых чисел.

FSUBR приемник,источник – обратное вычитание вещественных чисел (вычитание приемника из источника).

FSUBRP приемник,источник – обратное вычитание с выталкиванием.

FISUBR источник – обратное вычитание целых чисел.

FMUL приемник,источник – умножение вещественных чисел.

FMULP приемник,источник – умножение с выталкиванием из стека.

FIMUL источник – умножение целых чисел.

FDIV приемник,источник – деление вещественных чисел (некоторые ассемблеры безоперандную версию команды FDIV выполняют как команду FDIVP, т.е. безоперандная мнемоника FDIV выполняет ST(1)=ST(1)/ST(0) и выталкивает из стека верхний элемент, после чего результат оказывается в ST(0)).

FDIVP приемник,источник – деление с выталкиванием из стека. FIDIV источник – деление целых чисел.

FDIVR приемник,источник – обратное деление вещественных чисел.

FDIVRP приемник,источник – обратное деление с выталкиванием из стека.

FIDIVR источник – обратное деление целых чисел. FABS – найти абсолютное значение ST(0). FCHS – изменить знак ST(0).

FRNDINT – округлить ST(0) до целого в соответствии с режимом округления, заданным битами RC.

FSCALE – масштабировать по степеням двойки – умножает ST(0) на 2 в степени ST(1) (значение ST(1) предварительно округляется в сторону нуля до целого числа) и записывает результат в ST(0).

FXTRACT – извлечь экспоненту и мантиссу из числа в ST(0) (действие, обратное FSCALE) – разделяет число на мантиссу и экспоненту так, что мантисса оказывается в ST(0), а экспонента – в ST(1).

FSQRT – извлечь квадратный корень из ST(0), сохраняет результат в ST(0).

-- команды сравнения (основные):

FCOM источник – сравнить вещественные числа.

FCOMP источник – сравнить и вытолкнуть из стека.

FCOMPP источник – сравнить и вытолкнуть из стека два числа.

Команды типа FCOM выполняют сравнение содержимого регистра ST(0) с источником (32- или 64-битная переменная или регистр ST(n), если операнд не указан – ST(1)) .

После команд сравнения с помощью команд FSTSW AX и SAHF можно перевести флаги С3, С2 и С0 во флаги ZF, PF и CF соответственно, после чего 61 все условные команды могут использовать этот результат сравнения, как после команды СМР.

FICOM источник – сравнить целые числа

FICOMP источник – сравнить целые числа и вытолкнуть из стека.

Команды типа FICOM сравнивают содержимое регистра ST(0) и источника (16- или 32-битная переменная), причем считается, что источник содержит целое число:

FTST – сравнить SP(0) с нулем и установить флаги C3, C2, C0

- трансцендентные операции – выполняют операцию над числом, находящемся в ST(0), и обычно сохраняют результат в этом же регистре; для всех тригонометрических команд, операнд считается заданным в радианах.

FPTAN – тангенс (ST(0) содержит 1, тангенс в ST(1)). Единица помещается в стек для того, чтобы можно было получить котангенс вызовом команды FDIVR сразу после FPTAN.

FPATAN – арктангенс числа, получаемого при делении ST(1) на ST(0).

F2XMI – вычисление 2х -1 (x в ST(0) и должно быть в диапазоне [-1..+1]).

FYL2X – вычисление у\*log2(x) (x – в ST(0) и должно быть неотрицательным, y – в ST(1)).

FSIN – синус.

FCOS – косинус.

-- команды записи констант – помещают в ST(0) часто используемую в вычислениях точную константу:

FLD1 – 1,0.

FLDZ – 0,0.

FLDPI – число π.

FLDL2E – log2(e).

FLDL2T – log2(10).

FLDLN2 – ln(2).

FLDLG2 – lg(2).

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 (C++)**

#include <iostream>

#include <limits>

#pragma inline

#define SIZE 10

float mas[SIZE];

void asmFunc();

void OutputArr();

void InputArr();

int main() {

InputArr();

OutputArr();

std::cout << std::endl;

asmFunc();

OutputArr();

return 0;

}

void asmFunc() {

\_\_asm {

finit

lea ebx, mas

mov ecx, 10

func:

fld[ebx]

fsin

fstp[ebx]

add ebx, 4

loop func

fwait

}

}

void OutputArr() {

std::cout << "Your massive: " << std::endl;

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

std::cout << mas[i]<<' ';

}

void InputArr() {

std::cout << "Input elements of array: " << std::endl;

for (int i = 0; i < SIZE; i++) {

std::cin >> mas[i];

while (!std::cin) {

std::cout << "Incorrect input." << std::endl;

std::cin.clear();

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cin >> mas[i];

}

}

}

**Вывод программы**

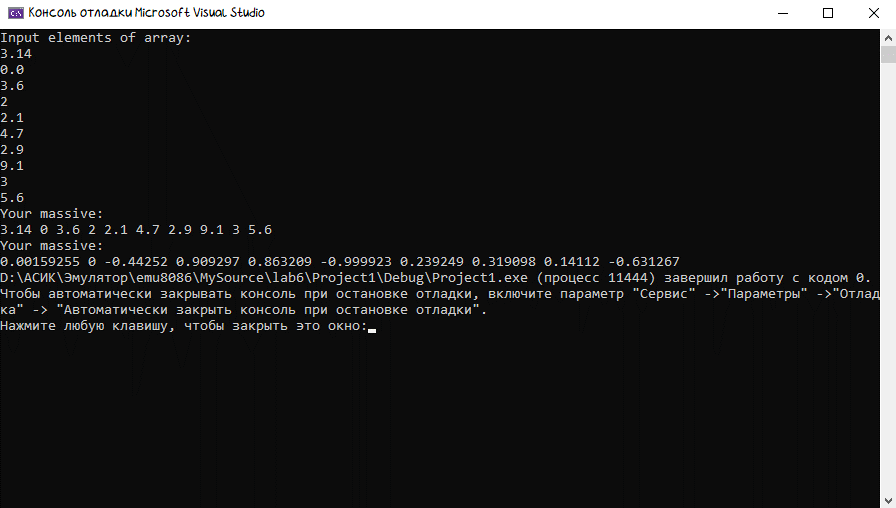


Рисунок 1 – Результат работы программы