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»

Вариант 2

Выполнил студент гр. 150502: Альхимович Н.Г.

Проверил:        Туровец Н.О.

Минск 2022

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.

Вариант задания:

Ввести массив чисел с плавающей точкой на 10 элементов. Вычислить среднее значение элементов массива.

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 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 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 – 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 пропис- ные 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

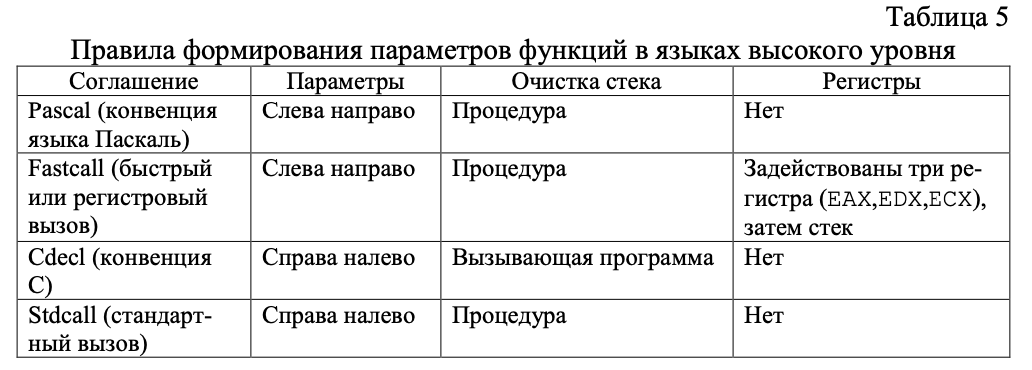
модулей, необходимо выполнить следующие требования:  
- если процедура на языке ассемблера вызывается из программы на языке C\С++, то такая процедура в языке ассемблера должна быть описана как PUBLIC;

- если переменная объявлена в программе на языке ассемблера, то в програм- ме 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 ассем- блере она должна иметь атрибут EXTRN.

-- Согласование параметров.  
Для стыковки ассемблерных процедур с головной программой следует

знать правила передачи параметров (см. таблицу 5).



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

Вызываемая таким образом процедура может формироваться так:

some\_proc proc

push bp

mov bp,sp

a equ [bp+4]

b equ [bp+6]

с equ [bp+8]

d equ [bp+10]

e equ [bp+12]

...

pop bp

ret

some\_proc endp

; создать стековый кадр

; простой доступ к параметру

; исполняемый код процедуры

Ассемблеры поддерживают и такой формат вызова при помощи услож- ненной формы директивы proc с указанием языка С:

.model small,с

some\_proc proc near uses si di, a:word, b:word

local x:word, y:word

...

...

ret

some\_proc endp  
Функции some\_proc соответствует следующий прототип в С програм- ме:

int some\_proc(int a, int b);

При входе в ассемблерную процедуру в стеке будут сохранены регистры SI и DI и размещены локальные переменные х и у. Доступ к этим данным ор- ганизуется с помощью адресации по базе с использованием регистра ВР. При этом нет необходимости вычислять смещения вручную, поскольку ассемблер автоматически генерирует макроподстановки типа:

a EQU <WORD PTR [bp+6]>

поэтому в тексте программы в качестве операндов можно использовать имена локальных переменных и передаваемых параметров.

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).

Ниже приведен пример, где головная программа для DOS написана на языке С и находится в файле C\_ASM.C, а вызываемая процедура написана на языке ассемблера и находится в файле С\_ASM.ASM.

// C\_ASM.C

#include <stdio.h>

extern "C" void asm\_proc(int \*i1, int \*i2, unsigned long l1);

void main(void)

{

int i=5, j=7;

unsigned long l=0x12345678;

printf("i=%d; j=%d; l=%lx\n", i, j, l);

asm\_proc(&i, &j, l);

printf("i=%d; j=%d; l=%lx\n", i, j, l);

}

; С\_ASM.ASM

.model small

.code

PUBLIC C asm\_proc

asm\_proc PROC near

push bp

mov bp,sp

push si di

mov si,[bp+4]

mov di,[bp+6]

mov bx,[bp+8]

mov ax,[bp+10]

mov cx,[si]

xchg cx,[di]

mov [si],cx

pop di si

pop bp

ret

asm\_proc endp

end

Пример головной программы WIN\_ASM.CPP и ассемблерного модуля WIN\_ASM.ASM для Windows:

// WIN\_ASM.CPP

#include <iostream>

using namespace std;

extern "C" int MAS\_PROC (int \*, int);

int main() {

int \*mas, n, k;

system("chcp 1251");

system("cls");

cout << "Введите размер массива: ";

cin >> n;

mas = new int[n];

cout << "Введите элементы массива: " << endl;

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

{

cout << "mas[" << i <<"]= ";

cin >> mas[i];

}

k = MAS\_PROC(mas, n);

cout << mas[1] << "\*2= " << k;

cin.get(); cin.get();

return 0;

}

; WIN\_ASM.ASM

.586

.MODEL FLAT, C

.CODE

MAS\_PROC PROC C mas:dword, n:dword

mov esi,mas

mov eax,[esi+4]

shl eax, 1

ret

MAS\_PROC ENDP

END

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- гданеобходимозадатьдирективу#pragma inline–эта директива сообщает компилятору, что программа содержит внутренний ассемблерный код, что важ- но при оптимизации программы.

Ниже приведен пример программы на языке C++ с использованием встроенного ассемблера.

#include <iostream.h>

#pragma inline

void main(void)

{

int a=10, b=20, c;

cout << " a= " << a << " b= " << b << "\n";

asm mov ax,10; // в ax значение 10

asm mul a; // умножение ax = ax \* a

c = \_AX;

cout << "c=" << c << "\n";

asm {

mov ax,a

mov bx,b

xchg ax,bx

mov a,ax

mov b,bx

}

cout << " a= " << a << " b= " << b << "\n";

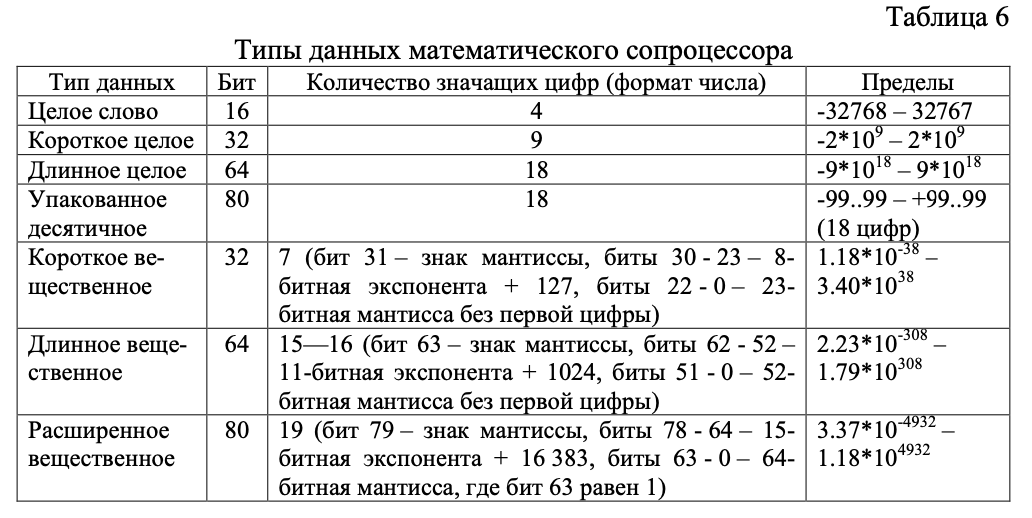
}

В командах встроенного ассемблера можно свободно использовать пере- менные 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 специаль- ный 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 (см. таблицу 6).



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 (см. рисунок 2).

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

Рисунок 2 – Формат регистра состояний

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 (см. рисунок 3).  
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

Рисунок 3 – Формат регистра управления

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 дан- ных (биты 15 - 14 описывают регистр R7, 13 - 12 — R6 и т.д.): 00 – регистр со- держит 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).  
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–освободитьрегистрданныхST(n).  
FINIT – инициализировать FPU.  
FCLEX – обнулить флаги исключений.  
FSTCW приемник – сохранить регистр CR в приемник (16-битная переменная).  
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 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).

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 верши- ну 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)–эквивалентноFADD 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(безоперандов)–эквивалентноFADDP ST(1),ST(0).

FIADD источник–сложениецелыхчисел,когдаисточникомявляется16- или 32-битная переменная, содержащая целое число, а приемником – ST(0).

Варианты задания операндов для ниже перечисленных команд аналогич- ны 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: источник–вычитаниецелыхчисел.

FISUB 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 округ- ления, 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).

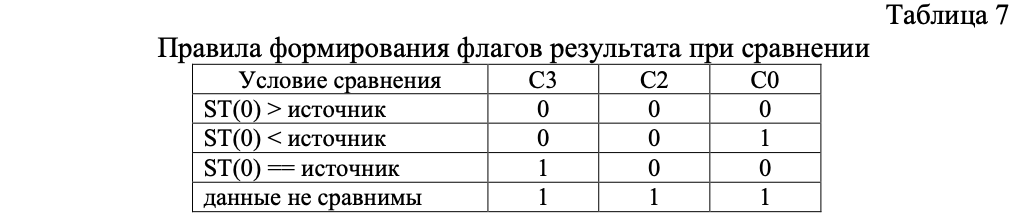
FXTRACT – извлечь экспоненту и мантиссу из числа в ST(0) (действие, об- ратное FSCALE) – разделяет число на мантиссу и экспоненту так, что мантисса оказывается в ST(0), а экспонента – в ST(1).

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)) и устанавливают флаги согласно таблице 7.

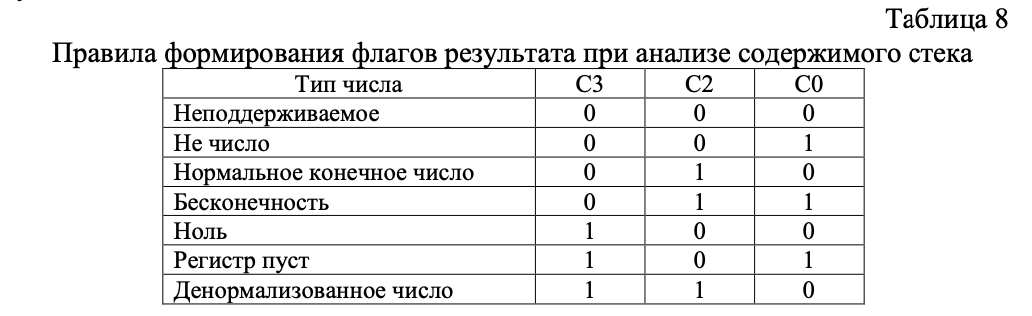


После команд сравнения с помощью команд FSTSW AX и SAHF можно перевести флаги С3, С2 и С0 во флаги ZF, PF и CF соответственно, после чего все условные команды могут использовать этот результат сравнения, как после команды СМР.

FICOM источник – сравнить целые числа  
FICOMP источник–сравнить целые числа и вытолкнуть из стека.

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

FTST – сравнить SP(0) с нулем и установить флаги C3, C2, C0.  
FXAM – проанализировать содержимое ST(0), устанавливает флаги, как указано в таблице 8.



-- трансцендентные операции – выполняют операцию над числом, находя- щемся в ST(0), и обычно сохраняют результат в этом же регистре; для всех тригонометрических команд, операнд считается заданным в радианах и не мо- жет быть больше 263 или меньше -263.

FPTAN – тангенс (ST(0) содержит 1, тангенс в ST(1)). Единица помещает- ся 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 неотрицатель- ным, y – в ST(1)).

FSIN – синус.

FCOS – косинус.

-- команды записи констант – помещают в ST(0) часто используемую в вы- числениях 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:

finit

fld b

fld x

loop\_start:

fcom

// инициализация сопроцессора

// b загрузить в стек

// ST(0)=b

// x загрузить в стек

// ST(0)=x, ST(1)=b

// метка начала цикла

// сравнить ST(0) и ST(1)

// копировать регистр состояния в AX

FIST I

FWAIT

MOV AX,I

; скопировать число в память I

; ожидать готовности сопроцессора

; загрузить данные в центральный процессор

Команда FWAIT приостанавливает работу центрального процессора, ко- торый может загрузить данные в регистр AX быстрее, чем нужные данные ско- пируются из сопроцессора. В современных процессорах такие операции син- хронизации обычно выполняются автоматически.

Пример C программы с использованием встроенных команд сопроцессора для MS Visual Studio приведен ниже:

// вычислить сумму функций

// F(x) = sin(x + 1) / (cos(x) + 2)

// на интервале [a, b] с шагом step

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

void main(void)

{

double a = -1000, b = 1000, step = 0.0001, x, f;

int two = 2;

clock\_t start, end;

x = a;

f = 0;

start = clock();

\_asm {

fstsw ax

and ah,01000101b // проверить (биты: 08=С0, 10=C2, 14=C3)

jz loop\_end // если x==b, то завершить цикл

fld1

fadd x

fsin

fld x

fcos

fiadd two

fdiv

fadd f

fstp f

fadd step

fst x

// загрузить константу 1

// ST(0)=1, ST(1)=x, ST(2)=b

// сложить ST(0) = ST(0) + x

// ST(0)=1+x, ST(1)=x, ST(2)=b

// вычислить синус ST(0) = sin(ST(0))

// ST(0)=sin(1+x), ST(1)=x, ST(2)=b

// х загрузить в стек

// ST(0)=x, ST(1)=sin(1+x), ST(2)=x, ST(3)=b

// косинус ST(0) = cos(ST(0))

// ST(0)=cos(x),ST(1)=sin(1+x),ST(2)=x, ST(3)=b

// целочисленно сложить ST(0) = ST(0) + two

// ST(0)=cos(x)+2,ST(1)=sin(1+x),ST(2)=x,ST(3)=b

// делить ST(1) = ST(1) / ST(0) с выталкиванием

// ST(0) аналогично команде fdivp ST(1),ST(0)

// ST(0)=((sin(1+x)/(cos(x)+2)),ST(1)=x,ST(2)=b

// сложить

// ST(0) = ((sin(1+x)/(cos(x)+2))) + f

// ST(1) = x, ST(2) = b

// переместить ST(0) в f

// ST(0) = x, ST(1) = b

// сложить

// ST(0) = x + step, ST(1) = b

// cкопировать ST(0) в x

// ST(0) = x + step, ST(1) = b

jmp loop\_start // перейти на начало цикла

loop\_end:

fwait // синхронизировать

}

end = clock();

printf("%f\n", f);

printf("\ntime asm %f\n", (double)(end-start)/ CLK\_TCK);

x = a;

f = 0;

start = clock();

while(x <= b)

{

f += sin(x+1) / (cos(x) + 2);

x += step; };

end = clock();

printf("%f\n", f);

printf("\ntime c

getch();

}

%f\n", (double)(end-start)/ CLK\_TCK);

Код программы:

#include <iostream>

#pragma inline

#define n 10

using namespace std;

int main()

{

int i;

float\* array = new float[n], ten = 10.0, result = 0.0;

cout << "Enter elements of the array: " << endl;

for(i = 0; i < n; i++)

{

cout << "array[" << i << "] = ";

cin >> array[i];

while(cin.fail()) //if input is incorrect

{

cin.clear(); //reset

cin.ignore(100, '\n'); //clear the buffer

cout << "Entered value isn't a number, try again: ";

cin >> array[i];

}

}

\_asm

{

xor ecx, ecx

mov cx, n

finit //initialise a floating-point unit (FPU)

mov eax, array

fld ten //push ten -> ST(2)

fld result //ST(1)

calc\_mean:

fadd [eax] //add the element to ST(0)

add eax, 4 //move to the next element

cmp cx, 0

dec cx

jnz calc\_mean

fdiv ST(0), ST(1)

fst result //copy from ST(0) to variable

}

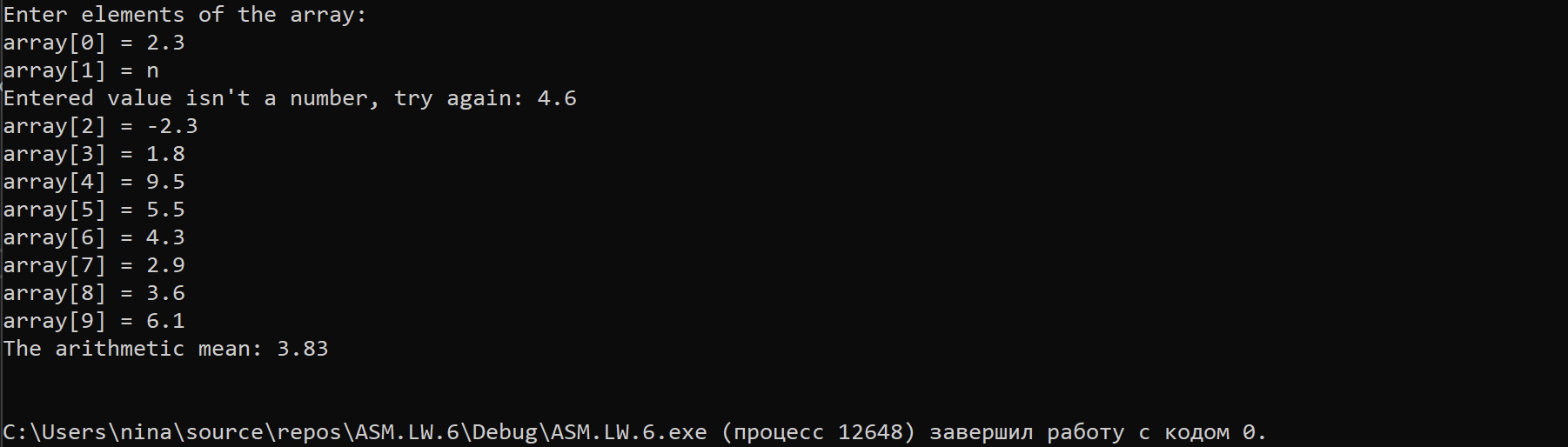
cout << endl << "The arithmetic mean: " << result << endl << endl;

delete[] array;

return 0;

}

Примеры работы программы:



Вывод:

В ходе лабораторной работы было написано консольное приложение С++, вычисляющее с помощью ассемблерной процедуры среднее арифметическое значение элементов массива вещественных чисел. Я ознакомилась с вариантами внедрения ассемблерной процедуры в программу, написанную на языках программирования высокого уровня C\C++, изучила архитектуру математического сопроцессора и команды работы с ним.