МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра электронных вычислительных машин

Отчет по предмету
Проектирование на языке ассемблера
Вариант 1

Лабораторная работа 6
«Встроенный ассемблер и математический сопроцессор»

Выполнил:
Студент группы 150503
Шарай П.Ю.

Проверил:
Туровец Н.О.

Минск 2022

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.

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

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

- если процедура на языке ассемблера вызывается из программы на языке С\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\C++, то в программе на ассемблере она должна иметь атрибут EXTRN.

-- Согласование параметров.

При входе в ассемблерную процедуру в стеке будут сохранены регистры SI и DI и размещены локальные переменные x и y. Доступ к этим данным организуется с помощью адресации по базе с использованием регистра BP.

По команде RET автоматически генерируются команды восстановления регистров SI, DI, BP, 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 значение обычно передается в регистре AX.

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:
asm 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 C++.

В программе на языке C++, 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 пере-
менные из языка высокого уровня, так как они автоматически преобразуются в соответствующие выражения.

3.*Математический сопроцессор*

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

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

Флаги регистра состояний:

- C3 – C0 – результат выполнения предыдущей команды, используются для условных переходов;

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

- ES – общий флаг ошибки;

- SF – ошибка стека;

- OE – флаг переполнения;

- ZE – флаг деления на ноль;

- IE – флаг недопустимой операции.

Команды математического сопроцессора:

-FINIT – инициализировать FPU.

-FCLEX – обнулить флаги исключений.

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

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

-FSAVE приемник – сохранить состояние FPU в область памяти размером 94 или 108 байт

-FPU аналогично команде FINIT.

-FWAIT (WAIT) – ожидание готовности сопроцессора

--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 TOP 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 TOP 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 TOP 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).

-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 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).

-FSUB приемник,источник – вычитание вещественных чисел.

-FSUBP приемник,источник – вычитание с выталкиванием из стека:

-FISUB источник – 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

-FDIVP приемник,источник – деление с выталкиванием из стека.

-FIDIV источник – деление целых чисел.

-FABS – найти абсолютное значение ST(0).

-FCHS – изменить знак ST(0).

-FXTRACT – извлечь экспоненту и мантиссу из числа в ST(0) (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.

-FICOM источник – сравнить целые числа

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

Код программы:

```
#include <iostream>
#include <conio.h>
#pragma inline
using namespace std;

void main() {
	float* array = new float[10];		//массив чисел
	short len = 10;				//размер массива
```

```
	float sum = 0;                          //сумма элементов массива
	cout << "Input array:" << endl;
	for (int i = 0; i < len; i++)           //цикл ввода элментов массива
		cin >> array[i];
	system("cls");
	cout << "Array:" << endl;
	for (int i = 0; i < len; i++)           //вывод массива
		cout << array[i] << endl;
	_asm {
		xor ecx, ecx                    //обнуление регистра
		mov cx, len                     //счетчик цикла
		dec cx
		finit                           //Без FINIT
		mov eax, array                  //адрес массива чисел
		fld[eax]                        //занесение первого элемента
массива в стек
	Cycle:
		add eax, 4                      //переход к следющему элементу
массива
		fadd[eax]                       //добавление элементамассива к
вершине стека
		loop Cycle

		fst sum                         //Сохранить вещественное
значение с извлечением из стека
	};
	cout<< "Sum: "<< sum <<endl;            //вывод суммы элментов массива
	_getch();                               //задержка консоли
}
```

Результат работы программы: