Лабораторная работа 7. Компоновка ассемблерных функций с программами, написанными на языке высокого уровня

Цель работы

Познакомиться с механизмом вызова функций языка C. Освоить методы написания ассемблерных функций для программ, написанных на языке C.

Используемое программное обеспечение

1. GNU Binutils.
2. Клиент сетевой системы для проведения лабораторных работ asm7.jar.

Порядок выполнения лабораторной работы

1. Зарегистрируйтесь в системе под своим именем.
2. Ознакомьтесь с заданием по лабораторной работе.
3. Напишите программу, соответствующую Вашему заданию.
4. Нажмите кнопку Проверить и выберите файл с исходным кодом.

Задание

Требуется написать функцию на языке ассемблера, выполняющую обработку комплексных чисел. Функция должна называться _process и иметь следующую сигнатуру:

```
int process( int re1, int im1, int re2, int im2 );
```

Параметры функции - вещественные и мнимые части первого и второго комплексного числа. Функция должна выполнить некоторые действия с этими комплексными числами и вернуть квадрат модуля результата. Для упрощения задания все операции являются целочисленными. Действия, выполняемые с комплексными числами опредляются Вашим вариантом задания.

Обратите внимание, что ко всем идентификаторам в коде на языке C на этапе компиляции добавляется префикс в виде символа подчёркивания. Поэтому, чтобы использовать эти идентификаторы в ассемблерном коде, нужно добавлять к ним префикс _, а в C-коде использовать те же идентификаторы без префикса.

Для проверки корректности выполнения задания можно использовать следующую программу на языке C:

```
#include <stdio.h>

extern int process( int, int, int, int );

int main() {
    int re1, im1, re2, im2;
    printf( "Enter four integer numbers and have fun\n" );
    scanf( "%d", &re1 );
    scanf( "%d", &im1 );
    scanf( "%d", &re2 );
    scanf( "%d", &im2 );
    int result = process( re1, im1, re2, im2 );
    printf( "The answer is %d. Have you expected such result? "
        "Think about it and make it work.\n", result );
}
```

Допустим, эта программа содержится в файле main.c, а функция на ассемблере – в файле func.s. Тогда, чтобы скомпоновать их в одну программу program и запустить её, нужно выполнить следующие команды (Mac OS X 32):

```
gcc -c -arch i386 -o main.o main.c
as -arch i386 -o func.o func.s
gcc -arch i386 -o program main.o func.o
./program
```

Пример выполнения задания

Для хранения параметров функций и локальных переменных в языке C используется стек-фрейм. Это область памяти, выделенная на стеке. Вершина стека содержится в регистре esp. Перед вызовом функции в стек помещаются её параметры, при этом вершина стека сдвигается вниз. Вызываемая функция использует стек для хранения локальных переменных и передачи параметров другим функциям. Поэтому, для удобства значение регистра esp в момент входа в функцию сохраняется в регистр ebp, после чего регистр esp можно свободно изменять, а регистр ebp служит для доступа к параметрам функции (вверх от ebp) и локальным переменным (вниз от ebp). Таким образом образуется стек-фрейм. Перед завершением функция должна восстановить стек-фрейм вызывающей функции.

Программы на языке C используют следующее соглашение о порядке вызова функций:

1. Вызывающий код помещает параметры вызываемой функции в стек (инструкция push) и вызывает функцию (инструкция call).

2. Вызываемая функция сохраняет текущее значение регистра ebp (база стек-фрейма вызывающей функции) в стек, а в регистр ebp помещает текущую вершину стека, содержащуюся в регистре esp.

3. Если функция должна возвращать какое-то значение, это значение сохраняется в регистр eax.

4. Перед завершением функция должна восстановить значения регистров esp и ebp.

5. Для возврата в вызывающий код нужно выполнить инструкцию ret.

Рассмотрим реализацию функций на языке ассемблера на примере функции, которая возвращает сумму двух своих параметров. На языке C такая функции выглядела бы следующим образом:

```
int sum( int a, int b ) {
    return a + b;
}
```

Эта же функция на языке ассемблера:

```
.intel_syntax noprefix

.text

/* Начало функции sum */
.globl _sum
_sum:

/* Сохранение стек-фрейма вызывающей функции */
push ebp
mov ebp, esp
/* Доступ к параметрам функции осуществляется через регистр ebp
Первый параметр: [ebp + 8]
Второй параметр: [ebp + 12]
Третий параметр: [ebp + 16]
и так далее */
mov eax, [ebp + 8]
/* Результат сохраняется в регистре eax */
add eax, [ebp + 12]
/* Восстановление стек-фрейма вызывающей функции */
mov esp, ebp
pop ebp
/* Выход из функции */
ret
```