МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования «Санкт-Петербургский политехнический университет

Петра Великого»

Институт компьютерных наук и кибербезопасности

Направление: 02.03.01 Математика и компьютерные науки

Основы архитектуры ЦВМ

**Отчет о выполнении лабораторной работы №7**

Команды и способы адресации для х86 в 32-битном режиме

Студент,

группы 5130201/30002 \_\_\_\_\_\_\_\_\_\_\_\_\_ Филиппов Г. М.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_ Вербова Н. М.

Санкт-Петербург - 2024 г.

# Задачи работы

Познакомиться с системой процессорных команд и способами использования этих команд.

**Методика**:

1) взять фрагмент программы на языке Си, заданный преподавателем, или написать такой фрагмент самостоятельно по заданной спецификации.

2) оттранслировать программу содержащую этот фрагмент

3) перейти в режим отладки и изучить дизассемблированный код фрагмента

а. Система команд

1. Набор команд и их использование

2. Кодирование: длина и содержимое полей

b. Способы адресации

1. Непосредственные константы – способы задания

2. Обращение по абсолютному адресу – как это организовать

3. Косвенно-регистровая адресация – в каких ситуациях используется (например, обращение к элементам массива)

4. Многокомпонентная адресация – в каких ситуациях используется

5. Адресация в командах переходов – какие способы реализованы в архитектуре х86

# Листинг. Исходный код на высокоуровневом языке

Исходный код:

unsigned char someChar;

void Fn(char A, int statInt, unsigned char \*someChar);

void main() {

static int statInt = 0;

char A = 'A';

Fn(A, statInt, &someChar);

}

void Fn(char A, int statInt, unsigned char \*someChar) {

statInt += A \* 3;

A = 'B' + 4;

\*someChar = A + statInt;

}

# Дизассемблированный код

1: unsigned char someChar;

2:

3: void Fn(char A, int statInt, unsigned char \*someChar);

4:

5: void main() {

0x00000000 <+0>: lea 0x4(%esp),%ecx // Резервируем 4 байта для дальнейшей

// работы с локальными переменными (привязанными к началу функции)

0x00000004 <+4>: and $0xfffffff0,%esp // Выравниваем значение адреса на 16

// байт - для того, чтобы соответствовать архитектуре x86

0x00000007 <+7>: push -0x4(%ecx) // Помещаем в стек неизменное

// значение адреса, чтобы после вызова подфункции мы не потеряли

// наши локальные переменные

0x0000000a <+10>: push %ebp // Создаем еще один фрейм стека

// (устанавливаем новый базовый указатель)

0x0000000b <+11>: mov %esp,%ebp // Сохраняем значение esp,

// чтобы не потерять его

0x0000000d <+13>: push %ecx // Создаем еще один фрейм стека, для

// дальнейшего использования ecx и возврату к его использованию

0x0000000e <+14>: sub $0x14,%esp // Относительно регистра esp

// резервируем 20 байт памяти (отодвигая адрес на 20 байт назад)

0x00000011 <+17>: call 0x12 <main+18> // Вероятнее всего это вызов какой-то

// внутр. подфункции, которая инициализирует глобальные переменные

0x00000016 <+22>: add $0x1,%eax // Вероятнее всего это адрес,

// полученный в результату нашей подфункции + 1

6: static int statInt = 0;

7: char A = 'A';

0x0000001b <+27>: movb $0x41,-0x9(%ebp) // Записываем в нашу локальную

// переменную A значение ‘A’ в кодах ASCII

8: Fn(A, statInt, &someChar);

0x0000001f <+31>: mov 0x4(%eax),%ecx // Загружаем 4 байта, начиная с адреса

// %eax, в регистр %ecx. В нашем случае это адрес statInt

0x00000025 <+37>: movsbl -0x9(%ebp),%edx // Загружаем байт по адресу %ebp - 9

// (значение 'A') и расширяет его в 32-битное значение в регистр %edx.

0x00000029 <+41>: sub $0x4,%esp // Резервирует 4 байта для аргумента

0x0000002c <+44>: lea 0x0(%eax),%eax // Загружаем адрес из eax в сам регистр

0x00000032 <+50>: push %eax // Сохраняем текущее значение %eax на

// стек, чтобы использовать в дальнейшей функции

0x00000033 <+51>: push %ecx // Аналогично прошлому

0x00000034 <+52>: push %edx // Аналогично прошлому

0x00000035 <+53>: call 0x36 <main+54> // Вызываем нашу функцию Fn,

// сохраняя при этом адрес следующей команды, чтобы в дальнейшем с

// помощью ret вернуться к метке <main+54>

0x0000003a <+58>: add $0x10,%esp // Увеличиваем значение стека на 16

// байт, что соответствует освобождению трех параметров, записанных

// ранее на стек

9: }

0x0000003d <+61>: nop // Заглушка

0x0000003e <+62>: mov -0x4(%ebp),%ecx // Возвращаемся к нашему начальному

// указателю на функцию в 32-ух битах

0x00000041 <+65>: leave // Восстанавливаем базовый указатель

0x00000042 <+66>: lea -0x4(%ecx),%esp // Перемещаем еще раз на 4 стека

// вниз, чтобы вернуться к самому началу

0x00000045 <+69>: ret // Возвращаемся к самому началу,

// выходя на верхний уровень стека

11: void Fn(char A, int statInt, unsigned char \*someChar) {

0x00000046 <+0>: push %ebp // Сохраняем значение регистра базового

// указателя (%ebp) на стеке. Это нужно для восстановления состояния

// стека при выходе из функции.

0x00000047 <+1>: mov %esp,%ebp // Установим базовый указатель (%ebp) на

// текущее значение указателя стека (%esp). Это позволяет работать с

// локальными переменными и параметрами функции через смещения

0x00000049 <+3>: sub $0x4,%esp // Выделим 4 байта на стеке для локальной

// переменной. Это будет место для хранения значения, которое будет

// использоваться в дальнейших вычислениях.

0x0000004c <+6>: call 0x4d <Fn+7> // Здесь вызывается другая функция

// (адрес 0x4d) — возможно, это вспомогательная функция. Результат

// вызова будет автоматически помещен в регистр %eax

0x00000051 <+11>: add $0x1,%eax // Увеличиваем результат, полученный от

// предыдущего вызова функции, на 1, помещая его обратно в %eax.

0x00000056 <+16>: mov 0x8(%ebp),%eax // Загружаем значение второго

// параметра (statInt) в регистр %eax.

0x00000059 <+19>: mov %al,-0x4(%ebp) // Сохраняем младший байт результата

// (значение A, приведенное к unsigned char) по адресу -0x4(%ebp), что

// соответствует выделенной локальной переменной

12: statInt += A \* 3;

0x0000005c <+22>: movsbl -0x4(%ebp),%edx // Загружаем значение из локальной

// переменной (значение A, сохраненное ранее) в регистр %edx с

// расширением знака для 32-битного представления.

0x00000060 <+26>: mov %edx,%eax // Копируем значение из %edx в %eax,

// чтобы далее работать с ним.

0x00000062 <+28>: add %eax,%eax // Умножаем регистр eax на 2

0x00000064 <+30>: add %edx,%eax // Добавляем начальное значение для

// итогового умножения на 3

0x00000066 <+32>: add %eax,0xc(%ebp) // Добавляем результат к statInt,

// который находится по адресу 0xc(%ebp).

13: A = 'B' + 4;

0x00000069 <+35>: movb $0x46,-0x4(%ebp) // Здесь 0x46 — это код буквы 'B' с

// добавлением 4, т.е. ‘F’. Этот код сохраняет ‘F’ в локальную

// переменную -0x4(%ebp)

14: \*someChar = A + statInt;

0x0000006d <+39>: mov 0xc(%ebp),%eax // Загружаем значение переменной

// statInt, которая находится по адресу 0xc(%ebp), в регистр %eax.

// Здесь мы получаем текущее значение statInt.

0x00000070 <+42>: mov %eax,%edx // Копируем значение statInt, которое

// сейчас в %eax, в %edx. Далее мы будем использовать его для

// дальнейших вычислений.

0x00000072 <+44>: movzbl -0x4(%ebp),%eax // Загружаем значение переменной A // в -0x4(%ebp), при этом происходит нулевая раскладка (zero-extend)

// младшего байта в 32-битное представление.

0x00000076 <+48>: add %eax,%edx // Добавляем значение A к значению

// statInt, которое уже находится в %edx.

0x00000078 <+50>: mov 0x10(%ebp),%eax // Загружаем адрес указателя someChar,

// который находится по адресу 0x10(%ebp) в регистр %eax.

0x0000007b <+53>: mov %dl,(%eax) // Записываем конечное значение в

// память по адресу eax. Регистры LD|DL используются для хранения

// младшего байта, который теперь содержит результат

15: }

0x0000007d <+55>: nop // Заглушка

0x0000007e <+56>: leave // Восстанавливаем базовый указатель

0x0000007f <+57>: ret // Возвращаемся к самому началу,

// выходя на верхний уровень стека