

**Виктор Вацаев Андреевич**

**Лабораторная**

Вацаев Виктор Андреевич

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>8</b>
<b>4</b>	<b>Выводы</b>	<b>10</b>

## **Список иллюстраций**

## **Список таблиц**

# 1 Цель работы

Изучение команд условного и безусловного переходов. Приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга. # Задание Задание 1 Создаем каталог для программ лабораторной работы № 7, переходим в него и создаем файл lab7-1.asm. Задание 2 Рассматриваем пример программы с использованием инструкции jmp. Вводим в файл lab7-1.asm текст программы из листинга 7.1. Ошибочно написал jump) задание 3: Создаем файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. Изучаем текст программы из листинга 7.3 и вводим его в lab7-2.asm. Задание 4: Создаем файл листинга для программы из файла lab7-2.asm. Открываем файл lab7-2.asm, удаляем один операнд в любой инструкции с двумя операндами. Выполняем трансляцию программы для получения файла листинга. разберём три строки по выбору из листинга, чтобы понять, что в них происходит.

Выбранные строки:

1 Строка 8: cmp byte [eax], 0 2 Строка 14: sub eax, ebx 3 Строка 35: int 80h

1 Строка 8: cmp byte [eax], 0

Описание:

Это команда сравнения (compare). Она сравнивает содержимое байта, на который указывает регистр eax, с числом 0. Регистр eax указывает на текущий символ строки. Если текущий символ равен 0 (конец строки), флаг результата операции обновляется.

Цель:

Проверить, достиг ли указатель конца строки. В строках, используемых в ас-

семблере, конец строки обозначается символом с кодом 0 (null-терминатор).

2 Строка 14: `sub eax, ebx`

Описание:

Команда вычитания. Она вычитает значение регистра `ebx` из регистра `eax`. В данном случае `ebx` содержит начальный адрес строки, а `eax` — адрес конца строки.

Цель:

Вычислить длину строки. После выполнения цикла в функции `slen`, `eax` указывает на конец строки. Разница между `eax` и `ebx` (адрес начала строки) равна количеству символов в строке.

3 Строка 35: `int 80h`

Описание:

Это программный прерывание, используемое для вызова системных функций в Linux. В данном случае: `eax = 4` — номер системного вызова `write` (запись данных). `ebx = 1` — файл, в который производится запись (1 означает стандартный вывод, т.е. экран). `ecx` — адрес данных, которые нужно вывести. `edx` — количество байт для вывода.

Цель:

Напечатать сообщение на экран. Используется для реализации функции вывода строки в функции `sprint`.

Эти строки показывают очень важные аспекты программы!!! проверку конца строки, вычисление длины строки и вывод строки на экран. Задание 4: любое название, ссылка: </home/victor/Pictures/Снимки экрана/4.png> и еще одна ссылка </home/victor/Pictures/Снимки экрана/4.1.png> Самостоятельная работа: требовалось в задание для самостоятельной работы

Написать программу для нахождения наименьшей из трёх целочисленных переменных `aa`, `bb`,

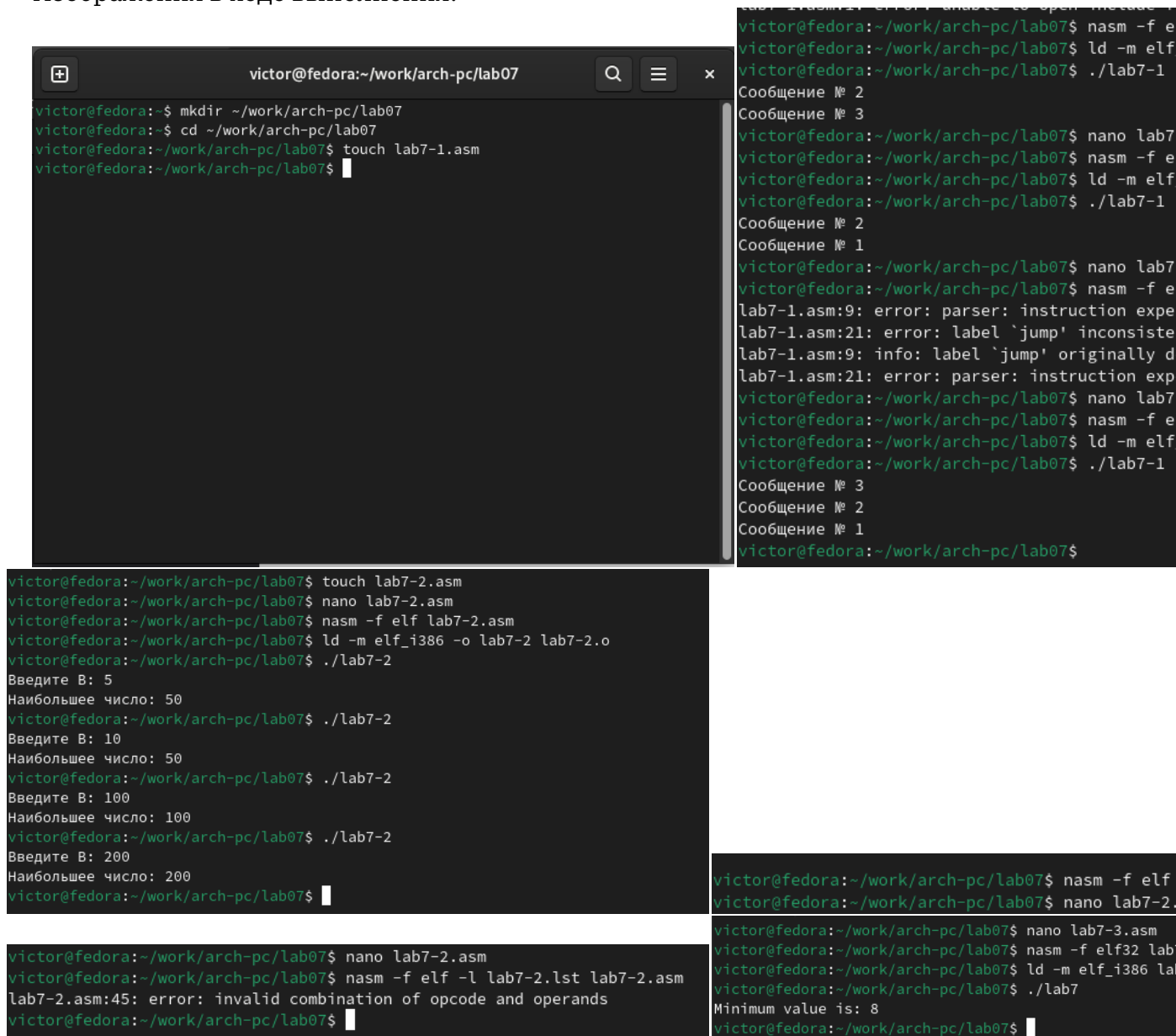
Написать программу, которая принимает с клавиатуры значения `xx` и `aa`, вычисляет значения

## 2 Теоретическое введение

Для реализации ветвлений в ассемблере используются так называемые команды передачи управления или команды перехода. Можно выделить 2 типа переходов: • условный переход – выполнение или не выполнение перехода в определенную точку программы в зависимости от проверки условия. • безусловный переход – выполнение передачи управления в определенную точку программы без каких-либо условий.

## 3 Выполнение лабораторной работы

Изображения в ходе выполнения:



```
victor@fedora:~/work/arch-pc/lab07$ mkdir ~/work/arch-pc/lab07
victor@fedora:~/work/arch-pc/lab07$ cd ~/work/arch-pc/lab07
victor@fedora:~/work/arch-pc/lab07$ touch lab7-1.asm
victor@fedora:~/work/arch-pc/lab07$ nano lab7-1.asm
lab7-1.asm:9: error: parser: instruction expected
lab7-1.asm:9: info: label 'jump' originally defined
lab7-1.asm:21: error: parser: instruction expected
victor@fedora:~/work/arch-pc/lab07$ nano lab7-1.asm
victor@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
victor@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
victor@fedora:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
victor@fedora:~/work/arch-pc/lab07$ nano lab7-2.asm
victor@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
victor@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
victor@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 5
Наибольшее число: 50
victor@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 10
Наибольшее число: 50
victor@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 100
Наибольшее число: 100
victor@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 200
Наибольшее число: 200
victor@fedora:~/work/arch-pc/lab07$ nano lab7-2.asm
victor@fedora:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:45: error: invalid combination of opcode and operands
victor@fedora:~/work/arch-pc/lab07$ nano lab7-3.asm
victor@fedora:~/work/arch-pc/lab07$ nasm -f elf32 lab7-3.asm
victor@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-3.o
victor@fedora:~/work/arch-pc/lab07$ ./lab7-3
Minimum value is: 8
victor@fedora:~/work/arch-pc/lab07$
```



/home/victor/Pictures/Снимки экрана/самостоятельная 2.png

```
victor@fedora:~/work/arch-pc/lab07$ nano lab7-3.a
victor@fedora:~/work/arch-pc/lab07$ nasm -f elf32
victor@fedora:~/work/arch-pc/lab07$ ld -m elf_i386
victor@fedora:~/work/arch-pc/lab07$ ./lab7
Minimum value is: 8
victor@fedora:~/work/arch-pc/lab07$
```

## 4 Выводы

Я научился писать код и команды безусловного и условного переходов в Nasm. Это очень мне поможет в будущей профессии, оказалось не легко. Команды безусловного и условного переходов в Nasm упрощают работу. # Список литературы{unnumbered}