

Российский Университет Дружбы Народов
Факультет физико-математических и естественных наук

Отчет по лабораторной работе №7
“ Команды безусловного и условного
переходов в NASM, программирование ветвлений”

Студентка: Богомолова Полина Петровна

Группа: НКАбд-01-25

Москва, 2025

Оглавление

Цель.....	5
Теоретическое введение.....	6
Команды безусловного перехода.....	6
Таблица 1.....	6
Команды условного перехода.....	7
Регистр флагов.....	7
Описание инструкции <code>str</code>	7
Описание команд условного перехода.....	7
Файл листинга и его структура.....	8
Выполнение лабораторной работы.....	9
Рис. 1.....	9
Рис. 2.....	9
Рис. 3.....	9
Рис. 4.....	10
Рис. 5.....	10
Рис. 6.....	10
Рис. 7.....	11
Рис. 8.....	11
Рис. 9.....	12
Рис. 10.....	13
Рис. 11.....	13
Рис. 12.....	14

Рис. 13	14
Рис. 14	14
Рис. 15	15
Рис. 16	16
Рис. 17	17
Рис. 18	17
Рис. 19	17
Рис. 20	18
Задание для самостоятельной работы	18
Таблица 7.5	19
Рис. 21	19
Рис. 22	19
Рис. 23	20
Таблица 7.6	20
Рис. 24	21
Рис. 25	21
Рис. 26	22
Рис. 27	22
Листинги	23
Листинг 7.1	23
Листинг 7.2	23
Листинг 7.2.2	24
Листинг 7.3	25

Листинг задание 1 для самостоятельной работы	26
Листинг задание 2 для самостоятельной работы	28
Вывод.....	30
Список литературы.....	31

Цель

Изучение команд условного и безусловного переходов. Приобретение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

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.

Команды безусловного перехода

Безусловный переход выполняется инструкцией `jmp` (от англ. `jump` – прыжок), которая включает в себя адрес перехода, куда следует передать управление:

`jmp <адрес_перехода>`

Адрес перехода может быть либо меткой, либо адресом области памяти, в которую предварительно помещен указатель перехода. Кроме того, в качестве операнда можно использовать имя регистра, в таком случае переход будет осуществляться по адресу, хранящемуся в этом регистре (см. таблицу 1)

Таблица 1

Тип операнда	Описание
<code>jmp label</code>	переход на метку <code>label</code>
<code>jmp [label]</code>	переход по адресу в памяти, помеченному меткой <code>label</code>
<code>jmp eax</code>	переход по адресу из регистра <code>eax</code>

Команды условного перехода

Как отмечалось выше, для условного перехода необходима проверка какого-либо условия. В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра флагов.

Регистр флагов

Флаг – это бит, принимающий значение 1 («флаг установлен»), если выполнено некоторое условие, и значение 0 («флаг сброшен») в противном случае. Флаги работают независимо друг от друга, и лишь для удобства они помещены в единый регистр — регистр флагов, отражающий текущее состояние процессора

Описание инструкции `сmp`

Инструкция `сmp` является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Инструкция `сmp` является командой сравнения двух операндов и имеет такой же формат, как и команда вычитания: `сmp , Команда сmp`, так же как и команда вычитания, выполняет вычитание -, но результат вычитания никуда не записывается и единственным результатом команды сравнения является формирование флагов.

Описание команд условного перехода

Команда условного перехода имеет вид:

`j < Мнемоника перехода> label`

Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов. В таблицах 2-3 представлены инструкции условной передачи управления по результатам арифметического сравнения `сmp a,b`

Файл листинга и его структура

Листинг (в рамках понятийного аппарата NASM) — это один из выходных файлов, создаваемых транслятором. Он имеет текстовый вид и нужен при отладке программы, так как кроме строк самой программы он содержит дополнительную информацию.

Структура листинга:

- номер строки — это номер строки файла листинга (нужно помнить, что номер строки в файле листинга может не соответствовать номеру строки в файле с исходным текстом программы);
- адрес — это смещение машинного кода от начала текущего сегмента;
- машинный код представляет собой ассемблированную исходную строку в виде шестнадцатеричной последовательности. (например, инструкция `int 80h` начинается по смещению `00000020` в сегменте кода; далее идёт машинный код, в который ассемблируется инструкция, то есть инструкция `int 80h` ассемблируется в `CD80` (в шестнадцатеричном представлении); `CD80` — это инструкция на машинном языке, вызывающая прерывание ядра);
- исходный текст программы — это просто строка исходной программы вместе с комментариями (некоторые строки на языке ассемблера, например, строки, содержащие только комментарии, не генерируют никакого машинного кода, и поля «смещение» и «исходный текст программы» в таких строках отсутствуют, однако номер строки им присваивается).

Выполнение лабораторной работы

1. Создадим каталог для программ лабораторной работы № 7, перейдем в него и создадим файл lab7-1.asm. Для этого будем использовать команды mkdir, cd, touch. Результат представлен на рисунке 1. Подготовимся к выполнению работы, предварительно скопировав файл in_out.asm в каталог лабораторной работы номер 7, с помощью команды cp. Результат представлен на рисунке 2.

Рис. 1

```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~$ mkdir ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~$ cd ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ touch lab7-1.asm
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$
```

Рис. 2

```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~$ cd ~/Загрузки
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/Загрузки$ cp in_out.asm ~/work/arch-pc/lab07/
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/Загрузки$ cd ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ls
in_out.asm  lab7-1.asm
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$
```

2. Инструкция jmp в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции jmp. Введем в файл lab7-1.asm текст программы из листинга 7.1. Для работы с файлами будем использовать Midnight Commander, для его запуска будем использовать команду mc. Создадим исполняемый файл и запустим его. Для этого будем использовать команды nasm -f elf, ld -m elf_i386, ./ .Результат представлен на рисунках 3-6.

Рис. 3

```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ mc
```

Рис. 4

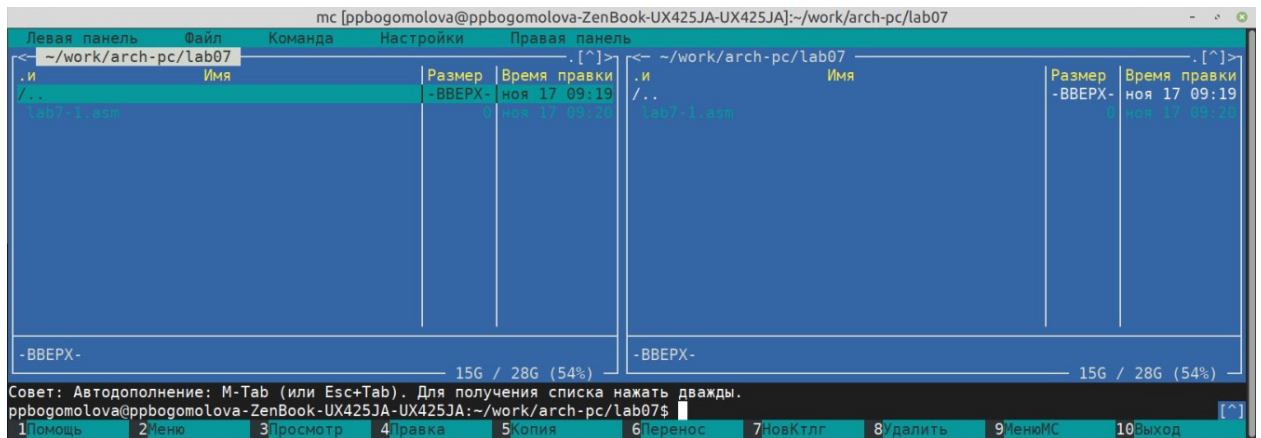


Рис. 5

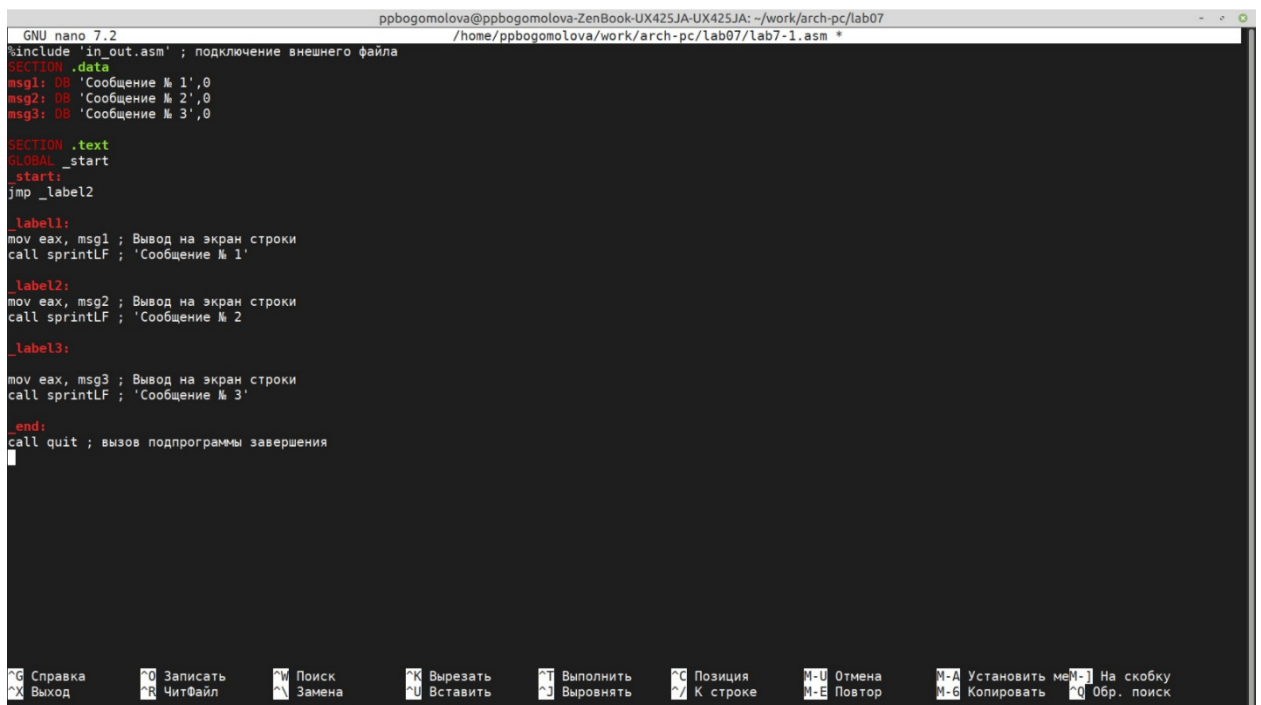
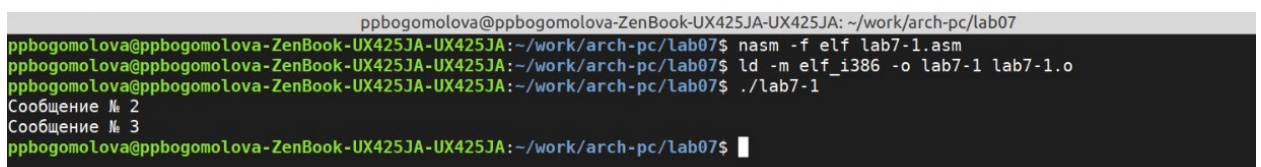


Рис. 6



Таким образом, использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения. Инструкция `jmp` позволяет осуществлять переходы не только вперед, но и назад.

Изменим программу таким образом, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу. Для этого в текст программы после вывода сообщения № 2 добавим инструкцию `jmp` с меткой `_label1` (т.е. переход к инструкциям вывода сообщения № 1) и после вывода сообщения № 1 добавим инструкцию `jmp` с меткой `_end` (т.е. переход к инструкции `call quit`). Изменим текст программы в соответствии с листингом 7.2. Создадим исполняемый файл и проверим его работу, используя команды `nasm -f elf, ld -m elf_i386, ./`. Результат представлен на рисунках 7-8.

Рис. 7

```
GNU nano 7.2 ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
#include "in_out.asm" ; подключение внешнего файла

SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0

SECTION .text
GLOBAL _start
_start:

jmp _label2

_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
jmp _end

_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
jmp _label1

_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'

_end:
call quit ; вызов подпрограммы завершения
```

Рис. 8

```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$
```

- Использование инструкции `jmp` приводит к переходу в любом случае. Однако, часто при написании программ необходимо использовать условные переходы, т.е. переход должен происходить если выполнено какое-либо условие. В качестве примера рассмотрим программу, которая определяет и выводит на

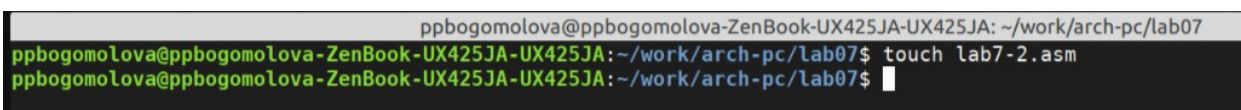
экран наибольшую из 3 целочисленных переменных: А, В и С. Значения для А и С задаются в программе, значение В вводится с клавиатуры.

Создадим файл lab7-2.asm в каталоге ~/work/arch-pc/lab07 с помощью команды touch. Внимательно изучим текст программы из листинга 7.3 и введем его в lab7-2.asm.

Создадим исполняемый файл и проверим его работу для разных значений В, используя команды `nasm -f elf, ld -m elf_i386, ./`. Результат представлен на рисунках 9 - 13.

Обратим внимание на то, что в данном примере переменные А и С сравниваются как символы, а переменная В и максимум из А и С как числа (для этого используется функция `atoi` преобразования символа в число). Это сделано для демонстрации сравнения данных. Данную программу можно упростить и сравнивать все 3 переменные как символы (т.е. не использовать функцию `atoi`). Однако если переменные преобразовать из символов числа, над ними можно корректно проводить арифметические операции.

Рис. 9



```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ touch lab7-2.asm
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ █
```

Рис. 10

```

GNU nano 7.2
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
/home/ppbogomolova/work/arch-pc/lab07/lab7-2.asm *
#include "in_out.asm"
section .data
msg1 db 'Введите B: ',0h
msg2 db "Наибольшее число: ",0h
A dd '20'
C dd '50'
section .bss
max resb 10
B resb 10
section .text
global _start
_start:
; ----- Вывод сообщения 'Введите B: '
mov eax,msg1
call sprint
; ----- Ввод 'B'
mov ecx,B
mov edx,10
call sread
; ----- Преобразование 'B' из символа в число
mov eax,B
call atoi ; Вызов подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в 'B'
; ----- Записываем 'A' в переменную 'max'
mov ecx,[A] ; 'ecx = A'
mov [max],ecx ; 'max = A'
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jg check_B ; если 'A>C', то переход на метку 'check_B',
mov ecx,[C] ; иначе 'ecx = C'
mov [max],ecx ; 'max = C'
; ----- Преобразование 'max(A,C)' из символа в число
check_B:
mov eax,max
call atoi ; Вызов подпрограммы перевода символа в число
mov [max],eax ; запись преобразованного числа в 'max'
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx,[max]
cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
; ----- Вывод результата
fin:
mov eax,msg2
call sprint ; Вывод сообщения 'Наибольшее число: '
mov eax,[max]
call iprintf ; Вывод 'max(A,B,C)'
call quit ; Выход

```

Рис. 11

```

GNU nano 7.2
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
/home/ppbogomolova/work/arch-pc/lab07/lab7-2.asm *
_start:
; ----- Вывод сообщения 'Введите B: '
mov eax,msg1
call sprint
; ----- Ввод 'B'
mov ecx,B
mov edx,10
call sread
; ----- Преобразование 'B' из символа в число
mov eax,B
call atoi ; Вызов подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в 'B'
; ----- Записываем 'A' в переменную 'max'
mov ecx,[A] ; 'ecx = A'
mov [max],ecx ; 'max = A'
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jg check_B ; если 'A>C', то переход на метку 'check_B',
mov ecx,[C] ; иначе 'ecx = C'
mov [max],ecx ; 'max = C'
; ----- Преобразование 'max(A,C)' из символа в число
check_B:
mov eax,max
call atoi ; Вызов подпрограммы перевода символа в число
mov [max],eax ; запись преобразованного числа в 'max'
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx,[max]
cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
jg fin ; если 'max(A,C)>B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'
mov [max],ecx
; ----- Вывод результата
fin:
mov eax,msg2
call sprint ; Вывод сообщения 'Наибольшее число: '
mov eax,[max]
call iprintf ; Вывод 'max(A,B,C)'
call quit ; Выход

```


Рис. 12

```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 1
Наибольшее число: 50
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 2
Наибольшее число: 50
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 3
Наибольшее число: 50
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 4
Наибольшее число: 50
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 22
Наибольшее число: 50
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ █
```

Рис. 13

```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 50
Наибольшее число: 50
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 77
Наибольшее число: 77
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 999
Наибольшее число: 999
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ █
```

4. Обычно `nasm` создаёт в результате ассемблирования только объектный файл. Получить файл листинга можно, указав ключ `-l` и задав имя файла листинга в командной строке. Создадим файл листинга для программы из файла `lab7-2.asm` с помощью команды `nasm -f elf -l lab7-2.lst lab7-2.asm`. Откроем файл листинга `lab7-2.lst` с помощью любого текстового редактора `mcedit`. Результат представлен на рисунках 14-15

Рис. 14

```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ mcedit lab7-2.lst
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ █
```

```

/home/ppbogomolova/work/arch-pc/lab07/lab7-2.lst 6 L: 1+ 0 1/225] *(6 /14458b) 0032 0x020 [*][X]
1      ;include 'in_out.asm'
2      ;-----
3      ; Функция вычисления длины сообщения
4      ; slen:
5      00000000 53      <1>      push     ebx
6      00000001 89C3    <1>      mov      ebx, eax
7      ; nextchar:
8      00000003 803800    <1>      cmp      byte [eax], 0
9      00000006 7403    <1>      jz       finished
10     00000008 40      <1>      inc      eax
11     00000009 EBF8    <1>      jmp      nextchar
12     ; finished:
13     0000000B 29D8    <1>      sub      eax, ebx
14     0000000D 5B      <1>      pop      ebx
15     0000000E C3      <1>      ret
16     ;-----
17     ; sprintf
18     ; Функция печати сообщения
19     ; входные данные: mov eax, <message>
20     ; sprintf:
21     0000000F 52      <1>      push     edx
22     00000010 51      <1>      push     ecx
23     00000011 53      <1>      push     ebx
24     00000012 50      <1>      push     eax
25     00000013 E8E8FFFFFF <1>      call    slen
26     ;
27     00000018 89C2    <1>      mov      edx, eax
28     0000001A 58      <1>      pop      eax
29     ;
30     0000001B 89C1    <1>      mov      ecx, eax
31     0000001D B801000000 <1>      mov      ebx, 1
32     00000022 B804000000 <1>      mov      eax, 4
33     00000027 CD80    <1>      int      80h
34     ;
35     00000029 58      <1>      pop      ebx
36     0000002A 59      <1>      pop      ecx
37     0000002B 5A      <1>      pop      edx
38     0000002C C3      <1>      ret

```

В листинге ассемблера каждая строка состоит из 3 частей: 1 часть – адрес инструкции (к примеру, 00000001), 2 часть – машинный код (к примеру, 89 C3), 3 часть - ассемблерная команда (к примеру, mov ebx, eax). По адресу будет находиться первая инструкция байта. Адрес нужен, чтобы понимать, куда будет переходить программа при командах jmp, call и т.д. Машинный код – это те байты, которые выполняет процессор, каждая команда имеет свое байтовое представление. Ассемблерная команда – это та же команда из машинного кода, имеющая понятный и читабельный вид для человека. Рассмотрим строку 5. Команда mov ebx, eax копирует значение из регистра eax в регистр ebx. Рассмотрим строку 35. Рассмотрим строку int 80h. Команда выполняет программное прерывание с номером 0x80. Это механизм обращения программы к ядру Linux для выполнения системных вызовов. Рассмотрим строку 84. Команда inc ecx увеличивает содержимое регистра ecx на 1. Рассмотрим строку 71. Команда ret завершает выполнение функции и возвращает управление туда, откуда эта функция была вызвана.

Откроем файл с программой lab7-2.asm и в инструкции с двумя операндами удалим один операнд. Выполним трансляцию с получением файла листинга:

```
nasm -f elf -l lab7-2.lst lab7-2.asm
```

При трансляции с получением файла листинга мы получаем ошибку. Создается только файл листинга, объектный файл не создается, так как есть синтаксическая ошибка. Файл листинга будет содержать пометку ошибки на проблемной строке.

Результат представлен на рисунках 16-20

Рис. 16



```
GNU nano 7.2
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
/home/ppbogomolova/work/arch-pc/lab07/lab7-2.asm

_start:
; ----- Вывод сообщения 'Введите B: '
mov eax,msg1
call sprint
; ----- Ввод 'B'
mov ecx,B
mov edx,10
call sread
; ----- Преобразование 'B' из символа в число
mov eax,B
call atoi ; Вызов подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в 'B'
; ----- Записываем 'A' в переменную 'max'
mov ecx,[A] ; 'ecx = A'
mov [max],ecx ; 'max = A'
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jg check_B ; если 'A>C', то переход на метку 'check_B',
mov ecx,[C] ; иначе 'ecx = C'
mov [max],ecx ; 'max = C'
; ----- Преобразование 'max(A,C)' из символа в число
check_B:
mov eax,max
call atoi ; Вызов подпрограммы перевода символа в число
mov [max],eax ; запись преобразованного числа в 'max'
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx,[max]
cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
jg fin ; если 'max(A,C)>B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'
mov [max],ecx
; ----- Вывод результата
fin:
mov eax, msg2
call sprint ; Вывод сообщения 'Наибольшее число: '
mov eax,[max]
call iprintf ; Вывод 'max(A,B,C)'
call quit ; Выход

^G Справка      ^O Записать    ^W Поиск      ^X Вырезать   ^I Выполнить  ^C Позиция    ^U Отмена     ^M Установить мен- ^J На скобку
^X Выход        ^R ЧитФайл    ^N Замена     ^U Вставить   ^D Выводить   ^_ К строке   ^E Повтор     ^B Установить мен- ^Q Обр. поиск
```


Рис. 17

```

GNU nano 7.2                                ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
/home/ppbogomolova/work/arch-pc/lab07/lab7-2.asm *
; ----- Вывод сообщения 'Введите B: '
_start:
mov eax,msg1
call sprint
; ----- Ввод 'B'
mov ecx,B
mov edx,10
call sread
; ----- Преобразование 'B' из символа в число
mov eax,B
call atoi ; Вызов подпрограммы перевода символа в число
mov [max] ; запись преобразованного числа в 'B'
; ----- Записываем 'A' в переменную 'max'
mov ecx,[A] ; 'ecx = A'
mov [max],ecx ; 'max = A'
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jg check_B ; если 'A>C', то переход на метку 'check_B',
mov ecx,[C] ; иначе 'ecx = C'
mov [max],ecx ; 'max = C'
; ----- Преобразование 'max(A,C)' из символа в число
check_B:
mov eax,max
call atoi ; Вызов подпрограммы перевода символа в число
mov [max],eax ; запись преобразованного числа в 'max'
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx,[max]
cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
jg fin ; если 'max(A,C)>B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'
mov [max],ecx
; ----- Вывод результата
fin:
mov eax,msg2
call sprint ; Вывод сообщения 'Наибольшее число: '
mov eax,[max]
call iprintf ; Вывод 'max(A,B,C)'
call quit ; Выход

```

Рис. 18

```

ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:23: error: invalid combination of opcode and operands
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ls
in_out.asm lab7-1 lab7-1.asm lab7-1.o lab7-2 lab7-2.asm lab7-2.lst
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$

```

Рис. 19

```

ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ mcedit lab7-2.lst

```

```

ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
/home/ppbogomolova/work/arch-pc/lab07/lab7-3.asm [8---] 90 L: [184+15 199/226] *(12375/14545b) 0010 0x00A [*][X]
 9 0000000A <res Ah>      B resb 10
10                                section .text
11                                global _start
12                                _start:
13                                ; ----- Вывод сообщения 'Введите B: '
14 000000E8 B8[00000000]    mov eax,msg1
15 000000ED E81DFFFFFF    call sprint
16                                ; ----- Ввод 'B'
17 000000F2 B9[0A000000]    mov ecx,B
18 000000F7 BA0A000000    mov edx,10
19 000000FC E842FFFFFF    call sread
20                                ; ----- Преобразование 'B' из символа в число
21 00000101 B8[0A000000]    mov eax,B
22 00000106 E891FFFFFF    call atoi ; Вызов подпрограммы перевода символа в число
23 0000010B E891FFFFFF    mov eax ; запись преобразованного числа в 'B'
24                                ***** error: invalid combination of opcode and operands *****
25                                ; ----- Записываем 'A' в переменную 'max'
26 0000010B 8B0D[35000000]    mov ecx,[A] ; 'ecx = A'
27 00000111 890D[00000000]    mov [max],ecx ; 'max = A'
28                                ; ----- Сравниваем 'A' и 'C' (как символы)
29 00000117 3B0D[39000000]    cmp ecx,[C] ; Сравниваем 'A' и 'C'
30 0000011D 7F0C            jg check_B ; если 'A>C', то переход на метку 'check_B',
31 0000011F 8B0D[39000000]    mov ecx,[C] ; иначе 'ecx = C'
32 00000125 890D[00000000]    mov [max],ecx ; 'max = C'
33                                ; ----- Преобразование 'max(A,C)' из символа в число
34 0000012B B8[00000000]    mov eax,max
35 00000130 E867FFFFFF    call atoi ; Вызов подпрограммы перевода символа в число
36 00000135 A3[00000000]    mov [max],eax ; запись преобразованного числа в 'max'
37                                ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
38 0000013A 8B0D[00000000]    mov ecx,[max]
39 00000140 3B0D[0A000000]    cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
40 00000146 7F0C            jg fin ; если 'max(A,C)>B', то переход на 'fin',
41 00000148 8B0D[0A000000]    mov ecx,[B] ; иначе 'ecx = B'
42 0000014E 890D[00000000]    mov [max],ecx
43                                ; ----- Вывод результата
44                                fin:
45 00000154 B8[13000000]    mov eax,msg2
46 00000159 E8B1FFFFFF    call sprint ; Вывод сообщения 'Наибольшее число: '
47 0000015E A1[00000000]    mov eax,[max]
48 00000163 E81EFFFFFF    call iprintf ; Вывод 'max(A,B,C)'

```

Задание для самостоятельной работы

1. Напишите программу нахождения наименьшей из 3 целочисленных переменных a , b и c . Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу.

Создадим файл lab7-3.asm с помощью команды touch. Будем использовать Midnight Commander для работы с файлами, запускаем его с помощью команды mc. Создадим исполняемый файл и запустим его. Для этого будем использовать команды `nasm -f elf, ld -m elf_i386, ./`. Значение переменных возьмем из таблицы 7.5. Возьмем значения переменных из варианта 3, который был получен во время выполнения лабораторной работы номер 6 (в лабораторной работе номер 7 вариант получить мы не могли ни в одном задании). Результат представлен на рисунках 21-23.

Таблица 7.5

Номер варианта	Значения a, b, c	Номер варианта	Значения a, b, c
1	17,23,45	11	21,28,34
2	82,59,61	12	99,29,26
3	94,5,58	13	84,32,77
4	8,88,68	14	81,22,72
5	54,62,87	15	32,6,54
6	79,83,41	16	44,74,17
7	45,67,15	17	26,12,68
8	52,33,40	18	83,73,30
9	24,98,15	19	46,32,74
10	41,62,35	20	95,2,61

Рис. 21

```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ touch lab7-3.asm
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$
```

Рис. 22

```
GNU nano 7.2 ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
/home/ppbogomolova/work/arch-pc/lab07/lab7-3.asm *
#include "in_out.asm"

section .data
a dd 94
b dd 5
c dd 58
msg db "Наименьшее число: ", 0

section .bss
min resd 1

section .text
global _start

_start:
mov eax, [a]
mov [min], eax

mov eax, [b]
cmp eax, [min]
jge check_c
mov [min], eax

check_c:
mov eax, [c]
cmp eax, [min]
jge print_result
mov [min], eax

print_result:
mov eax, msg
call sprint

mov eax, [min]
call iprintf
call quit

Сохранить изменённый буфер?
Y Да
N Нет Отмена
```

```

ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ nasm -f elf lab7-3.asm
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.o
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ./lab7-3
Наименьшее число: 5
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ █

```

2. Напишите программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции $f(x)$ и выводит результат вычислений. Вид функции $f(x)$ выбрать из таблицы 7.6 вариантов заданий в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте его работу для значений x и a из 7.6

Создадим файл lab7-4.asm с помощью команды touch. Будем использовать Midnight Commander для работы с файлами, запускаем его с помощью команды mc.

Таблица 7.6

Номер варианта	Выражение для $f(x)$	(x_1, a_1)	(x_2, a_2)
1	$\begin{cases} 2a - x, & x < a \\ 8, & x \geq a \end{cases}$	(1;2)	(2;1)
2	$\begin{cases} a - 1, & x < a \\ x - 1, & x \geq a \end{cases}$	(5;7)	(6;4)
3	$\begin{cases} 3x, & x = 3 \\ a + 1, & x \neq 3 \end{cases}$	(3;4)	(1;4)
4	$\begin{cases} 2x + a, & a \neq 0 \\ 2x + 1, & a = 0 \end{cases}$	(3;0)	(3;2)
5	$\begin{cases} 2(x - a), & x > a \\ 15, & x \leq a \end{cases}$	(1;2)	(2;1)
6	$\begin{cases} x + a, & x = a \\ 5x, & x \neq a \end{cases}$	(2;2)	(2;1)
7	$\begin{cases} 6a, & x = a \\ a + x, & x \neq a \end{cases}$	(1;1)	(2;1)

Рис. 24

```
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ touch lab7-4.asm
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ls
in_out.asm lab7-1 lab7-1.asm lab7-1.o lab7-2 lab7-2.asm lab7-2.lst lab7-3 lab7-3.asm lab7-3.o lab7-4.asm
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$
```

Рис. 25

```
GNU nano 7.2 ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
/home/ppbogomolova/work/arch-pc/lab07/lab7-4.asm *
#include 'in_out.asm'

section .data
msg_x db 'Введите x: ',0h
msg_a db 'Введите a: ',0h
msg_res db 'f(x) = ',0h

section .bss
x resb 10
a resb 10
res resb 10

section .text
global _start
_start:
mov eax, msg_x
call sprint
mov ecx, x
mov edx, 10
call sread
mov eax, x
call atoi
mov [x], eax

mov eax, msg_a
call sprint
mov ecx, a
mov edx, 10
call sread
mov eax, a
call atoi
mov [a], eax

mov eax, [x]
cmp eax, 3
jne x_ne_ravno3
imul eax, eax, 3
jmp zapis_resultata

Ctrl-S Справка      Ctrl-Z Записать
Ctrl-X Выход        Ctrl-R ЧитФайл
Ctrl-F Поиск        Ctrl-Y Вырезать
Ctrl-B Замена       Ctrl-V Вставить
Ctrl-J Выполнить    Ctrl-N Выровнять
Ctrl-C Позиция      Ctrl-M К строке
Ctrl-U Отмена       Ctrl-O Повтор
Ctrl-A Установить м Ctrl-I На скобку
Ctrl-G Копировать   Ctrl-H Обр. поиск
```

Рис. 26

```

GNU nano 7.2 ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
/home/ppbogomolova/work/arch-pc/lab07/lab7-4.asm *
global _start

_start:
mov eax, msg_x
call sprint
mov ecx, x
mov edx, 10
call sread
mov eax, x
call atoi
mov [x], eax

mov eax, msg_a
call sprint
mov ecx, a
mov edx, 10
call sread
mov eax, a
call atoi
mov [a], eax

mov eax, [x]
cmp eax, 3
jne x_ne_ravno3
imul eax, eax, 3
jmp zapis_resultata

x_ne_ravno3:
mov eax, [a]
add eax, 1

zapis_resultata:
mov [res], eax
mov eax, msg_res
call sprint
mov eax, [res]
call iprintLF
call quit

```

Рис. 27

```

ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab07
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ nasm -f elf lab7-4.asm
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-4 lab7-4.o
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ./lab7-4
Введите x: 3
Введите a: 4
f(x) = 9
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$ ./lab7-4
Введите x: 1
Введите a: 4
f(x) = 5
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab07$

```

Ссылка на мой репозиторий в GitHub:

https://github.com/bogomolova-pp/study_2025-2026_arh-pc

Листинги

Листинг 7.1

```
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label2
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения
```

Листинг 7.2

```
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение № 1',0
```

```

msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label2
_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; 'Сообщение № 1'
jmp _end_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения

```

Листинг 7.2.2

```

#include 'in_out.asm'

SECTION .data
    msg1: DB 'Сообщение № 1'1',0
    msg2: DB 'Сообщение № 2'2',0
    msg3: DB 'Сообщение № 3'3',0

SECTION .text
GLOBAL _start
_start:

    jmp _label3

_label1:

```



```

        mov eax, msg1
        call sprintfLF
        jmp _end

_label2:
        mov eax, msg2
        call sprintfLF
        jmp _label1

_label3:
        mov eax, msg3
        call sprintfLF
        jmp _label2

_end:
        call quit

```

Листинг 7.3

```

%include 'in_out.asm'
section .data
msg1 db 'Введите В: ',0h
msg2 db "Наибольшее число: ",0h
A dd '20'
C dd '50'
section .bss
max resb 10
B resb 10
section .text
global _start
_start:
; ----- Вывод сообщения 'Введите В: '
mov eax,msg1
call sprintf
; ----- Ввод 'В'
mov ecx,B

```

```

mov edx,10
call sread
; ----- Преобразование 'B' из символа в число
mov eax,B
call atoi ; Вызов подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в 'B'
; ----- Записываем 'A' в переменную 'max'
mov ecx,[A] ; 'ecx = A'
mov [max],ecx ; 'max = A'
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jg check_B ; если 'A>C', то переход на метку 'check_B',
mov ecx,[C] ; иначе 'ecx = C'
mov [max],ecx ; 'max = C'
; ----- Преобразование 'max(A,C)' из символа в
число
check_B:
mov eax,max
call atoi ; Вызов подпрограммы перевода символа в число
mov [max],eax ; запись преобразованного числа в 'max'
; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
mov ecx,[max]
cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
jg fin ; если 'max(A,C)>B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'
mov [max],ecx
; ----- Вывод результата
fin:
mov eax, msg2
call sprint ; Вывод сообщения 'Наибольшее число: '
mov eax,[max]
call iprintLF ; Вывод 'max(A,B,C)'
call quit ; Выход

```

**Листинг задания 1 для
самостоятельной работы**

```
%include "in_out.asm"
```

```

section .data
    a dd 94
    b dd 5
    c dd 58
    msg db "Наименьшее число: ", 0

section .bss
    min resd 1

section .text
global _start

_start:
    mov eax, [a]
    mov [min], eax

    mov eax, [b]
    cmp eax, [min]
    jge check_c
    mov [min], eax

check_c:
    mov eax, [c]
    cmp eax, [min]
    jge print_result
    mov [min], eax

print_result:
    mov eax, msg
    call sprint

    mov eax, [min]
    call iprintLF

    call quit

```

**Листинг задания 2 для
самостоятельной
работы**

```
%include 'in_out.asm'

section .data
    msg_x    db 'Введите x: ',0h
    msg_a    db 'Введите a: ',0h
    msg_res  db 'f(x) = ',0h

section .bss
    x        resb 10
    a        resb 10
    res      resb 10

section .text
global _start

_start:
    mov eax, msg_x
    call sprint
    mov ecx, x
    mov edx, 10
    call sread
    mov eax, x
    call atoi
    mov [x], eax

    mov eax, msg_a
    call sprint
    mov ecx, a
    mov edx, 10
    call sread
```

```

    mov eax, a
    call atoi
    mov [a], eax

    mov eax, [x]
    cmp eax, 3
    jne x_ne_ravno3

    imul eax, eax, 3
    jmp zapis_resultata

x_ne_ravno3:
    mov eax, [a]
    add eax, 1

zapis_resultata:
    mov [res], eax

    mov eax, msg_res
    call sprint
    mov eax, [res]
    call iprintLF

call quit

```

Вывод

В результате выполнения лабораторной работы я изучила команды условного и безусловного переходов, приобрела навыки написания программ с использованием переходов, познакомилась с назначением и структурой файла листинга.

Список литературы

1. Демидова А.В – Лабораторная работа №7. Команды безусловного и условного переходов в Nasm. Программирование ветвлений.