

0.1 Front matter

title: “Отчёт по лабораторной работе №7” subtitle: “Дисциплина: архитектура компьютеров и операционные системы” author: “Беспутин Глеб Антонович”

0.2 Generic options

lang: ru-RU toc-title: “Содержание”

0.3 Bibliography

bibliography: bib/cite.bib csl: pandoc/csl/gost-r-7-0-5-2008-numeric.csl

0.4 Pdf output format

toc: true # Table of contents toc-depth: 2 lof: true # List of figures lot: true # List of tables
fontsize: 12pt linestretch: 1.5 papersize: a4 documentclass: scrreprt ## I18n polyglossia
polyglossia-lang: name: russian options: - spelling=modern - babelshorthands=true
polyglossia-otherlangs: name: english ## I18n babel babel-lang: russian babel-otherlangs:
english ## Fonts mainfont: PT Serif romanfont: PT Serif sansfont: PT Sans monofont: PT
Mono mainfontoptions: Ligatures=TeX romanfontoptions: Ligatures=TeX sansfontoptions:
Ligatures=TeX,Scale=MatchLowercase monofontoptions: Scale=MatchLowercase,Scale=0.9
Biblatex biblatex: true biblio-style: “gost-numeric” biblatexoptions: - parenttracker=true
- backend=biber - hyperref=auto - language=auto - autolang=other* - citestyle=gost-
numeric ## Pandoc-crossref LaTeX customization figureTitle: “Рис.” tableTitle: “Таблица”
listingTitle: “Листинг” lofTitle: “Список иллюстраций” lotTitle: “Список таблиц” lolTitle:
“Листинги” ## Misc options indent: true header-includes: -

- **keep figures where there are in the text**
 - **# keep figures where there are in the text**

1 Цель работы

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

2 Задание

1. Реализация переходов в NASM.
2. Изучение структуры файлы листинга.
3. Задания для самостоятельной работы.

3 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.

Безусловный переход выполняется инструкцией `jmp`. Инструкция `cmp` является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Инструкция `cmp` является командой сравнения двух операндов и имеет такой же формат, как и команда вычитания.

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.

4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM

Создаю каталог для программ лабораторной работы № 7, перехожу в него и создаю файл `lab7-1.asm`. (рис. 1).

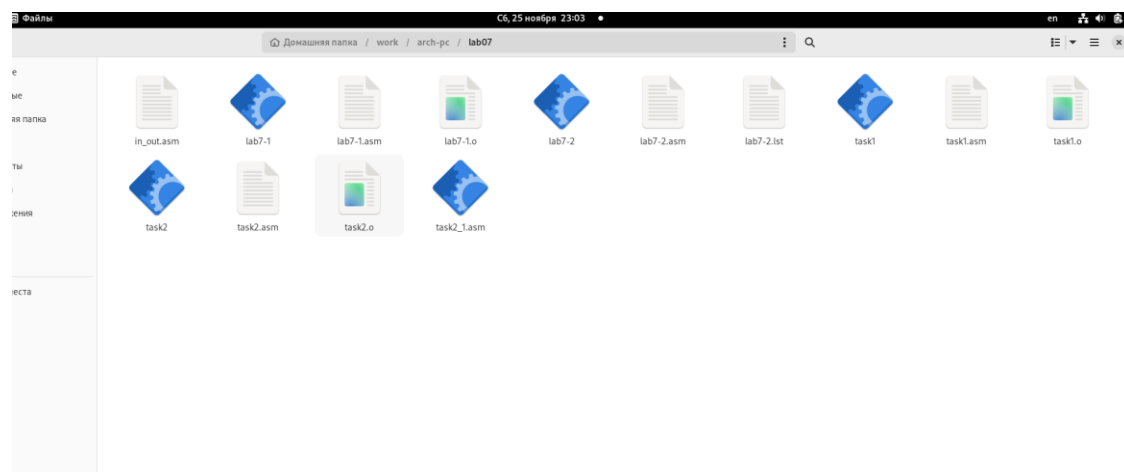
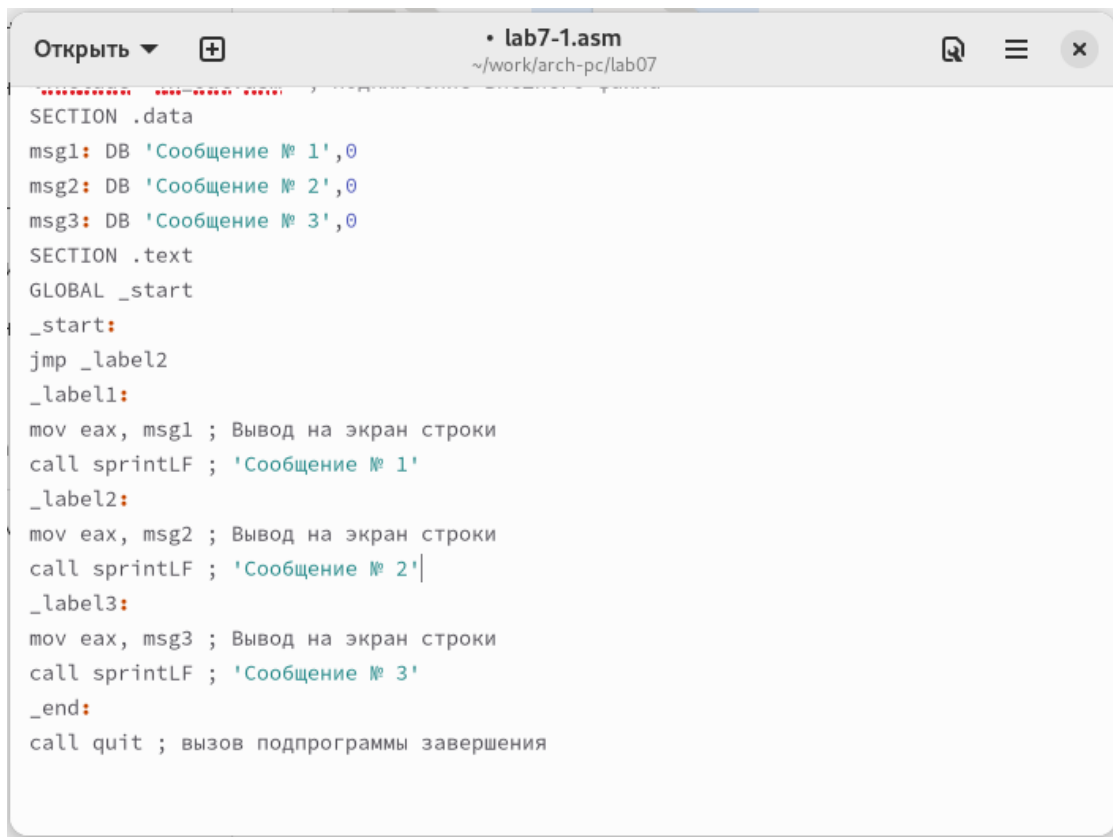


Figure 1: Создание файлов для лабораторной работы

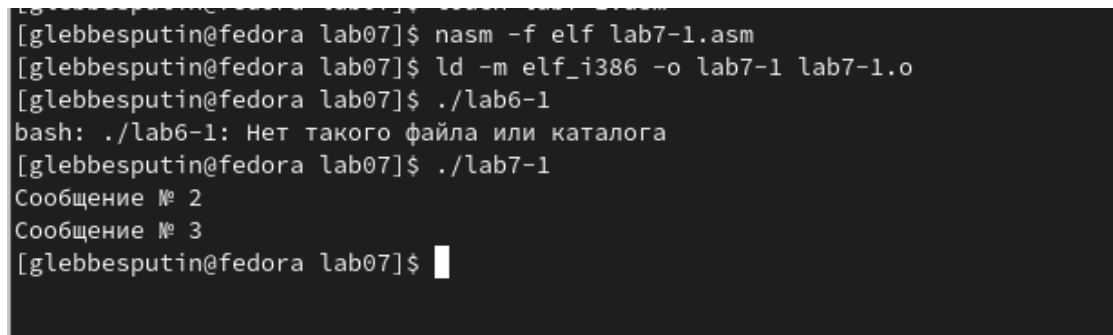
Ввожу в файл `lab7-1.asm` текст программы из листинга 7.1. (рис. 2).



```
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
```

Figure 2: Ввод текста программы из листинга 7.1

Создаю исполняемый файл и запускаю его. (рис. 3).

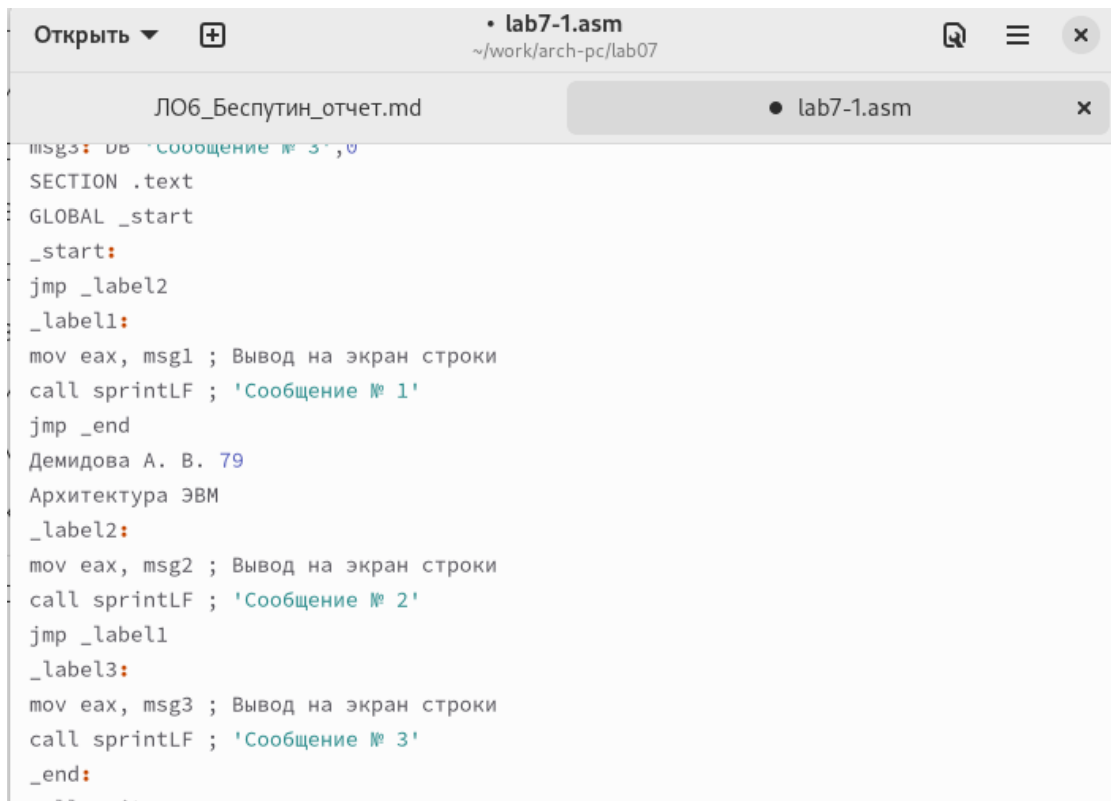


```
[glebbesputin@fedora lab07]$ nasm -f elf lab7-1.asm
[glebbesputin@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[glebbesputin@fedora lab07]$ ./lab6-1
bash: ./lab6-1: Нет такого файла или каталога
[glebbesputin@fedora lab07]$ ./lab7-1
Сообщение № 2
Сообщение № 3
[glebbesputin@fedora lab07]$
```

Figure 3: Запуск программного кода

Таким образом, использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения.

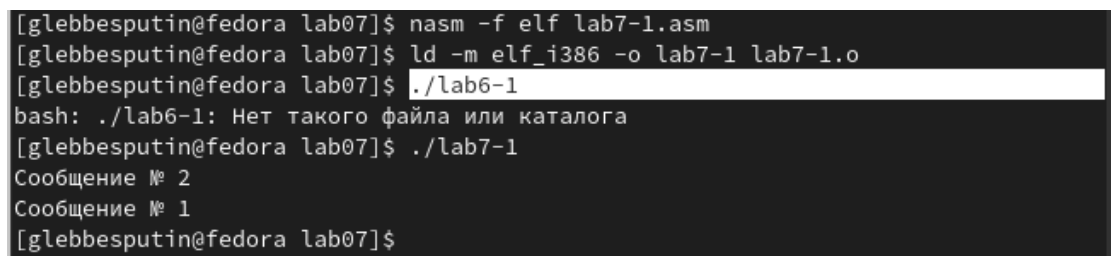
Изменяю программу таким образом, чтобы она выводила сначала 'Сообщение № 2', потом 'Сообщение № 1' и завершала работу. Для этого изменяю текст программы в соответствии с листингом 7.2. (рис. 4).



```
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
Демидова А. В. 79
Архитектура ЭВМ
_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:
```

Figure 4: Изменение текста программы

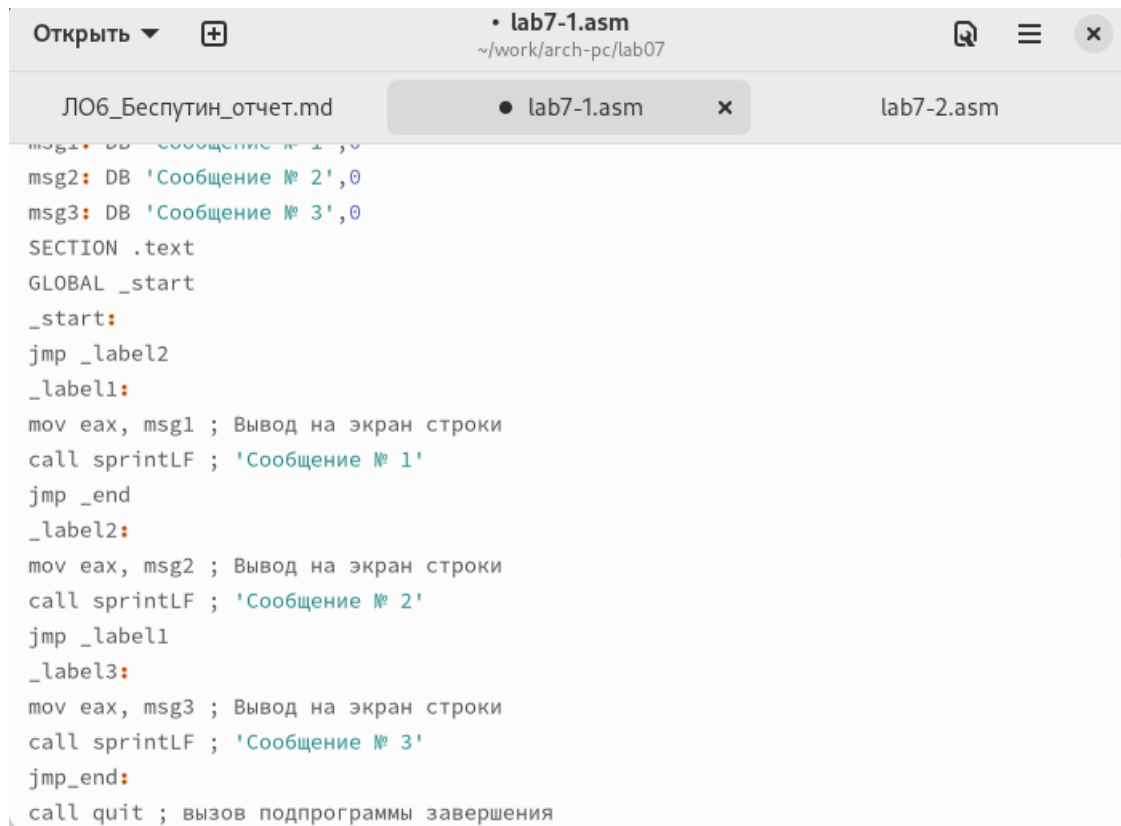
Создаю исполняемый файл и проверяю его работу. (рис. 5).



```
[glebbesputin@fedora lab07]$ nasm -f elf lab7-1.asm
[glebbesputin@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[glebbesputin@fedora lab07]$ ./lab7-1
bash: ./lab7-1: Нет такого файла или каталога
[glebbesputin@fedora lab07]$ ./lab7-1
Сообщение № 2
Сообщение № 1
[glebbesputin@fedora lab07]$
```

Figure 5: Создание исполняемого файла

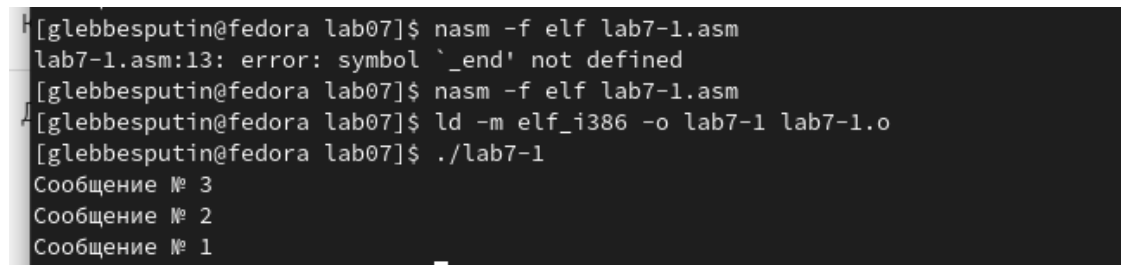
Затем изменяю текст программы, добавив в начале программы `jmp _label3`, `jmp _label2` в конце метки `jmp _label3`, `jmp _label1` добавляю в конце метки `jmp _label2`, и добавляю `jmp _end` в конце метки `jmp _label1`, (рис. 6).



```
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'
jmp_end:
call quit ; вызов подпрограммы завершения
```

Figure 6: Изменение текста программы

чтобы вывод программы был следующим: (рис. 7).



```
[glebbesputin@fedora lab07]$ nasm -f elf lab7-1.asm
lab7-1.asm:13: error: symbol '_end' not defined
[glebbesputin@fedora lab07]$ nasm -f elf lab7-1.asm
[glebbesputin@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[glebbesputin@fedora lab07]$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
```

Figure 7: Вывод программы

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.

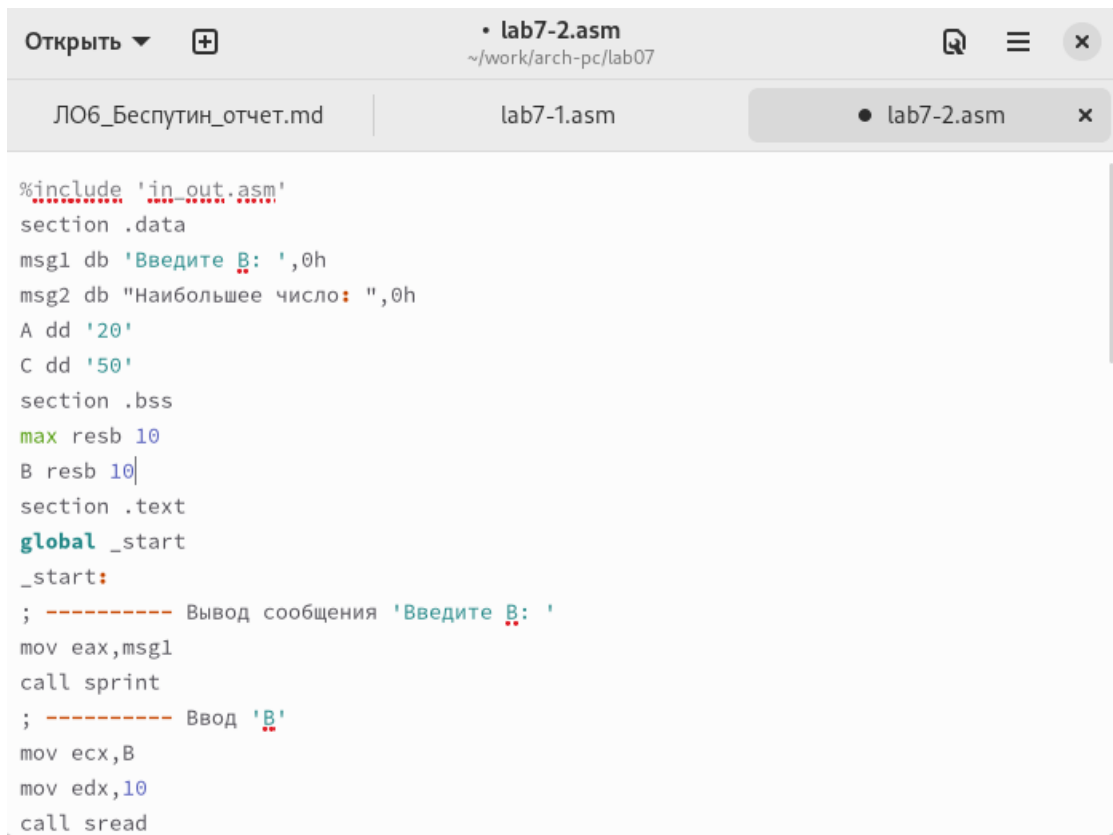
Создаю файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. (рис. 8).



```
[glebbesputin@fedora lab07]$ touch lab7-2.asm
[glebbesputin@fedora lab07]$
```

Figure 8: Создание файла

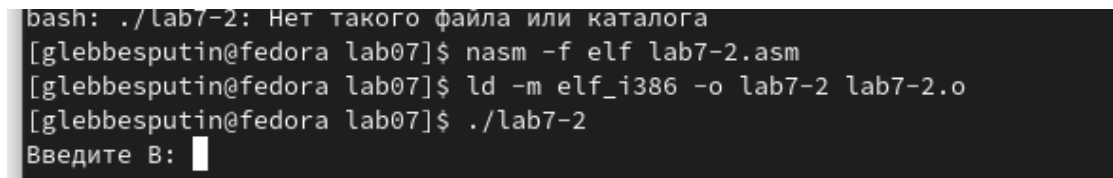
Текст программы из листинга 7.3 ввожу в lab7-2.asm. (рис. 9).



```
%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
```

Figure 9: Ввод текста программы из листинга 7.3

Создаю исполняемый файл и проверьте его работу. (рис. 10).



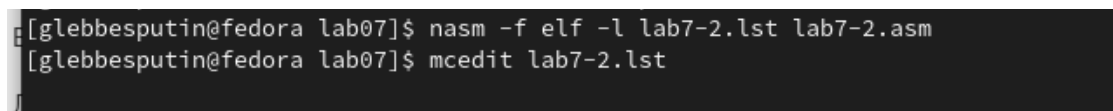
```
bash: ./lab7-2: Нет такого файла или каталога
[glebbesputin@fedora lab07]$ nasm -f elf lab7-2.asm
[glebbesputin@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[glebbesputin@fedora lab07]$ ./lab7-2
Введите B: █
```

Figure 10: Проверка работы файла

Файл работает корректно.

4.2 Изучение структуры файлы листинга

Создаю файл листинга для программы из файла lab7-2.asm. (рис. 11).



```
[glebbesputin@fedora lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm
[glebbesputin@fedora lab07]$ mcedit lab7-2.lst
```

Figure 11: Создание файла листинга

Открываю файл листинга lab7-2.lst с помощью текстового редактора и внимательно изучаю его формат и содержимое. (рис. 12).

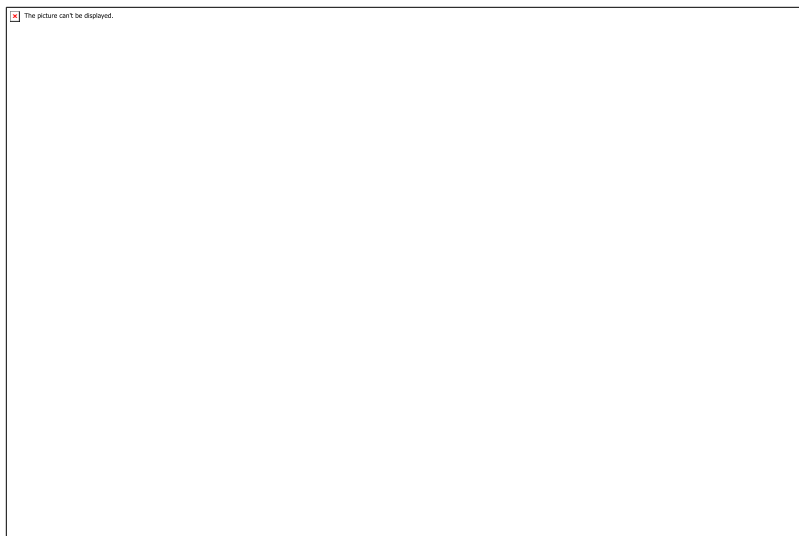


Figure 12: Изучение файла листинга

В представленных трех строчках содержатся следующие данные: (рис. 13).

```

2          <1> ; Функция вычисления длины сообщения
3          <1> slen:.....
4 00000000 53          <1>      push     ebx.....

```

Figure 13: Выбранные строки файла

“2” - номер строки кода, “; Функция вычисления длины сообщения” - комментарий к коду, не имеет адреса и машинного кода.

“3” - номер строки кода, “slen” - название функции, не имеет адреса и машинного кода.

“4” - номер строки кода, “00000000” - адрес строки, “53” - машинный код, “push ebx” - исходный текст программы, инструкция “push” помещает операнд “ebx” в стек.

Открываю файл с программой lab7-2.asm и в выбранной мной инструкции с двумя операндами удаляю выделенный операнд. (рис. 14).

```

; ----- Сравниваем 'А' и 'С' (как символы)
cmp ecx,[C] ; Сравниваем 'А' и 'С'
jg check_V ; если 'А>С', то переход на метку 'check_V',
1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск 8Удалить 9МенюМС10Выл

```

Figure 14: Удаление выделенного операнда из кода

Выполняю трансляцию с получением файла листинга. (рис. 15).

```
[glebbesputin@fedora lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:28: error: symbol `ecx2' not defined
[glebbesputin@fedora lab07]$
```

Figure 15: Получение файла листинга

На выходе я не получаю ни одного файла из-за ошибки: инструкция `mov` (единственная в коде содержит два операнда) не может работать, имея только один операнд, из-за чего нарушается работа кода.

4.3 Задания для самостоятельной работы

1. Пишу программу нахождения наименьшей из 3 целочисленных переменных `a`, `b` и `c`. Значения переменных выбираю из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 7. Мой вариант под номером 10, поэтому мои значения - 41, 62 и 35. (рис. 16).

```
%include 'in_out.asm'
section .data
msg db "Наименьшее число: ",0h
A dd '79'
B dd '83'
C dd '41'
section .bss
min resb 10
section .text
global _start
_start:
; ----- Записываем 'A' в переменную 'min'
mov ecx,[A] ; 'ecx = A'
mov [min],ecx ; 'min = 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 [min],ecx ; 'min = C'
```

Figure 16: Написание программы

Создаю исполняемый файл и проверяю его работу, подставляя необходимые значения. (рис. 17).

Figure 17: Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

```
%include 'in_out.asm'
```



```

section .data
msg db "Наименьшее число:",0h
A dd '79'
B dd '83'
C dd '41'

section .bss
min resb 10

section .text
global _start

_start:
; ----- Записываем 'A' в переменную 'min'
mov ecx,[A] ; 'ecx = A'
mov [min],ecx ; 'min = A'
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'A' и 'C'
jg check_B
mov ecx,[C] ; иначе 'ecx = C'
mov [min],ecx ; 'min = C'
; ----- Преобразование 'min(A,C)' из символа в число
check_B:
mov eax,min
call atoi ; Вызов подпрограммы перевода символа в число
mov [min],eax ; запись преобразованного числа в min
; ----- Сравниваем 'min(A,C)' и 'B' (как числа)
mov ecx,[min]
cmp ecx,[B] ; Сравниваем 'min(A,C)' и 'B'
jl fin ; если 'min(A,C)<B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'

```

```
mov [min],ecx
```

```
; ———- Вывод результата
```

```
fin:
```

```
mov eax, msg
```

```
call sprint ; Вывод сообщения 'Наименьшее число:'
```

```
mov eax,[min]
```

```
call iprintLF ; Вывод 'min(A,B,C)'
```

```
call quit ; Выход
```

2. Пишу программу, которая для введенных с клавиатуры значений x и a вычисляет значение и выводит результат вычислений заданной для моего варианта функции $f(x)$:

$x - 2$, если $x > 2$

$3 \cdot a$, если $x \leq 2$

(рис. 18).



Figure 18: Написание программы

Создаю исполняемый файл и проверяю его работу для значений x и a соответственно: (3;0), (1;2). (рис. 19).

```
[glebbesputin@fedora lab07]$ nasm -f elf task2.asm
[glebbesputin@fedora lab07]$ ld -m elf_i386 -o task2 task2.o
[glebbesputin@fedora lab07]$ ./task2
Введите x: 2
Введите a: 2
F(x)=4
[glebbesputin@fedora lab07]$ ./task2
Введите x: 2
Введите a: 1
F(x)=10
[glebbesputin@fedora lab07]$
```

Figure 19: Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

```
%include 'in_out.asm' section .data msg1 DB 'Введите x:',0h msg2 DB "Введите a:",0h otv:
DB 'F(x)=',0h section .bss x: RESB 80 a: RESB 80 res: RESB 80 section .text global _start
_start: mov eax,msg1 call sprint mov ecx,x mov edx,80 call sread mov eax,x call atoi mov
[x],eax mov eax,msg2 call sprint mov ecx,a mov edx,80 call sread mov eax,a call atoi mov
[a],eax mov eax,[x] cmp eax,[a] je x_is_3 mov eax,[x] imul eax,5 jmp calc_res x_is_3: mov
eax,[x] add eax,[a] calc_res: mov [res],eax fin: mov eax,otv call sprint mov eax,[res] call
iprintLF call quit
```

5 Выводы

По итогам данной лабораторной работы я изучил команды условного и безусловного переходов, приобрел навыки написания программ с использованием переходов и ознакомился с назначением и структурой файла листинга, что поможет мне при выполнении последующих лабораторных работ.

6 Список литературы