

Отчёт по лабораторной работе №7

Дисциплина: архитектура компьютеров и операционные системы

Хань Цзянтао

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Реализация переходов в NASM	8
4.2	Изучение структуры файлы листинга.....	13
4.3	Задания для самостоятельной работы.....	14
5	Выводы	20
6	Список литературы	21

Список иллюстраций

4.1	Создание файлов для лабораторной работы	8
4.2	Ввод текста программы из листинга 7.1	9
4.3	Запуск программного кода	9
4.4	Изменение текста программы.....	10
4.5	Создание исполняемого файла	10
4.6	Изменение текста программы.....	11
4.7	Вывод программы	11
4.8	Создание файла.....	12
4.9	Ввод текста программы из листинга 7.3	12
4.10	Проверка работы файла	12
4.11	Создание файла листинга	13
4.12	Изучение файла листинга.....	13
4.13	Выбранные строки файла	13
4.14	Удаление выделенного операнда из кода.....	14
4.15	Получение файла листинга.....	14
4.16	Написание программы	15
4.17	Запуск файла и проверка его работы	15
4.18	Написание программы	17
4.19	Запуск файла и проверка его работы	18

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.

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.

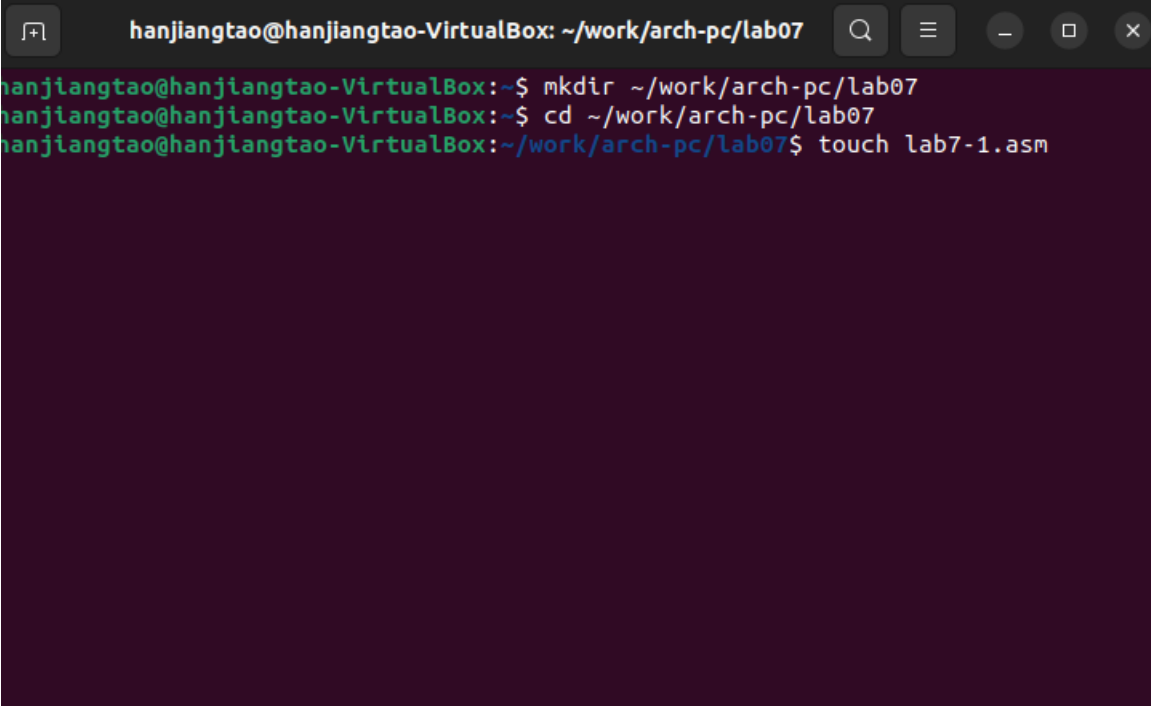
Безусловный переход выполняется инструкцией `jmp`. Инструкция `cmp` является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Инструкция `cmp` является командой сравнения двух операндов и имеет такой же формат, как и команда вычитания.

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

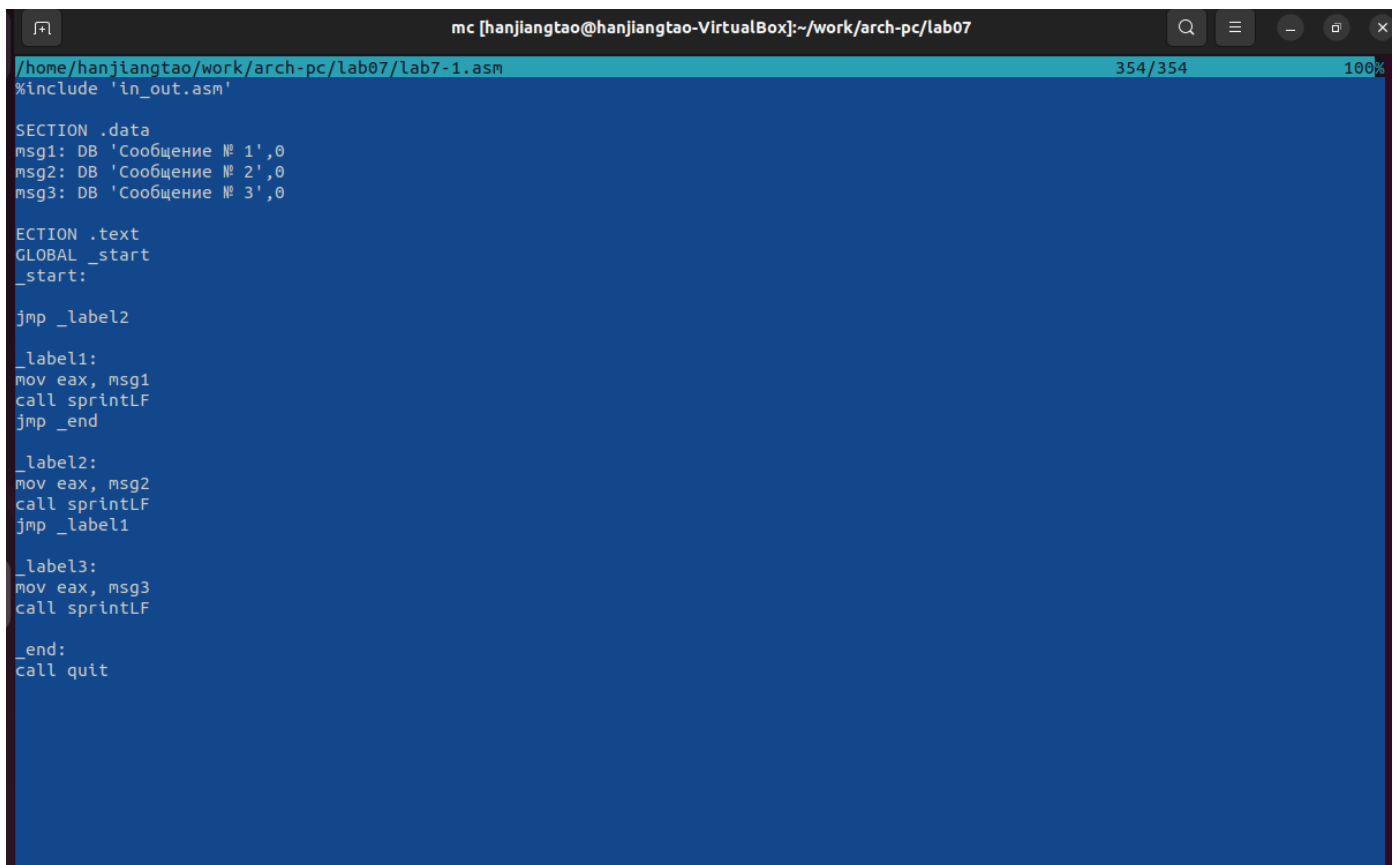
Создаю каталог для программ лабораторной работы № 7, перехожу в него и создаю файл lab7-1.asm. (рис. 4.19).



```
hanjiangtao@hanjiangtao-VirtualBox: ~/work/arch-pc/lab07
hanjiangtao@hanjiangtao-VirtualBox:~$ mkdir ~/work/arch-pc/lab07
hanjiangtao@hanjiangtao-VirtualBox:~$ cd ~/work/arch-pc/lab07
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ touch lab7-1.asm
```

Рис. 4.1: Создание файлов для лабораторной работы

Ввожу в файл lab7-1.asm текст программы из листинга 7.1. (рис. 4.19).



```
mc [hanjiangtao@hanjiangtao-VirtualBox]:~/work/arch-pc/lab07

/home/hanjiangtao/work/arch-pc/lab07/lab7-1.asm 354/354 100%
#include 'in_out.asm'

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
call sprintf
jmp _end

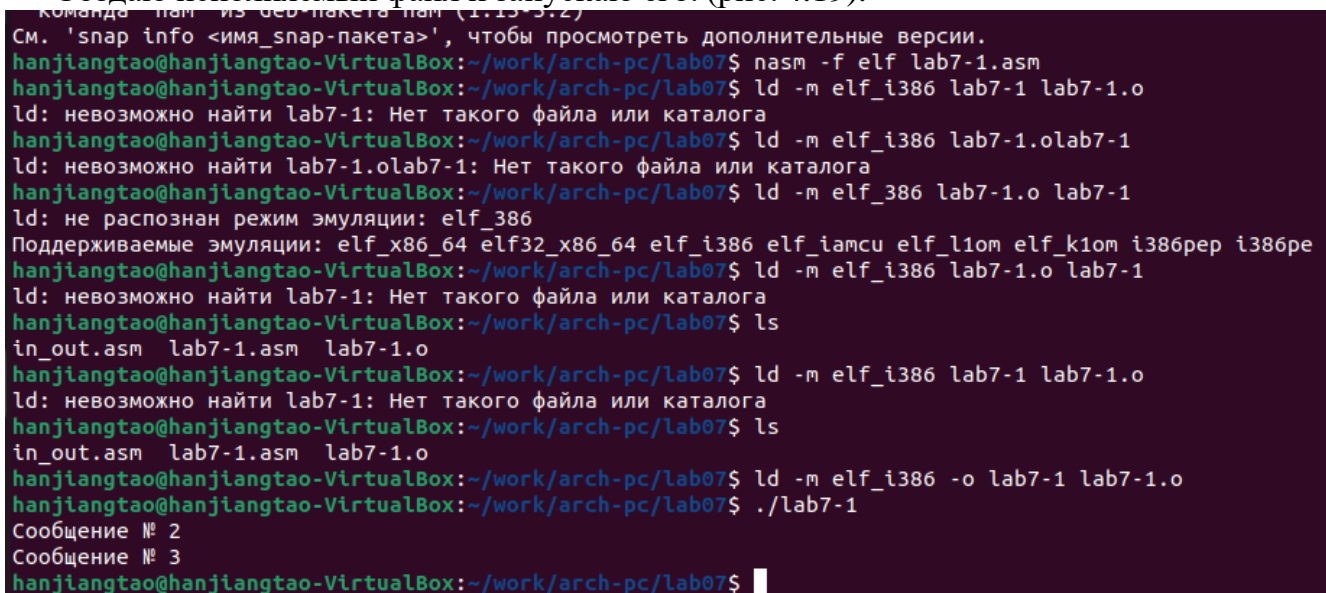
_label2:
mov eax, msg2
call sprintf
jmp _label1

_label3:
mov eax, msg3
call sprintf

_end:
call quit
```

Рис. 4.2: Ввод текста программы из листинга 7.1

Создаю исполняемый файл и запускаю его. (рис. 4.19).



```
См. 'snap info <имя_snap-пакета>', чтобы просмотреть дополнительные версии.
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-1 lab7-1.o
ld: невозможно найти lab7-1: Нет такого файла или каталога
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-1.o lab7-1
ld: невозможно найти lab7-1.o lab7-1: Нет такого файла или каталога
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_386 lab7-1.o lab7-1
ld: не распознан режим эмуляции: elf_386
Поддерживаемые эмуляции: elf_x86_64 elf32_x86_64 elf_i386 elf_iamcu elf_l1om elf_k1om i386pep i386pe
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-1.o lab7-1
ld: невозможно найти lab7-1: Нет такого файла или каталога
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ls
in_out.asm lab7-1.asm lab7-1.o
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-1 lab7-1.o
ld: невозможно найти lab7-1: Нет такого файла или каталога
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ls
in_out.asm lab7-1.asm lab7-1.o
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$
```

Рис. 4.3: 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. 4.19).

```
mc [hanjiangtao@hanjiangtao-VirtualBox]:~/work/arch-pc/lab07
/home/hanjiangtao/work/arch-pc/lab07/lab7-1.asm 354/354 100%
%include 'in_out.asm'

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
call sprintf
jmp _end

_label2:
mov eax, msg2
call sprintf
jmp _label1

_label3:
mov eax, msg3
call sprintf

_end:
call quit
```

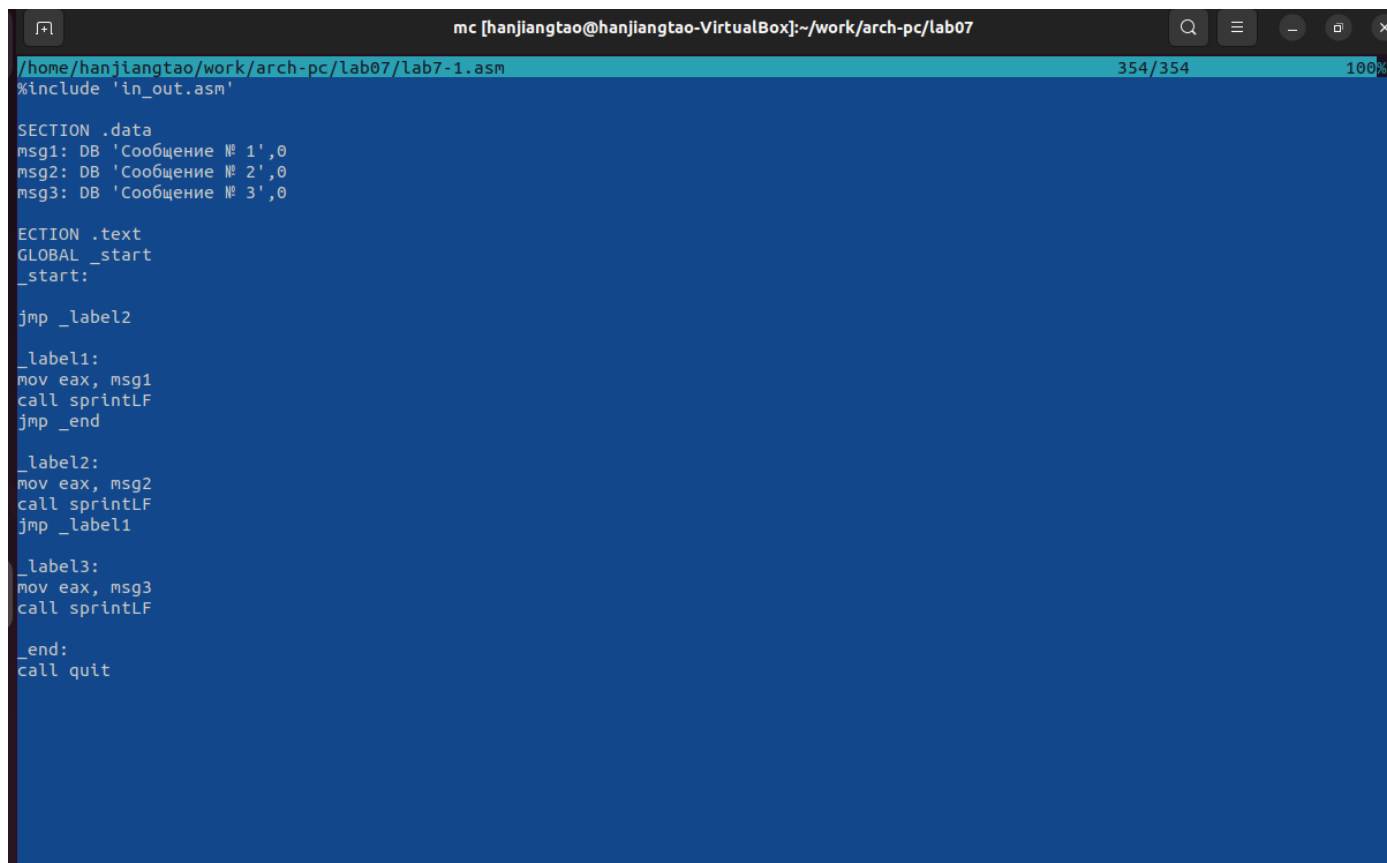
Рис. 4.4: Изменение текста программы

Создаю исполняемый файл и проверяю его работу. (рис. 4.19).

```
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-1 lab7-1.o
ld: невозможно найти lab7-1: Нет такого файла или каталога
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-1.o lab7-1
ld: невозможно найти lab7-1.o: Нет такого файла или каталога
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_386 lab7-1.o lab7-1
ld: не распознан режим эмуляции: elf_386
Поддерживаемые эмуляции: elf_x86_64 elf32_x86_64 elf_i386 elf_iamcu elf_l1om elf_k1om i386pep i386pe
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-1.o lab7-1
ld: невозможно найти lab7-1: Нет такого файла или каталога
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ls
in_out.asm lab7-1.asm lab7-1.o
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-1 lab7-1.o
ld: невозможно найти lab7-1: Нет такого файла или каталога
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ls
in_out.asm lab7-1.asm lab7-1.o
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$
```

Рис. 4.5: 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. 4.19).



```
mc [hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07]
/home/hanjiangtao/work/arch-pc/lab07/lab7-1.asm 354/354 100%
#include 'in_out.asm'

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
call sprintLF
jmp _end

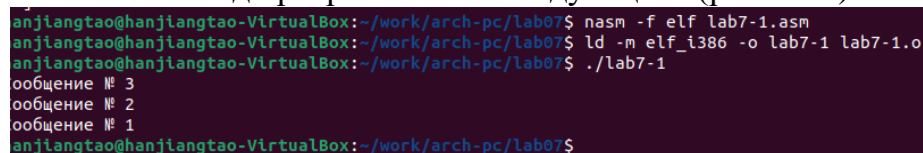
_label2:
mov eax, msg2
call sprintLF
jmp _label1

_label3:
mov eax, msg3
call sprintLF

_end:
call quit
```

Рис. 4.6: Изменение текста программы

чтобы вывод программы был следующим: (рис. 4.19).



```
anjlangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
anjlangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
anjlangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ ./lab7-1
сообщение № 3
сообщение № 2
сообщение № 1
anjlangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$
```

Рис. 4.7: Вывод программы

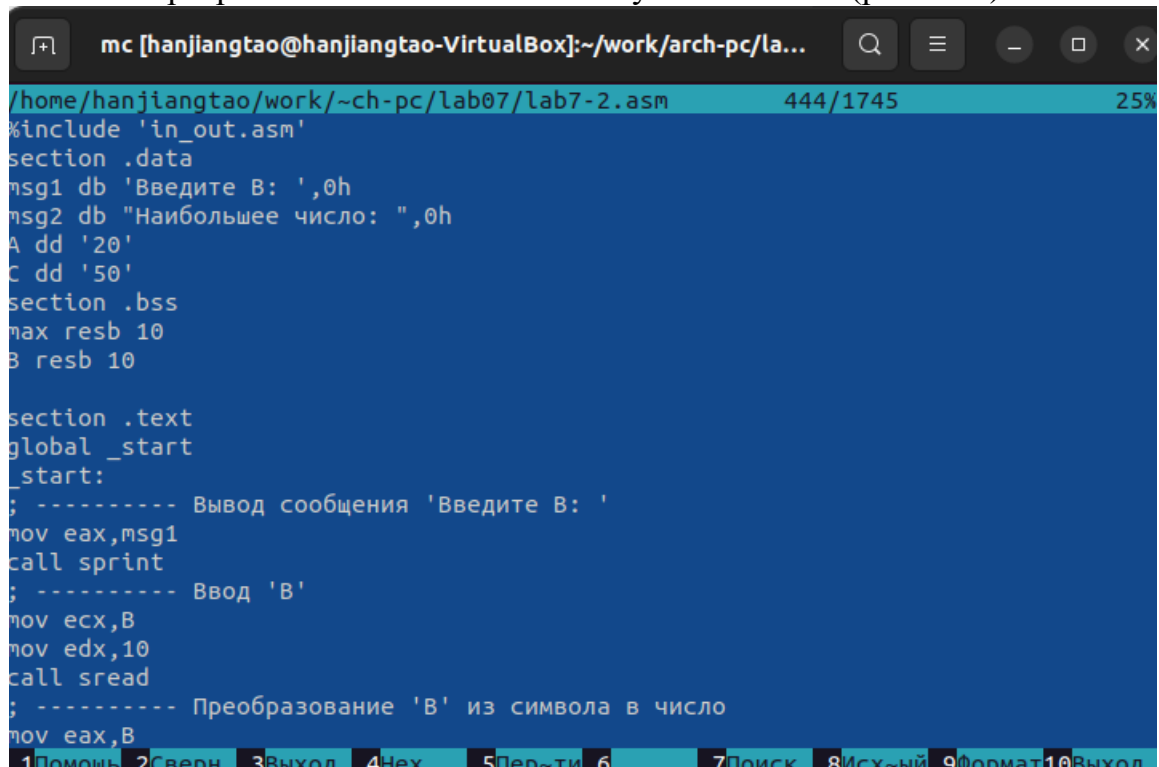
Рассмотрим программу, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: А,В и С. Значения для А и С задаются в программе, значение В вводится с клавиатуры.

Создаю файл lab7-2.asm в каталоге ~/work/arch-pc/lab07. (рис. 4.19).

```
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ touch lab7-2.asm
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$
```

Рис. 4.8: Создание файла

Текст программы из листинга 7.3 ввожу в lab7-2.asm. (рис. 4.19).



```
mc [hanjiangtao@hanjiangtao-VirtualBox]:~/work/arch-pc/la... 444/1745 25%
/home/hanjiangtao/work/~ch-pc/lab07/lab7-2.asm
%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 sprint
; ----- Ввод 'В'
mov ecx,B
mov edx,10
call sread
; ----- Преобразование 'В' из символа в число
mov eax,B
```

Рис. 4.9: Ввод текста программы из листинга 7.3

Создаю исполняемый файл и проверьте его работу. (рис. 4.19).

```
[eapostnova@fedora lab07]$ nasm -f elf lab7-2.asm
[eapostnova@fedora lab07]$ ld -m elf_i386 lab7-2.o -o lab7-2
[eapostnova@fedora lab07]$ ./lab7-2
Введите В: 70
Наибольшее число: 70
```

Рис. 4.10: Проверка работы файла

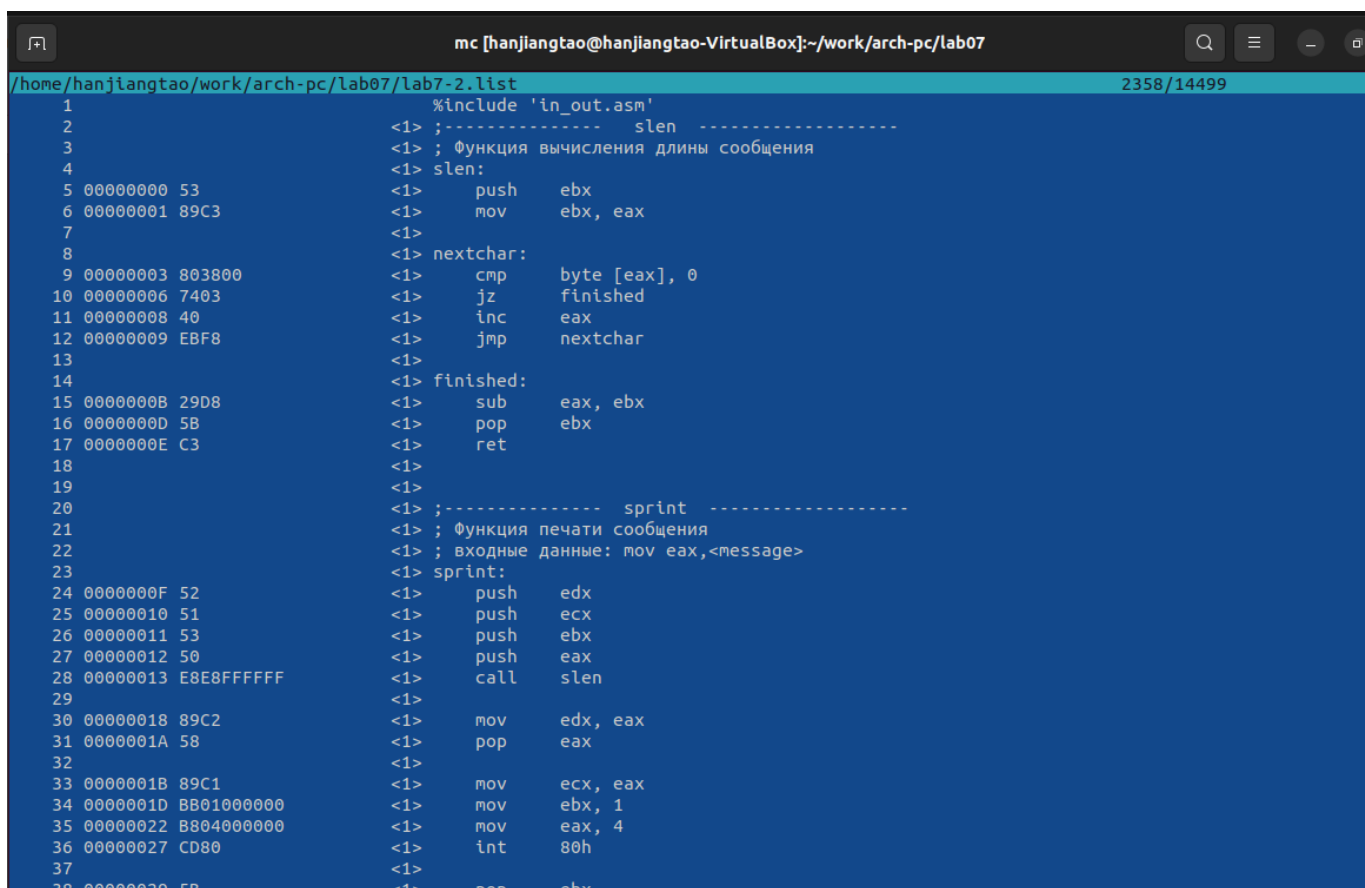
Файл работает корректно.

4.2 Изучение структуры файлы листинга

Создаю файл листинга для программы из файла lab7-2.asm. (рис. 4.19).

```
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
hanjiangtao@hanjiangtao-VirtualBox:~/work/arch-pc/lab07$
```

Рис. 4.11: Создание файла листинга



```
mc [hanjiangtao@hanjiangtao-VirtualBox]:~/work/arch-pc/lab07
/home/hanjiangtao/work/arch-pc/lab07/lab7-2.lst 2358/14499
1          %include 'in_out.asm'
2          <1> ;----- slen -----
3          <1> ; Функция вычисления длины сообщения
4          <1> slen:
5 00000000 53          <1> push    ebx
6 00000001 89C3        <1> mov     ebx, eax
7
8          <1> nextchar:
9 00000003 803800      <1> cmp     byte [eax], 0
10 00000006 7403        <1> jz      finished
11 00000008 40          <1> inc     eax
12 00000009 EBF8        <1> jmp     nextchar
13
14          <1> finished:
15 0000000B 29D8        <1> sub     eax, ebx
16 0000000D 5B          <1> pop     ebx
17 0000000E C3          <1> ret
18
19          <1>
20          <1> ;----- sprint -----
21          <1> ; Функция печати сообщения
22          <1> ; входные данные: mov eax,<message>
23          <1> sprint:
24 0000000F 52          <1> push    edx
25 00000010 51          <1> push    ecx
26 00000011 53          <1> push    ebx
27 00000012 50          <1> push    eax
28 00000013 E8E8FFFFFF <1> call    slen
29
30 00000018 89C2        <1> mov     edx, eax
31 0000001A 58          <1> pop     eax
32
33 0000001B 89C1        <1> mov     ecx, eax
34 0000001D BB01000000    <1> mov     ebx, 1
35 00000022 B804000000    <1> mov     eax, 4
36 00000027 CD80      <1> int     80h
37
38 00000029 5B          <1> pop     ebx
```

Открываю файл листинга lab7-2.lst с помощью текстового редактора и внимательно изучаю его формат и содержимое. (рис. 4.19).

Рис. 4.12: Изучение файла листинга

В представленных трех строчках содержатся следующие данные: (рис. 4.19).

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

Рис. 4.13: 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. 4.19).

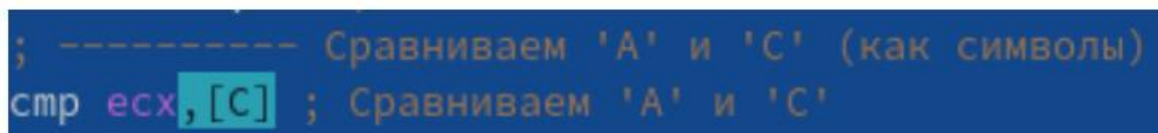


Рис. 4.14: Удаление выделенного операнда из кода

Выполняю трансляцию с получением файла листинга. (рис. 4.19).

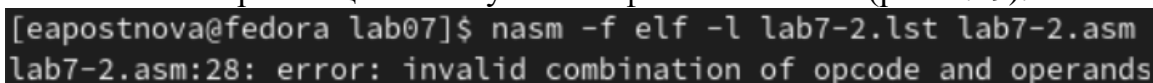


Рис. 4.15: Получение файла листинга

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. 4.19).

```

task1.asm      [-----] 15 L:[ 1+11 12/ 38] *(190 /1301b) 1
%include 'in_out.asm'
section .data
msg db "Наименьшее число: ",0h
A dd '41'
B dd '62'
C dd '35'
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'
; ----- Преобразование 'min(A,C)' из символа в число
check_B:
mov eax,min

```

Рис. 4.16: Написание программы

```

[eapostnova@fedora lab07]$ touch task1.asm
[eapostnova@fedora lab07]$ nasm -f elf task1.asm
[eapostnova@fedora lab07]$ ld -m elf_i386 task1.o -o task1
[eapostnova@fedora lab07]$ ./task1
Наименьшее число: 35

```

Создаю исполняемый файл и проверяю его работу, подставляя необходимые значения. (рис. 4.19).

Рис. 4.17: Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

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

```
section .data
```

```
msg db "Наименьшее число:",0h
```

```
A dd '41'
```

```

B dd '62'
C dd '35'
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

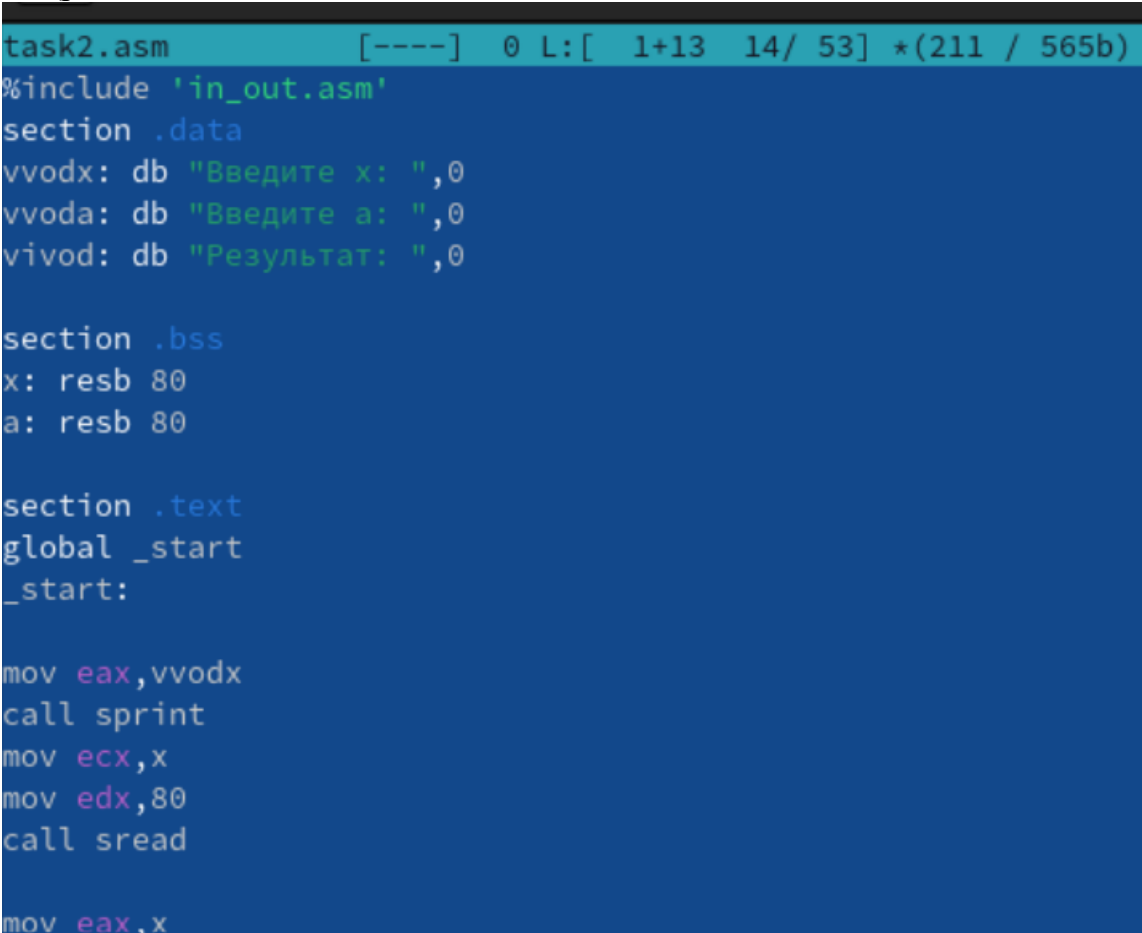
```

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 $x > 2$

$3*a$, если $x \leq 2$

(рис. 4.19).



```

task2.asm      [----]  0 L:[ 1+13 14/ 53] *(211 / 565b)
%include 'in_out.asm'
section .data
vvodb: db "Введите x: ",0
vveda: db "Введите a: ",0
vivod: db "Результат: ",0

section .bss
x: resb 80
a: resb 80

section .text
global _start
_start:

mov  eax,vvodb
call sprint
mov  ecx,x
mov  edx,80
call sread

mov  eax,x

```

Рис. 4.18: Написание программы

Создаю исполняемый файл и проверяю его работу для значений x и a соответ-

ственно: $(3;0)$, $(1;2)$. (рис. 4.19).

```

[eapostnova@fedora lab07]$ touch task2.asm
[eapostnova@fedora lab07]$ nasm -f elf task2.asm
[eapostnova@fedora lab07]$ ld -m elf_i386 -o task2 task2.o
[eapostnova@fedora lab07]$ ./task2
Введите x: 3
Результат: 1
[eapostnova@fedora lab07]$ ./task2
Введите x: 1
Введите a: 2
Результат: 6

```

Рис. 4.19: Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

```

#include 'in_out.asm'

section .data
vvodb: db "Введите x:",0
vvoda: db "Введите a:",0
vivodb: db "Результат:",0

section .bss
x: resb 80
a: resb 80

section .text
global _start
_start:
mov eax,vvodb
call sprint
mov ecx,x
mov edx,80
call sread
mov eax,x
call atoi
cmp eax,2

```

```
jg _functionx
mov eax,vvoda
call sprint
mov ecx,a
mov edx,80
call sread
mov eax,a
call atoi
jmp _functiona
_functiona:
mov edx,3
mul edx
jmp _end
_functionx:
add eax,-2
jmp _end
_end:
mov ecx,eax
mov eax,vivod
call sprint
mov eax,ecx
call iprintLF
call quit
```


5 Выводы

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

6 Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM.— 2021.— URL: <https://www.opennet.ru/docs/RUS/nasm/>.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix.— 2-е изд.— М.: МАКС Пресс, 2011.— URL: http://www.stolyarov.info/books/asm_unix.

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).