

Отчёт по лабораторной работе №7

Дисциплина: архитектура компьютеров

Даровских Александра Сергеевна

Содержание

1	Цель работы	3
2	Задание	4
3	Теоретическое введение.....	5
4	Выполнение лабораторной работы.....	6
4.1	Реализация переходов в NASM	6
4.2	Изучение структуры файлы листинга.....	10
4.3	Задания для самостоятельной работы	11
5	Выводы	16
6	Список литературы	17

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

С помощью команды `mkdir` создаем директорию, в которой будем создавать файлы с программами для лабораторной работы №7 (рис. 1). Переходим в созданный каталог с помощью команды `cd`

```
[asdarovskikh@fedora ~]$ mkdir ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab07  
[asdarovskikh@fedora ~]$ cd work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab07
```

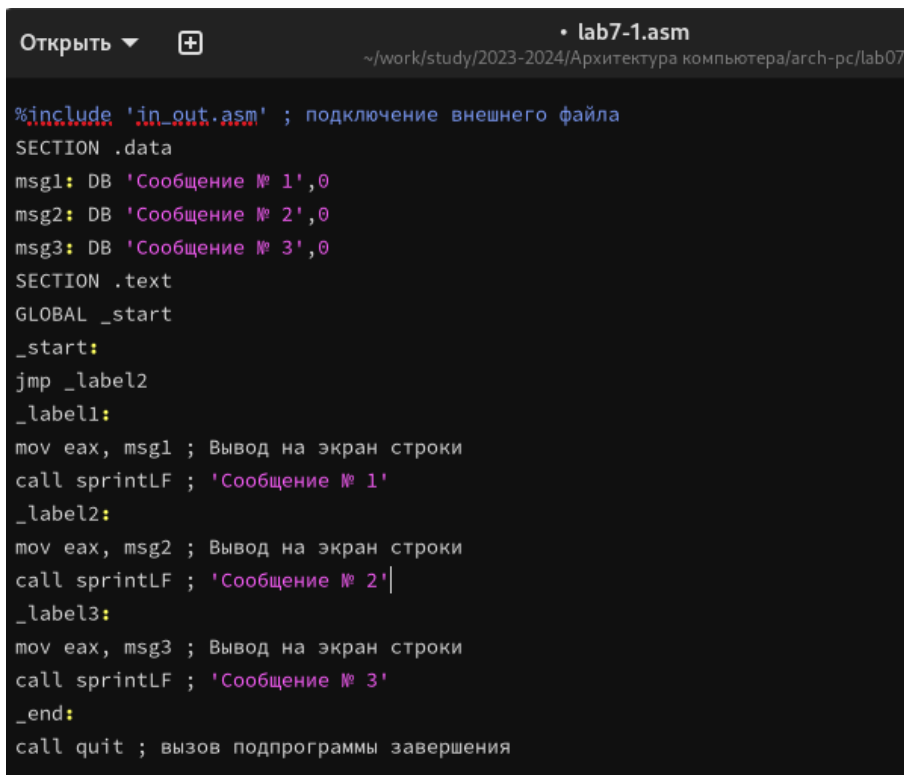
Рис. 1: Создание директории

С помощью команды `touch` создаем файл `lab7-1.asm` (рис. 2).

```
[asdarovskikh@fedora lab07]$ touch lab7-1.asm  
[asdarovskikh@fedora lab07]$ ls  
lab7-1.asm
```

Рис. 2: Создание файла

Открываем созданный файл `lab7-1.asm`, вставляем программу из листинга 7.1 (рис.3).



```
Открыть ▾ + lab7-1.asm  
~/work/study/2023-2024/Архитектура компьютера/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'  
_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
```

Рис. 3: Ввод текста программы из листинга 7.1

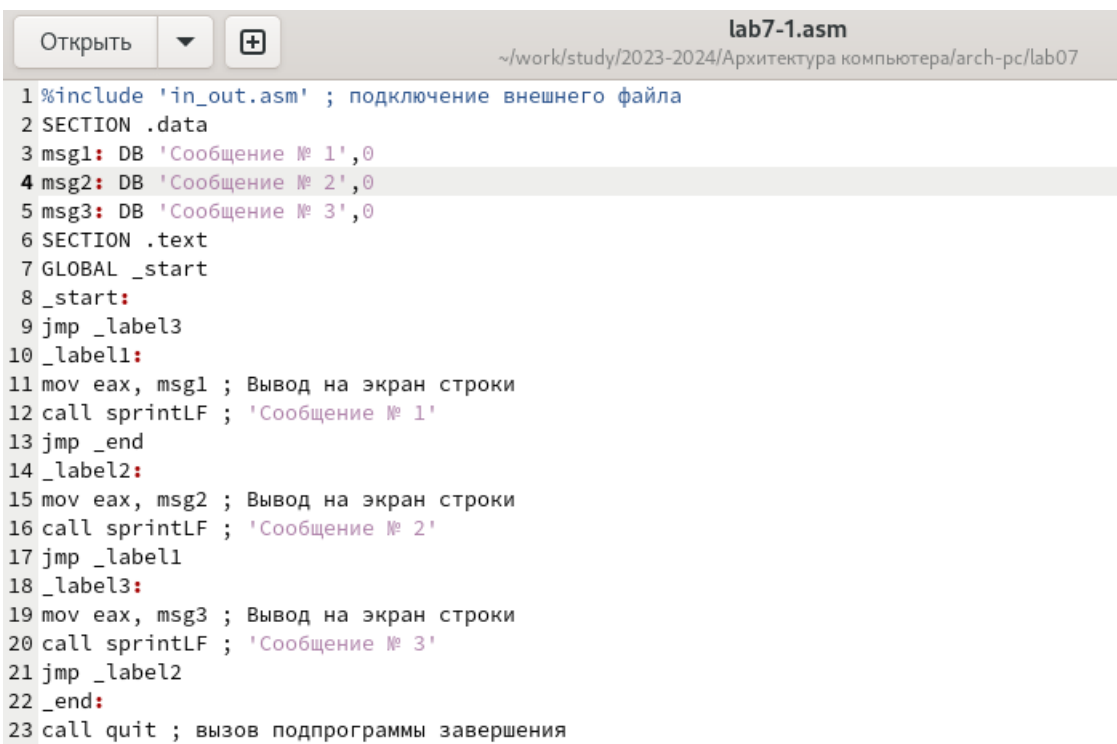
Создаем исполняемый файл и запускаем его. (рис.4).

```
[asdarovskikh@fedora lab07]$ nasm -f elf lab7-1.asm
[asdarovskikh@fedora lab07]$ ld -m elf_i386 lab7-1.o -o lab7-1
[asdarovskikh@fedora lab07]$ ./lab7-1
Сообщение № 2
Сообщение № 3
```

Рис. 4: Запуск программного кода

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

Изменяем программу, чтобы она выводила сначала 'Сообщение № 2', потом 'Сообщение № 1' и завершала работу в соответствии с листингом 7.2. (рис.5).



```
lab7-1.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07

1 %include 'in_out.asm' ; подключение внешнего файла
2 SECTION .data
3 msg1: DB 'Сообщение № 1',0
4 msg2: DB 'Сообщение № 2',0
5 msg3: DB 'Сообщение № 3',0
6 SECTION .text
7 GLOBAL _start
8 _start:
9 jmp _label3
10 _label1:
11 mov eax, msg1 ; Вывод на экран строки
12 call sprintf ; 'Сообщение № 1'
13 jmp _end
14 _label2:
15 mov eax, msg2 ; Вывод на экран строки
16 call sprintf ; 'Сообщение № 2'
17 jmp _label1
18 _label3:
19 mov eax, msg3 ; Вывод на экран строки
20 call sprintf ; 'Сообщение № 3'
21 jmp _label2
22 _end:
23 call quit ; вызов подпрограммы завершения
```

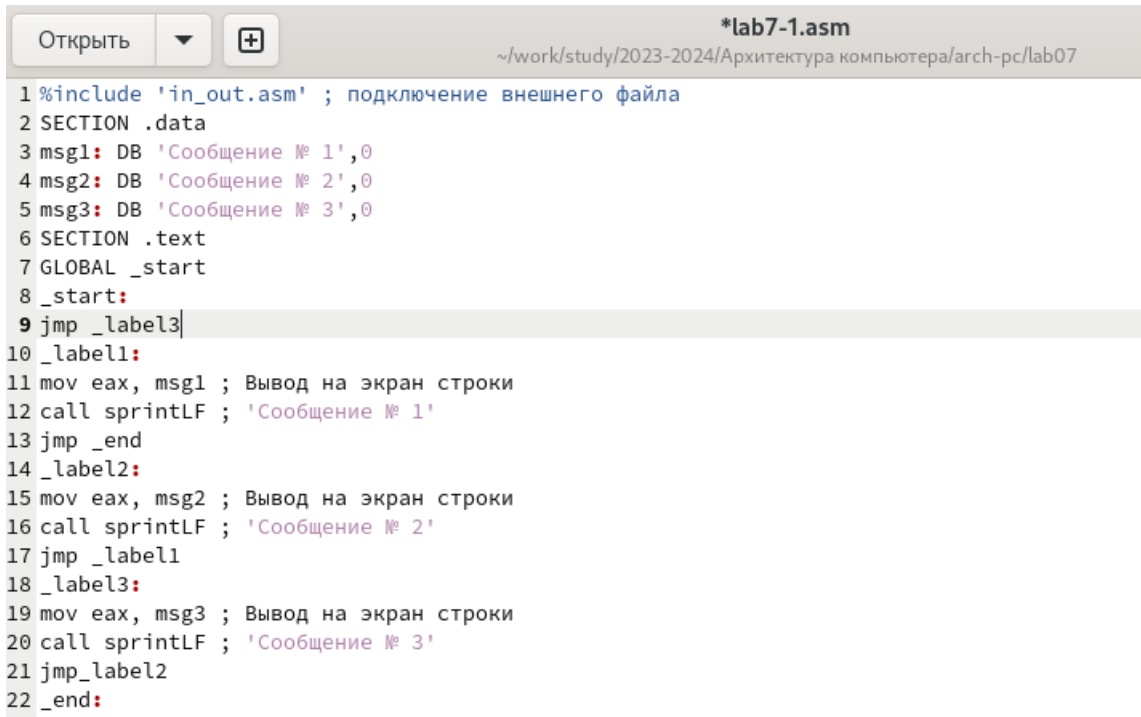
Рис. 5: Изменение текста программы

Создаем новый исполняемый файл программы и запускаем его. (рис. 6).

```
[asdarovskikh@fedora lab07]$ nasm -f elf lab7-1.asm
[asdarovskikh@fedora lab07]$ ld -m elf_i386 lab7-1.o -o lab7-1
[asdarovskikh@fedora lab07]$ ./lab7-1
Сообщение № 2
Сообщение № 1
```

Рис. 6: Создание исполняемого файла

Затем изменяем текст программы, добавив в начале программы `jmp _label3`, `jmp _label2` в конце метки `jmp _label3`, `jmp _label1` добавляем в конце метки `jmp _label2`, и добавляем `jmp _end` в конце метки `jmp _label1`, (рис. 7).

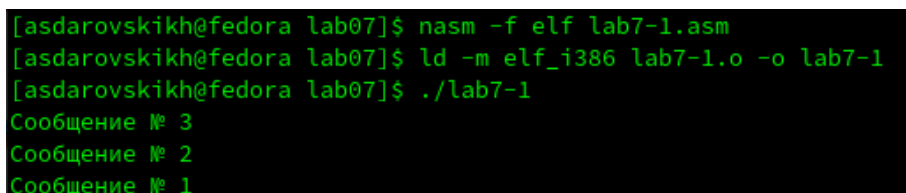


```
*lab7-1.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07

1 %include 'in_out.asm' ; подключение внешнего файла
2 SECTION .data
3 msg1: DB 'Сообщение № 1',0
4 msg2: DB 'Сообщение № 2',0
5 msg3: DB 'Сообщение № 3',0
6 SECTION .text
7 GLOBAL _start
8 _start:
9 jmp _label3
10 _label1:
11 mov eax, msg1 ; Вывод на экран строки
12 call sprintf ; 'Сообщение № 1'
13 jmp _end
14 _label2:
15 mov eax, msg2 ; Вывод на экран строки
16 call sprintf ; 'Сообщение № 2'
17 jmp _label1
18 _label3:
19 mov eax, msg3 ; Вывод на экран строки
20 call sprintf ; 'Сообщение № 3'
21 jmp _label2
22 _end:
```

Рис. 7: Изменение текста программы

Создаем новый исполняемый файл программы и запускаем его: (рис. 8).

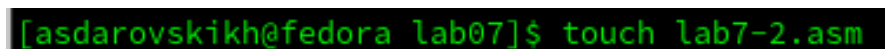


```
[asdarovskikh@fedora lab07]$ nasm -f elf lab7-1.asm
[asdarovskikh@fedora lab07]$ ld -m elf_i386 lab7-1.o -o lab7-1
[asdarovskikh@fedora lab07]$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
```

Рис. 8: 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.

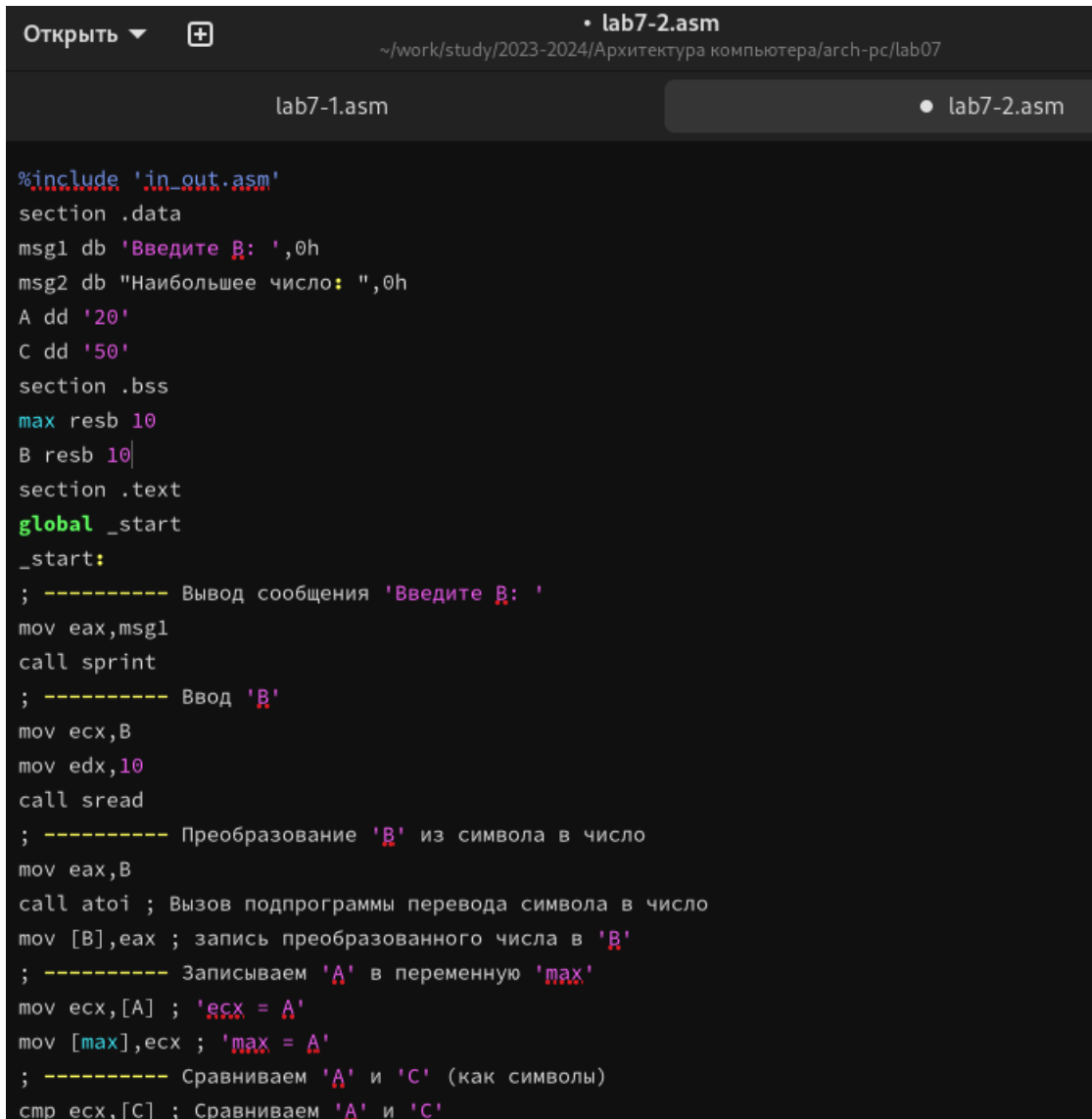
Создаю файл lab7-2.asm в каталоге `~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab07`. (рис. 9).



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

Рис. 9: Создание файла

Текст программы из листинга 7.3 вводим в lab7-2.asm. (рис. 10).



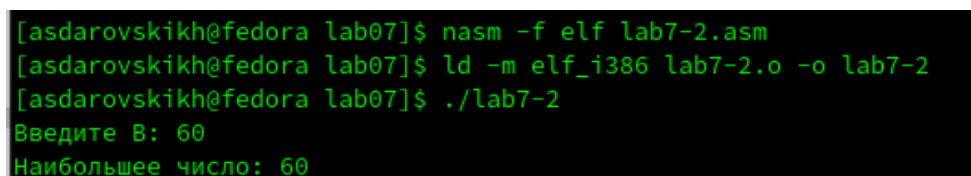
```
Открыть ▾ + • lab7-2.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07

lab7-1.asm 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'
```

Рис. 10: Ввод текста программы из листинга 7.3

Создаем новый исполняемый файл программы и запускаем его. (рис. 11).



```
[asdarovskikh@fedora lab07]$ nasm -f elf lab7-2.asm
[asdarovskikh@fedora lab07]$ ld -m elf_i386 lab7-2.o -o lab7-2
[asdarovskikh@fedora lab07]$ ./lab7-2
Введите B: 60
Наибольшее число: 60
```

Рис. 11: Проверка работы файла

Файл работает корректно.

4.2 Изучение структуры файлы листинга

Создаем файл листинга для программы из файла lab7-2.asm. (рис. 12).

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

Рис. 12: Создание файла листинга

Открываем файл листинга lab7-2.lst с помощью текстового редактора и изучаем содержимое. (рис. 13).

lab7-2.lst		
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07		
1	1	%include 'in_out.asm'
2	1	<1> ;----- slen -----
3	2	<1> ; Функция вычисления длины сообщения
4	3	<1> slen:
5	4 00000000 53	<1> push ebx
6	5 00000001 89C3	<1> mov ebx, eax
7	6	<1>
8	7	<1> nextchar:
9	8 00000003 803800	<1> cmp byte [eax], 0
10	9 00000006 7403	<1> jz finished
11	10 00000008 40	<1> inc eax
12	11 00000009 EBF8	<1> jmp nextchar
13	12	<1>
14	13	<1> finished:
15	14 0000000B 29D8	<1> sub eax, ebx
16	15 0000000D 5B	<1> pop ebx
17	16 0000000E C3	<1> ret
18	17	<1>
19	18	<1>
20	19	<1> ;----- sprint -----
21	20	<1> ; Функция печати сообщения
22	21	<1> ; входные данные: mov eax,<message>
23	22	<1> sprint:
24	23 0000000F 52	<1> push edx
25	24 00000010 51	<1> push ecx
26	25 00000011 53	<1> push ebx

Рис. 13: Изучение файла листинга

В представленных трех строчках содержатся следующие данные: (рис. 14).

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

Рис. 14: Выбранные строки файла

“2” - номер строки кода, “; Функция вычисления длинны 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.

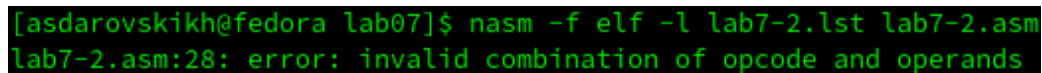
Открываем файл с программой lab7-2.asm и удаляем выделенный операнд. (рис. 15).



```
; ----- Сравниваем 'A' и 'C' (как символы)
cmp ecx, [C] ; Сравниваем 'A' и 'C'
```

Рис. 15: Удаление выделенного операнда из кода

Выполняем трансляцию с получением файла листинга. (рис. 16).



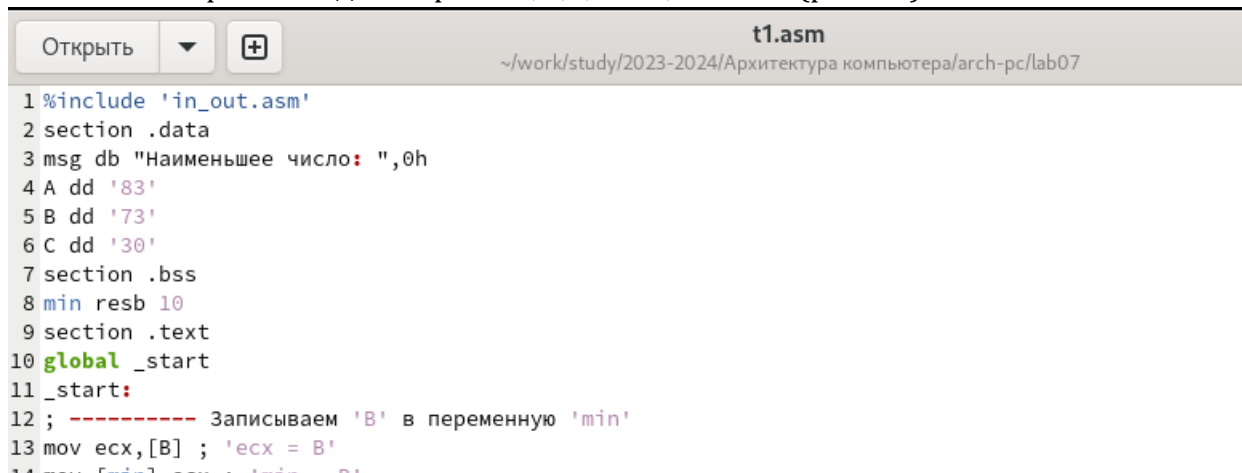
```
[asdarovskikh@fedora lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:28: error: invalid combination of opcode and operands
```

Рис. 16: Получение файла листинга

Инструкция mov (единственная в коде, содержащая два операнда) не может работать только с одним операндом, что нарушает код.

4.3 Задания для самостоятельной работы

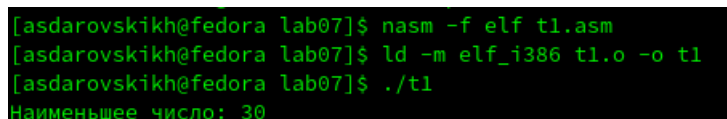
1. Мой вариант под номером 18, a,b,c - 83, 73 и 30. (рис. 17).



```
1 %include 'in_out.asm'
2 section .data
3 msg db "Наименьшее число: ",0h
4 A dd '83'
5 B dd '73'
6 C dd '30'
7 section .bss
8 min resb 10
9 section .text
10 global _start
11 _start:
12 ; ----- Записываем 'B' в переменную 'min'
13 mov ecx, [B] ; 'ecx = B'
```

Рис. 17: Написание программы

Создаю исполняемый файл и проверяю его работу, подставляя необходимые значение. (рис. 18).



```
[asdarovskikh@fedora lab07]$ nasm -f elf t1.asm
[asdarovskikh@fedora lab07]$ ld -m elf_i386 t1.o -o t1
[asdarovskikh@fedora lab07]$ ./t1
Наименьшее число: 30
```

Рис. 18: Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

```
include 'in_out.asm'

section .data
msg db "Наименьшее число: ",0h
A dd '83'
B dd '73'
C dd '30'

section .bss
min resb 10

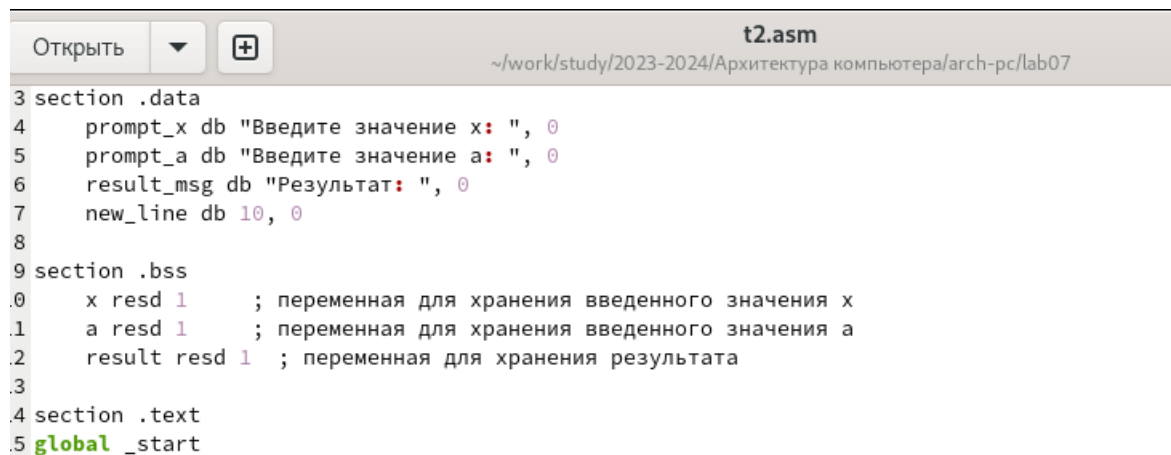
section .text
global _start
_start:
; ----- Записываем 'B' в переменную 'min'
mov ecx,[B] ; 'ecx = B'
mov [min],ecx ; 'min = B'
; ----- Сравниваем 'B' и 'C' (как символы)
cmp ecx,[C] ; Сравниваем 'B' и 'C'
jg check_B ; если 'B<C', то переход на метку 'check_A',
mov ecx,[C] ; иначе 'ecx = C'
mov [min],ecx ; 'min = C'
; ----- Преобразование 'min(B,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(B,C)' и 'A' (как числа)
mov ecx,[min]
cmp ecx,[A] ; Сравниваем 'min(B,C)' и 'A'
jl fin ; если 'min(B,C)<A', то переход на 'fin',
```

```

mov ecx,[A] ; иначе 'ecx = A'
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
Код программы

```

2. a^2 , $a=1$; $10+x$, $a=1$ (рис. 19).



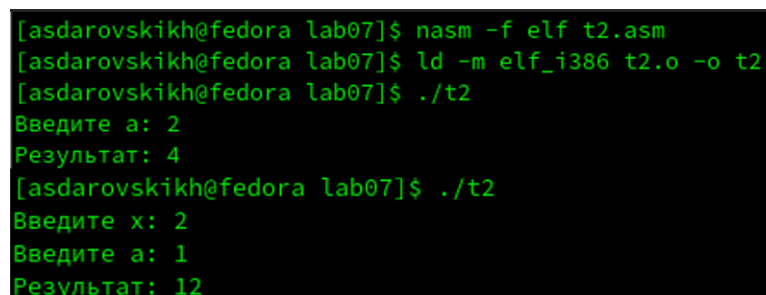
```

t2.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/lab07
3 section .data
4     prompt_x db "Введите значение x: ", 0
5     prompt_a db "Введите значение a: ", 0
6     result_msg db "Результат: ", 0
7     new_line db 10, 0
8
9 section .bss
10    x resd 1 ; переменная для хранения введенного значения x
11    a resd 1 ; переменная для хранения введенного значения a
12    result resd 1 ; переменная для хранения результата
13
14 section .text
15 global _start

```

Рис. 19: Написание программы

Создаю исполняемый файл и проверяю его работу для значений x и a соответственно: (1;2), (2;1). (рис. 20).



```

[asdarovskikh@fedora lab07]$ nasm -f elf t2.asm
[asdarovskikh@fedora lab07]$ ld -m elf_i386 t2.o -o t2
[asdarovskikh@fedora lab07]$ ./t2
Введите a: 2
Результат: 4
[asdarovskikh@fedora lab07]$ ./t2
Введите x: 2
Введите a: 1
Результат: 12

```

Рис. 20: Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

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

```
section .data
```

```
    prompt_x db "Введите значение x: ", 0
```

```
    prompt_a db "Введите значение a: ", 0
```

```
    result_msg db "Результат: ", 0
```

```
    new_line db 10, 0
```

```
section .bss
```

```
    x resd 1 ; переменная для хранения введенного значения x
```

```
    a resd 1 ; переменная для хранения введенного значения a
```

```
    result resd 1 ; переменная для хранения результата
```

```
section .text
```

```
global _start
```

```
_start:
```

```
    ; запрос значения x
```

```
    mov edx, prompt_x
```

```
    call sprint
```

```
    call sread
```

```
    mov [x], eax ; сохраняем x
```

```
    ; запрос значения a
```

```
    mov edx, prompt_a
```

```
    call sprint
```

```
    call sread
```

```
    mov [a], eax ; сохраняем a
```

```
    ; вычисление и вывод результата
```

```
    mov eax, [a]
```

```
    cmp eax, 1
```

```
    je a_is_1 ; если a=1, переходим к вычислению 10 + x
```

```
    imul eax, eax ; умножаем а на само себя
    jmp print_result
a_is_1:
    add eax, [x] ; вычисляем 10 + x
print_result:
    mov [result], eax ; сохраняем результат
    mov edx, result_msg
    call sprint
    mov eax, [result]
    call iprintLF
    ; ВЫХОД
    call quit
```

5 Выводы

При выполнении данной лабораторной работы я 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. Лабораторная работа №7. Команды безусловного и условного переходов в Nasm. Программирование ветвлений.