РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра фундаментальной информатики и информационных технологий

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №7

Тема: Команды безусловного и условного переходов в Nasm.

Программирование ветвлений.

Дисциплина: Архитектура компьютеров

Студент: Герчет Вячеслав

Группа: НКАбд-03-25

Студ. билет № 1132255650

Преподаватель: Штепа Кристина Александровна

МОСКВА

2025 г.

Содержание

Список иллюстраций

1. Цель работы

Цель данной работы - освоить принципы использования команд условного и безусловного переходов в NASM, а также изучить назначение и структуру файла листинга. В процессе выполнения работы требуется научиться изменять последовательность выполнения программы с помощью переходов и анализировать машинный код на основе полученного листинга.

2. Задание

Написать программы для решения системы выражений.

3. Теоретическое введение

Команды переходов в ассемблере NASM используются для изменения обычного линейного порядка выполнения программы. В отличие от языков высокого уровня, где ветвления оформляются через `if`, `else`, `while` и другие конструкции, в ассемблере управление осуществляется напрямую с помощью переходов - условных и безусловных.

Безусловный переход (`jmp`) просто изменяет адрес следующей выполняемой команды. Он используется, когда нужно перейти в указанный участок программы независимо от каких-либо условий: например, для организации цикла, выхода из него или пропуска части кода.

Условные переходы позволяют выполнять ветвление программы, основываясь на результате предыдущей операции. Такие команды анализируют флаги процессора - специальные биты, которые автоматически устанавливаются после выполнения арифметических и логических инструкций. Основные флаги, участвующие в проверках:

ZF (Zero Flag) - устанавливается, если результат равен нулю;

SF (Sign Flag) - отражает знак результата;

CF (Carry Flag) - показывает перенос бита при операциях;

OF (Overflow Flag) - говорит о переполнении знакового результата.

На основе этих флагов работают команды `je`, `jne`, `jg`, `jl`, `jge`, `jle` и другие. Каждая из них передаёт управление на указанную метку, если условие выполняется. Например, `je` делает переход, если предыдущая операция дала нулевой результат (ZF = 1), а `jg` срабатывает, если значение оказалось больше при учёте знака.

Также важную роль в лабораторной работе играет *файл листинга.* Это специальный текстовый файл, который создаётся во время компиляции. Он показывает, как каждая строка исходного кода превращается в машинные инструкции. Листинг позволяет увидеть адреса команд, их

машинные коды, а также помогает анализировать, как работают переходы и ветвления на уровне низкоуровневых инструкций.

Таким образом, команды переходов - это основа для построения логики программы на ассемблере. Они позволяют реализовывать разветвления, циклы, выбор минимального или максимального значения, а также различные алгоритмы, которые в языках высокого уровня выглядят как обычные условные конструкции.

4. Выполнение лабораторной работы

4.1 Реализация переходов в NASM

Создаем файл (рис. 3.1).

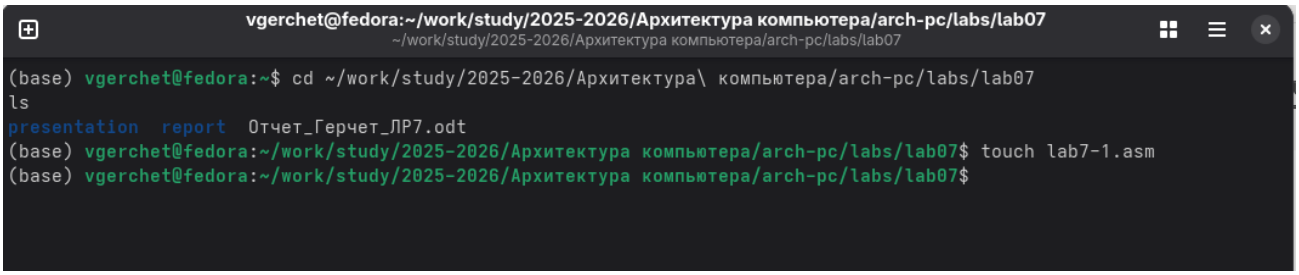

Рис. 3.1: Создаем файл с помощью команды touch

Открываем файл в Midnight Commander и заполняем его в соответствии с листингом 7.1 (рис. 3.2).

Рис. 3.2: Заполняем файл

Создаем исполняемый файл и запускаем его (рис. 3.3).

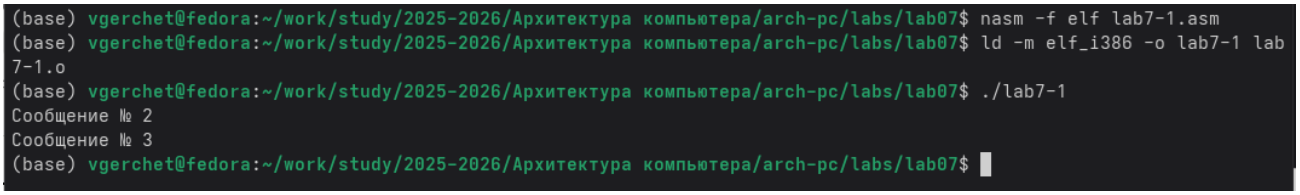

Рис. 3.3: Запускаем файл и смотрим на его работу

Снова открываем файл для редактирования и изменяем его в соответствии с листингом 7.2 (рис. 3.4).

Рис. 3.4: Изменяем файл

Создаем исполняемый файл и запускаем его (рис. 3.5).

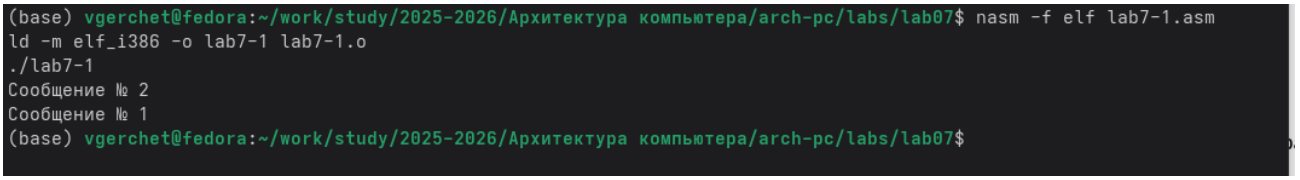

Рис. 3.5: Запускаем файл и смотрим на его работу

Снова открываем файл для редактирования и изменяем его, чтобы произошел данный вывод (рис. 3.6).

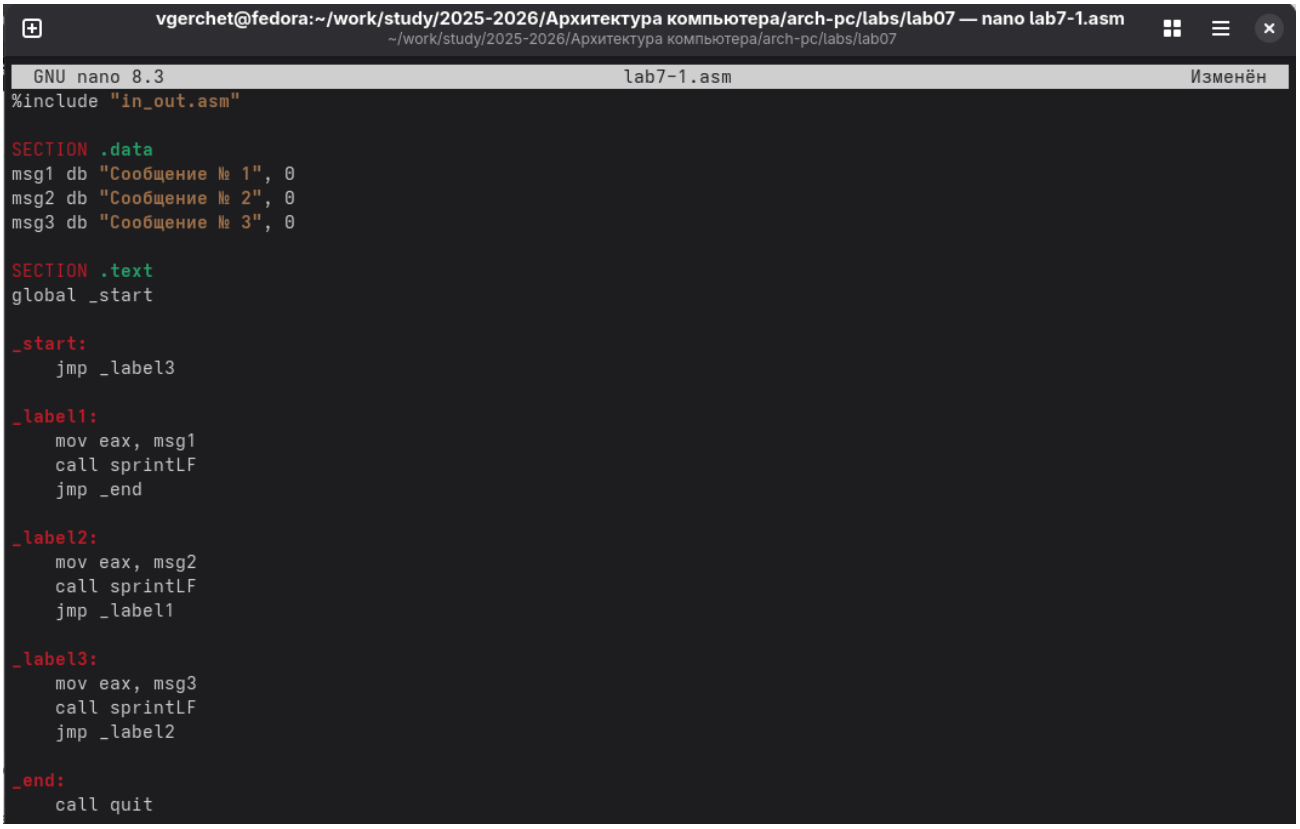

Рис. 3.6: Редактируем файл

Создаем исполняемый файл и запускаем его (рис. 3.7).

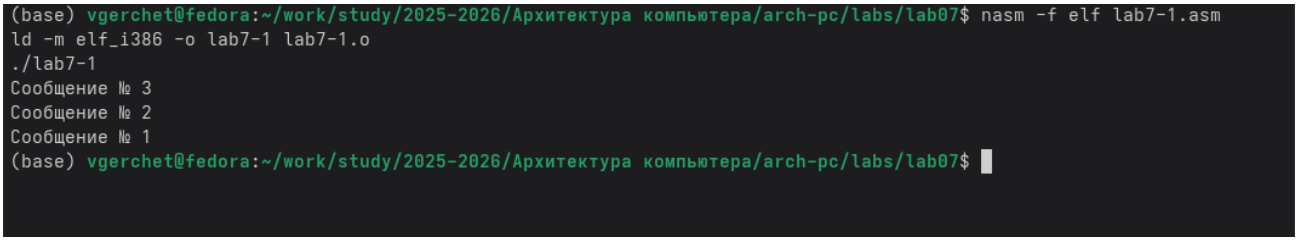

Рис. 3.7: Проверяем, сошелся ли наш вывод с данным в условии выводом

Создаем новый файл (рис. 3.8).

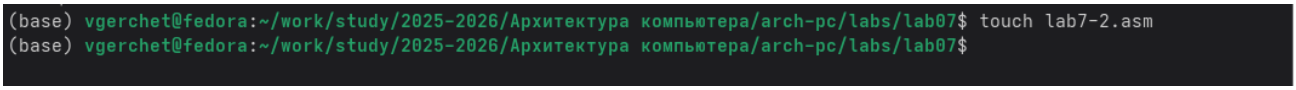

Рис. 3.8: Создаем файл командой touch

Открываем файл в Midnight Commander и заполняем его в соответствии с

листингом 7.3 (рис. 3.9)

vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab07 — nano lab7-2.asm
~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab07

GNU nano 8.3 lab7-2.asm Изменён

```
%include "in_out.asm"

section .data
    msg1 db "Введите B: ", 0
    msg2 db "Наибольшее число: ", 0

    A_str db "20", 0
    C_str db "50", 0

section .bss
    B_str resb 16
    max   resd 1

section .text
    global _start

_start:
    ; выводим приглашение
    mov eax, msg1
    call sprint

    ; читаем B как строку
    mov ecx, B_str
    mov edx, 16
    call sread

    ; переводим A в число и кладём в max
    mov eax, A_str
    call atoi
    mov [max], eax        ; max = A

    ; переводим B в число и сравниваем с max
    mov eax, B_str
    call atoi
    cmp eax, [max]
    jle check_C           ; если B <= max, ничего не меняем
    mov [max], eax        ; иначе max = B

check_C:
    ; переводим C в число и сравниваем с max
    mov eax, C_str
    call atoi
    cmp eax, [max]
    jle fin               ; если C <= max, ничего не меняем
    mov [max], eax        ; иначе max = C
```

Рис. 3.9: Заполняем файл

Создаем исполняемый файл и проверяем его работу, вводя разные значения B (рис. 3.10).

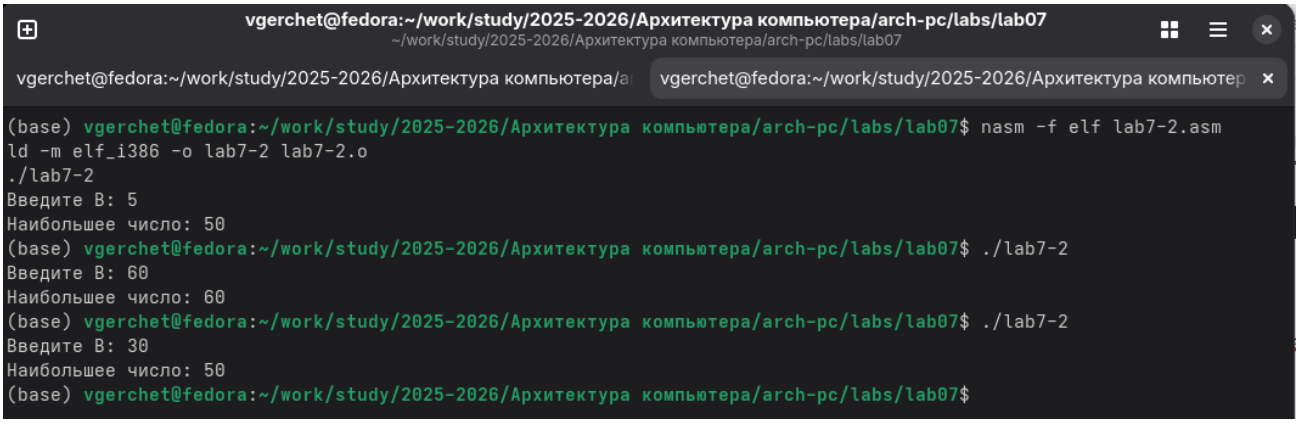

Рис. 3.10: Смотрим на работу программ

3.2 Изучение структуры файлы листинга

Создаем файл листинга для программы lab7-2.asm (рис. 3.11).

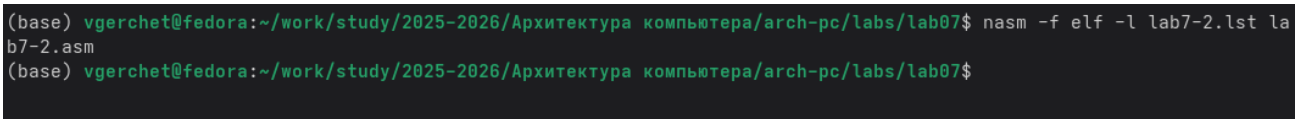

Рис. 3.11: Создаем файл листинга

Открываем файл листинга с помощью команды mcedit и изучаем его (рис. 3.12).

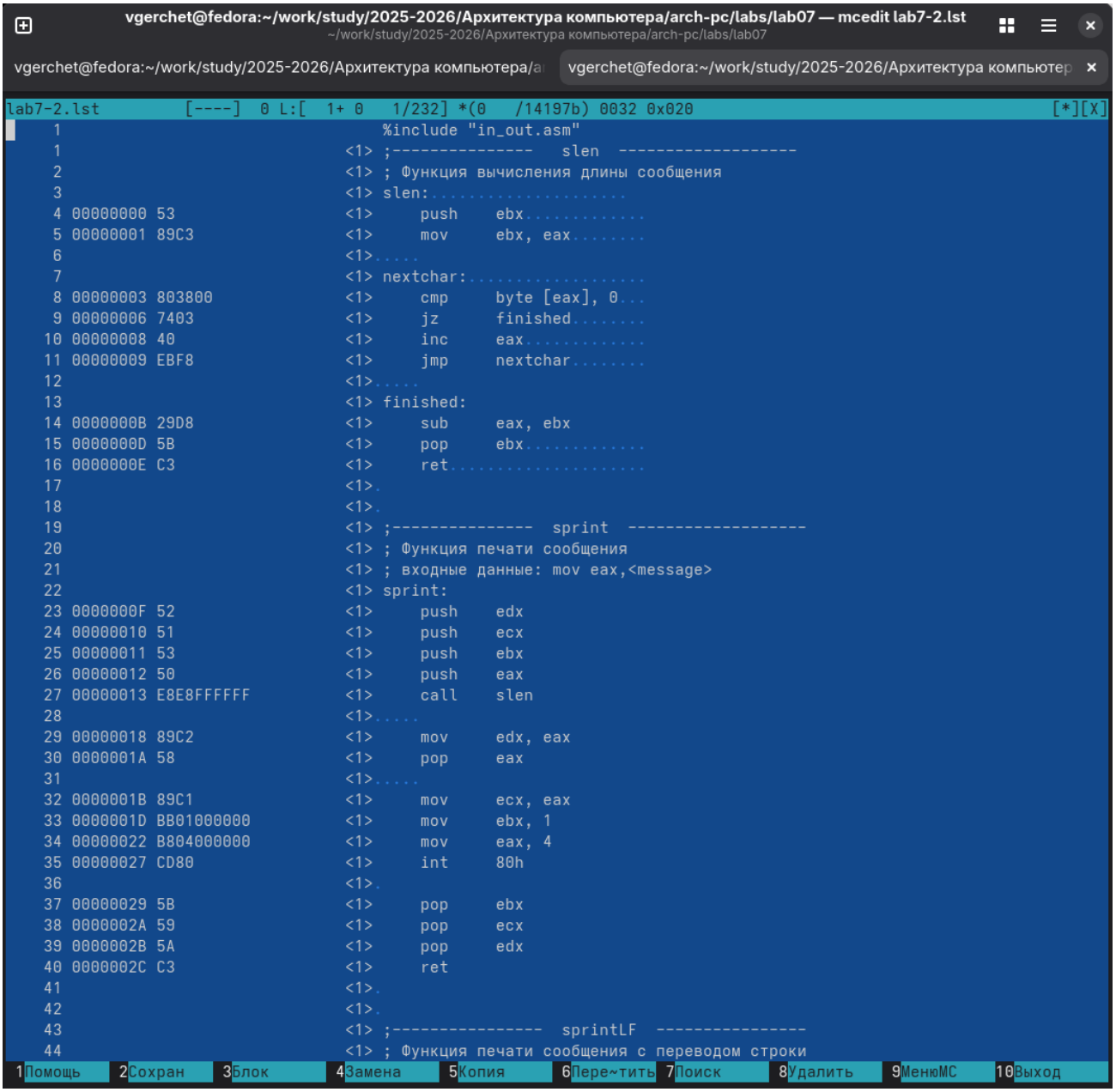

Рис. 3.12: Изучаем файл

Строка 33: 0000001D-адрес в сегменте кода, BB01000000-машинный код, mov ebx,1-присвоение переменной ecx значения 1.

Строка 34: 00000022-адрес в сегменте кода, B804000000-машинный код, mov eax,4-присвоение переменной eax значения 4.

11Строка 35 00000027-адрес в сегменте кода, CD80-машинный код, int 80h-вызов ядра.

Открываем файл и удаляем один операндум (рис. 3.13).

```
_start:
    ; выводим приглашение
    mov eax
    call sprint

    ; читаем B как строку
    mov ecx, B_str
    mov edx, 16
    call sread
```

Рис. 3.13: Удаляем операндум из файла

Транслируем с получением файла листинга (рис. 3.14).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf -l lab7-2.lst la
b7-2.asm
lab7-2.asm:19: error: invalid combination of opcode and operands
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab07$ ls
in_out.asm  lab7-1  lab7-1.asm  lab7-1.o  lab7-2  lab7-2.asm  lab7-2.lst  presentation  report  Отчет_Герчет_ЛР7.odt
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab07$
```

Рис. 3.14: Транслируем файл

При трансляции файла, выдается ошибка, но создаются исполнительный файл lab7-2 и lab7-2.lst

Снова открываем файл листинга и изучаем его (рис. 3.15).

```
   18                                   ; выводим приглашение
   19                                   mov eax
   19          ******************       error: invalid combination of opcode and operands
   20 000000E8 E822FFFFFF               call sprint
   21 ..................................
   22                                   ; читаем B как строку
   23 000000ED B9[00000000]             mov ecx, B_str
   24 000000F2 BA10000000               mov edx, 16
```

Рис. 3.15: Изучаем файл с ошибкой

5. Выполнение самостоятельной работы

ВАРИАНТ-11

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

Создаем новый файл (рис. 3.16).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab07$ touch lab7-3.asm
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab07$
```

Рис. 3.16: Создаем файл командой touch

Открываем его и пишем программу, которая выберет наименбшее число из трех(2 числа уже в программе, 3е вводится из консоли) (рис. 3.17). (a=21, b=28, c=34)

vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/a vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютер

GNU nano 8.3 lab7-3.asm Изменён

```
%include "in_out.asm"

section .data
    a_val   dd 21
    b_val   dd 28

    msg_c   db "Введите c: ", 0
    msg_min db "Минимальное число: ", 0

section .bss
    c_buf   resb 32
    c_val   resd 1
    min     resd 1

section .text
    global _start

_start:
    ; --- ввод c с клавиатуры ---
    mov eax, msg_c
    call sprint

    mov ecx, c_buf
    mov edx, 32
    call sread

    mov eax, c_buf
    call atoi          ; eax = c
    mov [c_val], eax   ; сохраняем c

    ; --- min = a ---
    mov eax, [a_val]
    mov [min], eax

    ; --- если b < min, то min = b ---
    mov eax, [b_val]
    cmp eax, [min]
    jge check_c
    mov [min], eax

check_c:
    ; --- если c < min, то min = c ---
    mov eax, [c_val]
```

Рис. 3.17: Пишем программу

Транслируем файл и смотрим на работу программы (рис. 3.18).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf lab7-3.asm
ld -m elf_i386 -o lab7-3 lab7-3.o
./lab7-3
Введите c: 34
Минимальное число: 21
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab07$
```

Рис. 3.18: Смотрим на рабботу программы(всё верно)

2. Напишите программу, которая для введенных с клавиатуры значений и вычисляет значение заданной функции ( ) и выводит результат вычислений. Вид функции ( ) 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 и из 7.6.

Создаем новый файл (рис. 3.19).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab07$ touch lab7-4.asm
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab07$
```

Рис. 3.19: Создаем файл командой touch

Открываем его и пишем программу, которая решит систему уравнений, при

даных, введенных в консоль (рис. 3.20).

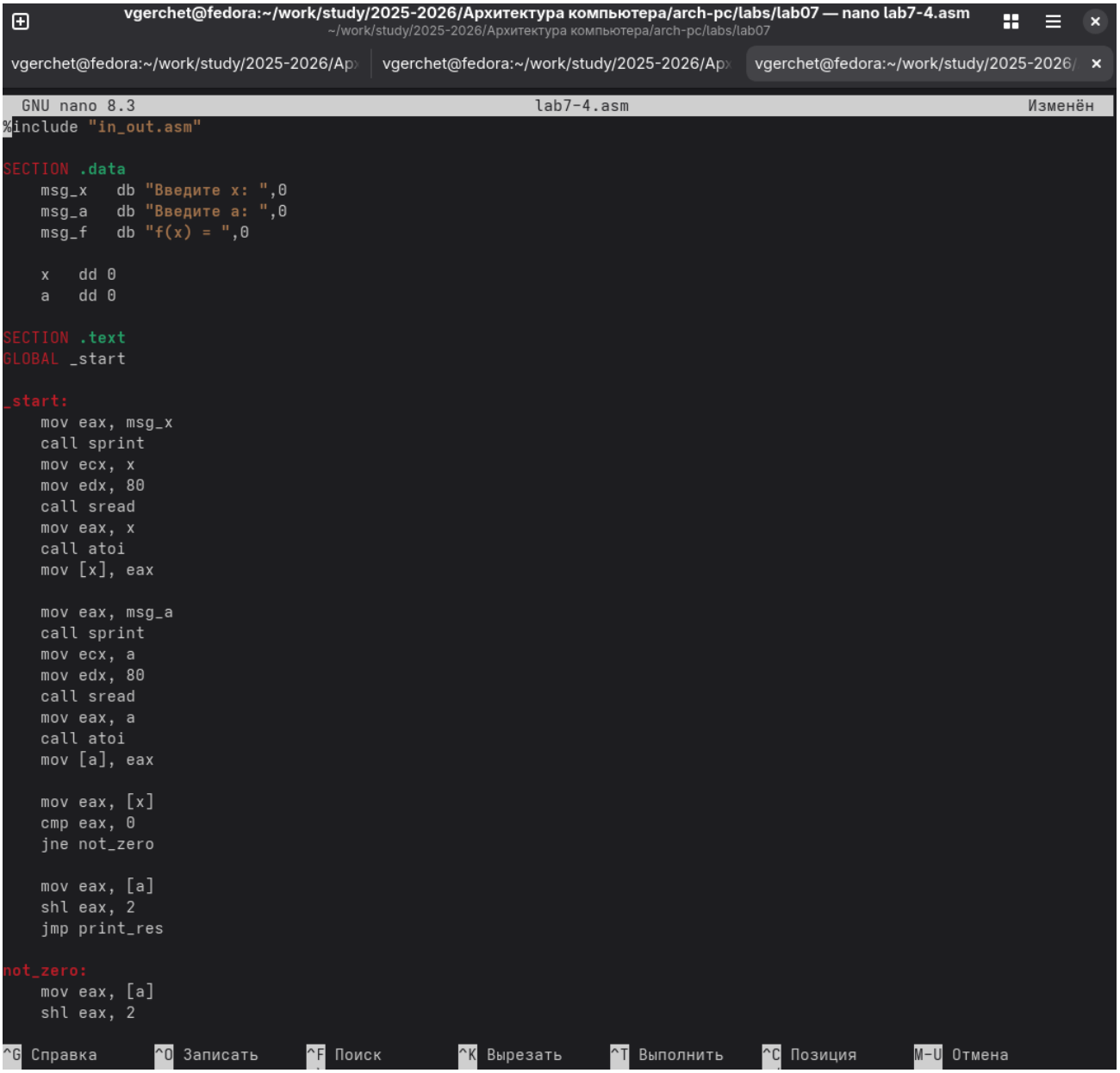

Рис. 3.20: Пишем программу

Транслируем файл и проверяем его работу при x=0 и a=3(рис. 3.21).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab07$ nasm -f elf lab7-4.asm
ld -m elf_i386 -o lab7-4 lab7-4.o
./lab7-4
Введите x: 0
Введите a: 3
f(x) = 12
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab07$
```

Рис. 3.21: Проверяем работу программы

Транслируем файл и проверяем его работу при x=1 и a=2(рис. 3.22).

```
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab07$ ./lab7-4
Введите x: 1
Введите a: 2
f(x) = 9
(base) vgerchet@fedora:~/work/study/2025-2026/Архитектура компьютера/arch-pc/labs/lab07$
```

Рис. 3.22: Проверяем работу программы

4. Выводы

В ходе выполнения лабораторной работы я на практике разобрался, как в NASM работают безусловные и условные переходы, и как с их помощью можно управлять логикой программы. Я увидел, как результат арифметических операций отражается во флагах процессора и как именно команды переходов используют эти флаги для принятия решений.

Также я научился составлять простые алгоритмы ветвления, использовать метки, организовывать проверку условий и выбирать нужные команды перехода. Отдельно познакомился с листингом - теперь понимаю, как исходный код отображается в виде машинных команд и как по ним отслеживать работу программы.

В итоге я лучше понял, как на низком уровне реализуются конструкции типа `if`, `else` и сравнения в высокоуровневых языках, и получил практический опыт построения программ с разветвлённой логикой на ассемблере.

5. Список литературы

Методические указания к лабораторной работе №7.

Виноградов В. А. Ассемблер для архитектуры x86.

Art Of Assembly Language, Randall Hyde – Bol https://www.bol.com/nl/nl/f/art-of-assembly-language-2nd-edition/9200000033684674/

Documentation https://www.nasm.us/docs.html