

Отчёт по лабораторной работе № 7

**дисциплина: Архитектура компьютера. Команды
безусловного и условного переходов в Nasm.
Программирование ветвлений**

Студент: Святашова Ксения Евгеньевна

Содержание

1	Цель работы	4
2	Теоритическое введение	5
3	Выполнение лабораторной работы	8
3.1	Реализация переходов в NASM	8
3.2	Изучение структуры файлы листинга	17
4	Задания для самостоятельной работы	21
5	Вывод	28

Список иллюстраций

3.1	Каталог lab07	8
3.2	Ввод текста из листинга 7.1	9
3.3	Запуск исполняемого файла	10
3.4	Ввод текста из листинга 7.2	11
3.5	Запуск исполняемого файла	12
3.6	Изменение текста программы	13
3.7	Запуск исполняемого файла	13
3.8	Создание файла lab7-2.asm	14
3.9	Ввод текста из листинга 7.3	14
3.10	Запуск исполняемого файла	17
3.11	Запуск исполняемого файла	17
3.12	Создание файла листинга lab7-2.asm	17
3.13	Файл листинга	18
3.14	Строка листинга	18
3.15	Строка листинга	19
3.16	Строка листинга	19
3.17	Попытка трансляции файла	19
3.18	Ошибка в файле листинга	20
4.1	Создание файла lab7-3.asm	21
4.2	Программа вычисления выражение из варианта 9	22
4.3	Запуск исполняемого файла	22
4.4	Создание файла lab7-4.asm	23
4.5	Программа вычисления выражение из варианта 9	24
4.6	Запуск исполняемого файла	25
4.7	Запуск исполняемого файла	25

1 Цель работы

Целью работы является изучение команд условного и безусловного переходов, приобретение навыков написания программ с использованием переходов и знакомство с назначением и структурой файла листинга.

2 Теоритическое 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` (от англ. `jump` – прыжок), которая включает в себя адрес перехода, куда следует передать управление. 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.

Для условного перехода необходима проверка какого-либо условия. В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра 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.

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.

Все ошибки и предупреждения, обнаруженные при ассемблировании, транслятор выводит на экран, и файл листинга не создаётся.

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).

3 Выполнение лабораторной работы

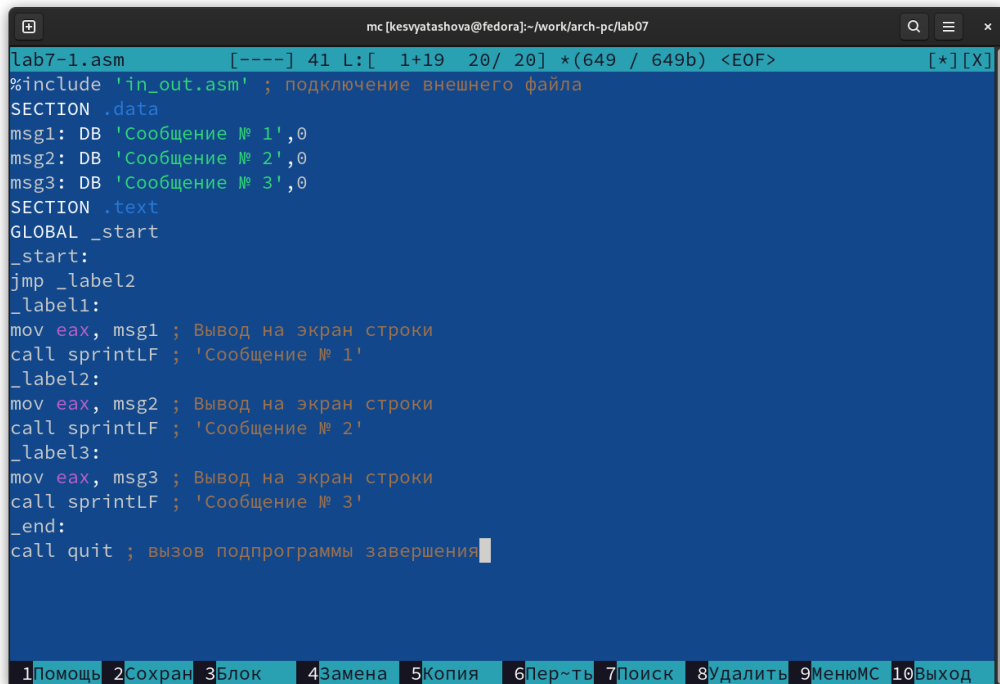
3.1 Реализация переходов в NASM

1. Создадим каталог для программ лабораторной работы № 7, перейдем в него и создадим файл lab7-1.asm(рис. 3.1):

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

Рис. 3.1: Каталог 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.(рис. 3.2):



```
lab7-1.asm [----] 41 L: [ 1+19 20/ 20] *(649 / 649b) <EOF> [*] [X]
#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. 3.2: Ввод текста из листинга 7.1

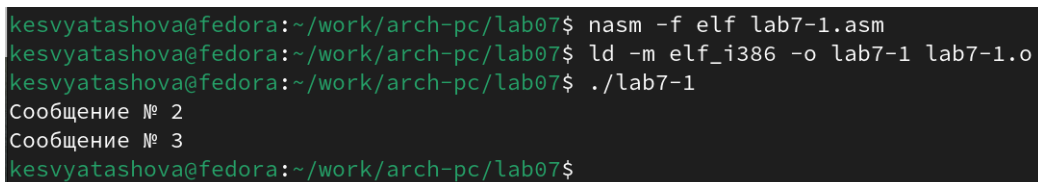
Листинг 7.1. Программа с использованием инструкции jmp

```
%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 sprintLF ; 'Сообщение № 1'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 2'
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintLF ; 'Сообщение № 3'
_end:
call quit ; вызов подпрограммы завершения
Создадим исполняемый файл и запустим его(рис. 3.3):

```



```

kesvyatashova@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
kesvyatashova@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
kesvyatashova@fedora:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
kesvyatashova@fedora:~/work/arch-pc/lab07$

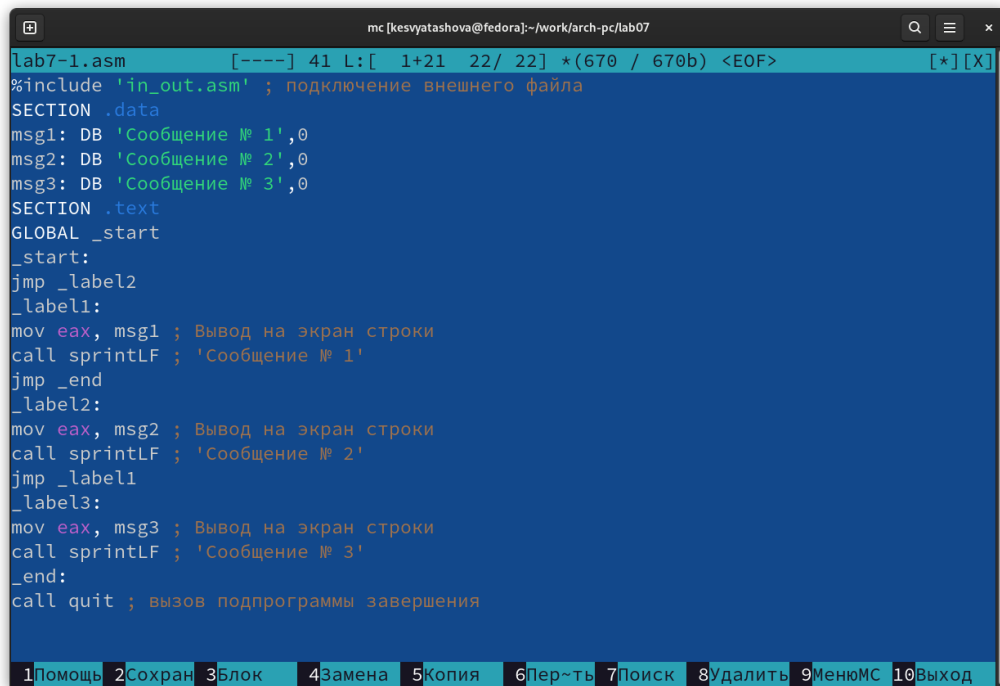
```

Рис. 3.3: Запуск исполняемого файла

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

Изменим программу так, 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.(рис. 3.4):



```
lab7-1.asm      [----] 41 L: [ 1+21  22/ 22] *(670 / 670b) <EOF>  [*] [X]
#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

1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Удалить 9МенюМС 10Выход
```

Рис. 3.4: Ввод текста из листинга 7.2

Листинг 7.2. Программа с использованием инструкции jmp

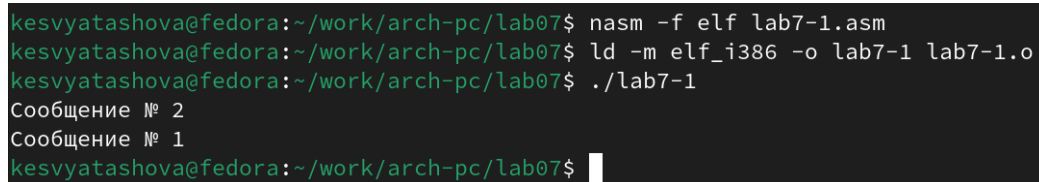
```
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение № 1',0
msg2: DB 'Сообщение № 2',0
msg3: DB 'Сообщение № 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label2
```

```

_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

```

Создадим исполняемый файл и запустим его(рис. 3.5):



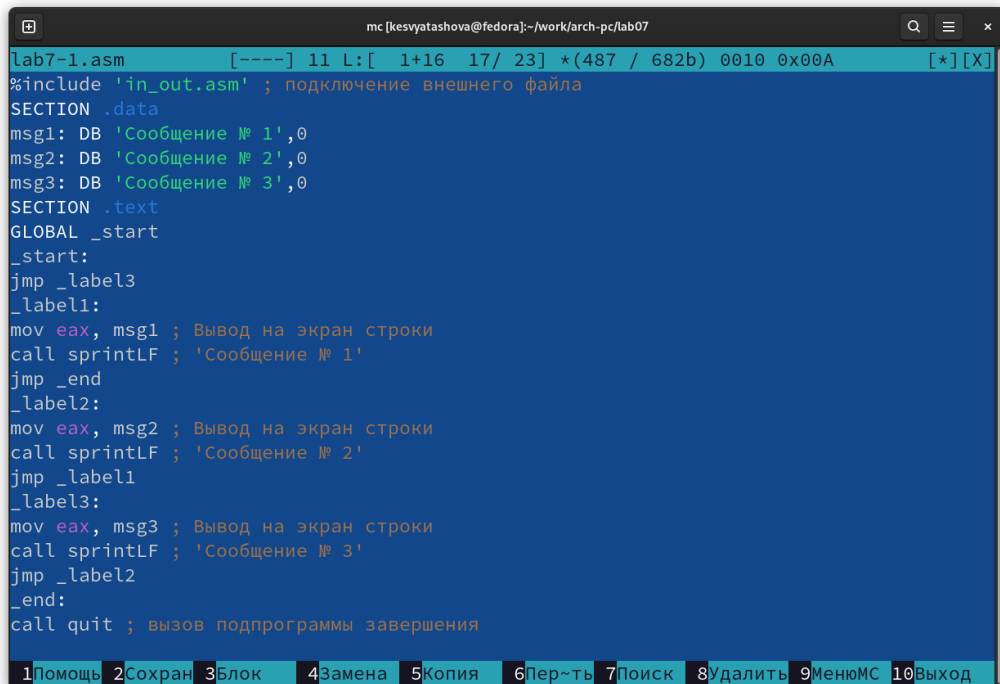
```

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

```

Рис. 3.5: Запуск исполняемого файла

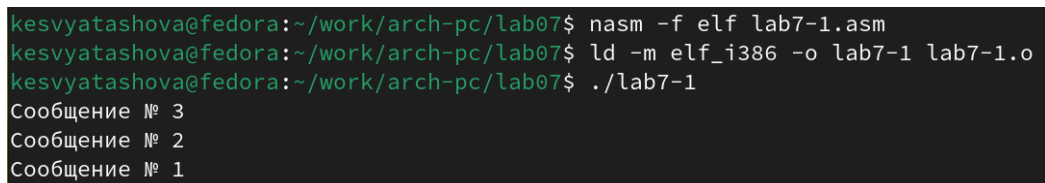
Изменим текст программы(рис. 3.6) добавив или изменив инструкции `jmp`, чтобы вывод программы был следующим(рис. 3.7):



```
lab7-1.asm [----] 11 L: [ 1+16 17/ 23] *(487 / 682b) 0010 0x00A [*][X]
#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 _label3
_label1:
mov eax, msg1 ; Вывод на экран строки
call printf ; 'Сообщение № 1'
jmp _end
_label2:
mov eax, msg2 ; Вывод на экран строки
call printf ; 'Сообщение № 2'
jmp _label1
_label3:
mov eax, msg3 ; Вывод на экран строки
call printf ; 'Сообщение № 3'
jmp _label2
_end:
call quit ; вызов подпрограммы завершения

1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Удалить 9МенюМС 10Выход
```

Рис. 3.6: Изменение текста программы



```
kesvyatashova@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
kesvyatashova@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
kesvyatashova@fedora:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
```

Рис. 3.7: Запуск исполняемого файла

3. . 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(рис. 3.8):

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

Рис. 3.8: Создание файла lab7-2.asm

Далее в этот файл введем текст программы из листинга 7.3(рис. 3.9):

Рис. 3.9: Ввод текста из листинга 7.3

Листинг 7.3. Программа, которая определяет и выводит на экран наибольшую из 3 целочисленных переменных: А,В и С.

```
%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
call atoi ; Вызов подпрограммы перевода символа в число
mov [B],eax ; запись преобразованного числа в 'В'
; ----- Записываем 'А' 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

```

Создадим исполняемый файл и запустим его для разных значений B (рис. 3.10) и (рис. 3.11):


```

kesvyatashova@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
kesvyatashova@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
kesvyatashova@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 7
Наибольшее число: 50
kesvyatashova@fedora:~/work/arch-pc/lab07$

```

Рис. 3.10: Запуск исполняемого файла

```

kesvyatashova@fedora:~/work/arch-pc/lab07$ ./lab7-2
Введите В: 70
Наибольшее число: 70
kesvyatashova@fedora:~/work/arch-pc/lab07$

```

Рис. 3.11: Запуск исполняемого файла

3.2 Изучение структуры файлы листинга

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` (рис. 3.12) и откроем этот файл с помощью текстового редактора `mcedit` (рис. 3.13):

```

kesvyatashova@fedora:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
kesvyatashova@fedora:~/work/arch-pc/lab07$ mcedit lab7-2.lst

```

Рис. 3.12: Создание файла листинга `lab7-2.asm`

```

lab7-2.lst  [----]  0 L:[ 1+ 0 1/225] *(0 /14458b) 0032 0x020 [*] [X]
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          <1> .....
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         <1> .....
14         <1> finished:
15         0000000B 29D8       <1> sub     eax, ebx
16         0000000D 5B         <1> pop     ebx.....
17         0000000E C3         <1> ret.....
18         <1> .
19         <1> ;----- sprint -----
20         <1> ; функция печати сообщения
21         <1> ; входные данные: mov eax,<message>
22         <1> sprint:
23         0000000F 52         <1> push    edx
24         00000010 51         <1> push    ecx
25         00000011 53         <1> push    ebx
26         00000012 50         <1> push    eax

```

Рис. 3.13: Файл листинга

Внимательно ознакомившись с форматом и содержимым файла листинга, можно объяснить содержимое его строк. Например:

- 1) Это строка(рис. 3.14) находится находится на 22 месте, ее адрес “00000106”, машинный код - “E891FFFFFF”. call atoi означает, вызов подпрограммы перевода символа, лежащего в строке выше, в число.

```

21 00000101 B8[0A000000] mov eax,B
22 00000106 E891FFFFFF call atoi
23 0000010B A3[0A000000] mov [B] eax

```

Рис. 3.14: Строка листинга

- 2) Это строка(рис. 3.15) находится находится на 25 месте, ее адрес “00000110”, машинный код - “8B0D[35000000]”. `mov ecx,[A]` означает, что в регистр `ecx` мы записываем число, хранящееся в переменной `A`.

```
24 ; -----  
25 00000110 8B0D[35000000] mov ecx,[A]  
26 00000116 890D[00000000] mov [max],ecx
```

Рис. 3.15: Строка листинга

- 3) Это строка(рис. 3.16) находится находится на 36 месте, ее адрес “00000135”, машинный код - “A3[00000000]”. `mov [max],eax` означает, запись преобразованного числа из регистра `eax` в `max`.

```
35 00000130 E867FFFFFF call atoi ; Вы  
36 00000135 A3[00000000] mov [max],eax  
37 ; ----- C
```

Рис. 3.16: Строка листинга

В строке `mov eax,max` я удалила операнду `max` и попробывала выполнить трансляцию с получением файла листинга(рис. 3.17):

```
kesvyatashova@fedora:~/work/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm  
lab7-2.asm:34: error: invalid combination of opcode and operands  
kesvyatashova@fedora:~/work/arch-pc/lab07$
```

Рис. 3.17: Попытка трансляции файла

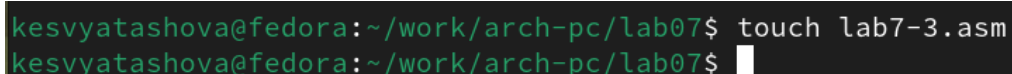
При трансляции файла выходит ошибка, потому что для корректной работы программы нужно два операнда. В файле листинга показывается, где совершена ошибка и почему она выходит(рис. 3.18):

```
34          *****      error: invalid combination of opcode and operands
35 00000130 E867FFFFFF      call atoi ; Вызов подпрограммы перевода символа в число
36 00000135 A3[00000000]      mov [max],eax ; запись преобразованного числа в 'max'
```

Рис. 3.18: Ошибка в файле листинга

4 Задания для самостоятельной работы

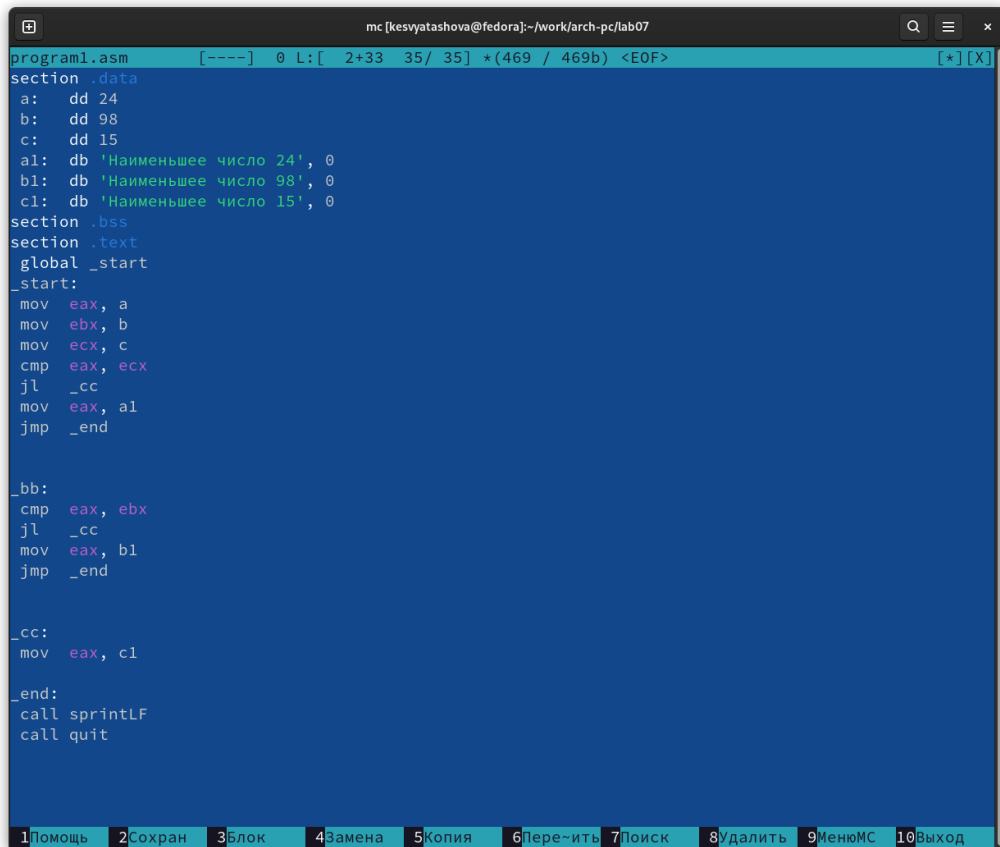
1. Создадим файл lab7-3.asm для выполнения первого задания самостоятельной работы(рис. 4.1):



```
kesvyatashova@fedora:~/work/arch-pc/lab07$ touch lab7-3.asm
kesvyatashova@fedora:~/work/arch-pc/lab07$
```

Рис. 4.1: Создание файла lab7-3.asm

Напишем программу(рис. 4.2) нахождения наименьшей из 3 целочисленных переменных a,b и c. Значения переменных я выбрала в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Мой вариант - 9.



```
mc [kesvyatashova@fedora:~/work/arch-pc/lab07]
program1.asm  [----]  0 L: [ 2+33 35/ 35] *(469 / 469b) <EOF>  [*][X]
section .data
a: dd 24
b: dd 98
c: dd 15
a1: db 'Наименьшее число 24', 0
b1: db 'Наименьшее число 98', 0
c1: db 'Наименьшее число 15', 0
section .bss
section .text
global _start
_start:
mov eax, a
mov ebx, b
mov ecx, c
cmp eax, ecx
jl _cc
mov eax, a1
jmp _end

_bb:
cmp eax, ebx
jl _cc
mov eax, b1
jmp _end

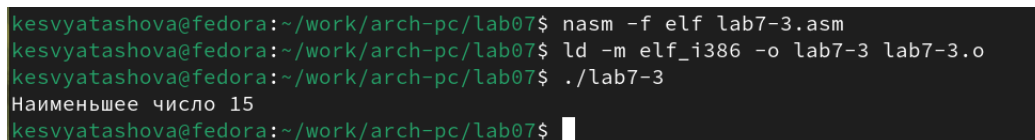
_cc:
mov eax, c1

_end:
call sprintf
call quit

1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пере-ить 7Поиск 8Удалить 9МенюМС 10Выход
```

Рис. 4.2: Программа вычисления выражение из варианта 9

Создадим исполняемый файл и проверим его работу(рис. 4.3)



```
kesvyatashova@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-3.asm
kesvyatashova@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.o
kesvyatashova@fedora:~/work/arch-pc/lab07$ ./lab7-3
Наименьшее число 15
kesvyatashova@fedora:~/work/arch-pc/lab07$
```

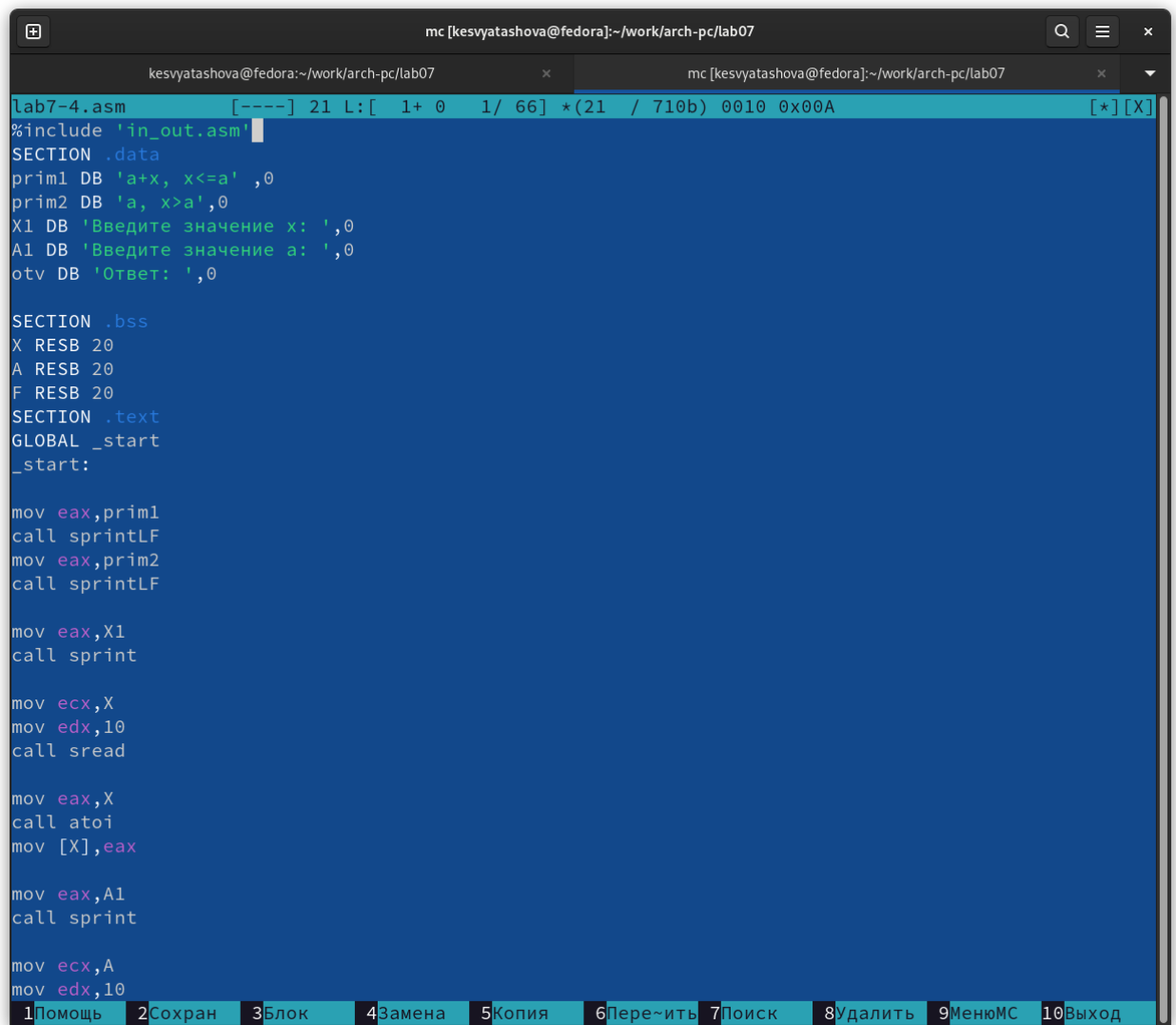
Рис. 4.3: Запуск исполняемого файла

2. Создадим файл lab7-4.asm для выполнения второго задания самостоятельной работы(рис. 4.4):

```
kesvyatashova@fedora:~/work/arch-pc/lab07$ touch lab7-4.asm  
kesvyatashova@fedora:~/work/arch-pc/lab07$
```

Рис. 4.4: Создание файла lab7-4.asm

Напишем программу(рис. 4.5), которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции $f(x)$ и выводит результат вычислений. Вид функции $f(x)$ я выбрала также в соответствии с вариантом, полученным при выполнении лабораторной работы № 6.



```
lab7-4.asm [----] 21 L: [ 1+ 0 1/ 66] *(21 / 710b) 0010 0x00A [*] [X]
#include 'in_out.asm'
SECTION .data
prim1 DB 'a+x, x<=a' ,0
prim2 DB 'a, x>a',0
X1 DB 'Введите значение x: ',0
A1 DB 'Введите значение a: ',0
otv DB '0ответ: ',0

SECTION .bss
X RESB 20
A RESB 20
F RESB 20
SECTION .text
GLOBAL _start
_start:

mov eax,prim1
call sprintLF
mov eax,prim2
call sprintLF

mov eax,X1
call sprint

mov ecx,X
mov edx,10
call sread

mov eax,X
call atoi
mov [X],eax

mov eax,A1
call sprint

mov ecx,A
mov edx,10
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пере-ить 7Поиск 8Удалить 9МенюМС 10Выход
```

Рис. 4.5: Программа вычисления выражение из варианта 9

Создадим исполняемый файл и проверьте его работу для значений x и a(рис. 4.6) и (рис. 4.7):


```

kesvyatashova@fedora:~/work/arch-pc/lab07$ nasm -f elf lab7-4.asm
kesvyatashova@fedora:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-4 lab7-4.o
kesvyatashova@fedora:~/work/arch-pc/lab07$ ./lab7-4
a+x, x<=a
a, x>a
Введите значение x: 5
Введите значение a: 7
Ответ: 12
kesvyatashova@fedora:~/work/arch-pc/lab07$ █

```

Рис. 4.6: Запуск исполняемого файла

```

kesvyatashova@fedora:~/work/arch-pc/lab07$ ./lab7-4
a+x, x<=a
a, x>a
Введите значение x: 6
Введите значение a: 4
Ответ: 4
kesvyatashova@fedora:~/work/arch-pc/lab07$

```

Рис. 4.7: Запуск исполняемого файла

Текст программы:

```

%include 'in_out.asm'

SECTION .data
prim1 DB 'a+x, x<=a',0
prim2 DB 'a, x>a',0
X1 DB 'Введите значение x:',0
A1 DB 'Введите значение a:',0
otv DB 'Ответ:',0

```

```

SECTION .bss
X RESB 20
A RESB 20
F RESB 20
SECTION .text
GLOBAL _start
_start:
mov eax,prim1
call sprintLF
mov eax,prim2
call sprintLF
mov eax,X1
call sprint
mov ecx,X
mov edx,10
call sread
mov eax,X
call atoi
mov [X],eax
mov eax,A1
call sprint
mov ecx,A
mov edx,10
call sread
mov eax,A
call atoi
mov [A],eax

```

```
mov ecx,[X]
mov [F],ecx
cmp [A],ecx
jg check_or
mov eax,[A]
mov [F],eax
jmp fin
check_or:
mov eax,[X]
add eax,[A]
mov [F],eax
fin:
mov eax,otv
call sprint
mov eax,[F]
call iprintLF call quit
```

5 Вывод

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