

ОтчеТт по лабораторной работе №7

дисциплина: Архитектура компьютера

Дельгадильо Валерия

Содержание

1	Цель работы	5
2	Теоретическое введение	6
2.1	Команды безусловного перехода	6
3	Лабораторной работы	7
3.1	Реализация переходов в NASM	7
3.2	Изучение структуры файлы листинга	13
4	Задание для самостоятельной работы	17
5	Выводы	22
6	Список литературы	23

Список иллюстраций

3.1	7
3.2	8
3.3	8
3.4	9
3.5	10
3.6	10
3.7	11
3.8	12
3.9	13
3.10	13
3.11	14
3.12	15
3.13	16
4.1	18
4.2	18
4.3	20
4.4	21

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 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.

2.1 Команды безусловного перехода

Безусловный переход выполняется инструкцией `jmp` (от англ. `jump` – прыжок), которая включает в себя адрес перехода, куда следует передать управление:

```
jmp <адрес_перехода>
```

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.

3 Лабораторной работы

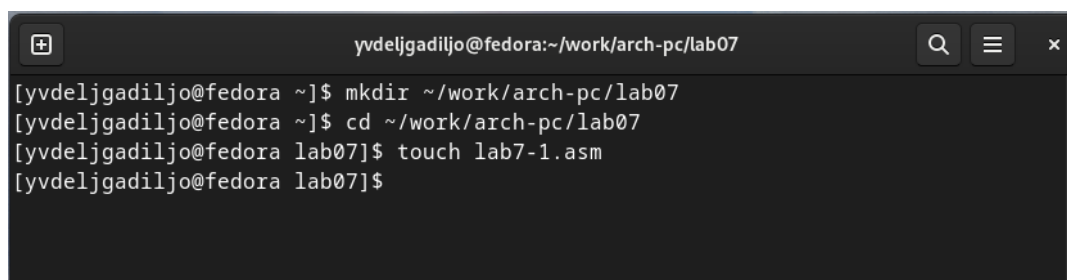
3.1 Реализация переходов в NASM

Создайте каталог для программ лабораторной работы № 7, перейдите в него и создайте файл lab7-1.asm:

```
mkdir ~/work/arch-pc/lab07
```

```
cd ~/work/arch-pc/lab07
```

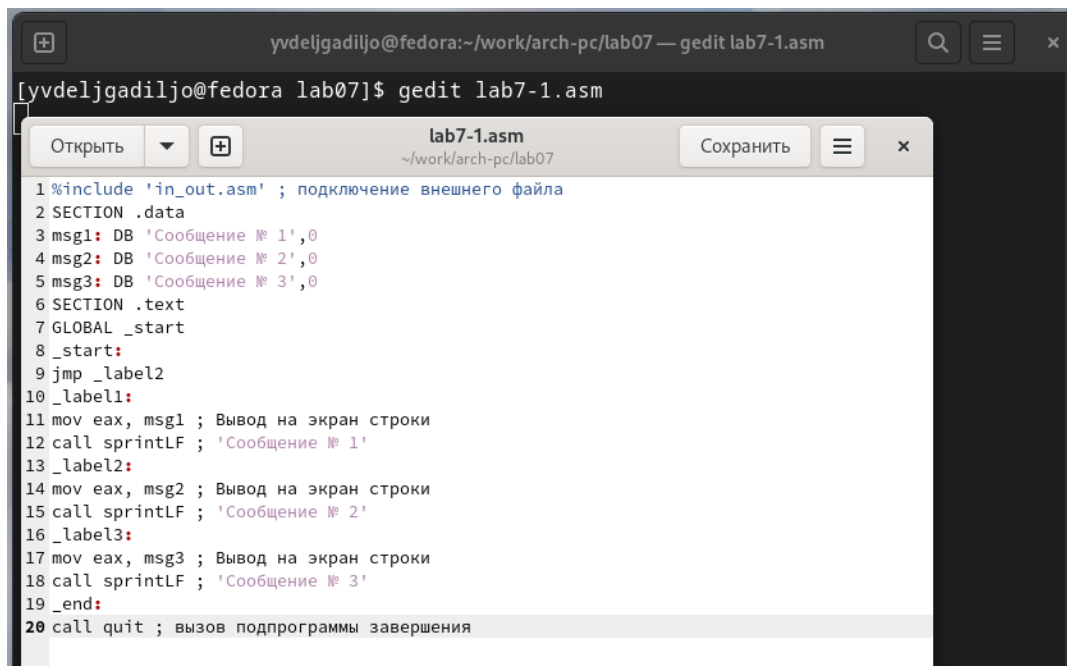
```
touch lab7-1.asm
```

A screenshot of a terminal window with a dark background. The window title is 'yvdeljgadiljo@fedora:~/work/arch-pc/lab07'. The terminal shows the following commands and their outputs: [yvdeljgadiljo@fedora ~]\$ mkdir ~/work/arch-pc/lab07, [yvdeljgadiljo@fedora ~]\$ cd ~/work/arch-pc/lab07, [yvdeljgadiljo@fedora lab07]\$ touch lab7-1.asm, and [yvdeljgadiljo@fedora lab07]\$.

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

Рис. 3.1:

Инструкция `jmp` в NASM используется для реализации безусловных переходов. Рассмотрим пример программы с использованием инструкции `jmp`. Введите в файл lab7-1.asm текст программы из листинга 7.1.



```
[yvdeljgadiljo@fedora lab07]$ gedit lab7-1.asm

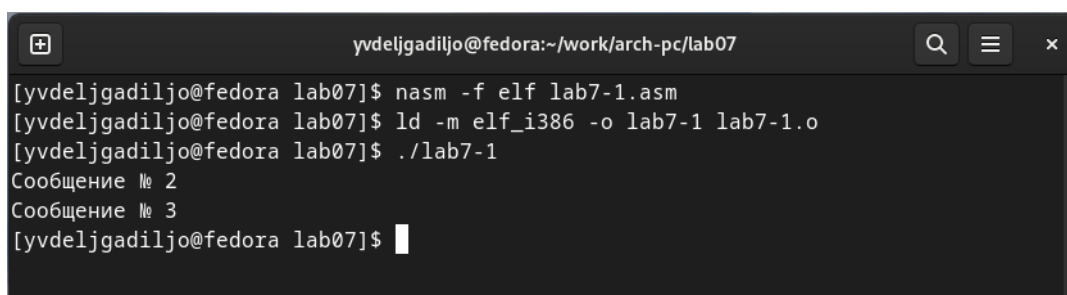
lab7-1.asm
~/work/arch-pc/lab07

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 _label2
10 _label1:
11 mov eax, msg1 ; Вывод на экран строки
12 call sprintf ; 'Сообщение № 1'
13 _label2:
14 mov eax, msg2 ; Вывод на экран строки
15 call sprintf ; 'Сообщение № 2'
16 _label3:
17 mov eax, msg3 ; Вывод на экран строки
18 call sprintf ; 'Сообщение № 3'
19 _end:
20 call quit ; вызов подпрограммы завершения
```

Рис. 3.2:

Создайте исполняемый файл и запустите его. Результат работы данной программы будет следующим:

```
user@dk4n31:~$ ./lab7-1
Сообщение № 2
Сообщение № 3
user@dk4n31:~$
```



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

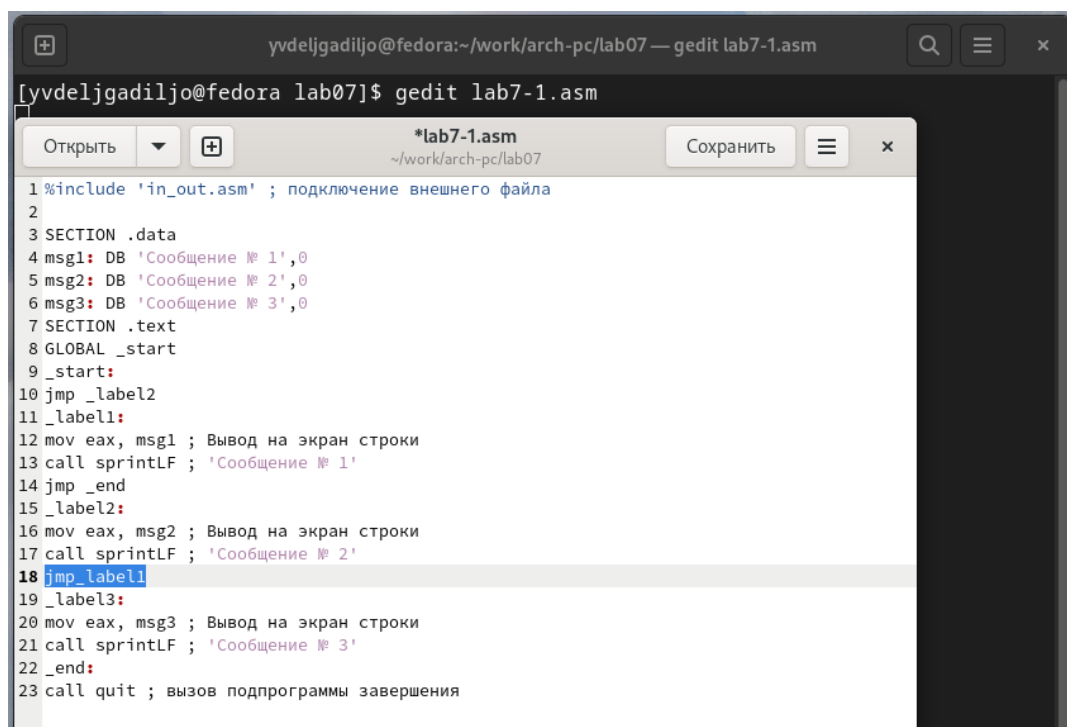
Рис. 3.3:

Таким образом, использование инструкции `jmp _label2` меняет порядок испол-

нения

инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения.

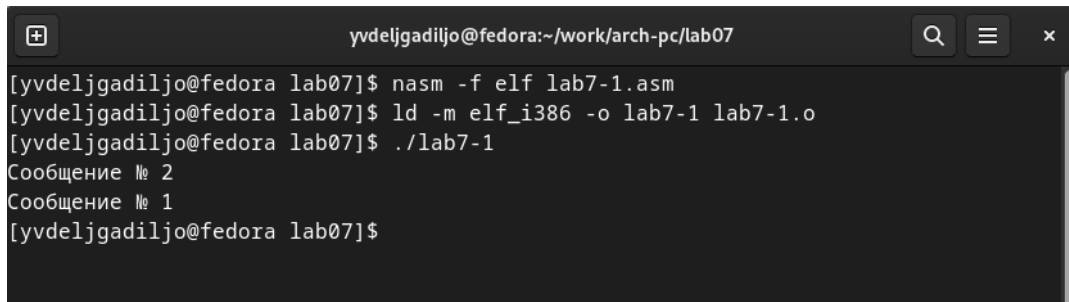
Инструкция `jmp` позволяет осуществлять переходы не только вперед но и назад. Изменим программу таким образом, чтобы она выводила сначала 'Сообщение № 2', потом 'Сообщение № 1' и завершала работу. Для этого в текст программы после вывода сообщения № 2 добавим инструкцию `jmp` с меткой `_label1` (т.е. переход к инструкциям вывода сообщения № 1) и после вывода сообщения № 1 добавим инструкцию `jmp` с меткой `_end` (т.е. переход к инструкции `call quit`). Измените текст программы в соответствии с листингом 7.2



```
[yvdeljgadiljo@fedora lab07]$ gedit lab7-1.asm
1 %include 'in_out.asm' ; подключение внешнего файла
2
3 SECTION .data
4 msg1: DB 'Сообщение № 1',0
5 msg2: DB 'Сообщение № 2',0
6 msg3: DB 'Сообщение № 3',0
7 SECTION .text
8 GLOBAL _start
9 _start:
10 jmp _label2
11 _label1:
12 mov eax, msg1 ; Вывод на экран строки
13 call sprintf ; 'Сообщение № 1'
14 jmp _end
15 _label2:
16 mov eax, msg2 ; Вывод на экран строки
17 call sprintf ; 'Сообщение № 2'
18 jmp _label1
19 _label3:
20 mov eax, msg3 ; Вывод на экран строки
21 call sprintf ; 'Сообщение № 3'
22 _end:
23 call quit ; вызов подпрограммы завершения
```

Рис. 3.4:

Создайте исполняемый файл и проверьте его работу.

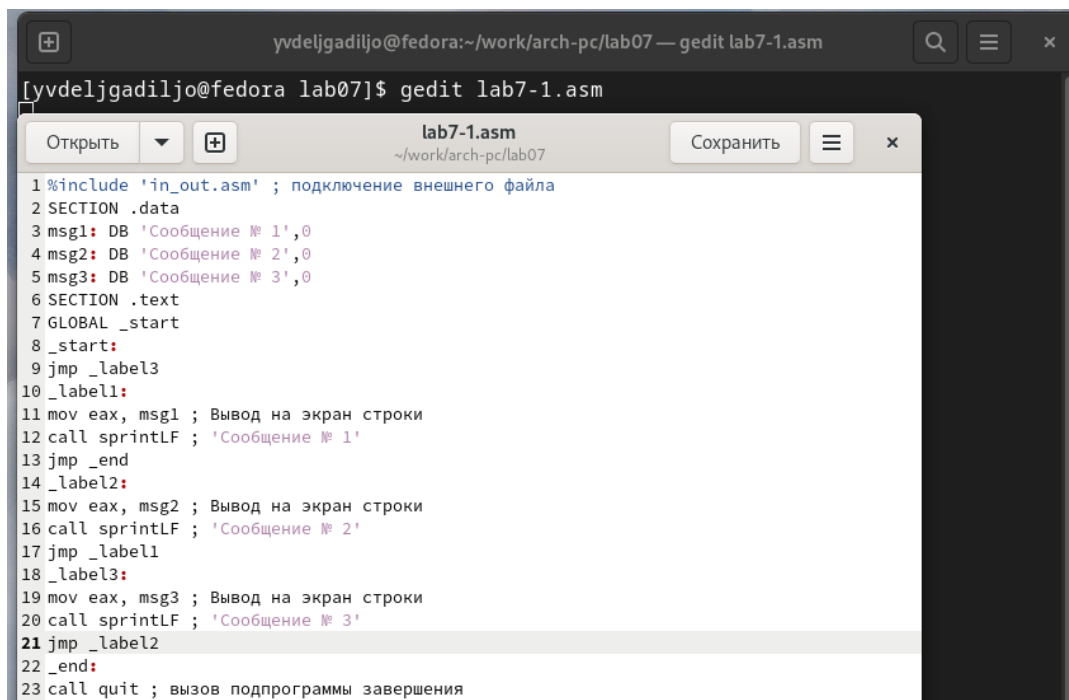


```
yvdeljgadiljo@fedora:~/work/arch-pc/lab07
[yvdeljgadiljo@fedora lab07]$ nasm -f elf lab7-1.asm
[yvdeljgadiljo@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[yvdeljgadiljo@fedora lab07]$ ./lab7-1
Сообщение № 2
Сообщение № 1
[yvdeljgadiljo@fedora lab07]$
```

Рис. 3.5:

Измените текст программы добавив или изменив инструкции `jmp`, чтобы вывод программы был следующим:

```
user@dk4n31:~$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
user@dk4n31:~$
```

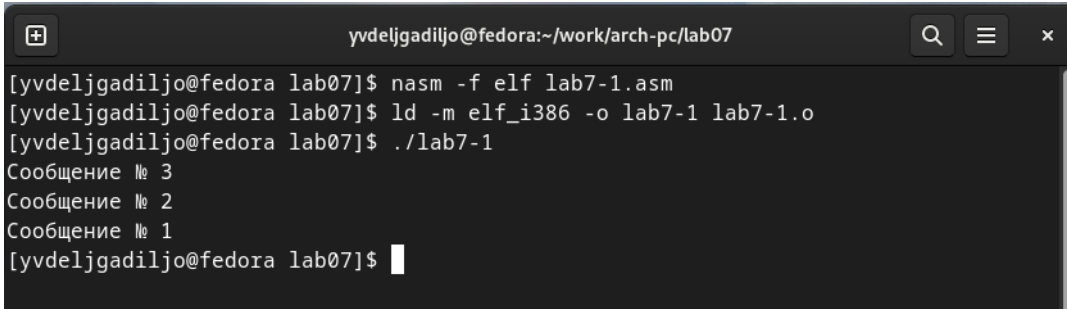


```
yvdeljgadiljo@fedora:~/work/arch-pc/lab07 — gedit lab7-1.asm
[yvdeljgadiljo@fedora lab07]$ gedit lab7-1.asm

lab7-1.asm
~/work/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. 3.6:

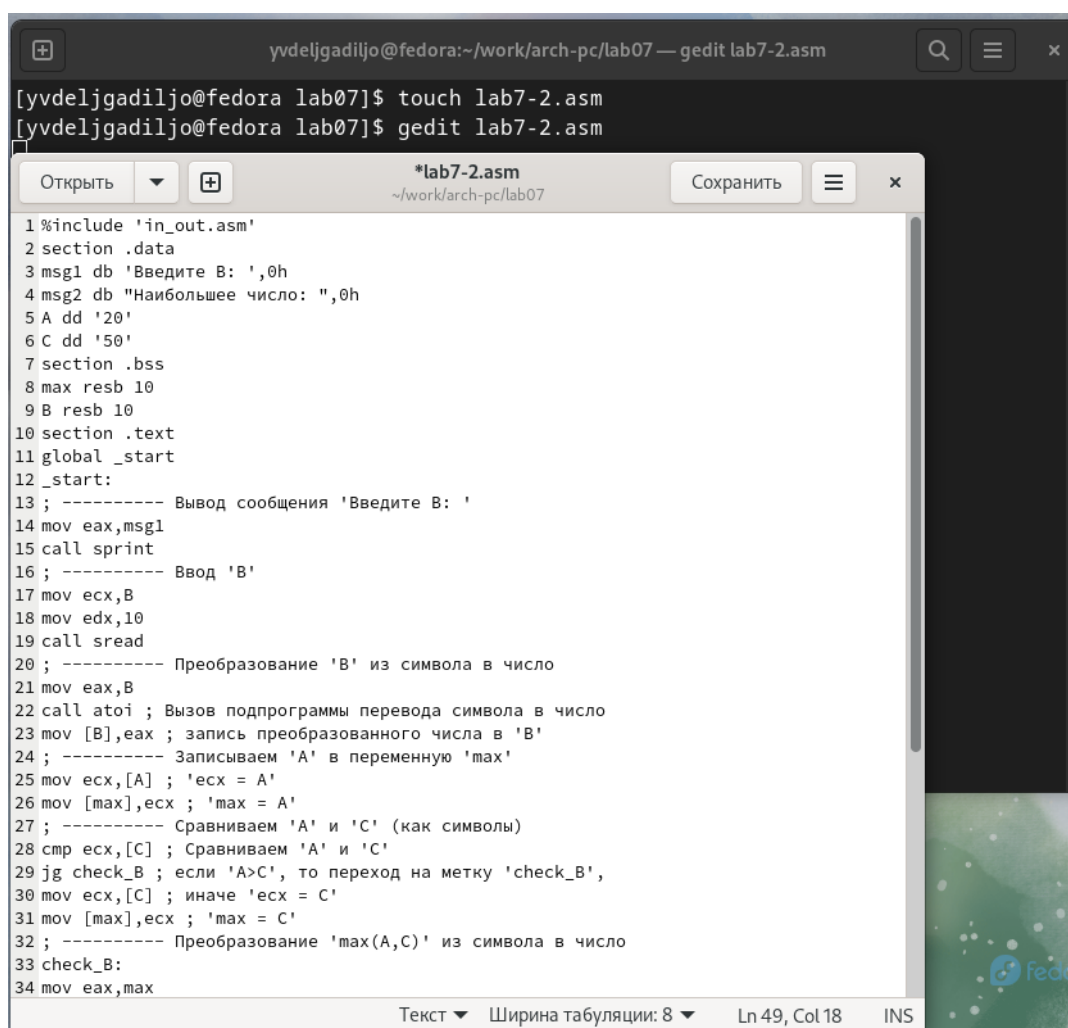
A terminal window with a dark background. The title bar shows the user 'yvdeljgatiljo' on a 'fedora' machine, in the directory '~/work/arch-pc/lab07'. The terminal contains the following text:

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

Рис. 3.7:

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.

Создайте файл `lab7-2.asm` в каталоге `~/work/arch-pc/lab07`. Внимательно изучите текст программы из листинга 7.3 и введите в `lab7-2.asm`.



The image shows a terminal window at the top and a text editor window below it. The terminal shows the user creating and editing a file named lab7-2.asm. The text editor displays the following assembly code:

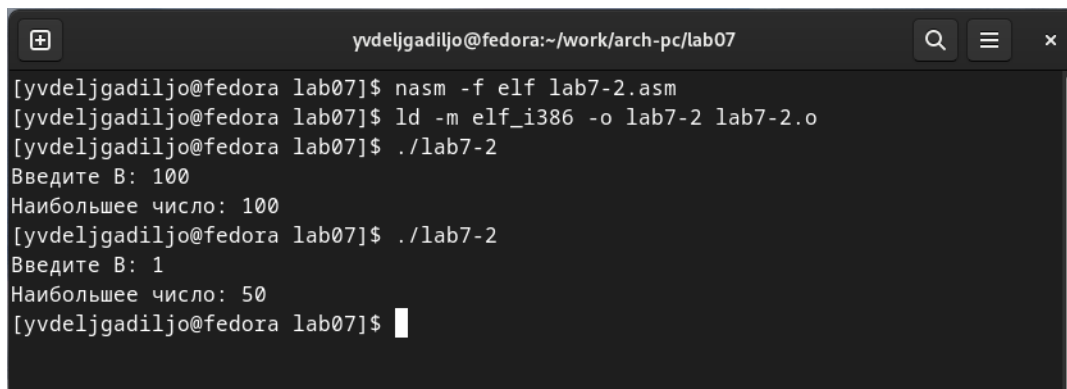
```
1 %include 'in_out.asm'
2 section .data
3 msg1 db 'Введите B: ',0h
4 msg2 db "Наибольшее число: ",0h
5 A dd '20'
6 C dd '50'
7 section .bss
8 max resb 10
9 B resb 10
10 section .text
11 global _start
12 _start:
13 ; ----- Вывод сообщения 'Введите B: '
14 mov eax,msg1
15 call sprint
16 ; ----- Ввод 'B'
17 mov ecx,B
18 mov edx,10
19 call sread
20 ; ----- Преобразование 'B' из символа в число
21 mov eax,B
22 call atoi ; Вызов подпрограммы перевода символа в число
23 mov [B],eax ; запись преобразованного числа в 'B'
24 ; ----- Записываем 'A' в переменную 'max'
25 mov ecx,[A] ; 'ecx = A'
26 mov [max],ecx ; 'max = A'
27 ; ----- Сравниваем 'A' и 'C' (как символы)
28 cmp ecx,[C] ; Сравниваем 'A' и 'C'
29 jg check_B ; если 'A>C', то переход на метку 'check_B',
30 mov ecx,[C] ; иначе 'ecx = C'
31 mov [max],ecx ; 'max = C'
32 ; ----- Преобразование 'max(A,C)' из символа в число
33 check_B:
34 mov eax,max
```

The status bar at the bottom of the text editor indicates: Текст, Ширина табуляции: 8, Ln 49, Col 18, INS.

Рис. 3.8:

Создайте исполняемый файл и проверьте его работу для разных значений B. Обратите внимание, в данном примере переменные A и C сравниваются как символы, а переменная B и максимум из A и C как числа (для этого используется функция `atoi` преобразования символа в число). Это сделано для демонстрации того, как сравниваются данные.

Данную программу можно упростить и сравнивать все 3 переменные как символы (т.е. не использовать функцию `atoi`). Однако если переменные преобразовать из символов числа, над ними можно корректно проводить арифметические операции.



```
yvdeljgatiljo@fedora:~/work/arch-pc/lab07
[yvdeljgatiljo@fedora lab07]$ nasm -f elf lab7-2.asm
[yvdeljgatiljo@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[yvdeljgatiljo@fedora lab07]$ ./lab7-2
Введите В: 100
Наибольшее число: 100
[yvdeljgatiljo@fedora lab07]$ ./lab7-2
Введите В: 1
Наибольшее число: 50
[yvdeljgatiljo@fedora lab07]$
```

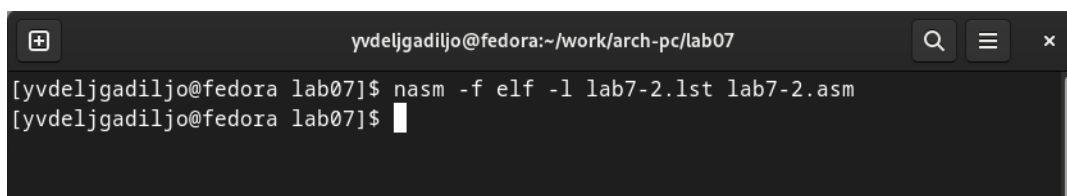
Рис. 3.9:

3.2 Изучение структуры файлы листинга

Обычно `nasm` создаёт в результате ассемблирования только объектный файл. Получить файл листинга можно, указав ключ `-l` и задав имя файла листинга в командной строке.

Создайте файл листинга для программы из файла `lab7-2.asm`

```
nasm -f elf -l lab7-2.lst lab7-2.asm
```



```
yvdeljgatiljo@fedora:~/work/arch-pc/lab07
[yvdeljgatiljo@fedora lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm
[yvdeljgatiljo@fedora lab07]$
```

Рис. 3.10:

Откройте файл листинга `lab7-2.lst` с помощью любого текстового редактора, например `mcedit`:

```
mcedit lab7-2.lst
```

```

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
27    26      00000012 50        <1>      push     eax
28    27      00000013 E8E8FFFF    <1>      call    slen
29    28      <1>
30    29      00000018 89C2      <1>      mov     edx, eax
31    30      0000001A 58        <1>      pop     eax
32    31      <1>

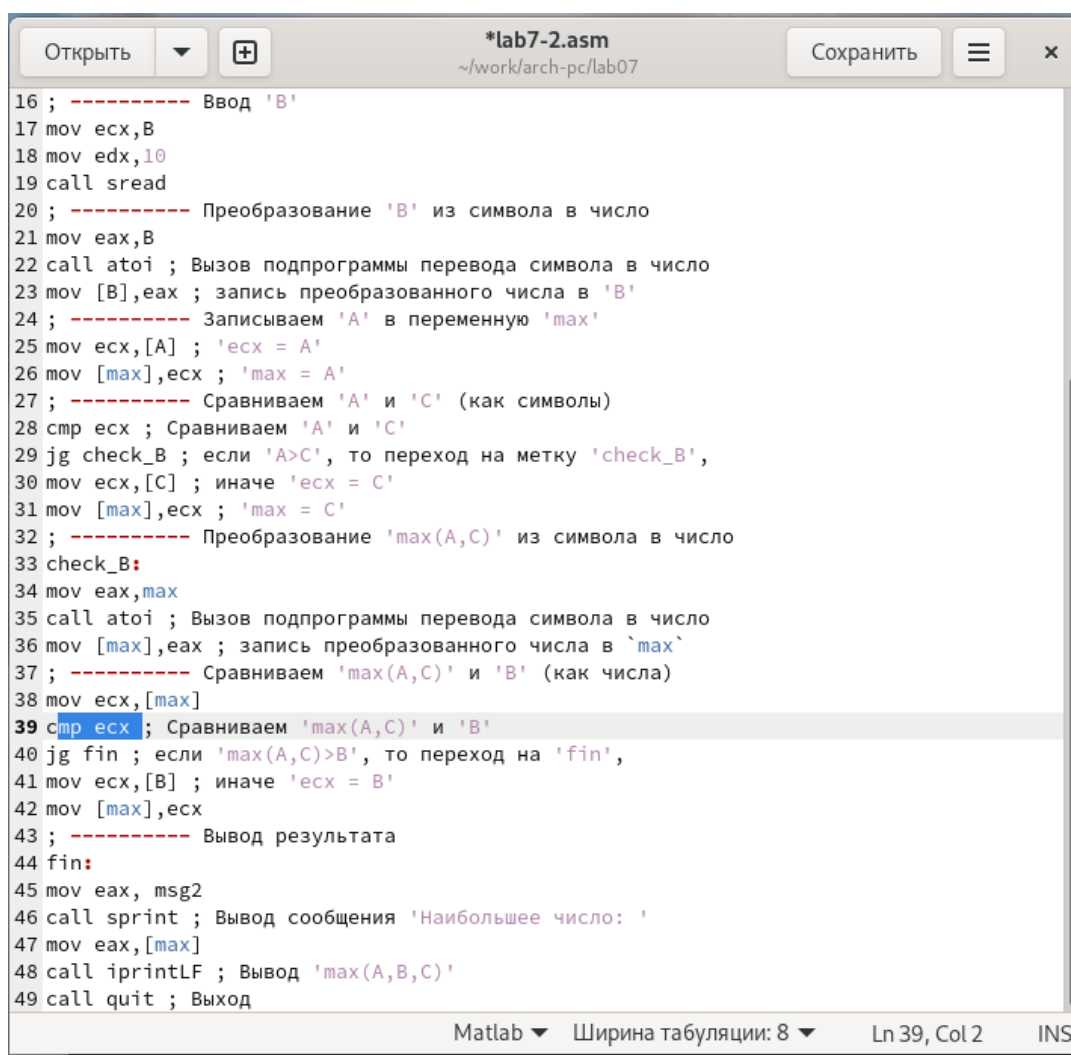
```

Рис. 3.11:

Внимательно ознакомиться с его форматом и содержимым. Подробно объяснить содержимое трёх строк файла листинга по выбору.

в строке 9 содержится собственно номер строки [9], адресс [00000003], машинный код [803800] и содержимое строки кода [cmp byte [eax], 0] в строке 11 содержится номер строки [11], адресс [00000008], машинный код [40] и содержимое строки кода [inc eax] в строке 24 содержится номер строки [24], адресс [0000000F], машинный код [52] и содержимое строки кода [push edx].

Откройте файл с программой lab7-2.asm и в любой инструкции с двумя операндами удалить один операнд.



```
16 ; ----- Ввод 'B'
17 mov ecx,B
18 mov edx,10
19 call sread
20 ; ----- Преобразование 'B' из символа в число
21 mov eax,B
22 call atoi ; Вызов подпрограммы перевода символа в число
23 mov [B],eax ; запись преобразованного числа в 'B'
24 ; ----- Записываем 'A' в переменную 'max'
25 mov ecx,[A] ; 'ecx = A'
26 mov [max],ecx ; 'max = A'
27 ; ----- Сравниваем 'A' и 'C' (как символы)
28 cmp ecx ; Сравниваем 'A' и 'C'
29 jg check_B ; если 'A>C', то переход на метку 'check_B',
30 mov ecx,[C] ; иначе 'ecx = C'
31 mov [max],ecx ; 'max = C'
32 ; ----- Преобразование 'max(A,C)' из символа в число
33 check_B:
34 mov eax,max
35 call atoi ; Вызов подпрограммы перевода символа в число
36 mov [max],eax ; запись преобразованного числа в 'max'
37 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
38 mov ecx,[max]
39 cmp ecx ; Сравниваем 'max(A,C)' и 'B'
40 jg fin ; если 'max(A,C)>B', то переход на 'fin',
41 mov ecx,[B] ; иначе 'ecx = B'
42 mov [max],ecx
43 ; ----- Вывод результата
44 fin:
45 mov eax, msg2
46 call sprintf ; Вывод сообщения 'Наибольшее число: '
47 mov eax,[max]
48 call fprintf ; Вывод 'max(A,B,C)'
49 call quit ; Выход
```

Рис. 3.12:

Выполните трансляцию с получением файла листинга:

```
nasm -f elf -l lab7-2.lst lab7-2.asm
```

Какие выходные файлы создаются в этом случае? Что добавляется в листинге?

Ошибка в файле листинга

```

yvdeljgadijlo@fedora:~/work/arch-pc/lab07 — gedit lab7-2.lst
[yvdeljgadijlo@fedora lab07]$ nasm -f elf -l lab7-2.lst lab7-2.asm

lab7-2.lst
~/work/arch-pc/lab07
Открыть Сохранить

201 26 00000116 890D[00000000] mov [max],ecx ; 'max = A'
202 27 ; ----- Сравниваем 'A' и 'C' (как символы)
203 28 cmp ecx ; Сравниваем 'A' и 'C'
204 28 *****
205 29 0000011C 7F0C jg check_B ; если 'A>C', то переход на метку 'check_B',
206 30 0000011E 8B0D[39000000] mov ecx,[C] ; иначе 'ecx = C'
207 31 00000124 890D[00000000] mov [max],ecx ; 'max = C'
208 32 ; ----- Преобразование 'max(A,C)' из символа в число
209 33 check_B:
210 34 0000012A B8[00000000] mov eax,max
211 35 0000012F E868FFFFFF call atoi ; Вызов подпрограммы перевода символа в число
212 36 00000134 A3[00000000] mov [max],eax ; запись преобразованного числа в 'max'
213 37 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
214 38 00000139 8B0D[00000000] mov ecx,[max]
215 39 cmp ecx ; Сравниваем 'max(A,C)' и 'B'
216 39 *****
217 40 0000013F 7F0C jg fin ; если 'max(A,C)>B', то переход на 'fin',
218 41 00000141 8B0D[0A000000] mov ecx,[B] ; иначе 'ecx = B'
219 42 00000147 890D[00000000] mov [max],ecx
220 43 ; ----- Вывод результата
221 44 fin:
222 45 0000014D B8[13000000] mov eax,msg2
223 46 00000152 E8B8FFFFFF call sprint ; Вывод сообщения 'Наибольшее число: '
224 47 00000157 A1[00000000] mov eax,[max]
225 48 0000015C E825FFFFFF call iprintLF ; Вывод 'max(A,B,C)'
226 49 00000161 E875FFFFFF call quit ; Выход

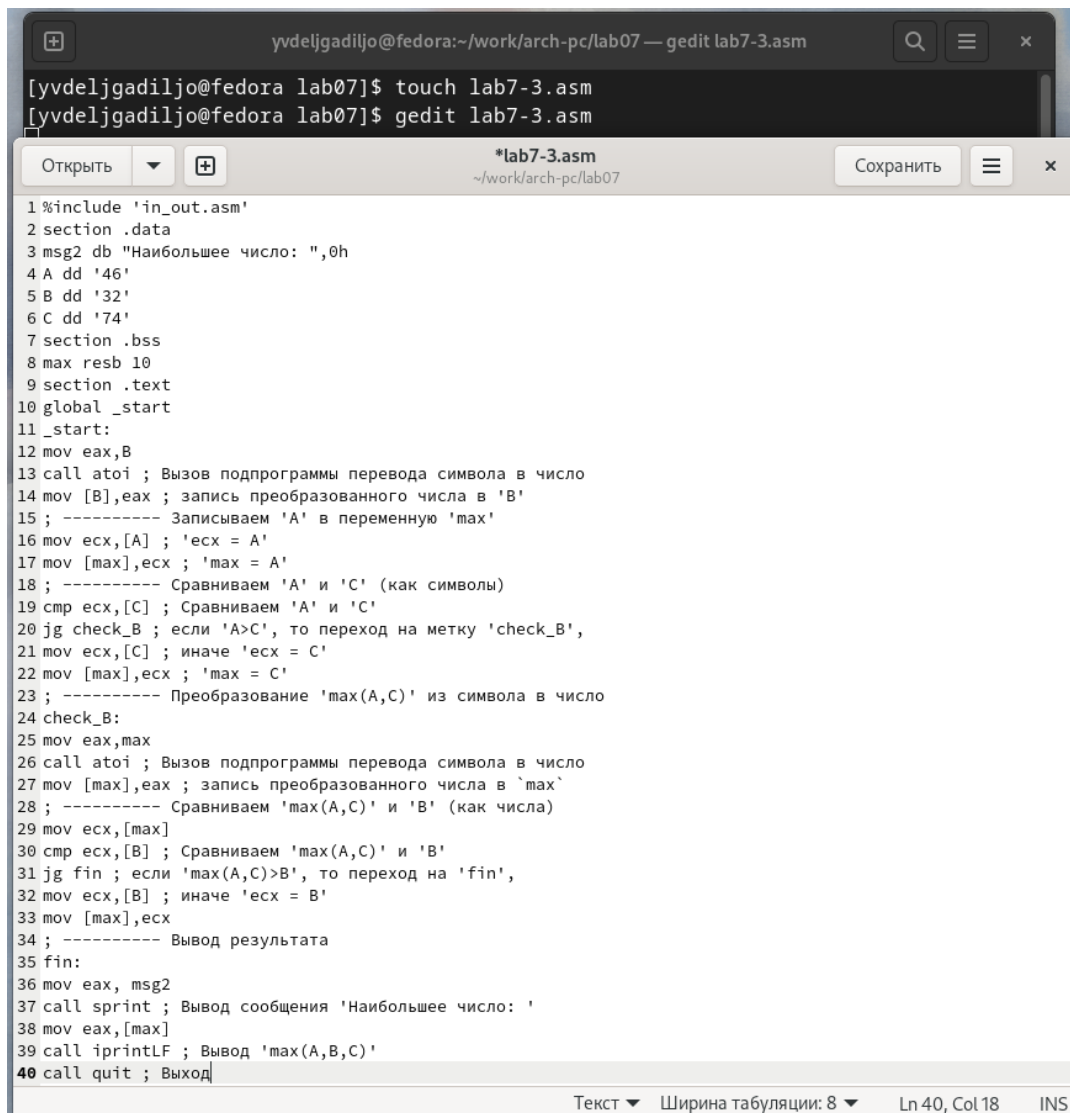
Текст Ширина табуляции: 8 Ln 2, Col 65

```

Рис. 3.13:

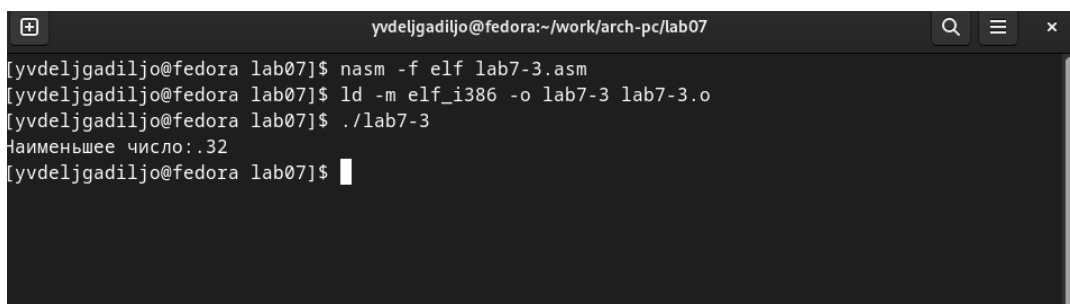
4 Задание для самостоятельной работы

Напишите программу нахождения наименьшей из 3 целочисленных переменных a , b и c . Значения переменных выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 7 (Вар 19). Создайте исполняемый файл и проверьте его работу.



```
1 %include 'in_out.asm'
2 section .data
3 msg2 db "Наибольшее число: ",0h
4 A dd '46'
5 B dd '32'
6 C dd '74'
7 section .bss
8 max resb 10
9 section .text
10 global _start
11 _start:
12 mov eax,B
13 call atoi ; Вызов подпрограммы перевода символа в число
14 mov [B],eax ; запись преобразованного числа в 'B'
15 ; ----- Записываем 'A' в переменную 'max'
16 mov ecx,[A] ; 'ecx = A'
17 mov [max],ecx ; 'max = A'
18 ; ----- Сравниваем 'A' и 'C' (как символы)
19 cmp ecx,[C] ; Сравниваем 'A' и 'C'
20 jg check_B ; если 'A>C', то переход на метку 'check_B',
21 mov ecx,[C] ; иначе 'ecx = C'
22 mov [max],ecx ; 'max = C'
23 ; ----- Преобразование 'max(A,C)' из символа в число
24 check_B:
25 mov eax,max
26 call atoi ; Вызов подпрограммы перевода символа в число
27 mov [max],eax ; запись преобразованного числа в 'max'
28 ; ----- Сравниваем 'max(A,C)' и 'B' (как числа)
29 mov ecx,[max]
30 cmp ecx,[B] ; Сравниваем 'max(A,C)' и 'B'
31 jg fin ; если 'max(A,C)>B', то переход на 'fin',
32 mov ecx,[B] ; иначе 'ecx = B'
33 mov [max],ecx
34 ; ----- Вывод результата
35 fin:
36 mov eax, msg2
37 call sprint ; Вывод сообщения 'Наибольшее число: '
38 mov eax,[max]
39 call iprintLF ; Вывод 'max(A,B,C)'
40 call quit ; Выход
```

Рис. 4.1:



```
[yvdeljgadiljo@fedora lab07]$ nasm -f elf lab7-3.asm
[yvdeljgadiljo@fedora lab07]$ ld -m elf_i386 -o lab7-3 lab7-3.o
[yvdeljgadiljo@fedora lab07]$ ./lab7-3
Наименьшее число: .32
[yvdeljgadiljo@fedora lab07]$
```

Рис. 4.2:

Напишите программу, которая для введенных с клавиатуры значений x и a вычисляет

значение заданной функции $f(x)$ и выводит результат вычислений. Вид функции $f(x)$

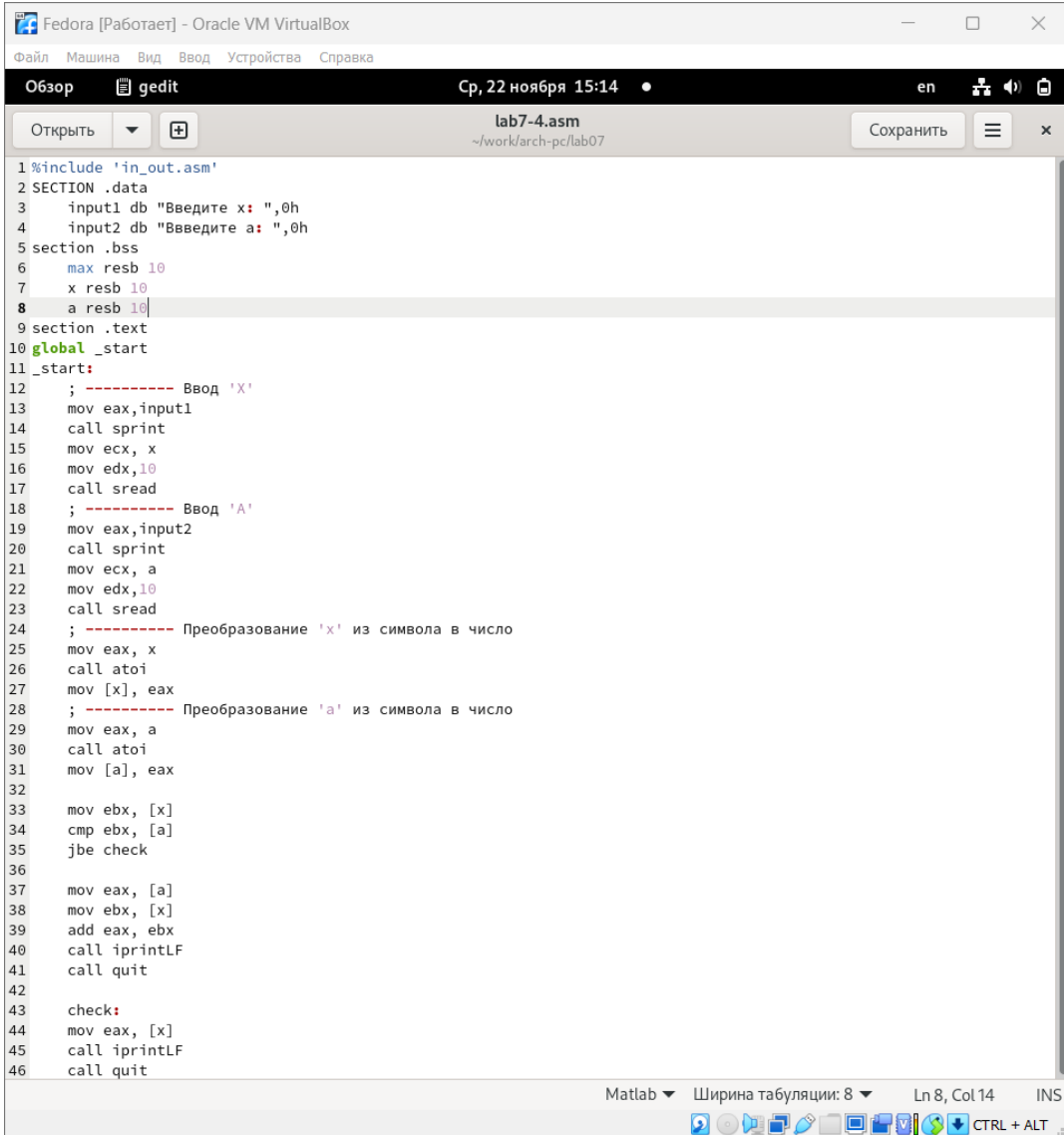
выбрать из таблицы 7.6 вариантов заданий в соответствии с вариантом, полученным

при выполнении лабораторной работы № 7. Создайте исполняемый файл и проверьте

его работу для значений x и a из 7.6. Вар 19:

$a + x, x > a$

$x, x \leq a$ Код программы



Fedora [Работает] - Oracle VM VirtualBox

Файл Машина Вид Ввод Устройства Справка

Обзор gedit Ср, 22 ноября 15:14 en Сохранить

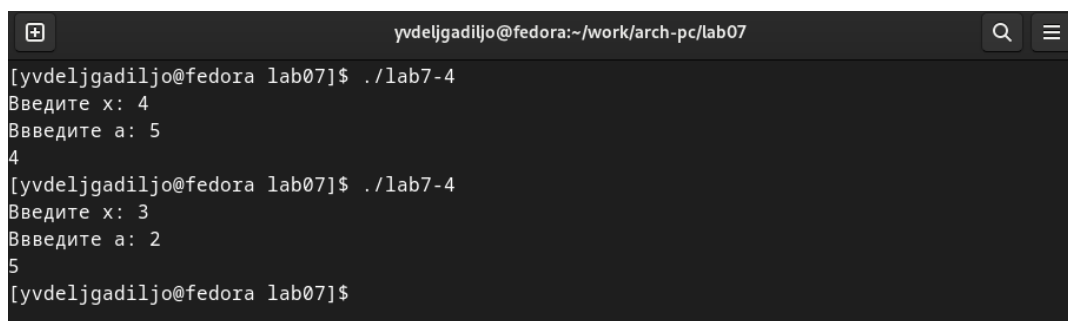
Открыть lab7-4.asm ~/work/arch-pc/lab07

```
1 %include 'in_out.asm'
2 SECTION .data
3     input1 db "Введите x: ",0h
4     input2 db "Введите a: ",0h
5 section .bss
6     max resb 10
7     x resb 10
8     a resb 10
9 section .text
10 global _start
11 _start:
12     ; ----- Ввод 'X'
13     mov eax,input1
14     call sprint
15     mov ecx, x
16     mov edx,10
17     call sread
18     ; ----- Ввод 'A'
19     mov eax,input2
20     call sprint
21     mov ecx, a
22     mov edx,10
23     call sread
24     ; ----- Преобразование 'x' из символа в число
25     mov eax, x
26     call atoi
27     mov [x], eax
28     ; ----- Преобразование 'a' из символа в число
29     mov eax, a
30     call atoi
31     mov [a], eax
32
33     mov ebx, [x]
34     cmp ebx, [a]
35     jbe check
36
37     mov eax, [a]
38     mov ebx, [x]
39     add eax, ebx
40     call iprintLF
41     call quit
42
43     check:
44     mov eax, [x]
45     call iprintLF
46     call quit
```

Matlab Ширина табуляции: 8 Ln 8, Col 14 INS CTRL + ALT

Рис. 4.3:

Результат выполнения программы



A terminal window with a dark background and light text. The title bar at the top shows the user 'yvdeljgadiljo' on a 'fedora' machine, in the directory '~/work/arch-pc/lab07'. The terminal contains two separate executions of a program. The first execution starts with the command './lab7-4', followed by prompts 'Введите x: 4' and 'Введите a: 5', and the user input '4'. The second execution also starts with './lab7-4', followed by prompts 'Введите x: 3' and 'Введите a: 2', and the user input '5'. Both executions end with a new prompt line.

```
yvdeljgadiljo@fedora:~/work/arch-pc/lab07
[yvdeljgadiljo@fedora lab07]$ ./lab7-4
Введите x: 4
Введите a: 5
4
[yvdeljgadiljo@fedora lab07]$ ./lab7-4
Введите x: 3
Введите a: 2
5
[yvdeljgadiljo@fedora lab07]$
```

Рис. 4.4:

5 Выводы

Я изучила команды условного и безусловного переходов и научилась писать программы с использованием этих переходов.

6 Список литературы

- GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
- GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
- Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
- NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
- 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>.
- Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
- The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
- Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
- Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
- Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
- Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
- Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.

- Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
- 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.
- Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
- Таненбаум Э., Бос Х. Современные операционн