

Отчёт по лабораторной работе №7

Дисциплина: архитектура компьютера

Барбакова Алиса Саяновна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
4.1	Реализация переходов в NASM	8
4.2	Изучение структуры файла листинга	15
4.3	Задания для самостоятельной работы	17
5	Выводы	24
	Список литературы	25

Список иллюстраций

4.1	Создание каталога и файла для программы	8
4.2	Создание копии файла	9
4.3	Сохранение программы	10
4.4	Запуск программы	10
4.5	Изменение программы	11
4.6	Запуск измененной программы	11
4.7	Изменение программы	12
4.8	Проверка изменений	13
4.9	Сохранение новой программы	14
4.10	Проверка программы из листинга	14
4.11	Проверка файла листинга	15
4.12	Удаление операнда из программы	16
4.13	Просмотр ошибки	17
4.14	Первая программа самостоятельной работы	18
4.15	Проверка работы первой программы	20
4.16	Вторая программа самостоятельной работы	21
4.17	Проверка работы второй программы	23

1 Цель работы

Цель данной лабораторной работы - изучение команд условного и безусловного переходов, приобретение навыков написания программ с использованием переходов, знакомство с назначением и структурой файла листинга.

2 Задание

1. Символьные и численные данные в NASM
2. Выполнение арифметических операций в NASM
3. Выполнение заданий 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.

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.

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 пред-

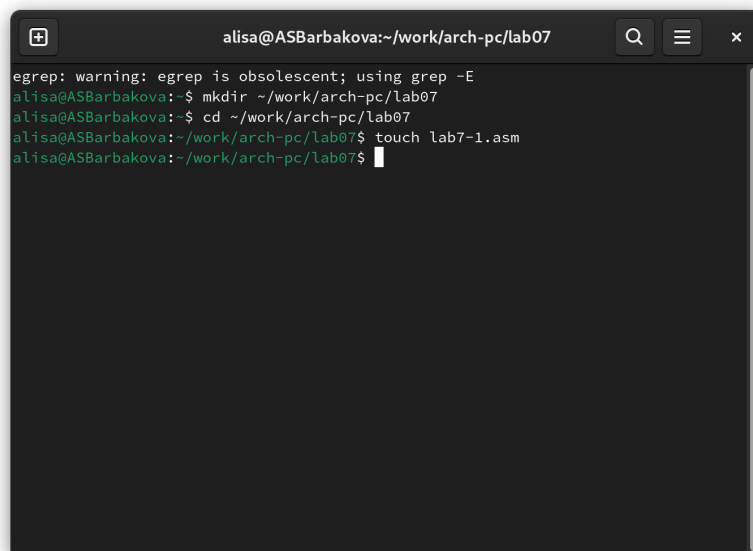
ставлении); CD80 — это инструкция на машинном языке, вызывающая прерывание ядра);

- 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).

4 Выполнение лабораторной работы

4.1 Реализация переходов в NASM

Создаю каталог для программ лабораторной работы №7 и файл lab7-1.asm (рис. -fig. 4.1).



```
alisa@ASBarbakova:~/work/arch-pc/lab07
egrep: warning: egrep is obsolescent; using grep -E
alisa@ASBarbakova:~$ mkdir ~/work/arch-pc/lab07
alisa@ASBarbakova:~$ cd ~/work/arch-pc/lab07
alisa@ASBarbakova:~/work/arch-pc/lab07$ touch lab7-1.asm
alisa@ASBarbakova:~/work/arch-pc/lab07$
```

Рис. 4.1: Создание каталога и файла для программы

Копирую в текущий каталог файл in_out.asm (рис. 4.2).

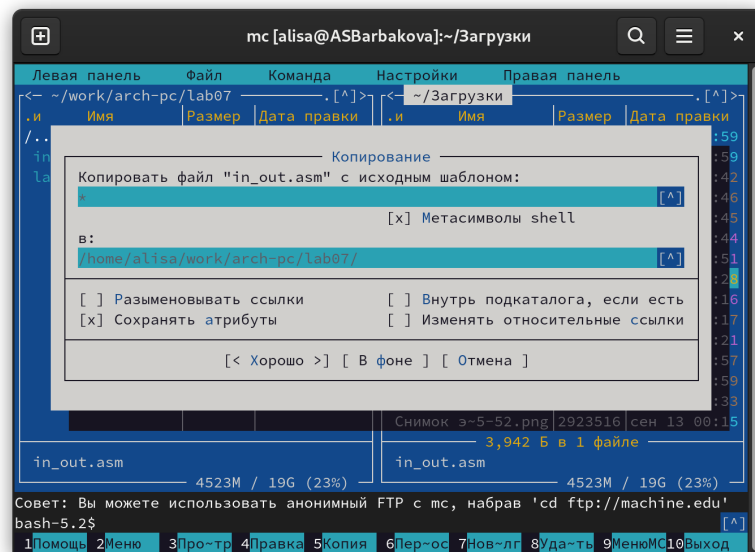
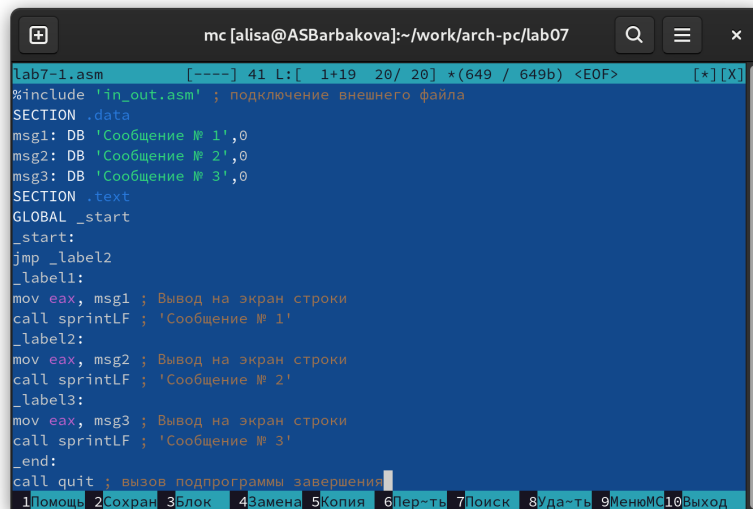


Рис. 4.2: Создание копии файла

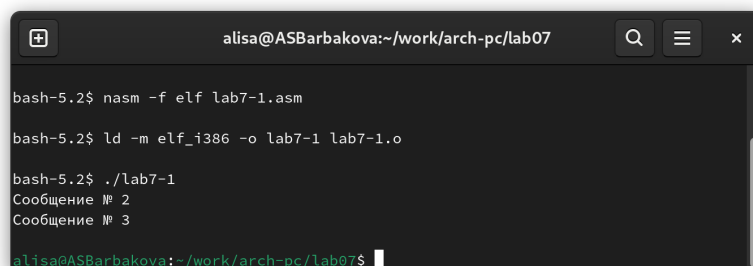
Копирую код из листинга в файл будущей программы. (рис. -fig. 4.3).



```
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
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Уда-ть 9МенюMC10Выход
```

Рис. 4.3: Сохранение программы

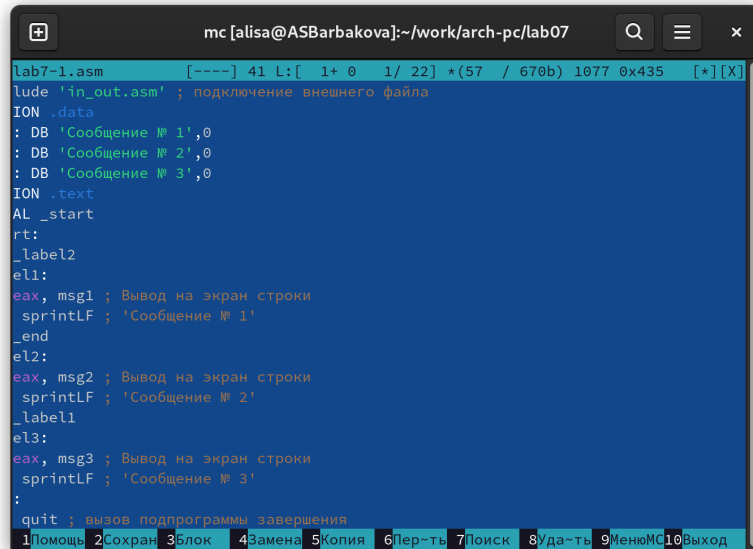
Запускаю программу. Вижу, что безусловный переход действительно изменяет порядок выполнения инструкций (рис. -fig. 4.4).



```
alisa@ASBarbakova:~/work/arch-pc/lab07
bash-5.2$ nasm -f elf lab7-1.asm
bash-5.2$ ld -m elf_i386 -o lab7-1 lab7-1.o
bash-5.2$ ./lab7-1
Сообщение № 2
Сообщение № 3
alisa@ASBarbakova:~/work/arch-pc/lab07$
```

Рис. 4.4: Запуск программы

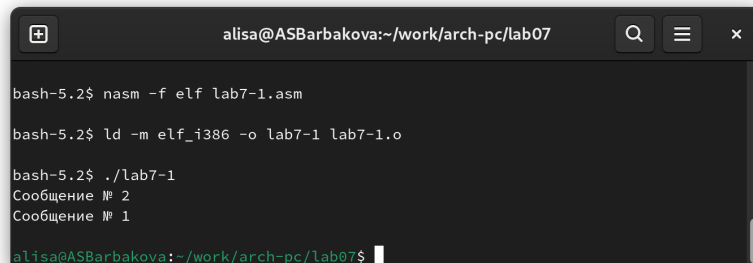
Изменяю программу таким образом, чтобы поменялся порядок выполнения функций (рис. -fig. 4.5).



```
lab7-1.asm [----] 41 L: [ 1+ 0 1/ 22] *(57 / 670b) 1077 0x435 [*][X]
lude 'in_out.asm' ; подключение внешнего файла
ION .data
: DB 'Сообщение № 1',0
: DB 'Сообщение № 2',0
: DB 'Сообщение № 3',0
ION .text
AL _start
rt:
_label2
el1:
eax, msg1 ; Вывод на экран строки
sprintf ; 'Сообщение № 1'
_end
el2:
eax, msg2 ; Вывод на экран строки
sprintf ; 'Сообщение № 2'
_label1
el3:
eax, msg3 ; Вывод на экран строки
sprintf ; 'Сообщение № 3'
:
quit ; вызов подпрограммы завершения
```

Рис. 4.5: Изменение программы

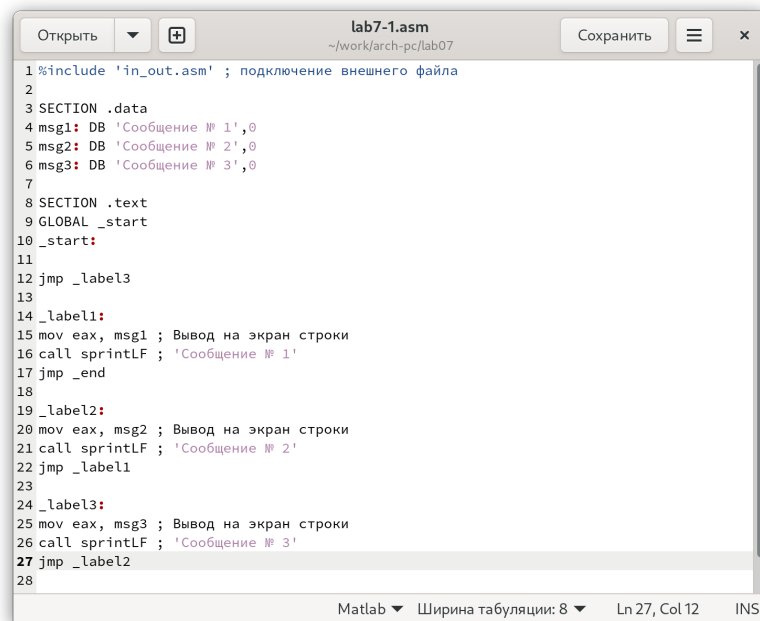
Запускаю программу и проверяю, что примененные изменения верны (рис. -fig. 4.6).



```
alisa@ASBarbakova:~/work/arch-pc/lab07
bash-5.2$ nasm -f elf lab7-1.asm
bash-5.2$ ld -m elf_i386 -o lab7-1 lab7-1.o
bash-5.2$ ./lab7-1
Сообщение № 2
Сообщение № 1
alisa@ASBarbakova:~/work/arch-pc/lab07$
```

Рис. 4.6: Запуск измененной программы

Далее я изменяю текст программы так, чтобы все три сообщения вы-
велись в обратном порядке (рис. -fig. 4.7).

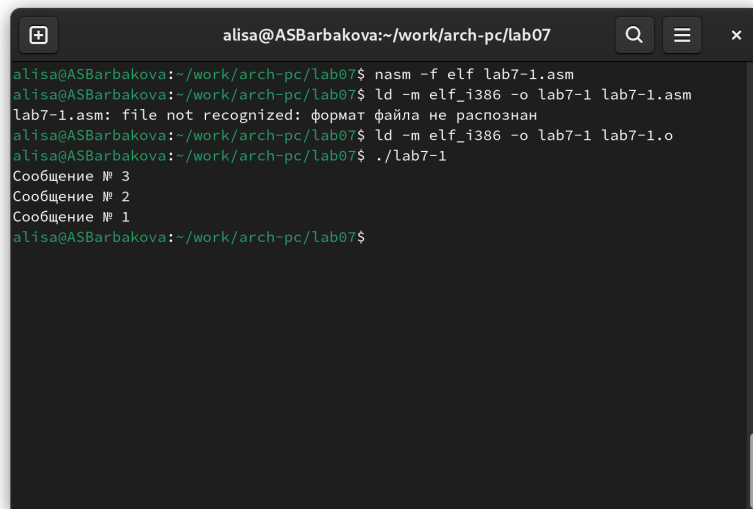


```
1 %include 'in_out.asm' ; подключение внешнего файла
2
3 SECTION .data
4 msg1: DB 'Сообщение № 1',0
5 msg2: DB 'Сообщение № 2',0
6 msg3: DB 'Сообщение № 3',0
7
8 SECTION .text
9 GLOBAL _start
10 _start:
11
12 jmp _label3
13
14 _label1:
15 mov eax, msg1 ; Вывод на экран строки
16 call sprintf ; 'Сообщение № 1'
17 jmp _end
18
19 _label2:
20 mov eax, msg2 ; Вывод на экран строки
21 call sprintf ; 'Сообщение № 2'
22 jmp _label1
23
24 _label3:
25 mov eax, msg3 ; Вывод на экран строки
26 call sprintf ; 'Сообщение № 3'
27 jmp _label2
28
```

Matlab Ширина табуляции: 8 Ln 27, Col 12 INS

Рис. 4.7: Изменение программы

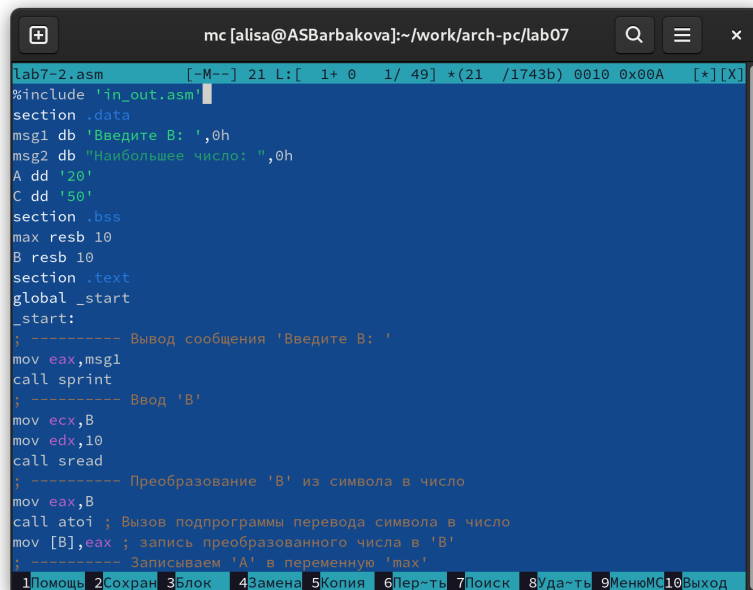
Запускаю программу, сообщения выводятся в нужном мне порядке (рис. -fig. 4.8).



```
alisa@ASBarbakova:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
alisa@ASBarbakova:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.asm
lab7-1.asm: file not recognized: формат файла не распознан
alisa@ASBarbakova:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
alisa@ASBarbakova:~/work/arch-pc/lab07$ ./lab7-1
Сообщение № 3
Сообщение № 2
Сообщение № 1
alisa@ASBarbakova:~/work/arch-pc/lab07$
```

Рис. 4.8: Проверка изменений

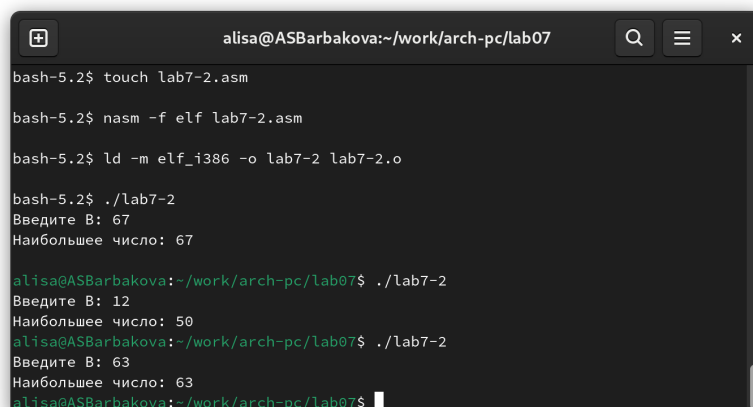
Создаю новый рабочий файл lab7-2.asm и вставляю в него код из следующего листинга (рис. -fig. 4.9).



```
lab7-2.asm [-M--] 21 L: [ 1+ 0 1/ 49] *(21 /1743b) 0010 0x00A [*] [X]
#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'
1Помощь 2Сохран 3Блок 4Замена 5Копия 6Пер-ть 7Поиск 8Уда-ть 9Меню10Выход
```

Рис. 4.9: Сохранение новой программы

Программа выводит значение переменной с максимальным значением, проверяю работу программы с разными входными данными (рис. -fig. 4.10).



```
alisa@ASBarbakova:~/work/arch-pc/lab07
bash-5.2$ touch lab7-2.asm
bash-5.2$ nasm -f elf lab7-2.asm
bash-5.2$ ld -m elf_i386 -o lab7-2 lab7-2.o
bash-5.2$ ./lab7-2
Введите B: 67
Наибольшее число: 67
alisa@ASBarbakova:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 12
Наибольшее число: 50
alisa@ASBarbakova:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 63
Наибольшее число: 63
alisa@ASBarbakova:~/work/arch-pc/lab07$
```

Рис. 4.10: Проверка программы из листинга

4.2 Изучение структуры файла листинга

Создаю файл листинга с помощью команды `nasm -f elf -l lab7-2.lst lab7-2.asm` и открываю его с помощью текстового редактора `mcedit` (рис. - fig. 4.11).

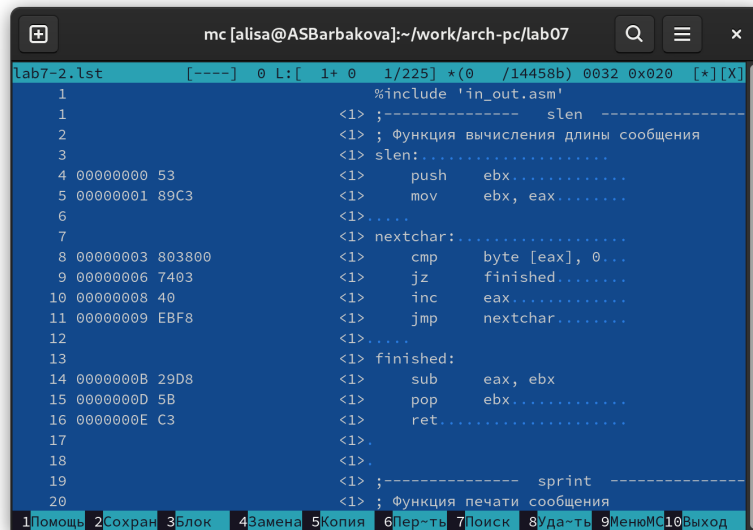


Рис. 4.11: Проверка файла листинга

1. Строка 4: 00000000 53 <1> push ebx.....

Адрес: 00000000 — это адрес в памяти, где находится инструкция.
Код операции: 53 — это байт-код ассемблерной инструкции `push ebx`. Эта инструкция помещает значение регистра `ebx` в стек.

2. Строка 5: 00000001 89C3 <1> mov ebx, eax.....

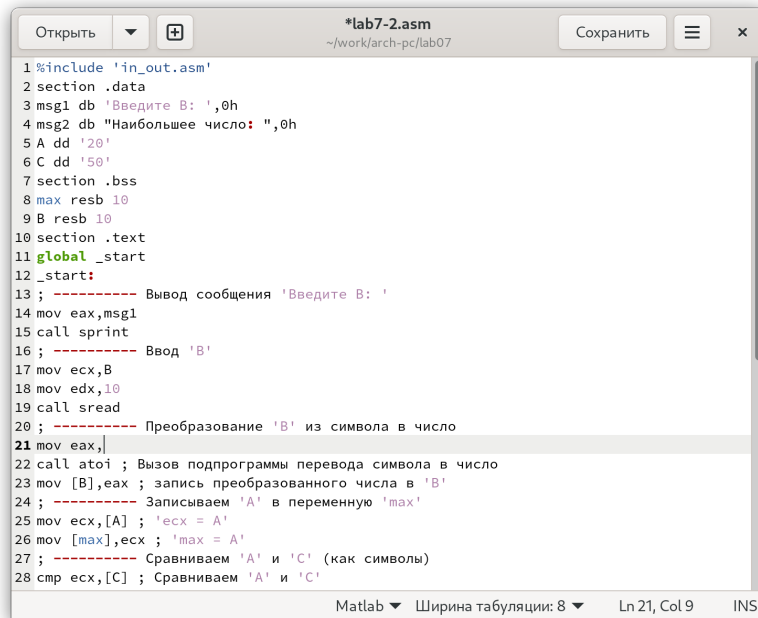
Адрес: 00000001 — адрес в памяти, где эта инструкция будет выполнена. Код операции: 89C3 — это байт-код для инструкции

mov ebx, eax. Инструкция копирует содержимое регистра eax в регистр ebx.

3. Строка 8: 00000003 803800 <1> cmp byte [eax], 0...

Адрес: 00000003 — адрес этой инструкции в памяти. Код операции: 803800 — это собственный код для инструкции cmp byte [eax], 0. Эта инструкция сравнивает байт, на который указывает eax, с нулем. Эти три строки из листинга вычисляет длину строки. Первая строка сохраняет состояние регистра, вторая сохраняет начало строки, а третья проверяет каждый символ строки на равенство нулю, что позволяет завершить вычисление длины строки.

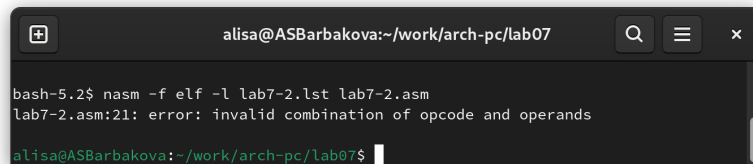
Удаляю один операнд из случайной инструкции, чтобы проверить поведение файла листинга в дальнейшем (рис. -fig. 4.12).



```
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,|
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'
```

Рис. 4.12: Удаление операнда из программы

Новый файл листинга у меня не создаётся, выходит ошибка при попытке трансляции файла. Никакие выходные файлы не создаются. (рис. -fig. 4.13).

A screenshot of a terminal window with a dark background. The window title is 'alisa@ASBarbakova:~/work/arch-pc/lab07'. The terminal shows the command 'bash-5.2\$ nasm -f elf -l lab7-2.lst lab7-2.asm' and the resulting error message 'lab7-2.asm:21: error: invalid combination of opcode and operands'. The prompt 'alisa@ASBarbakova:~/work/arch-pc/lab07\$' is visible at the bottom.

```
alisa@ASBarbakova:~/work/arch-pc/lab07
bash-5.2$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:21: error: invalid combination of opcode and operands
alisa@ASBarbakova:~/work/arch-pc/lab07$
```

Рис. 4.13: Просмотр ошибки

4.3 Задания для самостоятельной работы

Для выполнения самостоятельной работы использую свой вариант - восьмой - из лабораторной работы №6. Копирую файл lab7-2.asm и изменяю программу так, чтобы она выводила переменную с наименьшим значением (рис. -fig. 4.14).

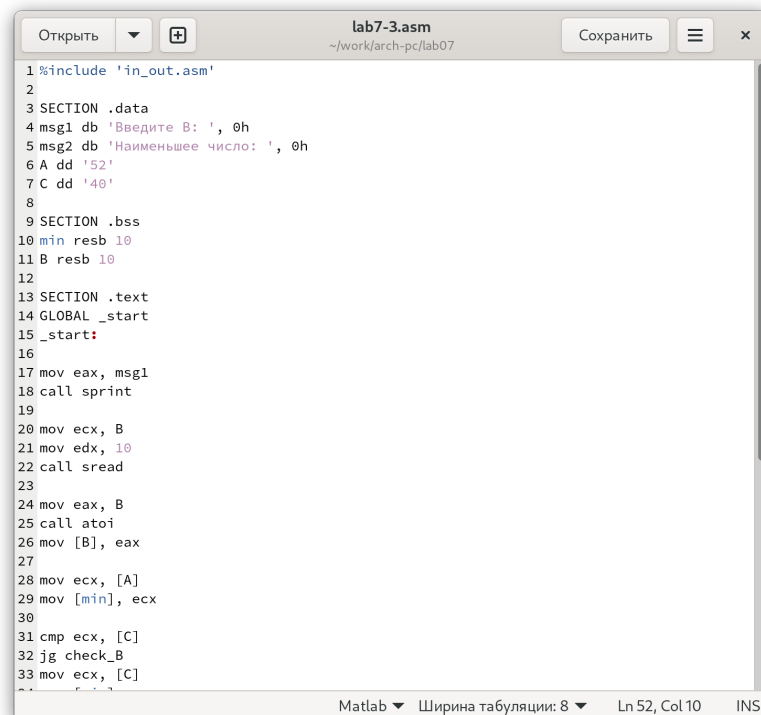


Рис. 4.14: Первая программа самостоятельной работы

Код первой программы:

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

```
SECTION .data
```

```
msg1 db 'Введите B: ', 0h
```

```
msg2 db 'Наименьшее число: ', 0h
```

```
A dd '52'
```

```
C dd '40'
```

```
SECTION .bss
```

```
min resb 10
```

```
B resb 10
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov eax, msg1
```

```
call sprint
```

```
mov ecx, B
```

```
mov edx, 10
```

```
call sread
```

```
mov eax, B
```

```
call atoi
```

```
mov [B], eax
```

```
mov ecx, [A]
```

```
mov [min], ecx
```

```
cmp ecx, [C]
```

```
jg check_B
```

```
mov ecx, [C]
```

```
mov [min], ecx
```

```
check_B:
```

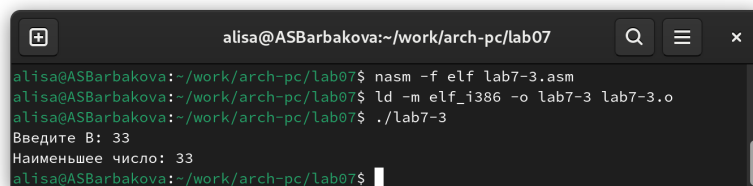
```
mov eax, min
```

```
call atoi
mov [min], eax
```

```
mov ecx, [min]
cmp ecx, [B]
jb fin
mov ecx, [B]
mov [min], ecx
```

```
fin:
mov eax, msg2
call sprint
mov eax, [min]
call iprintLF
call quit
```

Проверяю корректность написания первой программы (рис. -fig. 4.15).

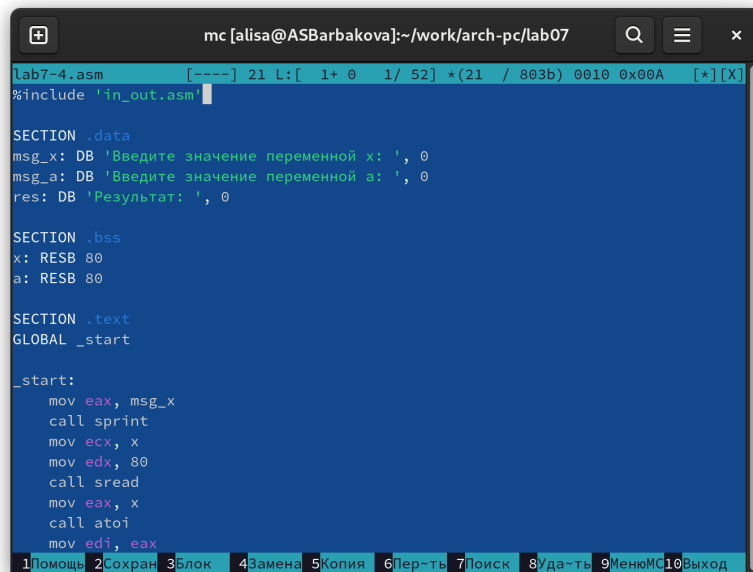


```
alisa@ASBarbakova:~/work/arch-pc/lab07
alisa@ASBarbakova:~/work/arch-pc/lab07$ nasm -f elf lab7-3.asm
alisa@ASBarbakova:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.o
alisa@ASBarbakova:~/work/arch-pc/lab07$ ./lab7-3
Введите B: 33
Наименьшее число: 33
alisa@ASBarbakova:~/work/arch-pc/lab07$
```

Рис. 4.15: Проверка работы первой программы

Пишу программу, которая будет вычислять значение заданной функции согласно 8-му варианту для введенных с клавиатуры переменных

а и х (рис. -fig. 4.16).



```
mc [alisa@ASBarbakova]:~/work/arch-pc/lab07
lab7-4.asm [----] 21 L: [ 1+ 0 1/ 52] *(21 / 803b) 0010 0x00A [*] [X]
#include 'in_out.asm'

SECTION .data
msg_x: DB 'Введите значение переменной x: ', 0
msg_a: DB 'Введите значение переменной a: ', 0
res: DB 'Результат: ', 0

SECTION .bss
x: RESB 80
a: RESB 80

SECTION .text
GLOBAL _start
_start:
    mov eax, msg_x
    call sprint
    mov ecx, x
    mov edx, 80
    call sread
    mov eax, x
    call atoi
    mov edi, eax
```

Рис. 4.16: Вторая программа самостоятельной работы

Код второй программы:

```
%include 'in_out.asm'

SECTION .data
msg_x: DB 'Введите значение переменной x: ', 0
msg_a: DB 'Введите значение переменной a: ', 0
res: DB 'Результат: ', 0

SECTION .bss
x: RESB 80
a: RESB 80
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
    mov eax, msg_x
```

```
    call sprint
```

```
    mov ecx, x
```

```
    mov edx, 80
```

```
    call sread
```

```
    mov eax, x
```

```
    call atoi
```

```
    mov edi, eax
```

```
    mov eax, msg_a
```

```
    call sprint
```

```
    mov ecx, a
```

```
    mov edx, 80
```

```
    call sread
```

```
    mov eax, a
```

```
    call atoi
```

```
    mov esi, eax
```

```
    cmp esi, 3
```

```
    jl calculate_three_a
```

```
    mov eax, edi
```

```

    inc eax
    jmp print_result
calculate_three_a:
    mov eax, esi
    imul eax, 3

print_result:
    mov edi, eax
    mov eax, res
    call sprint
    mov eax, edi
    call iprintLF
    call quit

```

Транслирую и компоную файл, запускаю и проверяю работу программы для различных значений а и х в соответствии с моим вариантом (рис. -fig. 4.17).

```

alisa@ASBarbakova:~/work/arch-pc/lab07
bash-5.2$ touch ~/work/arch-pc/lab07/lab7-4.asm
bash-5.2$ nasm -f elf lab7-4.asm
bash-5.2$ ld -m elf_i386 lab7-4 lab7-4.o
ld: невозможно найти lab7-4: Нет такого файла или каталога
alisa@ASBarbakova:~/work/arch-pc/lab07$ ld -m elf_i386 lab7-4 lab7-4.o
ld: невозможно найти lab7-4: Нет такого файла или каталога
alisa@ASBarbakova:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-4 lab7-4.o
alisa@ASBarbakova:~/work/arch-pc/lab07$ ./lab7-4
Введите значение переменной x: 1
Введите значение переменной a: 4
Результат: 2
alisa@ASBarbakova:~/work/arch-pc/lab07$ ./lab7-4
Введите значение переменной x: 1
Введите значение переменной a: 2
Результат: 6
alisa@ASBarbakova:~/work/arch-pc/lab07$

```

Рис. 4.17: Проверка работы второй программы

5 Выводы

При выполнении лабораторной работы я изучила команды условных и безусловных переходов, а также приобрела навыки написания программ с использованием переходов, познакомилась с назначением и структурой файлов листинга.

Список литературы

1. Лабораторная работа №7