

Отчёт по лабораторной работе №7

Дисциплина: архитектура компьютера

Черкашина Ангелина Максимовна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Реализация переходов в NASM	8
4.2	Изучение структуры файла листинга	13
4.3	Выполнение заданий для самостоятельной работы	15
5	Выводы	22
6	Список литературы	23

Список иллюстраций

4.1	Создание каталога и файла	8
4.2	Создание копии файла	8
4.3	Редактирование созданного файла	9
4.4	Запуск исполняемого файла	9
4.5	Изменение текста программы	10
4.6	Запуск исполняемого файла	10
4.7	Изменение текста программы	11
4.8	Запуск исполняемого файла	11
4.9	Создание файла	11
4.10	Редактирование нового файла	12
4.11	Проверка работы программы	12
4.12	Повторная проверка работы программы	13
4.13	Создание файла листинга	13
4.14	Изучение файла листинга	13
4.15	Выбранные строки файла	14
4.16	Удаление выделенного операнда из кода	14
4.17	Трансляция с получением файла листинга	14
4.18	Создание файла	15
4.19	Написание программы	15
4.20	Запуск исполняемого файла и проверка его работы	16
4.21	Создание файла	17
4.22	Написание программы	18
4.23	Запуск исполняемого файла и проверка его работы	19

Список таблиц

1 Цель работы

Целью данной лабораторной работы является 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 файла листинга
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.

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

Создаю каталог для программ лабораторной работы № 7, перехожу в него и создаю файл lab7-1.asm с помощью команды touch (рис. 4.1).

```
amcherkashina@dk8n70 ~ $ mkdir ~/work/arch-pc/lab07
amcherkashina@dk8n70 ~ $ cd ~/work/arch-pc/lab07
amcherkashina@dk8n70 ~/work/arch-pc/lab07 $ touch lab7-1.asm
amcherkashina@dk8n70 ~/work/arch-pc/lab07 $
```

Рис. 4.1: Создание каталога и файла

С помощью команды cp копирую в текущий каталог файл in_out.asm, т.к. он будет использоваться в других программах (рис. 4.2).

```
amcherkashina@dk8n64 ~/work/arch-pc/lab07 $ cp ~/Загрузки/in_out.asm in_out.asm
amcherkashina@dk8n64 ~/work/arch-pc/lab07 $
```

Рис. 4.2: Создание копии файла

Открываю файл lab7-1.asm для редактирования и ввожу в него текст программы с использованием инструкции jmp в соответствии с листингом 7.1 (рис. 4.3).

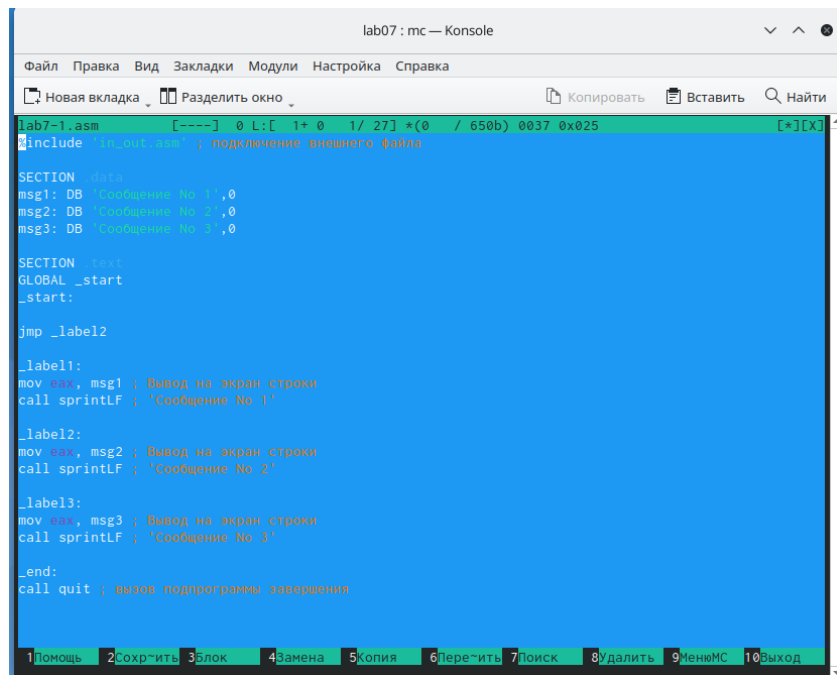


Рис. 4.3: Редактирование созданного файла

Создаю исполняемый файл и запускаю его (рис. 4.4).

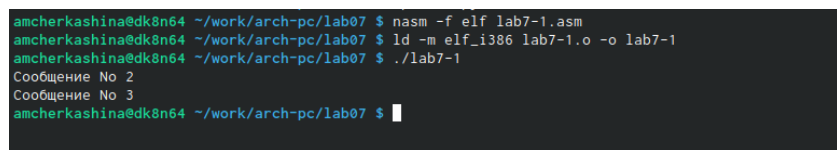
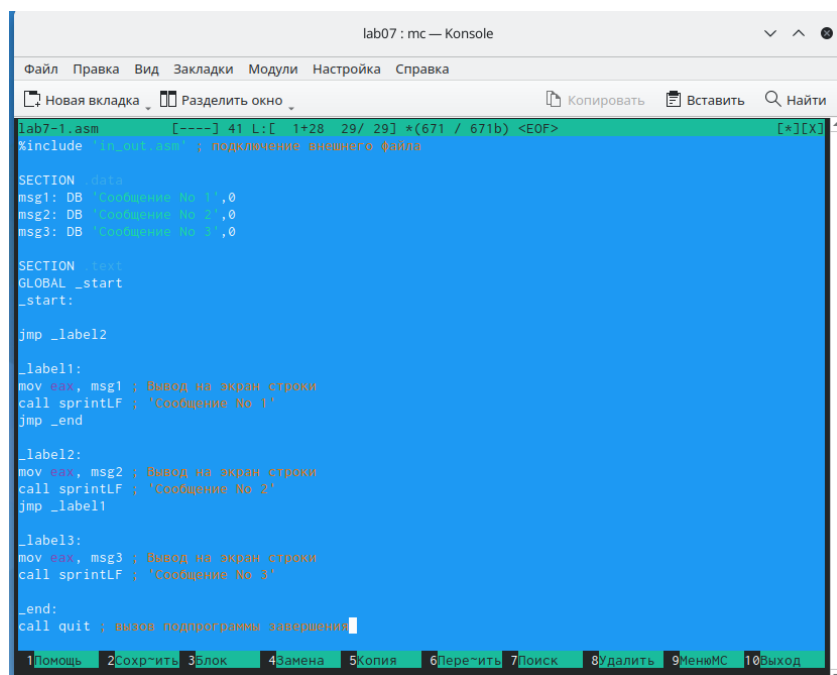


Рис. 4.4: Запуск исполняемого файла

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

Изменяю программу таким образом, чтобы она выводила сначала ‘Сообщение № 2’, потом ‘Сообщение № 1’ и завершала работу. Для этого изменяю текст программы в соответствии с листингом 7.2 (рис. 4.5).



```
lab7-1.asm [---] 41 L: [ 1+28 29/ 29] *(671 / 671b) <EOF> [*][X]
#include "lab7-1.inc" ; подключение внешнего файла

SECTION .data
msg1: DB "Сообщение No 1",0
msg2: DB "Сообщение No 2",0
msg3: DB "Сообщение No 3",0

SECTION .text
GLOBAL _start
_start:

jmp _label2

_label1:
mov eax, msg1 ; Вывод на экран строки
call sprintf ; "Сообщение No 1"
jmp _end

_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; "Сообщение No 2"
jmp _label1

_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; "Сообщение No 3"

_end:
call quit ; вызов подпрограммы завершения
```

Рис. 4.5: Изменение текста программы

Создаю исполняемый файл и проверяю его работу (рис. 4.6).



```
amcherkashina@dk8n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
amcherkashina@dk8n64 ~/work/arch-pc/lab07 $ ld -m elf_i386 lab7-1.o -o lab7-1
amcherkashina@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-1
Сообщение No 2
Сообщение No 1
amcherkashina@dk8n64 ~/work/arch-pc/lab07 $
```

Рис. 4.6: Запуск исполняемого файла

Далее изменяю текст программы так, чтобы она выводила сначала ‘Сообщение № 3’, потом ‘Сообщение № 2’, а затем ‘Сообщение № 1’ и завершала работу. Для этого добавляю jmp _label3 в начале программы и jmp _label2 в конце метки _label3 (рис. 4.7).

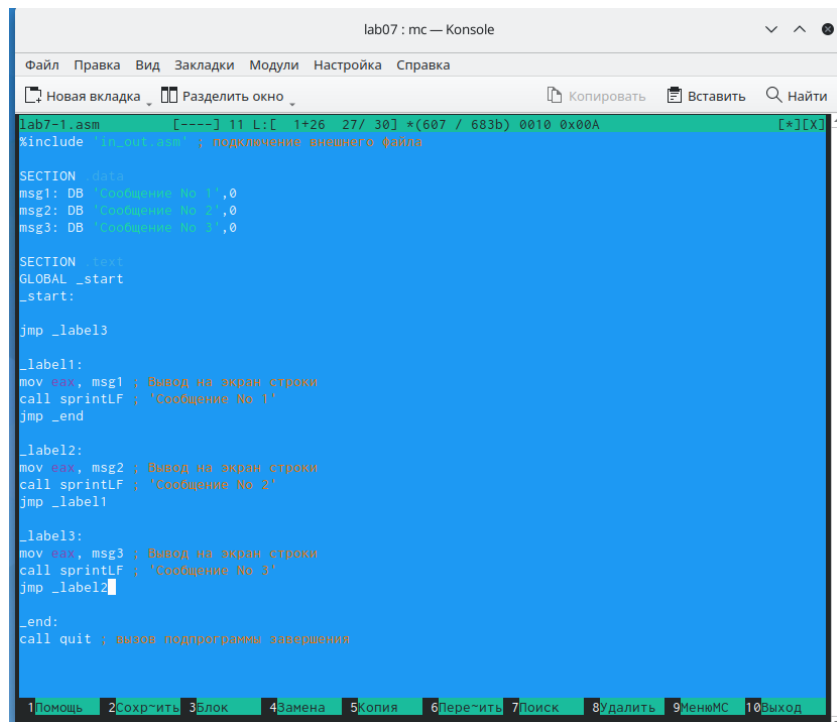


Рис. 4.7: Изменение текста программы

Создаю исполняемый файл и проверяю его работу (рис. 4.8).

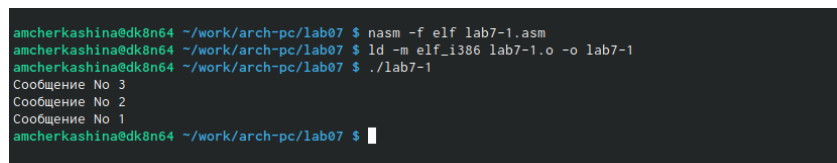


Рис. 4.8: Запуск исполняемого файла

Создаю файл lab7-2.asm в каталоге ~/work/arch-pc/lab07 (рис. 4.9).

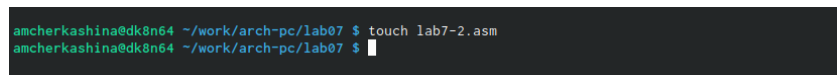
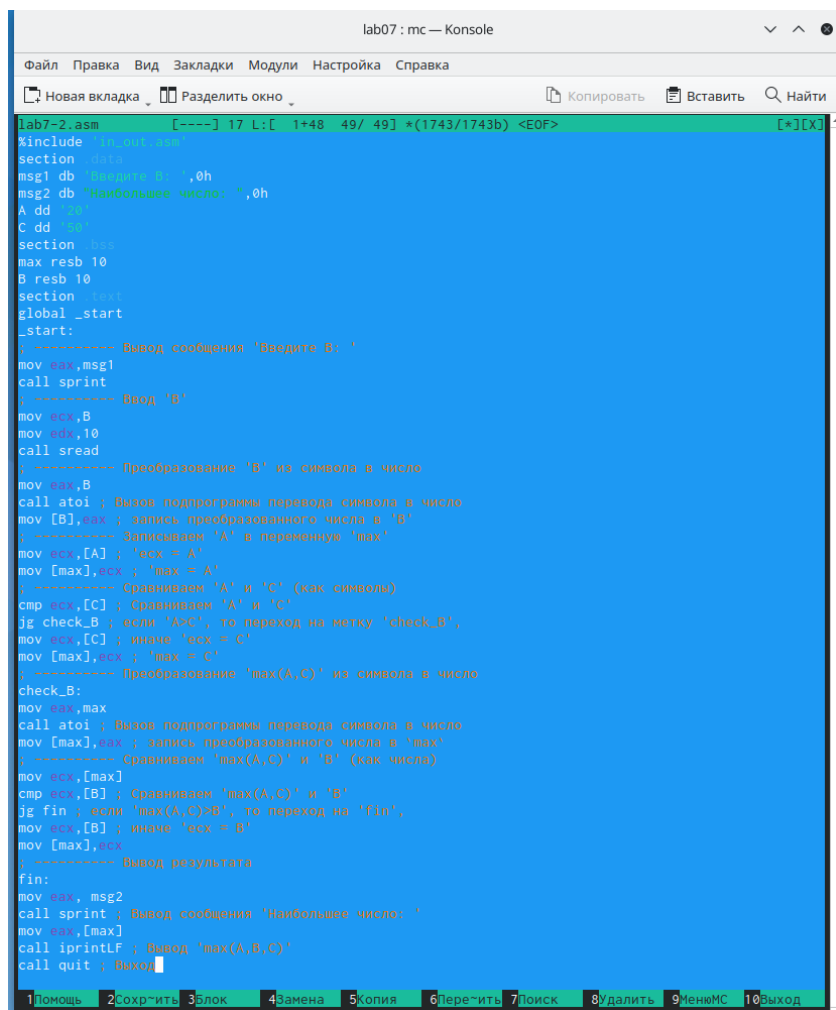


Рис. 4.9: Создание файла

Открываю файл lab7-2.asm для редактирования и ввожу в него текст программы, которая определяет и выводит на экран наибольшую из 3 целочисленных

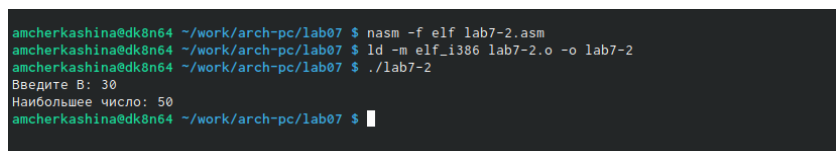
переменных: A, B и C (в соответствии с листингом 7.3) (рис. 4.10).



```
lab7-2.asm [----] 17 L: [ 1+48 49/ 49] *(1743/1743b) <EOF> [*][X]
%include "lab7-1.asm"
section .data
msg1 db "Введите B: ",0h
msg2 db "Наибольшее число: ",0h
A dd 10
C dd 50
section .code
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'
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
```

Рис. 4.10: Редактирование нового файла

Создаю исполняемый файл, запуская его, вводя сначала одно значение B (рис. 4.11), а затем другое (рис. 4.12).



```
amcherkashina@dk8n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
amcherkashina@dk8n64 ~/work/arch-pc/lab07 $ ld -m elf_i386 lab7-2.o -o lab7-2
amcherkashina@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-2
Введите B: 30
Наибольшее число: 50
amcherkashina@dk8n64 ~/work/arch-pc/lab07 $
```

Рис. 4.11: Проверка работы программы

```

amcherkashina@dk8n64 ~/work/arch-pc/lab07 $ ./lab7-2
Введите В: 60
Наибольшее число: 60
amcherkashina@dk8n64 ~/work/arch-pc/lab07 $

```

Рис. 4.12: Повторная проверка работы программы

4.2 Изучение структуры файла листинга

Создаю файл листинга для программы из файла lab7-2.asm (рис. 4.13).

```

amcherkashina@dk8n64 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
amcherkashina@dk8n64 ~/work/arch-pc/lab07 $ ls
in_out.asm lab7-1 lab7-1.asm lab7-1.o lab7-2 lab7-2.asm lab7-2.lst lab7-2.o
amcherkashina@dk8n64 ~/work/arch-pc/lab07 $

```

Рис. 4.13: Создание файла листинга

Открываю файл листинга lab7-2.lst с помощью текстового редактора gedit и внимательно изучаю его формат и содержимое (рис. 4.14).

```

1 1 %include 'in_out.asm'
2 2 ;----- slen -----
3 3 ; Функция вычисления длины сообщения
4 4 slen:
5 5 00000000 53 <1> push ebx
6 6 00000001 89C3 <1> mov ebx, eax
7 7 <1>
8 8 <1> nextchar:
9 9 00000003 803800 <1> cmp byte [eax], 0
10 10 00000006 7403 <1> jz finished
11 11 00000008 40 <1> inc eax
12 12 00000009 EBF8 <1> jmp nextchar
13 13 <1>
14 14 <1> finished:
15 15 0000000B 29D8 <1> sub eax, ebx
16 16 0000000D 5B <1> pop ebx
17 17 0000000E C3 <1> ret
18 18 <1>
19 19 <1>
20 20 ;----- sprint -----
21 21 ; Функция печати сообщения
22 22 ; входные данные: mov eax, <message>
23 23 sprint:
24 24 0000000F 52 <1> push edx
25 25 00000010 51 <1> push ecx
26 26 00000011 53 <1> push ebx
27 27 00000012 50 <1> push eax
28 28 00000013 E8E8FFFFFF <1> call slen
29 29 <1>
30 30 00000018 89C2 <1> mov edx, eax
31 31 0000001A 58 <1> pop eax
32 32 <1>
33 33 0000001B 89C1 <1> mov ecx, eax
34 34 0000001D BB01000000 <1> mov ebx, 1

```

Рис. 4.14: Изучение файла листинга

В трех представленных строках (рис. 4.15) содержатся следующие данные:

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

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

Рис. 4.15: Выбранные строки файла

Открываю файл с программой lab7-2.asm и в выбранной мной инструкции с двумя операндами удаляю выделенный операнд (рис. 4.16).

```
; ----- 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'
```

Рис. 4.16: Удаление выделенного операнда из кода

Выполняю трансляцию с получением файла листинга (рис. 4.17).

```
amcherkashina@dk8n56 ~/work/arch-pc/lab07 $ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:28: error: invalid combination of opcode and operands
amcherkashina@dk8n56 ~/work/arch-pc/lab07 $
```

Рис. 4.17: Трансляция с получением файла листинга

На выходе я не получаю ни одного файла из-за ошибки: инструкция `cmp` не может работать, имея только один операнд, из-за чего нарушается работа кода.

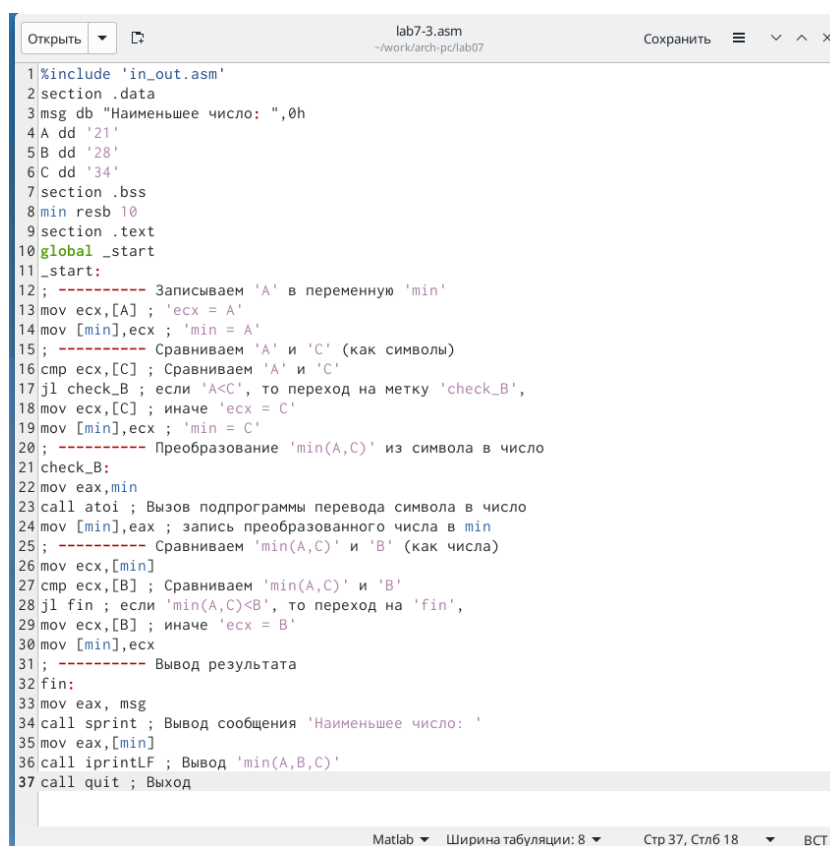
4.3 Выполнение заданий для самостоятельной работы

1. С помощью команды `touch` создаю файл `lab7-3.asm` для выполнения самостоятельного задания №1 (вариант 11) (рис. 4.18).

```
amcherkashina@dk8n64 ~/work/arch-pc/lab07 $ touch lab7-3.asm
amcherkashina@dk8n64 ~/work/arch-pc/lab07 $
```

Рис. 4.18: Создание файла

В созданном файле пишу программу нахождения наименьшей из 3 целочисленных переменных `a`, `b` и `c`. Значения переменных беру из таблицы 7.5 в соответствии с вариантом, полученным мной при выполнении лабораторной работы № 6. Мой вариант под номером 11, поэтому мои значения - 21, 28, 34 (рис. 4.19).



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

Рис. 4.19: Написание программы

Создаю исполняемый файл и проверяю его работу (рис. 4.20).

```
amcherkashina@dk8n80 ~/work/arch-pc/lab07 $ gedit lab7-3.asm
amcherkashina@dk8n80 ~/work/arch-pc/lab07 $ nasm -f elf lab7-3.asm
amcherkashina@dk8n80 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-3 lab7-3.o
amcherkashina@dk8n80 ~/work/arch-pc/lab07 $ ./lab7-3
Наименьшее число: 21
amcherkashina@dk8n80 ~/work/arch-pc/lab07 $
```

Рис. 4.20: Запуск исполняемого файла и проверка его работы

Программа работает корректно.

Листинг 4.1. Программа нахождения наименьшей целочисленной переменной

```
%include 'in_out.asm'

section .data
msg db "Наименьшее число: ",0h
A dd '21'
B dd '28'
C dd '34'

section .bss
min resb 10

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

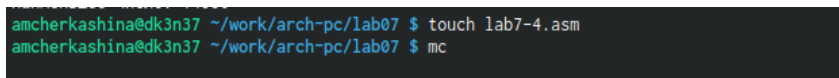


```

mov [min],ecx ; 'min = C'
; ----- Преобразование 'min(A,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(A,C)' и 'B' (как числа)
mov ecx,[min]
cmp ecx,[B] ; Сравниваем 'min(A,C)' и 'B'
jl fin ; если 'min(A,C) < B', то переход на 'fin',
mov ecx,[B] ; иначе 'ecx = B'
mov [min],ecx
; ----- Вывод результата
fin:
mov eax, msg
call sprintf ; Вывод сообщения 'Наименьшее число: '
mov eax,[min]
call iprintLF ; Вывод 'min(A,B,C)'
call quit ; Выход

```

2. С помощью команды touch создаю файл lab7-4.asm для выполнения самостоятельного задания №2 (вариант 11) (рис. 4.21).



```

amcherkashina@dk3n37 ~/work/arch-pc/lab07 $ touch lab7-4.asm
amcherkashina@dk3n37 ~/work/arch-pc/lab07 $ mc

```

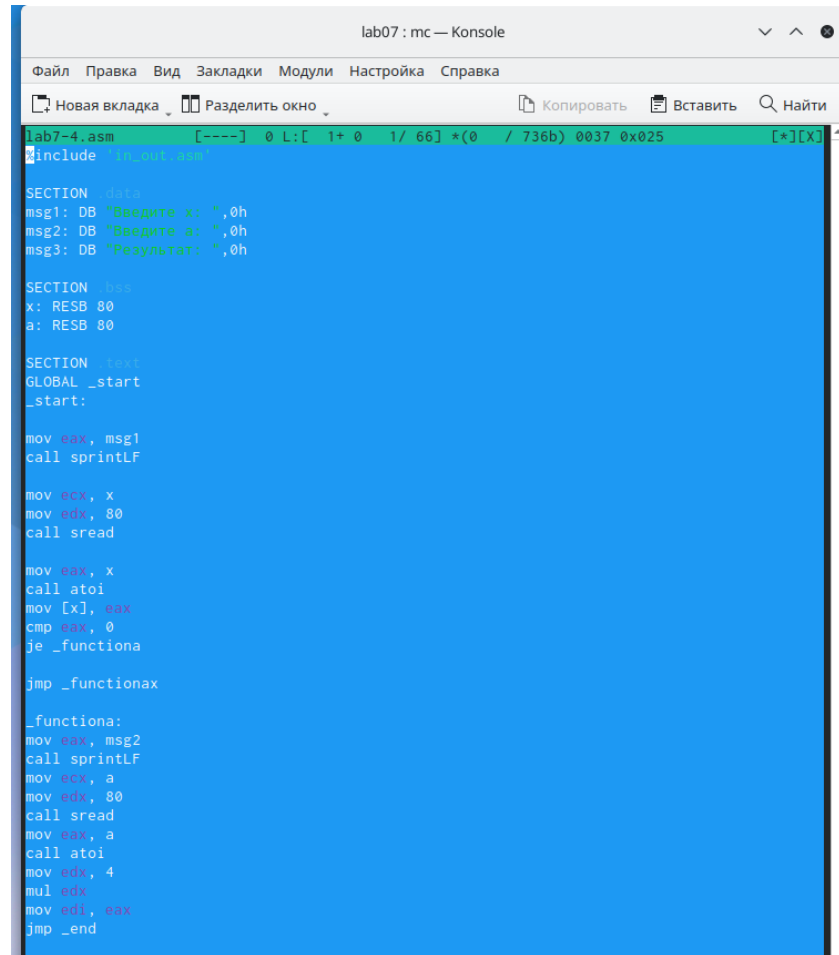
Рис. 4.21: Создание файла

Пишу программу, которая для введенных с клавиатуры значений x и a вычисляет значение заданной функции $f(x)$ и выводит результат вычислений (рис. 4.22).

Мой вариант 11, поэтому я пишу программу для следующей функции:

$4 \cdot a, x=0$

$4 \cdot a + x, x \neq 0$



```
lab07-4.asm [----] 0 L: [ 1+ 0 1/ 66] *(0 / 736b) 0037 0x025 [*][X]
#include "stdafx.h"

SECTION .data
msg1: DB "Введите x: ",0h
msg2: DB "Введите a: ",0h
msg3: DB "Результат: ",0h

SECTION .bss
x: RESB 80
a: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax, msg1
call sprintf

mov ecx, x
mov edx, 80
call sread

mov eax, x
call atoi
mov [x], eax
cmp eax, 0
je _functiona

jmp _functionax

_functiona:
mov eax, msg2
call sprintf
mov ecx, a
mov edx, 80
call sread
mov eax, a
call atoi
mov edx, 4
mul edx
mov edi, eax
jmp _end
```

Рис. 4.22: Написание программы

Создаю исполняемый файл и проверяю его работу для следующих значений x и a соответственно: (0;3) и (1;2) (рис. 4.23).

```
amcherkashina@dk8n68 ~/work/arch-pc/lab07 $ nasm -f elf lab7-4.asm
amcherkashina@dk8n68 ~/work/arch-pc/lab07 $ ld -m elf_i386 lab7-4.o -o lab7-4
amcherkashina@dk8n68 ~/work/arch-pc/lab07 $ ./lab7-4
Введите x:
0
Введите a:
3
Результат: 12
amcherkashina@dk8n68 ~/work/arch-pc/lab07 $ ./lab7-4
Введите x:
1
Введите a:
2
Результат: 9
amcherkashina@dk8n68 ~/work/arch-pc/lab07 $
```

Рис. 4.23: Запуск исполняемого файла и проверка его работы

Программа работает корректно.

Листинг 4.2. Программа вычисления значения функции $f(x)$

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

```
SECTION .data
```

```
msg1: DB "Введите x: ",0h
```

```
msg2: DB "Введите a: ",0h
```

```
msg3: DB "Результат: ",0h
```

```
SECTION .bss
```

```
x: RESB 80
```

```
a: RESB 80
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov eax, msg1
```

```
call sprintLF
```

```
mov ecx, x
mov edx, 80
call sread

mov eax, x
call atoi
mov [x], eax
cmp eax, 0
je _functiona

jmp _functionax
```

```
_functiona:
mov eax, msg2
call sprintLF
mov ecx, a
mov edx, 80
call sread
mov eax, a
call atoi
mov edx, 4
mul edx
mov edi, eax
jmp _end
```

```
_functionax:
mov eax, msg2
call sprintLF
```

```
mov ecx, a
mov edx, 80
call sread
mov eax, a
call atoi
mov edx, 4
mul edx
mov edx, [x]
add eax, edx
mov edi, eax
jmp _end
```

```
_end:
mov ecx, eax
mov eax, msg3
call sprint
mov eax, edi
call iprintLF
call quit
```

5 Выводы

При выполнении данной лабораторной работы я изучила команды условного и безусловного переходов, приобрела навыки написания программ с использованием переходов и ознакомилась с назначением и структурой файла листинга.

6 Список литературы

1. Архитектура ЭВМ