

# **Отчет по лабораторной работе №7**

**Дисциплина: архитектура компьютера**

**Закиров Нурислам Дамирович**

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
4.1	Реализация переходов в NASM . . . . .	7
4.2	Изучение структуры файлы листинга . . . . .	12
4.3	Задания для самостоятельной работы . . . . .	13
<b>5</b>	<b>Выводы</b>	<b>21</b>
<b>6</b>	<b>Список литературы</b>	<b>22</b>

## Список иллюстраций

4.1	Создание файлов для лабораторной работы . . . . .	7
4.2	Ввод текста программы из листинга 7.1 . . . . .	8
4.3	Запуск программного кода . . . . .	8
4.4	Изменение текста программы . . . . .	9
4.5	Создание исполняемого файла . . . . .	9
4.6	Изменение текста программы . . . . .	10
4.7	Вывод программы . . . . .	10
4.8	Создание файла . . . . .	11
4.9	Ввод текста программы из листинга 7.3 . . . . .	11
4.10	Проверка работы файла . . . . .	11
4.11	Создание файла листинга . . . . .	12
4.12	Изучение файла листинга . . . . .	12
4.13	Выбранные строки файла . . . . .	12
4.14	Удаление выделенного операнда из кода . . . . .	13
4.15	Получение файла листинга . . . . .	13
4.16	Написание программы . . . . .	14
4.17	Запуск файла и проверка его работы . . . . .	14
4.18	Написание программы . . . . .	18
4.19	Запуск файла и проверка его работы . . . . .	18
4.20	Запуск файла и проверка его работы . . . . .	18

# 1 Цель работы

Изучение команд условного и безусловного переходов. Освоение навыков написания программ с использованием переходов. Знакомство с назначением и структурой файла листинга.

## 2 Задание

1. Реализация переходов в NASM.
2. Изучение структуры файлы листинга.
3. Задания 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.

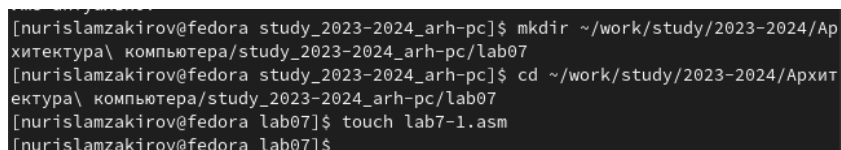
Безусловный переход выполняется инструкцией `jmp`. Инструкция `cmp` является одной из инструкций, которая позволяет сравнить операнды и выставляет флаги в зависимости от результата сравнения. Инструкция `cmp` является командой сравнения двух операндов и имеет такой же формат, как и команда вычитания.

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. (рис. 4.1).



```
[nurislamzakirov@fedora study_2023-2024_arh-pc]$ mkdir ~/work/study/2023-2024/Архитектура\ компьютера/study_2023-2024_arh-pc/lab07
[nurislamzakirov@fedora study_2023-2024_arh-pc]$ cd ~/work/study/2023-2024/Архитектура\ компьютера/study_2023-2024_arh-pc/lab07
[nurislamzakirov@fedora lab07]$ touch lab7-1.asm
[nurislamzakirov@fedora lab07]$
```

Рис. 4.1: Создание файлов для лабораторной работы

Ввожу в файл lab7-1.asm текст программы из листинга 7.1. (рис. 4.2).

```

mc [nurislamzakirov@fedora]:~/work/study/2023-2024/Архите...
lab7-1.asm [----] 0 L: [ 1+ 0 1/ 21] *(0 / 644b) 003
%include 'in_out.asm' ; 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'
_label2:
mov eax, msg2 ; Вывод на экран строки
call sprintf ; 'Сообщение No 2'
_label3:
mov eax, msg3 ; Вывод на экран строки
call sprintf ; 'Сообщение No 3'
_end:
call quit ; вызов подпрограммы завершения

```

Рис. 4.2: Ввод текста программы из листинга 7.1

Создаю исполняемый файл и запускаю его. (рис. 4.3).

```

[nurislamzakirov@fedora lab07]$ nasm -f elf lab7-1.asm
[nurislamzakirov@fedora lab07]$ ld -m elf_i386 lab7-1.o -o lab7-1
[nurislamzakirov@fedora lab07]$ ./lab7-1
bash: ./lab7-1: Нет такого файла или каталога
[nurislamzakirov@fedora lab07]$ ld -m elf_i386 lab7-1.o -o lab7-1
[nurislamzakirov@fedora lab07]$ ./lab7-1
Сообщение No 2
Сообщение No 3

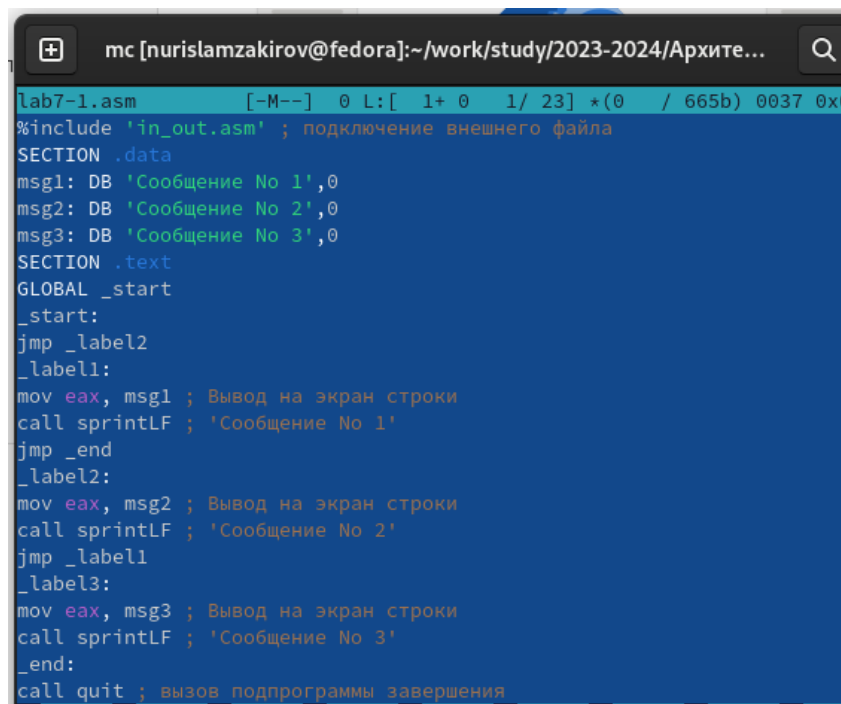
```

Рис. 4.3: Запуск программного кода

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

Измените программу таким образом, чтобы она выдавала сначала «Сообщение No 2», а затем «Сообщение No 1», чтобы завершить процесс. Для достижения этой цели изменяю код программы в соответствии с листингом 7.2. (рис. 4.4).

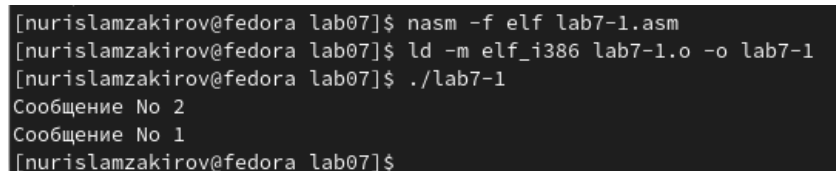




```
lab7-1.asm      [-M--]  0 L:[ 1+ 0  1/ 23] *(0  / 665b) 0037 0x
#include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg1: DB 'Сообщение No 1',0
msg2: DB 'Сообщение No 2',0
msg3: DB 'Сообщение No 3',0
SECTION .text
GLOBAL _start
_start:
jmp _label2
_label1:
mov eax, msg1 ; Вывод на экран 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. 4.4: Изменение текста программы

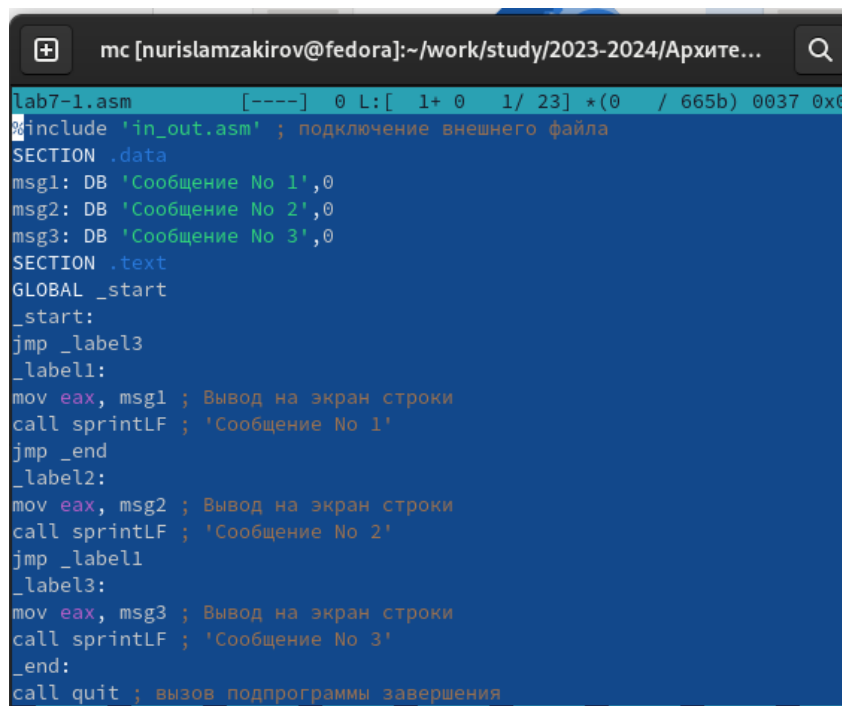
Создаю исполняемый файл и проверяю его работу. (рис. 4.5).



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

Рис. 4.5: Создание исполняемого файла

Следующим шагом я изменяю текст программы, добавив `jmp _label3` в начале программы, `jmp _label2` в конце метки `jmp _label3`, `jmp _label1` в конце метки `jmp _label2` и `jmp _end` в конце метки `jmp _label1`. (рис. 4.6).



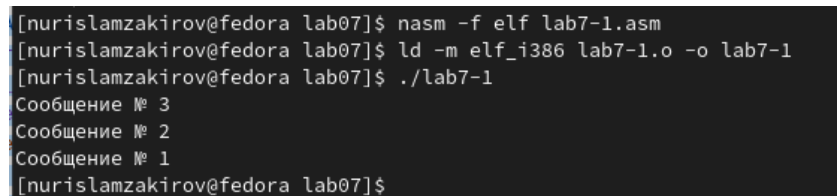
```

lab7-1.asm  [----]  0 L: [ 1+ 0 1/ 23] *(0 / 665b) 0037 0xc
%include 'in_out.asm' ; 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 _label3
_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. 4.6: Изменение текста программы

Для того, чтобы вывод программы был следующим: (рис. 4.7).



```

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

```

Рис. 4.7: Вывод программы

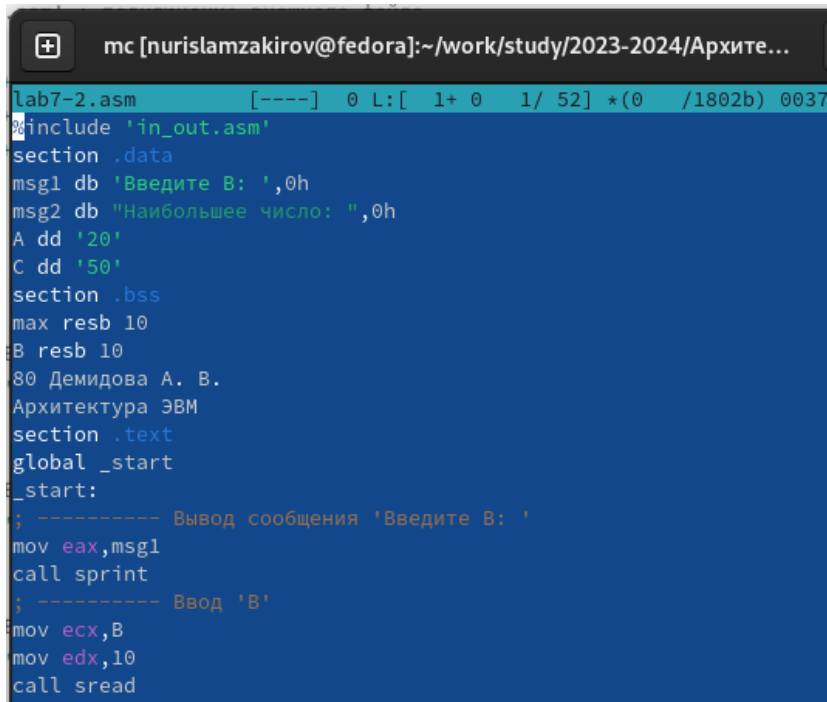
Рассмотрим программу, которая находит и показывает наибольшую из трех целочисленных переменных (А, В и С). Значения А и С задаются программой, а значение В вводится с клавиатуры.

Создаю файл lab7-2.asm (рис. 4.8).

```
[nurislamzakirov@fedora lab07]$ touch lab7-2.asm
[nurislamzakirov@fedora lab07]$
```

Рис. 4.8: Создание файла

Текст программы из листинга ввожу в lab7-2.asm. (рис. 4.9).



```
lab7-2.asm [----] 0 L: [ 1+ 0 1/ 52] *(0 /1802b) 0037
%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
80 Демидова А. В.
Архитектура ЭВМ
section .text
global _start
_start:
; ----- Вывод сообщения 'Введите B: '
mov eax,msg1
call sprint
; ----- Ввод 'B'
mov ecx,B
mov edx,10
call sread
```

Рис. 4.9: Ввод текста программы из листинга 7.3

Создаю исполняемый файл и проверяю его работу. (рис. 4.10).

```
[nurislamzakirov@fedora lab07]$ nasm -f elf lab7-2.asm
[nurislamzakirov@fedora lab07]$ ld -m elf_i386 lab7-2.o -o lab7-2
[nurislamzakirov@fedora lab07]$ ./lab7-2
Введите B: 22
Наибольшее число: 50
[nurislamzakirov@fedora lab07]$ ./lab7-2
Введите B: 70
Наибольшее число: 70
[nurislamzakirov@fedora lab07]$
```

Рис. 4.10: Проверка работы файла

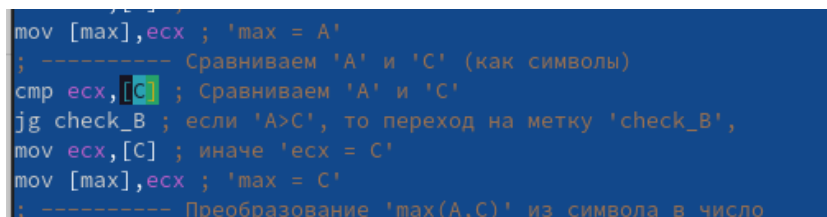
Файл работает корректно.



“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.

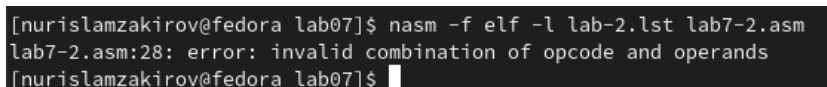
Удаляю выделенный операнд из выбранной мной инструкции с двумя операндами в файле lab7-2.asm, который я открыл (рис. 4.14).



```
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
```

Рис. 4.14: Удаление выделенного операнда из кода

Выполняю трансляцию с получением файла листинга. (рис. 4.15).



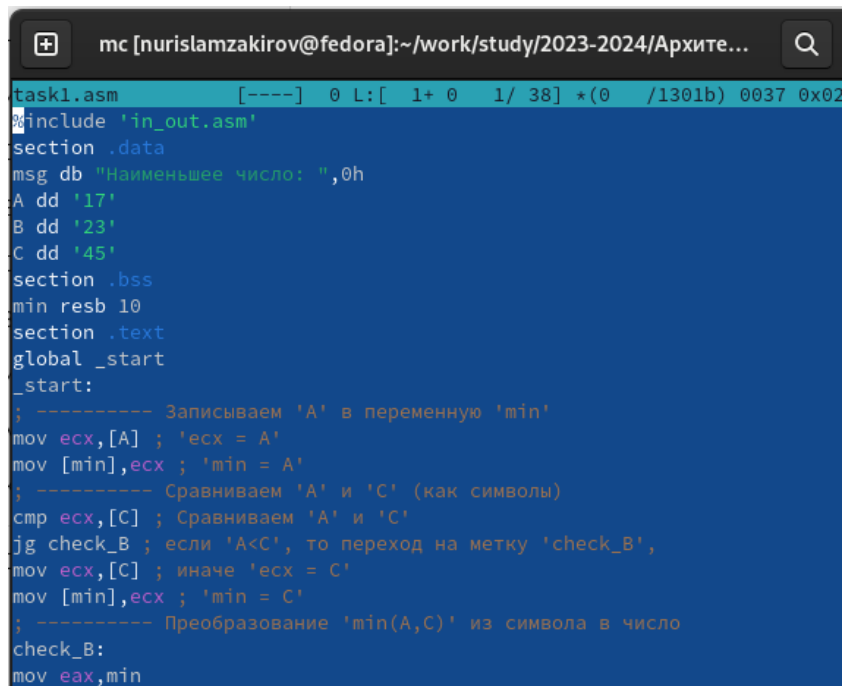
```
[nurislamzakirov@fedora lab07]$ nasm -f elf -l lab-2.lst lab7-2.asm
lab7-2.asm:28: error: invalid combination of opcode and operands
[nurislamzakirov@fedora lab07]$
```

Рис. 4.15: Получение файла листинга

На выходе я не получаю ни одного файла из-за ошибки:инструкция mov (единственная в коде содержит два операнда) не может работать, имея только один операнд, из-за чего нарушается работа кода.

## 4.3 Задания для самостоятельной работы

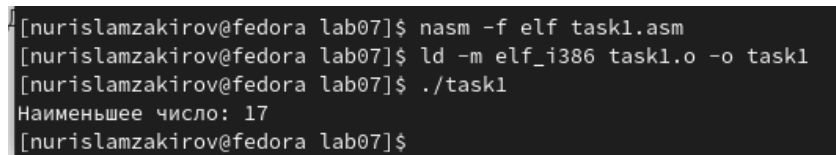
1. Пишу программу нахождения наименьшей из 3 целочисленных переменных a, b и c. Значения переменных выбираю из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 6. Мой вариант под номером 1, поэтому мои значения - 17, 23 и 45. (рис. 4.16).



```
task1.asm      [----]  0 L: [ 1+ 0  1/ 38] *(0  /1301b) 0037 0x02
%include 'in_out.asm'
section .data
msg db "Наименьшее число: ",0h
A dd '17'
B dd '23'
C dd '45'
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'
jg 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
```

Рис. 4.16: Написание программы

Создаю исполняемый файл и проверяю его работу, подставляя необходимые значение. (рис. 4.17).



```
[nurislamzakirov@fedora lab07]$ nasm -f elf task1.asm
[nurislamzakirov@fedora lab07]$ ld -m elf_i386 task1.o -o task1
[nurislamzakirov@fedora lab07]$ ./task1
Наименьшее число: 17
[nurislamzakirov@fedora lab07]$
```

Рис. 4.17: Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

```
%include 'in_out.asm'

section .data

msg db "Наименьшее число: ",0h
```

```
A dd '17'
```

```
B dd '23'
```

```
C dd '45'
```

```
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'
```

```
jg 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 sprint ; Вывод сообщения 'Наименьшее число: '
```

```
mov eax,[min]
```

```
call iprintLF ; Вывод 'min(A,B,C)'
```

```
call quit ; Выход
```

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

$2 \cdot a - x$ , если  $x < a$

8, если  $x \geq a$

(рис. 4.18).

```
task2.asm      [----]  0 L: [ 1+ 0  1/ 54] *(0  /2732b) 0037 0x025  [*][X]
#include 'in_out.asm' ; подключение внешнего файла

SECTION .data ; секция иницированных данных
msg1: DB 'Введите значение переменной x: ',0
msg2: DB 'Введите значение переменной a: ',0
rem: DB 'Результат: ',0

SECTION .bss ; секция не иницированных данных
x: RESB 80 ; Переменная, чье значение будем вводить с клавиатуры, выделенный раз
a: RESB 80 ; Переменная, чье значение будем вводить с клавиатуры, выделенный раз
SECTION .text ; Код программы
GLOBAL _start ; Начало программы
_start: ; Точка входа в программу

mov eax, msg1 ; запись адреса выводимого сообщения в eax
call sprint ; вызов подпрограммы печати сообщения
mov ecx, x ; запись адреса переменной в ecx
mov edx, 80 ; запись длины вводимого значения в edx
call sread ; вызов подпрограммы ввода сообщения
mov eax, x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, eax=x

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

Рис. 4.18: Написание программы

Создаю исполняемый файл и проверяю его работу для значений x и a соответственно: (1;2), (2;1). (рис. 4.19 , 4.20).

```
[nurislamzakirov@fedora lab07]$ ./task2
Введите значение переменной x: 1
Введите значение переменной a: 2
Результат: 3
```

Рис. 4.19: Запуск файла и проверка его работы

```
[nurislamzakirov@fedora lab07]$ ./task2
Введите значение переменной x: 2
Введите значение переменной a: 1
Результат: 8
```

Рис. 4.20: Запуск файла и проверка его работы

Программа работает корректно.

Код программы:

**%include** 'in\_out.asm' ; подключение внешнего файла

**SECTION** .data ; секция инициированных данных

msg1: **DB** 'Введите значение переменной x: ',0

msg2: **DB** 'Введите значение переменной a: ',0

rem: **DB** 'Результат: ',0

**SECTION** .bss ; секция не инициированных данных

x: **RESB** 80 ; Переменная, чьё значение будем вводить с клавиатуры, выделенный размер

a: **RESB** 80 ; Переменная, чьё значение будем вводить с клавиатуры, выделенный размер

**SECTION** .text ; Код программы

**GLOBAL** \_start ; Начало программы

\_start: ; Точка входа в программу

**mov** **eax**, msg1 ; запись адреса выводимого сообщения в **eax**

**call** sprint ; вызов подпрограммы печати сообщения

**mov** **ecx**, x ; запись адреса переменной в **ecx**

**mov** **edx**, 80 ; запись длины вводимого значения в **edx**

**call** sread ; вызов подпрограммы ввода сообщения

**mov** **eax**,x; вызов подпрограммы преобразования

**call** atoi ; ASCII кода в число, **eax**=x

**mov** [x],**eax**

**mov** **eax**, msg2 ; запись адреса выводимого сообщения в **eax**

**call** sprint ; вызов подпрограммы печати сообщения

**mov** **ecx**,a ; запись адреса переменной в **ecx**

**mov** **edx**, 85 ; запись длины вводимого значения в **edx**

**call** sread ; вызов подпрограммы ввода сообщения

**mov** **eax**,a ; вызов подпрограммы преобразования

```

call atoi ; ASCII кода в число, eax=x
mov [a],eax
;-----
cmp eax,[x] ; Сравниваем 'x' и 'a'
jb check_B ; если 'x<a', то переход на метку 'check_B',
jae check_A
;-----
check_A:
mov eax,[a]
shl eax,1
sub eax,[x]; eax =2a - x
mov edi,eax ; запись результата вычисления в 'edi'
jmp _end
;-----
check_B: ;
mov edi,8 ; запись результата вычисления в 'edi'
jmp _end
; ---- Вывод результата на экран
_end:
mov eax,rem ; вызов подпрограммы печати
call sprint ; сообщения 'Результат: '
mov eax,edi ; вызов подпрограммы печати значения
call iprintLF ; из 'edi' в виде символов
call quit ; вызов подпрограммы завершения

```

## 5 Выводы

По завершении этой лабораторной работы я научился использовать команды условного и безусловного перехода, овладел навыками написания программ с использованием этих переходов и узнал о целях и структуре файла листинга. Все это поможет мне в выполнении следующих лабораторных работ.

## 6 Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. 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>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. 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](http://www.stolyarov.info/books/asm_unix).

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).