

# **Лабораторная работа №7.**

**Команды безусловного и условного переходов в Nasm.  
Программирование ветвлений.**

Зейнап Дагделен Реджеповна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
3.1	Команды безусловного перехода . . . . .	7
3.2	Команды условного перехода . . . . .	7
3.2.1	Регистр флагов . . . . .	8
3.2.2	Описание инструкции <code>cmp</code> . . . . .	8
3.2.3	Описание команд условного перехода. . . . .	8
3.2.4	Файл листинга и его структура . . . . .	9
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>10</b>
4.1	Реализация переходов в <code>NASM</code> . . . . .	10
4.2	Изучение структуры файлы листинга . . . . .	16
4.3	Задание для самостоятельной работы . . . . .	18
<b>5</b>	<b>Выводы</b>	<b>25</b>
<b>6</b>	<b>Список литературы</b>	<b>26</b>

## Список иллюстраций

4.1	Создание каталога и файла в ней . . . . .	10
4.2	Код программы . . . . .	11
4.3	Создание исполняемого файла и его запуск . . . . .	11
4.4	Текст программы . . . . .	12
4.5	Создание исполняемого файла и его запуск . . . . .	12
4.6	Текст программы . . . . .	13
4.7	Создание исполняемого файла и его запуск . . . . .	13
4.8	Создание файла . . . . .	15
4.9	Текст программы . . . . .	15
4.10	Создание исполняемого файла и его запуск . . . . .	15
4.11	Создание файла листинга . . . . .	16
4.12	Строки кода . . . . .	16
4.13	Строки кода . . . . .	17
4.14	Трансляция с получением файла листинга . . . . .	17
4.15	Работа команды . . . . .	17
4.16	Создание файла . . . . .	18
4.17	Код программы . . . . .	19
4.18	Создание исполняемого файла и его запуск . . . . .	19
4.19	Создание файла . . . . .	21
4.20	Создание исполняемого файла и его запуск . . . . .	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 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.

### 3.1 Команды безусловного перехода

Безусловный переход выполняется инструкцией `jmp`, которая включает в себя адрес перехода, куда следует передать управление:

`jmp`

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

### 3.2 Команды условного перехода

Как отмечалось выше, для условного перехода необходима проверка какого-либо условия. В ассемблере команды условного перехода вычисляют условие перехода анализируя флаги из регистра флагов.

### 3.2.1 Регистр 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. В следующей таблице указано положение битовых флагов в регистре флагов.

Флаги состояния (биты 0, 2, 4, 6, 7 и 11) отражают результат выполнения арифметических инструкций, таких как ADD, SUB, MUL, DIV.

### 3.2.2 Описание инструкции `cmp`

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:

`cmp` ,

Команда `cmp`, так же как и команда вычитания, выполняет вычитание -, но результат вычитания никуда не записывается и единственным результатом команды сравнения является формирование флагов.

### 3.2.3 Описание команд условного перехода.

Команда условного перехода имеет вид

`j label`

Мнемоника перехода связана со значением анализируемых флагов или со способом формирования этих флагов. В мнемокодах указывается тот результат сравнения, при котором надо делать переход. Мнемоники, идентичные по своему действию. Программист выбирает, какую из них применить, чтобы получить более простой для понимания текст программы.



### 3.2.4 Файл листинга и его структура

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. Итак, 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).

## 4 Выполнение лабораторной работы

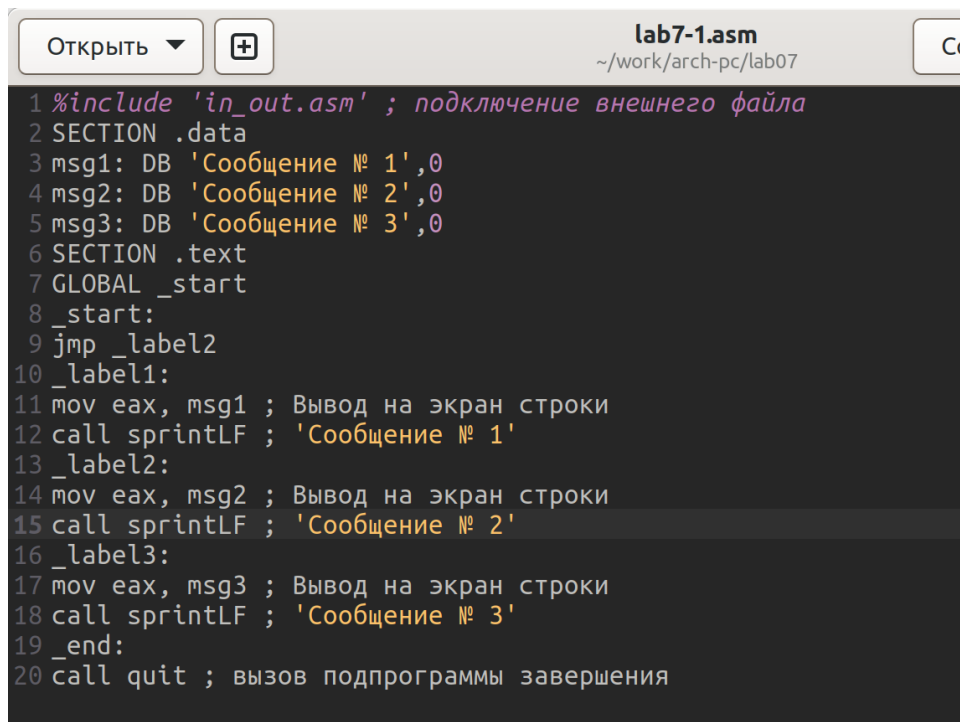
### 4.1 Реализация переходов в NASM

1. Создаю каталог для программ лабораторной работы № 7 (используя команду `mkdir`), перехожу в него (с помощью `cd`) и создаю файл `lab7-1.asm` благодаря `touch` (рис. [4.1]).

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

Рис. 4.1: Создание каталога и файла в ней

2. Инструкция `jmp` в NASM используется для реализации безусловных переходов. Ввожу в файл `lab7-1.asm` текст программы из листинга 7.1 (рис. [4.2]).



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

Рис. 4.2: Код программы

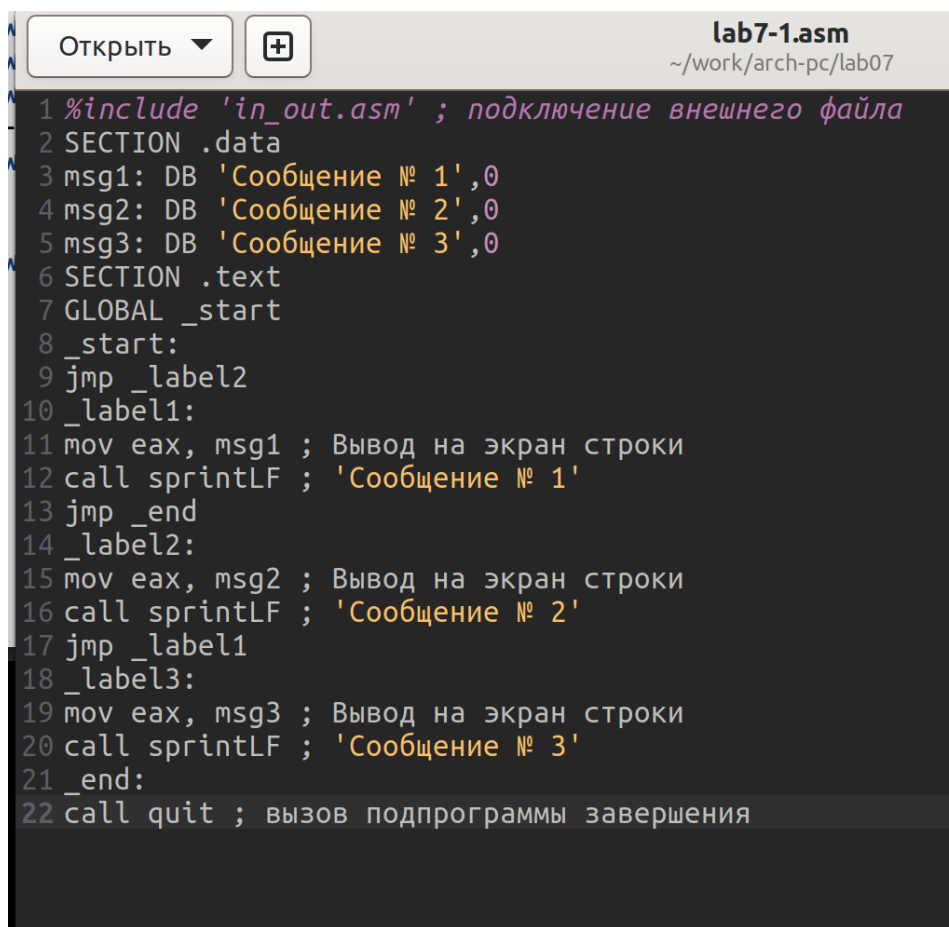
Создаю исполняемый файл и запускаю его. Результат работы данной программы будет следующий (рис. [4.3]).

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

Рис. 4.3: Создание исполняемого файла и его запуск

Таким образом, использование инструкции `jmp _label2` меняет порядок исполнения инструкций и позволяет выполнить инструкции начиная с метки `_label2`, пропустив вывод первого сообщения. Инструкция `jmp` позволяет осуществлять переходы не только вперед, но и назад. Изменяю программу таким образом, чтобы она выводила сначала 'Сообщение № 2', потом 'Сообщение № 1' и завершала

работу. Изменяю текст программы в соответствии с листингом 7.2 (рис. [4.4]).



```
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 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 _end:
22 call quit ; вызов подпрограммы завершения
```

Рис. 4.4: Текст программы

Создаю исполняемый файл и проверяю его работу(рис. [4.5]). Все работает правильно.

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

Рис. 4.5: Создание исполняемого файла и его запуск

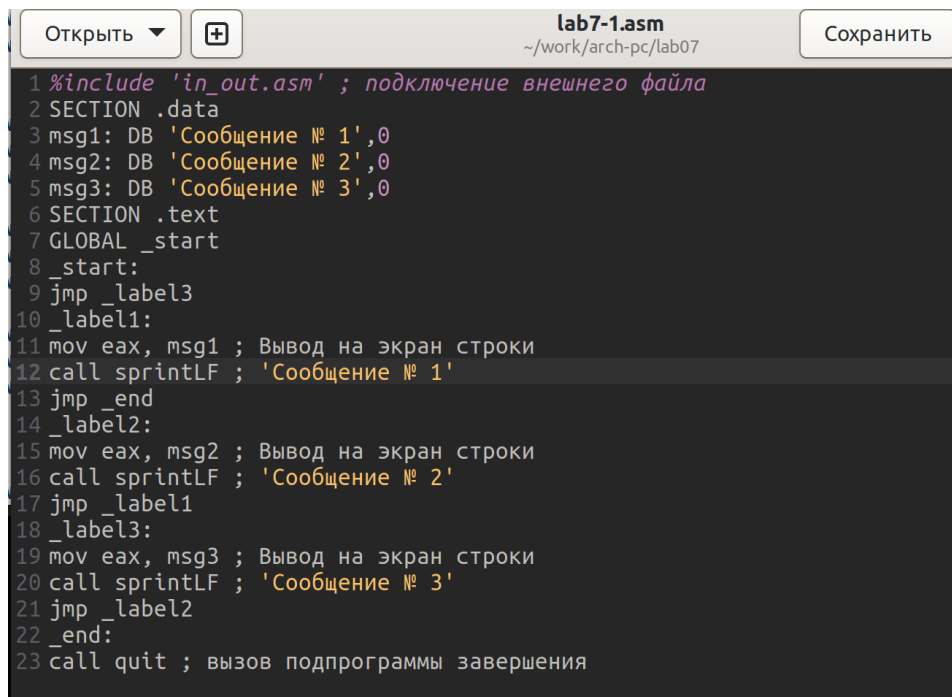
Изменяю текст программы так, чтобы вывод программы был следующим:

Сообщение № 3

Сообщение № 2

Сообщение № 1

Для этого добавляю в текст программы команды `jmp _label3` и `jmp _label2` следующим образом (рис. [4.6]).



```
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. 4.6: Текст программы

Создаю исполняемый файл и проверяю его работу(рис. [4.7]). Все работает правильно.

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

Рис. 4.7: Создание исполняемого файла и его запуск

Текст программы:

```

%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 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'
jmp _label2
_end:
call quit ; вызов подпрограммы завершения

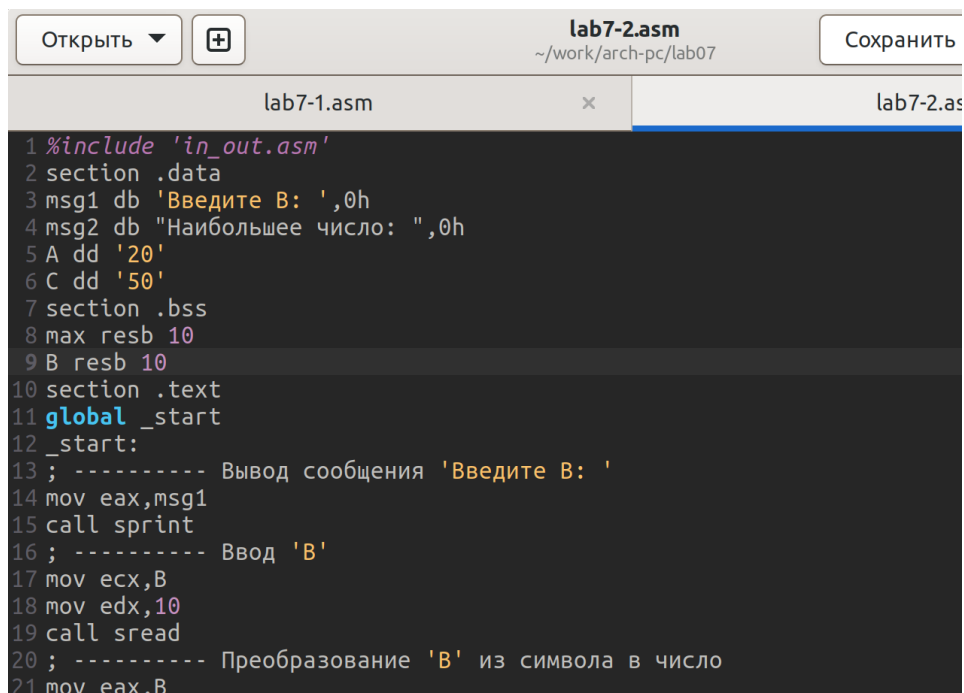
```

3. Использование инструкции `jmp` приводит к переходу в любом случае. Но часто при написании программ необходимо использовать условные переходы. Создаю файл `lab7-2.asm` в каталоге `~/work/arch-pc/lab07` с помощью `touch`(рис. [4.8]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ touch lab7-2.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ ls
in_out.asm lab7-1 lab7-1.asm lab7-1.o lab7-2.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$
```

Рис. 4.8: Создание файла

Ввожу текст программы из листинга 7.3 в lab7-2.asm (рис. [4.9]).



```
lab7-2.asm
~/work/arch-pc/lab07
Сохранить

lab7-1.asm x lab7-2.as

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
```

Рис. 4.9: Текст программы

Создаю исполняемый файл и проверяю его работу для разных значений B. Создаю исполняемый файл и проверяю его работу(рис. [4.10]). Все работает правильно.

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 5
Наибольшее число: 50
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ ./lab7-2
Введите B: 109
Наибольшее число: 109
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$
```

Рис. 4.10: Создание исполняемого файла и его запуск

В данном примере переменные А и С сравниваются как символы, а переменная В и  $\max(A, C)$  как числа.

## 4.2 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. Создаю файл листинга для программы из файла `lab7-2.asm` и открываю файл листинга `lab7-2.lst` с помощью текстового редактора `mcedit`(рис. [4.11]).

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

Рис. 4.11: Создание файла листинга

Объяснение содержимое трёх строк файла листинга (рис. [4.12]):

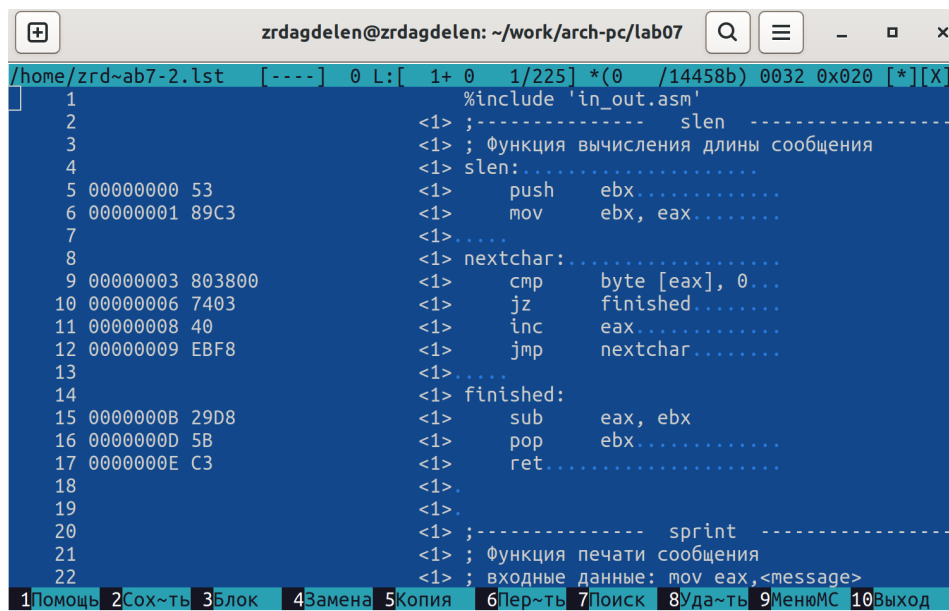


Рис. 4.12: Строки кода

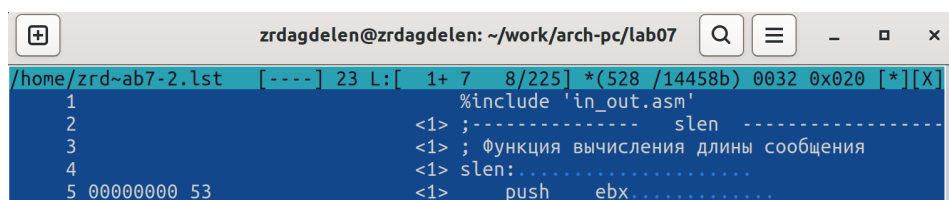


“1” - номер строки кода, “%include in\_out.asm” - подключение внешнего файла in\_out.asm, программа использует этот файл, чтобы выполнять некоторые дальнейшие функции.

“2” - номер строки кода, “; Функция вычисления длины сообщения” - комментарий к коду, 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.

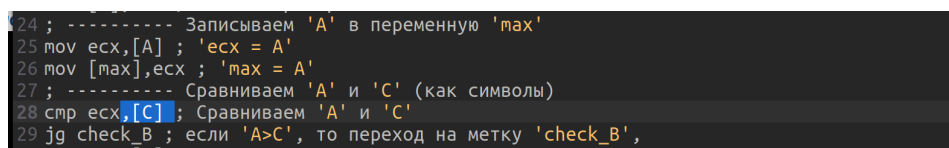
Открываю файл с программой lab7-2.asm и в одной инструкции с двумя операндами удаляю один операнд (рис. [4.13]):



```
zrdagdelen@zrdagdelen: ~/work/arch-pc/lab07
/home/zrd~ab7-2.lst [----] 23 L: [ 1+ 7 8/225] *(528 /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-----
```

Рис. 4.13: Строки кода

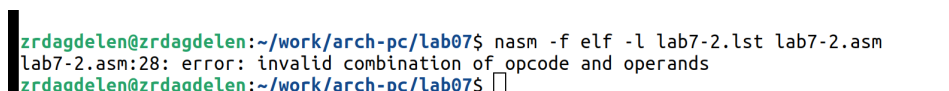
Выполняю трансляцию с получением файла листинга (рис. [4.14]):



```
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] ; инициализация 'ecx = C'
```

Рис. 4.14: Трансляция с получением файла листинга

Результат (рис. [4.15]):



```
zrdagdelen@zrdagdelen:~/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
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$
```

Рис. 4.15: Работа команды

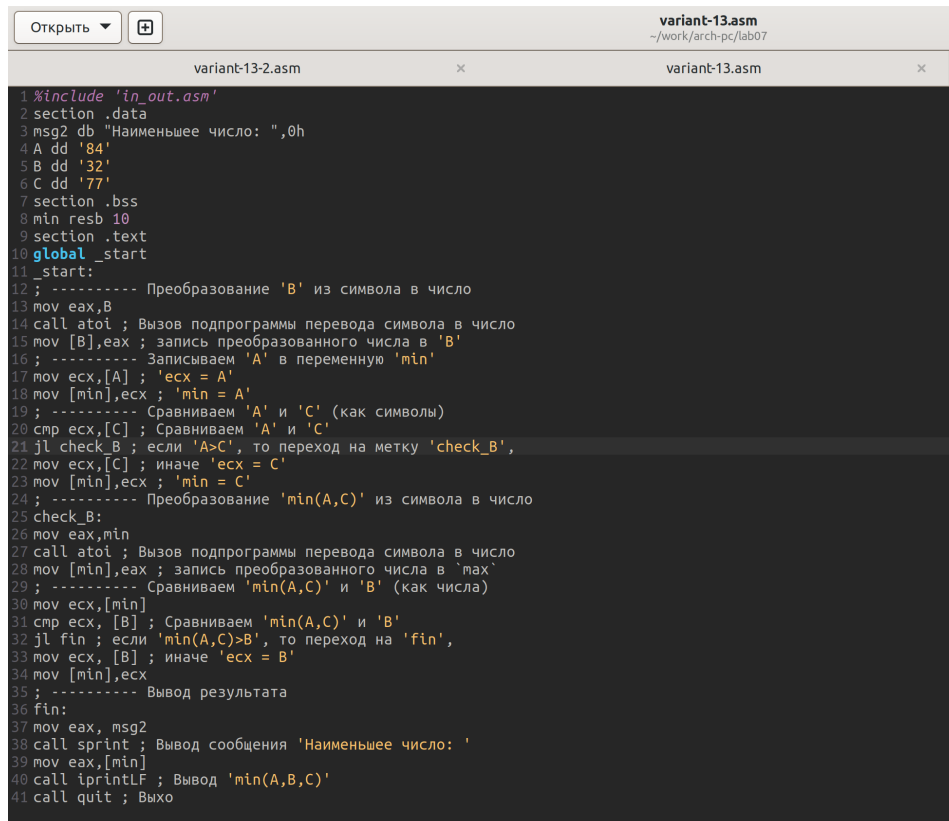
На выходе получается ни один файл из-за ошибки: инструкция `str` не может работать (не с чем сравнить), имея только один операнд, из-за чего нарушается работа кода.

### 4.3 Задание для самостоятельной работы

1. Напишу программу нахождения наименьшей из 3 целочисленных переменных `x`, `y` и `z`. Значения переменных нужно выбрать из табл. 7.5 в соответствии с вариантом, полученным при выполнении лабораторной работы № 6 – у меня 13-ый вариант. Создаю файл `variant-13.asm` с помощью `touch` и пишу программу(рис. [4.16] - рис. [4.17]):

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ touch variant-13.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ ls
in_out.asm  lab7-1  lab7-1.asm  lab7-1.o  lab7-2  lab7-2.asm  lab7-2.lst  variant-13.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$
```

Рис. 4.16: Создание файла



```
1 %include 'in_out.asm'
2 section .data
3 msg2 db "Наименьшее число: ",0h
4 A dd '84'
5 B dd '32'
6 C dd '77'
7 section .bss
8 min resb 10
9 section .text
10 global _start
11 _start:
12 ; ----- Преобразование 'B' из символа в число
13 mov eax,B
14 call atoi ; Вызов подпрограммы перевода символа в число
15 mov [B],eax ; запись преобразованного числа в 'B'
16 ; ----- Записываем 'A' в переменную 'min'
17 mov ecx,[A] ; 'ecx = A'
18 mov [min],ecx ; 'min = A'
19 ; ----- Сравниваем 'A' и 'C' (как символы)
20 cmp ecx,[C] ; Сравниваем 'A' и 'C'
21 jnl check_B ; если 'A>C', то переход на метку 'check_B',
22 mov ecx,[C] ; иначе 'ecx = C'
23 mov [min],ecx ; 'min = C'
24 ; ----- Преобразование 'min(A,C)' из символа в число
25 check_B:
26 mov eax,min
27 call atoi ; Вызов подпрограммы перевода символа в число
28 mov [min],eax ; запись преобразованного числа в 'min'
29 ; ----- Сравниваем 'min(A,C)' и 'B' (как числа)
30 mov ecx,[min]
31 cmp ecx,[B] ; Сравниваем 'min(A,C)' и 'B'
32 jnl fin ; если 'min(A,C)>B', то переход на 'fin',
33 mov ecx,[B] ; иначе 'ecx = B'
34 mov [min],ecx
35 ; ----- Вывод результата
36 fin:
37 mov eax,msg2
38 call sprint ; Вывод сообщения 'Наименьшее число: '
39 mov eax,[min]
40 call iprintLF ; Вывод 'min(A,B,C)'
41 call quit ; Выход
```

Рис. 4.17: Код программы

Создаю исполняемый файл и проверяю его работу (рис. [4.18]):

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ nasm -f elf variant-13.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ ld -m elf_i386 -o variant-13 variant-13.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ ./variant-13
Наименьшее число: 32
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$
```

Рис. 4.18: Создание исполняемого файла и его запуск

Код программы:

```
%include 'in_out.asm'

section .data
msg2 db "Наименьшее число: ",0h
A dd '84'
B dd '32'
```

```

C dd '77'

section .bss
min resb 10

section .text
global _start
_start:
; ----- 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 '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 `max`
; ----- 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, msg2
call sprint ; Вывод сообщения 'Наименьшее число: '
mov eax,[min]
call iprintLF ; Вывод 'min(A,B,C)'
call quit ; Выход

```

2. Напишу программу, которая для введенных с клавиатуры значений  $x$  и  $a$  вычисляет значение заданной функции  $f(x)$  и выводит результат вычислений. Создаю файл variant-13-2.asm с помощью touch (рис. [4.19]):

```

zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ touch variant-13-2.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ ls
in_out.asm lab7-1.asm lab7-2 lab7-2.lst variant-13-2.asm variant-13.o
lab7-1 lab7-1.o lab7-2.asm variant-13 variant-13.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ 

```

Рис. 4.19: Создание файла

Пишу код программы и создаю исполняемый файл и проверяю его работу для значений  $x=3$ ,  $a=9$  и  $x=4$ ,  $a=6$ (рис. [4.20]). Все работает верно.

```

zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ nasm -f elf variant-13-2.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ ld -m elf_i386 -o variant-13-2 variant-13-2.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ ./variant-13-2
Введите a: 9
Введите x: 3
Результат: 2
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ ./variant-13-2
Введите a: 4
Введите x: 6
Результат: 24
zrdagdelen@zrdagdelen:~/work/arch-pc/lab07$ 

```

Рис. 4.20: Создание исполняемого файла и его запуск

Код программы:

```

#include 'in_out.asm'
section .data
msg1 db 'Введите x: ',0h

```

```

msg2 db 'Введите a: ',0h
msg3 db "Результат: ",0h
section .bss
rez resb 80
rez2 resb 80
a resb 80
x resb 80
section .text
global _start
_start:
; ----- Вывод сообщения 'Введите a: '
mov eax, msg2
call sprint
; ----- Ввод 'a'
mov ecx, a
mov edx, 80
call sread
;-----
; ----- Преобразование 'a' из символа в число
mov eax, a
call atoi ; Вызов подпрограммы перевода символа в число
mov [a], eax
; ----- Вывод сообщения 'Введите x: '
mov eax, msg1
call sprint
; ----- Ввод 'x'
mov ecx, x
mov edx, 80
call sread

```

```

; ----- Преобразование 'x' из символа в число
mov eax, x
call atoi ; Вызов подпрограммы перевода символа в число
mov [x], eax

mov eax, [a]
add eax, -7; a=a-7
mov [rez], eax ; 'rez = a-7'

; ----- Сравниваем a и 7
mov ecx, [a]
cmp ecx, 7 ; Сравниваем a и 7
jge fin; если 'a>=7', то переход на метку 'fin',
jmp f_x

f_x:
mov edi, [a]
mov eax, [x]
mul edi
mov [rez2], eax
jmp fin2

fin:
mov eax, msg3
call sprint ; Вывод сообщения
mov eax, [rez]
call iprintLF ; Вывод 'rez'
jmp fin3

```

```
fin2:
mov eax, msg3
call sprint ; Вывод сообщения
mov eax, [rez2]
call iprintLF ; Вывод 'rez2'
jmp fin3
```

```
fin3:
call quit ; Выход
```



## 5 Выводы

Я изучила команды условного и безусловного переходов, приобрела навыки написания программ с использованием переходов. Познакомилась с назначением и структурой файла листинга.

## **6 Список литературы**

Архитектура ЭВМ