

# **Отчёт по лабораторной работе №7**

**Дисциплина: Архитектура компьютера**

Луангсуваннавонг Сайпхачан

# 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>8</b>
4.1	Реализация переходов в NASM . . . . .	8
4.2	Изучение структуры файлы листинга . . . . .	14
<b>5</b>	<b>Выполнение заданий для самостоятельной работы</b>	<b>18</b>
<b>6</b>	<b>Выводы</b>	<b>27</b>
<b>7</b>	<b>Ответы на вопросы для самопроверки</b>	<b>28</b>
<b>8</b>	<b>Список литературы</b>	<b>31</b>

# Список иллюстраций

4.1	Создание файла и каталога . . . . .	8
4.2	Копирование файла . . . . .	8
4.3	Редактирование файла . . . . .	9
4.4	Запуск исполняемого файла . . . . .	10
4.5	Редактирование файла . . . . .	11
4.6	Запуск исполняемого файла . . . . .	12
4.7	Создание файла . . . . .	12
4.8	Редактирование файла . . . . .	13
4.9	Запуск исполняемого файла . . . . .	14
4.10	Компиляция файла . . . . .	14
4.11	Открытие файла . . . . .	15
4.12	Содержимое в файле . . . . .	15
4.13	Редактирование файла . . . . .	17
4.14	Компиляция файла . . . . .	17
4.15	Содержимое в файле . . . . .	17
5.1	Создание файла . . . . .	18
5.2	Редактирование файла . . . . .	19
5.3	Запуск исполняемого файла . . . . .	20
5.4	Создание файла . . . . .	20
5.5	Редактирование файла . . . . .	21
5.6	Запуск исполняемого файла . . . . .	22

# 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 файлы листинга
3. Выполнение заданий для самостоятельной работы

### 3 Теоретическое введение

Для реализации ветвлений в ассемблере используются команды перехода, которые можно разделить на два типа:

- Условный переход – выполнение или невыполнение перехода в определенную точку программы в зависимости от проверки условия.
- Безусловный переход – выполнение перехода в заданную точку программы без каких-либо условий.

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 меткой или адресом области памяти, в которую заранее помещен указатель перехода. Также в качестве операнда может использоваться имя регистра, в таком случае переход осуществляется по адресу, хранящемуся в этом регистре.

Команды условного перехода в ассемблере вычисляют условие перехода, анализируя флаги из регистра 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 помещены в единый регистр флагов, который отражает текущее состояние процессора.

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

Листинг (в рамках `NASM`) — это один из выходных файлов, создаваемых транс-

лятором. Он имеет текстовый вид и используется при отладке программы, так как, помимо строк программы, содержит дополнительную информацию.

Все ошибки и предупреждения, обнаруженные при ассемблировании, выводятся на экран, и файл листинга не создается

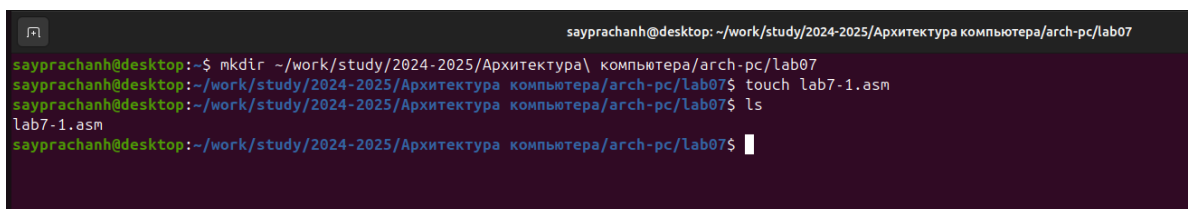
## 4 Выполнение лабораторной работы

### Примечание:

В этой лабораторной работе я буду использовать дистрибутив linux: **Ubuntu**, поскольку я хочу попробовать новый дистрибутив Linux, а также получить практические навыки работы с различными дистрибутивами, и я уже клонировал все предыдущие лабораторные работы в этот новый дистрибутив.

### 4.1 Реализация переходов в NASM

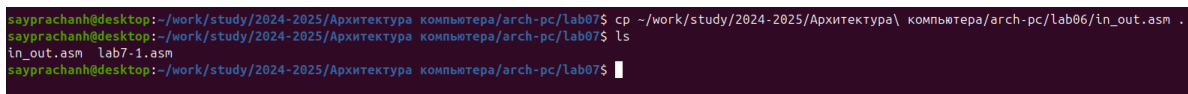
Я создаю новую директорию, в которой буду создавать файлы с программами для лабораторной работы № 7, используя команду `mkdir`. Затем я перехожу в созданный каталог и создаю файл `lab7-1.asm`, используя команду `touch`. (Рис .4.1)



```
sayprachanh@desktop: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07
sayprachanh@desktop:~$ mkdir ~/work/study/2024-2025/Архитектура\ компьютера/arch-pc/lab07
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ touch lab7-1.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ls
lab7-1.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$
```

Рис. 4.1: Создание файла и каталога

Я копирую файл `in_out.asm` из последней лабораторной работы, потому что он будет использоваться в других программах (Рис .4.2)



```
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ cp ~/work/study/2024-2025/Архитектура\ компьютера/arch-pc/lab06/in_out.asm .
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ls
in_out.asm lab7-1.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$
```

Рис. 4.2: Копирование файла



Я открываю созданный файл lab7-1.asm, затем вставляю программу, которая реализует безусловные переходы.(Рис .4.3)

```
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  GLOBAL _start
9  _start:
10
11  jmp _label2
12
13  _label1:
14  mov eax, msg1 ; Вывод на экран строки
15  call sprintfLF ; 'Сообщение № 1'
16
17  _label2:
18  mov eax, msg2 ; Вывод на экран строки
19  call sprintfLF ; 'Сообщение № 2'
20
21  _label3:
22  mov eax, msg3 ; Вывод на экран строки
23  call sprintfLF ; 'Сообщение № 3'
24
25  _end:
26  call quit      ; вызов подпрограммы завершения
```

Рис. 4.3: Редактирование файла

Я создаю новый исполняемый файл и запускаю его, программа выводит текст ” Сообщение № 2” и ” Сообщение № 3”(Рис .4.4). Используя инструкцию jmp \_label1, которая изменяет порядок выполнения инструкций и позволяет нам выполнять инструкции, начиная с метки \_label2, пропуская вывод первого сообщения.

```
sayprachanh@desktop: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf lab7-1.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 3
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$
```

Рис. 4.4: Запуск исполняемого файла

Я снова захожу к файлу программы и меняю программу, добавляя инструкции `jmp _end` и `jmp _label1`(Рис .4.5)

```

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  GLOBAL _start
9  _start:
10
11  jmp _label2
12
13  _label1:
14  mov eax, msg1 ; Вывод на экран строки
15  call sprintf ; 'Сообщение № 1'
16  jmp _end
17
18  _label2:
19  mov eax, msg2 ; Вывод на экран строки
20  call sprintf ; 'Сообщение № 2'
21  jmp _label1
22
23  _label3:
24  mov eax, msg3 ; Вывод на экран строки
25  call sprintf ; 'Сообщение № 3'
26
27  _end:
28  call quit      ; вызов подпрограммы завершения
~

```

Рис. 4.5: Редактирование файла

Я создаю новый исполняемый файл и запускаю его.

На этот раз он выводит текст “Сообщение № 2” и “Сообщение № 1”(Рис .4.6). Поскольку инструкция `jmp _label2` позволяет нам выполнять инструкцию, начинающуюся с метки `_label2`, то в метке `_label2` есть инструкция `jmp _label1`, изменяющая порядок выполнения на метку `_label1`. После этого, используя ко-

манду `jump _end` в метке `_label1`, изменяет выполнение на метку `_end` (переход к инструкции `call quit`), которая завершает работу программы, пропуская вывод третьего сообщения.

```
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf lab7-1.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-1
Сообщение № 2
Сообщение № 1
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$
```

Рис. 4.6: Запуск исполняемого файла

Я создаю `lab7-2.asm` с помощью команды `touch` (Рис .4.7)

```
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ touch lab7-2.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$
```

Рис. 4.7: Создание файла

Я открываю созданный файл и вставляю программу, которая использовала условный переход, чтобы отобразить наибольшую из 3 целочисленных переменных: `A`, `B` и `C` (Рис .4.8)

```

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

```

Рис. 4.8: Редактирование файла

Я создаю новый исполняемый файл и запускаю его, поскольку значения A и C заданы в программе, программа требует от пользователя только ввести значение B.(Рис .4.9) Я ввожу значение B, которое равно 30, оно выдает наибольшее число 50. Поскольку в программе значение A равно 20, а значение C равно 50,

следовательно, наибольшее число между А, В и С равно 50, что равно С Я снова ввожу значение В, но на этот раз я ввожу 60, программа выдает, что наибольшее число равно 60. Что верно, поскольку В - это наибольшее число ( $B > C > A$ ).

```
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf lab7-2.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-2
Введите В: 30
Наибольшее число: 50
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-2
Введите В: 60
Наибольшее число: 60
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$
```

Рис. 4.9: Запуск исполняемого файла

## 4.2 Изучение структуры файлы листинга

Я создаю файл списка, используя ключ -l и указывая имя списка, я использую команду ls для проверки работы выполненной команды(Рис .4.10)

```
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/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
```

Рис. 4.10: Компиляция файла

Затем, используя текстовый редактор helix, я открываю файл lab7-2.lst(Рис .4.11)

```

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. 4.11: Открытие файла

В файле lab7-2.lst одержит подробный вывод, показывающий исходный код ассемблера и соответствующий ему машинный код (или объектный код), который создаёт ассемблер.(Рис .4.12) Я выбираю 3 строки из файла со списком и подробно объясняю это

```

201    26 00000116 890D[00000000]    mov [max],ecx ; 'max = A'
202    27      ;-----Сравниваем 'A' и 'C' (как символы)
203    28 0000011C 3B0D[39000000]    cmp ecx,[C]   ; Сравниваем 'A' и 'C'
204    29 00000122 7F0C              jg check_B    ; если 'A>C', то переход на метку 'check_B',
205    30 00000124 8B0D[39000000]    mov ecx,[C]   ; иначе 'ecx = C'

```

Рис. 4.12: Содержимое в файле

#### Пояснение:

#### Первая строка

26 00000116 890D[00000000] mov [max], ecx ; 'max = A'

Эта инструкция перемещает значение из регистра `ecx` в память по адресу, который обозначен как `max`.

- `mov [max], ecx` означает, что значение из регистра `ecx` сохраняется по адресу `max`.
- Машинный код `890D[00000000]` представляет эту операцию, где `890D` — это код операции для `mov` с операндом в памяти, а `[00000000]` — это адрес `max`.
- Комментарий (`'max = A'`): Это помогает пояснить, что если значение `A` хранится в регистре `ecx`, то оно записывается в переменную `max` (`max = A.`).

### Вторая строка

`28 0000011C 3B0D[39000000] cmp ecx,[C] ; Сравниваем 'A' и 'C'`

Инструкция `cmp ecx, [C]` сравнивает значение в регистре `ecx` (которое хранит `A`) с значением по адресу `C`.

- Машинный код `3B0D[39000000]` соответствует инструкции `cmp` с операндом в памяти. Здесь `3B0D` — это код операции для `cmp`, а `[39000000]` — это адрес переменной `C`.
- Комментарий (`'Сравниваем 'A' и 'C'`): Комментарий точно описывает, что эта инструкция сравнивает значения `A` и `C`.

### Третья строка

`30 00000124 8B0D[39000000] mov ecx,[C] ; 'ecx = C'`

Инструкция `mov ecx, [C]` перемещает значение из памяти по адресу `C` в регистр `ecx`.

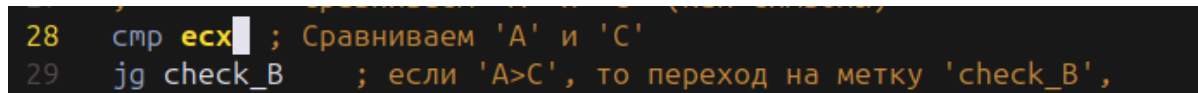
- Машинный код `8B0D[39000000]` соответствует этой операции. `8B0D` — это код операции для `mov` с операндом в памяти, а `[39000000]` — это адрес переменной `C`.



- Комментарий ('ecx = C'): Это правильное уточнение. После этой инструкции регистр ecx будет содержать значение переменной C.

После этого я снова открываю lab7-2.asm, затем удаляю один операнд в одной из инструкций с двумя операндами.

В этой инструкции cmp я удаляю [C] из инструкции. Это должно привести к ошибке, так как для работы этой инструкции требуется два операнда (Рис .4.13)



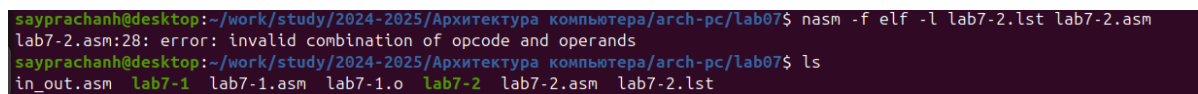
```

28  cmp ecx ; Сравнимаем 'A' и 'C'
29  jg check_B ; если 'A>C', то переход на метку 'check_B',

```

Рис. 4.13: Редактирование файла

Затем я попытался создать объектный файл и файл списка, и это действительно выдало ошибку(Рис .4.14). В файле lab7-2.lst в строке инструкции код инструкции заменяется символом (\*\*\*), который указывает на то, что в этой строке инструкции содержится ошибка.(Рис .4.15)

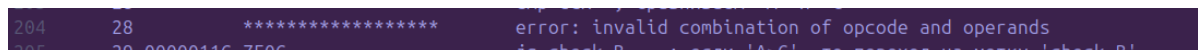


```

sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf -l lab7-2.lst lab7-2.asm
lab7-2.asm:28: error: invalid combination of opcode and operands
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ls
in_out.asm lab7-1 lab7-1.asm lab7-1.o lab7-2 lab7-2.asm lab7-2.lst

```

Рис. 4.14: Компиляция файла



```

204 28 ***** error: invalid combination of opcode and operands
205 29 00000116 750C jg check_B ; если 'A>C', то переход на метку 'check_B',

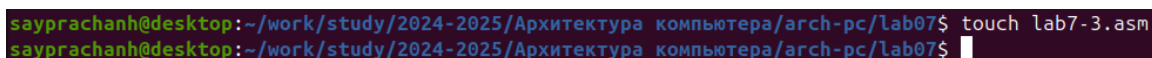
```

Рис. 4.15: Содержимое в файле

В этом случае файловый объектный файл lab7-2.o сгенерирован не будет, поскольку ассемблер не может обработать недопустимую инструкцию, а также в lab7-2.lst мы увидим символ (\*\*\*\*\*) для недопустимой инструкции в файле листинга.

## 5 Выполнение заданий для самостоятельной работы

Я создаю lab7-3.asm для выполнения файла задачи 1 с помощью команды touch(Рис .5.1)



```
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ touch lab7-3.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$
```

Рис. 5.1: Создание файла

Я открываю созданный файл для редактирования, и поскольку мой вариант из последней лабораторной работы (лабораторная работа № 6) равен 13, то мои значения для ввода задачи будут равны 84, 32, 77(Рис .5.2)

```

1  %include 'in_out.asm'
2
3  section .data
4      msg1 db 'Введите a: ', 0h
5      msg2 db 'Введите b: ', 0h
6      msg3 db 'Введите c: ', 0h
7      msg_result db 'Наименьшее число: ', 0h
8
9  section .bss
10     a resb 10
11     b resb 10
12     c resb 10
13     min resb 10
14
15  section .text
16  global _start
17  _start:
18
19  ; -- a
20  mov eax, msg1
21  call sprint
22
23  mov ecx, a
24  mov edx, 10
25  call sread
26
27  ; -- b
28  mov eax, msg2
29  call sprint
30
31  mov ecx, b
32  mov edx, 10
33  call sread
34  █
35  ; -- c
36  mov eax, msg3
37  call sprint
38

```

Рис. 5.2: Редактирование файла

Я создаю и запускаю исполняемый файл, ввожу входные значения: 84, 32, 77

по порядку, и он выдает 32, что является правильным ответом, поскольку 32 - наименьшее число. Чтобы проверить корректность работы программы, я снова ввожу значения, я ввожу 5, 10 и 8, и она выдает 5.(Рис .5.3)

Доказательство того, что программа работает правильно

```
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf lab7-3.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ld -m elf_i386 -o lab7-3 lab7-3.o
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-3
Введите a: 84
Введите b: 32
Введите c: 77
Наименьшее число: 32
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-3
Введите a: 5
Введите b: 10
Введите c: 8
Наименьшее число: 5
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$
```

Рис. 5.3: Запуск исполняемого файла

Я создаю lab7-4.asm для выполнения файла задачи 2 с помощью команды touch(Рис .5.4)

```
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ touch lab7-4.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$
```

Рис. 5.4: Создание файла

Я вхожу в программу, которая требует от пользователя ввода значений 'x' и 'a', и вычисляю их в заданном выражении  $f(x)$ . Поскольку мой вариант равен 13, мои значения (x,a) равны (3;9) и (6;4).(Рис .5.5)

```

1 %include 'in_out.asm'
2 section .data
3 msg1 db 'Введите x: ',0h
4 msg2 db 'Введите a: ',0h
5 msg_result db 'Результат: ',0h
6
7 section .bss
8 result resb 10
9 x resb 10
10 a resb 10
11 section .text
12 global _start
13 _start:
14 ; -- x
15 mov eax, msg1
16 call sprint
17 mov ecx, x
18 mov edx, 10
19 call sread
20 ;--- a
21 mov eax, msg2
22 call sprint
23 mov ecx, a
24 mov edx, 10
25 call sread
26
27 ;-----
28 mov eax, x
29 call atoi
30 mov [x], eax
31
32 mov eax, a
33 call atoi
34 mov [a], eax
35
36 ; -----
37 mov ecx, [x]

```

Рис. 5.5: Редактирование файла

Я создаю и запускаю исполняемый файл, я ввожу значение x, равное 3, и значение a, равное 9, оно выводит 2. (Рис .5.6) Я ввожу значение снова, на этот раз значение x равно 6, а значение a равно 4, и оно выводит 24. Я проверяю работу

программы, рассчитывая самостоятельно. Программа работает правильно.

```
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ nasm -f elf lab7-4.asm
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ld -m elf_i386 -o lab7-4 lab7-4.o
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-4
Введите x: 3
Введите a: 9
Результат: 2
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$ ./lab7-4
Введите x: 6
Введите a: 4
Результат: 24
sayprachanh@desktop:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab07$
```

Рис. 5.6: Запуск исполняемого файла

### Программа для выполнения задачи 1

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

section .data

msg1 db 'Введите a: ', 0h

msg2 db 'Введите b: ', 0h

msg3 db 'Введите c: ', 0h

msg\_result db 'Наименьшее число: ', 0h

section .bss

a resb 10

b resb 10

c resb 10

min resb 10

; Код программы

section .text

global \_start

\_start:

; -- a

```
mov eax, msg1
call sprint
```

```
mov ecx, a
mov edx, 10
call sread
```

```
; -- b
mov eax, msg2
call sprint
```

```
mov ecx, b
mov edx, 10
call sread
```

```
; -- c
mov eax, msg3
call sprint
```

```
mov ecx, c
mov edx, 10
call sread
```

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

```
mov eax, b
call atoi
```

```

mov [b], eax

mov eax, c
call atoi
mov [c], eax

; --
mov eax, [a] ; eax = a
mov ebx, [b] ; ebx = b
cmp eax, ebx
jnl compare_bc ; если a < c
mov eax, ebx ; eax = ebx (b)

compare_bc:
mov ebx, [c]
cmp eax, ebx
jle fin ; если min(a,b) <= c
mov eax, ebx

fin:
mov [min], eax
mov eax, msg_result ; Вывод сообщения 'Наименьшее число: '
call sprint
mov eax, [min]
call iprintLF
call quit

```

## **Программа для выполнения задачи 2**

```
%include 'in_out.asm' ; подключение внешнего файла
```



```
section .data
msg1 db 'Введите x: ',0h
msg2 db 'Введите a: ',0h
msg_result db 'Результат: ',0h
```

```
section .bss
result resb 10
x resb 10
a resb 10
```

```
; Код программы
```

```
section .text
global _start
_start:
; -- x
mov eax, msg1
call sprint
mov ecx, x
mov edx, 10
call sread
;--- a
mov eax, msg2
call sprint
mov ecx, a
mov edx, 10
call sread
```

```
;-----
```

```

mov eax, x
call atoi
mov [x], eax

mov eax, a
call atoi
mov [a], eax

; -----
mov eax, [a] ; eax = a
mov ebx, 7   ; ebx = 7
cmp eax, ebx
jge cal_1 ; если a >= ebx (7)
mov ebx, [x] ; ebx = x
mul ebx ; eax = eax (a) * ebx(x)
jmp fin

cal_1:
mov ebx, 7
sub eax, ebx ; eax = eax - ebx(7)

fin:
mov [result], eax
mov eax, msg_result
call sprint ; Вывод сообщения 'Наименьшее число: '
mov eax, [result]
call iprintLF
call quit

```

## **6 Выводы**

При выполнении данной лабораторной работы, Я изучил команды условного и безусловного перехода, приобрел навыки написания программ с использованием переходов, ознакомился с назначением и структурой файла листинга.

## 7 Ответы на вопросы для самопроверки

### 1. Для чего нужен файл листинга NASM? В чём его отличие от текста программы?

Файл листинга NASM (.lst) нужен для того, чтобы показать, как программа на ассемблере была преобразована в машинный код. Он отличается от обычного текста программы тем, что кроме исходного кода, в нём есть:

- Сгенерированный машинный код.
- Адреса, по которым будут загружены инструкции.
- Ошибки и предупреждения.

Текст программы — это только исходный код без этих дополнительных данных.

### 2. Каков формат файла листинга NASM? Из каких частей он состоит?

Файл листинга обычно состоит из нескольких частей:

- Исходный код — сама программа на ассемблере.
- Машинный код — это бинарное представление команд.
- Адреса памяти — показывают, где будут находиться инструкции в памяти.
- Таблица символов — показывает, где находятся метки и переменные.

- Ошибки и предупреждения — если есть проблемы с кодом.

### **3. Как в программах на ассемблере можно выполнить ветвление?**

Ветвление в ассемблере выполняется с помощью команд перехода:

Безусловный переход (`jmp`) — всегда переходит по указанному адресу. Условные переходы — переходят только при выполнении определённого условия (например, если два значения равны). Условия определяются с помощью флагов, установленных командой `cmp`.

### **4. Какие существуют команды безусловного и условных переходов в языке ассемблера?**

Безусловный переход:

*jmp* — всегда переходит на указанную метку или адрес. Условные переходы:

*je (jump if equal)* — переход, если значения равны.

*jne (jump if not equal)* — переход, если значения не равны.

*jg (jump if greater)* — переход, если первое значение больше второго.

*jge (jump if greater or equal)* — переход, если первое значение больше или равно второму.

*jl (jump if less)* — переход, если первое значение меньше второго.

*jle (jump if less or equal)* — переход, если первое значение меньше или равно второму.

### **5. Опишите работу команды сравнения `cmp`.**

Команда `cmp` сравнивает два значения. Она не сохраняет результат, но обновляет флаги процессора (например, флаг равенства, переполнения). Эти флаги затем используются для условных переходов.

Пример:

```
cmp eax, ebx
```

Эта команда сравнивает значения в регистрах `eax` и `ebx` и обновляет флаги. Важно, что сама команда не сохраняет результат, а только изменяет флаги.

### **6. Каков синтаксис команд условного перехода?**

[команда перехода] [метка]

Пример:

*je label* ; Переход на метку label, если значения равны

*jne label* ; Переход на метку label, если значения не равны

Команды перехода зависят от флагов, установленных командой сравнения (cmp).

**7. Приведите пример использования команды сравнения и команд условного перехода.**

cmp eax, ebx ; Сравниваем eax с ebx

je equal ; Переходим на метку equal, если значения равны

jne not\_equal ; Переходим на метку not\_equal, если значения не равны

Если eax и ebx равны, программа перейдёт на метку equal. Если значения не равны, переход будет на not\_equal.

**8. Какие флаги анализируют команды безусловного перехода?**

Команды безусловного перехода (jmp) не анализируют флаги процессора. Они всегда выполняют переход, независимо от состояния флагов.

## **8 Список литературы**

Архитектура ЭВМ