

# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

## ОТЧЕТ

### ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6

*дисциплина:* Архитектура компьютера

Студент: Крыловецкий Денис Витальевич

Группа: НКАбд-03-25

МОСКВА

2025 г.

## Содержание

<u>1 Цель работы</u> .....	6
<u>2 Задание</u> .....	7
<u>3 Теоретическое введение</u> .....	8
<u>4 Выполнение лабораторной работы</u> .....	9
<u>Символьные и численные данные в NASM</u> .....	9
<u>Выполнение арифметических операций в NASM</u> .....	14
<u>Ответы на вопросы</u> .....	18
<u>Задания для самостоятельной работы</u> .....	19
<u>5 Вывод</u> .....	21
<u>6 Список литературы</u> .....	22

## Список иллюстраций

<a href="#">Рисунок 0.1 Создание нового каталога</a>	9
<a href="#">Рисунок 0.2 Переход в новый каталог</a>	9
<a href="#">Рисунок 0.3 Создание файла</a>	9
<a href="#">Рисунок 0.4 Запись текста программы</a>	10
<a href="#">Рисунок 0.5 Создание исполняемого файла</a>	10
<a href="#">Рисунок 0.6 Создание исполняемого файла</a>	10
<a href="#">Рисунок 0.7 Запуск исполняемого файла</a>	10
<a href="#">Рисунок 0.8 Запись программы с изменением</a>	11
<a href="#">Рисунок 0.9 Создание исполняемого файла</a>	11
<a href="#">Рисунок 0.10 Запуск исполняемого файла</a>	11
<a href="#">Рисунок 0.11 Создание файла</a>	11
<a href="#">Рисунок 0.12 Заход в файл</a>	12
<a href="#">Рисунок 0.13 Запись программы</a>	12
<a href="#">Рисунок 0.14 Создание исполняемого файла</a>	12
<a href="#">Рисунок 0.15 Создание исполняемого файла</a>	12
<a href="#">Рисунок 0.16 Запуск исполняемого файла</a>	12
<a href="#">Рисунок 0.17 Измененный текст программы</a>	13
<a href="#">Рисунок 0.18 Создание исполняемого файла</a>	13
<a href="#">Рисунок 0.19 Запуск файла</a>	13
<a href="#">Рисунок 0.20 Создание файла</a>	14
<a href="#">Рисунок 0.21 Запись текста программы</a>	14
<a href="#">Рисунок 0.22 Создание исполняемого файла</a>	15
<a href="#">Рисунок 0.23 Создание исполняемого файла</a>	15
<a href="#">Рисунок 0.24 Запуск исполняемого файла</a>	15
<a href="#">Рисунок 0.25 Измененный текст программы</a>	15
<a href="#">Рисунок 0.26 Создаем исполняемый файл</a>	15
<a href="#">Рисунок 0.27 Создаем исполняемый файл</a>	15
<a href="#">Рисунок 0.28 Запуск исполняемого файла</a>	16
<a href="#">Рисунок 0.29 Создание файла</a>	16
<a href="#">Рисунок 0.30 Запись текста программы</a>	16
<a href="#">Рисунок 0.31 Создание исполняемого файла</a>	17
<a href="#">Рисунок 0.32 Создание исполняемого файла</a>	17
<a href="#">Рисунок 0.33 Запуск исполняемого файла</a>	17
<a href="#">Рисунок 0.34 Проверка программы</a>	17
<a href="#">Рисунок 0.35 Создание файла</a>	19
<a href="#">Рисунок 0.36 Запись текста программы</a>	19
<a href="#">Рисунок 0.37 Создание исполняемого файла</a>	19
<a href="#">Рисунок 0.38 Проверка работы программы с x1</a>	20
<a href="#">Рисунок 0.39 Проверка работы программы с x2</a>	20

## **Список таблиц**

## **1 Цель работы**

Приобретение практических навыков работы в Midnight Commander.  
Освоение инструкций языка ассемблера mov и int.

## **2 Задание**

Символьные и численные данные в NASM

Выполнение арифметических операций в NASM

Выполнение заданий для самостоятельной работы

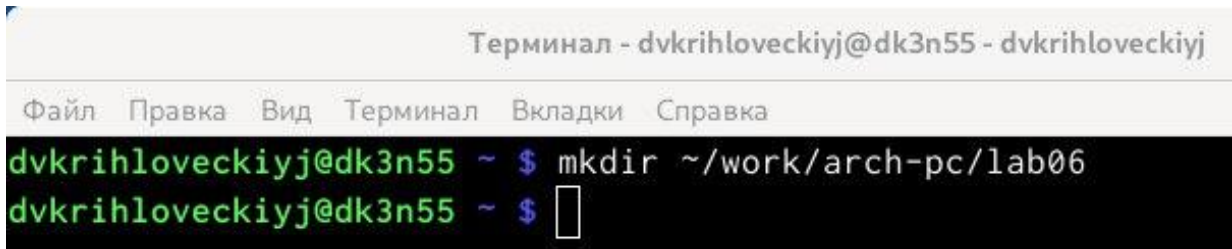
### 3 Теоретическое введение

Большинство инструкций на языке ассемблера требуют обработки операндов. Адрес операнда предоставляет место, где хранятся данные, подлежащие обработке. Это могут быть данные, хранящиеся в регистре или в ячейке памяти. Регистровая адресация—операнды хранятся в регистрах и в команде используются имена этих регистров, например: `movax, bx`. Непосредственная адресация—значение операнда задается непосредственно в команде, например: `movax, 2`. Адресация памяти—операнд задает адрес в памяти. В команде указывается символическое обозначение ячейки памяти, над содержимым которой требуется выполнить операцию. Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации производится согласно кодовой таблице символов ASCII. ASCII—сокращение от American Standard Code for Information Interchange (Американский стандартный код для обмена информацией). Согласно стандарту ASCII каждый символ кодируется одним байтом. Среди инструкций NASM нет такой, которая выводит числа (не в символьном виде). Поэтому, например, чтобы вывести число, надо предварительно преобразовать его цифры в ASCII-коды этих цифр и выводить на экран эти коды, а не само число. Если же выводить число на экран непосредственно, то экран воспримет его не как число, а как последовательность ASCII-символов—каждый байт числа будет воспринят как один ASCII-символ—и выведет на экран эти символы. Аналогичная ситуация происходит и при вводе данных с клавиатуры. Введенные данные будут представлять собой символы, что сделает невозможным получение корректного результата при выполнении над ними.

## 4 Выполнение лабораторной работы

### Символьные и численные данные в NASM

1. Создадим каталог для программ лабораторной работы № 6, перейдем в него и создадим файл lab6-1.asm( рисунок 0.1-0.3)



```
Терминал - dvkrihloveckiyj@dk3n55 - dvkrihloveckiyj
Файл  Правка  Вид  Терминал  Вкладки  Справка
dvkrihloveckiyj@dk3n55 ~ $ mkdir ~/work/arch-pc/lab06
dvkrihloveckiyj@dk3n55 ~ $
```

Рисунок 0.1 Создание нового каталога

```
dvkrihloveckiyj@dk3n55 ~ $ mkdir ~/work/arch-pc/lab06
dvkrihloveckiyj@dk3n55 ~ $ cd ~/work/arch-pc/lab06
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $
```

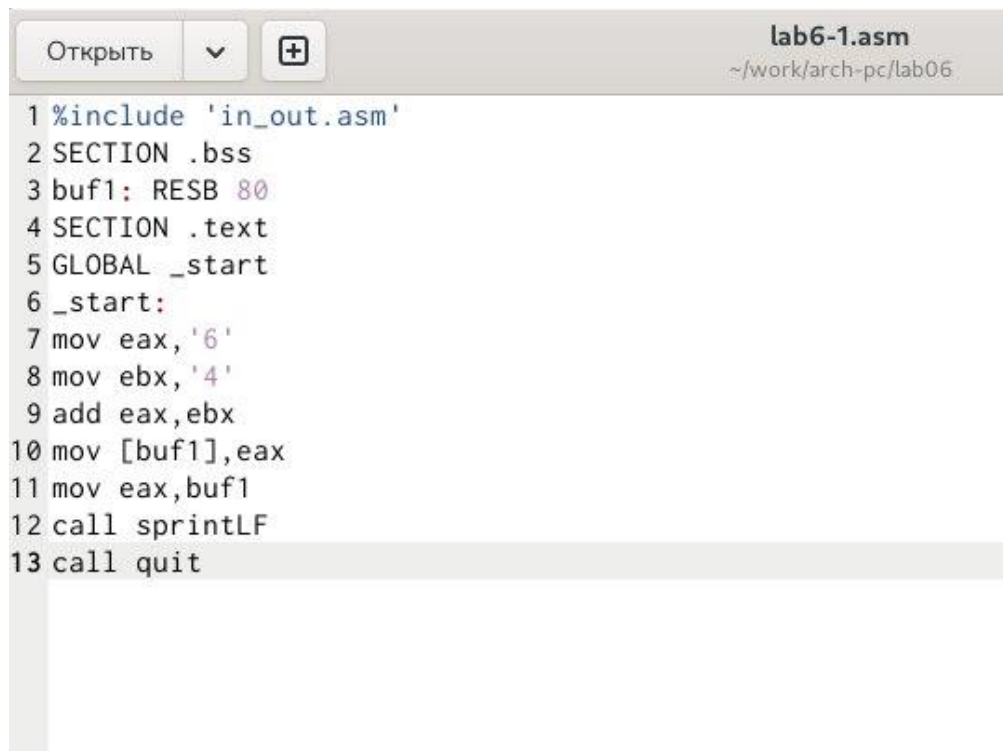
Рисунок 0.2 Переход в новый каталог

```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ touch lab6-1.asm
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $
```

Рисунок 0.3 Создание файла

2. Рассмотрим примеры программ вывода символьных и численных значений. Программы будут выводить значения, записанные в регистр eax.(рисунок 0.4-0.7)



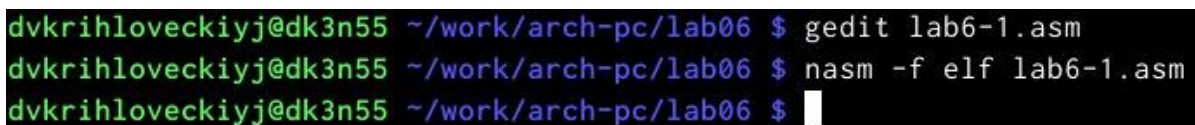


```
lab6-1.asm
~/work/arch-pc/lab06

1 %include 'in_out.asm'
2 SECTION .bss
3 buf1: RESB 80
4 SECTION .text
5 GLOBAL _start
6 _start:
7 mov eax, '6'
8 mov ebx, '4'
9 add eax, ebx
10 mov [buf1], eax
11 mov eax, buf1
12 call printf
13 call _exit
```

Рисунок 0.4 Запись текста программы

Создадим исполняемый файл и запустим его



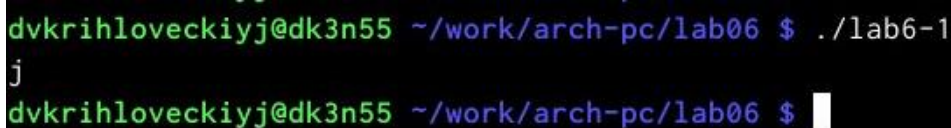
```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ gedit lab6-1.asm
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $
```

Рисунок 0.5 Создание исполняемого файла



```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
```

Рисунок 0.6 Создание исполняемого файла



```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-1
j
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $
```

Рисунок 0.7 Запуск исполняемого файла

Вывод программы отличается от предполагаемого, поскольку по таблице ASCII коды символов в сумме дают j

3. Далее изменим текст программы и вместо символов, запишем в регистры числа. Исправим текст программы следующим образом (рисунок 0.8)

```

dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ gedit lab6-1.asm
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ cat lab6-1.asm
#include 'in_out.asm'

SECTION .bss
buf1: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax,6
mov ebx,4
add eax,ebx
mov [buf1],eax
mov eax,buf1
call sprintLF

call quit

```

**Рисунок 0.8** Запись программы с изменением  
Создадим исполняемый файл и запустим его (рисунок 0.9-0.10)

```

dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o

```

**Рисунок 0.9** Создание исполняемого файла

```

dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-1

dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $

```

**Рисунок 0.10** Запуск исполняемого файла

На этот раз программа выдала пустую строку, поскольку символ 10 означает переход на новую строку

4. Создадим файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 и введем в него текст программы из листинга (рисунок 0.11-0.13)

```

dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/lab6-2.asm
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $

```

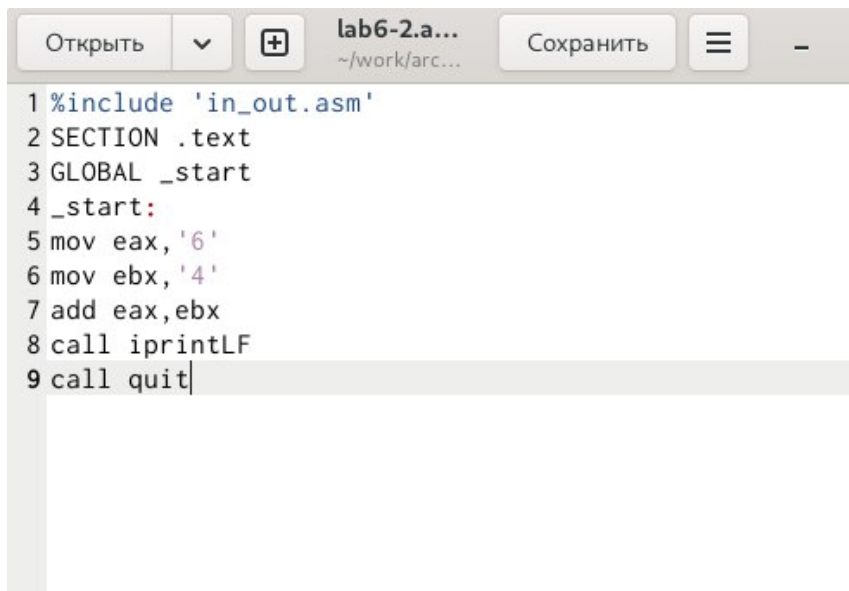
**Рисунок 0.11** Создание файла

```

dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ gedit lab6-2.asm

```

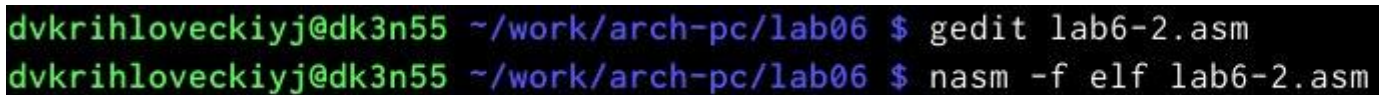
**Рисунок 0.12** Заход в файл



```
1 %include 'in_out.asm'
2 SECTION .text
3 GLOBAL _start
4 _start:
5 mov eax, '6'
6 mov ebx, '4'
7 add eax, ebx
8 call iprintLF
9 call quit
```

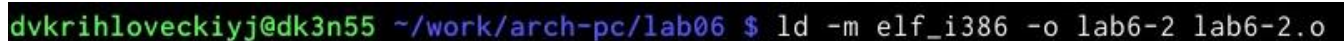
Рисунок 0.13 Запись программы

Создадим исполняемый файл и запустим его (рисунок 0.14-0.16)



```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ gedit lab6-2.asm
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
```

Рисунок 0.14 Создание исполняемого файла



```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
```

Рисунок 0.15 Создание исполняемого файла



```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-2
106
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $
```

Рисунок 0.16 Запуск исполняемого файла

В результате работы программы мы получим число 106. В данном случае, как и в первом, команда add складывает коды символов '6' и '4' ( $54+52=106$ ). Однако, в отличие от программы из листинга, функция `iprintLF` позволяет вывести число, а не символ, кодом которого является это число

5. Аналогично предыдущему примеру изменим символы на числа. Заменим строки (рисунок 0.17)



```
1 %include 'in_out.asm'
2 SECTION .text
3 GLOBAL _start
4 _start:
5 mov eax,6
6 mov ebx,4
7 add eax,ebx
8 call iprintLF
9 call quit
```

Рисунок 0.17 Измененный текст программы

Создадим исполняемый файл и запустим его (рисунок 0.18-0.19)

```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ gedit lab6-2.asm
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
```

Рисунок 0.18 Создание исполняемого файла

```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-2
10
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $
```

Рисунок 0.19 Запуск файла

При исполнении программы мы получим ожидаемый результат – 10

## Выполнение арифметических операций в NASM

6. В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения  $\square(\square) = (5 * 2 + 3) / 3$

Создадим файл lab6-3.asm в каталоге ~/work/arch-pc/lab06 (рисунок 0.23)

```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/lab6-3.asm
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $
```

Рисунок 0.20 Создание файла

Создадим исполняемый файл и запустим его (рисунок 0.21-0.24)

```

1 %include 'in_out.asm'
2
3 SECTION .data
4
5 div: DB 'Результат: ',0
6 rem: DB 'Остаток от деления: ',0
7 SECTION .text
8 GLOBAL _start
9 _start:
10
11 mov eax,5
12 mov ebx,2
13 mul ebx
14 add eax,3
15 xor edx,edx
16 mov ebx,3
17 div ebx
18
19 mov edi,eax
20
21 mov eax,div
22 call sprint
23 mov eax,edi
24 call iprintLF
25
26 mov eax,rem
27 call sprint
28 mov eax,edx
29 call iprintLF
30
31 call quit

```

Рисунок 0.21 Запись текста программы

```

dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ gedit lab6-3.asm
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm

```

Рисунок 0.22 Создание исполняемого файла



```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
```

Рисунок 0.23 Создание исполняемого файла

```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 4
Остаток от деления: 1
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $
```

Рисунок 0.24 Запуск исполняемого файла

Изменим текст программы для вычисления выражения  $\square(\square) = (4 * 6 + 2)/5$ .  
(рисунок 0.25)

```
1 %include 'in_out.asm'
2
3 SECTION .data
4
5 div: DB 'Результат: ',0
6 rem: DB 'Остаток от деления: ',0
7 SECTION .text
8 GLOBAL _start
9 _start:
10
11 mov eax,4
12 mov ebx,6
13 mul ebx
14 add eax,2
15 xor edx,edx
16 mov ebx,5
17 div ebx
18
19 mov edi,eax
20
21 mov eax,div
22 call sprint
23 mov eax,edi
24 call iprintLF
25
26 mov eax,rem
27 call sprint
28 mov eax,edx
29 call iprintLF
30
31 call quit
```

Рисунок 0.25 Измененный текст программы

Создадим исполняемый файл и проверим его работу. (рисунок 0.26-0.28)

```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
```

Рисунок 0.26 Создаем исполняемый файл

```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
```

Рисунок 0.27 Создаем исполняемый файл

```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 5
Остаток от деления: 1
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $
```

Рисунок 0.28 Запуск исполняемого файла

7. В качестве другого примера рассмотрим программу вычисления варианта

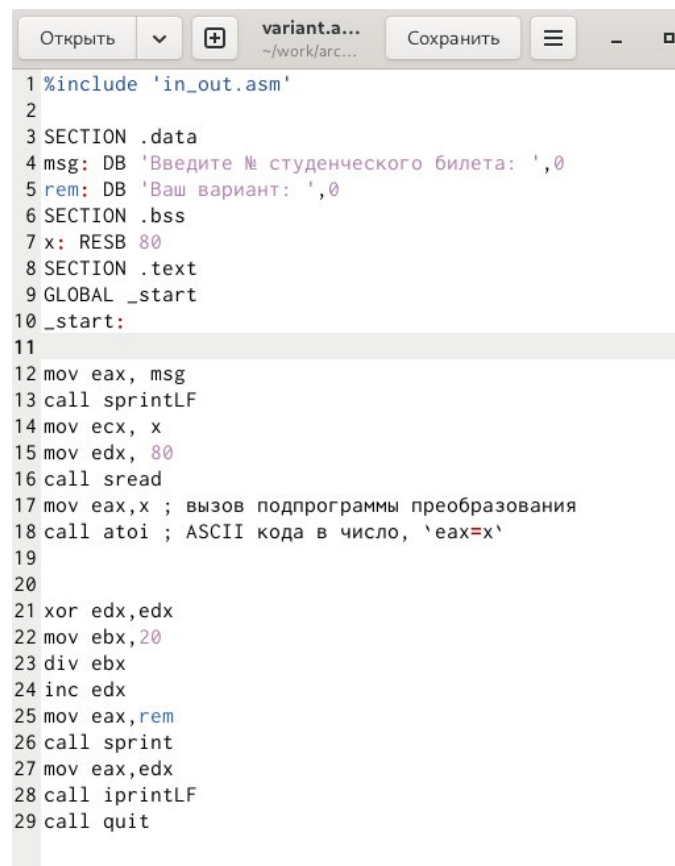
задания по номеру студенческого билета, работающую по следующему алгоритму:

- вывести запрос на введение № студенческого билета
- вычислить номер варианта по формуле:  $(\text{билет} \bmod 20) + 1$ , где билет—номер студенческого билета (В данном случае  $\text{билет} \bmod 20$ —это остаток деления билет на 20).
- вывести на экран номер варианта.

Создадим файл variant.asm в каталоге ~/work/arch-pc/lab06 (рисунок 0.29-0.30)

```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/variant.asm
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $
```

Рисунок 0.29 Создание файла



```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg: DB 'Введите № студенческого билета: ',0
5 rem: DB 'Ваш вариант: ',0
6 SECTION .bss
7 x: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11
12 mov eax, msg
13 call sprintLF
14 mov ecx, x
15 mov edx, 80
16 call sread
17 mov eax, x ; вызов подпрограммы преобразования
18 call atoi ; ASCII кода в число, 'eax=x'
19
20
21 xor edx, edx
22 mov ebx, 20
23 div ebx
24 inc edx
25 mov eax, rem
26 call sprint
27 mov eax, edx
28 call iprintLF
29 call quit
```

Рисунок 0.30 Запись текста программы

Создадим исполняемый файл и запустим его (рисунок 0.31-0.34)

```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ gedit variant.asm
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf variant.asm
```

Рисунок 0.31 Создание исполняемого файла

```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf variant.asm
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o variant variant.o
```

Рисунок 0.32 Создание исполняемого файла



```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ./variant
Введите № студенческого билета:
█
```

Рисунок 0.33 Запуск исполняемого файла

```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ./variant
Введите № студенческого билета:
1032253493
Ваш вариант: 14
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ █
```

Рисунок 0.34 Проверка программы

### Ответы на вопросы:

- 1) За вывод на экран сообщения «Ваш вариант:» отвечают строки:

```
mov eax,rem
```

```
call sprint
```

- 2) Данные инструкции используются для:

- `mov ecx, x` – помещает в `ecx` адрес буфера `x` для ввода
- `mov edx, 80` — задаёт максимальную длину вводимой строки.
- `call sread` — вызов функции чтения строки из стандартного

ввода.

- 3) Инструкция «`call atoi`» используется для:

Вывоза подпрограммы из внешнего файла, которая преобразует `ascii`-код символа в целое число и записывает результат в регистр `eax`

- 4) За вычисление варианта отвечают следующие строки:

```
xor edx,
```

```
edx mov ebx,20
```

```
div ebx
```

```
inc edx
```

- 5) Остаток от деления при выполнении инструкции «`div ebx`» записывается в регистр `edx`

- 6) Инструкция «`<>`» используется для увеличения значения регистра `edx` на 1

- 7) За вывод на экран результата вычислений отвечают следующие строки:

```
mov eax,edx
```

```
call iprintLF
```

## Задания для самостоятельной работы

1 В соответствии с выбранным вариантом по номеру студенческого билета (вариант номер 14), реализуем программу для подсчета функции  $f(x)=(x/2 + 8)*3$  (рисунок 0.35-0.37)

```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/lab6-4.asm
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $
```

Рисунок 0.35 Создание файла

```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ cat lab6-4.asm
#include 'in_out.asm'

SECTION .data
msg: DB 'Введите значение переменной x: ',0
rem: DB 'Результат: ',0
SECTION .bss
x: RESB 80
SECTION .text
GLOBAL _start
_start:
    mov eax, msg
    call sprint

    mov ecx, x
    mov edx, 80
    call sread

    mov eax, x
    call atoi

    mov ebx, 2      ; Делим x на 2
    xor edx, edx    ; Обнуляем edx перед делением
    div ebx         ; eax = x / 2

    add eax, 8      ; eax = x/2 + 8

    mov ebx, 3      ; Умножаем на 3
    mul ebx         ; eax = (x/2 + 8) * 3

    mov edi, eax    ; Сохраняем результат

    mov eax, rem
    call sprint

    mov eax, edi
    call iprint

    call quit
```

Рисунок 0.36 Запись текста программы

```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf lab6-4.asm
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-4 lab6-4.o
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $
```

Рисунок 0.37 Создание исполняемого файла

Проверим работу программы используя переменные  $x_1=1$ ,  $x_2=4$

Для  $x_1=1$  (рисунок 0.38-0.39)

```
dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-4
Введите значение переменной x: 1
Результат: 24dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $
```

Рисунок 0.38 Проверка работы программы с x1

Для  $x_2=4$ :

```
Результат: 24dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-4
Введите значение переменной x: 4
Результат: 30dvkrihloveckiyj@dk3n55 ~/work/arch-pc/lab06 $
```

Рисунок 0.39 Проверка работы программы с x2

## **5 Вывод**

При выполнении данной лабораторной работы я освоила арифметические инструкции языка ассемблера NASM

## 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/Learning-bash-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 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.—БХВ Петербург, 2010.—656 с.—ISBN 978-5-94157-538-1.
14. Столярова 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)