

# Отчет по лабораторной работе №6

Дисциплина: архитектура компьютера

Осипов Никита Александрович

## Содержание

1	Цель работы .....	1
2	Задание.....	1
3	Теоретическое введение.....	1
4	Выполнение лабораторной работы .....	2
4.1	Символьные и численные данные в NASM .....	2
4.2	Выполнение арифметических операций в NASM .....	6
4.3	Ответы на контрольные вопросы .....	9
4.4	Задание для самостоятельной работы .....	10
5	Выводы .....	11
6	Список литературы .....	11

## 1 Цель работы

Цель данной лабораторной работы - освоение арифметических инструкций языка ассемблера NASM.

## 2 Задание

1. Символьные и численные данные в NASM
2. Выполнение арифметических операций в NASM
3. Выполнение заданий для самостоятельной работы

## 3 Теоретическое введение

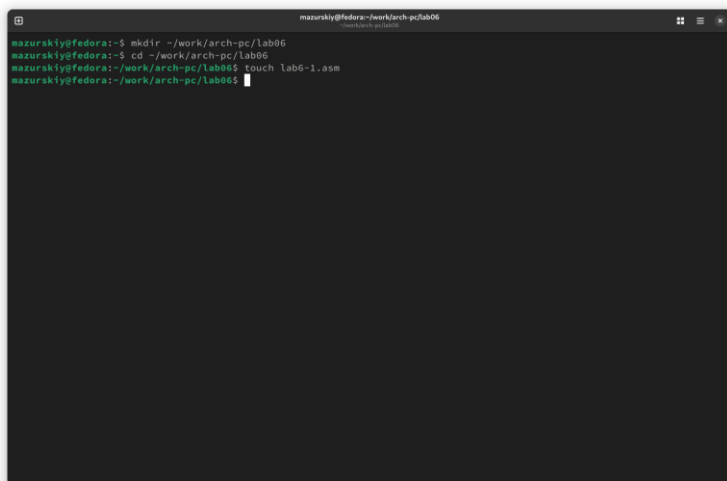
Большинство инструкций на языке ассемблера требуют обработки операндов. Адрес операнда предоставляет место, где хранятся данные, подлежащие обработке. Это могут быть данные хранящиеся в регистре или в ячейке памяти. - Регистровая адресация – операнды хранятся в регистрах и в команде используются имена этих регистров, например: `mov ax,bx`. - Непосредственная адресация – значение операнда задается непосредственно в команде, Например: `mov ax,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 арифметических операций. Для решения этой проблемы необходимо проводить преобразование ASCII символов в числа и обратно.

## 4 Выполнение лабораторной работы

### 4.1 Символьные и численные данные в NASM

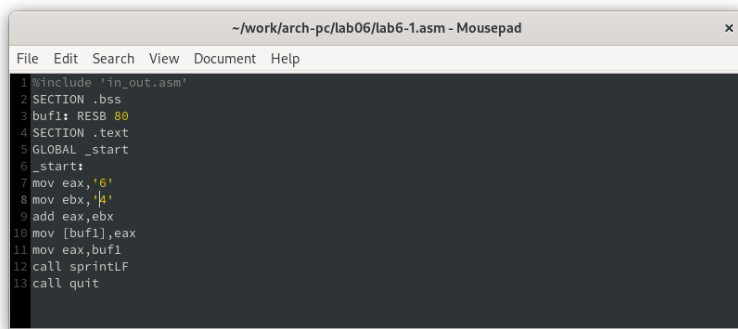
Создаю каталог для программ лабораторной работы №6 и перехожу в него, создаю там файл (рис. 1).



```
mazurskiy@fedora:~/work/arch-pc/lab06
mazurskiy@fedora:~$ mkdir -p /work/arch-pc/lab06
mazurskiy@fedora:~$ cd /work/arch-pc/lab06
mazurskiy@fedora:~/work/arch-pc/lab06$ touch lab06-1.asm
mazurskiy@fedora:~/work/arch-pc/lab06$
```

Рис. 1: Создание нового каталога

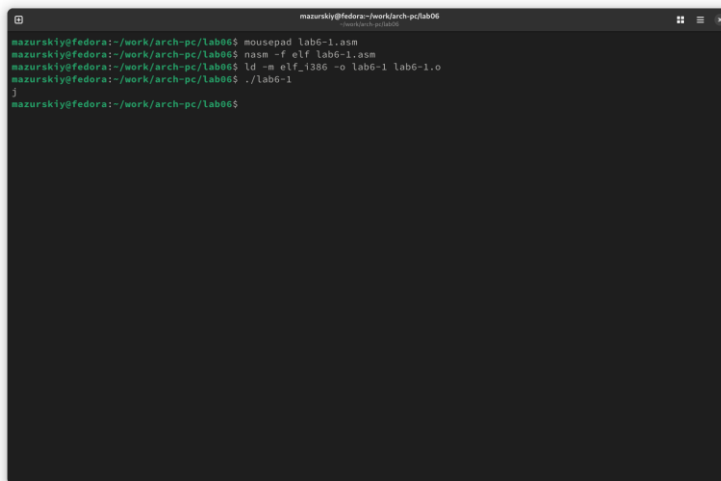
В созданном файле ввожу программу из листинга (рис. 2).



```
~/work/arch-pc/lab06/lab6-1.asm - Mousepad
File Edit Search View Document Help
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 sprintf
13 call quit
```

Рис. 2: Сохранение новой программы

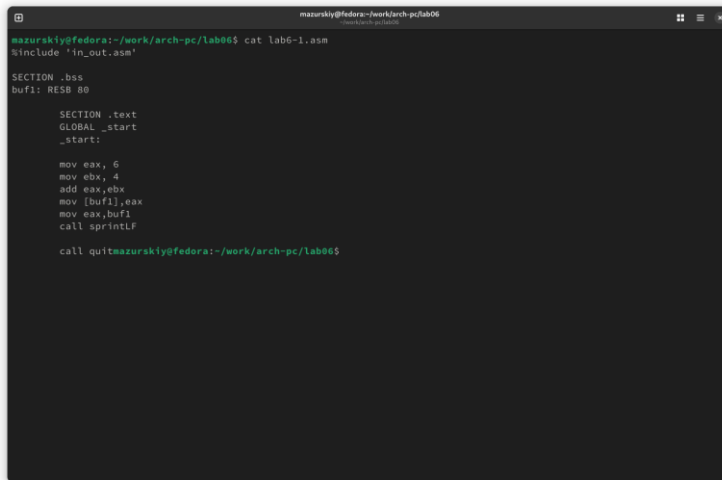
Создаю исполняемый файл и запускаю его, вывод программы отличается от предполагаемого изначально, ибо коды символов в сумме дают символ j по таблице ASCII.  
{#fig:003 width=70%}



```
mazurskiy@fedora:~/work/arch-pc/lab06$ mousepad lab6-1.asm
mazurskiy@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-1.asm
mazurskiy@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-1 lab6-1.o
mazurskiy@fedora:~/work/arch-pc/lab06$ ./lab6-1
j
mazurskiy@fedora:~/work/arch-pc/lab06$
```

Рис. 3: Запуск изначальной программы

Изменяю текст изначальной программы, убрав кавычки (рис. 4).



```
mazurskiy@fedora:~/work/arch-pc/lab06$ cat lab6-1.asm
#include "in_out.asm"

SECTION .bss
buf1: RESB 80

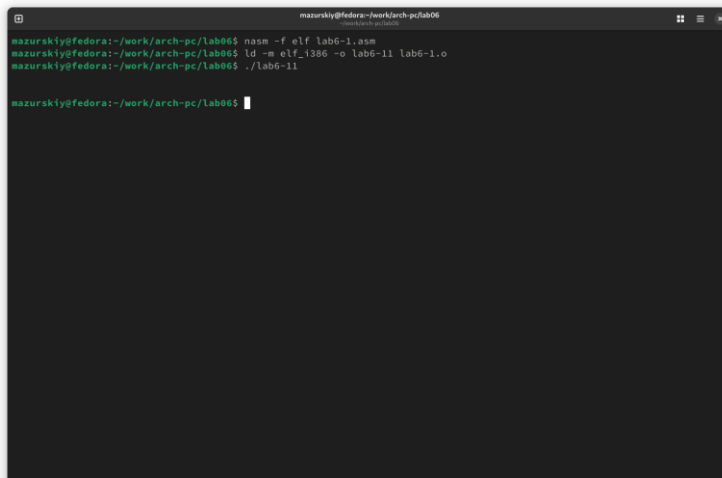
SECTION .text
GLOBAL _start
_start:

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

    call quitmazurskiy@fedora:~/work/arch-pc/lab06$
```

Рис. 4: Измененная программа

На этот раз программа выдала пустую строку, это связано с тем, что символ 10 означает переход на новую строку (рис. 5).

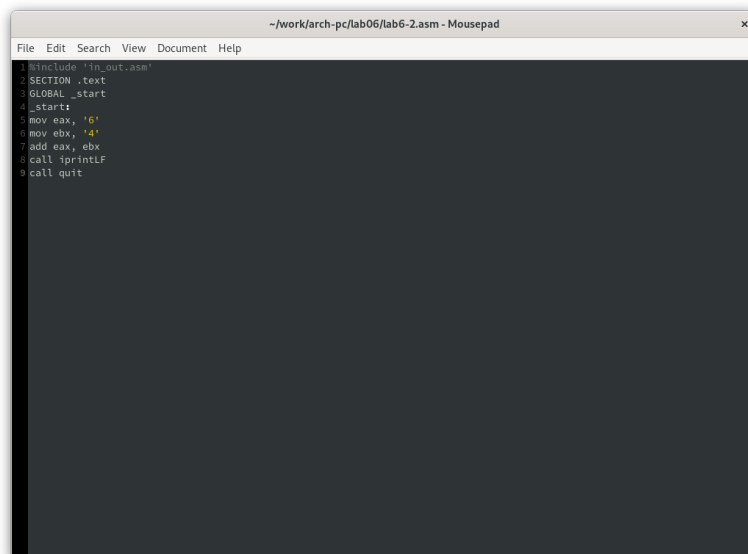


```
mazurskiy@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-1.asm
mazurskiy@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-11 lab6-1.o
mazurskiy@fedora:~/work/arch-pc/lab06$ ./lab6-11

mazurskiy@fedora:~/work/arch-pc/lab06$
```

Рис. 5: Запуск измененной программы

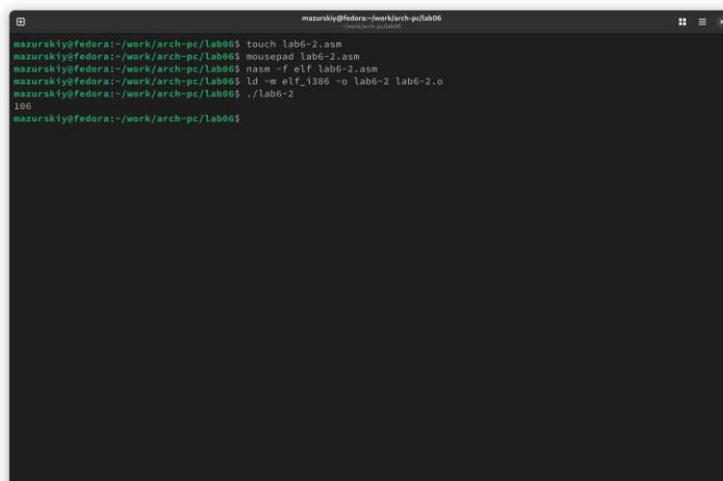
Создаю новый файл для будущей программы и записываю в нее код из листинга (рис. 6).



```
~/work/arch-pc/lab06/lab6-2.asm - Mousepad
File Edit Search View Document Help
;include "in_out.asm"
;SECTION .text
;GLOBAL _start
;_start:
;mov eax, '6'
;mov ebx, '4'
;add eax, ebx
;call _printf
;call _quit
```

*Рис. 6: Вторая программа*

Создаю исполняемый файл и запускаю его, теперь отображается результат 106, программа, как и в первый раз, сложила коды символов, но вывела само число, а не его символ, благодаря замене функции вывода на `iprintLF` (рис. 7).



```
mazurskiy@fedora:~/work/arch-pc/lab06$ touch lab6-2.asm
mazurskiy@fedora:~/work/arch-pc/lab06$ mousepad lab6-2.asm
mazurskiy@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
mazurskiy@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
mazurskiy@fedora:~/work/arch-pc/lab06$ ./lab6-2
106
mazurskiy@fedora:~/work/arch-pc/lab06$
```

*Рис. 7: Вывод второй программы*

Убрав кавычки в программе, я снова ее запускаю и получаю предполагаемый изначально результат. (рис. 8).

```
mazurskiy@fedora:~/work/arch-pc/lab06$ mousepad lab6-2.asm
mazurskiy@fedora:~/work/arch-pc/lab06$ cat lab6-2.asm
%include "in_out.asm"
SECTION .text
GLOBAL _start
_start:
mov eax, 5
mov ebx, 4
add eax, ebx
call iprintLF
call quit
mazurskiy@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
mazurskiy@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-21 lab6-2.o
mazurskiy@fedora:~/work/arch-pc/lab06$ ./lab6-21
10
mazurskiy@fedora:~/work/arch-pc/lab06$
```

Рис. 8: Вывод измененной второй программы

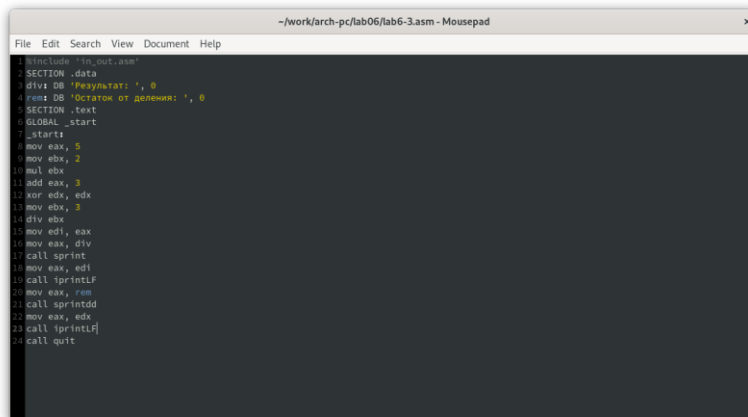
Заменяв функцию вывода на `iprint`, я получаю тот же результат, но без переноса строки (рис. 9).

```
mazurskiy@fedora:~/work/arch-pc/lab06$ cat lab6-2.asm
%include "in_out.asm"
SECTION .text
GLOBAL _start
_start:
mov eax, 6
mov ebx, 4
add eax, ebx
call iprint
call quit
mazurskiy@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
mazurskiy@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-22 lab6-2.o
mazurskiy@fedora:~/work/arch-pc/lab06$ ./lab6-22
10mazurskiy@fedora:~/work/arch-pc/lab06$
```

Рис. 9: Замена функции вывода во второй программе

## 4.2 Выполнение арифметических операций в NASM

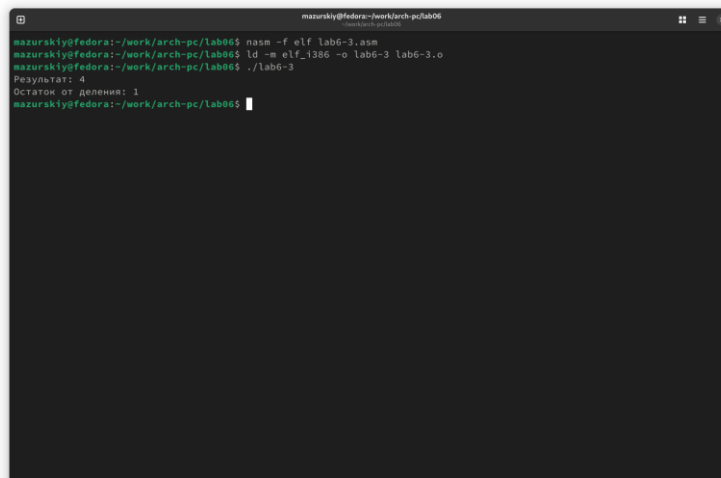
Создаю новый файл и копирую в него содержимое листинга (рис. 10).



```
1 include "fn_out.asm"
2 SECTION .data
3 divt DB "Результат: ", 0
4 remt DB "Остаток от деления: ", 0
5 SECTION .text
6 GLOBAL _start
7 _start:
8 mov eax, 5
9 mov ebx, 2
10 mul ebx
11 add eax, 2
12 xor edx, edx
13 mov ebx, 3
14 div ebx
15 mov edi, eax
16 mov eax, div
17 call sprint
18 mov eax, edi
19 call iprintf
20 mov eax, rem
21 call sprintd
22 mov eax, edx
23 call iprintf
24 call quit
```

Рис. 10: Третья программа

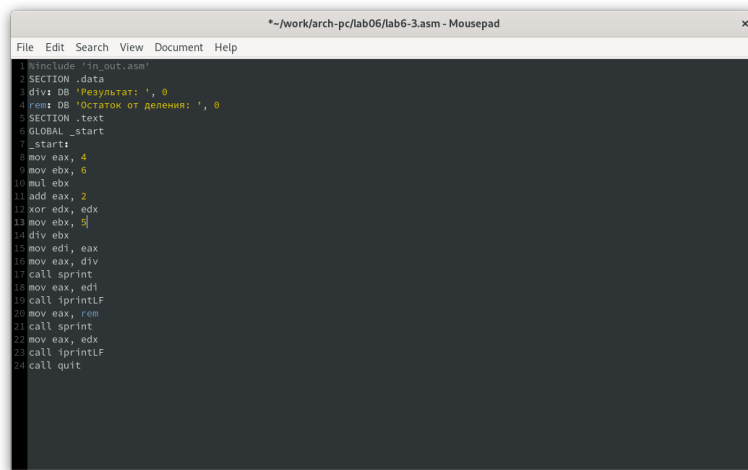
Программа выполняет арифметические вычисления, на вывод идет результирующее выражения и его остаток от деления (рис. 11).



```
mazurskiy@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
mazurskiy@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
mazurskiy@fedora:~/work/arch-pc/lab06$ ./lab6-3
Результат: 4
Остаток от деления: 1
mazurskiy@fedora:~/work/arch-pc/lab06$
```

Рис. 11: Запуск третьей программы

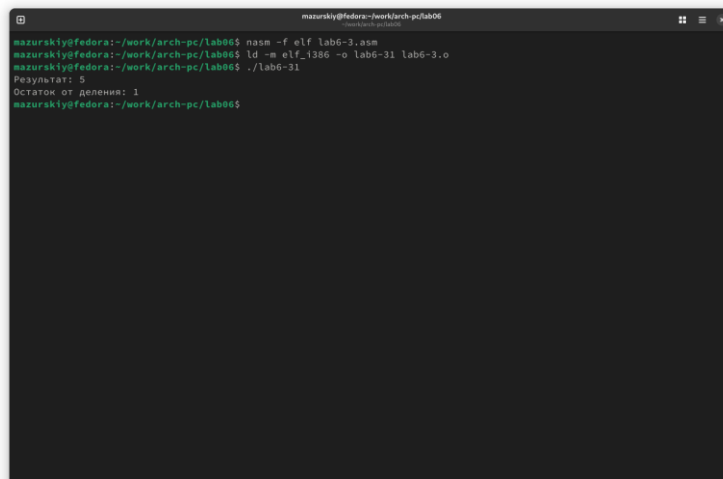
Заменив переменные в программе для выражения  $f(x) = (4*6+2)/5$  (рис. 12).



```
*~/work/arch-pc/lab06/lab6-3.asm - Mousepad
File Edit Search View Document Help
1 include "in_out.asm"
2 SECTION .data
3 divi DB 'Результат: ', 0
4 remi DB 'Остаток от деления: ', 0
5 SECTION .text
6 GLOBAL _start
7 _start:
8 mov eax, 4
9 mov ebx, 6
10 mul ebx
11 add eax, 2
12 xor edx, edx
13 mov ebx, 5
14 div ebx
15 mov edi, eax
16 mov eax, div
17 call sprint
18 mov eax, edi
19 call iprintf
20 mov eax, rem
21 call sprint
22 mov eax, edx
23 call iprintf
24 call quit
```

Рис. 12: Изменение третьей программы

Запуск программы дает корректный результат (рис. 13).

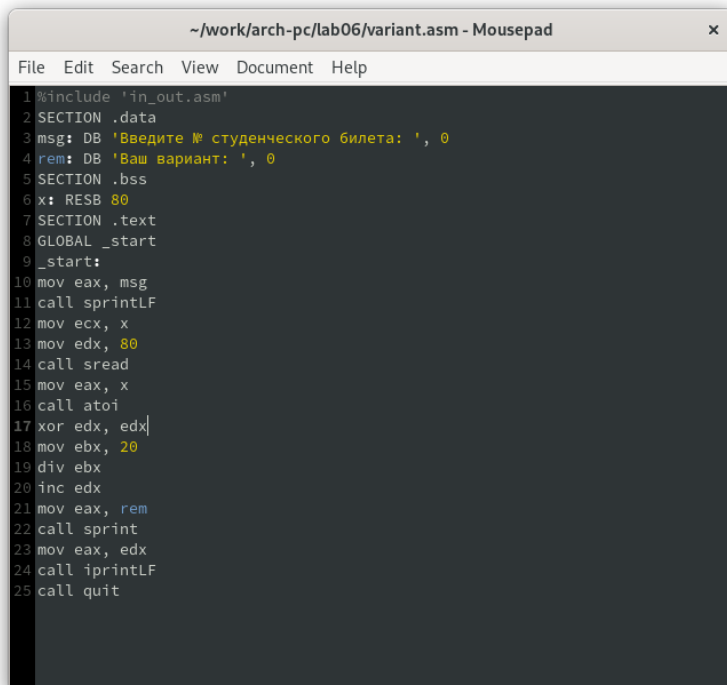


```
mazurskiy@fedora:~/work/arch-pc/lab06
mazurskiy@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
mazurskiy@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-31 lab6-3.o
mazurskiy@fedora:~/work/arch-pc/lab06$ ./lab6-31
Результат: 5
Остаток от деления: 1
mazurskiy@fedora:~/work/arch-pc/lab06$
```

Рис. 13: Запуск измененной третьей программы

Создаю новый файл и помещаю текст из листинга (рис. 14).

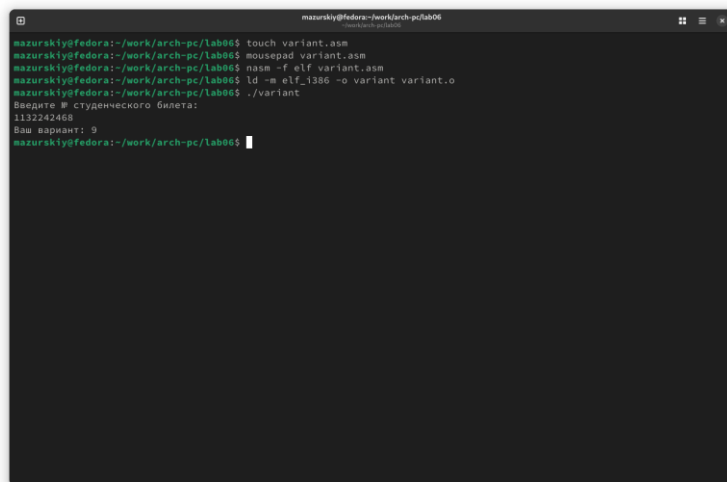




```
~/work/arch-pc/lab06/variant.asm - Mousepad
File Edit Search View Document Help
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите № студенческого билета: ', 0
4 rem: DB 'Ваш вариант: ', 0
5 SECTION .bss
6 x: RESB 80
7 SECTION .text
8 GLOBAL _start
9 _start:
10 mov eax, msg
11 call sprintf
12 mov ecx, x
13 mov edx, 80
14 call sread
15 mov eax, x
16 call atoi
17 xor edx, edx
18 mov ebx, 20
19 div ebx
20 inc edx
21 mov eax, rem
22 call sprintf
23 mov eax, edx
24 call iprintf
25 call quit
```

Рис. 14: Программа для подсчета варианта

Запустив программу и указав свой номер студенческого билета, я получил свой вариант для дальнейшей работы. (рис. 15).



```
mazurskiy@fedora:~/work/arch-pc/lab06$ touch variant.asm
mazurskiy@fedora:~/work/arch-pc/lab06$ mousepad variant.asm
mazurskiy@fedora:~/work/arch-pc/lab06$ nasm -f elf variant.asm
mazurskiy@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o variant variant.o
mazurskiy@fedora:~/work/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1132242468
Ваш вариант: 9
mazurskiy@fedora:~/work/arch-pc/lab06$
```

Рис. 15: Запуск программы для подсчета варианта

## 4.3 Ответы на контрольные вопросы

1. За вывод сообщения “Ваш вариант” отвечают строки кода:

```
mov eax, rem
call sprintf
```

- Инструкция `mov ecx, x` используется, чтобы положить адрес вводимой строки `x` в регистр `ecx` `mov edx, 80` - запись в регистр `edx` длины вводимой строки `call sread` - вызов подпрограммы из внешнего файла, обеспечивающей ввод сообщения с клавиатуры.
- `call atoi` используется для вызова подпрограммы из внешнего файла, которая преобразует `ascii`-код символа в целое число и записывает результат в регистр `eax`.
- За вычисления варианта отвечают строки:

```
xor edx,edx ; обнуление edx для корректной работы div
mov ebx,20 ; ebx = 20
div ebx ; eax = eax/20, edx - остаток от деления
inc edx ; edx = edx + 1
```

- При выполнении инструкции `div ebx` остаток от деления записывается в регистр `edx`.
- Инструкция `inc edx` увеличивает значение регистра `edx` на 1.
- За вывод на экран результатов вычислений отвечают строки:

```
mov eax,edx
call iprintLF
```

## 4.4 Задание для самостоятельной работы

В соответствии с выбранным вариантом, я реализую программу для подсчета функции  $f(x) = 10 + (31x - 5)$ , проверка на нескольких переменных показывает корректное выполнение программы (рис. 16).

```
mazurskiy@vbox:~/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, 31
mul ebx
sub eax, 5
add eax, 10
mov edi, eax
mov eax, rem
call sprint
mov eax, edi
call iprint
call quitmazurskiy@vbox:~/work/arch-pc/lab06$ nasm -f elf lab6-4.asm

mazurskiy@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-4 lab6-4.o

mazurskiy@vbox:~/work/arch-pc/lab06$ ./lab6-4
Введите значение переменной x: 3
Результат: 98mazurskiy@vbox:~/work/arch-pc/lab06$ ./lab6-4
Введите значение переменной x: 1
mazurskiy@vbox:~/work/arch-pc/lab06$
```

Рис. 16: Запуск и проверка программы

Прилагаю код своей программы:

```

%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, 31
mul ebx
sub eax, 5
add eax, 10
mov edi, eax
mov eax, rem
call sprint
mov eax, edi
call iprint

```

## 5 Выводы

При выполнении данной лабораторной работы я освоил арифметические инструкции языка ассемблера NASM.

## 6 Список литературы

1. Пример выполнения лабораторной работы
2. Курс на ТУИС
3. Лабораторная работа №6
4. Программирование на языке ассемблера NASM Столяров А. В.