Отчет по выполнению лабораторной работы №6

Дисциплина: Архитетура компьютера

Ефремова Полина Александровна

Содержание

# 1 Цель работы

Основная цель данной лабораторной работы - освоение арифметических инструкций языка ассемблера NASM.

# 2 Задание

1. Изучение теоретического материала по заданной теме
2. Определение существующих арифметических инструкций через ассемблер.
3. Работа с примерами кодов, решающих математические выражения.
4. Создание собственного кода.

# 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. Далее рассмотрены все существующие способы задания адреса хранения операндов – способы адресации.

Существует три основных способа адресации:

• **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.

Поговорим про арифметические операции.

1. Схема команды целочисленного сложения add *(от англ. addition - добавление)* выполняет сложение двух операндов и записывает результат по адресу первого операнда.
2. Команда целочисленного вычитания sub *(от англ. subtraction – вычитание)* работает аналогично команде add.
3. Довольно часто при написании программ встречается операция прибавления или вычитания единицы. Прибавление единицы называется инкрементом, а вычитание — декрементом. Для этих операций существуют специальные команды: inc *(от англ. increment)* и dec *(от англ. decrement)*, которые увеличивают и уменьшают на 1 свой операнд.
4. Команда neg рассматривает свой операнд как число со знаком и меняет знак операнда на противоположный. Операндом может быть регистр или ячейка памяти любого размера.
5. Для беззнакового умножения используется команда mul *(от англ. multiply – умножение)*.Для знакового умножения используется команда imul.
6. Для деления, как и для умножения, существует 2 команды div *(от англ. divide - деление)* и idiv

# 4 Выполнение лабораторной работы

## 4.1 Символьные и численные данные в NASM.

1. Создаю каталог для программам лабораторной работы № 6, перехожу в него и создаю файл lab6-1.asm (рис. 1).

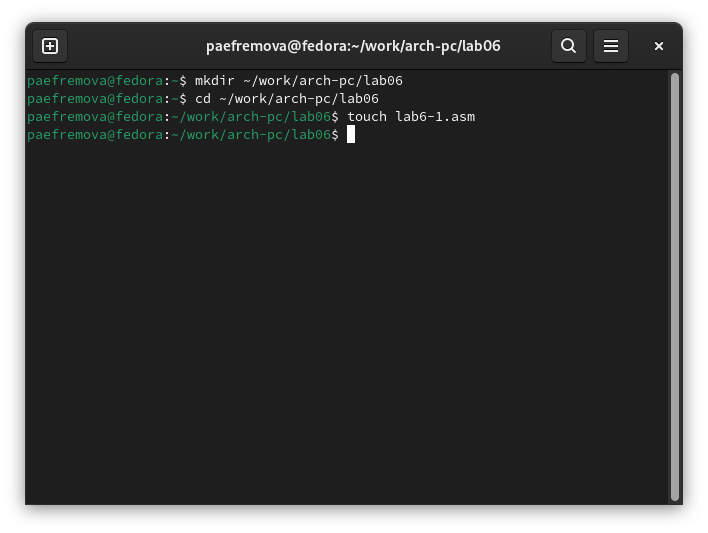


Рис. 1: Создание каталога и файла 1 для работы

1. Ввожу в файл lab6-1.asm текст программы из листинга 6.1. (рис. 2).

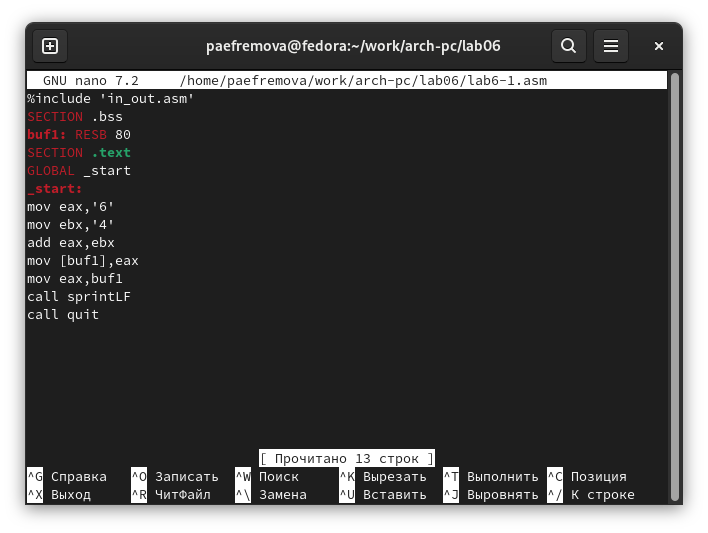


Рис. 2: Ввод программы 1

Ниже прикрепляю текст программы:

%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

1. Создаю исполняемый файл и запускаю его. (рис. 3).

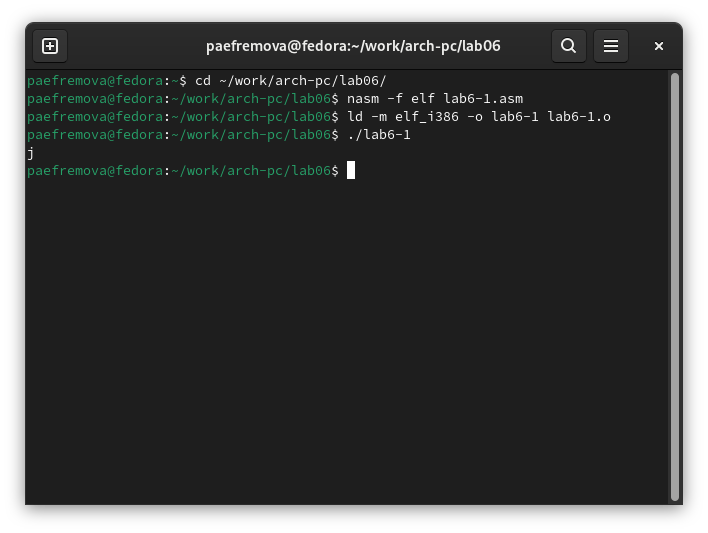


Рис. 3: Запуск файла 1

Вывод: В данном случае при выводе значения регистра eax вместо числа 10 мы видим символ j как разультат программы. Это происходит потому, что код символа 6 равен 00110110 в двоичном представлении, а код символа 4 – 00110100. Команда add eax,ebx запишет в регистр eax сумму кодов – 01101010 (106), что в свою очередь является кодом символа j.

1. Далее изменяю текст программы и вместо символов записываю в регистры числа.(рис. 4).

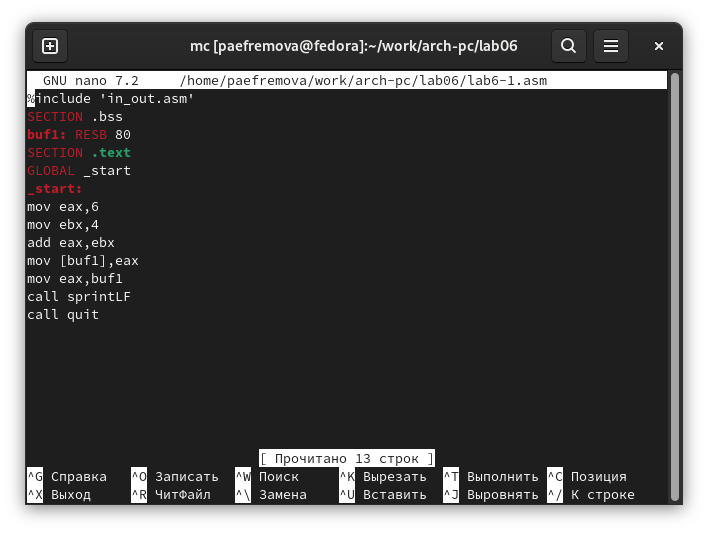


Рис. 4: Изменение файла 1

Текст программы:

%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

1. Создаю исполняемый файл и запускаю его. (рис. 5).

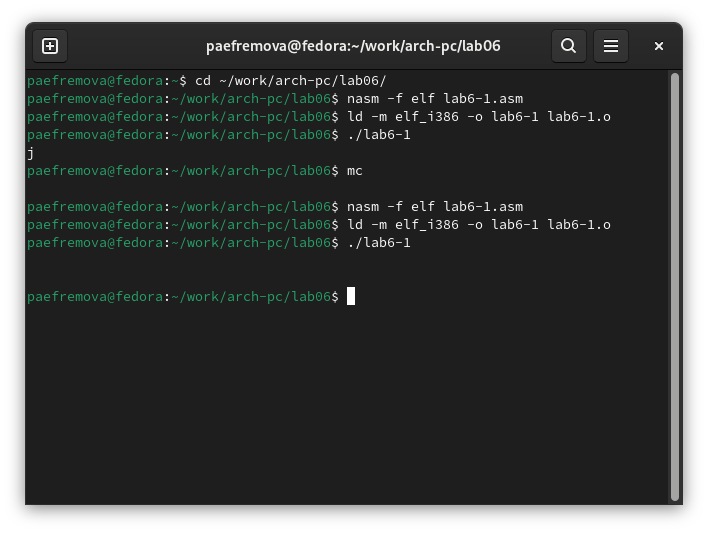


Рис. 5: Запуск файла 1

Вывод: Теперь вывелся символ с кодом 10, это символ перевода строки, этот символ не отображается при выводе на экран.

1. Создаю файл lab6-2.asm. (рис. 6).

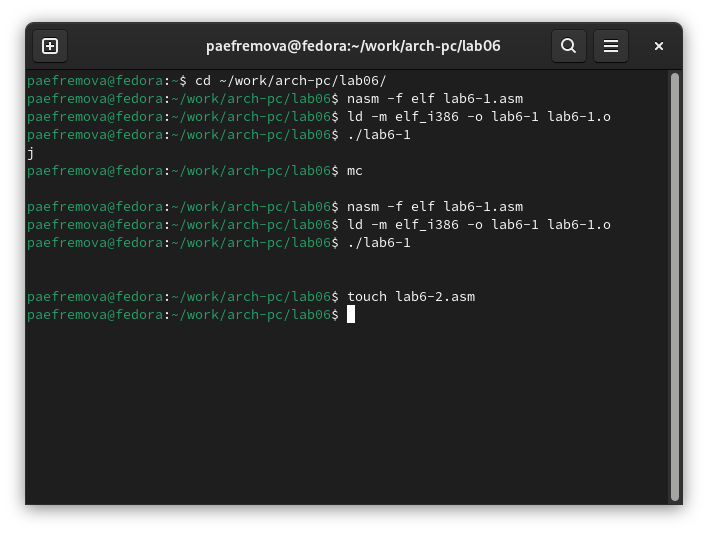


Рис. 6: Создание файла 2

1. Ввожу в файл 2 текст программы из листинга 6.2. (рис. 7).

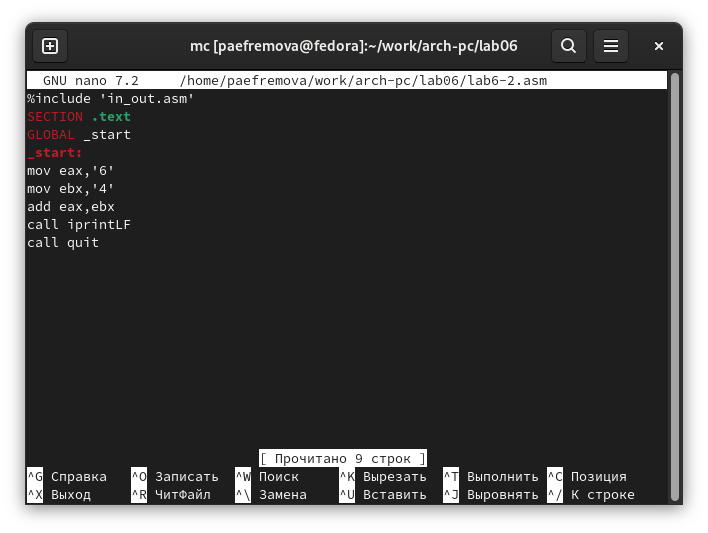


Рис. 7: Ввод программы 2

Текст программы вывода значения регистра eax.

%include 'in\_out.asm'  
SECTION .text  
GLOBAL \_start  
\_start:  
mov eax,'6'  
mov ebx,'4'  
add eax,ebx  
call iprintLF  
call quit

1. Создаю исполняемый файл 2 и запускаю его. (рис. 8).

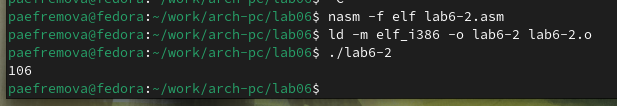


Рис. 8: Запуск файла 2

В результате работы данной программы я получила число 106. Данная программа схожа с предыдущей, только здесь iprintLF позволяет вывести число, а не символ, кодом которого является это число.

1. Аналогично предыдущему примеру изменяю символы на числа. (рис. 9).

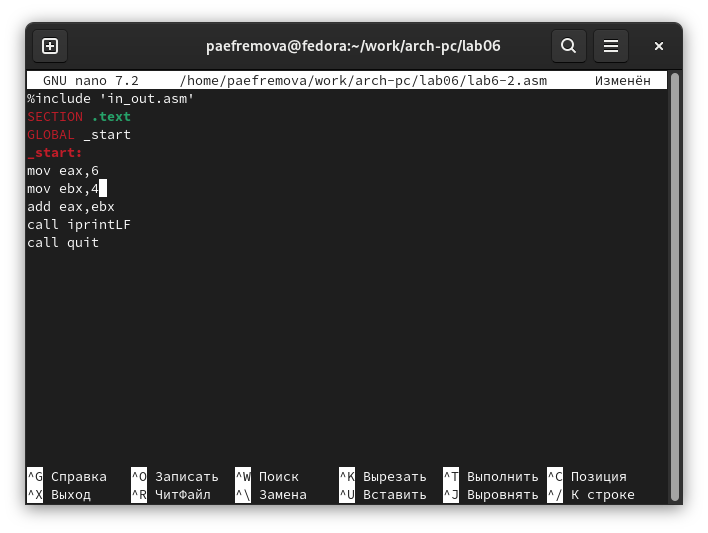


Рис. 9: Изменение файла 2

Текст измененной программы файла 2:

%include 'in\_out.asm'  
SECTION .text  
GLOBAL \_start  
\_start:  
mov eax,6  
mov ebx,4  
add eax,ebx  
call iprintLF  
call quit

1. Создаю исполняемый измененный файл 2 и запускаю его. (рис. 10).

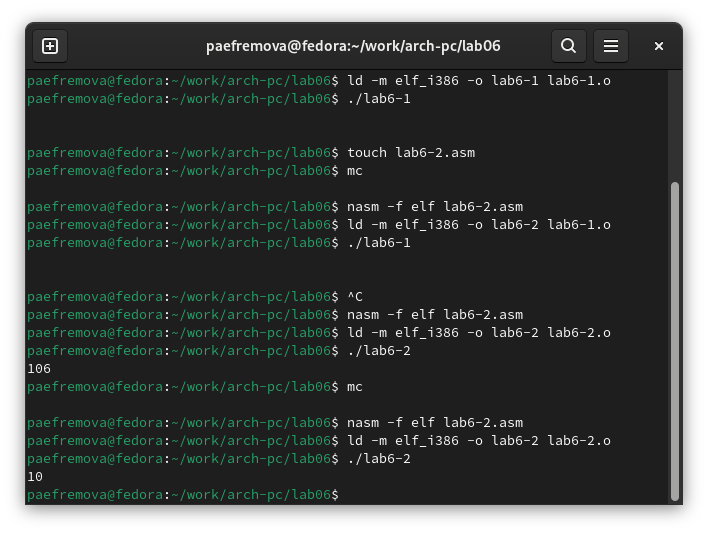


Рис. 10: Запуск измененного файла 2

Вывод: Теперь программа складывает не соответствующие символам коды в системе ASCII, а сами числа, поэтому вывод 10.

## 4.2 Выполнение арифметических операций в NASM

1. Создаю файл lab6-3.asm в каталоге ~/work/arch-pc/lab06 (рис. 11).

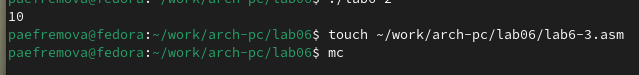


Рис. 11: Создание файла 3

1. Внимательно изучаю текст программы из листинга 6.3 и ввожу в lab6-3.asm (рис. 12).

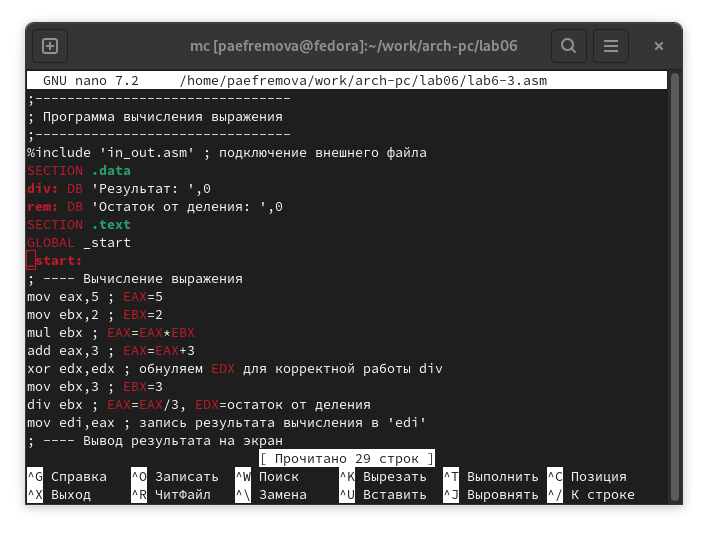


Рис. 12: Ввод программы 3

1. Создаю исполняемый файл 3 и запускаю его. (рис. 13).

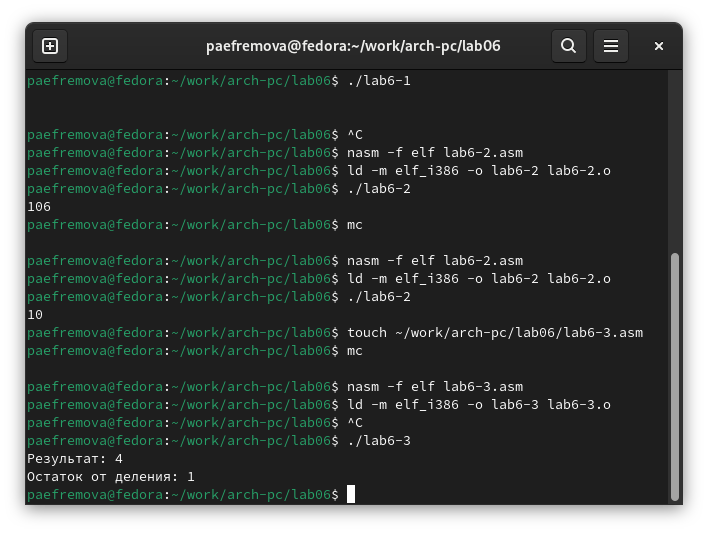


Рис. 13: Запуск файла 3

Текст программы 3: Программа вычисления выражения :

%include 'in\_out.asm'   
SECTION .data  
div: DB 'Результат: ',0  
rem: DB 'Остаток от деления: ',0  
SECTION .text  
GLOBAL \_start  
\_start:  
mov eax,5  
mov ebx,2   
mul ebx   
add eax,3   
xor edx,edx   
mov ebx,3   
div ebx   
mov edi,eax   
mov eax,div   
call sprint   
mov eax,edi   
call iprintLF   
mov eax,rem   
call sprint   
mov eax,edx   
call iprintLF   
call quit

1. Изменяю программу, чтобы она была способна вычислить выражение Программа вычисления выражения (рис. 14).

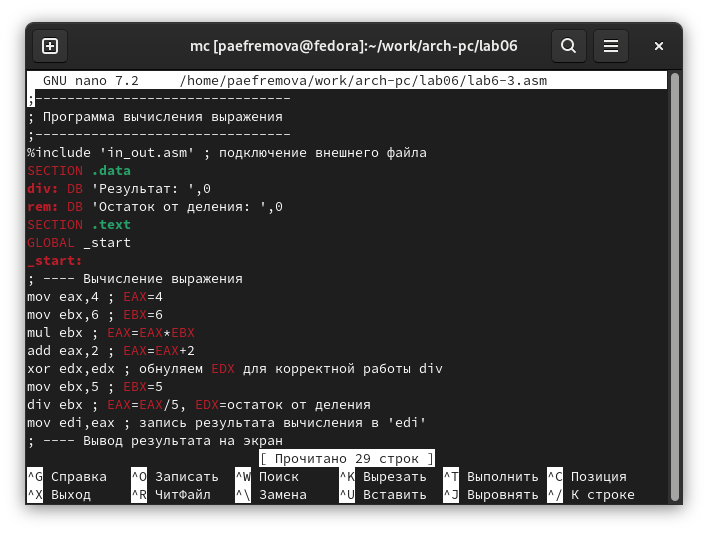


Рис. 14: Изменение файла 3

Текст измененной программы 3: Программа вычисления выражения :

%include 'in\_out.asm'   
SECTION .data  
div: DB 'Результат: ',0  
rem: DB 'Остаток от деления: ',0  
SECTION .text  
GLOBAL \_start  
\_start:  
mov eax,4   
mov ebx,6  
mul ebx   
add eax,2   
xor edx,edx   
mov ebx,5   
div ebx   
mov edi,eax   
mov eax,div   
call sprint   
mov eax,edi   
call iprintLF   
mov eax,rem   
call sprint   
mov eax,edx   
call iprintLF   
call quit

1. Создаю исполняемый измененный файл 3 и запускаю его. (рис. 15).

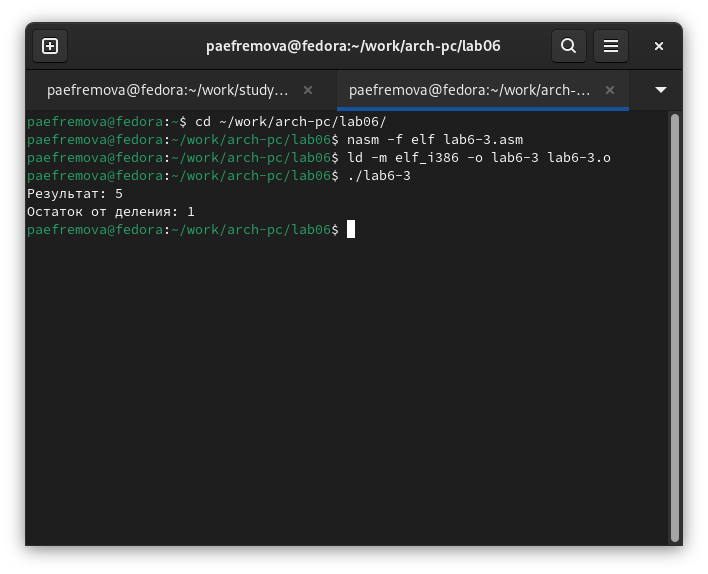


Рис. 15: Запуск измененного файла 3

1. Создаю файл variant.asm в каталоге ~/work/arch-pc/lab06 (рис. 16).

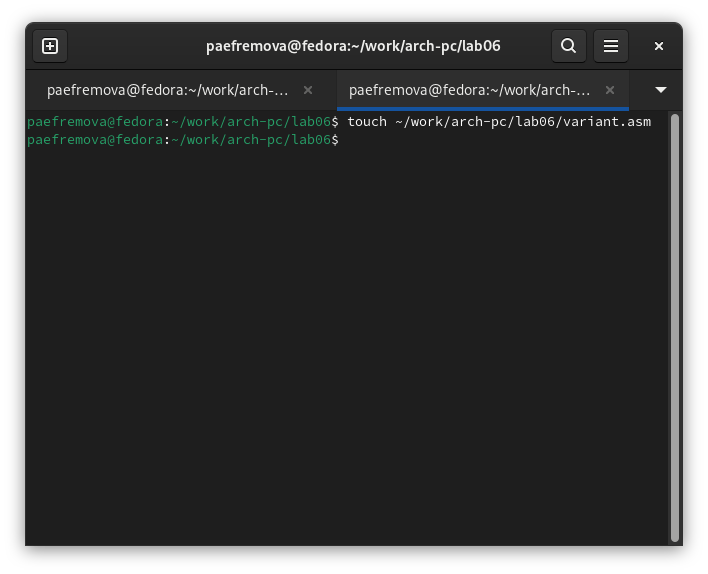


Рис. 16: Создание файла variant.asm

1. Внимательно изучаю текст программы из листинга 6.4 и ввожу в файл variant.asm. (рис. 17).

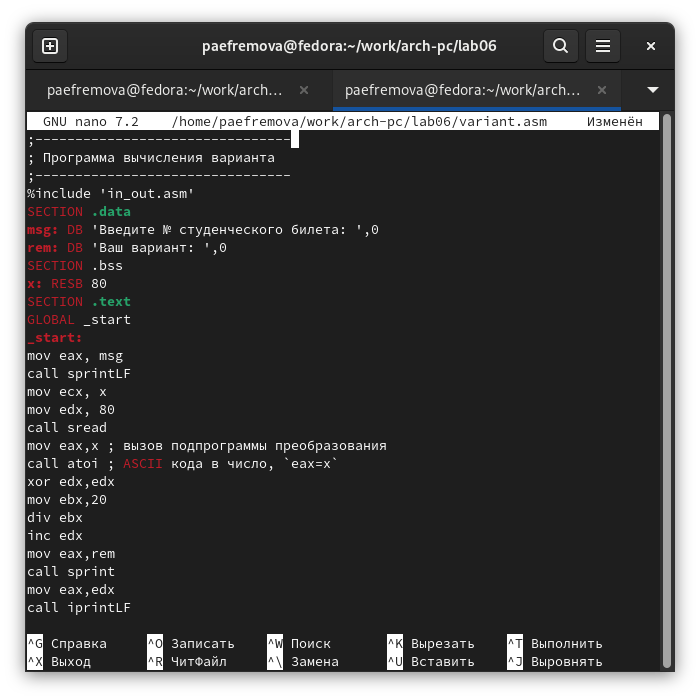


Рис. 17: Ввод программы

1. Создаю исполняемый измененный файл и запускаю его. (рис. 18).

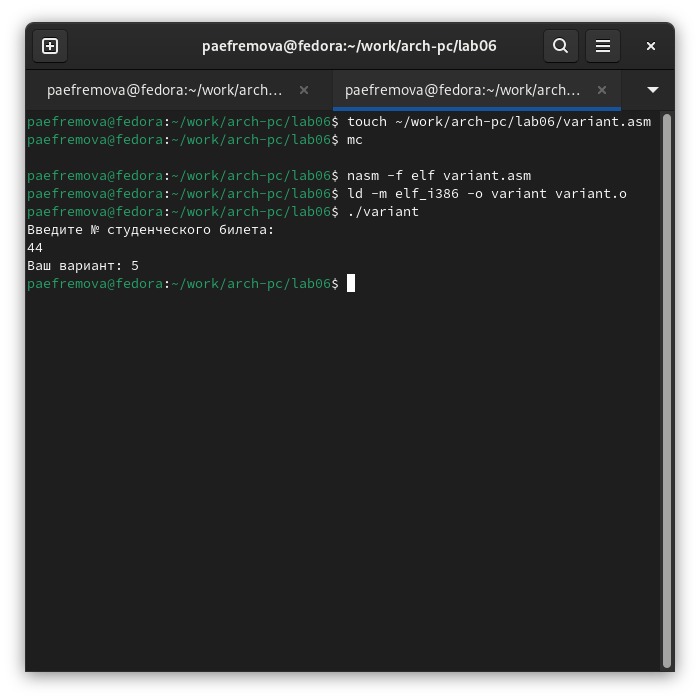


Рис. 18: Запуск файла

Программа показывает, чтов самостоятельной работе я буду писать программу для выражения из варианта 5.

### 4.2.1 Ответы на вопросы по программе

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

mov eax,rem  
call sprint

1. 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
2. call atoi используется для вызова подпрограммы из внешнего файла, которая преобразует ascii-код символа в целое число и записывает результат в регистр eax
3. 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

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

mov eax,edx  
call iprintLF

## 4.3 Выполнение заданий для самостоятельной работы.

1. Создаю файл lab6-4.asm в том же каталоге, что и другие файлы asm. (рис. 19).

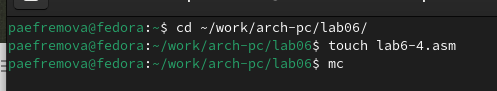


Рис. 19: Создание файла lab6-4.asm

1. Ввожу программу для выражения из варианта 5 (рис. 20).

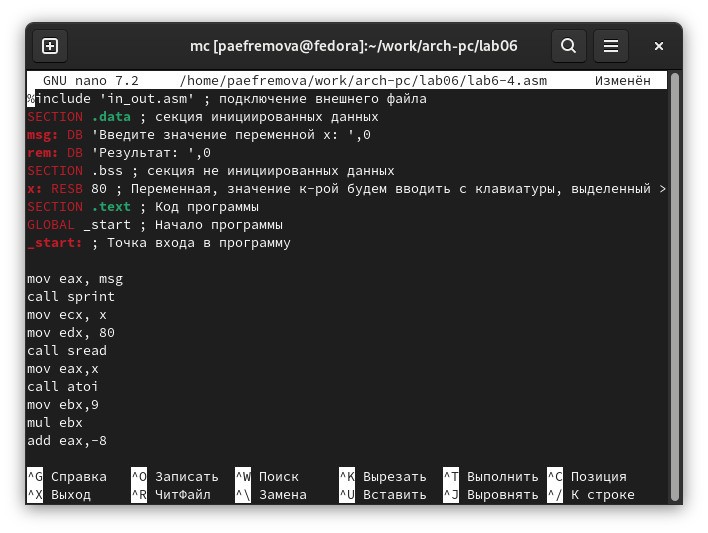


Рис. 20: Ввод программы в файл lab6-4.asm

Текст программы из файла lab6-4.asm:

%include 'in\_out.asm' ; подключение внешнего файла  
SECTION .data ; секция инициированных данных  
msg: DB 'Введите значение переменной х: ',0  
rem: DB 'Результат: ',0  
SECTION .bss ; секция не инициированных данных  
x: RESB 80 ; Переменная, значение к-рой будем вводить с клавиатуры, выделенный >  
SECTION .text ; Код программы  
GLOBAL \_start ; Начало программы  
\_start: ; Точка входа в программу  
  
mov eax, msg  
call sprint  
mov ecx, x  
mov edx, 80  
call sread  
mov eax,x  
call atoi  
mov ebx,9  
mul ebx  
add eax,-8  
mov ebx, 8  
div ebx  
mov edi,eax ; запись результата вычисления в 'edi'  
  
mov eax,rem ; вызов подпрограммы печати  
call sprint ; сообщения 'Результат: '  
mov eax,edi ; вызов подпрограммы печати значения  
call iprint ; из 'edi' в виде символов  
call quit ; вызов подпрограммы завершения

1. Создаю исполняемый измененный файл и запускаю его, проверяю его работу для значений x1 и x2 из 6.3. (рис. 21).

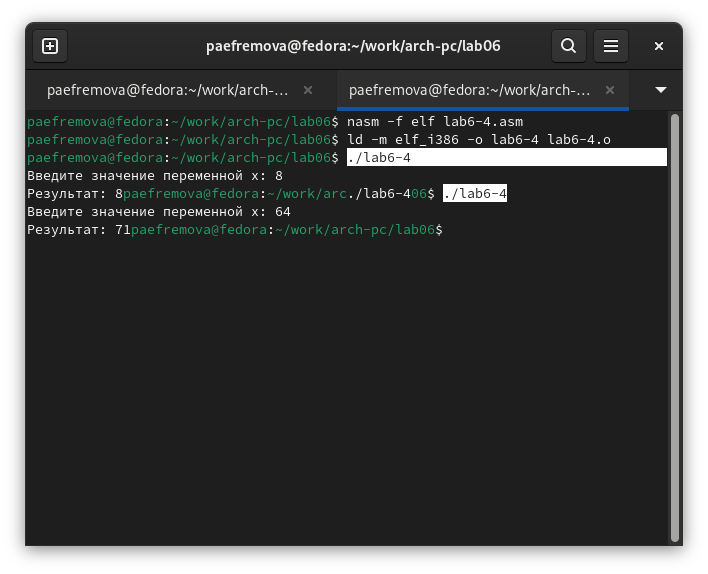


Рис. 21: Запуск файла lab6-4.asm

# 5 Выводы

При выполнении данной лабораторной работы я освоила арифметические инструкции языка ассемблера NASM. Я увидела разницу между символьными и численными данными в NASM, научилась выполнять арифметические операции, а также сама смогла написать программу нахождения значения выражения.

# Список литературы

1. [Архитектура ЭВМ](https://github.com/evdvorkina/study_2022-2023_arh-pc/blob/master/labs/lab07/report/Л07_Дворкина_отчет.md?plain=1)
2. [Синтаксис Markdown: подробная шпаргалка для веб-разработчиков / Skillbox Media](https://skillbox.ru/media/code/yazyk-razmetki-markdown-shpargalka-po-sintaksisu-s-primerami/)
3. [Руководство по NASM](https://www.opennet.ru/docs/RUS/nasm/nasm_ru3.html)