Отчёт по лабораторной работе №6

дисциплина: Архитектура компьютера

Баранова Анна Андреевна

Содержание

# 1 Цель работы

Освоить арифметические инструкции языка ассемблера NASM.

# 2 Задание

В ходе выполнения данной лабораторной работы необходимо изучить:

* Символьные и численные данные в NASM;
* как выполненять арифметические операции в NASM;
* адресация в NASM.

Выполнив эту работу, мы освоим арифметические инструкции языка ассемблера NASM.

# 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.

Схема команды целочисленного сложения add (от англ. addition - добавление) выполняет сложение двух операндов и записывает результат по адресу первого операнда. Команда add работает как с числами со знаком, так и без знака.

Команда целочисленного вычитания sub (от англ. subtraction – вычитание) работает аналогично команде add.

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.

Еще одна команда, которую можно отнести к арифметическим командам это команда изменения знака neg. Команда neg рассматривает свой операнд как число со знаком и меняет знак операнда на противоположный. Операндом может быть регистр или ячейка памяти любого размера.

Умножение и деление, в отличии от сложения и вычитания, для знаковых и беззнаковых чисел производиться по-разному, поэтому существуют различные команды. Для беззнакового умножения используется команда mul (от англ. multiply – умножение). Для знакового умножения используется команда imul.

Для деления, как и для умножения, существует 2 команды div (от англ. divide - деление) и idiv.

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.

Расширенная таблица ASCII состоит из двух частей. Первая (символы с кодами 0-127) является универсальной (см. Приложение.), а вторая (коды 128-255) предназначена для специальных символов и букв национальных алфавитов и на компьютерах разных типов может меняться.

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.

Для выполнения лабораторных работ в файле in\_out.asm реализованы подпрограммы для преобразования ASCII символов в числа и обратно.

# 4 Выполнение лабораторной работы

## 4.1 Символьные и численные данные в NASM

Создадим каталог для программ лабораторной работы № 6, перейдём в него и создадим файл lab6-1.asm (рис. 1).

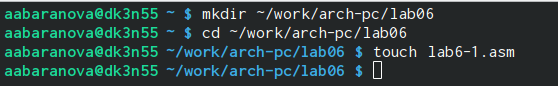


Рис. 1: Создание каталога для программ лабораторной работы № 6, переход в него и создание файла lab6-1.asm

Введём в файл lab6-1.asm текст программы (рис. 2).

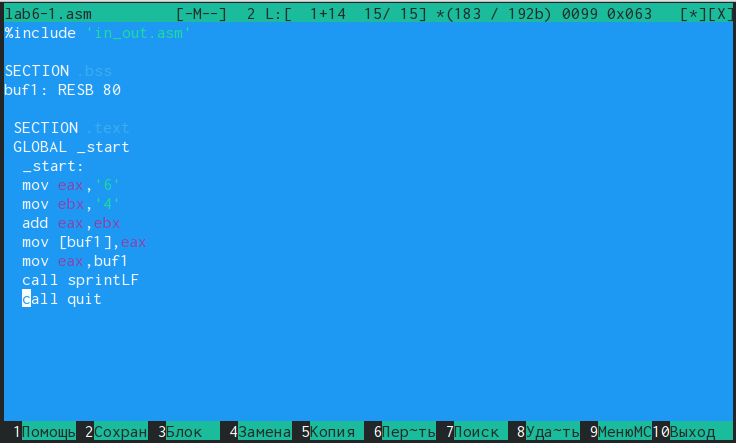


Рис. 2: Изменения в файле lab6-1.asm

Перед созданием исполняемого файла создадим копию файла in\_out.asm в каталоге ~/work/arch-pc/lab06. (рис. 3), (рис. 4).

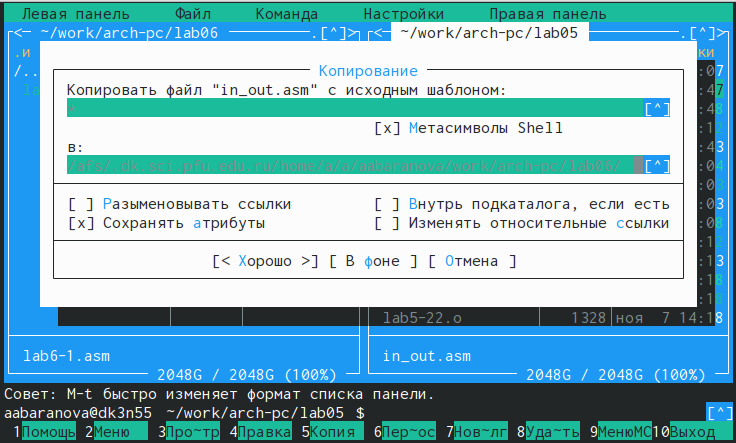


Рис. 3: Копирование файла in\_out.asm в каталог с файлом lab6-1.asm с помощью функциональной клавиши F5

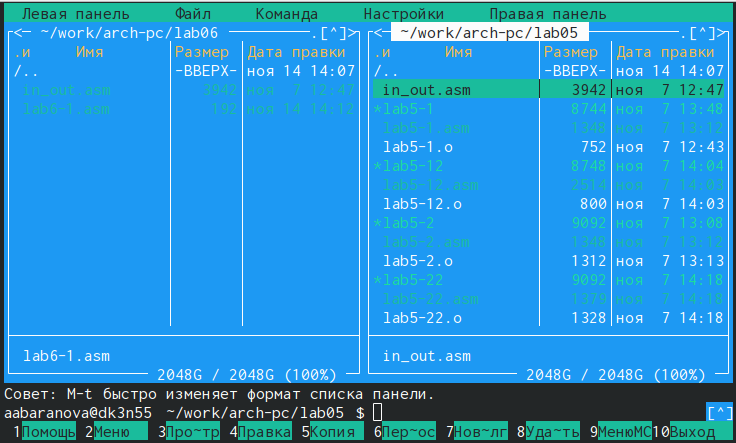


Рис. 4: Копирование файла in\_out.asm в каталог с файлом lab6-1.asm с помощью функциональной клавиши F5

Создадим исполняемый файл и запустим его (рис. 5), (рис. 6).

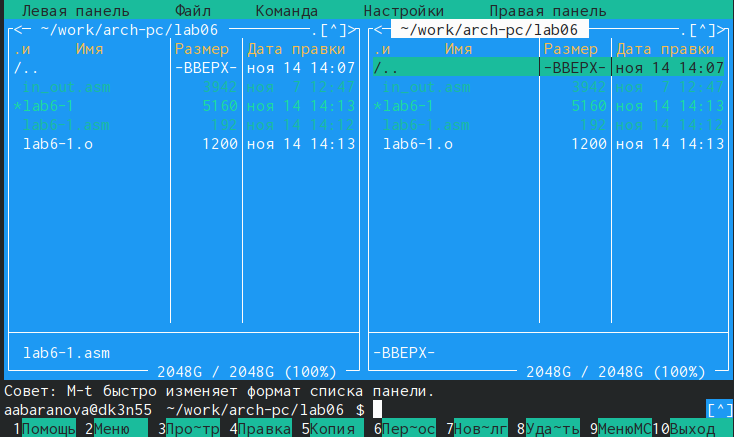


Рис. 5: Создание исполняемого файла и его запуск

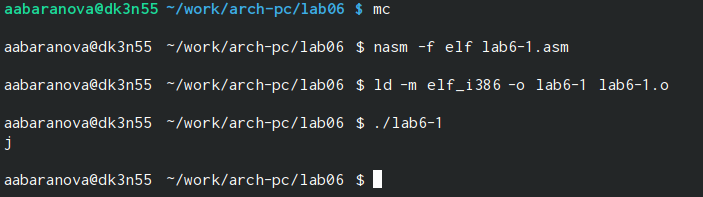


Рис. 6: Создание исполняемого файла и его запуск

Изменим текст программы и вместо символов, запишем в регистры числа (рис. 7).

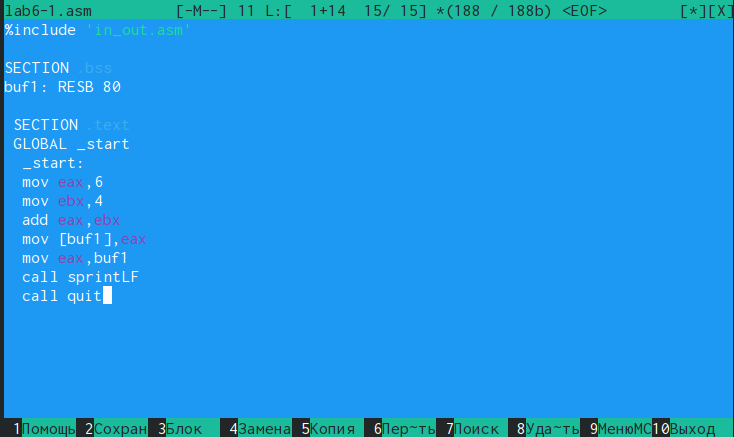


Рис. 7: Изменения в файле lab6-1.asm

Создадим исполняемый файл и запустим его (рис. 8).

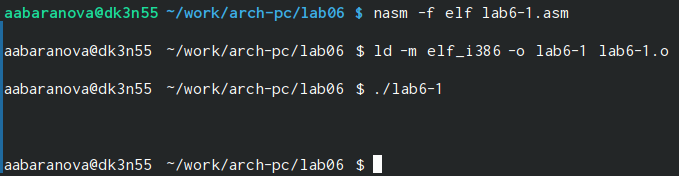


Рис. 8: Создание исполняемого файла и его запуск

Создадим файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 и введём в него текст программы (рис. 9), (рис. 10), (рис. 11).

Рис. 9: Создание файла lab6-2.asm

Рис. 9: Создание файла lab6-2.asm

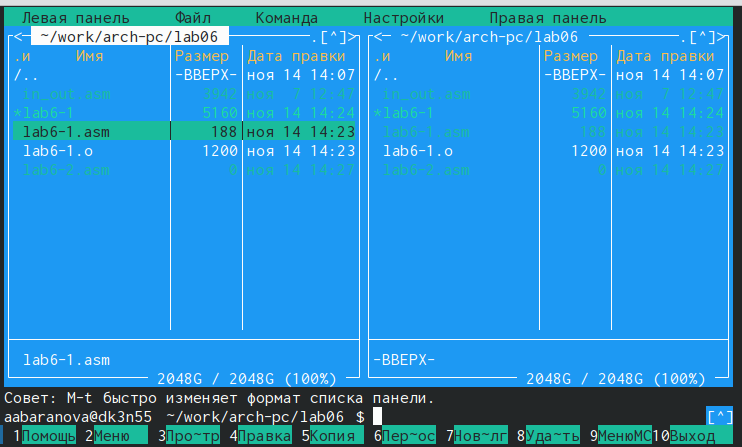


Рис. 10: Создание файла lab6-2.asm

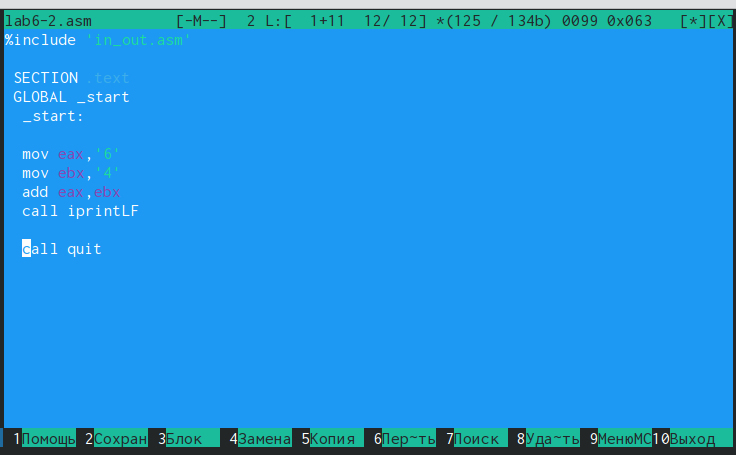


Рис. 11: Изменения в файле lab6-2.asm

Создадим исполняемый файл и запустим его (рис. 12), (рис. 13).

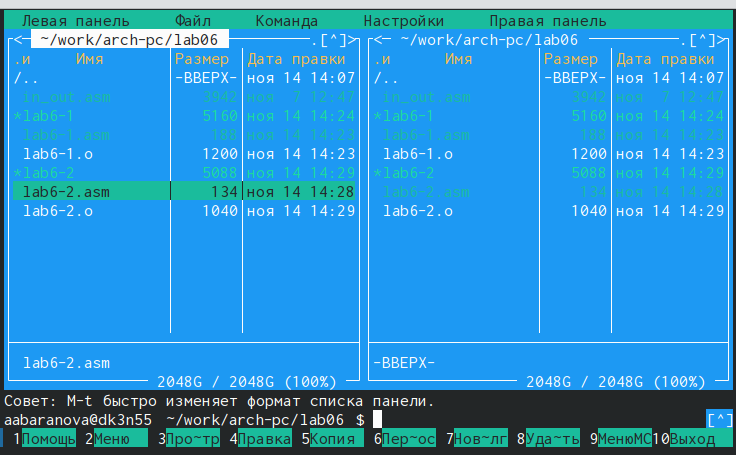


Рис. 12: Создание исполняемого файла и его запуск

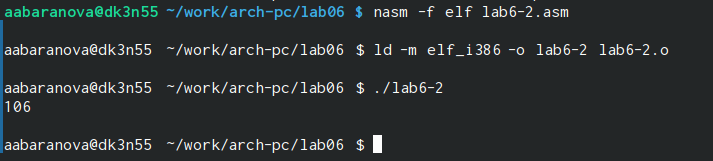


Рис. 13: Создание исполняемого файла и его запуск

Изменим текст программы и вместо символов, запишем в регистры числа (рис. 14).

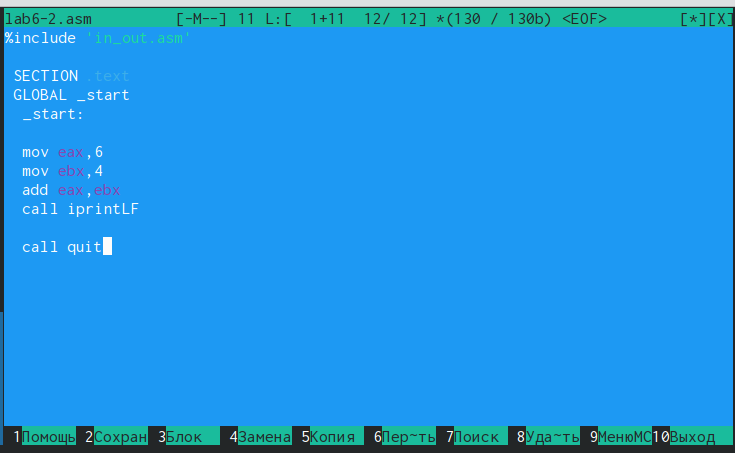


Рис. 14: Изменения в файле lab6-2.asm

Создадим исполняемый файл и запустим его (рис. 15).

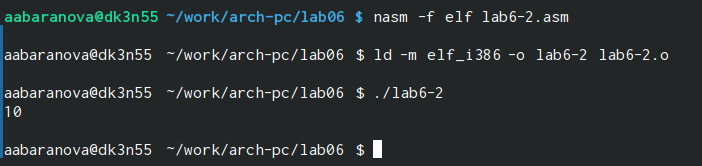


Рис. 15: Создание исполняемого файла и его запуск

Заменим функцию iprintLF на iprint. Создадим исполняемый файл и запустим его (рис. 16), (рис. 17).

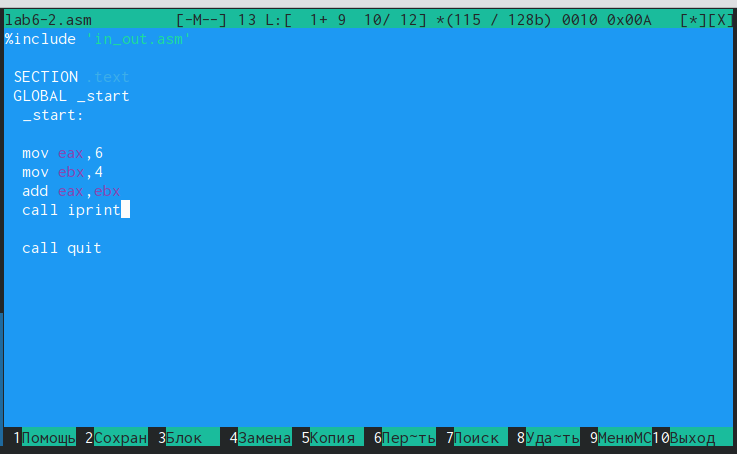


Рис. 16: Изменения в файле lab6-2.asm

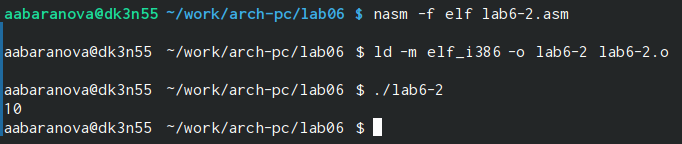


Рис. 17: Создание исполняемого файла и его запуск

## 4.2 Выполнение арифметических операций в NASM

Создадим файл lab6-3.asm в каталоге ~/work/arch-pc/lab06 (рис. 18), (рис. 19).

Рис. 18: Создание файла lab6-3.asm

Рис. 18: Создание файла lab6-3.asm

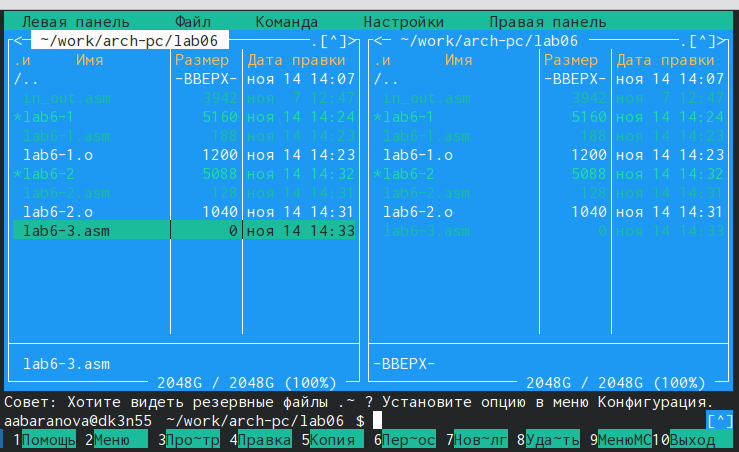


Рис. 19: Создание файла lab6-3.asm

Введём в файл lab6-3.asm текст программы (рис. 20).

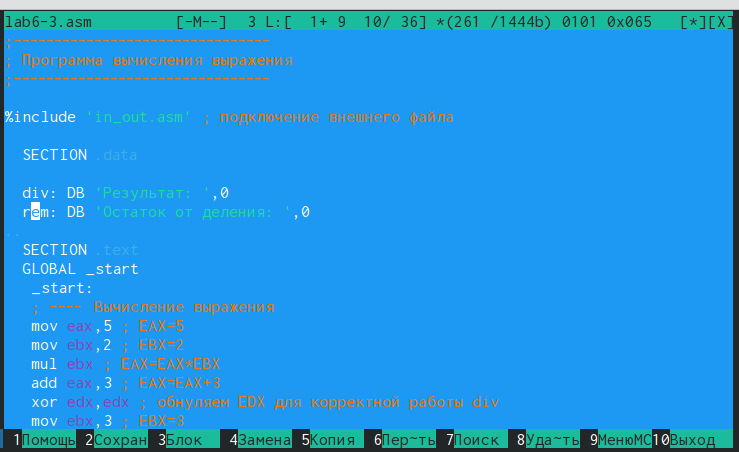


Рис. 20: Изменения в файле lab6-3.asm

Создадим исполняемый файл и запустим его (рис. 21), (рис. 22).

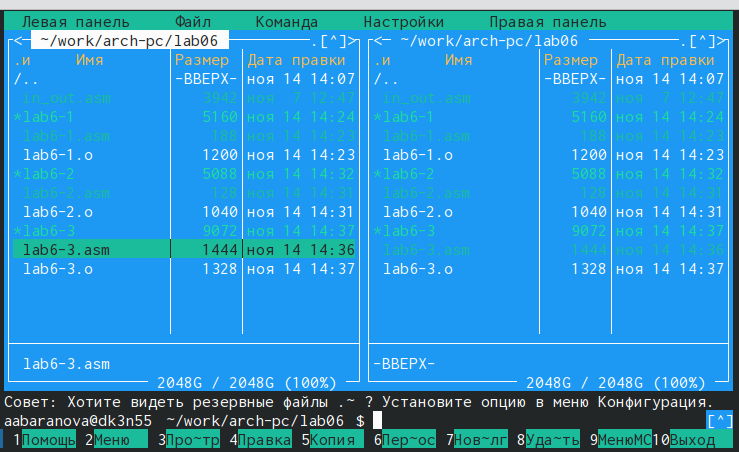


Рис. 21: Создание исполняемого файла и его запуск

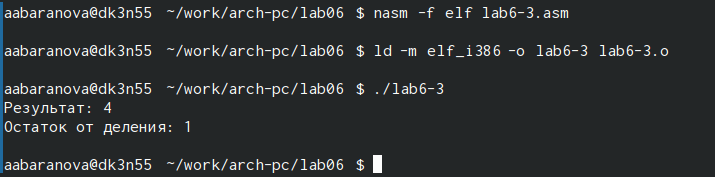


Рис. 22: Создание исполняемого файла и его запуск

Изменим текст программы для вычисления выражения 𝑓(𝑥) = (4 ∗ 6 + 2)/5 (рис. 23).

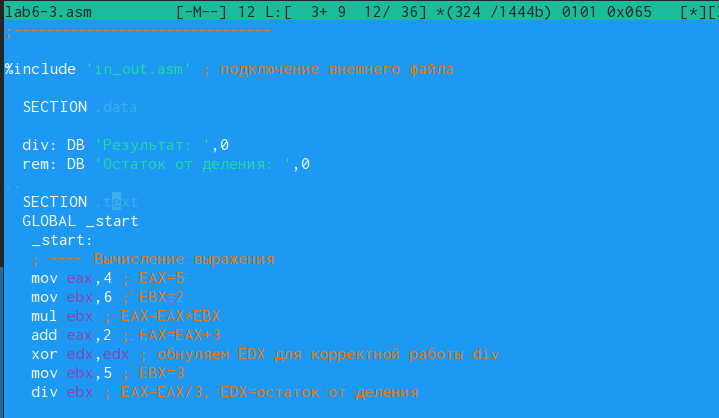


Рис. 23: Изменения в файле lab6-3.asm

Создадим исполняемый файл и запустим его (рис. 24).

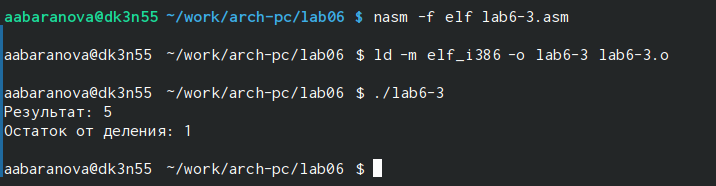


Рис. 24: Создание исполняемого файла и его запуск

Создадим файл variant.asm в каталоге ~/work/arch-pc/lab06 (рис. 25), (рис. 26).

Рис. 25: Создание файла variant.asm

Рис. 25: Создание файла variant.asm

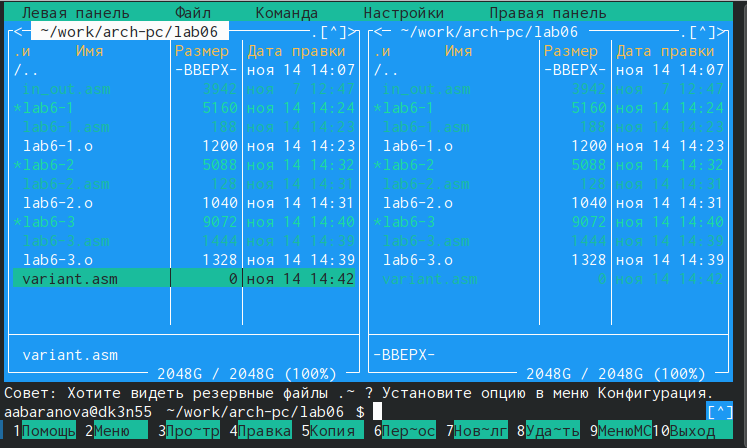


Рис. 26: Создание файла variant.asm

Введём в файл lab6-3.asm текст программы (рис. 27).

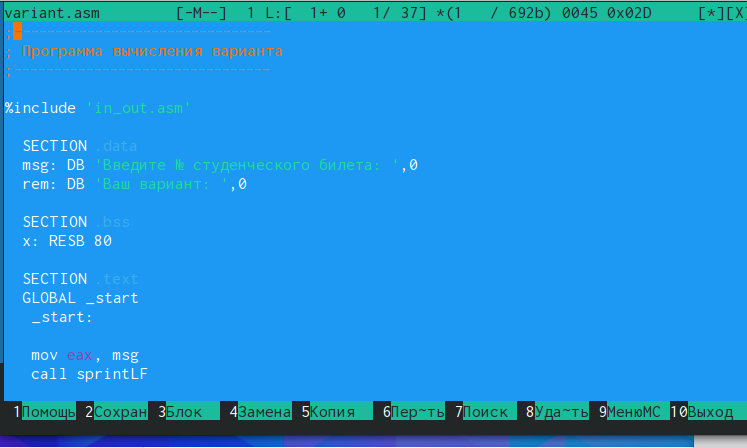


Рис. 27: Изменения в файле variant.asm

Создадим исполняемый файл и запустим его (рис. 28), (рис. 29).

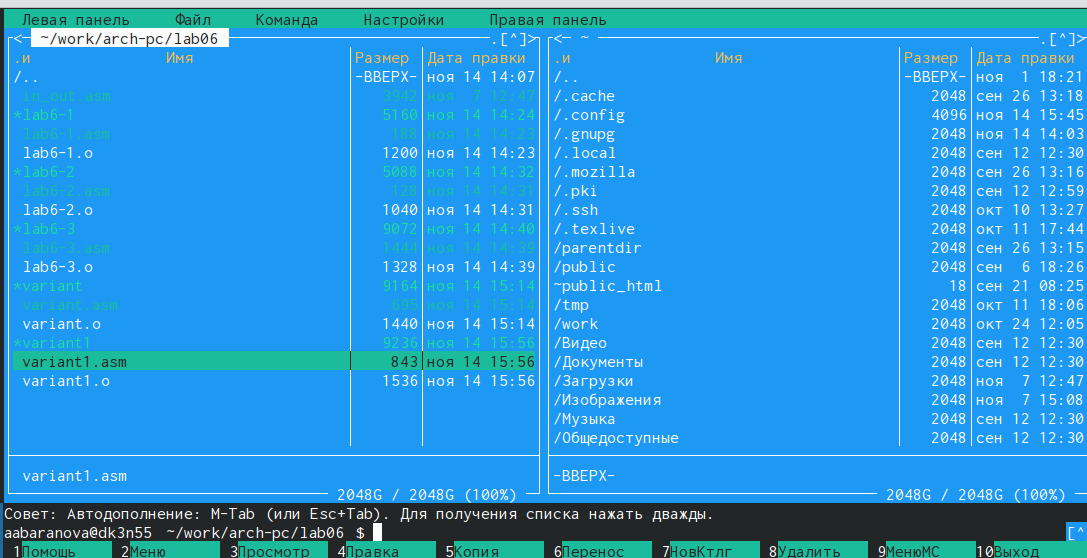


Рис. 28: Создание исполняемого файла и его запуск

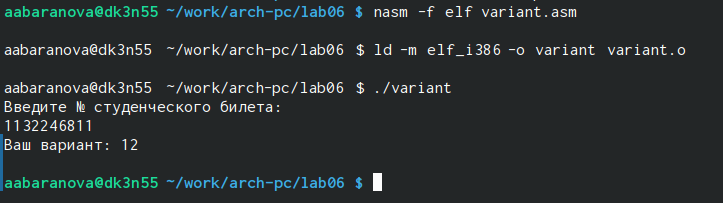


Рис. 29: Создание исполняемого файла и его запуск

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 Задание для самостоятельной работы

Напишем программу вычисления выражения 𝑦 = 𝑓(𝑥), которая должна выводить выражение для вычисления, выводить запрос на ввод значения 𝑥, вычислять заданное выражение в зависимости от введенного 𝑥, выводить результат вычислений (рис. 30), (рис. 31).



Рис. 30: Написание программы вычисления выражения 𝑦 = 𝑓(𝑥

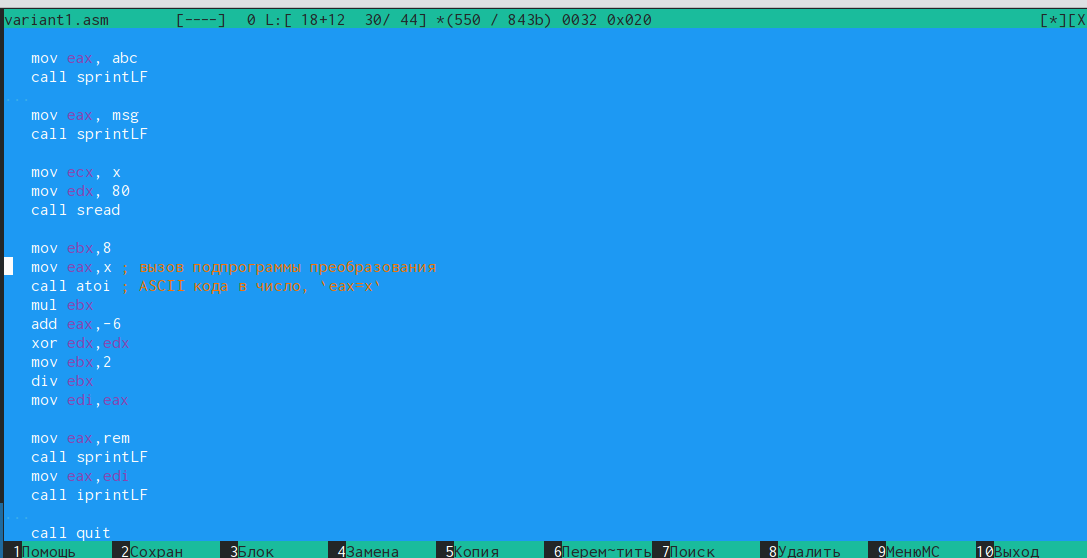


Рис. 31: Написание программы вычисления выражения 𝑦 = 𝑓(𝑥

Создадим исполняемый файл и проверим его работу для значений 𝑥1 и 𝑥2 (рис. 32), (рис. 33).

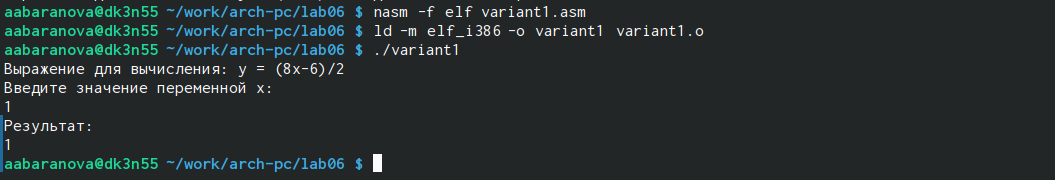


Рис. 32: Создание исполняемого файла и проверка его работы

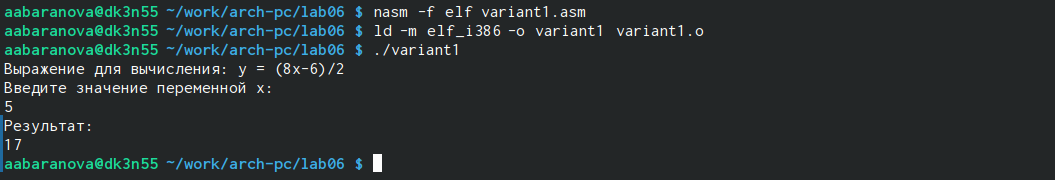


Рис. 33: Создание исполняемого файла и проверка его работы

# 5 Выводы

В ходе выполнения данной лабораторной работы были освоены арифметические инструкции языка ассемблера NASM.