Лабораторная работа №6

Арифметические операции в NASM.

Дагделен Зейнап Реджеповна

Содержание

# 1 Цель 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

## 3.1 Адресация в NASM

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 ко- манде, Например: mov ax,2. - *Адресация памяти* – операнд задает адрес в памяти. В команде указывается символи- ческое обозначение ячейки памяти, над содержимым которой требуется выполнить операцию.

## 3.2 Арифметические операции в NASM

### 3.2.1 Целочисленное сложение add.

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 и выглядит следующим образом:  
add ,   
Допустимые сочетания операндов для команды add аналогичны сочетаниям операндов для команды mov.

### 3.2.2 Целочисленное вычитание sub.

Команда целочисленного вычитания sub (от англ. subtraction – вычитание) работает аналогично команде add и выглядит следующим образом:  
sub ,

### 3.2.3 Команды инкремента и декремента.

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.  
Эти команды содержат один операнд и имеет следующий вид:  
inc   
dec   
Операндом может быть регистр или ячейка памяти любого размера. Команды инкремента и декремента выгодны тем, что они занимают меньше места, чем соответствующие команды сложения и вычитания.

### 3.2.4 Команда изменения знака операнда neg.

Еще одна команда, которую можно отнести к арифметическим командам это команда изменения знака neg:  
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):  
mul   
Для знакового умножения используется команда imul:  
imul   
Для команд умножения один из сомножителей указывается в команде и должен находиться в регистре или в памяти, но не может быть непосредственным операндом. Второй сомножитель в команде явно не указывается и должен находиться в регистре EAX,AX или AL, а результат помещается в регистры EDX:EAX, DX:AX или AX, в зависимости от размера операнда.

### 3.2.5 Команды деления div и idiv.

Для деления, как и для умножения, существует 2 команды div (от англ. divide - деление) и idiv:  
div ; Беззнаковое деление  
idiv ; Знаковое деление  
В командах указывается только один операнд – делитель, который может быть регистром или ячейкой памяти, но не может быть непосредственным операндом. Местоположение делимого и результата для команд деления зависит от размера делителя. Кроме того, так как в результате деления получается два числа – частное и остаток, то эти числа помещаются в определённые регистры.

## 3.3 Перевод символа числа в десятичную символьную запись

Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации производится согласно кодовой таблице символов ASCII. Согласно стандарту ASCII каждый символ кодируется одним байтом.

Среди инструкций NASM нет такой, которая выводит числа (не в символьном виде). Для решения этой проблемы необходимо проводить преобразование ASCII символов в числа и обратно.

# 4 Выполнение лабораторной работы

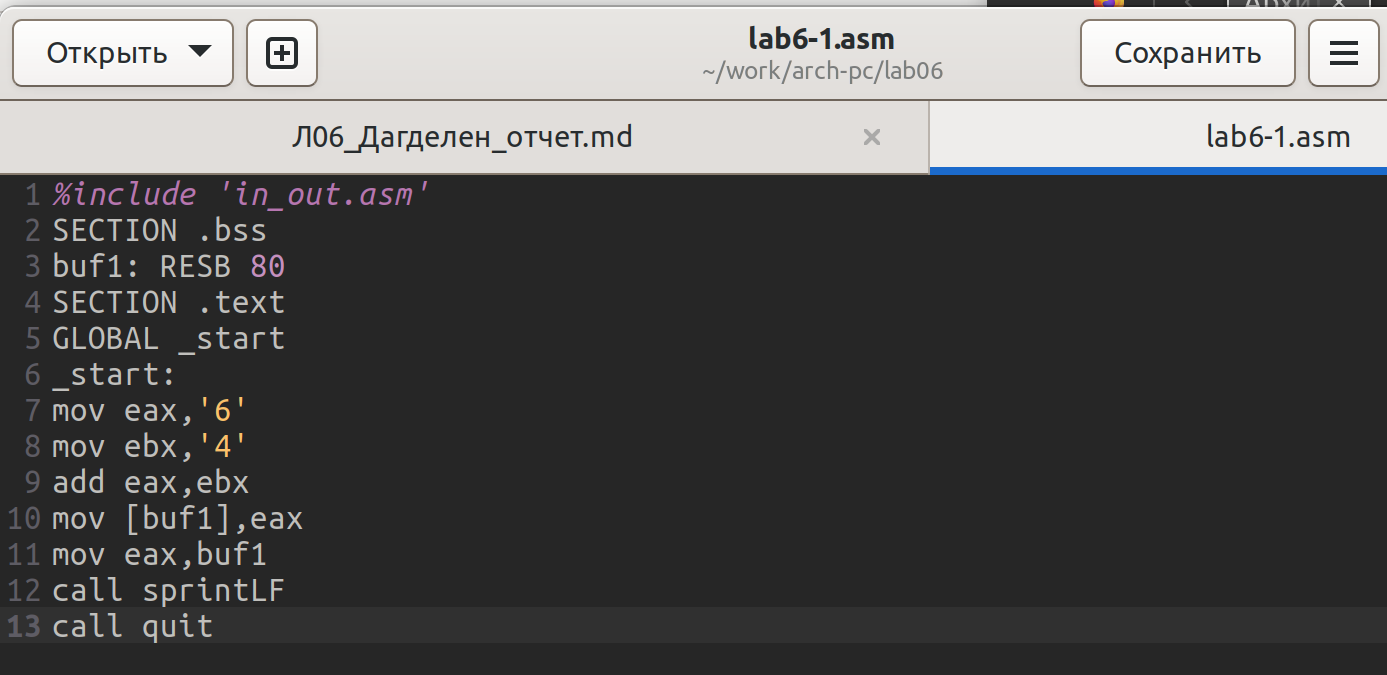
## 4.1 Символьные и численные данные в NASM

1. Создаю каталог для программам лабораторной работы № 6 с помощью команды mkdir, и, перейдя в него, создаю файл lab6-1.asm с помощью touch (рис. [??]).



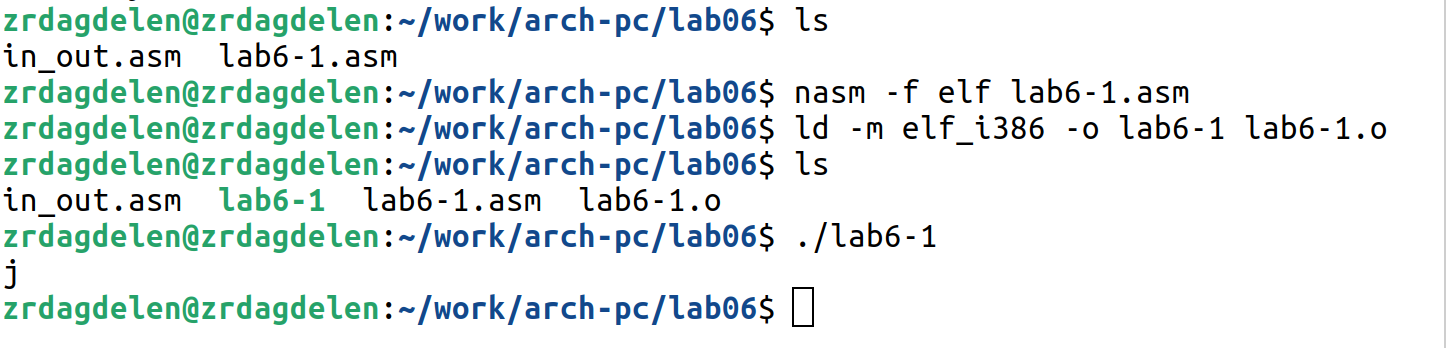
Создание каталога и файла в нем

1. Рассмотрю примеры программ вывода символьных и численных значений. Программы будут выводить значения записанные в регистр eax.  
   Ввожу в файл lab6-1.asm текст программы из листинга 6.1 (рис. [??]).



Текст программы

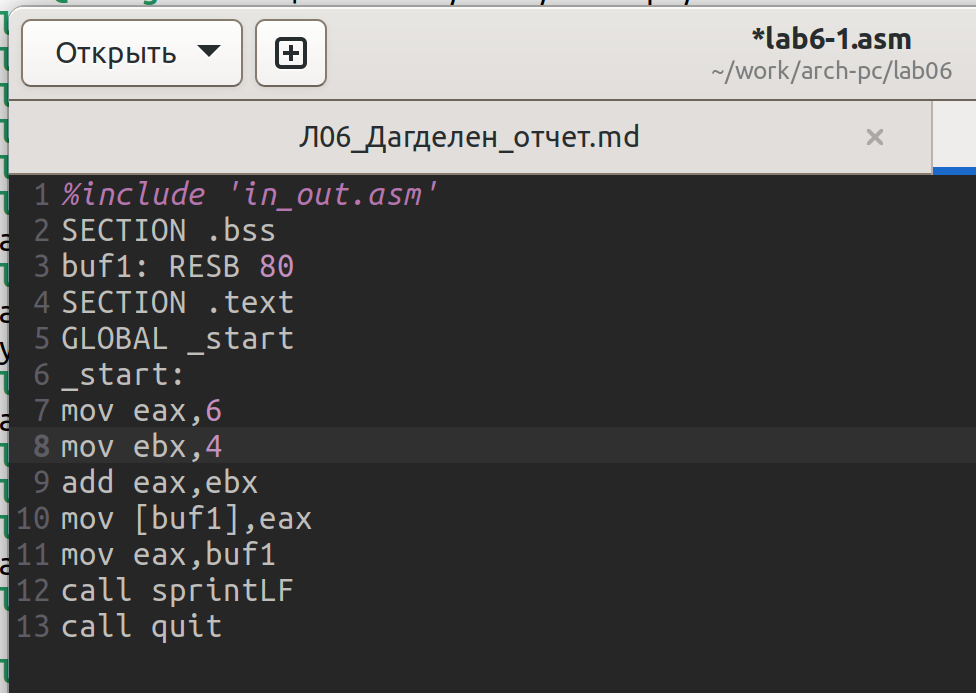
Создаю исполняемый файл и запускаю его(рис. [??]).



Создание исполняемого файла и его запуск

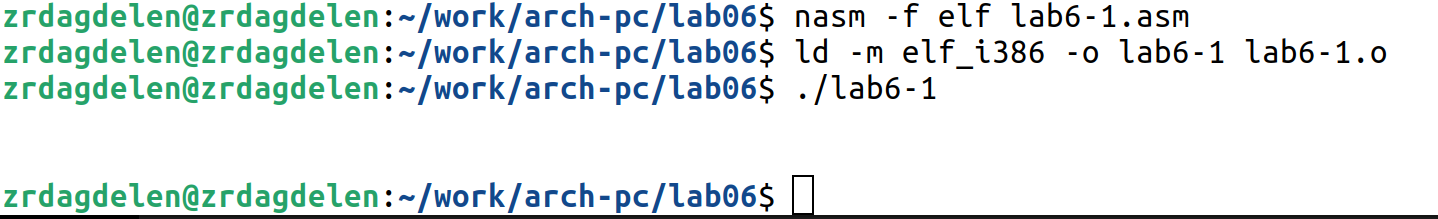
**Вывод программы:** символ j, потому что программа вывела символ, соответствующий по системе ASCII сумме двоичных кодов символов 4 и 6.

1. Далее измяю текст программы и вместо символов, записываю в регистры числа. Ис- правляю текст программы (Листинг 6.1) следующим образом (рис. [??]).



Текст программы

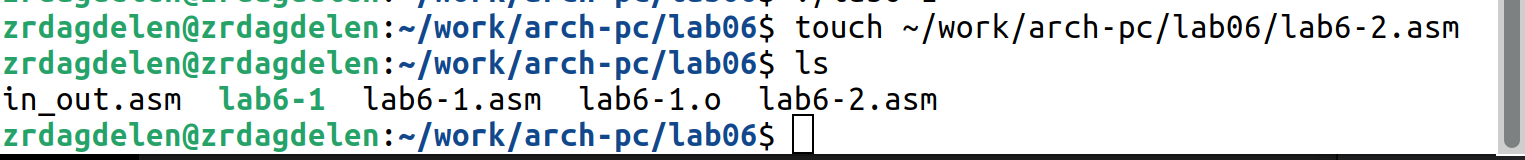
Создаю исполняемый файл и запускаю его(рис. [??]).



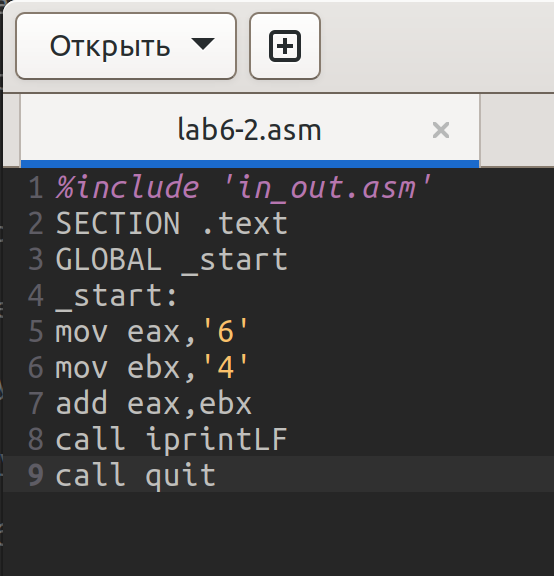
Создание исполняемого файла и его запуск

Как и в предыдущем случае при исполнении программы я не получила число 10. В данном случае выводится символ с кодом 10. Это символ перевода строки, к сожалению, он не отображается при выводе на экран.

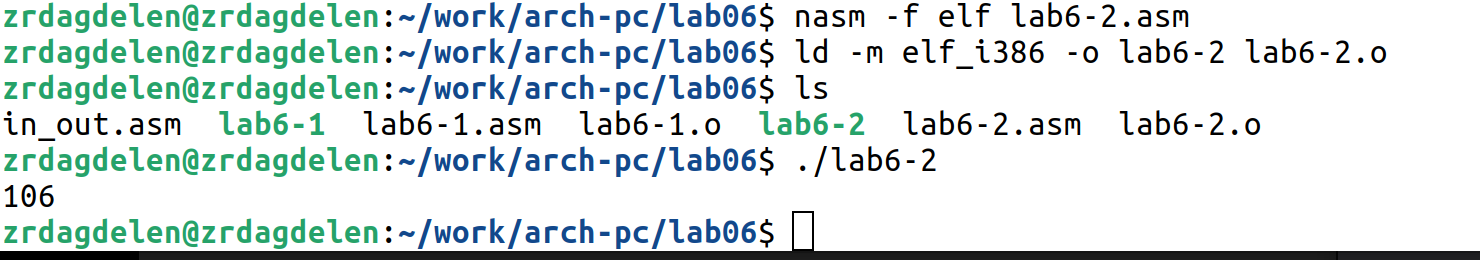
1. Создаю файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 с помощью touch и ввожу в него текст программы из листинга 6.2(рис. [??]- [??]). Создаю исполняемый файл и запускаю его (рис. [??]).



Создание файла lab-2.asm



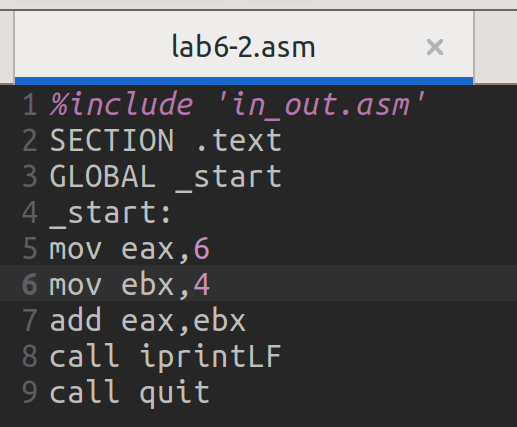
Текст программы



Создание исполняемого файла и его запуск

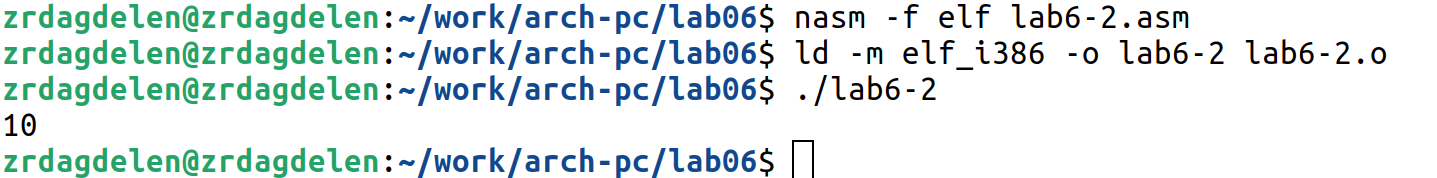
В результате работы программы я получила число 106. В данном случае, как и в первом, команда add складывает коды символов ‘6’ и ‘4’ (54+52=106). Однако, в отличии от программы из листинга 6.1, функция iprintLF позволяет вывести число, а не символ, кодом которого является это число.

1. Аналогично предыдущему примеру изменю символы на числа. Заменяю строки (рис. [??]).



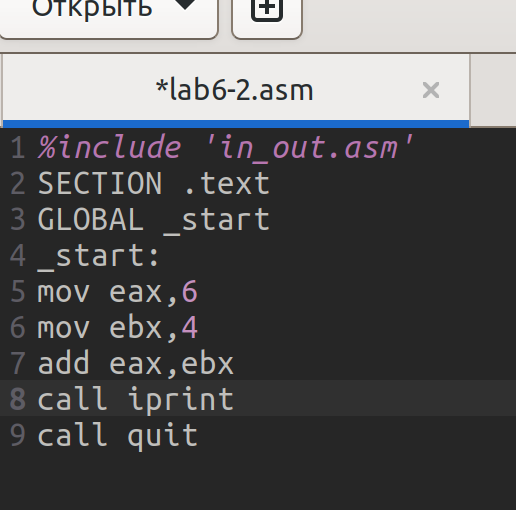
Текст программы

Создаю исполняемый файл и запускаю его(рис. [??]).



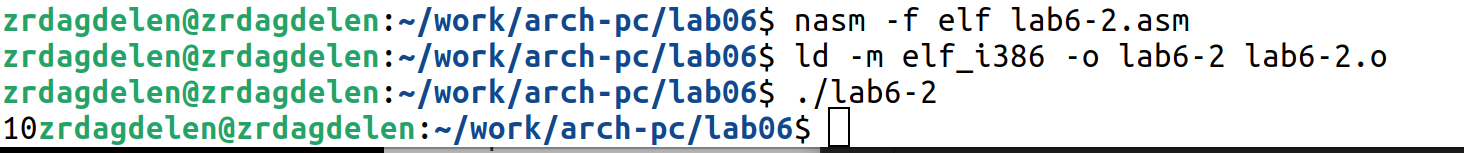
Создание исполняемого файла и его запуск

Теперь программа складывает не коды, соответствующие символам в системе ASCII, а сами числа, поэтому вывод 10. Заменяю функцию iprintLF на iprint(рис. [??]).



Текст программы после изменений

Создаю исполняемый файл и запускаю его(рис. [??]).

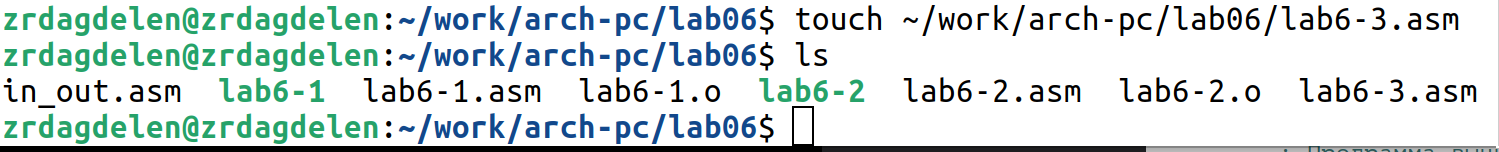


Создание исполняемого файла и его запуск

Чем отличается вывод функций iprintLF и iprint? Вывод не изменился, потому что символ переноса строки не отображался, когда программа исполнялась с функцией iprintLF, а iprint не добавляет к выводу символ переноса строки, в отличие от iprintLF.

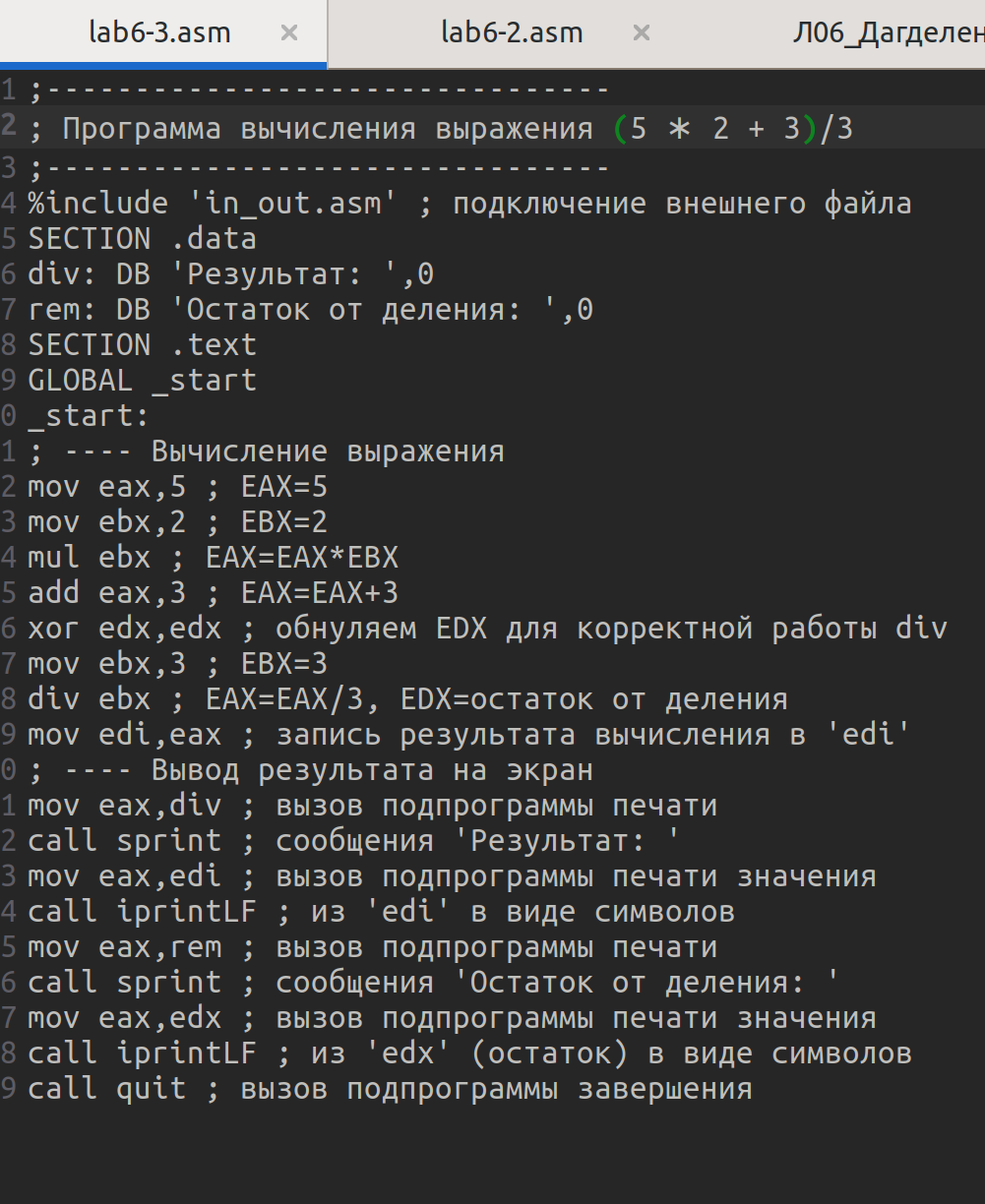
## 4.2 Выполнение арифметических операций в NASM

1. В качестве примера выполнения арифметических операций в NASM приведем про- грамму вычисления арифметического выражения f(x) = (5 ∗ 2 + 3)/3. Создаю файл lab6-3.asm в каталоге ~/work/arch-pc/lab06 с помощью touch (рис. [??]).



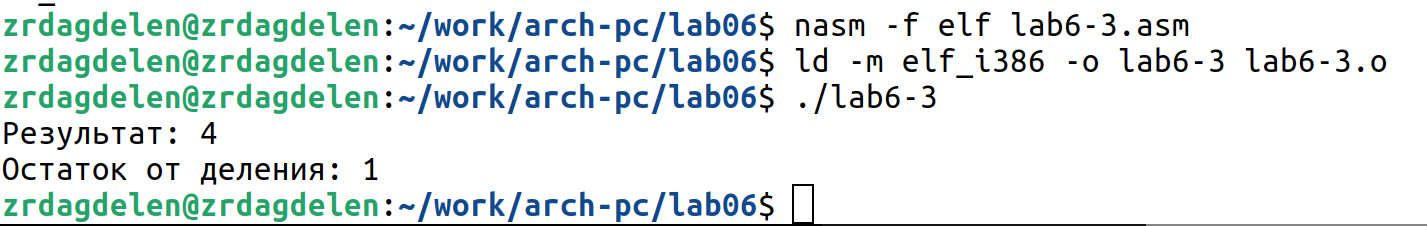
Создание файла lab6-3.asm

Ввожу текст программы из листинга 6.3 в lab6-3.asm(рис. [??]).



Текст программы

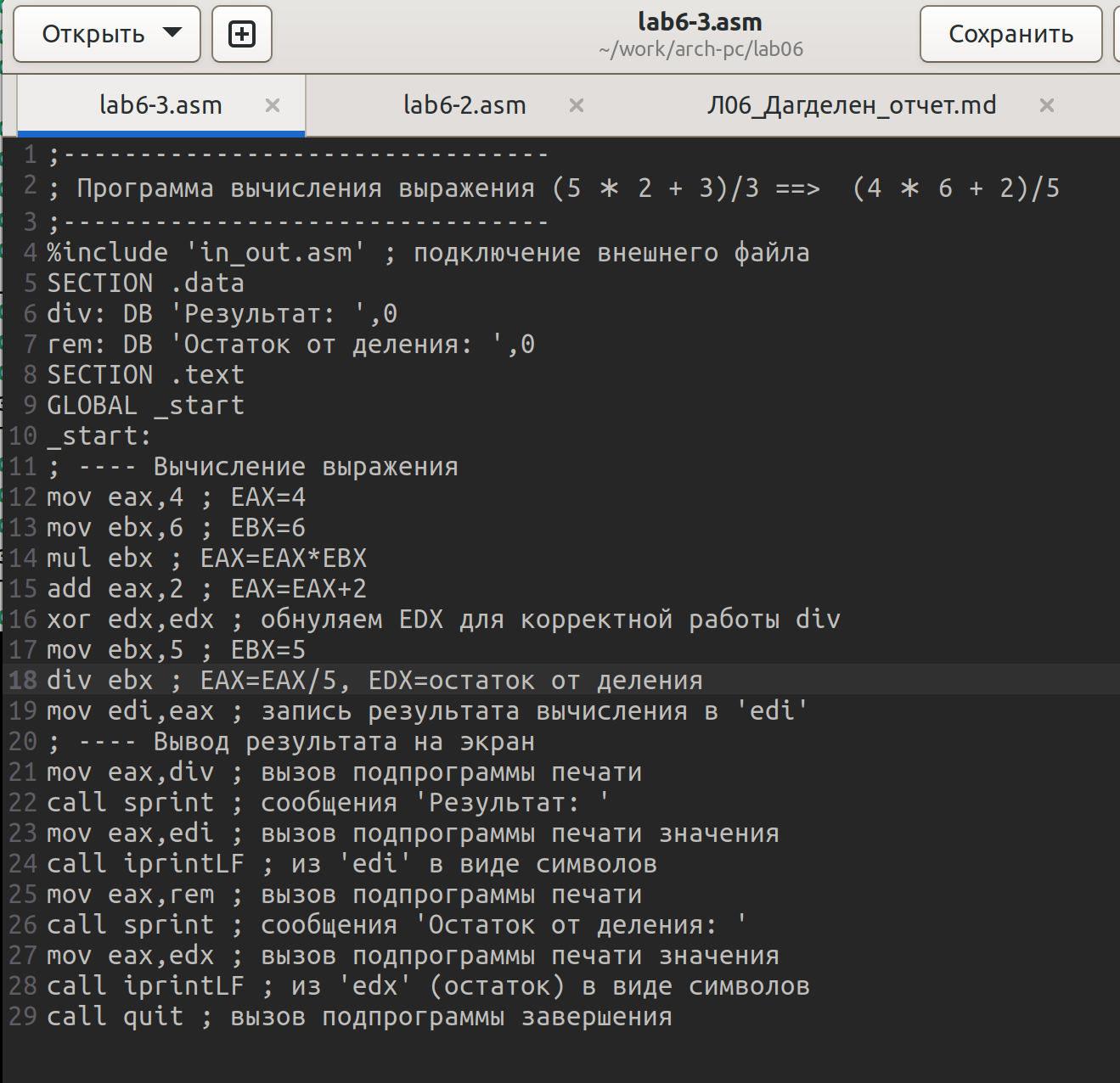
Создаю исполняемый файл и запускаю его(рис. [??]).



Создание исполняемого файла и его запуск

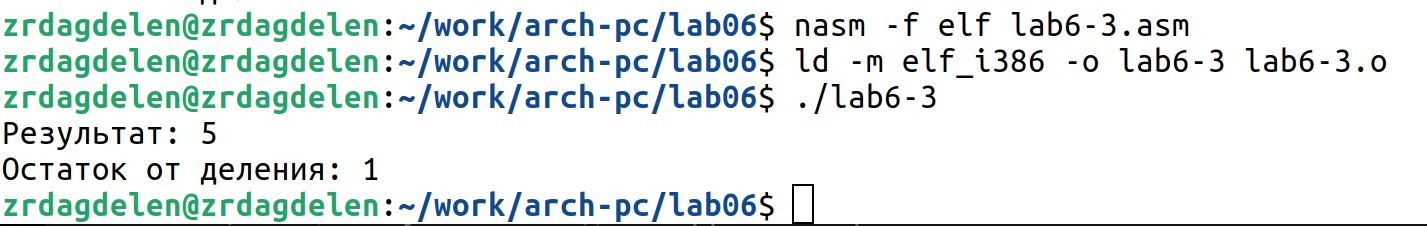
Результат работы программы следующий: результат = 4, остаток = 1.

Изменяю текст программы для вычисления выражения f(x) = (4 ∗ 6 + 2)/5(рис. [??]).



Текст программы

Создаю исполняемый файл и проверяю его работу(рис. [??]).

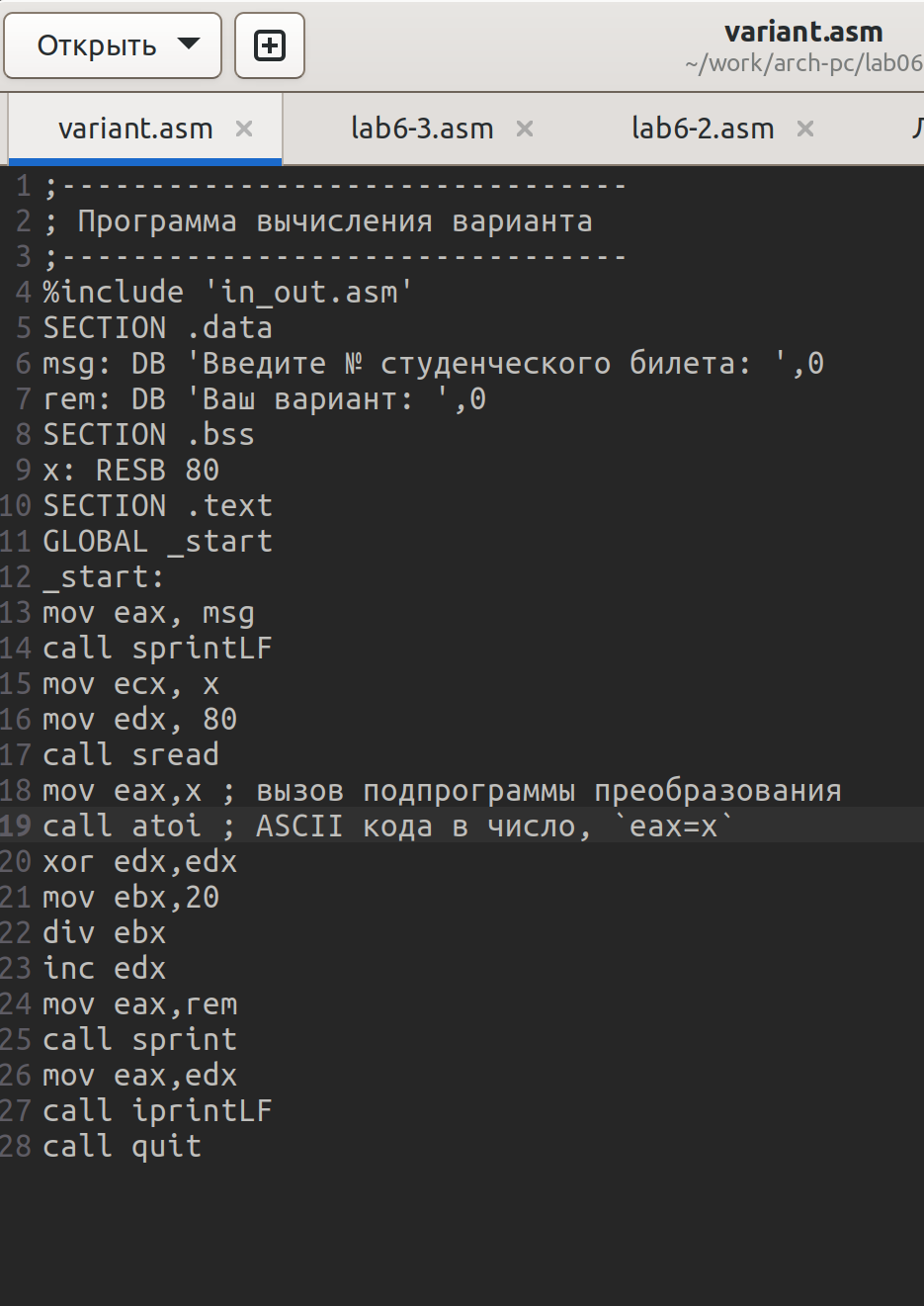


Создание исполняемого файла и его запуск

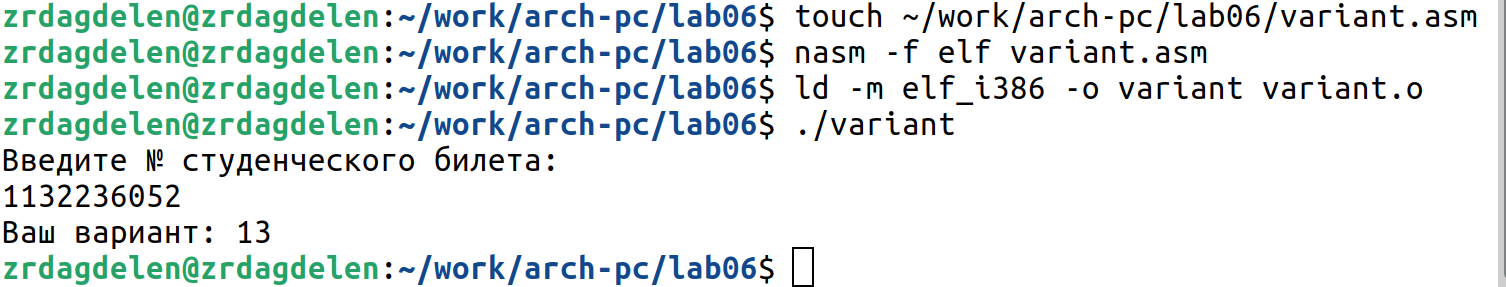
1. В качестве другого примера рассмотрю программу вычисления варианта задания по номеру студенческого билета, работающую по следующему алгоритму:

* вывести запрос на введение № студенческого билета
* вычислить номер варианта по формуле: (𝑆𝑛 mod 20) + 1, где 𝑆𝑛 – номер студен- ческого билета (В данном случае 𝑎 mod 𝑏 – это остаток от деления 𝑎 на 𝑏).
* вывести на экран номер варианта.

Создаю файл variant.asm в каталоге ~/work/arch-pc/lab06 с помощью touch, заполняю его в соответсвии с листингом 6.4(рис. [??]), создаю исполняемый файл и запускаю его(рис. [??]).



Текст программы



Создание файла variant.asm, его исполняемого файла и его запуск

Я посчитала для проверки правильности работы программы значение выражения самостоятельно, программа отработала верно.

## 4.3 Ответы на вопросы:

1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’? Ответ:

mov eax,rem  
call sprint

1. Для чего используется следующие инструкции? mov ecx, x mov edx, 80 call sread  
   Ответ: Инструкция mov ecx, x – 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. Для чего используется инструкция “call atoi”? Ответ: для вызова подпрограммы из внешнего файла, которая преобразует ascii-код символа в целое число и записывает результат в регистр eax
3. Какие строки листинга 6.4 отвечают за вычисления варианта? Ответ:

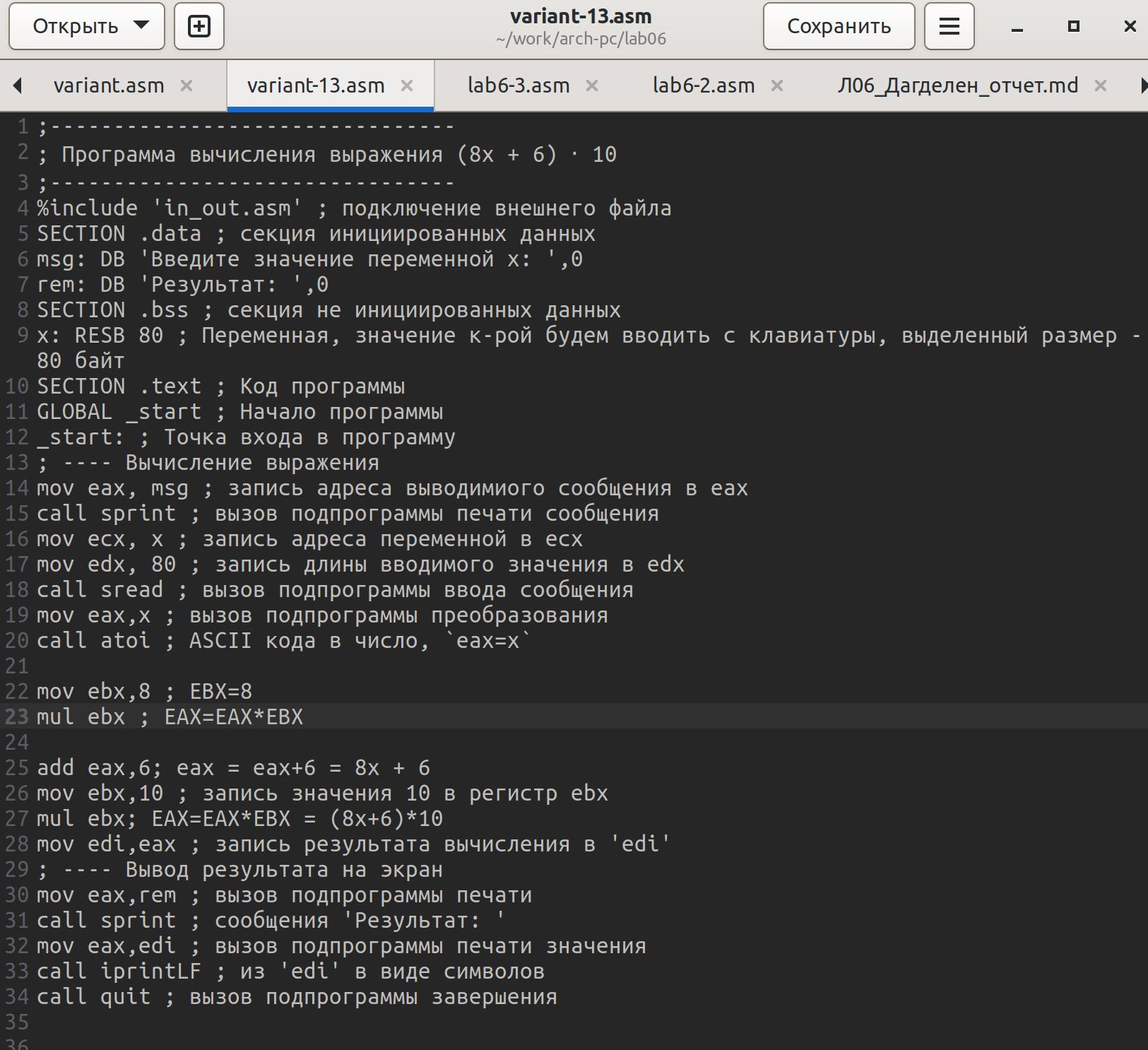
xor edx,edx ; обнуление edx для корректной работы div  
mov ebx,20 ; ebx = 20  
div ebx ; eax = eax/20, edx - остаток от деления  
inc edx ; edx = edx + 1

1. В какой регистр записывается остаток от деления при выполнении инструкции “div ebx”? Ответ: в регистр edx
2. Для чего используется инструкция “inc edx”? Ответ: для увеличения значения регистра edx на 1
3. Какие строки листинга 6.4 отвечают за вывод на экран результата вычислений? Ответ:

mov eax,edx  
call iprintLF

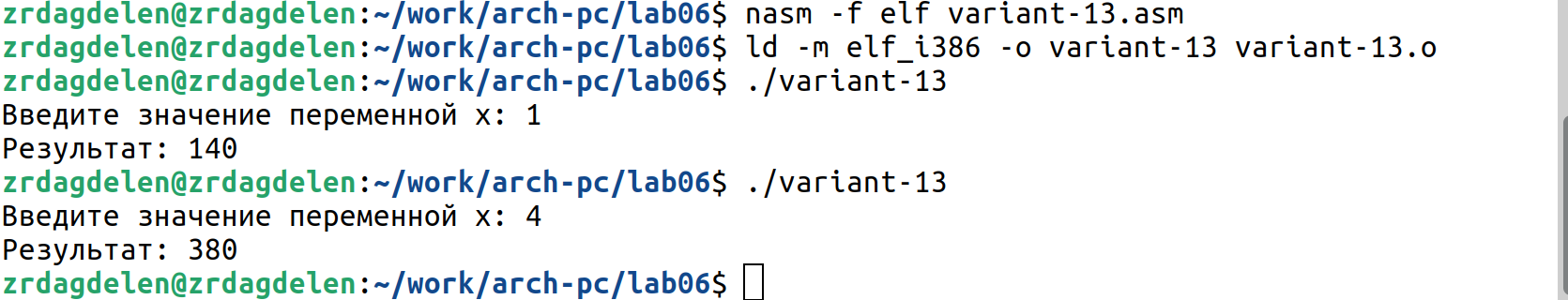
## 4.4 Задание для самостоятельной работы

**Напишу программу вычисления выражения y=f(x).**  
Создаю файл variant-13.asm. Открываю созданный файл для редактирования, ввожу в него текст программы для вычисления значения выражения (8x + 6) \* 10 (рис. [??]). Это выражение было под вариантом 13.



Текст программы

Создаю и запускаю исполняемый файл (рис. [??]).



Создание исполняемого файла и его запуск

При вводе значения 1, вывод - 140. При вводе значения 4, вывод - 380  
Программа отработала верно.

**Программа для вычисления значения выражения (8x + 6) \* 10.**

;--------------------------------  
; Программа вычисления выражения (8x + 6)\*10  
;--------------------------------  
%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 размер - 80 байт  
SECTION .text ; Код программы  
GLOBAL \_start ; Начало программы  
\_start: ; Точка входа в программу  
; ---- Вычисление выражения  
mov eax, msg ; запись адреса выводимиого сообщения в eax  
call sprint ; вызов подпрограммы печати сообщения  
mov ecx, x ; запись адреса переменной в ecx  
mov edx, 80 ; запись длины вводимого значения в edx  
call sread ; вызов подпрограммы ввода сообщения  
mov eax,x ; вызов подпрограммы преобразования  
call atoi ; ASCII кода в число, `eax=x`  
  
mov ebx,8 ; EBX=8  
mul ebx ; EAX=EAX\*EBX  
  
add eax,6; eax = eax+6 = 8x + 6  
mov ebx,10 ; запись значения 10 в регистр ebx  
mul ebx; EAX=EAX\*EBX = (8x+6)\*10  
mov edi,eax ; запись результата вычисления в 'edi'  
; ---- Вывод результата на экран  
mov eax,rem ; вызов подпрограммы печати  
call sprint ; сообщения 'Результат: '  
mov eax,edi ; вызов подпрограммы печати значения  
call iprintLF ; из 'edi' в виде символов  
call quit ; вызов подпрограммы завершения

# 5 Выводы

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

# 6 Список литературы

[Архитектура ЭВМ](https://esystem.rudn.ru/pluginfile.php/2089086/mod_resource/content/0/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%966.%20%D0%90%D1%80%D0%B8%D1%84%D0%BC%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5%20%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8%20%D0%B2%20NASM..pdf)