Отчёт по лабораторной работе №6

Дисциплина: Архитектура Компьютера

Азарцова Вероника Валерьевна

Содержание

# 1 Цель работы

Цель лабораторной работы - освоить арифметические инструкции в языке ассемблера NASM.

# 2 Задание

1. Ознакомление с теоретическим введением
2. Выполнение лабораторной работы
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 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.

## 3.2 Арифметические операции в NASM

1. Схема команды целочисленного сложения add (addition) выполняет сложение двух операндов и записывает результат по адресу первого операнда. Команда add работает как с числами со знаком, так и без знака:  
   add (операнд\_1), (операнд\_2)
2. Команда целочисленного вычитания sub (subtraction) работает аналогично команде add:  
   sub (операнд\_1), (операнд\_2)
3. Прибавление единицы называется инкрементом, а вычитание — декрементом. Для этих операций существуют специальные команды: inc (increment) и dec (decrement), которые увеличивают и уменьшают на 1 свой операнд:  
   inc (операнд) dec (операнд)
4. Команда изменения знака neg: neg (операнд)
5. Умножение и деление, в отличии от сложения и вычитания, для знаковых и беззнаковых чисел производиться по-разному, поэтому существуют различные команды - mul (multiply) и imul:  
   mul (операнд) - Беззнаковое умножение  
   imul (операнд) - Знаковое умножение  
   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 (табл. 1).

Таблица 1: Регистры, используемые командами умножения в NASM

| Размер операнда | Неявный множитель | Результат умножения |
| --- | --- | --- |
| 1 Байт | AL | AX |
| 2 Байта | AX | DX:AX |
| 3 Байта | EAX | EDX:EAX |

1. Для деления, как и для умножения, существует две команды - div и idiv: div (делитель) - Беззнаковое деление idiv (делитель) - 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 (табл. 2).

Таблица 2: Регистры, используемые командами деления в NASM

| Размер операнда (делителя) | Делимое | Частное | Остаток |
| --- | --- | --- | --- |
| 1 Байт | АX | AL | AH |
| 2 Байта | DX:AX | AX | DX |
| 3 Байта | EDX:EAX | EAX |  |

## 3.3 Перевод символа числа в десятичную символьную запись

При вводе данных с клавиатуры, введенные дан- ные 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 этого действия при выполнения лабораторных работ в файле in\_out.asm реализованы подпрограммы . Это:

* iprint – вывод на экран чисел в формате ASCII, перед вызовом iprint в регистр eax необходимо записать выводимое число - mov eax,(int).
* iprintLF – работает аналогично iprint, но при выводе на экран после числа добавляет к символ перевода строки.
* atoi – функция преобразует ascii-код символа в целое число и записает результат в регистр eax, перед вызовом atoi в регистр eax необходимо записать число - mov eax, (int)

# 4 Выполнение лабораторной работы

## 4.1 Символьные и численные данные в NASM

1. Создаю каталог для программы лабораторной работы №6, перехожу в него, создаю файл lab6-1.asm и проверяю действия с помощью ls (рис. 1).

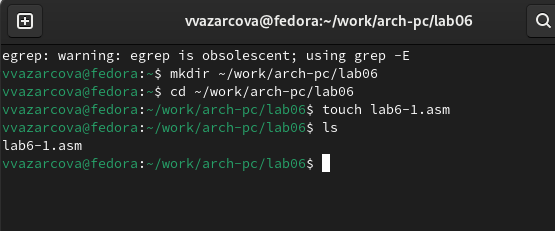


Рис. 1: Создание каталога для лабораторной работы и lab6-1.asm

1. Ввожу в файл lab6-1.asm текст программы, в которой в регистр eax записывается символ 6 (mov eax,‘6’), в регистр ebx символ 4 (mov ebx,‘4’), к значению в регистре eax прибавляется значение регистра ebx (add eax,ebx, результат сложения запишется в регистр eax), потом выводится результат. Так как для работы функции sprintLF в регистр eax должен быть записан адрес, необходимо использовать дополнительную переменную. Значение регистра eax записывается в переменную buf1 (mov [buf1],eax), а затем адрес переменной buf1 записывается в регистр eax (mov eax,buf1) и вызывается функция sprintLF.  
   Для этого, открываю Midnight Commander с помощью команды mc (рис. 2).

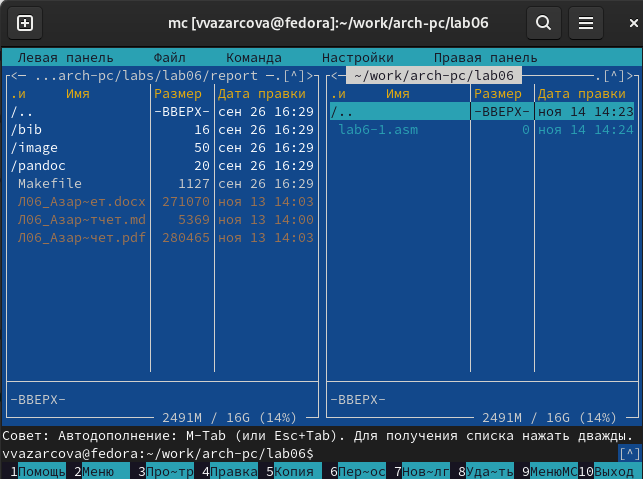


Рис. 2: Интерфейс Midnight Commander

Далее, открываю lab6-1.asm и ввожу текст программы (рис. 3).

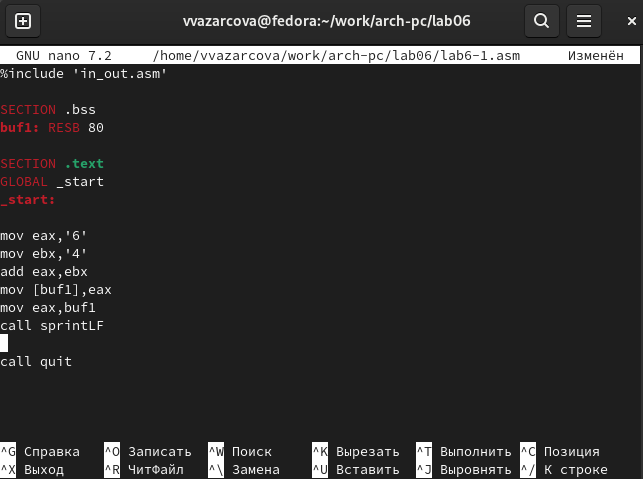


Рис. 3: Текст программы в lab6-1.asm

Создаю исполняемый файл, проверяю создание файла командой ls и запускаю его (рис. 4).

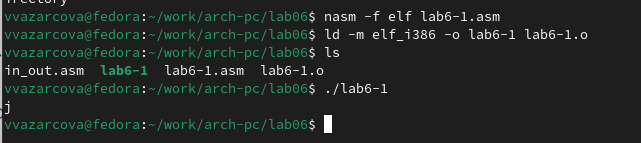


Рис. 4: Запуск lab6-1

Программа выводит значение регистра eax, ожидается число 10, но результатом будет символ j, потому что код символа 6 равен 00110110, а код символа 4 - 00110100, и команда add записывает в регистр eax сумму кодов 01101010, что является кодом символа j.

1. Изменяю текст программы так, чтобы вместо символов записать в регистр цифры (рис. 5).

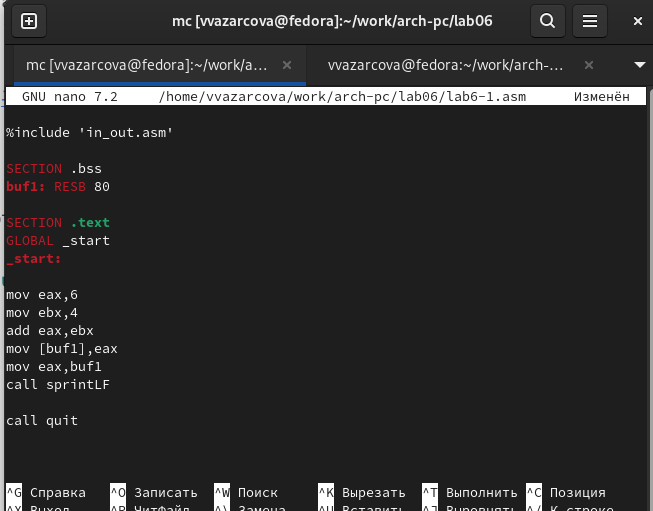


Рис. 5: Измененный текст lab6-1.asm

Создаю исполняемый файл, проверяю создание файла командой ls и запускаю его (рис. 6).

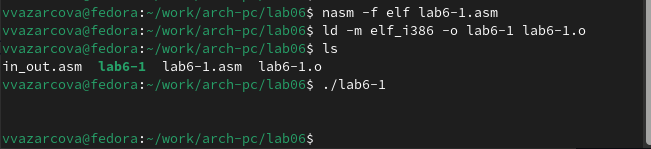


Рис. 6: Запуск измененного lab6-1

Программа должна выводить символ на новой строке, выводятся две пустые строки. Согласно таблице ASCII, символ с кодом 10 это LF, т.е. символ перехода на новую строку, т.е. символ корректно отображается при выводе на экран (в виде новой, второй строки).

1. Создаю файл lab6-2.asm командой touch и проверяю его наличие командой ls (рис. 7).

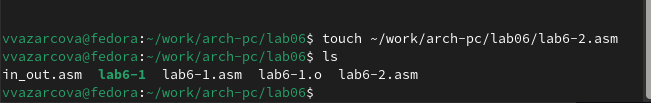


Рис. 7: Создание lab6-2.asm

Открываю файл с помощью MC и ввожу текст программы (рис. 8).

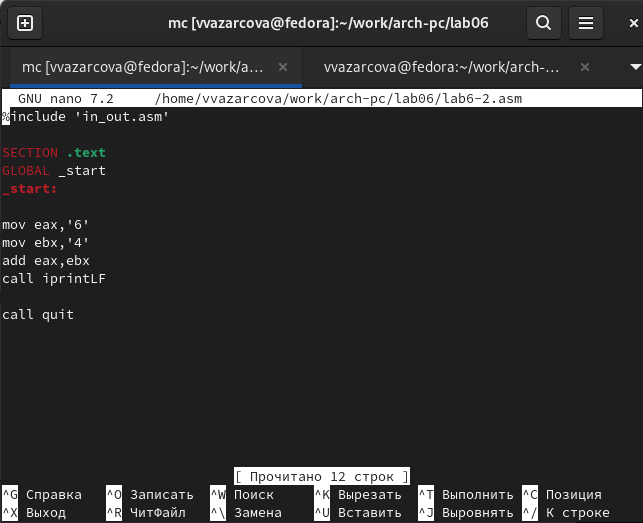


Рис. 8: Текст программы в lab6-2.asm

Создаю исполняемый файл, проверяю создание файла командой ls и запускаю его (рис. 9).

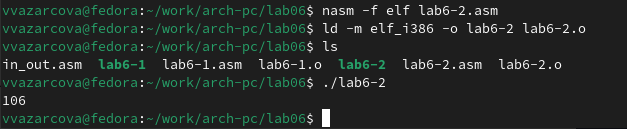


Рис. 9: Запуск lab6-2

В результате работы программы выводится число 106. Аналогично первой программе, команда add складывает коды символов ‘6’ и ‘4’, но в отличии от первой программы функция iprintLF выводит число, а не символ, кодом которого является это число.

1. Аналогично номеру три, изменю символы на числа в тексте программы (рис. 10).

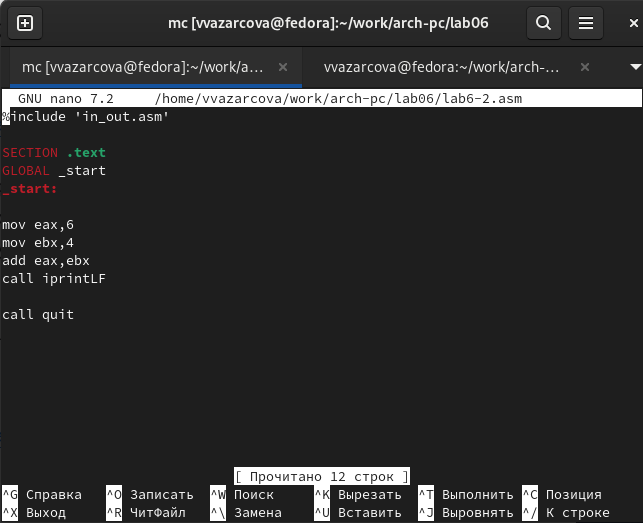


Рис. 10: Измененный текст программы в lab6-2.asm

Создаю исполняемый файл, проверяю создание файла командой ls и запускаю его (рис. 11).

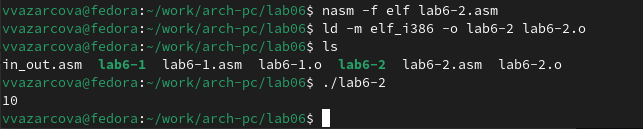


Рис. 11: Запуск измененного lab6-2

Программа выводит число 10, т.к. теперь команда add складывает числа 4 и 6 и выводит результат в виде числа - 10.  
Заменю функцию iprintLF на iprint (рис. 12).

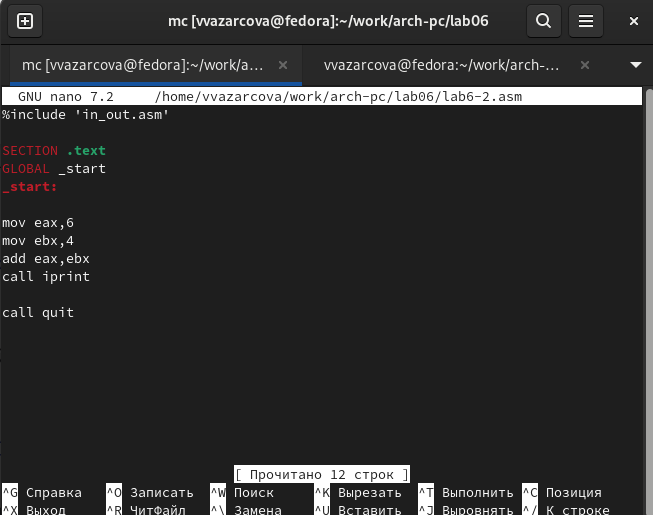


Рис. 12: Замена iprintLF на iprint

Создаю исполняемый файл, проверяю создание файла командой ls и запускаю его (рис. 13).

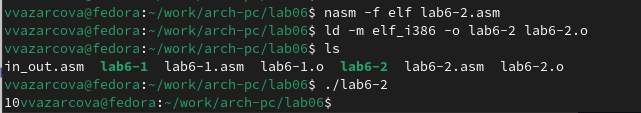


Рис. 13: Запуск еще раз измененного lab6-2

Программа выводит 10, но не на новой строке. В этом состоит разница между iprintLF и iprint - первая подпрограмма выводит результат на новую строку, а вторая нет.

## 4.2 Выполнение арифметических операций в NASM

1. Создаю файл lab6-3.asm в каталоге лаборатной работы (рис. 14).

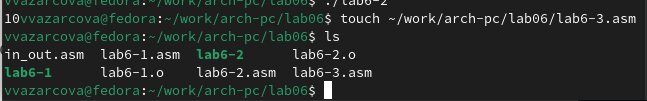


Рис. 14: Создание lab6-3.asm

Ввожу в lab6-3.asm программу вычисления выражения (рис. 15).

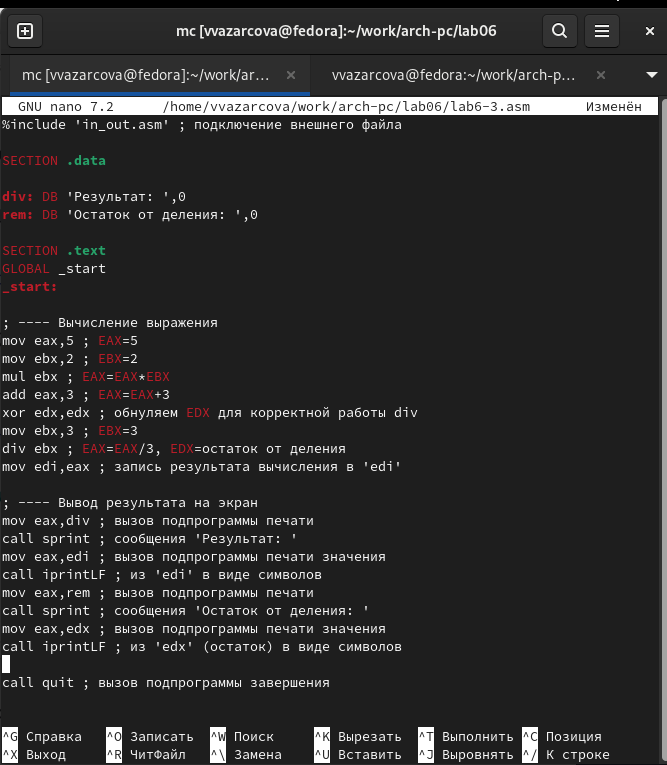


Рис. 15: Создание lab6-3.asm

Создаю исполняемый файл, проверяю создание файла командой ls и запускаю его (рис. 16).

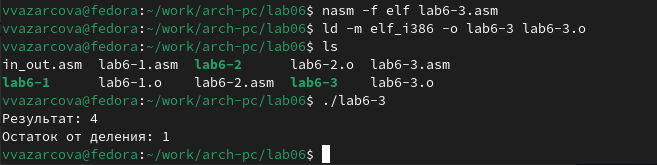


Рис. 16: Запуск lab6-3

Результат программы соответствует ожидаемому.

Изменяю текст программы для вычисления выражения (рис. 17).

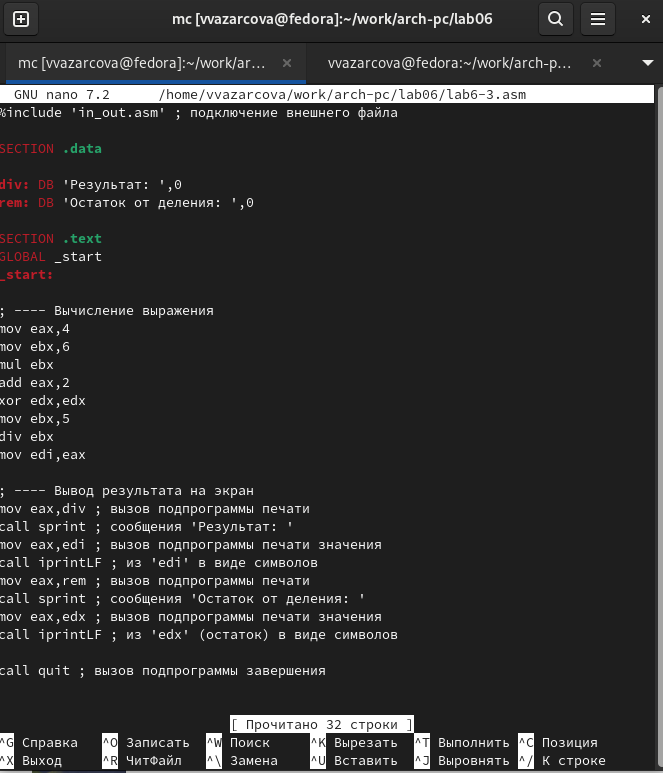


Рис. 17: Измененный текст программы в lab6-3.asm

Создаю исполняемый файл, проверяю создание файла командой ls и запускаю его (рис. 18).

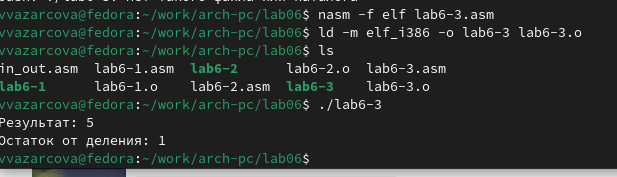


Рис. 18: Запуск измененного lab6-3

Программа выводит результат 5 и остаток 1, значит, она работает верно (4\*6=24; 24+2=26; 26/5=5, ост. 1).

1. Создаю файл variant.asm и проверяю его наличие с помощью ls (рис. 19).

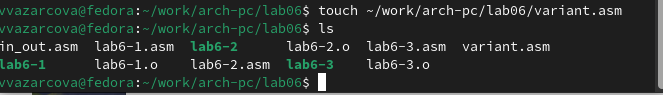


Рис. 19: Создание variant.asm

Ввожу текст программы вычисления варианта задания по номеру студенческого билета в variant.asm (рис. 20).

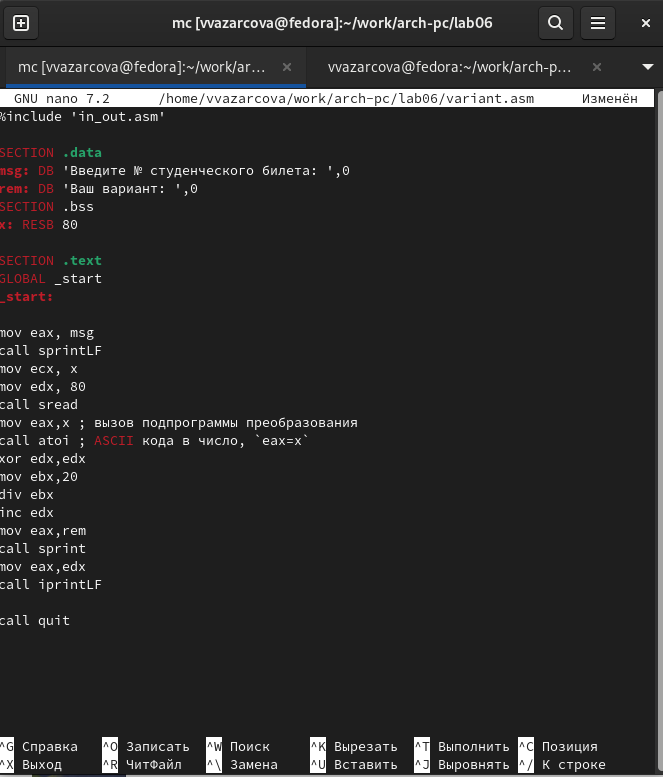


Рис. 20: Текст программы в variant.asm

Создаю исполняемый файл, проверяю создание файла командой ls и запускаю его (рис. 21).

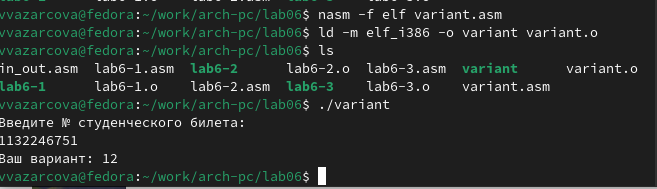


Рис. 21: Запуск измененного lab6-3

Программа запрашивает ввод. Я ввожу свой номер студенческого билета (1132246751), и программа выводит, что мой вариант - 12.  
Ответы на вопросы по выполнению лабораторной работы:

1. За вывод на экран сообщения ‘Ваш вариант’ отвечают следующие строки:  
   mov eax, rem  
   call sprint
2. Следующие инструкции  
   mov ecx, x  
   mov edx, 80  
   call sread  
   используются для того, чтобы загрузить адрес переменной x в регистр ecx.
3. Инструкция “call atoi” вызывает подпрограмму из файла in\_out.asm, которая переводит строку ASCII в целое число.
4. За вычисление варианта отвечают следующие строки:  
   xor edx, edx  
   mov ebx, 20  
   div ebx  
   Эти комманды ищут остаток от деления номера студенческого билета на 20.
5. Остаток от деления при выполнении инструкции “div ebx” записывается в регистр edx.
6. Инструкция inc edx увеличивает значение в регистре edx на 1, для того, чтобы список возможных вариантов задания, вычисленных программой, начинался не с 0 а с 1.
7. За вывод на экран результата вычислений отвечают строки:  
   mov eax, edx  
   call iprintLF

# 5 Задания для самостоятельной работы

1. Т.к. мой вариант задания - 12, мне нужно написать программу для вычисления выражения , которая будет выводить выражение для вычисления, выводить запрос на ввод значения x, вычислять заданное выражение в зависимости от введенного x и выводить результат вычислений.

Создаю файл var12.asm в каталоге для выполнения лабораторной работы с помощью команды touch и проверяю его наличие командой ls (рис. 22).

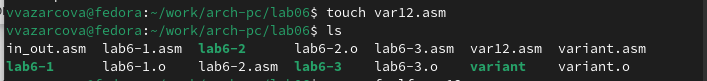


Рис. 22: Создание var12.asm

1. Открываю var12.asm с помощью MC и пишу код программы с пояснениями, аналогично программам, приведенным в пример в ходе лабораторной работы (рис. 23).

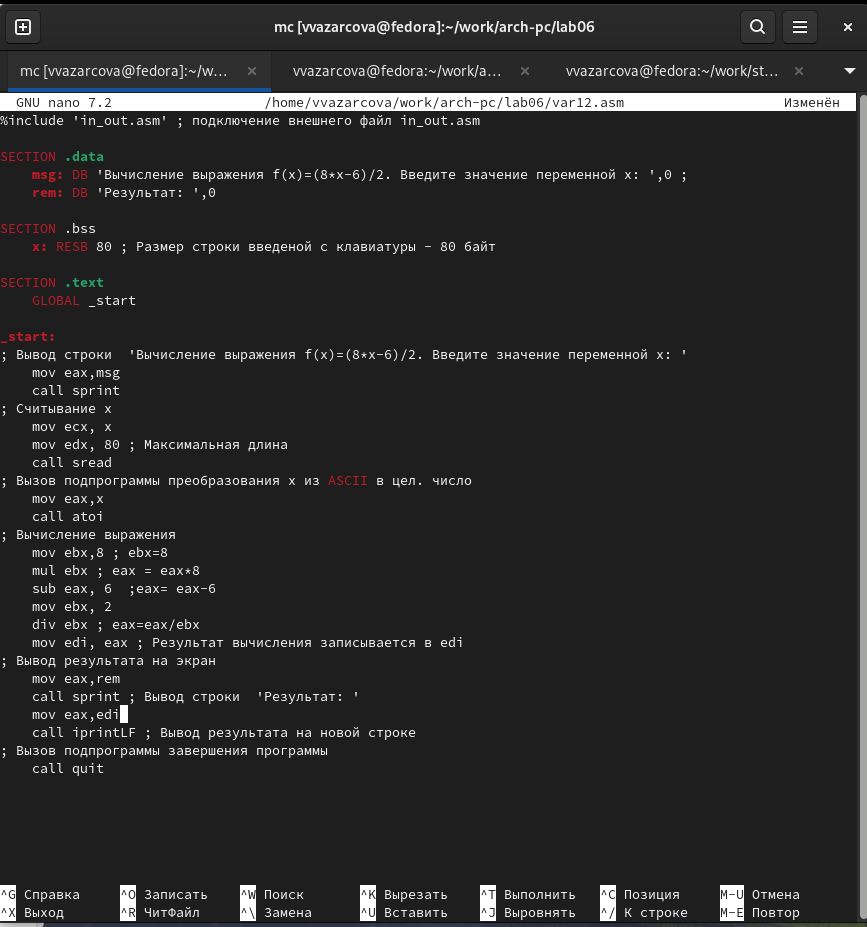


Рис. 23: Текст программы в var12.asm

Текст программы:

%include 'in\_out.asm' ; подключение внешнего файл in\_out.asm  
  
SECTION .data  
 msg: DB 'Вычисление выражения f(x)=(8\*x-6)/2. Введите значение переменной x: ',0 ;  
 rem: DB 'Результат: ',0  
  
SECTION .bss  
 x: RESB 80 ; Размер строки введеной с клавиатуры - 80 байт  
  
SECTION .text  
 GLOBAL \_start  
  
\_start:  
; Вывод строки 'Вычисление выражения f(x)=(8\*x-6)/2. Введите значение переменной x: '  
 mov eax,msg  
 call sprint  
; Считывание x  
 mov ecx, x  
 mov edx, 80 ; Максимальная длина  
 call sread  
; Вызов подпрограммы преобразования x из ASCII в цел. число  
 mov eax,x  
 call atoi  
; Вычисление выражения  
 mov ebx,8 ; ebx=8  
 mul ebx ; eax = eax\*8  
 sub eax, 6 ;eax= eax-6  
 mov ebx, 2  
 div ebx ; eax=eax/ebx  
 mov edi, eax ; Результат вычисления записывается в edi  
; Вывод результата на экран  
 mov eax,rem  
 call sprint ; Вывод строки 'Результат: '  
 mov eax,edi  
 call iprintLF ; Вывод результата на новой строке  
; Вызов подпрограммы завершения программы  
 call quit

* Сначала в программе я задаю переменные с текстом, который нужно будет вывести в её ходе, затем выделяю память на неинициированные данные, далее прописываю команду \_start для начала работы программы.
* В тексте программы я вызываю подпрограмму sprint вывода строки и вывожу текст вычисляемого выражения и просьбу ввести x, задаю максимальную длину x, вызываю подпрограмму ввода с клавиатуры sread и записываю x в ecx. Затем, вызываю подпрограмму atoi для перевода x из строчки ASCII в целое число - целое число x записывается в eax.
* Далее, с помощью подпрограмм mul, sub и div (умножения, вычитания, деления), провожу нужные арифметические действия, при этом записывая значения с которыми работаю в ebx. В конце записываю значение результата в edi.
* Последними я вызываю подпрограммы sprint и iprintLF, чтобы вывести строку, обьявляющую результат, и затем вывести результат выражения на новой строке, переместив выводимое значение в eax перед каждой из подпрограмм.

1. Создаю исполняемый файл, проверяю создание файла командой ls и запускаю его (рис. 24).

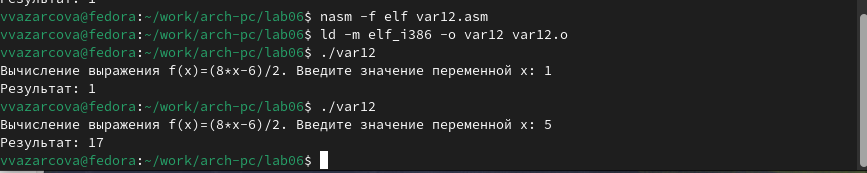


Рис. 24: Запуск var12.asm

В итоге, программа выводит результат 1 при вводе числа 1 и результат 17 при вводе числа 5; Это верно, поскольку:

* (8\*1-6)/2=2/2=1
* (8\*5-6)/2=(40-6)/2=34/2=17

Значит, программа написана корректно.

# 6 Выводы

Подводя итоги данной лабораторной работы, я закрепила знания работы подфункций из внешнего файла in\_out.asm, научилась пользоваться арифметическими функциями и успешно написала и запустила несколько программ, производящих арифметические действия в NASM.

# Список литературы