РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

по лабораторной работе № 6

дисциплина: Архитектура компьютера

Студент: Бессонов Андрей Максимович

Группа: НКАбд - 01 - 25

МОСКВА

2025 г.

Содержание

Список иллюстраций

1. Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

2. Теоретическое введение

2.1 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.

2.2. Арифметические операции в NASM

2.2.1. 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 <операнд_1>, < операнд_2>
Допустимые сочетания операндов для команды add аналогичны сочетаниям операндов для команды mov.

2.2.2. Целочисленное вычитание sub.

Команда целочисленного вычитания sub (от англ. subtraction – вычитание) работает аналогично команде add и выглядит следующим образом:
sub <операнд_1>, < операнд_2>

2.2.3. 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 <операнд>

dec <операнд>
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.

2.2.4. 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.

```
mov ax,1      ; AX = 1
neg ax        ; AX = -1
```

2.2.5. Команды умножения mul и imul.

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 <операнд>

Для знакового умножения используется команда imul:

imul <операнд>

2.2.6. 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
```

2.3. 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. 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. 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>).

3. Выполнение лабораторной работы

Создадим каталог для программам лабораторной работы № 6, перейдем в него и создадим файл lab6-1.asm.Введем в файл lab6-1.asm текст программы из листинга6.1.

Создадим исполняемый файл и запустим его. Программа выведет символ j.

```
ambessonov@fedora:~$ mkdir ~/work/arch-pc/lab06
ambessonov@fedora:~$ cd ~/work/arch-pc/lab06
ambessonov@fedora:~/work/arch-pc/lab06$ touch lab6-1.asm
ambessonov@fedora:~/work/arch-pc/lab06$
```

Открыть ▾ | lab6-1.asm ~/work/arch-pc/lab06

```
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
```

Рис 3.1: 31

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

Рис 3.2: 32

Далее изменим текст программы и вместо символов, запишем в регистры числа. Исправим текст программы (Листинг 6.1) следующим образом: заменим строки mov eax,'6' mov ebx,'4' на строки mov eax,6 mov ebx,4

Создадим исполняемый файл и запустим его. В данном случае выводится символ с кодом 10.

Код 10 согласно таблице ASCII соответствует символу: LF (Line Feed) — это управляющий символ, который означает "перевод строки".

Рис 3.3: 33

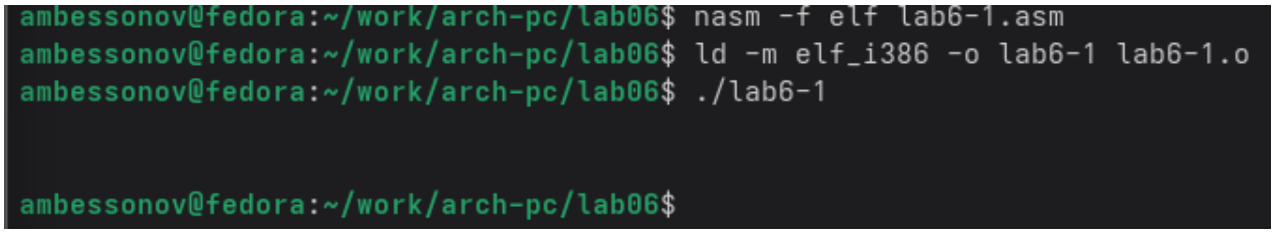

Рис 3.4: 34

Создадим файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 и введем в него текст программы из листинга 6.2.

Создадим исполняемый файл и запустим его.

В результате работы программы мы получим число 106.

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

Открыть — lab6-2.asm — ~/work/arch-pc/lab06

```
%include 'in_out.asm'

    SECTION .text
    GLOBAL _start
        _start:
        mov eax,'6'
        mov ebx,'4'
        add eax,ebx
        call iprintLF

        call quit
```

Рис 3.5: 35

Изменим символы на числа. Заменим строки mov eax,'6' mov ebx,'4' на строки mov eax,6 mov ebx,4.

Создадим исполняемый файл и запустим его. Результатом является: 10.

```
ambessonov@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
ambessonov@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
ambessonov@fedora:~/work/arch-pc/lab06$ ./lab6-2
10
ambessonov@fedora:~/work/arch-pc/lab06$
```

Открыть — lab6-2.asm — ~/work/arch-pc/lab06

```
%include 'in_out.asm'

    SECTION .text
    GLOBAL _start
        _start:
        mov eax,6
        mov ebx,4
        add eax,ebx
        call iprintLF

        call quit
```

Рис 3.6: 36

Заменим функцию iprintLF на iprint. Создадим исполняемый файл и запустим его. Вывод функций iprintLF и iprint отличается тем, что iptint выводит текст или данные ровно в той позиции, где находится курсор, и оставляет курсор сразу после выведенных данных. А iprintLF выводит текст, а затем добавляет символ перевода строки (LF, код 10, \n). Это перемещает курсор в начало следующей строки.

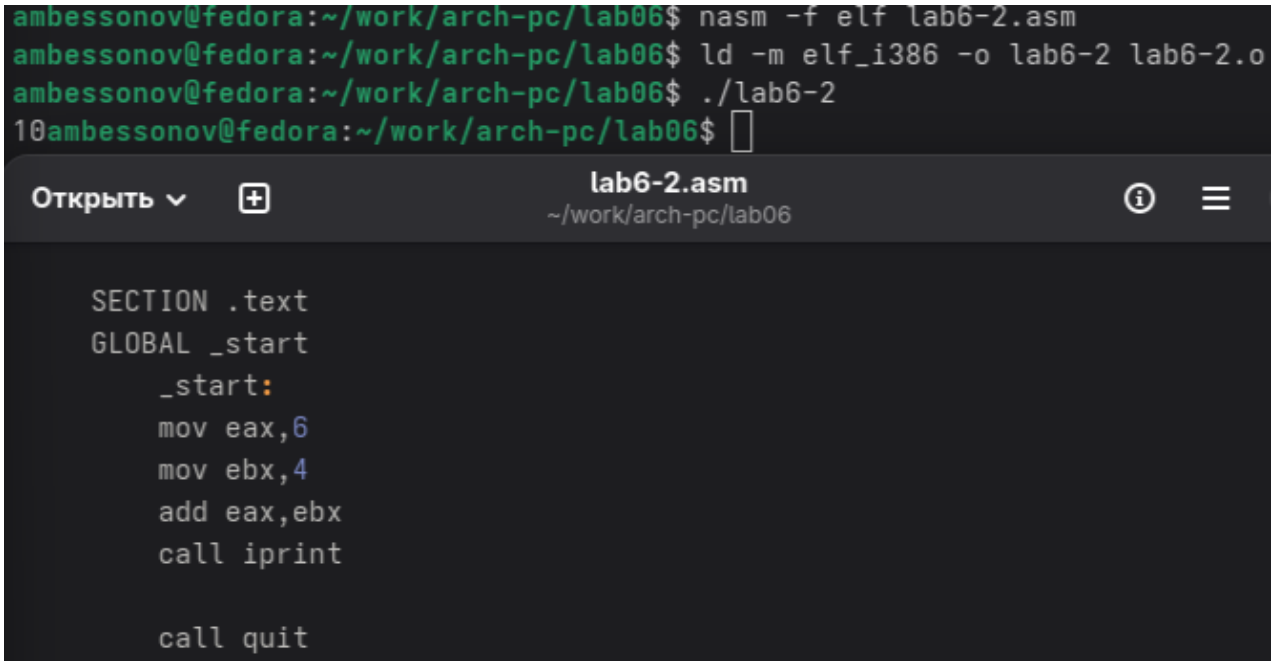

Рис 3.7: 37

Создадим файл lab6-3.asm в каталоге ~/work/arch-pc/lab06.

Создадим исполняемый файл и запустим его.

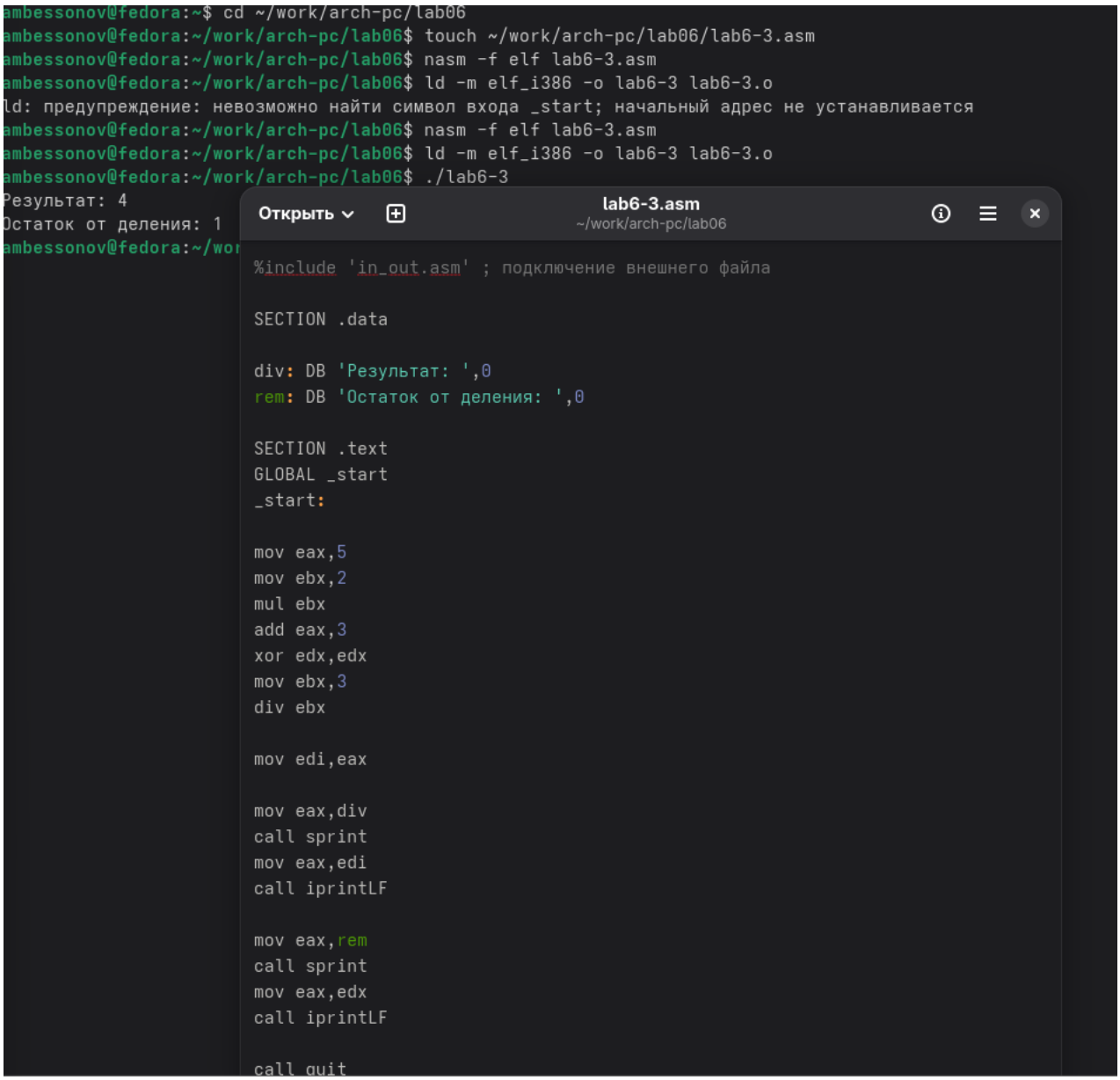

Рис 3.8: 38

Изменим текст программы для вычисления выражения $f(x) = (4 * 6 + 2)/5$. Создадим исполняемый файл и проверим его работу.

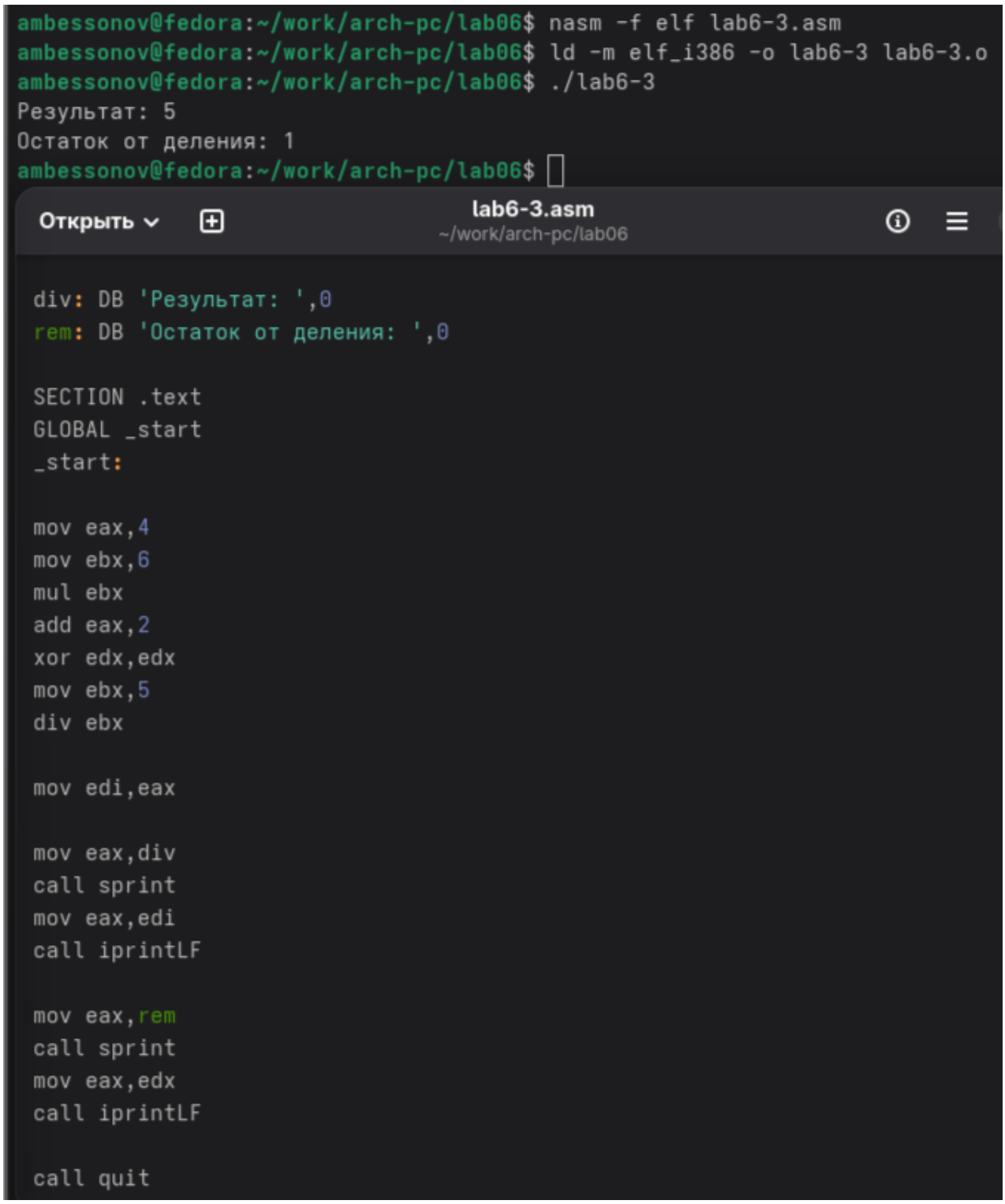

Рис 3.9: 39

Создадим файл variant.asm в каталоге ~/work/arch-pc/lab06.

Внимательно изучим текст программы из листинга 6.4 и введем в файл variant.asm.

Создадим исполняемый файл и запустим его. Заметим что результат совпадает с аналитическим решением.

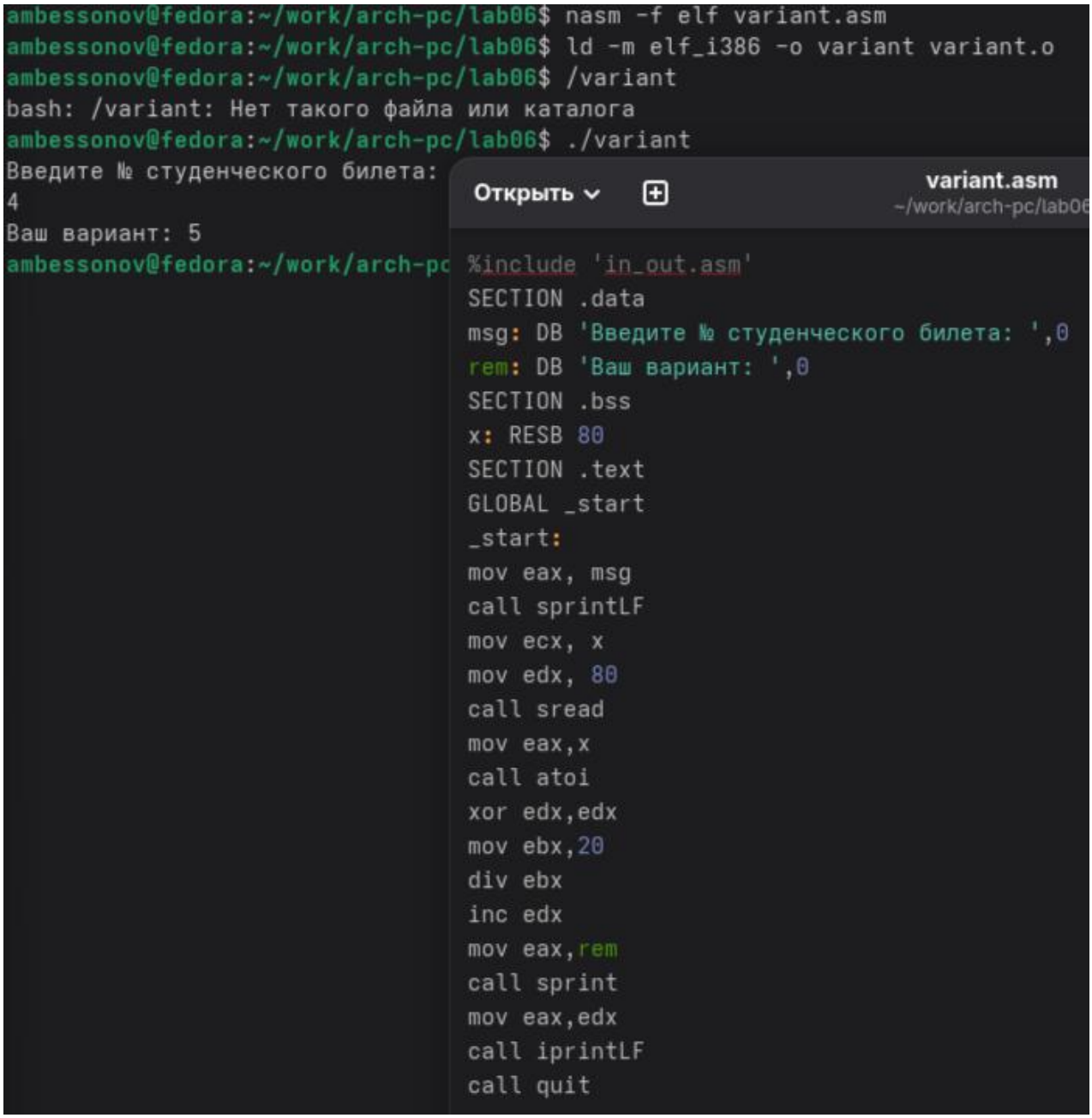

Рис 3.10: 310

Ответы на вопросы:

1) Строки, отвечающие за вывод сообщения “Ваш вариант: ”:

   mov eax,rem

   call sprint

   Здесь rem содержит строку “Ваш вариант: ”

2) Инструкции для ввода данных:

   mov ecx, x ; помещает адрес буфера x в ecx

   mov edx, 80 ; устанавливает максимальную длину ввода (80 байт)

   call sread ; вызывает функцию чтения строки из stdin

3) Инструкция “call atoi”:

   Преобразует ASCII-строку в число. Функция atoi принимает строку в eax и возвращает целое число в eax

4) Строки, отвечающие за вычисления варианта:

   xor edx,edx ; обнуляет edx (подготовка к делению)

   mov ebx,20 ; делитель = 20 (количество вариантов)

   div ebx ; eax = eax/ebx, edx = остаток от деления

   inc edx ; увеличивает остаток на 1 (варианты от 1 до 20)

5) Регистр для остатка от деления:

   При выполнении div ebx остаток записывается в регистр EDX

6) Инструкция “inc edx”:

   Увеличивает значение в регистре EDX на 1. Используется потому что остаток от деления может быть от 0 до 19, а варианты нужны от 1 до 20

7) Строки для вывода результата:

   mov eax,edx ; помещает результат (номер варианта) в eax

   call iprintLF ; выводит число и перевод строки

Вывод: выполнили задания лабораторной работы.

4. Выполнение самостоятельной работы

Напишем программу вычисления выражения $y = f(x)$, где $f(x)$ = (5 + x)^2 – 3.

Создадим файл work.asm в каталоге ~/work/arch-pc/lab06.

Создадим исполняемый файл и запустим его, проверим его для двух значений переменной x,

$x1$ = 5; $x2$ = 1

Результаты совпадают с аналитическим решением.

Открыть

work.asm
~/work/arch-pc/lab06

```
%include 'in_out.asm'

SECTION .data
    msg db 'Программа вычисления функции y = (5 + x)^2 - 3',0
    expr db 'Вычисляем: y = (5 + x)^2 - 3',0
    prompt db 'Введите значение x: ',0
    result db 'Результат: y = ',0
    newline db 10,0

SECTION .bss
    x resb 10

SECTION .text
global _start

_start:
    ; Вывод выражения для вычисления
    mov eax, msg
    call sprint
    mov eax, newline
    call sprint

    mov eax, expr
    call sprint
    mov eax, newline
    call sprint
    mov eax, newline
    call sprint

    ; Запрос ввода значения x
    mov eax, prompt
    call sprint

    ; Чтение ввода
    mov ecx, x
    mov edx, 10
    call sread

    ; Проверка и преобразование ввода
    mov eax, x
    call atoi        ; Преобразуем строку в число в EAX

    ; Вычисление y = (5 + x)^2 - 3
    mov ebx, eax     ; Сохраняем x в EBX

    add ebx, 5       ; ebx = 5 + x
    mov eax, ebx     ; eax = 5 + x
    imul eax, ebx    ; eax = (5 + x)^2
    sub eax, 3       ; eax = (5 + x)^2 - 3

    ; Сохраняем результат
    push eax

    ; Вывод результата
    mov eax, result
    call sprint

    pop eax          ; Восстанавливаем результат
    call iprint      ; Выводим число

    mov eax, newline
    call sprint

    call quit
```

Рис 4.1: 41

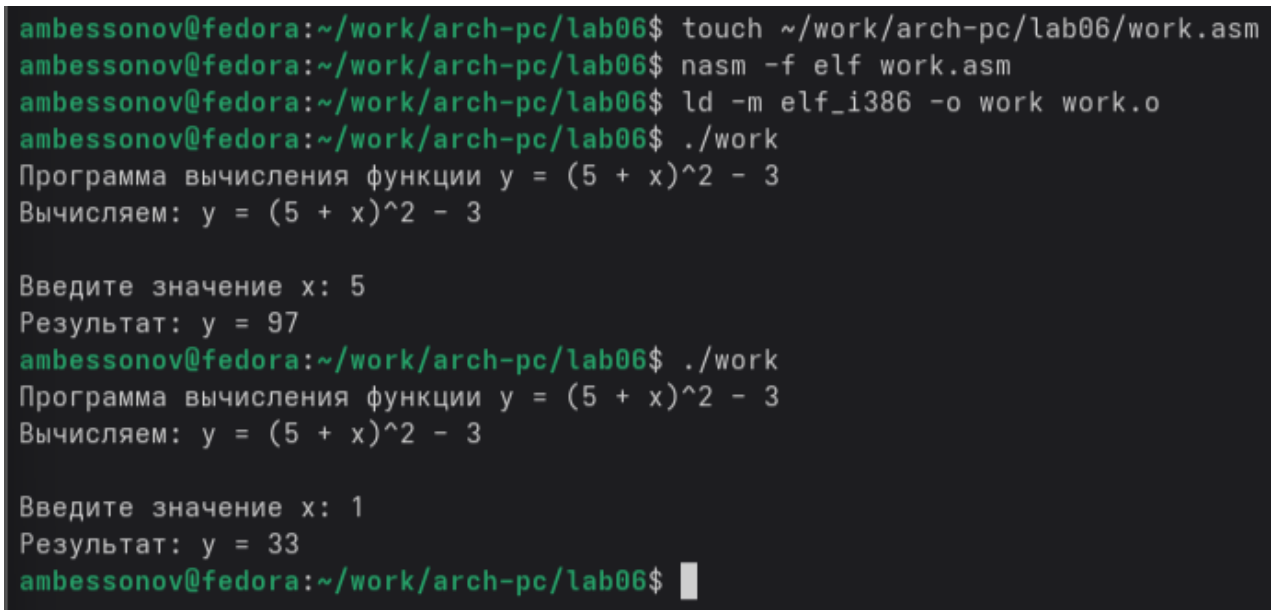

Рис 4.2: 42

Вывод: выполнили задания самостоятельной работы, отработали навыки полученные в ходе лабораторной работы.

5. Выводы

Освоили арифметические инструкции языка ассемблера NASM.