РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ
ФАКУЛЬТЕТ ФИЗИКО-МАТЕМАТИЧЕСКИХ И ЕСТЕСТВЕННЫХ НАУК

Лабораторная работа №6.

Арифметические операции в NASM.

Левищева Анастасия Петровна, НКАбд-02-25

Содержание

Список иллюстраций

Список таблиц

1. Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

2. Задание

1. Написать программу вычисления выражения $y = f(x)$. Программа должна выводить выражение для вычисления, выводить запрос на ввод значения x, вычислять заданное выражение в зависимости от введенного x, выводить результат вычислений. Вид функции $f(x)$ выбрать из таблицы 6.3 вариантов заданий в соответствии с номером, полученным при выполнении лабораторной работы. Создайте исполняемый файл и проверьте его работу для значений x1 и x2 из 6.3.

Таблица 6.3. Выражения для $f(x)$ для задания №1

Номер варианта	Выражение для $f(x)$	x_1	x_2
1	$(10 + 2x)/3$	1	10
2	$(12x + 3)5$	1	6
3	$(2 + x)^2$	2	8
4	$\frac{4}{3}(x - 1) + 5$	4	10
5	$(9x - 8)/8$	8	64
6	$x^3/2 + 1$	2	5
7	$5(x - 1)^2$	3	5
8	$(11 + x) \cdot 2 - 6$	1	9
9	$10 + (31x - 5)$	3	1
10	$5(x + 18) - 28$	2	3
11	$10(x + 1) - 10$	1	7
12	$(8x - 6)/2$	1	5
13	$(8x + 6) \cdot 10$	1	4
14	$(\frac{x}{2} + 8) \cdot 3$	1	4
15	$(5 + x)^2 - 3$	5	1
16	$(10x - 5)^2$	3	1
17	$18(x + 1)/6$	3	1
18	$3(x + 10) - 20$	1	5
19	$(\frac{1}{3}x + 5) \cdot 7$	3	9
20	$x^3 \cdot \frac{1}{3} + 21$	1	3

При выполнении задания преобразовывать (упрощать) выражения для $f(x)$ нельзя. При выполнении деления в качестве результата можно использовать только целую часть от деления и не учитывать остаток (т.е. 5 : 2 = 2).

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.

Например, определим переменную intg DD 3 – это означает, что задается область памяти размером 4 байта, адрес которой обозначен меткой intg. В таком случае, команда

mov eax, [intg]

копирует из памяти по адресу intg данные в регистр eax. В свою очередь команда

mov [intg], eax

запишет в память по адресу intg данные из регистра eax.

Также рассмотрим команду

mov eax,intg

В этом случае в регистр eax запишется адрес intg. Допустим, для intg выделена память начиная с ячейки с адресом 0x600144, тогда команда mov eax,intg аналогична команде mov eax,0x600144 – т.е. эта команда запишет в регистр eax число 0x600144.

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 <операнд_1>, <операнд_2>

Допустимые сочетания операндов для команды add аналогичны сочетаниям операндов для команды mov.

Так, например, команда add eax,ebx прибавит значение из регистра eax к значению из регистра ebx и запишет результат в регистр eax.

Примеры:

```
add ax,5      ; AX = AX + 5
add dx,cx     ; DX = DX + CX a
dd dx,cl      ; Ошибка: разный размер операндов.
```

3.2.2. Целочисленное вычитание sub

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

sub <операнд_1>, <операнд_2>

Так, например, команда sub ebx,5 уменьшает значение регистра ebx на 5 и записывает результат в регистр ebx.

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 <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.

Так, например, команда inc ebx увеличивает значение регистра ebx на 1, а команда inc ax уменьшает значение регистра ax на 1.

3.2.4. Команда изменения знака операнда neg

Еще одна команда, которую можно отнести к арифметическим командам это команда изменения знака neg:

neg <операнд>

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
```

3.2.5. 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>

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 6.1.

Таблица 6.1. Регистры используемые командами умножения в Nasm

Размер операнда	Неявный множитель	Результат умножения
1 байт	AL	AX
2 байта	AX	DX:AX
4 байта	EAX	EDX:EAX

Пример использования инструкции mul:

```
a dw 270

mov ax, 100     ; AX = 100
mul a           ; AX = AX*a,
mul bl          ; AX = AL*BL
mul ax          ; DX:AX = AX*AX
```

3.2.6. Команды деления div и idiv.

Для деления, как и для умножения, существует 2 команды div (от англ. divide - деление) 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 6.2.

Таблица 6.2. Регистры используемые командами деления в Nasm

Размер операнда (делителя)	Делимое	Частное	Остаток
1 байт	AX	AL	AH
2 байта	DX:AX	AX	DX
4 байта	EDX:EAX	EAX	EDX

Например, после выполнения инструкций

```
mov ax,31
mov dl,15
div dl
```

результат 2 (31/15) будет записан в регистр al, а остаток 1 (остаток от деления 31/15) — в регистр ah.

Если делитель — это слово (16-бит), то делимое должно записываться в регистрах dx:ax. Так в результате выполнения инструкций

```
mov ax,2          ; загрузить в регистровую
mov dx,1          ; пару `dx:ax` значение 10002h
mov bx,10h
div bx
```

в регистр ax запишется частное 1000h (результат деления 10002h на 10h), а в регистр dx — 2 (остаток от деления).

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. 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,).

• 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,).

4. Выполнение лабораторной работы

1. Создадим каталог для программам лабораторной работы № 6, перейдем в него и создадим файл lab6-1.asm:

Рис.1.Создание каталога

2. Рассмотрим примеры программ вывода символьных и численных значений. Программы будут выводить значения, записанные в регистр eax.

   Введем в файл lab6-1.asm текст программы из листинга 6.1:

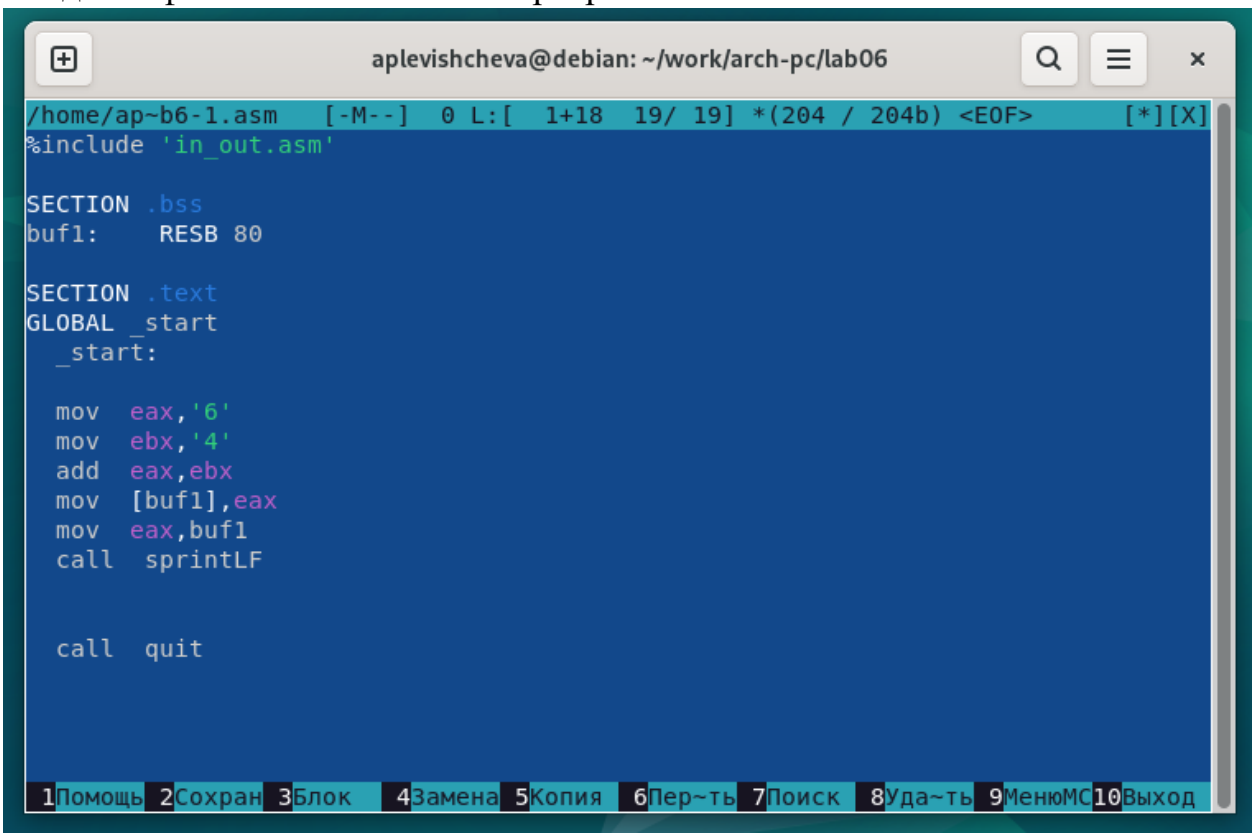

Рис.2.1.Текст программы lab6-1.asm

В данной программе в регистр eax записывается символ 6 (mov eax,'6'), в регистр ebx символ 4 (mov ebx,'4'). Далее к значению в регистре eax прибавляем значение регистра ebx (add eax,ebx, результат сложения запишется в регистр eax). Далее выводим результат. Так как для работы функции sprintLF в регистр eax должен быть записан адрес, необходимо

использовать дополнительную переменную. Для этого запишем значение регистра eax в переменную buf1 (mov [buf1], eax), а затем запишем адрес переменной buf1 в регистр eax (mov eax, buf1) и вызовем функцию sprintLF.

Создадим исполняемый файл и запустим его:

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

Рис.2.2.Исполняемый файл(1)

В данном случае при выводе значения регистра eax мы ожидаем увидеть число 10. Однако результатом будет символ j. Это происходит потому, что код символа 6 равен 00110110 в двоичном представлении (или 54 в десятичном представлении), а код символа 4 – 00110100(52). Команда add eax,ebx запишет в регистр eax сумму кодов – 01101010 (106), что в свою очередь является кодом символа j (см. таблицу ASCII в приложении).

3. Далее изменим текст программы и вместо символов, запишем в регистры числа.

Исправим текст программы (Листинг 6.1) следующим образом:

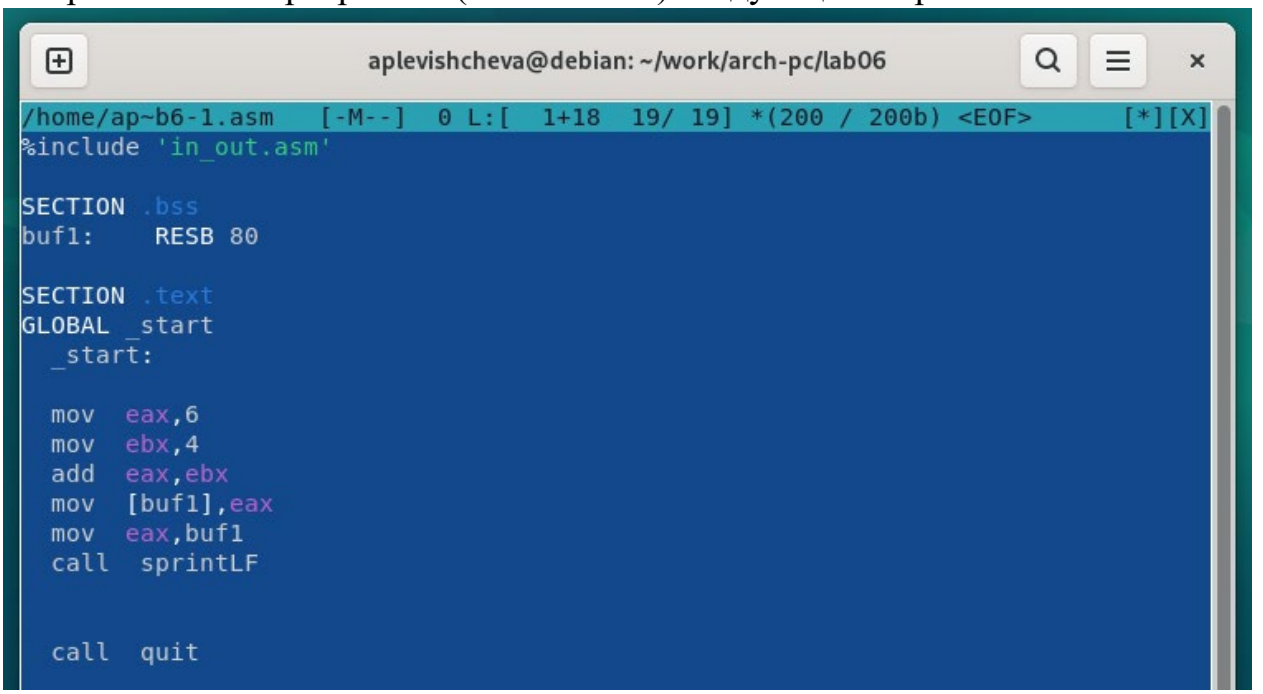

Рис.3.1.Внесение правок(1)

Создадим исполняемый файл и запустим его:

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


aplevishcheva@debian:~/work/arch-pc/lab06$
```

Рис.3.2.Исполняемый файл(2)

Как и в предыдущем случае при исполнении программы мы не получим число 10. В данном случае выводится символ с кодом 10.

Пользуясь таблицей ASCII, определим какому символу соответствует код 10:

10	12	0x0A	1010	LF, \n

Рис.3.3.Код 10

Этот символ не отображается, а создает новую строку.

4. Для работы с числами в файле in_out.asm реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Преобразуем текст программы из Листинга 6.1 с использованием этих функций.

Создадим файл lab6-2.asm в каталоге ~/work/arch-pc/lab06:

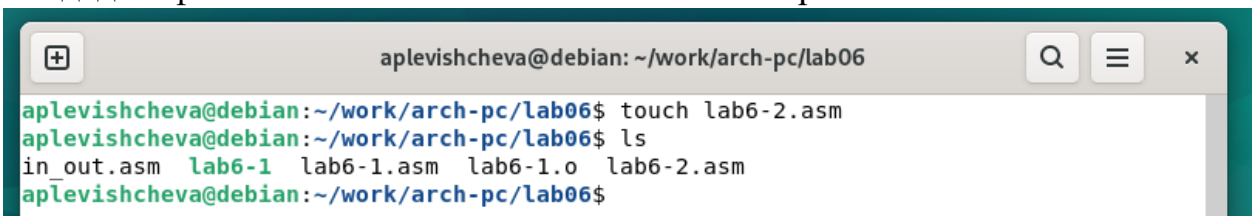

Рис.4.1.Файл lab6-2.asm

Введем в него текст программы из листинга 6.2:

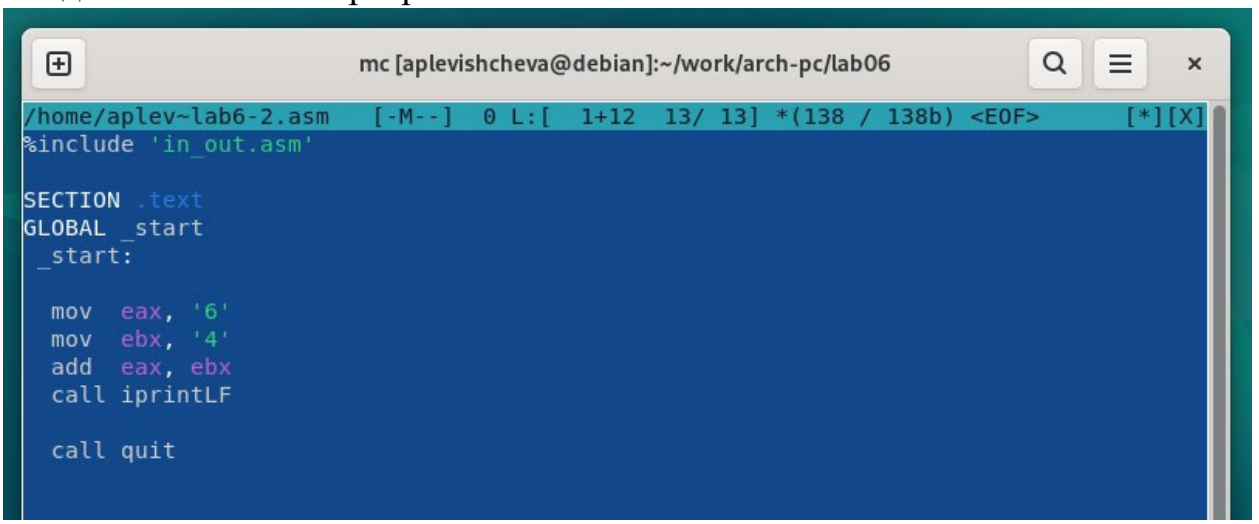

Рис.4.2.Текст программы lab6-2.asm

Создадим исполняемый файл и запустим его:

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

Рис.4.3.Исполняемый файл(3)

В результате работы программы мы получим 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.

5. Аналогично предыдущему примеру изменим символы на числа.

Исправим текст программы:

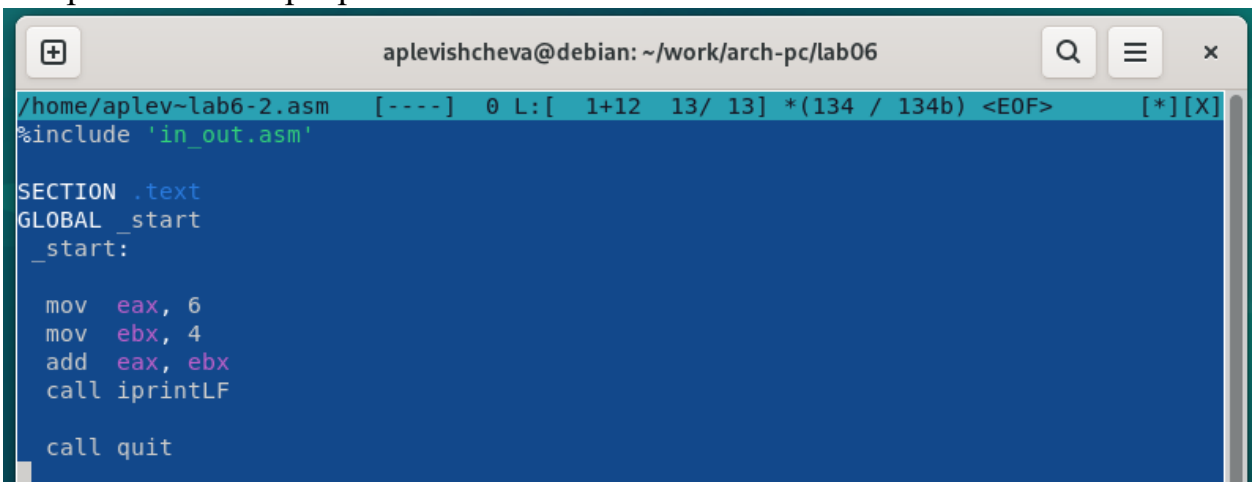

Рис.5.1.Внесение правок(2)

Создадим исполняемый файл и запустим его:

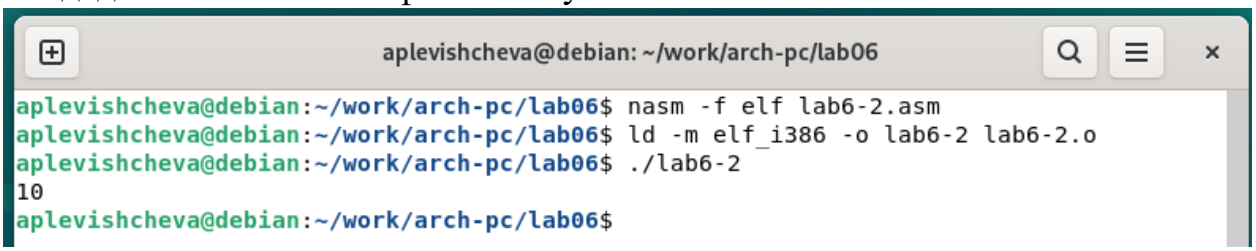

Рис.5.2.Исполняемый файл(4)

Результатом программы является число 10.

Заменим функцию iprintLF на iprint:

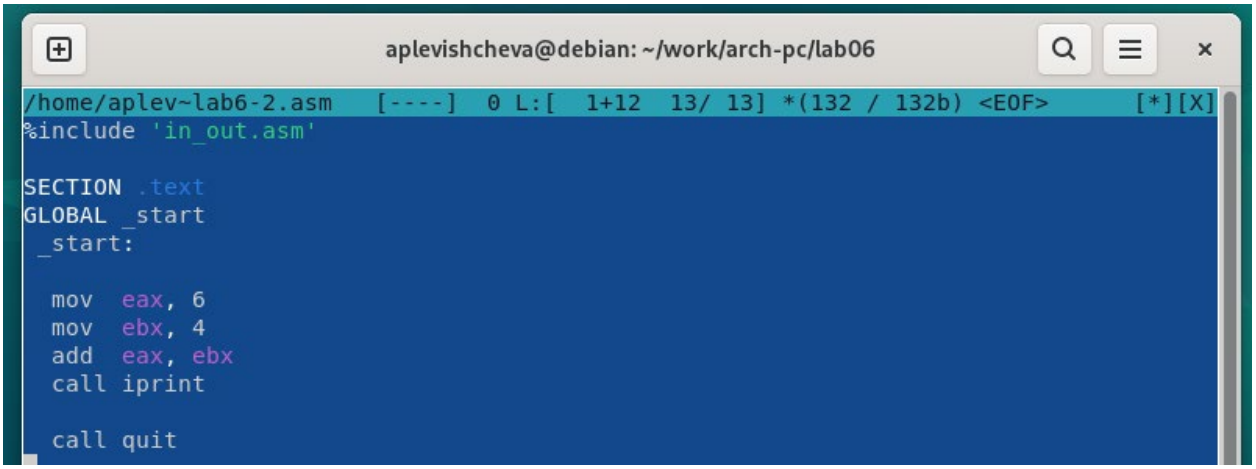

Рис.5.3.Замена

Создадим исполняемый файл и запустим его:

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

Рис.5.4.Исполняемый файл(5)

Отличие функции iprint от функции iprintLF в том, что она не выполняет перенос на новую строку после выполнения программы.

6. В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения $f(x) = (5 * 2 + 3)/3$.

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

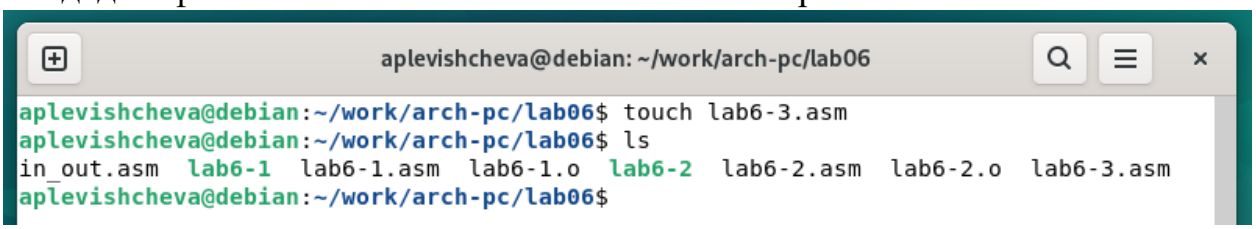

Рис.6.1.Файл lab6-3.asm

Внимательно изучим текст программы из листинга 6.3 и введем в lab6-3.asm:

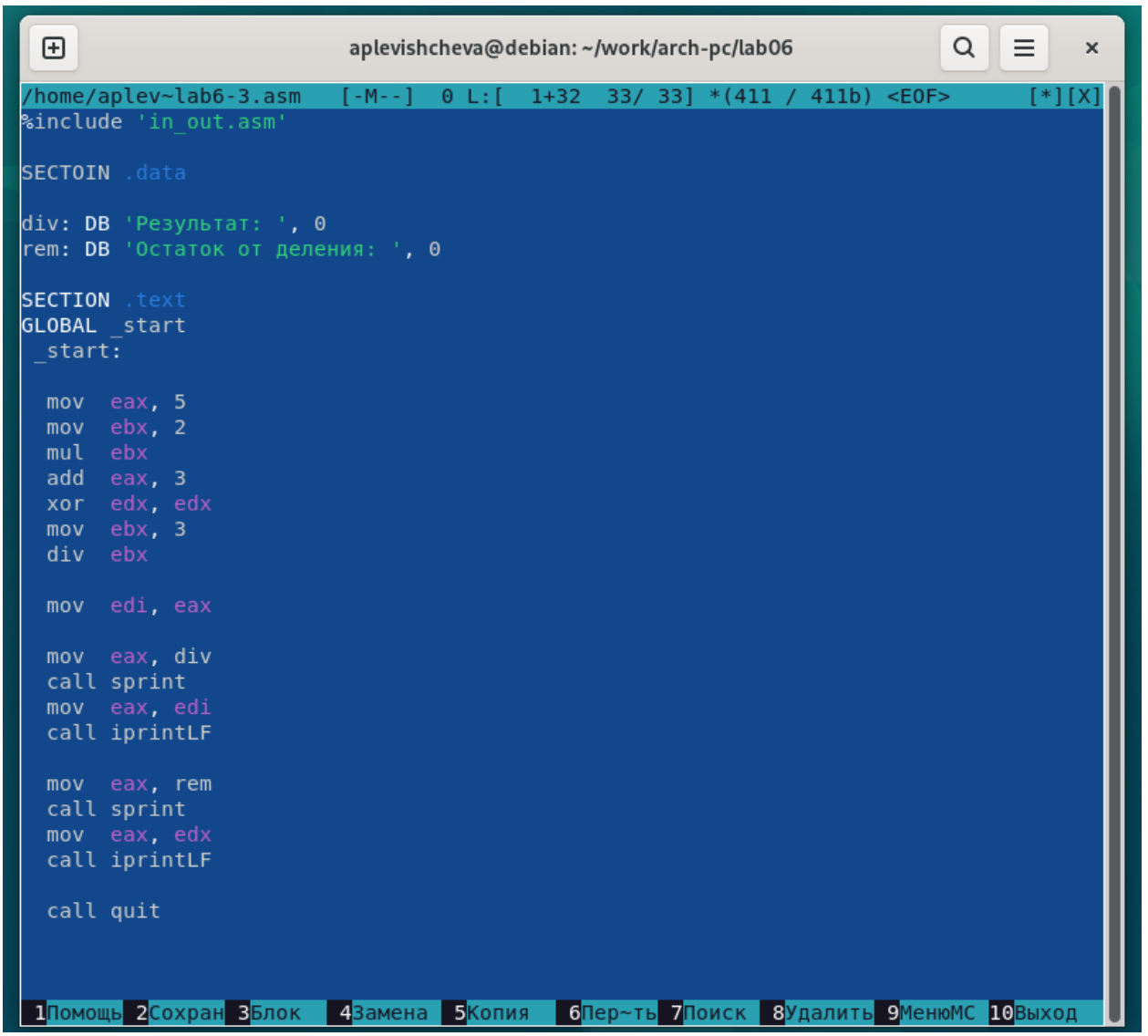

Рис.6.2.Текст программы lab6-3.asm

Создадим исполняемый файл и запустим его:

Рис.6.3.Исполняемый файл(6)

Результат работы программы верен.

Изменим текст программы для вычисления выражения $f(x) = (4 * 6 + 2)/5$:

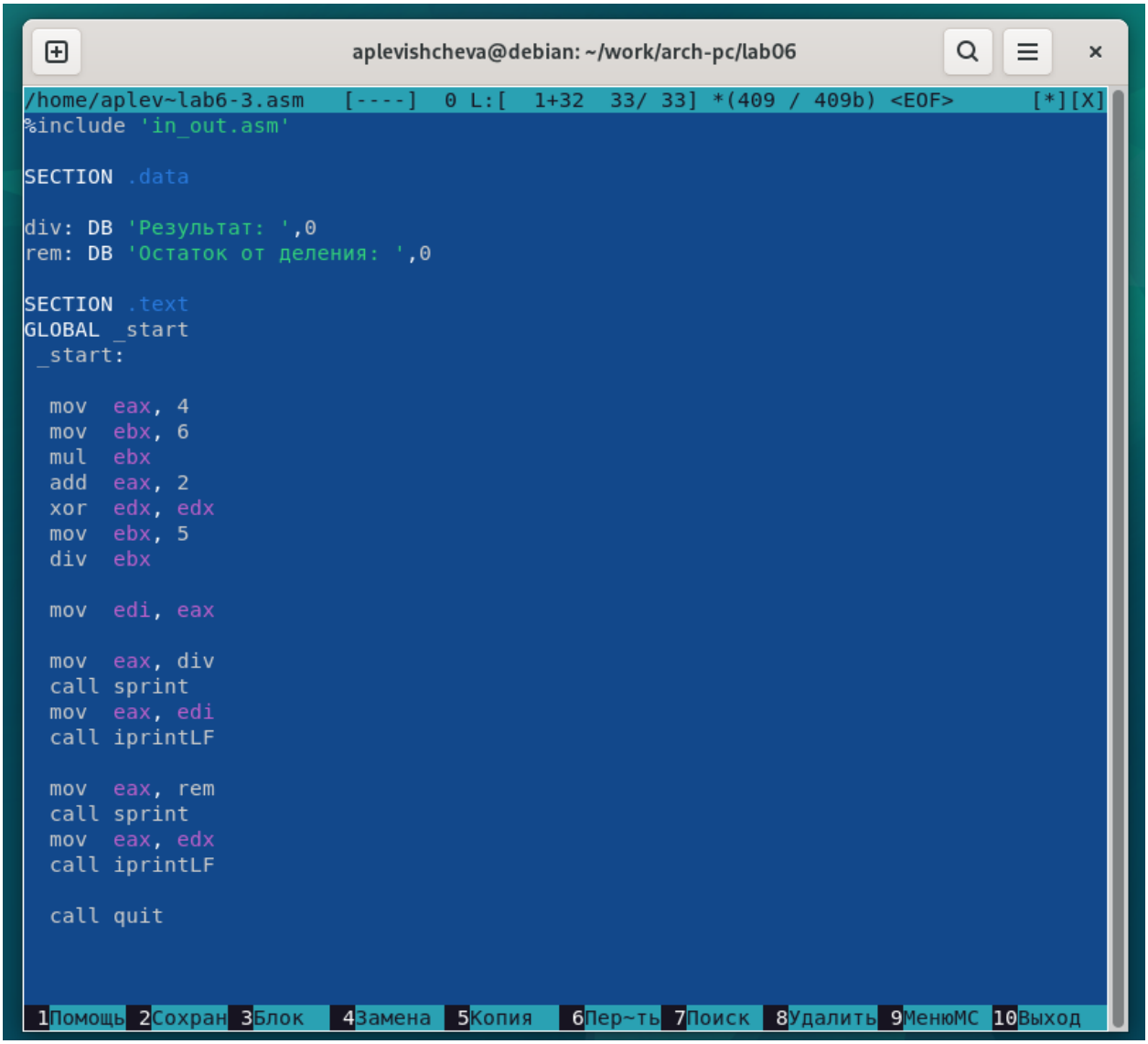

Рис.6.4.Внесение правок(3)

Создадим исполняемый файл и проверим его работу:

```
aplevishcheva@debian:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
aplevishcheva@debian:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
aplevishcheva@debian:~/work/arch-pc/lab06$ ./lab6-3
Результат: 5
Остаток от деления: 1
aplevishcheva@debian:~/work/arch-pc/lab06$
```

Рис.6.5.Исполняемый файл(7)

7. В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета, работающую по следующему алгоритму:
    - вывести запрос на введение № студенческого билета
    - вычислить номер варианта по формуле: $(S(n) \bmod 20) + 1$, где $S(n)$ – номер студенческого билета (В данном случае $a \bmod b$ – это остаток от деления a на b).
    - вывести на экран номер варианта

В данном случае число, над которым необходимо проводить арифметические операции, вводится с клавиатуры. Как отмечалось выше ввод с клавиатуры осуществляется в символьном виде и для корректной работы арифметических операций в NASM символы необходимо преобразовать в числа. Для этого может быть использована функция atoi из файла in_out.asm.

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

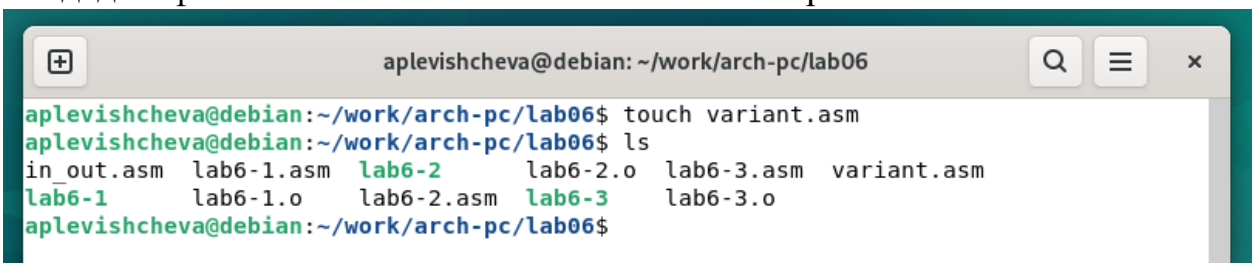

Рис.7.1.Файл variant.asm

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

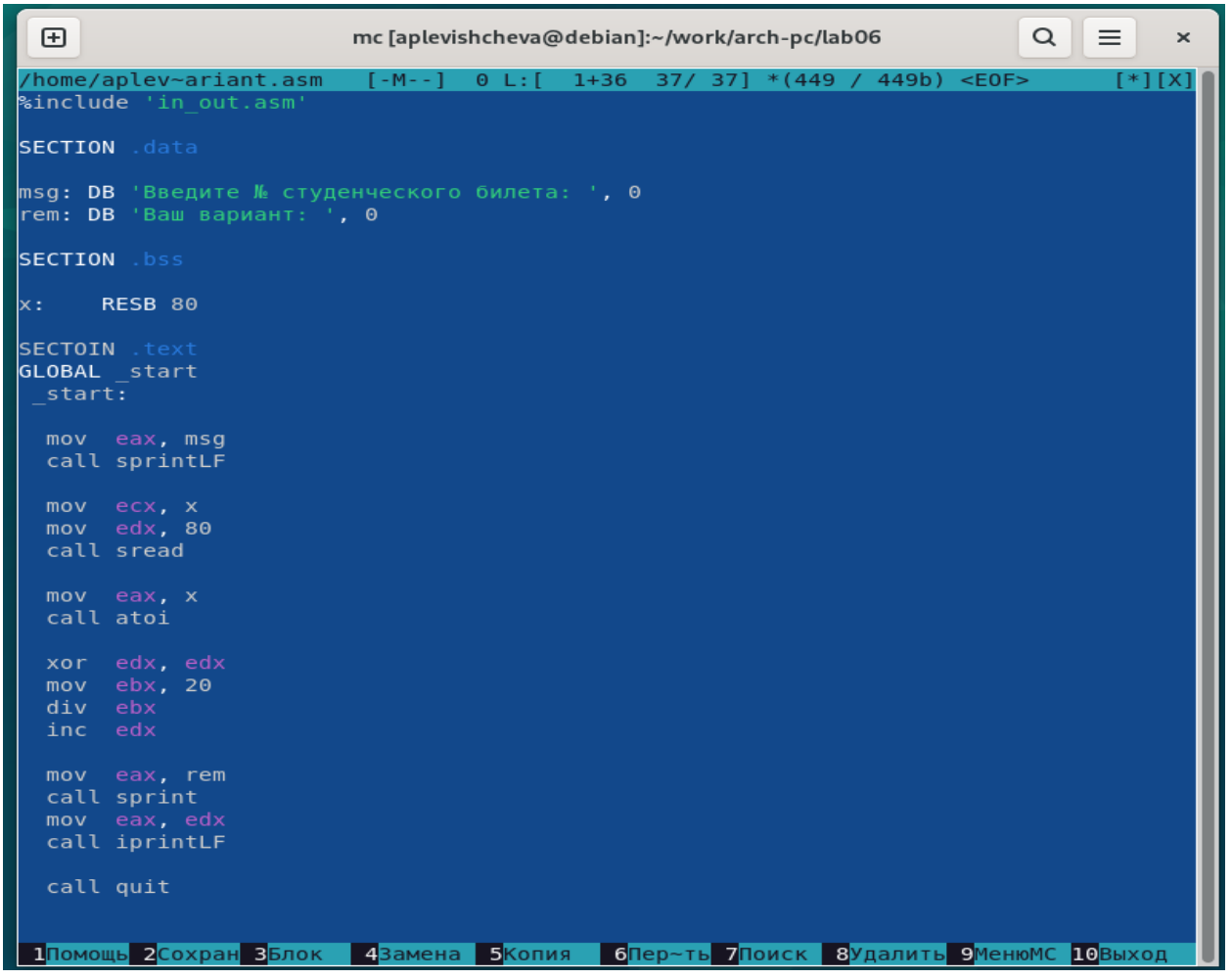

Рис.7.2.Текст файла variant.asm

Создадим исполняемый файл и запустим его:

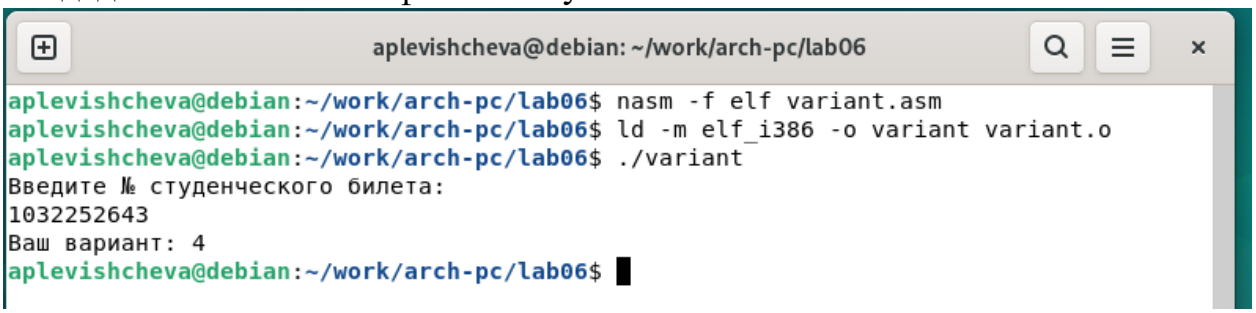

Рис.7.3.Исполняемый файл(8)

Проверим результат работы программы вычислив номер варианта аналитически. Результат верен.

Ответы на вопросы:

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

   За вывод отвечают строки:
   mov eax, rem
   call sprint

2) Для чего используется следующие инструкции?
   mov ecx, x
   mov edx, 80
   call sread

   1 строка – загрузить в ecx указатель на буфер x
   2 строка – положить в edx максимальную длину для чтения
   3 строка – вызвать процедуру чтения строки с клавиатуры

3) Для чего используется инструкция “call atoi”?

   «call atoi» вызывает функцию, которая преобразует ASCII-строку в целое число. Результат возвращается в регистр eax.

4) Какие строки листинга 6.4 отвечают за вычисления варианта?

   xor edx, adx
   mov ebx, 20
   div ebx
   inc edx

5) В какой регистр записывается остаток от деления при выполнении инструкции “div ebx”?

   В регистр edx.

6) Для чего используется инструкция “inc edx”?

   Увеличивает значение edx на 1.

7) Какие строки листинга 6.4 отвечают за вывод на экран результата вычислений?

   ```
   mov  eax, adx
   call  iprintLF
   ```

Задания для самостоятельной работы.

Задание 1(стр.6):

В ходе лабораторной работы был вычислен мой вариант – 4.

Создадим файл var4.asm в каталоге lab06:

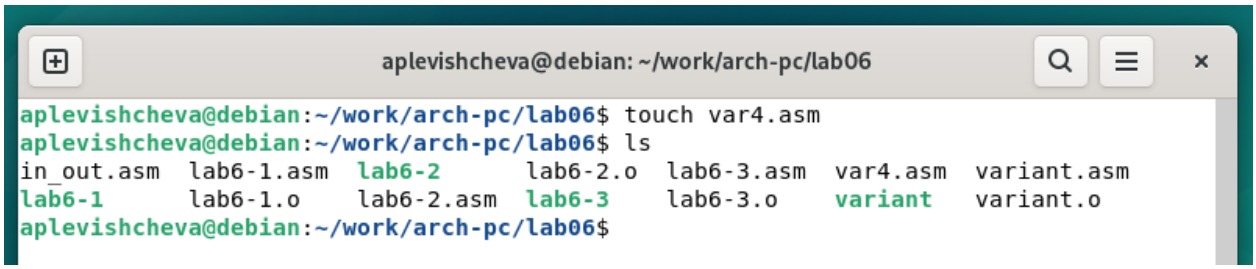

Рис.8.1.Файл var4.asm

Напишем текст программы для f(x) = 4/3(x-1)+5:

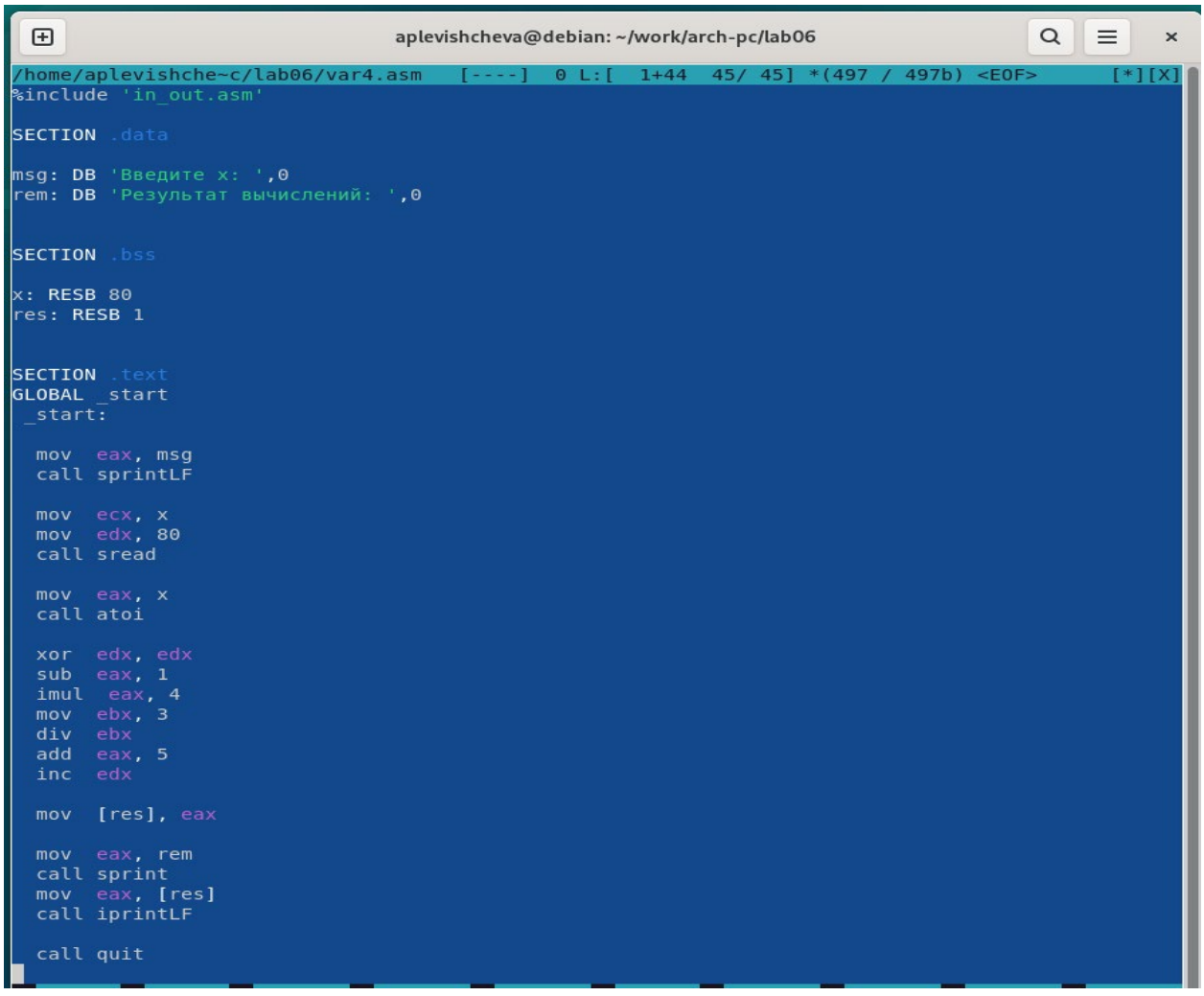

Рис.8.2.Текст программы var4.asm

Создадим исполняемый файл и проверим его работу при x1 = 4 и x2 = 10:

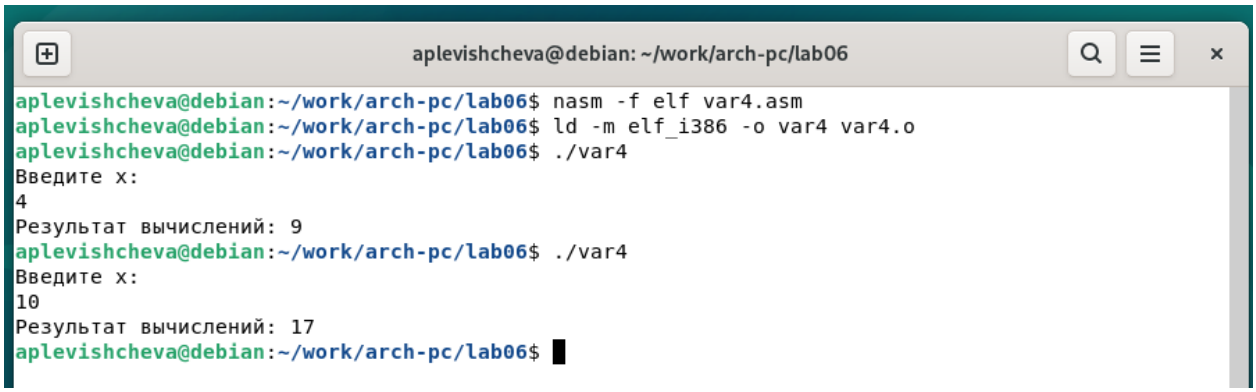

Рис.8.3.Исполняемый файл(9)

Проверим правильность выполнения программы аналитически. Результаты совпадают.

5. Выводы

В ходе выполнения лабораторной работы были успешно изучены и применены на практике арифметические инструкции языка ассемблера NASM.

Список литературы

1. GDB: The GNU Project Debugger. — URL: https://www.gnu.org/software/gdb/.

2. GNU Bash Manual. — 2016. — URL: https://www.gnu.org/software/bash/manual/.

3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. org/.

4. NASM Assembly Language Tutorials. — 2021. — URL: https://asmtutor.com/.

5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658.

6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.

7. The NASM documentation. — 2021. — URL: https://www.nasm.us/docs.php.

8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.

9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.

10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.

11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.

12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.

13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.

14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).

16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).