Российский Университет Дружбы Народов

Факультет физико-математических и естественных наук

Отчет по лабораторной работе №6

“Арифметические операции в NASM”

Студентка: Богомолова Полина Петровна

Группа: НКАбд-01-25

Москва 2025

Оглавление

Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

Теоретическое введение

Адресация в NASM

Большинство инструкций на языке ассемблера требует 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.

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

add dx,cl ; Ошибка: разный размер операндов.

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.

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.

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

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. Регистры, используемые командами умножения в NASM представлены на рисунке 1.

Рис. 1

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

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, представленные на рисунке 2.

Рис. 2

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

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 символу перевода строки.

• atoi – функция преобразует ascii-код символа в целое число и записает результат

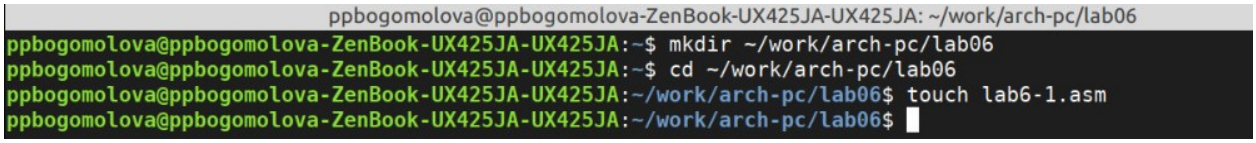
в регистр eax, перед вызовом atoi в регистр eax необходимо записать число (mov

eax,<int>).

Порядок выполнения лабораторной работы

1. Создадим каталог для программ лабораторной работы № 6, перейдем в него и создадим файл lab6-1.asm. Для этого будем использовать команды mkdir, cd, touch. Результат представлен на рисунке 3.

Рис. 3



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. 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. Для этого будем использовать команды nasm -f elf, ld -m elf_i386, ./

Предварительно создадим копию файла in_out.asm в каталоге ~/work/arch-pc/lab06, используя команду cp. Для работы с файлами перейдем в Midnight Commander с помощью команды mc. Результат представлен на рисунках 4-8.

Рис. 4

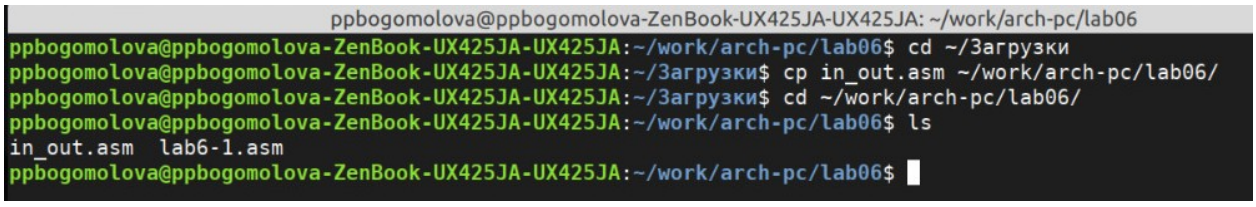


Рис. 5

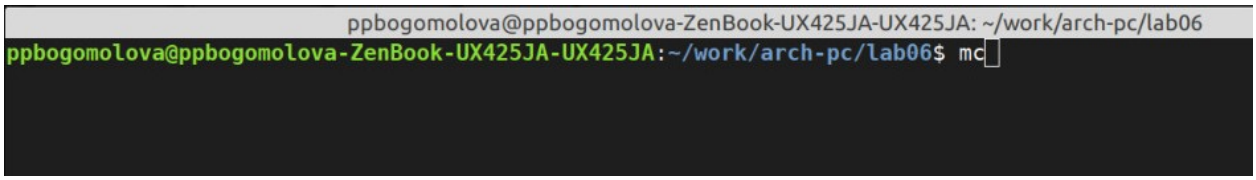

ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA: ~/work/arch-pc/lab06
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab06$ mc

Рис. 6

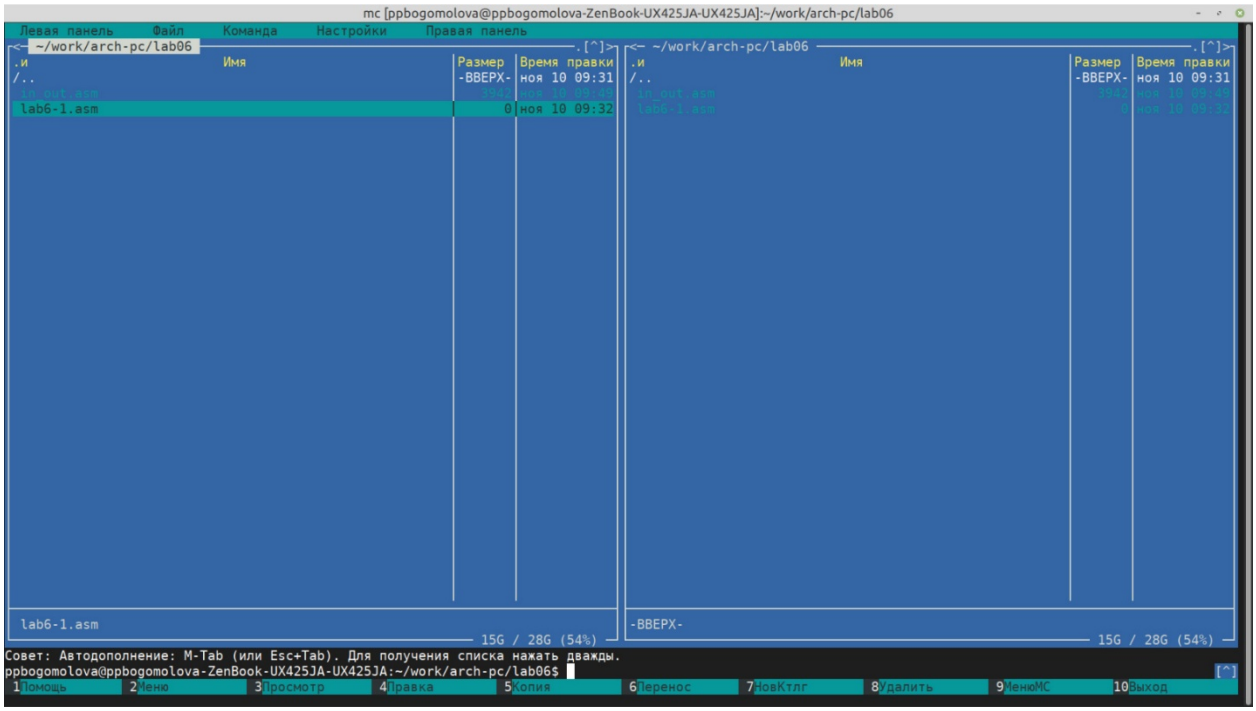

mc [ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA]:~/work/arch-pc/lab06
Левая панель Файл Команда Настройки Правая панель
~/work/arch-pc/lab06
.и Имя Размер Время правки
/.. -ВВЕРХ- ноя 10 09:31
in_out.asm 3743 ноя 10 09:45
lab6-1.asm 0 ноя 10 09:32
lab6-1.asm
15G / 28G (54%)
~/work/arch-pc/lab06
-ВВЕРХ-
Совет: Автодополнение: M-Tab (или Esc+Tab). Для получения списка нажать дважды.
ppbogomolova@ppbogomolova-ZenBook-UX425JA-UX425JA:~/work/arch-pc/lab06$
1Помощь 2Меню 3Просмотр 4Правка 5Копия 6Перенос 7НовКтлг 8Удалить 9МенюMC 10Выход

Рис. 7

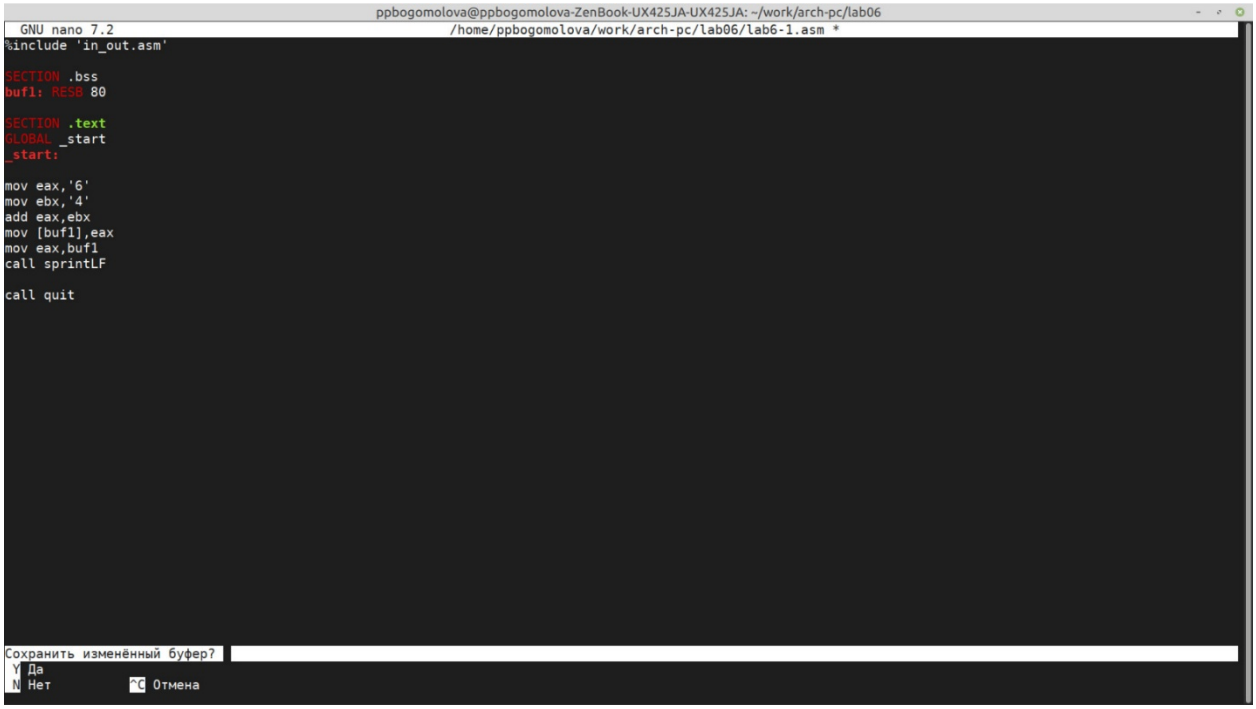
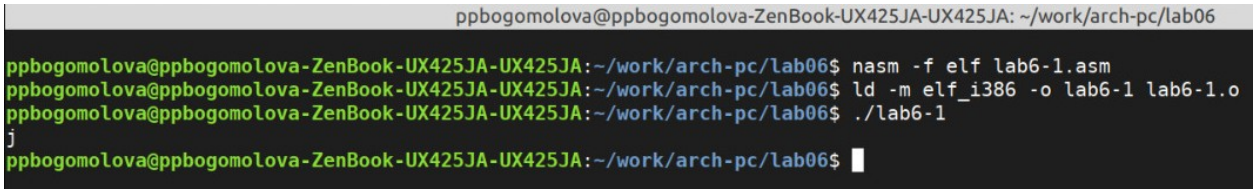


Рис. 8



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.

3. Далее изменим текст программы из листинга 6.1 и вместо символов, запишем в регистры числа. Исправим текст программы, заменив строки

   mov eax,'6'

   mov ebx,'4'

   на строки

mov eax,6

mov ebx,4

Создадим исполняемый файл и запустим его. Для этого будем использовать команды nasm -f elf, ld -m elf_i386, ./ .Для работы с файлами перейдем в Midnight Commander с помощью команды mc.

Как и в предыдущем случае при исполнении программы мы не получим число 10. В данном случае выводится символ с кодом 10. Код 10 в таблице ASCII соответствует символу LF (Line Feed), отвечающему за переход на новую строку. Этот символ не отображается на экране, а лишь выполняет свою функцию. Результат представлен на рисунках 9-10.

Рис. 9

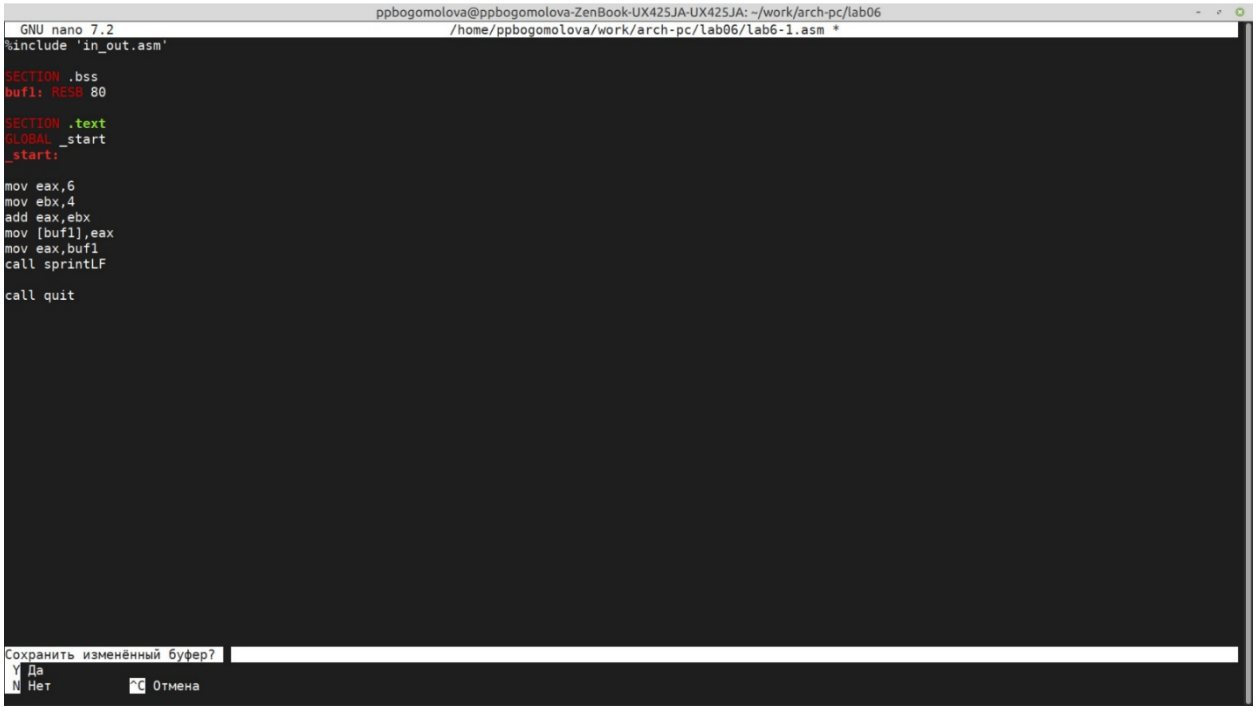
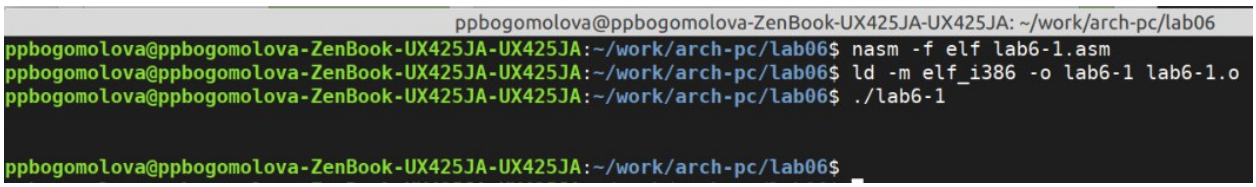


Рис. 10



4. Как отмечалось выше, для работы с числами в файле in_out.asm реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Преобразуем текст программы 6.1 с использованием этих функций. Создадим файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 и введем в него текст программы из листинга 6.2 . Для создания файла используем команду touch. Создадим исполняемый файл и запустим его. Для этого используем команды nasm -f elf, ld -m elf_i386, ./ . Для работы с файлами перейдем в Midnight Commander с помощью команды mc.

Результат представлен на рисунках 11-13.

Рис. 11

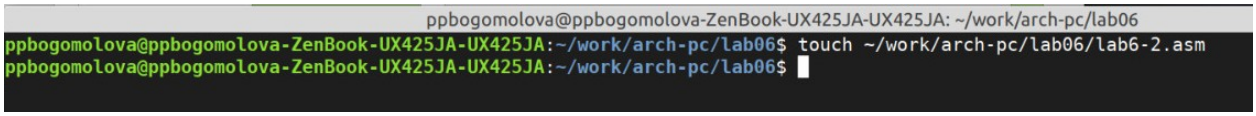


Рис. 12

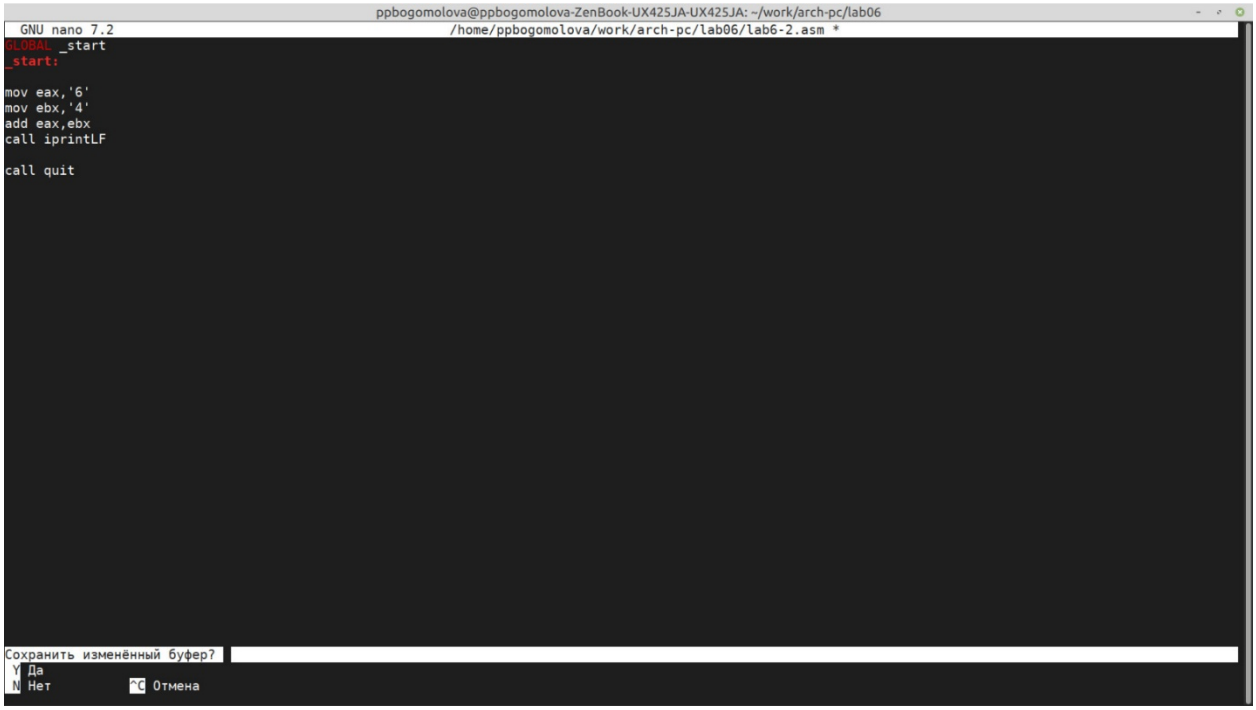
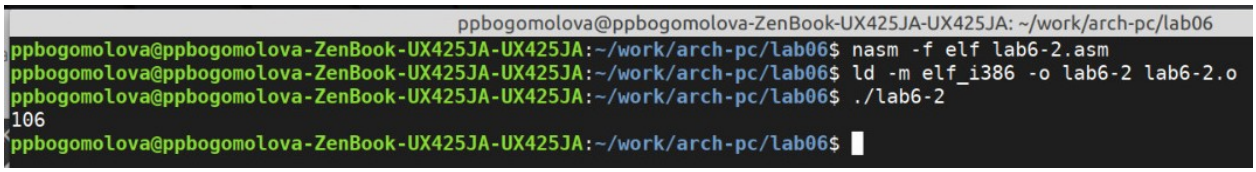


Рис. 13



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. Замените строки

   mov eax,'6'

   mov ebx,'4'

   на строки

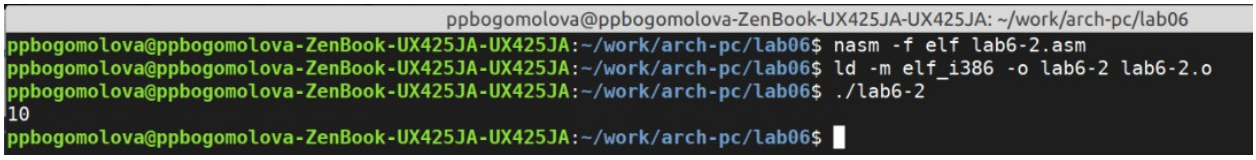
   mov eax,6

   mov ebx,4

   Создадим исполняемый файл и запустим его. Для этого будем использовать команды nasm -f elf, ld -m elf_i386, ./ . Для работы с файлами перейдем в Midnight Commander с помощью команды mc. Результат представлен на рисунках 14-15.

Рис. 14

Рис. 15

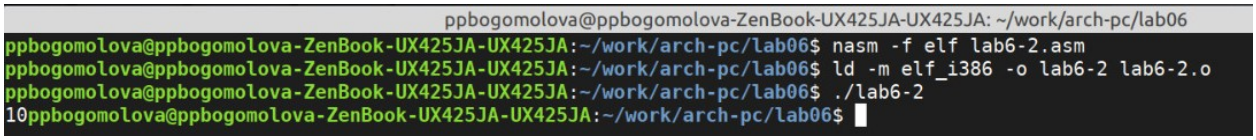


В данном варианте программы в регистры EAX и EBX помещаются не символы, а числа 6 и 4. В отличие от записей вида '6' и '4', которые представляют собой ASCII-коды 54 и 52, запись без кавычек означает числовые значения. Поэтому при выполнении операции сложения выполняется обычная арифметика: 6 + 4, результатом которой является число 10. Именно это число и выводится на экран.

Заменим функцию iprintLF на iprint. Создадим исполняемый файл и запустим его. Для этого будем использовать команды nasm -f elf, ld -m elf_i386, ./ . Для работы с файлами перейдем в Midnight Commander с помощью команды mc. Результат представлен на рисунках 16-17.

Рис. 16

Рис. 17



Вывод функции iprintLF отличается от вывода функции iprint наличием символа LF, отвечающим за переход на новую строку. То есть iprint выводит число без перехода на новую строку, а iprintLF выводит число и выполняет переход на новую строку.

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: touch ~/work/arch-pc/lab06/lab6-3.asm Внимательно изучитим текст программы из листинга 6.3 и введем в lab6-3.asm.

   Создадим исполняемый файл и запустим его. Для этого будем использовать команды nasm -f elf, ld -m elf_i386, ./ . Для работы с файлами перейдем в Midnight Commander с помощью команды mc. Результат представлен на рисунках 18-19.

Рис. 18

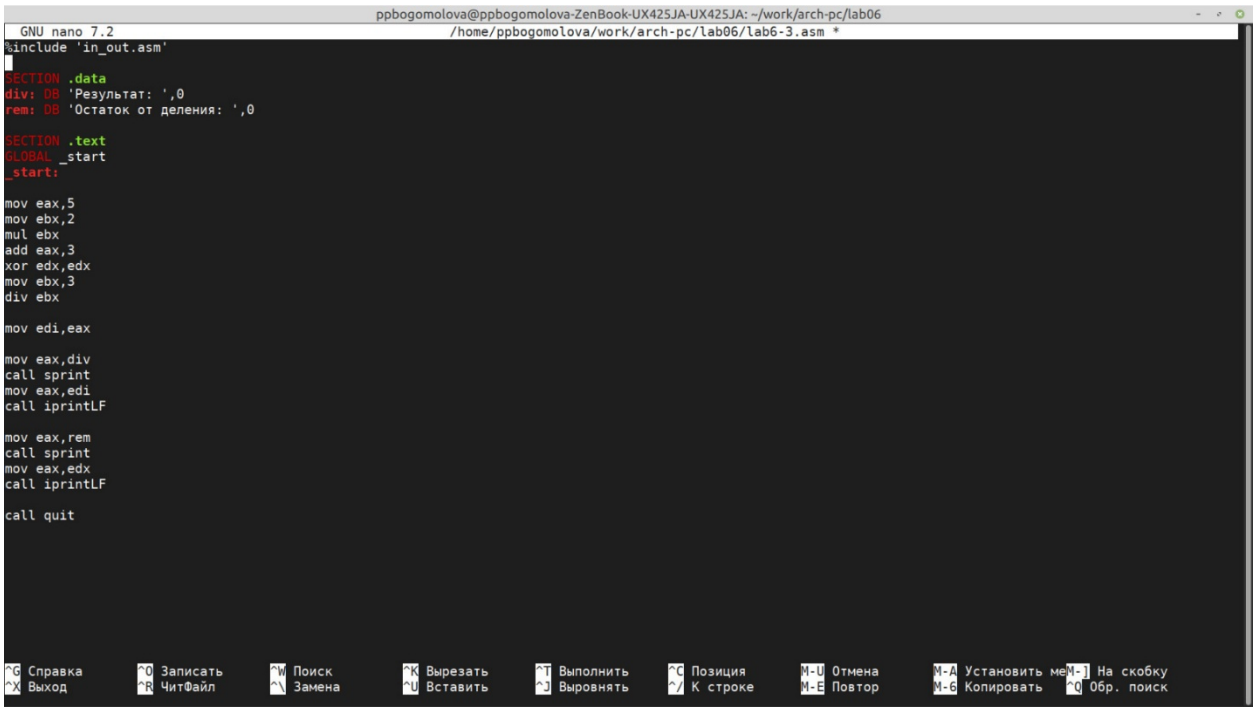
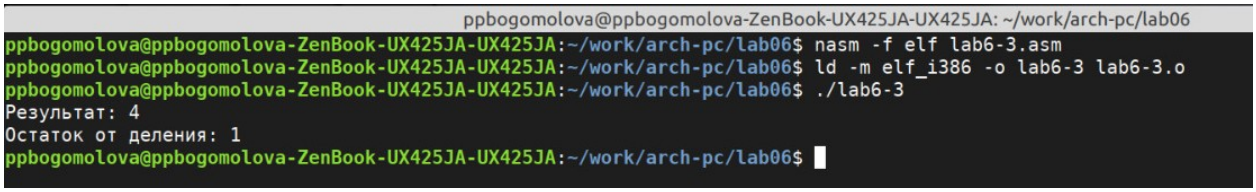


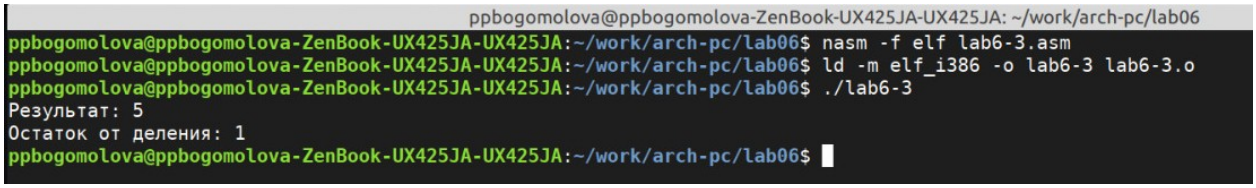
Рис. 19



Изменим текст программы для вычисления выражения $f(x) = (4 * 6 + 2)/5$. Создадим исполняемый файл и проверим его работу. Для этого будем использовать команды nasm -f elf, ld -m elf_i386, ./ . Для работы с файлами перейдем в Midnight Commander с помощью команды mc. Результат представлен на рисунках 20-21.

Рис. 20

Рис. 21



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: (Sn mod 20) + 1, где Sn – номер студенческого билета (в данном случае a mod 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: touch ~/work/arch-pc/lab06/variant.asm. Создадим исполняемый файл и проверим его работу. Для этого будем использовать команды nasm -f elf, ld -m elf_i386, ./ . Для работы с файлами перейдем в Midnight Commander с помощью команды mc. Результат представлен на рисунках. Внимательно изучим текст программы из листинга 6.4 и введем в файл variant.asm. Результат представлен на рисунках 22-24.

Рис. 22

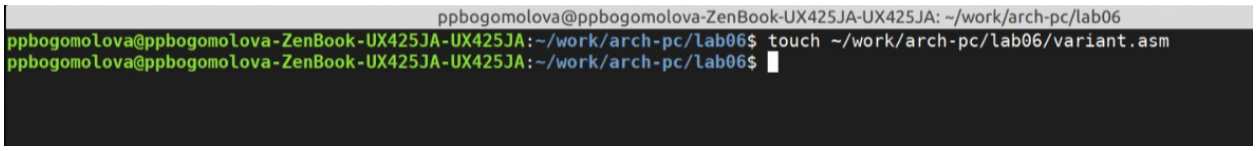
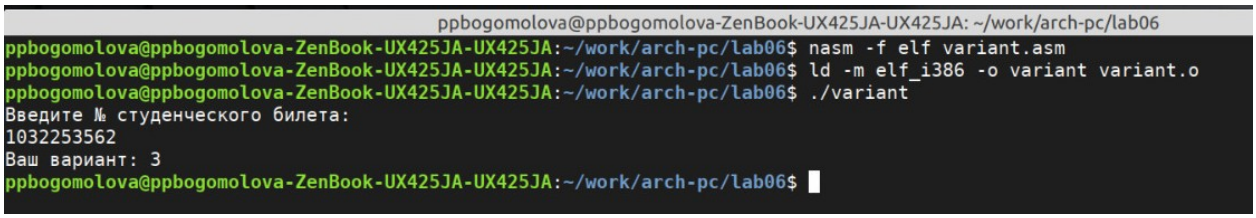


Рис. 23

Рис. 24



При вводе номера студенческого билета 1032253562 программа сначала считывает эту строку и преобразовывает её в число. Затем для определения номера варианта выполняется деление этого числа на 20, так как всего 20 вариантов. После деления получается остаток, который показывает, какое место в цикле из 20 вариантов соответствует данному номеру студенческого билета. Остаток равен 2. Т.к. нумерация вариантов начинается с 1, к остатку добавляется 1. Номер варианта: 3.

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

   Сообщение «Ваш вариант:» выводится за счёт двух команд:

   mov eax,rem
   call sprint

   Первая из них помещает в регистр EAX адрес строки rem, содержащей текст «Ваш вариант:», а вторая вызывает функцию sprint, которая выводит эту строку на экран.

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

   Эти строки подготавливают ввод с клавиатуры. Команда mov ecx, x указывает адрес буфера, куда будет записан ввод, mov edx, 80 задаёт максимальное количество вводимых символов, а вызов sread считывает

строку, введённую пользователем, и сохраняет её в буфере x. Таким образом, эти команды обеспечивают ввод номера студенческого билета.

3) Для чего используется инструкция “call atoi”?

Команда call atoi преобразует строку, введённую пользователем, в целое число. После выполнения этой команды числовое значение вводимого номера студенческого билета оказывается в регистре EAX, что позволяет использовать его в арифметических операциях.

4) Какие строки листинга 6.4 отвечают за вычисления варианта?

За вычисление номера варианта отвечает следующий фрагмент:

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

Здесь xor edx,edx обнуляет регистр EDX перед делением. Команда mov ebx,20 задаёт делитель, то есть количество вариантов. Инструкция div ebx выполняет деление введённого номера на 20: частное помещается в EAX, а остаток - в EDX. Наконец, команда inc edx увеличивает остаток на 1, превращая диапазон 0-19 в диапазон 1–20 - то есть корректный номер варианта.

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

При выполнении деления div ebx остаток записывается в регистр EDX, а частное - в регистр EAX.

6) Для чего используется инструкция “inc edx”?

Команда inc edx увеличивает остаток от деления на единицу. Это нужно потому, что остаток после деления может быть от 0 до 19, а варианты нумеруются от 1 до 20. Прибавление единицы обеспечивает корректный номер варианта.

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

Результат (номер варианта) выводится следующими строками:

```
mov eax,edx
call iprintLF
```

Первая команда помещает вычисленный номер варианта в регистр EAX, откуда функция iprintLF может его вывести. Затем вызов iprintLF печатает число и делает переход на новую строку.

Ссылка на мой репозиторий в GitHub:

https://github.com/bogomolova-pp/study_2025-2026_arh-pc

Задание для самостоятельной работы

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 1 вариантов заданий в соответствие с номером, полученным при выполнении лабораторной работы.

Таблица 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

Создадим файл lab6-4.asm с помощью команды touch. Создадим исполняемый файл и проверим его работу. Для этого будем использовать команды nasm -f elf, ld -

m elf_i386, ./ . Для работы с файлами перейдем в Midnight Commander с помощью команды mc. Результат представлен на рисунках 25-27.

Рис. 25

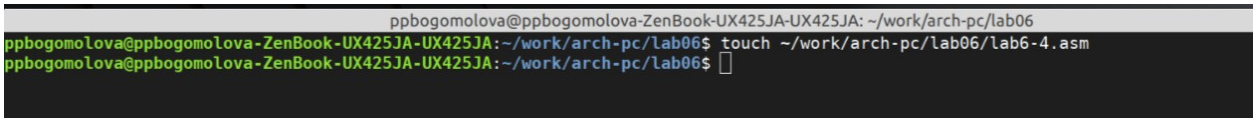


Рис. 26

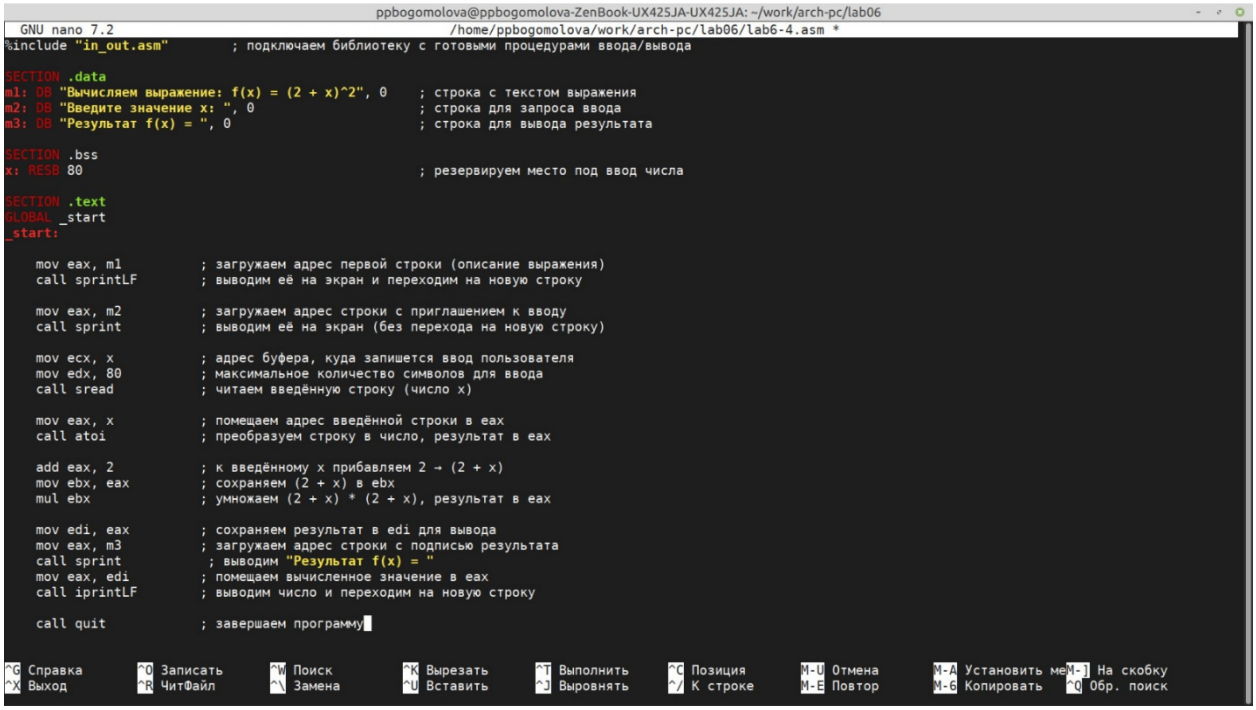


Рис. 27

Вывод

В результате выполнения лабораторной работы я освоила арифметические инструкции языка ассемблера NASM.

Список литературы

1. Демидова А.В - Лабораторная работа №6. Арифметические операции в NASM.