

Отчет по лабораторной работе 7

Арифметические операции в NASM

Гисматуллин Артём Вадимович НПИбд-01-22

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
3.1	Адресация в NASM	7
3.2	Арифметические операции в NASM	8
3.2.1	Целочисленное сложение add.	8
3.2.2	Целочисленное вычитание sub.	8
3.2.3	Команды инкремента и декремента.	8
3.2.4	Команда изменения знака операнда neg.	9
3.2.5	Команды умножения mul и imul.	9
3.2.6	Команды деления div и idiv.	9
3.3	Перевод символа числа в десятичную символьную запись	10
4	Выполнение лабораторной работы	12
5	Выполнение заданий для самостоятельной работы.	21
6	Выводы	24
	Список литературы	25

Список иллюстраций

4.1	Командная строка. каталог ~/work/arch-pc/lab07	12
4.2	Командная строка. Выполнение программы lab7-1.asm	13
4.3	Командная строка. Выполнение измененной программы lab7-1.asm	13
4.4	Командная строка. Выполнение программы lab7-2.asm	14
4.5	Командная строка. Выполнение измененной программы lab7-2.asm	15
4.6	Командная строка. Выполнение измененной программы lab7-2.asm 2	16
4.7	Командная строка. Выполнение программы lab7-3.asm	17
4.8	Командная строка. Выполнение измененной программы lab7-3.asm	18
4.9	Командная строка. Выполнение программы variant.asm	19
5.1	Командная строка. Код программы lab7-4.asm	22
5.2	Командная строка. Выполнение программы lab7-4.asm	23

Список таблиц

1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

2 Задание

1. Рассмотреть примеры программ вывода символьных и числовых значений (программы lab7-1.asm и lab7-2.asm).
2. Выполнить некоторые арифметические операции в NASM, вычислить несколько арифметических выражений (программа lab7-3.asm).
3. Вычислить вариант выражения по своему номеру студенческого билета по определенному алгоритму (программа variant.asm).
4. Ответить на вопросы.
5. Вычислить арифметическое выражение, вариантом которого является результат выполнения алгоритма по номеру студенческого билета.

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.

Существует три основных способа адресации:

1. Регистровая адресация – операнды хранятся в регистрах и в команде используются имена этих регистров, например: `mov ax,bx`.
2. Непосредственная адресация – значение операнда задается непосредственно в команде, Например: `mov ax,2`.
3. 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` анало-

гична 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` 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`. 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.

3.2.2 Целочисленное вычитание `sub`.

Команда целочисленного вычитания `sub` работает аналогично команде `add` и выглядит следующим образом: `sub` , 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` и `dec`, 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.

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

3.2.5 Команды умножения mul и imul.

Умножение и деление, в отличие от сложения и вычитания, для знаковых и беззнаковых чисел производиться по-разному, поэтому существуют различные команды. Для беззнакового умножения используется команда mul, а для знакового - 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.

3.2.6 Команды деления div и idiv.

Для деления, как и для умножения, существует 2 команды div 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. 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, 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 делает 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 под-

программы для преобразования ASCII символов в числа и обратно. Это:

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

4 Выполнение лабораторной работы

1. Перешли в каталог `~/work/arch-pc/lab07` и создали файл `lab7-1.asm`. (рис. 4.1)

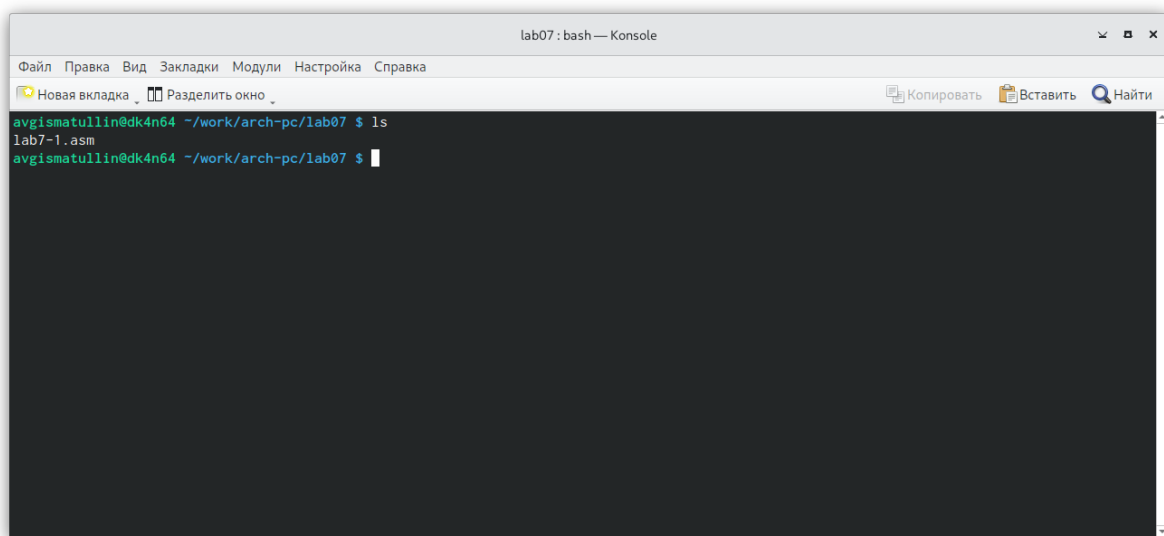
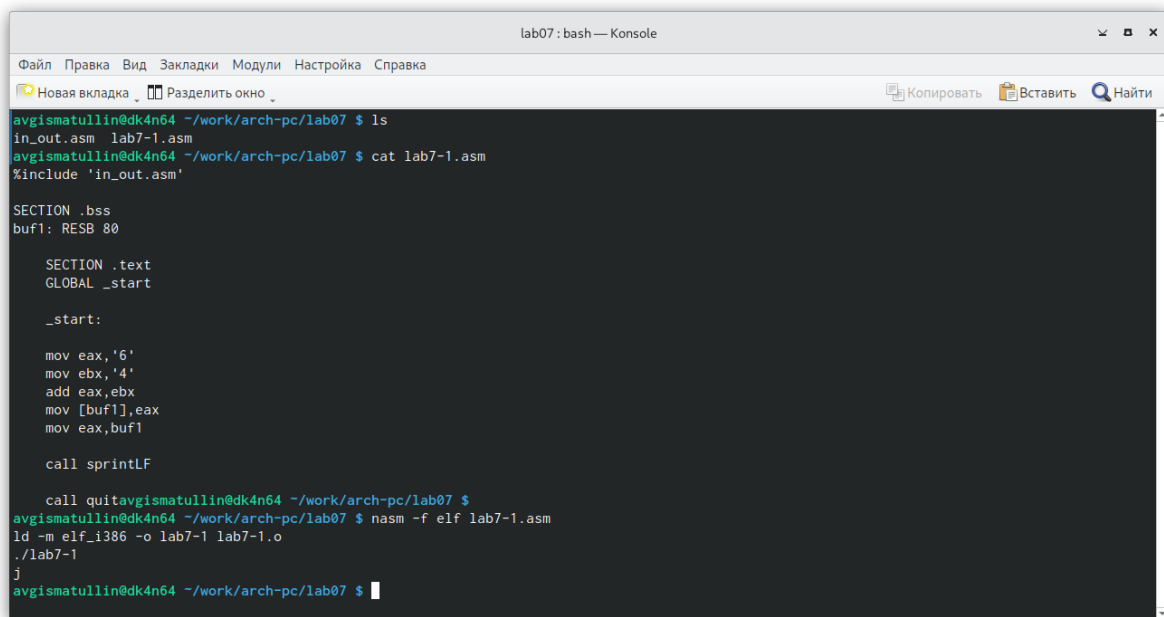


Рис. 4.1: Командная строка. каталог `~/work/arch-pc/lab07`

2. Ввели в файл `lab7-1.asm` текст программы листинга, создали исполняемый файл и посмотрели результат программы. (рис. 4.2)



```
lab07: bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно
Копировать  Вставить  Найти

avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ ls
in_out.asm  lab7-1.asm
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ cat lab7-1.asm
#include 'in_out.asm'

SECTION .bss
buf1: RESB 80

SECTION .text
GLOBAL _start

_start:

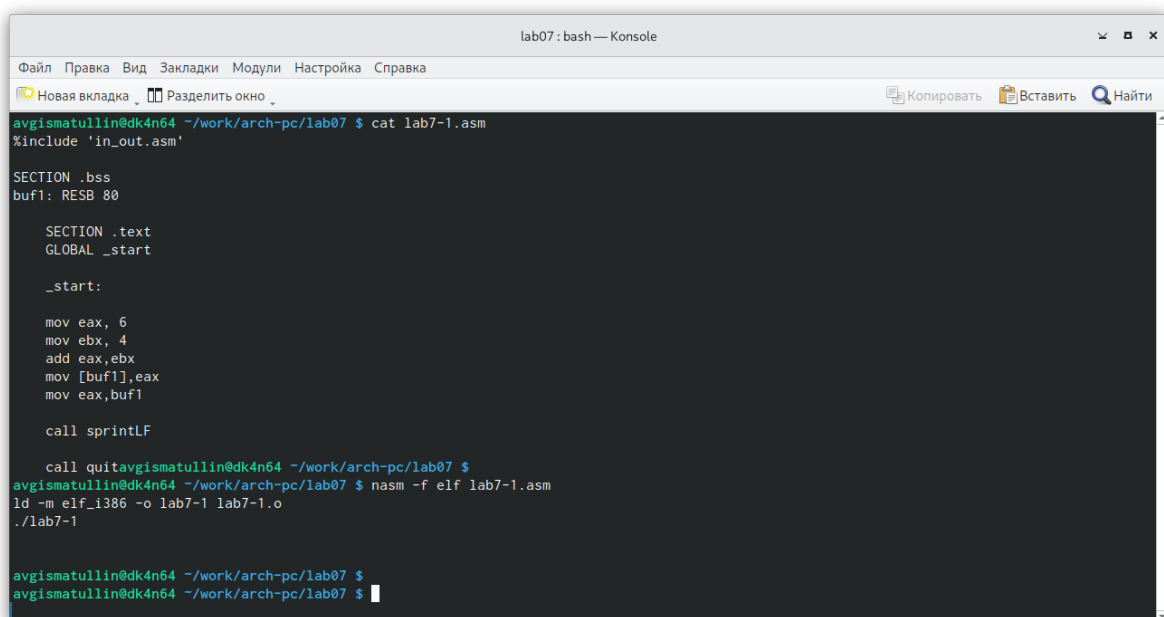
mov eax,'6'
mov ebx,'4'
add eax,ebx
mov [buf1],eax
mov eax,buf1

call sprintLF

call quitavgismatullin@dk4n64 ~/work/arch-pc/lab07 $
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
ld -m elf_i386 -o lab7-1 lab7-1.o
./lab7-1
j
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $
```

Рис. 4.2: Командная строка. Выполнение программы lab7-1.asm

Далее изменили текст программы и вместо символов записали в регистры числа. Запустили исполняемый файл. (рис. 4.3)



```
lab07: bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно
Копировать  Вставить  Найти

avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ cat lab7-1.asm
#include 'in_out.asm'

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 quitavgismatullin@dk4n64 ~/work/arch-pc/lab07 $
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-1.asm
ld -m elf_i386 -o lab7-1 lab7-1.o
./lab7-1

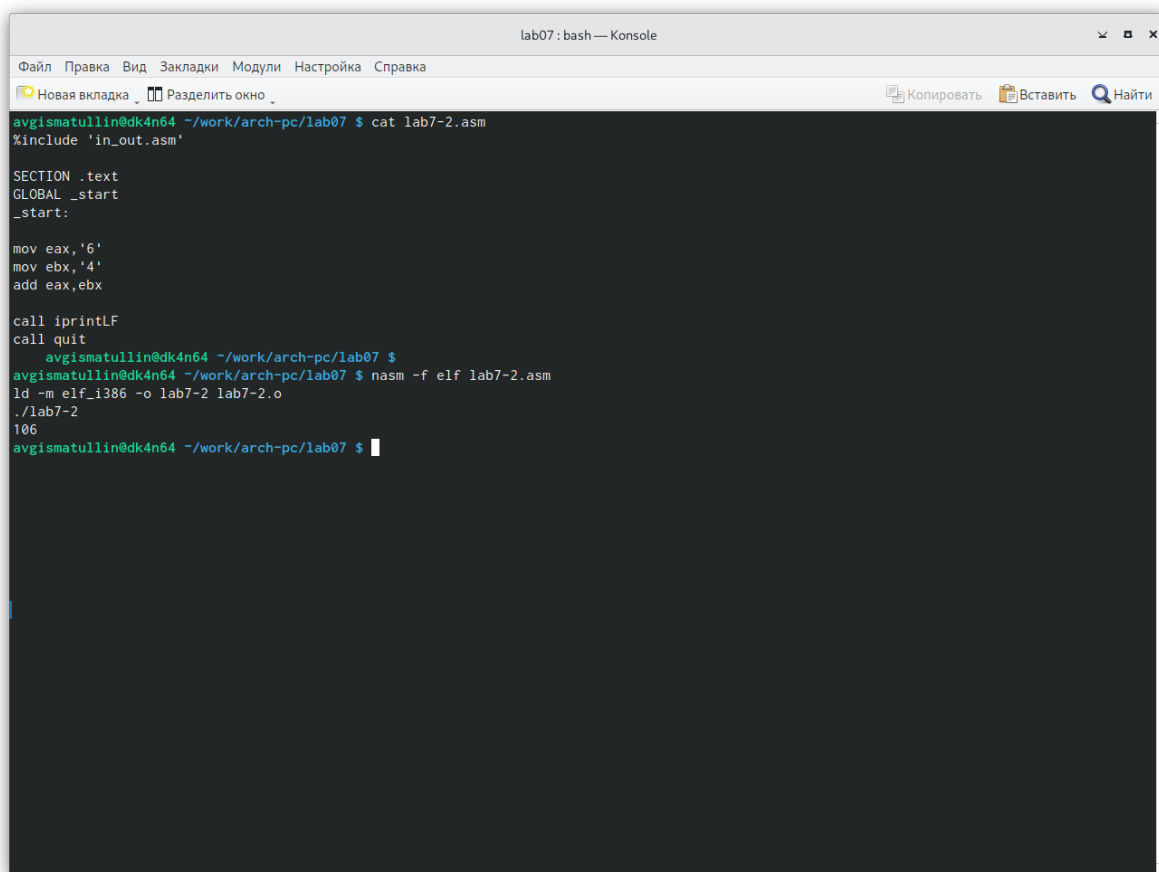
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $
```

Рис. 4.3: Командная строка. Выполнение измененной программы lab7-1.asm

Можно заметить, что символ LF,[символ перевода строки] (под 10 в таблице

ASCII) не отображается (замечен только перевод строки).

3. Создали файл lab7-2.asm и ввели туда текст следующего листинга. Создали и запустили исполняемый файл. (рис. 4.4)



```
lab07: bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
+ Новая вкладка  Разделить окно
Копировать  Вставить  Найти

avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ cat lab7-2.asm
#include 'in_out.asm'

SECTION .text
GLOBAL _start
_start:

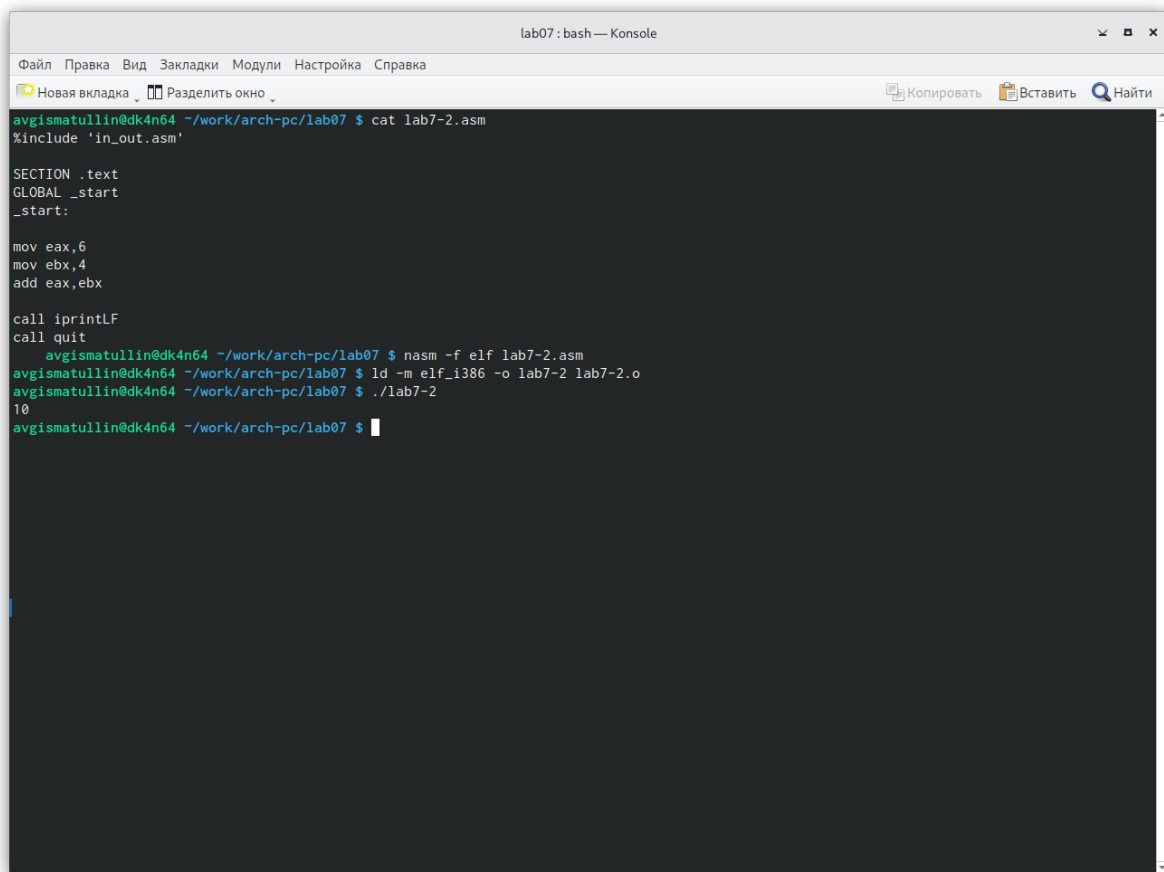
mov eax,'6'
mov ebx,'4'
add eax,ebx

call iprintLF
call quit

avgismatullin@dk4n64 ~/work/arch-pc/lab07 $
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
ld -m elf_i386 -o lab7-2 lab7-2.o
./lab7-2
106
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $
```

Рис. 4.4: Командная строка. Выполнение программы lab7-2.asm

Изменили текст программы аналогично lab7-1.asm (вместо символов записали в регистры числа) и посмотрели результат. (рис. 4.5)



```
lab07: bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ cat lab7-2.asm
#include 'in_out.asm'

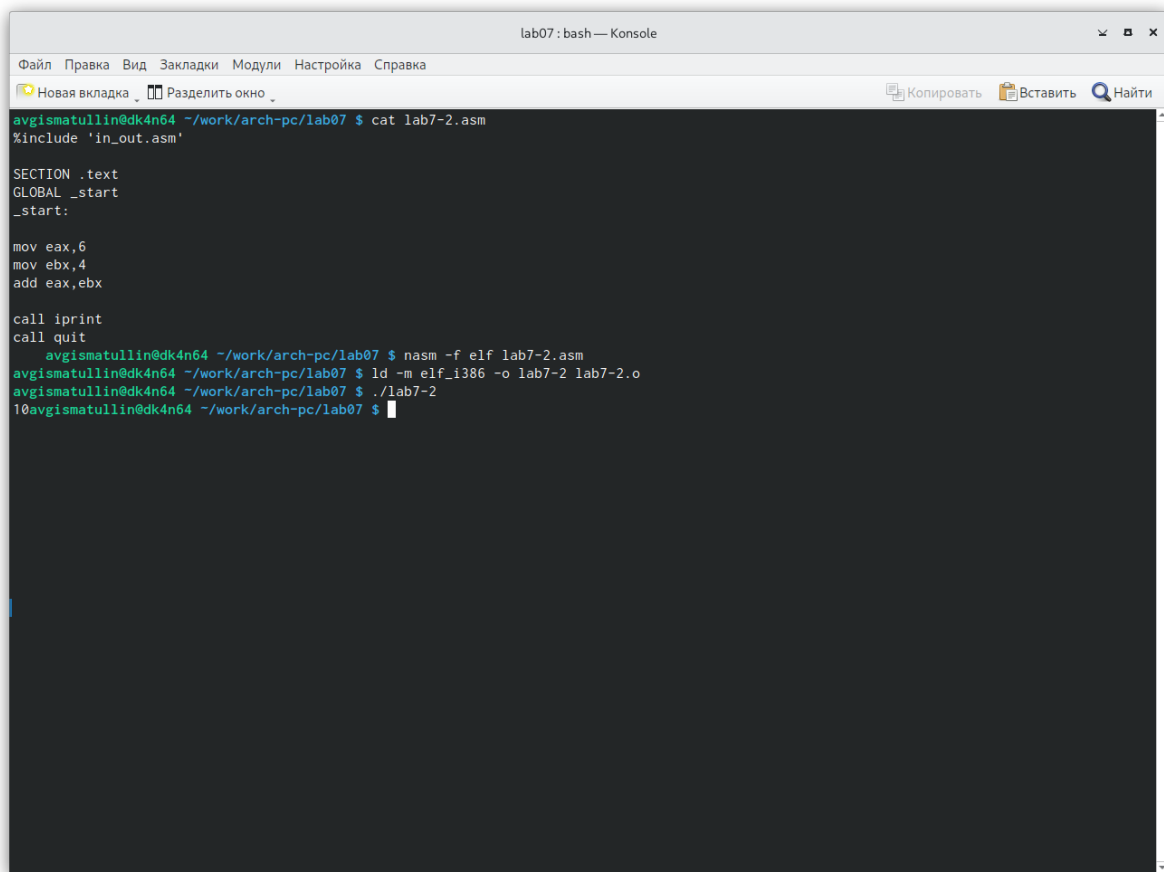
SECTION .text
GLOBAL _start
_start:

mov eax, 6
mov ebx, 4
add eax, ebx

call iprintLF
call quit
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ ./lab7-2
10
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $
```

Рис. 4.5: Командная строка. Выполнение измененной программы lab7-2.asm

Проверили разницу команды `call iprintLF` и `iprint`. Из теоретической части прошлой лабораторной работы мы можем сделать вывод, что разница будет заключаться в переводе на следующую строку, что наглядно видно на рисунке. (рис. 4.6)



```
lab07: bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ cat lab7-2.asm
%include 'in_out.asm'

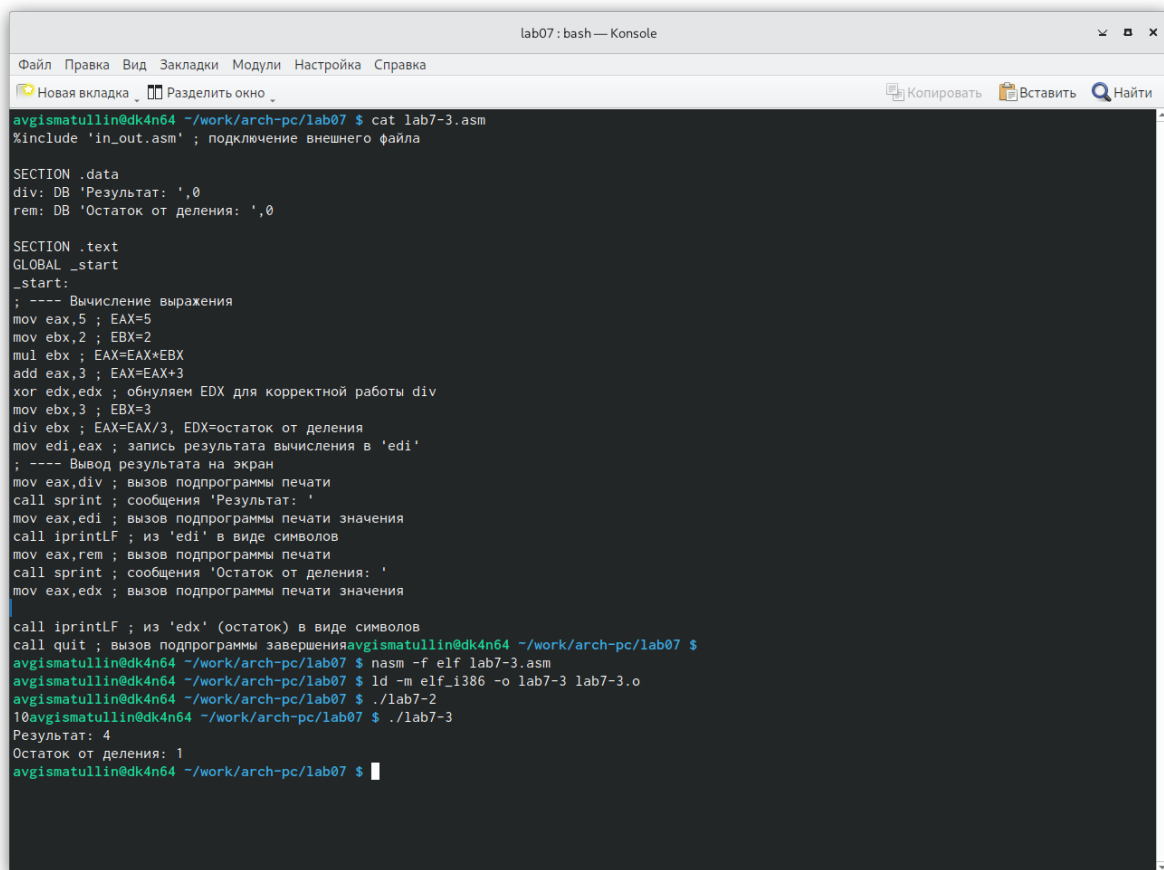
SECTION .text
GLOBAL _start
_start:

mov eax,6
mov ebx,4
add eax,ebx

call iprint
call quit
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ ./lab7-2
10avgismatullin@dk4n64 ~/work/arch-pc/lab07 $
```

Рис. 4.6: Командная строка. Выполнение измененной программы lab7-2.asm 2

- После этого создаем новый файл lab7-3.asm для выполнения различных операций в NASM. Внимательно изучили программу для вычисления выражения $f(x) = (5 * 2 + 3) / 3$. (рис. 4.7)



```
lab07: bash — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно Копировать Вставить Найти
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ cat lab7-3.asm
%include 'in_out.asm' ; подключение внешнего файла

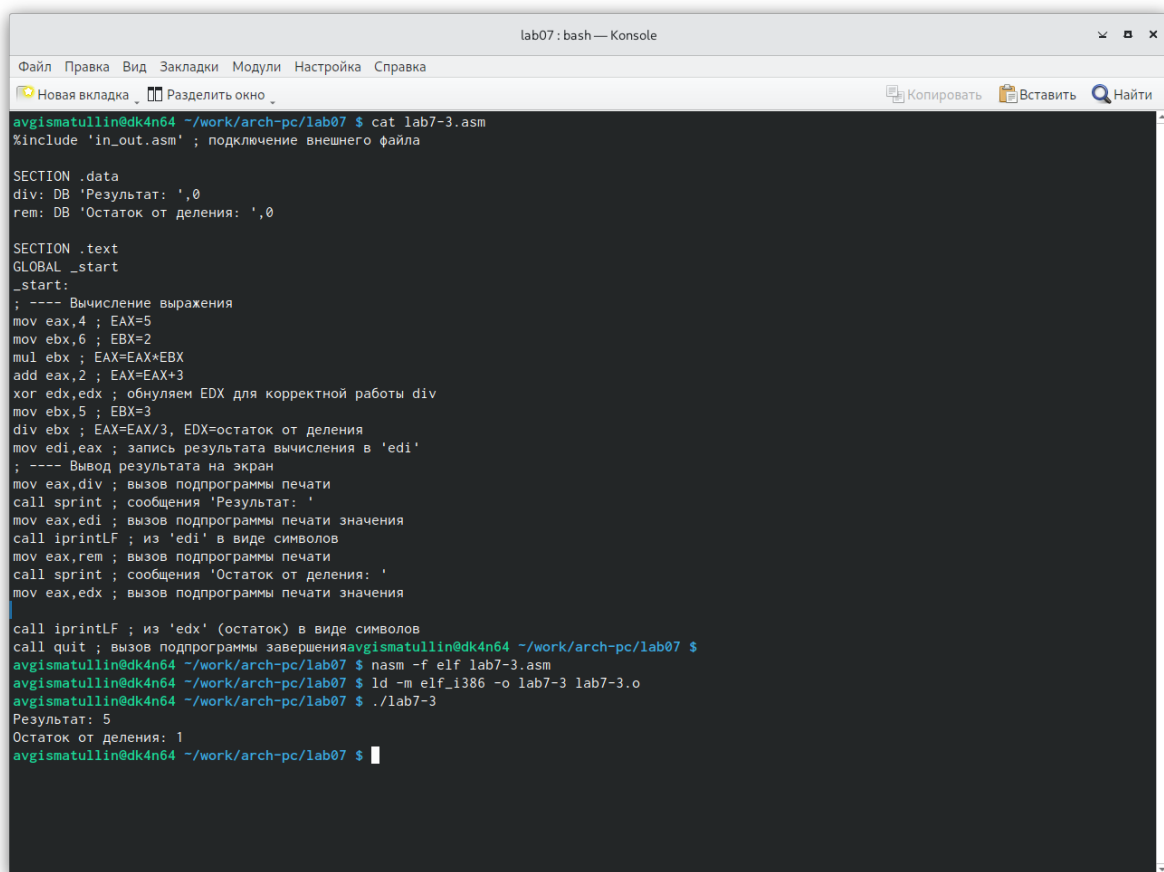
SECTION .data
div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0

SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения
mov eax,5 ; EAX=5
mov ebx,2 ; EBX=2
mul ebx ; EAX=EAX*EBX
add eax,3 ; EAX=EAX+3
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,3 ; EBX=3
div ebx ; EAX=EAX/3, EDX=остаток от деления
mov edi,eax ; запись результата вычисления в 'edi'
; ---- Вывод результата на экран
mov eax,div ; вызов подпрограммы печати
call sprint ; сообщения 'Результат: '
mov eax,edi ; вызов подпрограммы печати значения
call iprintLF ; из 'edi' в виде символов
mov eax,rem ; вызов подпрограммы печати
call sprint ; сообщения 'Остаток от деления: '
mov eax,edx ; вызов подпрограммы печати значения

call iprintLF ; из 'edx' (остаток) в виде символов
call quit ; вызов подпрограммы завершения
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-3.asm
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-3 lab7-3.o
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ ./lab7-2
10avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ ./lab7-3
Результат: 4
Остаток от деления: 1
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $
```

Рис. 4.7: Командная строка. Выполнение программы lab7-3.asm

Изменяем текст программы так, чтоб она вычисляла результат выражения $f(x)$
 $= (4 * 6 + 2) / 5$. (рис. 4.8)



```
lab07: bash — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно Копировать Вставить Найти
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ cat lab7-3.asm
%include 'in_out.asm' ; подключение внешнего файла

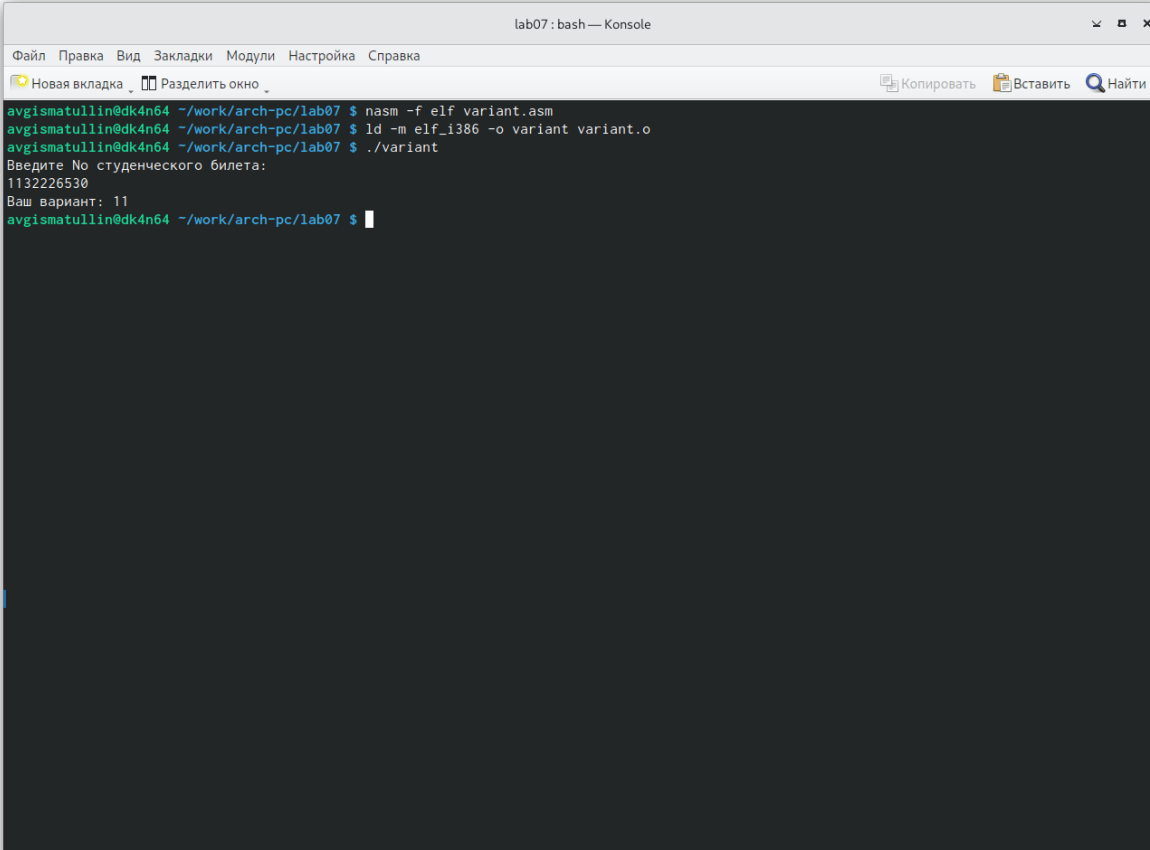
SECTION .data
div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0

SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения
mov eax,4 ; EAX=5
mov ebx,6 ; EBX=2
mul ebx ; EAX=EAX*EBX
add eax,2 ; EAX=EAX+3
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,5 ; EBX=3
div ebx ; EAX=EAX/3, EDX=остаток от деления
mov edi,eax ; запись результата вычисления в 'edi'
; ---- Вывод результата на экран
mov eax,div ; вызов подпрограммы печати
call sprint ; сообщения 'Результат: '
mov eax,edi ; вызов подпрограммы печати значения
call iprintLF ; из 'edi' в виде символов
mov eax,rem ; вызов подпрограммы печати
call sprint ; сообщения 'Остаток от деления: '
mov eax,edx ; вызов подпрограммы печати значения

call iprintLF ; из 'edx' (остаток) в виде символов
call quit ; вызов подпрограммы завершения
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-3.asm
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-3 lab7-3.o
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ ./lab7-3
Результат: 5
Остаток от деления: 2
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $
```

Рис. 4.8: Командная строка. Выполнение измененной программы lab7-3.asm

- Следующим шагом будет выисление варианта задания по номеру студенского билета. Для этого создаем файл variant.asm, изучаем алгоритм и выполняем соответствующую программу. (рис. 4.9)



```
lab07: bash — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно Копировать Вставить Найти
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ nasm -f elf variant.asm
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o variant variant.o
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ ./variant
Введите No студенческого билета:
1132226530
Ваш вариант: 11
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $
```

Рис. 4.9: Командная строка. Выполнение программы variant.asm

Таким образом, наш вариант для выполнения самостоятельной работы под номером 11.

6. Ответы на вопросы:

1. Какие строки листинга 7.4 отвечают за вывод на экран сообщения 'Ваш вариант:'?

Ответ: `mov eax, rem` `call sprint`.

2. Для чего используются следующие инструкции? `nasm` `mov esx, x` `mov edx, 80` `call sread`

Ответ: инструкция `nasm` необходима для трансляции программы файла с расширением `.asm` в объектный код (результат запишется в объектный файл

с расш. .o). `mov ecx, x` - Адрес x под вводимую строку `mov edx, 80` - Указание длины вводимой строки `call sread` - ввод сообщения с клавиатуры (до нее нужно указать, в какой регистр и какой длины сообщение инструкциями `mov`)

3. Для чего используется инструкция “`call atoi`”?

Ответ: `call atoi` - функция преобразует `ascii`-код символа в целое число и записывает результат в регистр `eax`.

4. Какие строки листинга 7.4 отвечают за вычисления варианта?

Ответ: `mov eax,x` `call atoi` `xor edx,edx` `mov ebx,20` `div ebx` `inc edx`

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

Ответ: В регистр `edx`

6. Для чего используется инструкция “`inc edx`”?

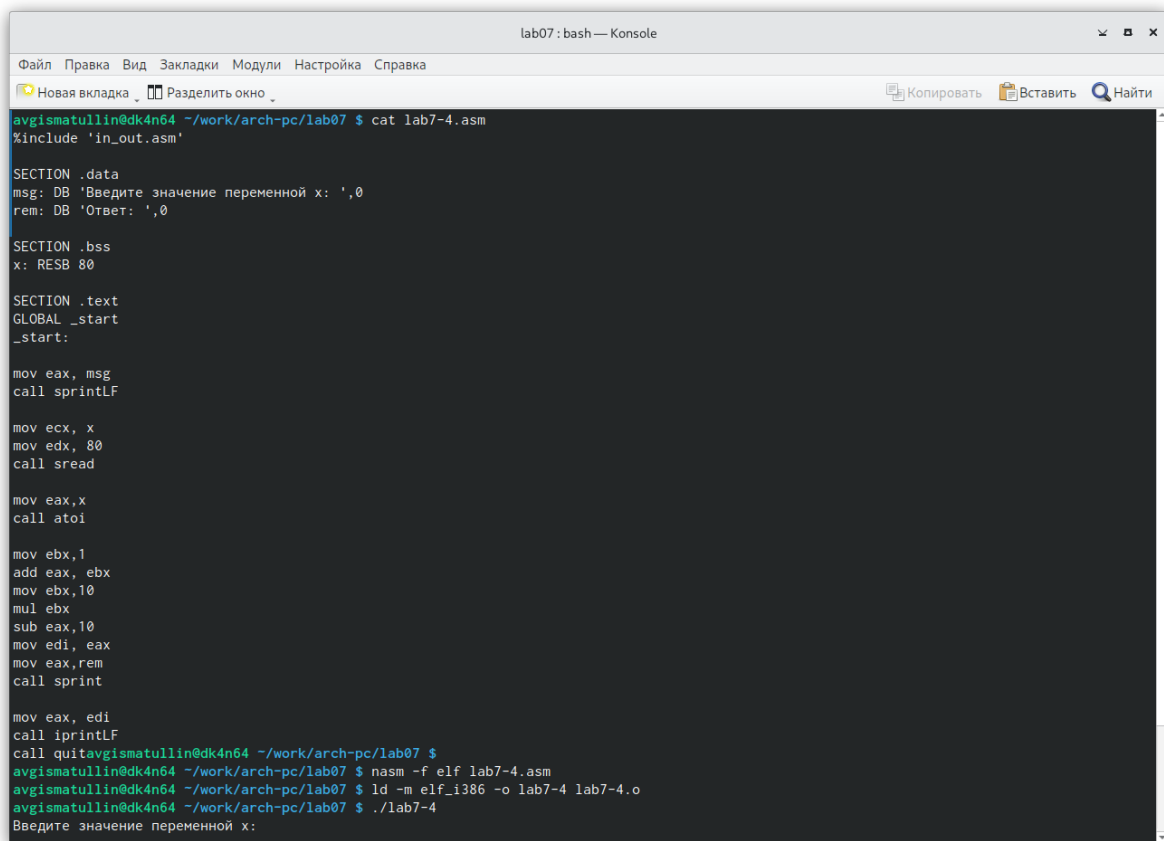
Ответ: Увеличение операнда регистра `edx` на единицу.

7. Какие строки листинга 7.4 отвечают за вывод на экран результата вычислений?

Ответ: `mov eax,edx` `call sprintf` `mov eax,edx` `call iprintLF`

5 Выполнение заданий для самостоятельной работы.

Суть задания для самостоятельной работы заключается в том, чтобы написать программу для решения некоторой функции. Так как мой вариант - 11, то мне предстоит работать с функцией $f(x) = 10 * (x + 1) - 10$. На основе лабораторной работы я реализовал данную программу (назвал ее lab7-4.asm) следующим образом: (рис. 5.1)



```
lab07: bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
Новая вкладка  Разделить окно  Копировать  Вставить  Найти
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ cat lab7-4.asm
#include 'in_out.asm'

SECTION .data
msg: DB 'Введите значение переменной x: ',0
rem: DB 'Ответ: ',0

SECTION .bss
x: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax, msg
call sprintf

mov ecx, x
mov edx, 80
call sread

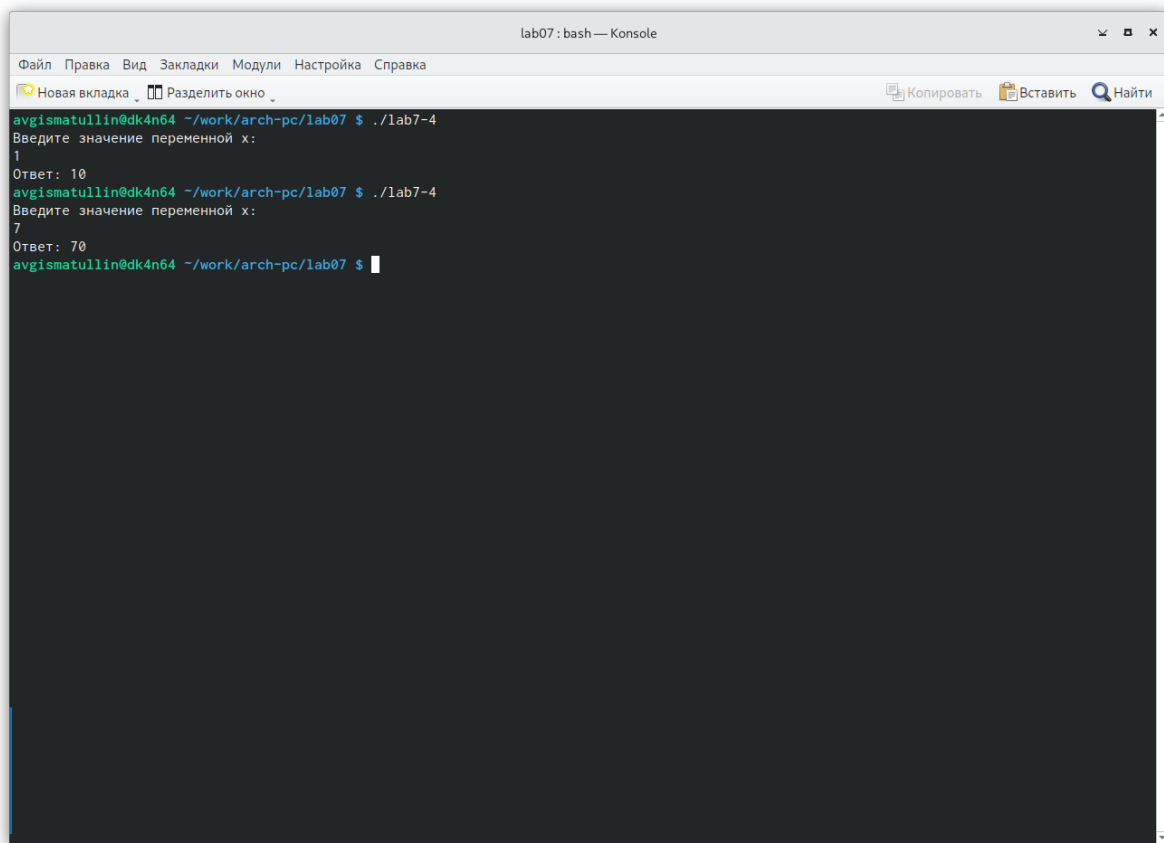
mov eax, x
call atoi

mov ebx, 1
add eax, ebx
mov ebx, 10
mul ebx
sub eax, 10
mov edi, eax
mov eax, rem
call sprintf

mov eax, edi
call iprintLF
call quitavgismatullin@dk4n64 ~/work/arch-pc/lab07 $
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ nasm -f elf lab7-4.asm
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-4 lab7-4.o
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ ./lab7-4
Введите значение переменной x:
```

Рис. 5.1: Командная строка. Код программы lab7-4.asm

Демонстрация работоспособности программы: (рис. 5.2)



```
lab07: bash — Konsole
Файл Правка Вид Закладки Модули Настройка Справка
Новая вкладка Разделить окно Копировать Вставить Найти
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ ./lab7-4
Введите значение переменной x:
1
Ответ: 10
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $ ./lab7-4
Введите значение переменной x:
7
Ответ: 70
avgismatullin@dk4n64 ~/work/arch-pc/lab07 $
```

Рис. 5.2: Командная строка. Выполнение программы lab7-4.asm

6 Выводы

В ходе выполнения работы я освоил арифметические инструкции языка ассемблера NASM.

Список литературы