

Отчёт по лабораторной работе № 7

Арифметические инструкции языка Ассемблер

Поляков Глеб Сергеевич

Содержание

1	Цель работы	5
2	Задание	6
2.0.1	Символьные и численные данные в NASM	6
2.0.2	Выполнение арифметических операций в NASM	6
3	Теоретическое введение	7
3.1	Адресация в NASM	7
3.1.1	Целочисленное вычитание sub.	8
3.1.2	Команда изменения знака операнда neg.	9
3.1.3	Команды умножения mul и imul.	10
3.1.4	Команды деления div и idiv.	11
3.1.5	Перевод символа числа в десятичную символьную запись .	12
4	Выполнение лабораторной работы	14
4.1	Выполнение задания для самостоятельной работы.	18
5	Выводы	20
	Список литературы	21

Список иллюстраций

4.1	Текст программы 7.1	14
4.2	Текст программы 7.2	15
4.3	Результат	16
4.4	Текст программы 7.4	16
4.5	Результат	17
4.6	Текст программы 7.5	17
4.7	Результат	18

Список таблиц

3.1	Регистры используемые командами умножения в Nasm	10
3.2	Регистры используемые командами деления в Nasm {#7:2}	11

1 Цель работы

Освоить арифметические инструкции языка Ассемблер

2 Задание

2.0.1 Символьные и численные данные в NASM

1. Создать каталог для программ лабораторной работы No 7, перейти в него и создать файл lab7-1.asm
2. Рассмотреть примеры программ вывода символьных и численных значений.
3. Далее изменить текст программы и вместо символов, записать в регистры числа. Исправьте текст программы (Листинг 1).
4. Как отмечалось выше, для работы с числами в файле in_out.asm реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Преобразуйте текст программы из Листинга 7.1 с использованием этих функций.
5. Аналогично предыдущему примеру измените символы на числа.

2.0.2 Выполнение арифметических операций в NASM

1. В качестве примера выполнения арифметических операций в NASM приведите программу вычисления арифметического выражения $\text{X}(\text{X}) = (5 \times 2 + 3)/3$.
2. В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета, работающую по следующему алгоритму.
3. Ответить на вопросы.

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 расс

`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`. ## Арифметические операции в NASM ### 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.`

3.1.1 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 имеют 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 `dec ax` уменьшает значение регистра `ax` на 1. операнд операнд

3.1.2 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
```

3.1.3 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 (от англ. multiply – умножение): mul Для знакового умножения используется команда imul: 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 7.1.

Таблица 3.1: Регистры используемые командами умножения в Nasm

Размер операнда	Неяв- ный мно- жи- тель		Результат умножения
	Регистр	Регистр	
1 байт	AL	AX	
2 байта	AX	DX:AX	
4 байта	EAX	EDX:EAX	

Таблица 7.1. Регистры используемые командами умножения в Nasm 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
```

3.1.4 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 7.2.

Таблица 3.2: Регистры используемые командами деления в Nasm {#7:2}

Размер операнда (делите- ля)	Дели- мое	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).

3.1.5 Перевод символа числа в десятичную символьную запись

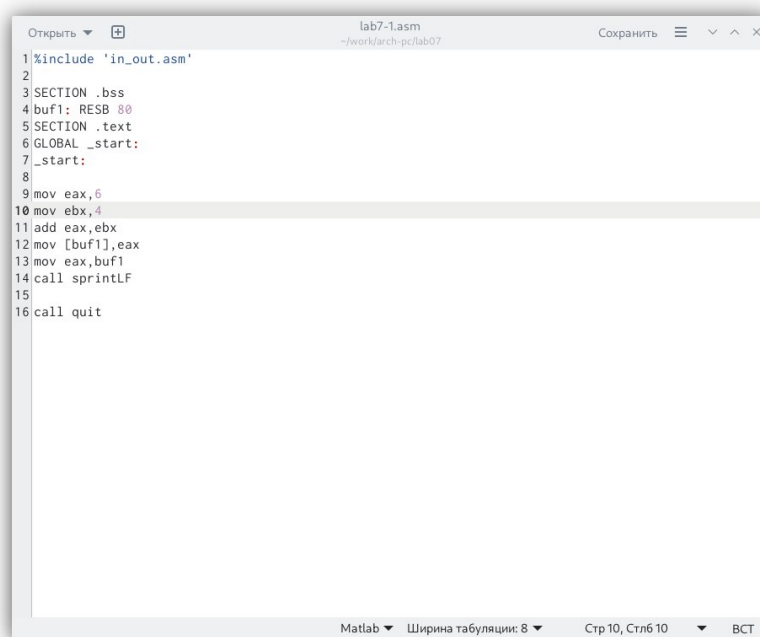
Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации производится согласно кодовой таблице символов ASCII. ASCII – сокращение от American Standard Code for Information Interchange (Американский стандартный код для обмена информацией). Согласно 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 под-программы для преобразования ASCII символов в числа и обратно. Это:

- `iprint` – вывод на экран чисел в формате ASCII, перед вызовом `iprint` в регистре `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 записывает результат в регистр `eax`, перед вызовом `atoi` в регистр `eax` необходимо записать число (`mov eax,`).

4 Выполнение лабораторной работы

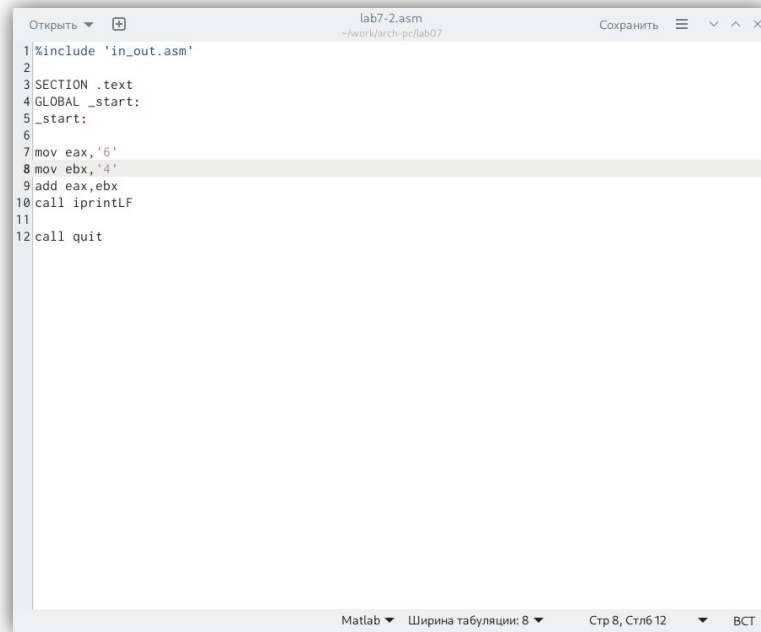
1. Создал каталог lab07 и создал файл lab7-1.asm (Рис. 1).
2. Написал программу по листингу 7.1 (Рис. 1).
3. Изменил текст программы, выходной символ не отображается.



```
1 %include 'in_out.asm'
2
3 SECTION .bss
4 buf1: RESB 80
5 SECTION .text
6 GLOBAL _start:
7 _start:
8
9 mov eax, 6
10 mov ebx, 4
11 add eax, ebx
12 mov [buf1], eax
13 mov eax, buf1
14 call printf
15
16 call quit
```

Рис. 4.1: Текст программы 7.1

4. Создал файл 7-2.asm и ввел программу (Рис. 2)



The screenshot shows a MATLAB editor window titled 'lab7-2.asm' with the file path '~\work\arch-pc\lab07'. The code is as follows:

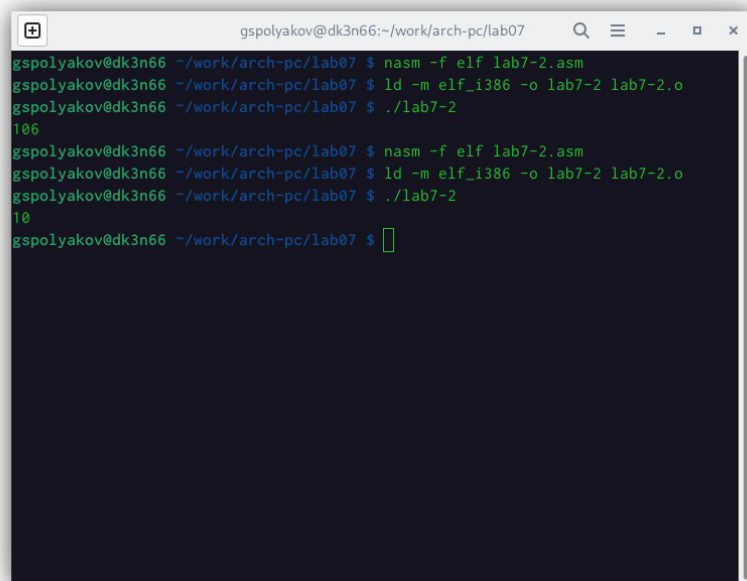
```
1 %include 'in_out.asm'
2
3 SECTION .text
4 GLOBAL _start:
5 _start:
6
7 mov eax, '6'
8 mov ebx, '4'
9 add eax, ebx
10 call iprintf
11
12 call quit
```

The status bar at the bottom indicates 'Matlab', 'Ширина табуляции: 8', 'Стр 8, Стлб 12', and 'ВСТ'.

Рис. 4.2: Текст программы 7.2

5.

6. Заменял текст программы. Получен результат 10.

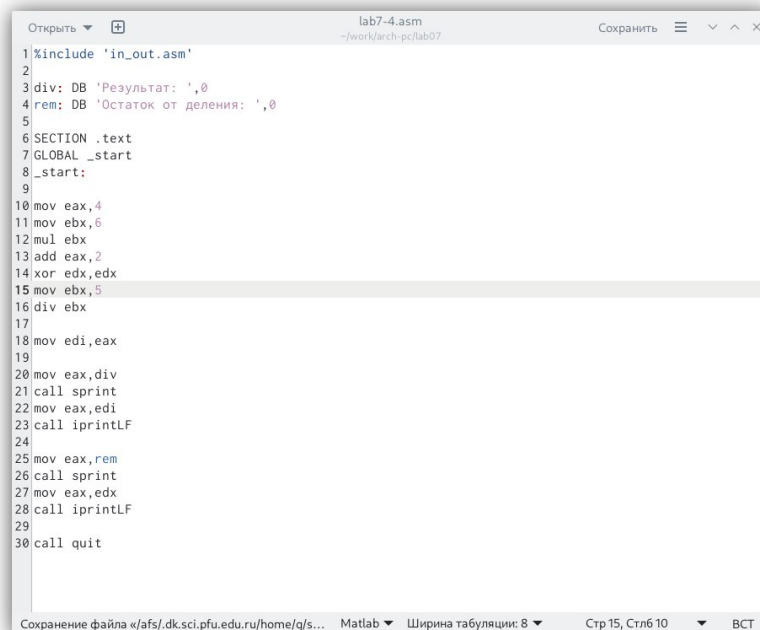


A terminal window with a dark background and light green text. The window title is 'gspolyakov@dk3n66:~/work/arch-pc/lab07'. The commands and their outputs are as follows:

```
gspolyakov@dk3n66 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
gspolyakov@dk3n66 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
gspolyakov@dk3n66 ~/work/arch-pc/lab07 $ ./lab7-2
106
gspolyakov@dk3n66 ~/work/arch-pc/lab07 $ nasm -f elf lab7-2.asm
gspolyakov@dk3n66 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-2 lab7-2.o
gspolyakov@dk3n66 ~/work/arch-pc/lab07 $ ./lab7-2
10
gspolyakov@dk3n66 ~/work/arch-pc/lab07 $
```

Рис. 4.3: Результат

6. Создал файл, ввел текст программы 7-4. Получил результат (Рис. 3) (Рис. 4)



A text editor window titled 'lab7-4.asm' with a light gray background. The code is as follows:

```
1 %include 'in_out.asm'
2
3 div: DB 'Результат: ',0
4 rem: DB 'Остаток от деления: ',0
5
6 SECTION .text
7 GLOBAL _start
8 _start:
9
10 mov eax,4
11 mov ebx,6
12 mul ebx
13 add eax,2
14 xor edx,edx
15 mov ebx,5
16 div ebx
17
18 mov edi,eax
19
20 mov eax,div
21 call sprint
22 mov eax,edi
23 call iprintLF
24
25 mov eax,rem
26 call sprint
27 mov eax,edx
28 call iprintLF
29
30 call quit
```

The status bar at the bottom shows: 'Сохранение файла «/afs/dk.sci.pfu.edu.ru/home/g/s... Matlab Ширина табуляции: 8 Стр 15, Стлб 10 ВСТ

Рис. 4.4: Текст программы 7.4

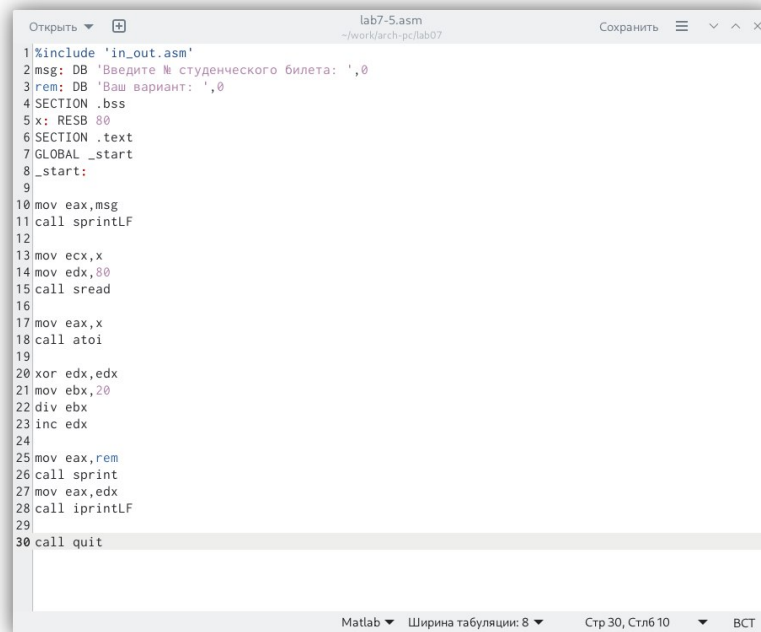

```

gspolyakov@dk3n66 ~/work/arch-pc/lab07 $ nasm -f elf lab7-3.asm
gspolyakov@dk3n66 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-3 lab7-3.o
gspolyakov@dk3n66 ~/work/arch-pc/lab07 $ ./lab7-3
Результат: 4
Остаток от деления: 1
gspolyakov@dk3n66 ~/work/arch-pc/lab07 $

```

Рис. 4.5: Результат

7. Создал файл и ввел текст программы 7-5. Получил результат 6.

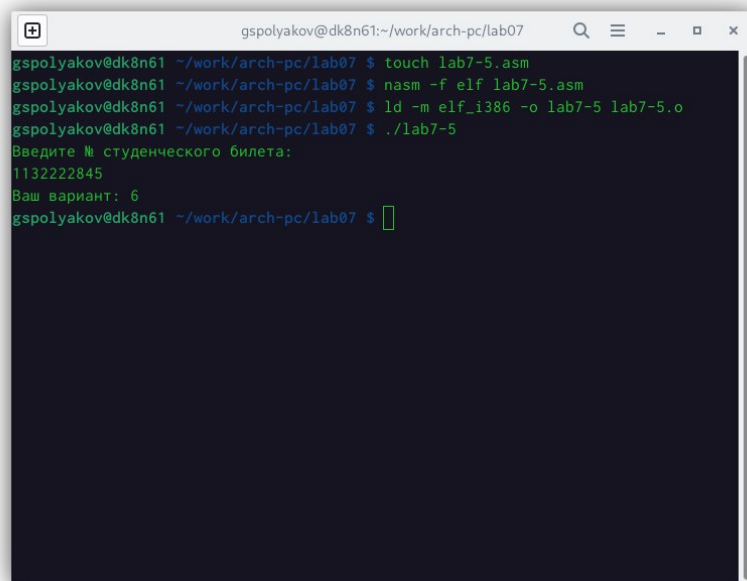


```

lab7-5.asm
~/work/arch-pc/lab07
Открыть + Сохранить
1 %include 'in_out.asm'
2 msg: DB 'Введите № студенческого билета: ',0
3 rem: DB 'Ваш вариант: ',0
4 SECTION .bss
5 x: RESB 80
6 SECTION .text
7 GLOBAL _start
8 _start:
9
10 mov eax,msg
11 call sprintf
12
13 mov ecx,x
14 mov edx,80
15 call sread
16
17 mov eax,x
18 call atoi
19
20 xor edx,edx
21 mov ebx,20
22 div ebx
23 inc edx
24
25 mov eax,rem
26 call sprint
27 mov eax,edx
28 call iprintf
29
30 call quit
Matlab Ширина табуляции: 8 Стр 30, Стлб 10 ВСТ

```

Рис. 4.6: Текст программы 7.5

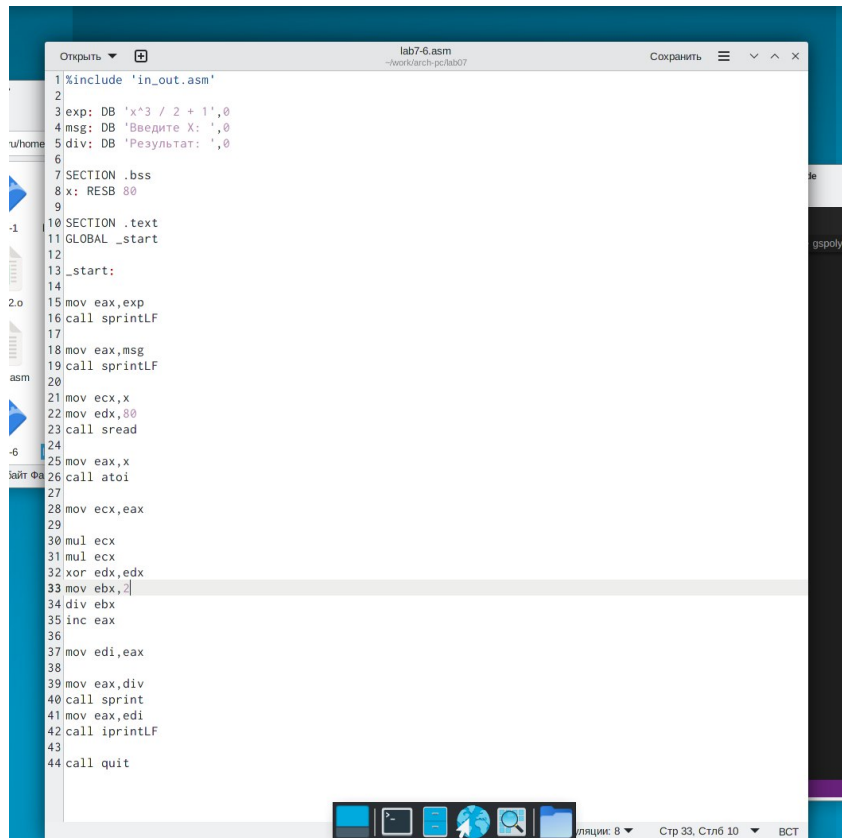
A terminal window with a dark background and light green text. The window title is 'gspolyakov@dk8n61:~/work/arch-pc/lab07'. The terminal shows the following commands and output:

```
gspolyakov@dk8n61 ~/work/arch-pc/lab07 $ touch lab7-5.asm
gspolyakov@dk8n61 ~/work/arch-pc/lab07 $ nasm -f elf lab7-5.asm
gspolyakov@dk8n61 ~/work/arch-pc/lab07 $ ld -m elf_i386 -o lab7-5 lab7-5.o
gspolyakov@dk8n61 ~/work/arch-pc/lab07 $ ./lab7-5
Введите № студенческого билета:
1132222845
Ваш вариант: 6
gspolyakov@dk8n61 ~/work/arch-pc/lab07 $
```

Рис. 4.7: Результат

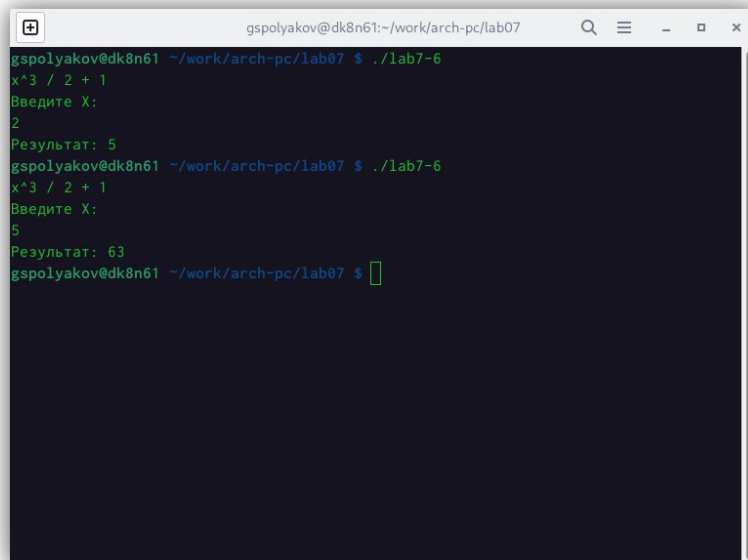
4.1 Выполнение задания для самостоятельной работы.

Ввел текст программы 7-6 варианта 6.



```
1 %include 'in_out.asm'
2
3 exp: DB 'x^3 / 2 + 1',0
4 msg: DB 'Введите X: ',0
5 div: DB 'Результат: ',0
6
7 SECTION .bss
8 x: RESB 80
9
10 SECTION .text
11 GLOBAL _start
12
13 _start:
14
15 mov eax,exp
16 call sprintf
17
18 mov eax,msg
19 call sprintf
20
21 mov ecx,x
22 mov edx,80
23 call sread
24
25 mov eax,x
26 call atoi
27
28 mov ecx,eax
29
30 mul ecx
31 mul ecx
32 xor edx,edx
33 mov ebx,2
34 div ebx
35 inc eax
36
37 mov edi,eax
38
39 mov eax,div
40 call sprintf
41 mov eax,edi
42 call iprintf
43
44 call quit
```

Получил результат.



```
gspolyakov@dk8n61:~/work/arch-pc/lab07 $ ./lab7-6
x^3 / 2 + 1
Введите X:
2
Результат: 5
gspolyakov@dk8n61:~/work/arch-pc/lab07 $ ./lab7-6
x^3 / 2 + 1
Введите X:
5
Результат: 63
gspolyakov@dk8n61:~/work/arch-pc/lab07 $
```

5 Выводы

Выполняя лабораторную работу №7, я научился использовать арифметические операции Ассемблера.

Список литературы