

Отчёт по лабораторной работе 5

Дисциплина: архитектура компьютера

Плугин Никита

Содержание

1	Цель работы	5
2	Задания	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Символьные и численные данные в NASM	9
4.2	Выполнение арифметических операций в NASM	15
4.2.1	Ответы на вопросы по программе variant.asm	20
4.3	Самостоятельное задание	21
5	Выводы	24

Список иллюстраций

4.1	Подготовил каталог	9
4.2	Программа в файле lab6-1.asm	10
4.3	Запуск программы lab6-1.asm	11
4.4	Программа в файле lab6-1.asm	12
4.5	Запуск программы lab6-1.asm	12
4.6	Программа в файле lab6-2.asm	13
4.7	Запуск программы lab6-2.asm	13
4.8	Программа в файле lab6-2.asm	14
4.9	Запуск программы lab6-2.asm	14
4.10	Запуск программы lab6-2.asm	15
4.11	Программа в файле lab6-3.asm	16
4.12	Запуск программы lab6-3.asm	16
4.13	Программа в файле lab6-3.asm	17
4.14	Запуск программы lab6-3.asm	18
4.15	Программа в файле variant.asm	19
4.16	Запуск программы variant.asm	19
4.17	Программа в файле work.asm	22
4.18	Запуск программы work.asm	23

Список таблиц

1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

2 Задания

1. Изучить инструкции ассемблера для арифметических действий
2. Написать программы по заданиям
3. Узнать свой вариант
4. Выполнить самостоятельное задание

3 Теоретическое введение

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.

Схема команды целочисленного сложения `add` (от англ. addition - добавление) выполняет сложение двух операндов и записывает результат по адресу первого операнда.

Команда целочисленного вычитания `sub` (от англ. subtraction – вычитание) работает аналогично команде `add`.

Существуют специальные команды: `inc` (от англ. increment) и `dec` (от англ. decrement), которые увеличивают и уменьшают на 1 свой операнд.

Умножение и деление, в отличие от сложения и вычитания, для знаковых и беззнаковых чисел производиться по-разному, поэтому существуют различ-

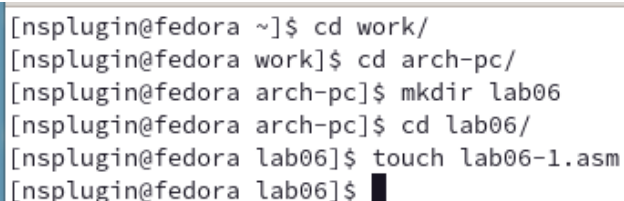
ные команды. Для беззнакового умножения используется команда `mul` (от англ. `multiply` – умножение), для знакового умножения используется команда `imul`.

Для деления, как и для умножения, существует 2 команды `div` (от англ. `divide` – деление) и `idiv`

4 Выполнение лабораторной работы

4.1 Символьные и численные данные в NASM

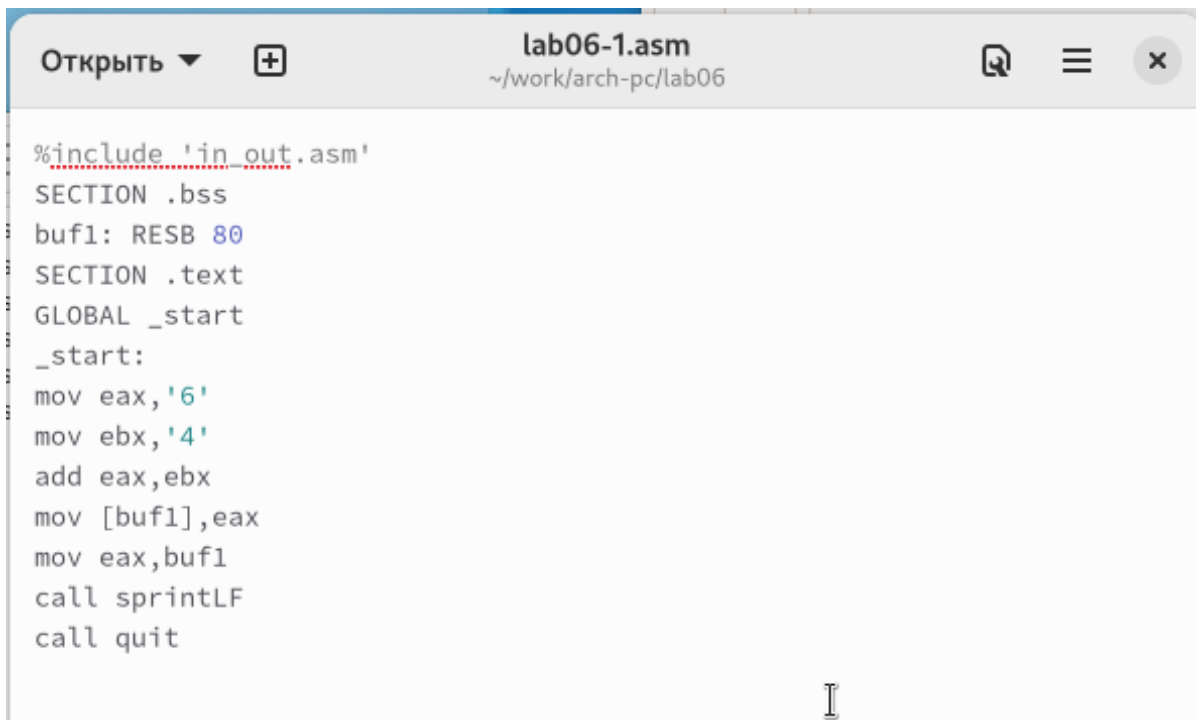
1. Создал каталог для программ лабораторной работы № 6, перешел в него и создал файл lab6-1.asm. (рис. [4.1])



```
[nsplugin@fedora ~]$ cd work/  
[nsplugin@fedora work]$ cd arch-pc/  
[nsplugin@fedora arch-pc]$ mkdir lab06  
[nsplugin@fedora arch-pc]$ cd lab06/  
[nsplugin@fedora lab06]$ touch lab06-1.asm  
[nsplugin@fedora lab06]$
```

Рис. 4.1: Подготовил каталог

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



```
Открыть ▾ + lab06-1.asm ~/work/arch-pc/lab06
%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 quit
```

Рис. 4.2: Программа в файле lab6-1.asm

В данной программе (рис. [4.2]) мы записываем символ '6' в регистр `eax` (`mov eax, '6'`), а символ '4' в регистр `ebx` (`mov ebx, '4'`). Затем мы добавляем значение регистра `ebx` к значению в регистре `eax` (`add eax, ebx`, результат сложения записывается в регистр `eax`). После этого мы выводим результат. Однако, для использования функции `sprintLF`, необходимо, чтобы в регистре `eax` был записан адрес, поэтому мы используем дополнительную переменную. Мы записываем значение регистра `eax` в переменную `buf1` (`mov [buf1], eax`), а затем записываем адрес переменной `buf1` в регистр `eax` (`mov eax, buf1`) и вызываем функцию `sprintLF`.

```

[nsplugin@fedora lab06]$ nasm -f elf lab06-1.asm
lab06-1.asm:1: error: unable to open include file `in_out.asm': No such file or
directory
[nsplugin@fedora lab06]$ nasm -f elf lab06-1.asm
[nsplugin@fedora lab06]$ ld -m elf_i386 lab06-1.o -o lab06-1
[nsplugin@fedora lab06]$ ./lab06-1
j
[nsplugin@fedora lab06]$ █

```

Рис. 4.3: Запуск программы lab6-1.asm

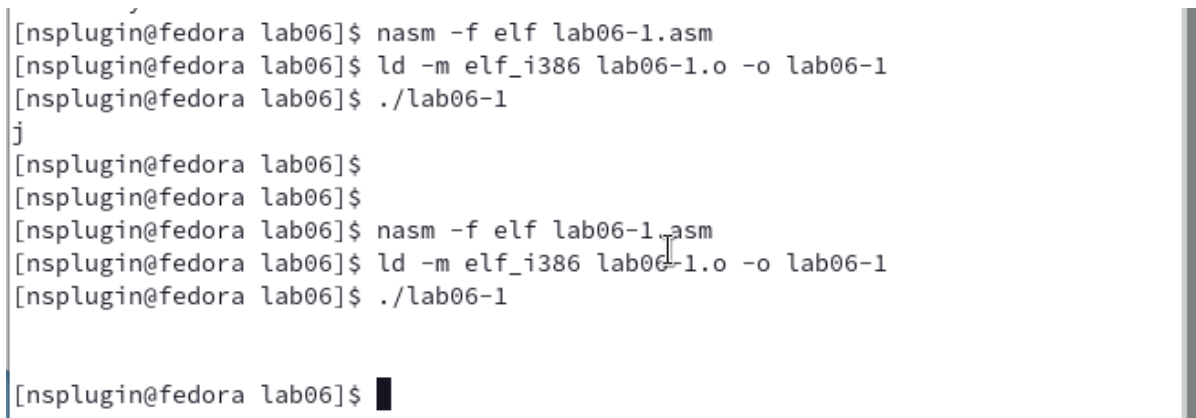
В данном случае, когда мы ожидаем увидеть число 10 при выводе значения регистра `eax`, фактическим результатом будет символ `'j'`. Это происходит из-за того, что код символа `'б'` равен 00110110 в двоичном представлении (или 54 в десятичном представлении), а код символа `'4'` равен 00110100 (или 52 в десятичном представлении). Когда мы выполняем команду `add eax, ebx`, результатом будет сумма кодов - 01101010 (или 106 в десятичном представлении), который соответствует символу `'j'`. (рис. [4.3])

3. Далее изменяю текст программы и вместо символов, запишем в регистры числа. (рис. [4.4])



```
Открыть + lab06-1.asm ~/work/arch-pc/lab06
%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 quit
```

Рис. 4.4: Программа в файле lab6-1.asm



```
[nsplugin@fedora lab06]$ nasm -f elf lab06-1.asm
[nsplugin@fedora lab06]$ ld -m elf_i386 lab06-1.o -o lab06-1
[nsplugin@fedora lab06]$ ./lab06-1
j
[nsplugin@fedora lab06]$
[nsplugin@fedora lab06]$
[nsplugin@fedora lab06]$ nasm -f elf lab06-1.asm
[nsplugin@fedora lab06]$ ld -m elf_i386 lab06-1.o -o lab06-1
[nsplugin@fedora lab06]$ ./lab06-1

[nsplugin@fedora lab06]$
```

Рис. 4.5: Запуск программы lab6-1.asm

Как и в предыдущем случае, при выполнении программы мы не получим число 10. Вместо этого выводится символ с кодом 10, который представляет собой символ конца строки (возврат каретки). (рис. [4.5]) Этот символ не отображается в консоли, но он добавляет пустую строку.

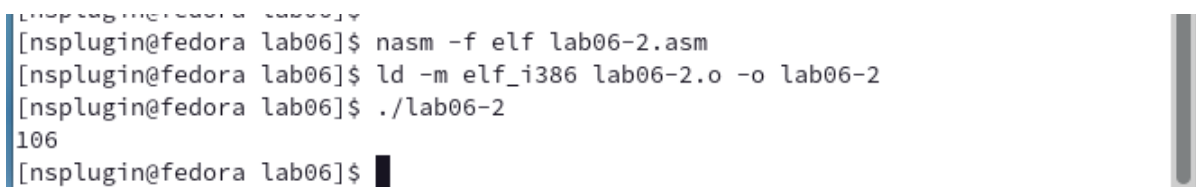
4. Как отмечалось выше, для работы с числами в файле `in_out.asm` реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Преобразовал текст программы с использованием этих функций. (рис. [4.6])



```
lab06-2.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. 4.6: Программа в файле `lab6-2.asm`



```
[nsplugin@fedora lab06]$ nasm -f elf lab06-2.asm
[nsplugin@fedora lab06]$ ld -m elf_i386 lab06-2.o -o lab06-2
[nsplugin@fedora lab06]$ ./lab06-2
106
[nsplugin@fedora lab06]$
```

Рис. 4.7: Запуск программы `lab6-2.asm`

В результате выполнения программы мы получим число 106. (рис. [4.7]) В данном случае, как и в первом случае, команда `add` складывает коды символов '6' и '4' ($54+52=106$). Однако, в отличие от предыдущей программы, функция `iprintLF` позволяет вывести число, а не символ, кодом которого является это число.

5. Аналогично предыдущему примеру изменим символы на числа. (рис. [4.8])

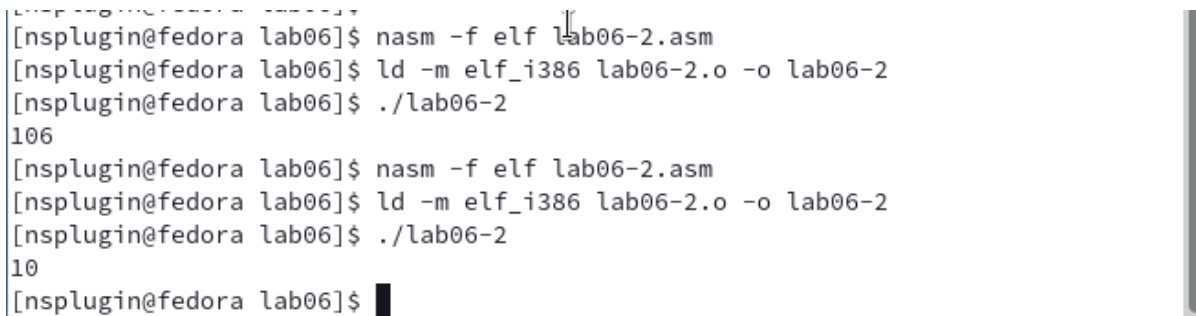


```
lab06-2.asm
~/work/arch-pc/lab06

%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprintLF
call quit
```

Рис. 4.8: Программа в файле lab6-2.asm

Функция iprintLF позволяет вывести число и операндами были числа (а не коды символов). Поэтому получаем число 10.(рис. [4.9])



```
[nsplugin@fedora lab06]$ nasm -f elf lab06-2.asm
[nsplugin@fedora lab06]$ ld -m elf_i386 lab06-2.o -o lab06-2
[nsplugin@fedora lab06]$ ./lab06-2
106
[nsplugin@fedora lab06]$ nasm -f elf lab06-2.asm
[nsplugin@fedora lab06]$ ld -m elf_i386 lab06-2.o -o lab06-2
[nsplugin@fedora lab06]$ ./lab06-2
10
[nsplugin@fedora lab06]$
```

Рис. 4.9: Запуск программы lab6-2.asm

Заменяю функцию iprintLF на iprint. Создаю исполняемый файл и запускаю его. Вывод отличается тем, что нет переноса строки.(рис. [4.10])

```

[nsplugin@fedora lab06]$
[nsplugin@fedora lab06]$ nasm -f elf lab06-2.asm
[nsplugin@fedora lab06]$ ld -m elf_i386 lab06-2.o -o lab06-2
[nsplugin@fedora lab06]$ ./lab06-2
106
[nsplugin@fedora lab06]$ nasm -f elf lab06-2.asm
[nsplugin@fedora lab06]$ ld -m elf_i386 lab06-2.o -o lab06-2
[nsplugin@fedora lab06]$ ./lab06-2
10
[nsplugin@fedora lab06]$ nasm -f elf lab06-2.asm
[nsplugin@fedora lab06]$ ld -m elf_i386 lab06-2.o -o lab06-2
[nsplugin@fedora lab06]$ ./lab06-2
10[nsplugin@fedora lab06]$

```

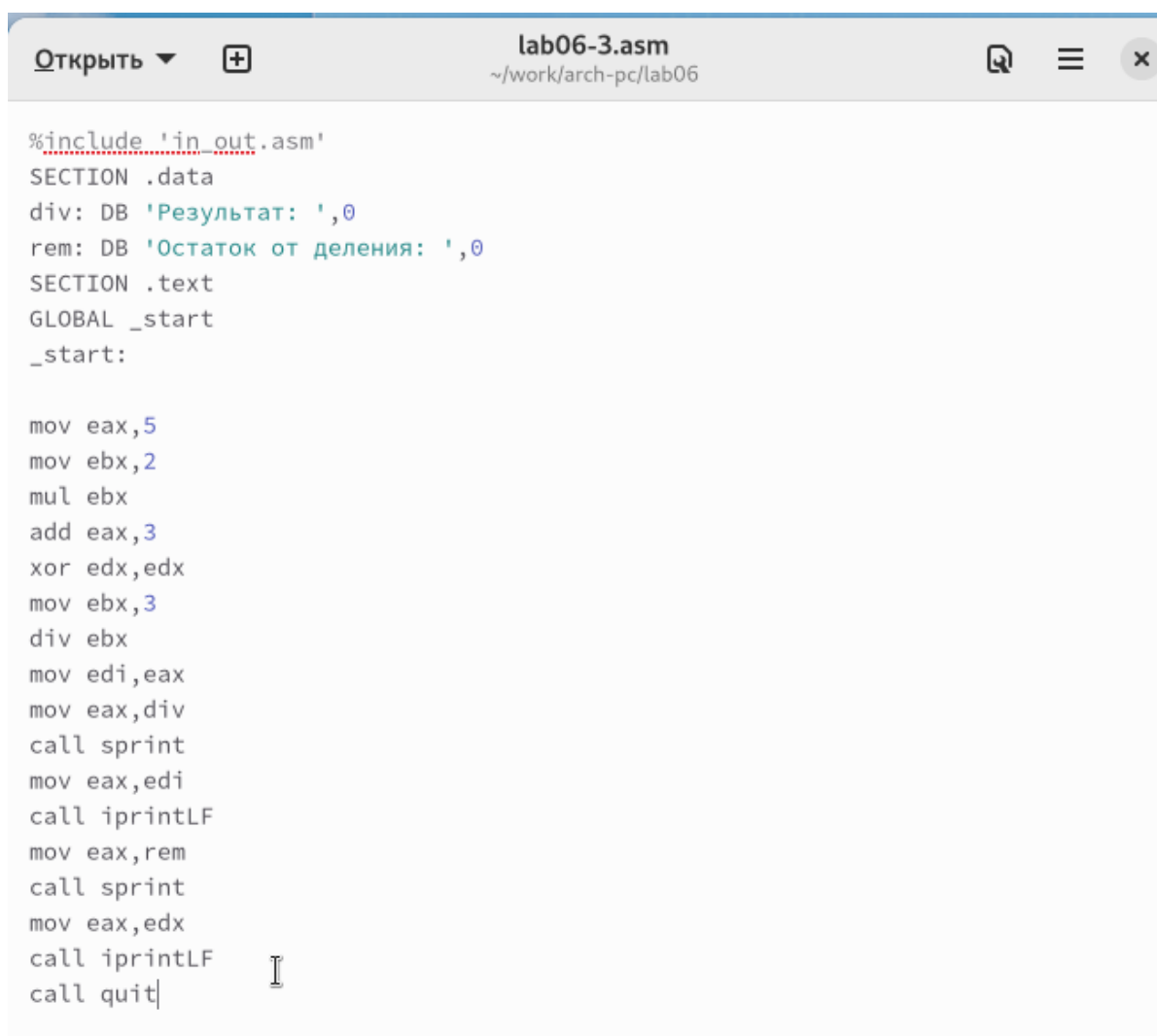
Рис. 4.10: Запуск программы lab6-2.asm

4.2 Выполнение арифметических операций в NASM

6. В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения (рис. [4.11]) (рис. [4.12])

$$f(x) = (5 * 2 + 3) / 3$$

.

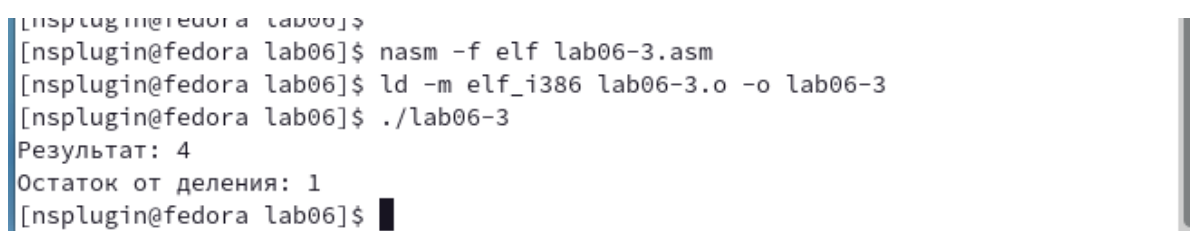


```
lab06-3.asm
~/work/arch-pc/lab06

%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0
SECTION .text
GLOBAL _start
_start:

mov eax,5
mov ebx,2
mul ebx
add eax,3
xor edx,edx
mov ebx,3
div ebx
mov edi,eax
mov eax,div
call sprint
mov eax,edi
call iprintLF
mov eax,rem
call sprint
mov eax,edx
call iprintLF
call quit
```

Рис. 4.11: Программа в файле lab6-3.asm



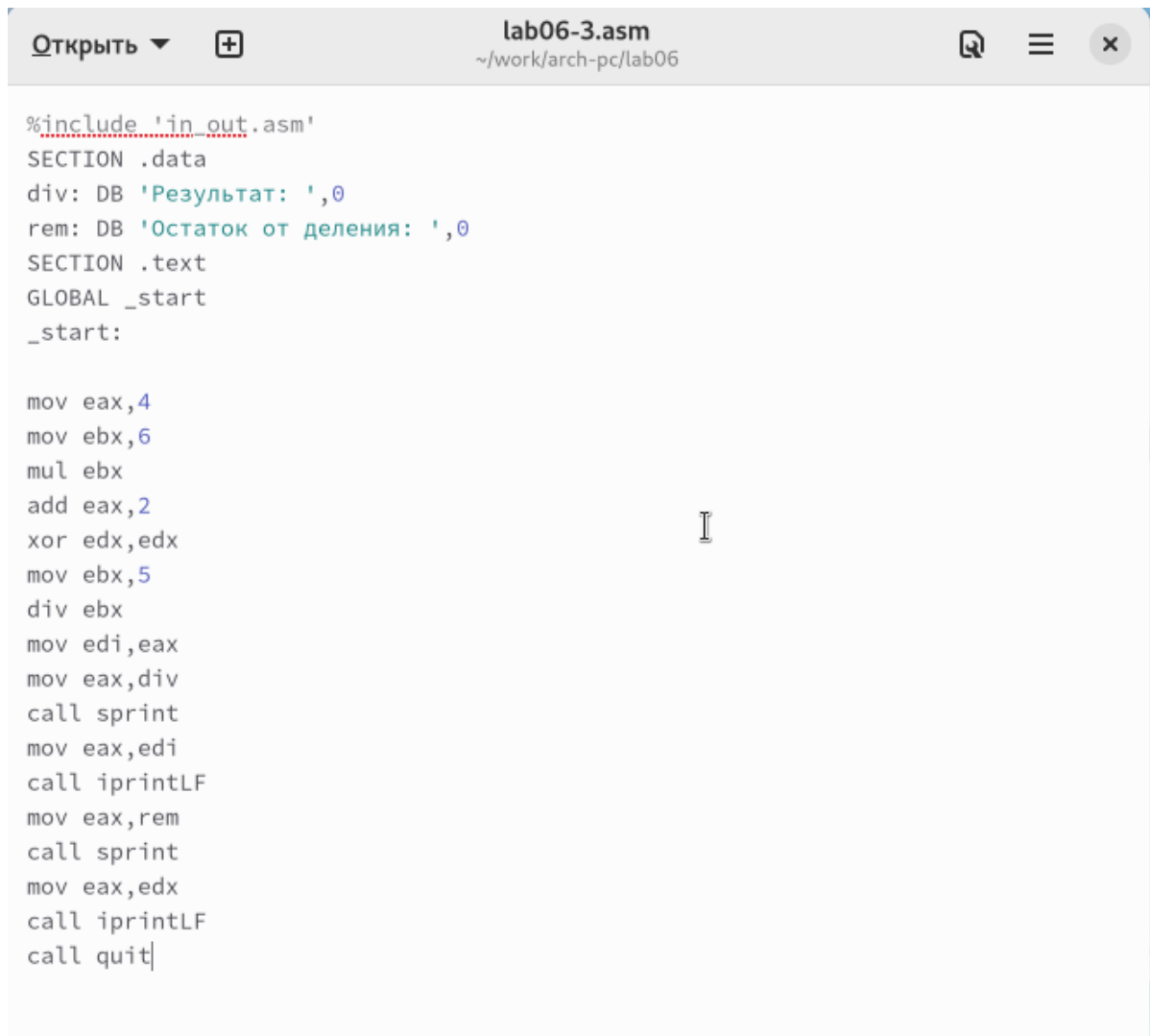
```
[nsplugin@fedora lab06]$ nasm -f elf lab06-3.asm
[nsplugin@fedora lab06]$ ld -m elf_i386 lab06-3.o -o lab06-3
[nsplugin@fedora lab06]$ ./lab06-3
Результат: 4
Остаток от деления: 1
[nsplugin@fedora lab06]$
```

Рис. 4.12: Запуск программы lab6-3.asm

Изменил текст программы для вычисления выражения

$$f(x) = (4 * 6 + 2) / 5$$

. Создал исполняемый файл и проверил его работу. (рис. [4.13]) (рис. [4.14])



```
lab06-3.asm
~/work/arch-pc/lab06

%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0
SECTION .text
GLOBAL _start
_start:

mov eax,4
mov ebx,6
mul ebx
add eax,2
xor edx,edx
mov ebx,5
div ebx
mov edi,eax
mov eax,div
call sprint
mov eax,edi
call iprintLF
mov eax,rem
call sprint
mov eax,edx
call iprintLF
call quit
```

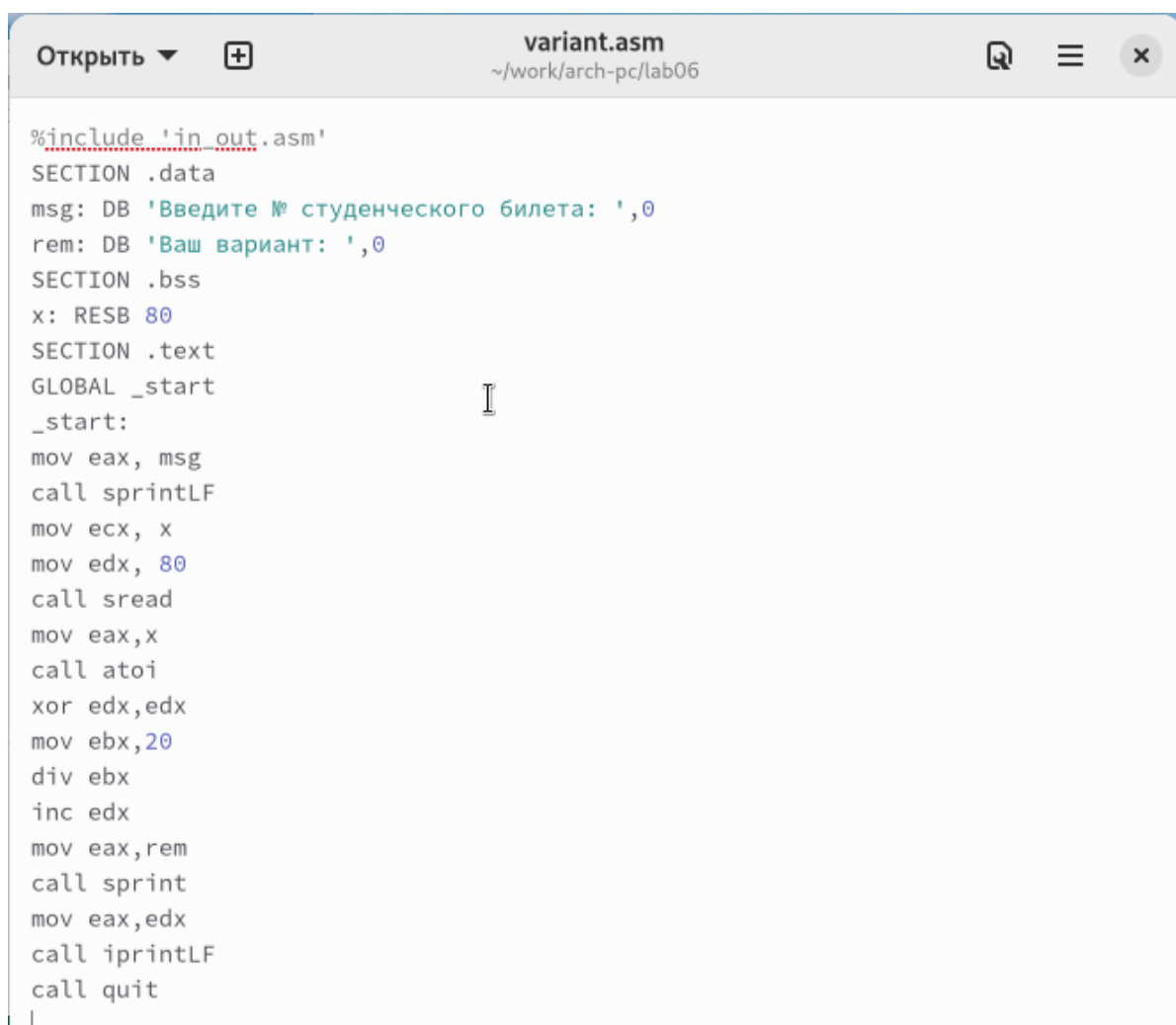
Рис. 4.13: Программа в файле lab6-3.asm

```
[nsplugin@fedora lab06]$ nasm -f elf lab06-3.asm
[nsplugin@fedora lab06]$ ld -m elf_i386 lab06-3.o -o lab06-3
[nsplugin@fedora lab06]$ ./lab06-3
Результат: 4
Остаток от деления: 1
[nsplugin@fedora lab06]$ nasm -f elf lab06-3.asm
[nsplugin@fedora lab06]$ ld -m elf_i386 lab06-3.o -o lab06-3
[nsplugin@fedora lab06]$ ./lab06-3
Результат: 5
Остаток от деления: 1
[nsplugin@fedora lab06]$
```

Рис. 4.14: Запуск программы lab6-3.asm

7. В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета. (рис. [4.15]) (рис. [4.16])

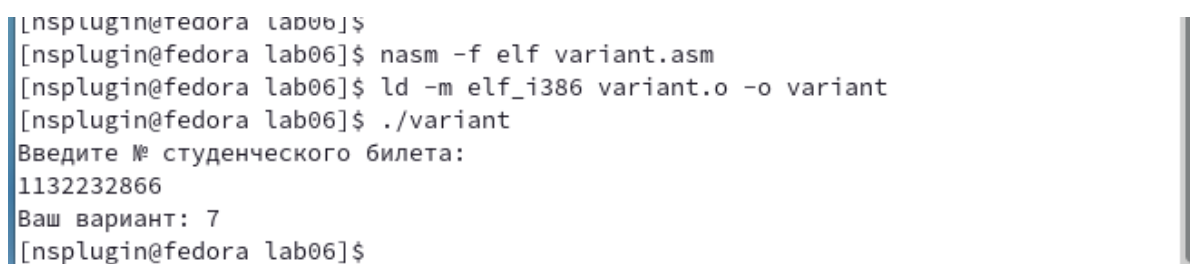
В данном случае число, над которым необходимо проводить арифметические операции, вводится с клавиатуры. Как отмечалось выше ввод с клавиатуры осуществляется в символьном виде и для корректной работы арифметических операций в NASM символы необходимо преобразовать в числа. Для этого может быть использована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



```
Открыть + variant.asm
~/work/arch-pc/lab06

%include 'in_out.asm'
SECTION .data
msg: DB 'Введите № студенческого билета: ',0
rem: DB 'Ваш вариант: ',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
xor edx, edx
mov ebx, 20
div ebx
inc edx
mov eax, rem
call sprintf
mov eax, edx
call iprintLF
call quit
|
```

Рис. 4.15: Программа в файле variant.asm



```
[nsplugin@fedora lab06]$
[nsplugin@fedora lab06]$ nasm -f elf variant.asm
[nsplugin@fedora lab06]$ ld -m elf_i386 variant.o -o variant
[nsplugin@fedora lab06]$ ./variant
Введите № студенческого билета:
1132232866
Ваш вариант: 7
[nsplugin@fedora lab06]$
```

Рис. 4.16: Запуск программы variant.asm

4.2.1 Ответы на вопросы по программе `variant.asm`

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

Строка `“mov eax, rem”` перекладывает в регистр значение переменной с фразой “Ваш вариант:”

Строка `“call sprint”` вызывает подпрограмму вывода строки

2. Для чего используются следующие инструкции?

Инструкция `“nasm”` используется для компиляции кода на языке ассемблера NASM

Инструкция `“mov ecx, x”` используется для перемещения значения переменной `x` в регистр `ecx`

Инструкция `“mov edx, 80”` используется для перемещения значения 80 в регистр `edx`

Инструкция `“call sread”` вызывает подпрограмму для считывания значения студенческого билета из консоли

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

Инструкция `“call atoi”` используется для преобразования введенных символов в числовой формат

4. Какие строки листинга отвечают за вычисления варианта?

Строка `“xor edx, edx”` обнуляет регистр `edx`

Строка `“mov ebx, 20”` записывает значение 20 в регистр `ebx`

Строка `“div ebx”` выполняет деление номера студенческого билета на 20

Строка `“inc edx”` увеличивает значение регистра `edx` на 1

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

Остаток от деления записывается в регистр `edx`

6. Для чего используется инструкция `inc edx`?

Инструкция `inc edx` используется для увеличения значения в регистре `edx` на 1, в соответствии с формулой вычисления варианта

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

Строка `mov eax, edx` перекладывает результат вычислений в регистр `eax`

Строка `call iprintLF` вызывает подпрограмму для вывода значения на экран

4.3 Самостоятельное 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 x_1 и x_2 из 6.3.

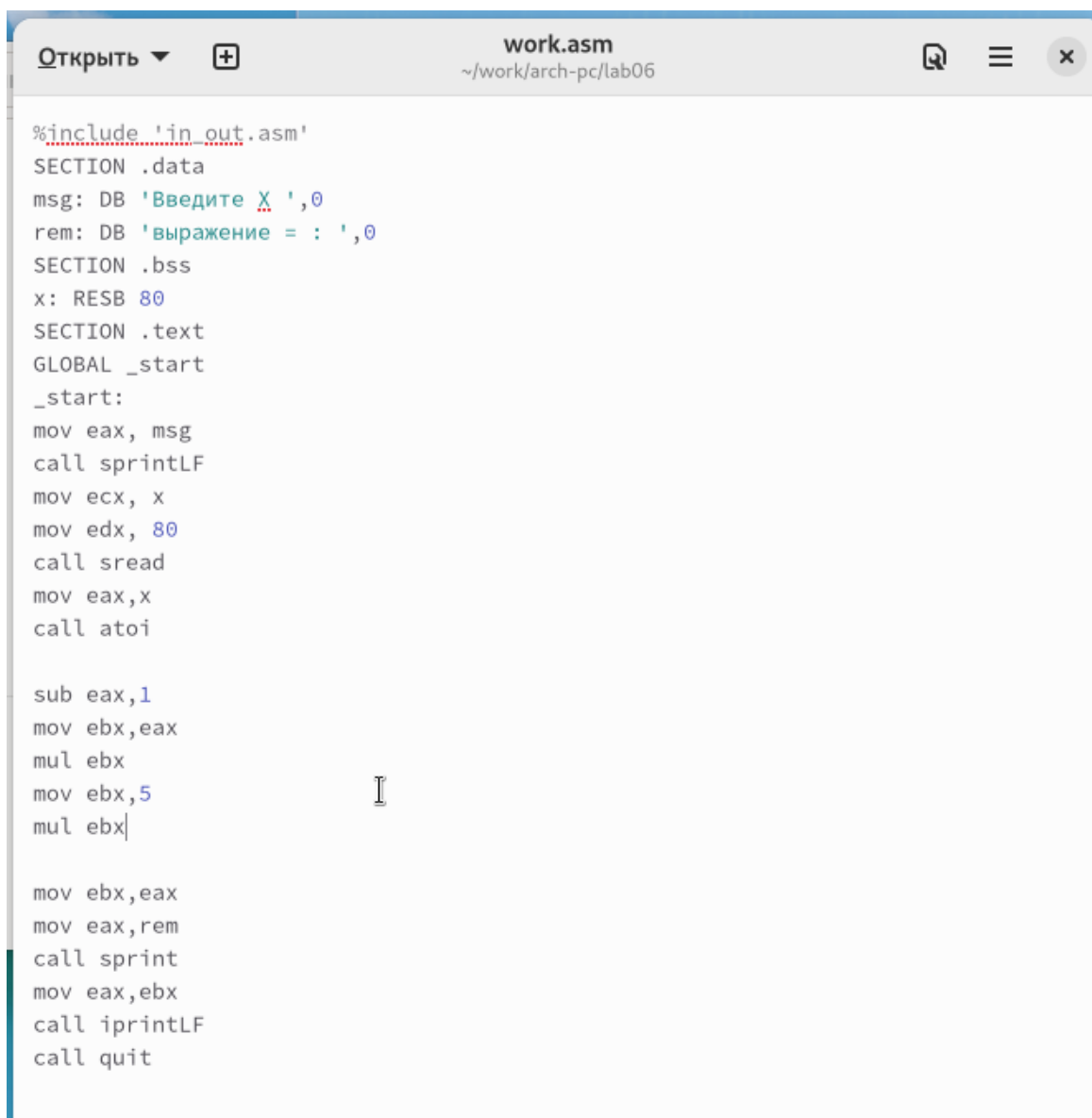
Получили вариант 7 -

$$5(x - 1)^2$$

для

$$x_1 = 3, x_2 = 5$$

(рис. [4.17]) (рис. [4.18])



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

%include 'in_out.asm'
SECTION .data
msg: DB 'Введите X ',0
rem: DB 'выражение = : ',0
SECTION .bss
x: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprintLF
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi

sub eax, 1
mov ebx, eax
mul ebx
mov ebx, 5
mul ebx

mov ebx, eax
mov eax, rem
call sprint
mov eax, ebx
call iprintLF
call quit
```

Рис. 4.17: Программа в файле work.asm

```
[nsplugin@fedora lab06]$ nasm -f elf work.asm
[nsplugin@fedora lab06]$ ld -m elf_i386 work.o -o work
[nsplugin@fedora lab06]$ ./work
Введите X
3
выражение = : 20
[nsplugin@fedora lab06]$ ./work
Введите X
5
выражение = : 80
[nsplugin@fedora lab06]$ █
```

Рис. 4.18: Запуск программы work.asm

5 Выводы

Изучили работу с арифметическими операциями.