

Отчёт по лабораторной работе 6

Дисциплина: архитектура компьютера

Ахмаров Роман

Содержание

1	Цель работы	5
2	Задание	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	16
4.3	ответы на вопросы по программе variant.asm	22
4.4	Задание для самостоятельной работы	23
5	Выводы	26
6	Список литературы	27

Список иллюстраций

4.1	Редактирую файл lab6-1.asm	10
4.2	Запуск файла lab6-1.asm	11
4.3	Редактирую файл lab6-1.asm	12
4.4	Запуск файла lab6-1.asm	13
4.5	Редактирую файл lab6-2.asm	14
4.6	Запуск файла lab6-2.asm	15
4.7	Редактирую файл lab6-2.asm	15
4.8	Запуск файла lab6-2.asm	16
4.9	Запуск файла lab6-2.asm	16
4.10	Редактирую файл lab6-3.asm	17
4.11	Запуск файла lab6-3.asm	18
4.12	Редактирую файл lab6-3.asm	19
4.13	Запуск файла lab6-3.asm	20
4.14	Редактирую файл variant.asm	21
4.15	Запуск файла variant.asm	22
4.16	Редактирую файл calc.asm	24
4.17	Запуск файла calc.asm	25

Список таблиц

1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

2 Задание

1. Изучение типов данных в ассемблере
2. Изучение арифметических операций в ассемблере
3. Выполнение заданий для самостоятельной работы

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:

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.
- 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 различные 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

Я создаю каталог для программ лабораторной работы № 6, перехожу в него и создаю файл lab6-1.asm.

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

В данной программе (рис. [4.1]) в регистр еах записывается символ 6 (`mov еах, '6'`), в регистр ебх символ 4 (`mov ебх, '4'`). Далее к значению в регистре еах прибавляем значение регистра ебх (`add еах, ебх`, результат сложения запишется в регистр еах). Далее выводим результат. Так как для работы функции `sprintf` в регистр еах должен быть записан адрес, необходимо использовать дополнительную переменную. Для этого запишем значение регистра еах в переменную `buf1` (`mov [buf1], еах`), а затем запишем адрес переменной `buf1` в регистр еах (`mov еах, buf1`) и вызовем функцию `sprintf`.

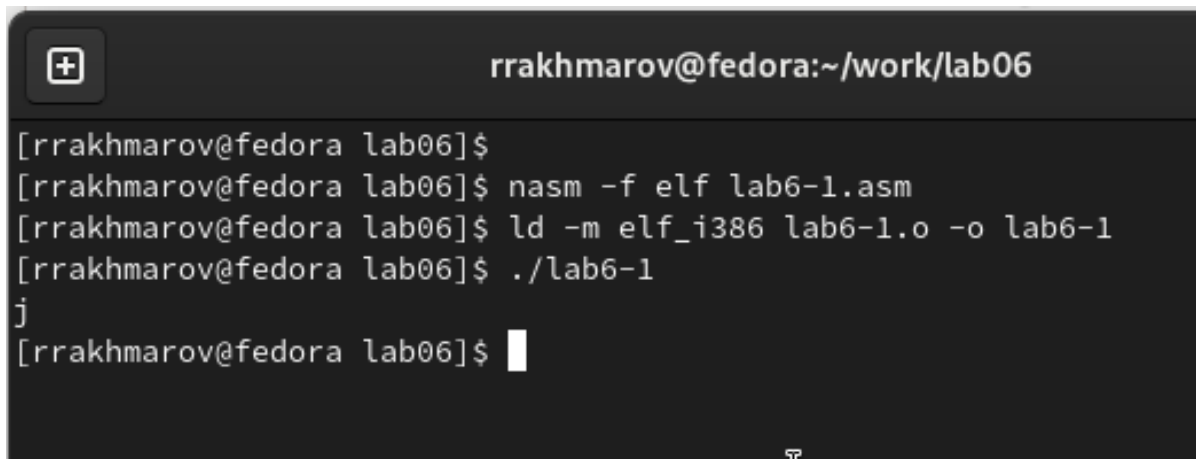
Открыть ▾ 

lab6-1.asm
~/work/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. 4.1: Редактирую файл lab6-1.asm

В данном случае при выводе значения регистра `eax` мы ожидаем увидеть число 10. Однако результатом будет символ `j`. (рис. [4.2])

A terminal window with a dark background. The title bar shows a plus icon and the text 'rrakhmarov@fedora:~/work/lab06'. The terminal content shows a series of commands: '[rrakhmarov@fedora lab06]\$' followed by 'nasm -f elf lab6-1.asm', 'ld -m elf_i386 lab6-1.o -o lab6-1', and './lab6-1'. The output of the last command is 'j'. The prompt returns to '[rrakhmarov@fedora lab06]\$' with a cursor.

```
rrakhmarov@fedora:~/work/lab06
[rrakhmarov@fedora lab06]$
[rrakhmarov@fedora lab06]$ nasm -f elf lab6-1.asm
[rrakhmarov@fedora lab06]$ ld -m elf_i386 lab6-1.o -o lab6-1
[rrakhmarov@fedora lab06]$ ./lab6-1
j
[rrakhmarov@fedora lab06]$
```

Рис. 4.2: Запуск файла lab6-1.asm

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`.

Далее изменяю текст программы и вместо символов, запишем в регистры числа. (рис. [4.3])

```
%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 sprintf
call quit
```

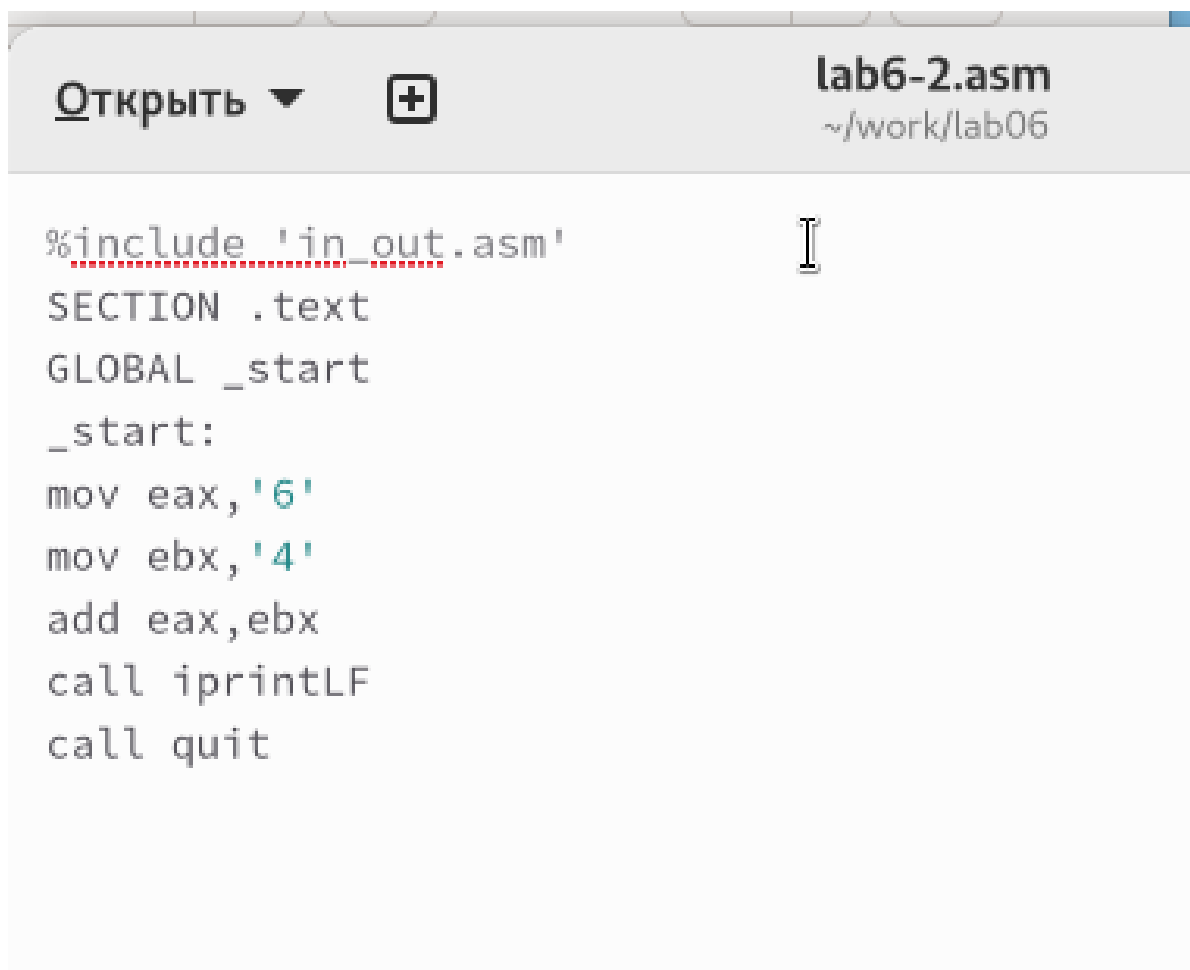
Рис. 4.3: Редактирую файл lab6-1.asm

Как и в предыдущем случае при исполнении программы мы не получим число 10. В данном случае выводится символ с кодом 10. (рис. [4.4]) Это символ конца строки (возврат каретки). В консоле он не отображается, но добавляет пустую строку.

```
[rrakhmarov@fedora lab06]$  
[rrakhmarov@fedora lab06]$ nasm -f elf lab6-1.asm  
[rrakhmarov@fedora lab06]$ ld -m elf_i386 lab6-1.o -o lab6-1  
[rrakhmarov@fedora lab06]$ ./lab6-1  
j  
[rrakhmarov@fedora lab06]$  
[rrakhmarov@fedora lab06]$ nasm -f elf lab6-1.asm  
[rrakhmarov@fedora lab06]$ ld -m elf_i386 lab6-1.o -o lab6-1  
[rrakhmarov@fedora lab06]$ ./lab6-1  
  
[rrakhmarov@fedora lab06]$
```

Рис. 4.4: Запуск файла lab6-1.asm

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. [4.5])



```
Открыть ▾  lab6-2.asm  
~/work/lab06  
  
%include 'in_out.asm'  
SECTION .text  
GLOBAL _start  
_start:  
mov eax, '6'  
mov ebx, '4'  
add eax, ebx  
call iprintLF  
call quit
```

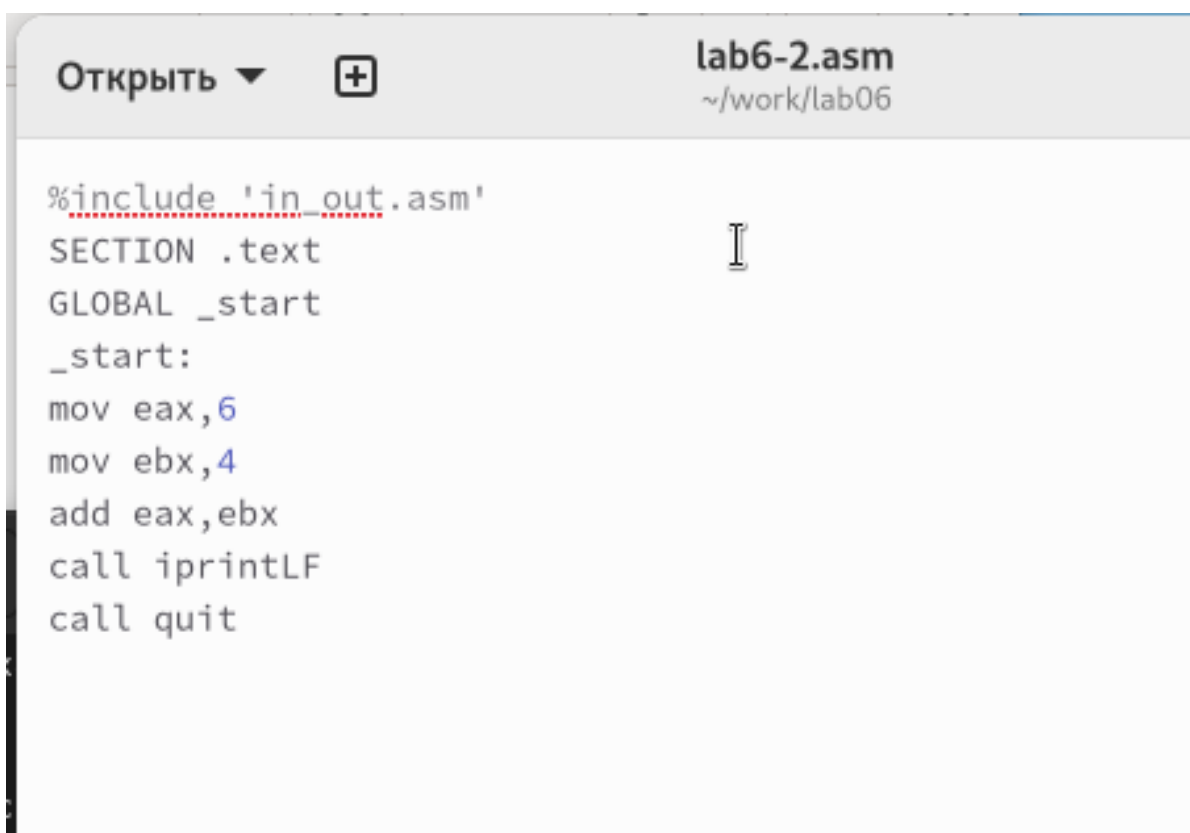
Рис. 4.5: Редактирую файл lab6-2.asm

В результате работы программы мы получим число 106. В данном случае, как и в первом, команда `add` складывает коды символов '6' и '4' ($54+52=106$). (рис. [4.6]) Однако, в отличие от прошлой программы, функция `iprintLF` позволяет вывести число, а не символ, кодом которого является это число.

```
[rrakhmarov@fedora lab06]$  
[rrakhmarov@fedora lab06]$ nasm -f elf lab6-2.asm  
[rrakhmarov@fedora lab06]$ ld -m elf_i386 lab6-2.o -o lab6-2  
[rrakhmarov@fedora lab06]$ ./lab6-2  
106  
[rrakhmarov@fedora lab06]$
```

Рис. 4.6: Запуск файла lab6-2.asm

Аналогично предыдущему примеру изменим символы на числа. (рис. [4.7])



```
lab6-2.asm  
~/work/lab06  
  
%include 'in_out.asm'  
SECTION .text  
GLOBAL _start  
_start:  
mov eax,6  
mov ebx,4  
add eax,ebx  
call iprintLF  
call quit
```

Рис. 4.7: Редактирую файл lab6-2.asm

Функция iprintLF позволяет вывести число и операндами были числа (а не коды символов). Поэтому получаем число 10. (рис. [4.8])

```

[rrakhmarov@fedora lab06]$
[rrakhmarov@fedora lab06]$ nasm -f elf lab6-2.asm
[rrakhmarov@fedora lab06]$ ld -m elf_i386 lab6-2.o -o lab6-2
[rrakhmarov@fedora lab06]$ ./lab6-2
106
[rrakhmarov@fedora lab06]$ nasm -f elf lab6-2.asm
[rrakhmarov@fedora lab06]$ ld -m elf_i386 lab6-2.o -o lab6-2
[rrakhmarov@fedora lab06]$ ./lab6-2
10
[rrakhmarov@fedora lab06]$ █

```

Рис. 4.8: Запуск файла lab6-2.asm

Заменяю функцию `iprintLF` на `iprint`. Создал исполняемый файл и запустил его. Вывод отличается тем, что нет переноса строки. (рис. [4.9])

```

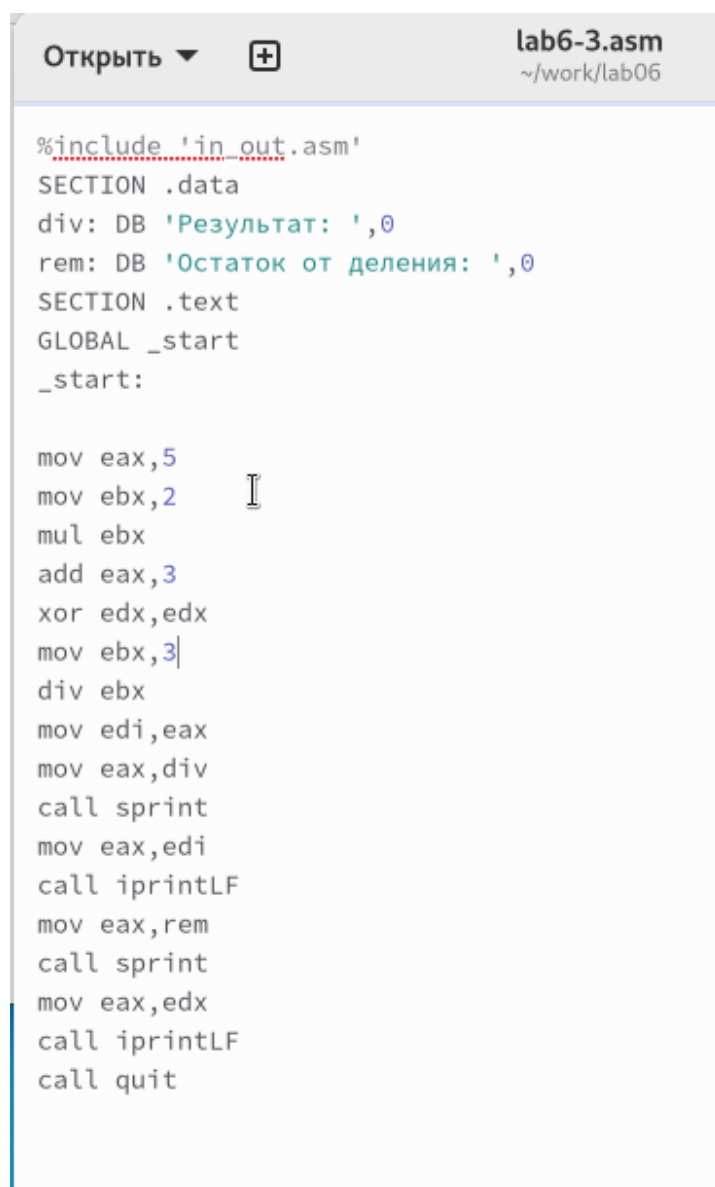
[rrakhmarov@fedora lab06]$
[rrakhmarov@fedora lab06]$ nasm -f elf lab6-2.asm
[rrakhmarov@fedora lab06]$ ld -m elf_i386 lab6-2.o -o lab6-2
[rrakhmarov@fedora lab06]$ ./lab6-2
106
[rrakhmarov@fedora lab06]$ nasm -f elf lab6-2.asm
[rrakhmarov@fedora lab06]$ ld -m elf_i386 lab6-2.o -o lab6-2
[rrakhmarov@fedora lab06]$ ./lab6-2
10
[rrakhmarov@fedora lab06]$ nasm -f elf lab6-2.asm
[rrakhmarov@fedora lab06]$ ld -m elf_i386 lab6-2.o -o lab6-2
[rrakhmarov@fedora lab06]$ ./lab6-2
10[rrakhmarov@fedora lab06]$
[rrakhmarov@fedora lab06]$ █

```

Рис. 4.9: Запуск файла lab6-2.asm

4.2 Выполнение арифметических операций в NASM

В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения $f(x) = (5 * 2 + 3) / 3$. (рис. [4.10] [4.11])



```
lab6-3.asm
~/.work/lab06

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

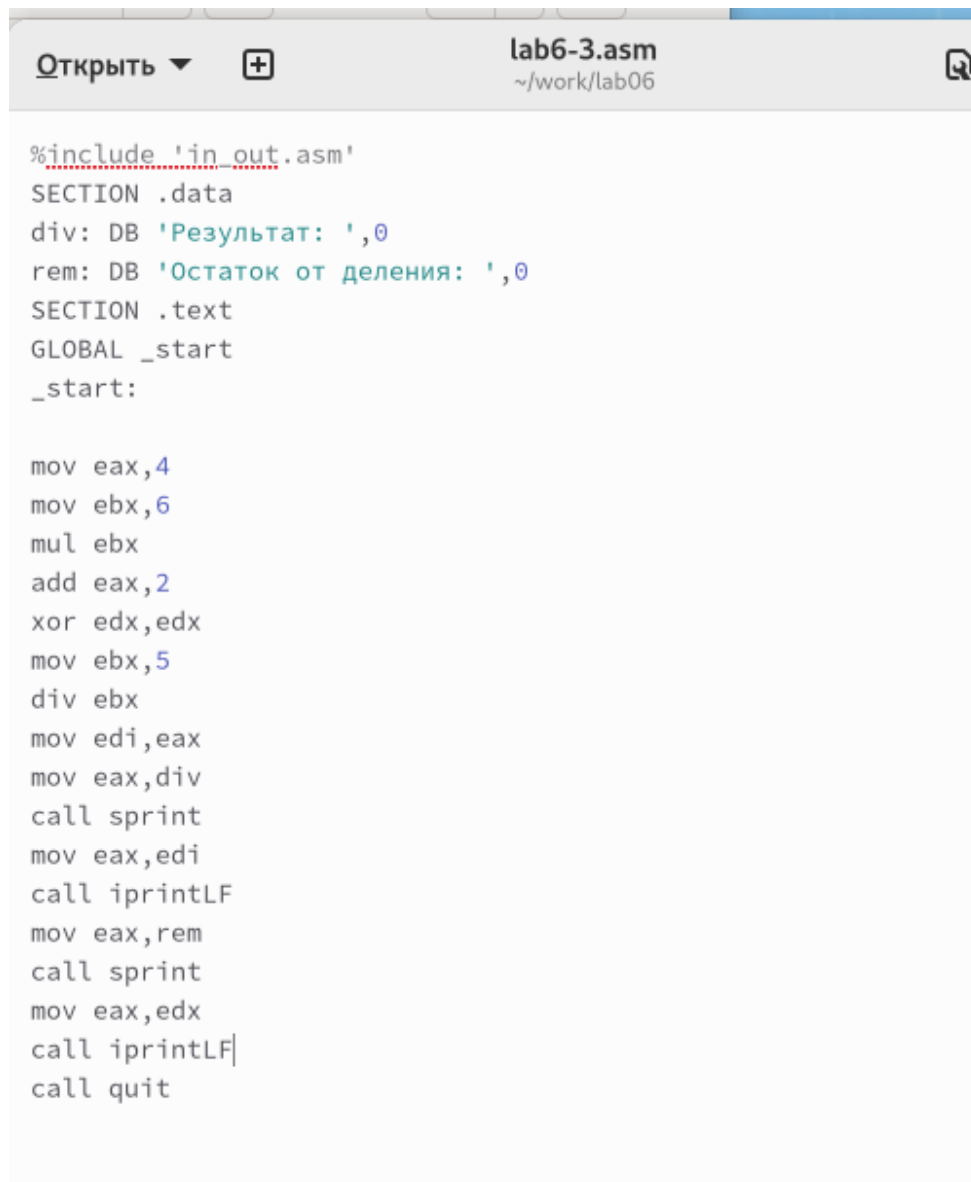
mov eax,5
mov ebx,2
mul ebx
add eax,3
xor edx,edx
mov ebx,3
div ebx
mov edi,eax
mov eax,div
call sprint
mov eax,edi
call iprintLF
mov eax,rem
call sprint
mov eax,edx
call iprintLF
call quit
```

Рис. 4.10: Редактирую файл lab6-3.asm

```
[rrakhmarov@fedora lab06]$  
[rrakhmarov@fedora lab06]$ nasm -f elf lab6-3.asm  
[rrakhmarov@fedora lab06]$ ld -m elf_i386 lab6-3.o -o lab6-3  
[rrakhmarov@fedora lab06]$ ./lab6-3  
Результат: 4  
Остаток от деления: 1  
[rrakhmarov@fedora lab06]$  
[rrakhmarov@fedora lab06]$
```

Рис. 4.11: Запуск файла lab6-3.asm

Изменил текст программы для вычисления выражения $f(x) = (4 * 6 + 2)/5$.
Создал исполняемый файл и проверил его работу. (рис. [4.12] [4.13])



```
%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
```

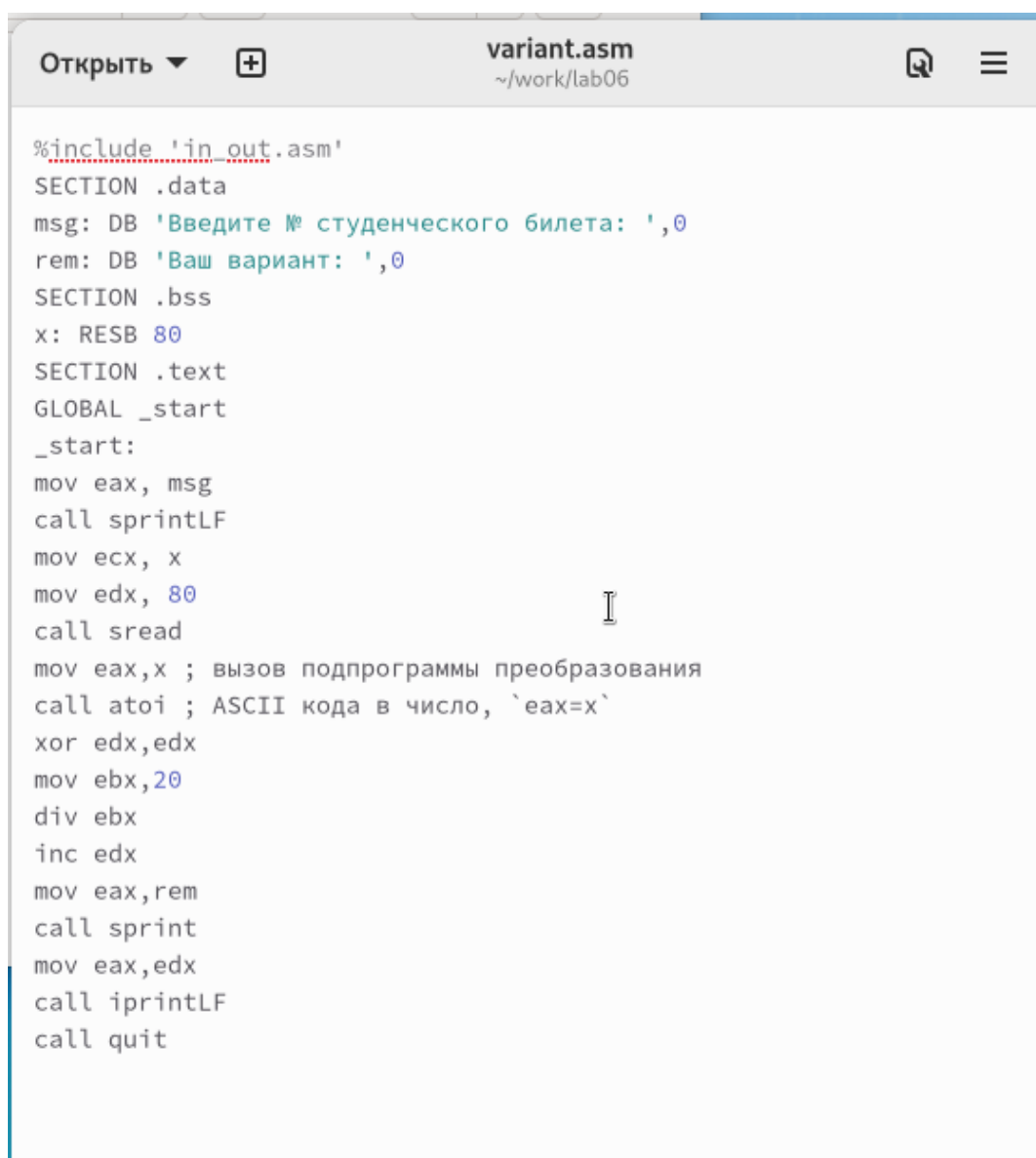
Рис. 4.12: Редактирую файл lab6-3.asm

```
[rrakhmarov@fedora lab06]$  
[rrakhmarov@fedora lab06]$ nasm -f elf lab6-3.asm  
[rrakhmarov@fedora lab06]$ ld -m elf_i386 lab6-3.o -o lab6-3  
[rrakhmarov@fedora lab06]$ ./lab6-3  
Результат: 5  
Остаток от деления: 1  
[rrakhmarov@fedora lab06]$  
[rrakhmarov@fedora lab06]$
```

Рис. 4.13: Запуск файла lab6-3.asm

В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета. (рис. [4.14])

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`. (рис. [4.15])



```
Открыть ▾ + variant.asm
~/work/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 ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x`
xor edx, edx
mov ebx, 20
div ebx
inc edx
mov eax, rem
call sprintf
mov eax, edx
call iprintLF
call quit
```

Рис. 4.14: Редактирую файл variant.asm

```

[rrakhmarov@fedora lab06]$
[rrakhmarov@fedora lab06]$ nasm -f elf variant.asm
[rrakhmarov@fedora lab06]$ ld -m elf_i386 variant.o -o variant
[rrakhmarov@fedora lab06]$ ./variant
Введите № студенческого билета:
1132232863
Ваш вариант: 4
[rrakhmarov@fedora lab06]$

```

Рис. 4.15: Запуск файла variant.asm

4.3 ответы на вопросы по программе variant.asm

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

Перекладывает значение переменной с фразой ‘Ваш вариант:’ в регистр eax:
`mov eax, rem`

Вызывает подпрограмму вывода строки: `call sprint`

2. Для чего используются следующие инструкции?

```

mov ecx, x
mov edx, 80
call read

```

Считывают значение студенческого билета в переменную X 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?

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

Выполняется деление номера студенческого билета на 20 и остаток сохраняется в регистре `edx`. Затем к остатку прибавляется 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 данном случае, она используется для добавления единицы к остатку от деления.

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

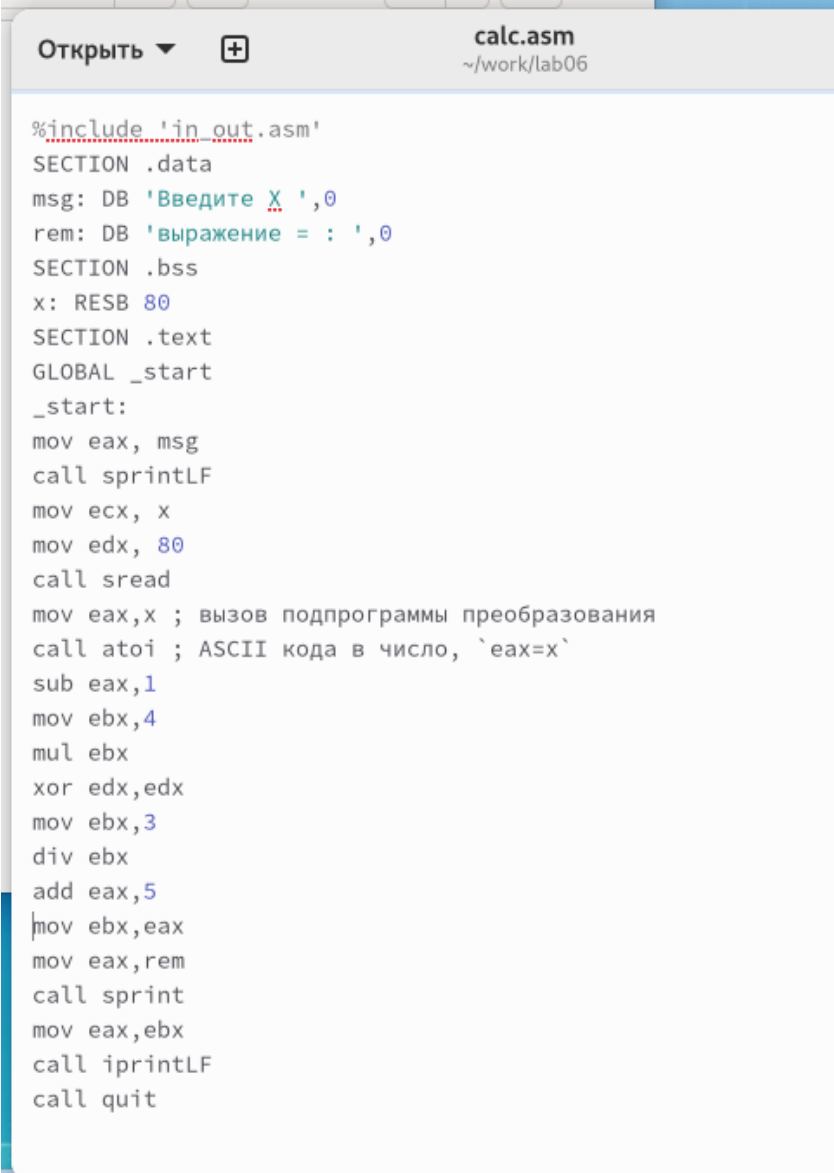
Результат вычислений перекладывается в регистр `eax`: `mov eax, edx`

Вызывается подпрограмма вывода строки: `call iprintLF`

4.4 Задание для самостоятельной работы

Написать программу вычисления выражения $y = f(x)$. Программа должна вывести 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. (рис. [4.16] [4.17])

Получили вариант $14 - 4/3(x - 1) + 5$ для $x = 4, x = 10$



```
Открыть ▾ + calc.asm
~/work/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 ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x`
sub eax,1
mov ebx,4
mul ebx
xor edx,edx
mov ebx,3
div ebx
add eax,5
mov ebx,eax
mov eax,rem
call sprint
mov eax,ebx
call iprintLF
call quit
```

Рис. 4.16: Редактирую файл calc.asm


```
[rrakhmarov@fedora lab06]$  
[rrakhmarov@fedora lab06]$ nasm -f elf calc.asm  
[rrakhmarov@fedora lab06]$ ld -m elf_i386 calc.o -o calc  
[rrakhmarov@fedora lab06]$ ./calc  
Введите X  
4  
выражение = : 9  
[rrakhmarov@fedora lab06]$ ./calc  
Введите X  
10  
выражение = : 17  
[rrakhmarov@fedora lab06]$
```

Рис. 4.17: Запуск файла calc.asm

Программа считает верно.

5 Выводы

Изучили работу с арифметическими операциями.

6 Список литературы

https://esystem.rudn.ru/pluginfile.php/1584628/mod_resource/content/1/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%965.pdf