

Отчёт по лабораторной работе №6

Дисциплина: архитектура компьютера

Хаджилари Гешлаг Никта

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.0.1	Символьные и численные данные в NASM	9
4.0.2	Выполнение арифметических операций в NASM	14
4.0.3	Ответы на вопросы по программе	20
5	Выполнение заданий для самостоятельной работы	21
6	Выводы	24
	Список литературы	25

Список иллюстраций

4.1	Перехожу в созданный каталог с помощью утилиты cd. С помощью утилиты touch создаю файл lab6-1.asm.	9
4.2	Копирование in_out.asm.	9
4.3	Открываю файл lab6-1.asm с помощью текстового редактора nano.	10
4.4	Запуск исполняемого файла.	10
4.5	Редактирование файла.	11
4.6	Запуск исполняемого файла.	11
4.7	Создание файла.	11
4.8	Редактирование файла.	12
4.9	Запуск исполняемого файла.	12
4.10	Редактирование файла.	13
4.11	Запуск исполняемого файла.	13
4.12	Редактирование файла.	14
4.13	Запуск исполняемого файла.	14
4.14	Создание и проверка файла.	15
4.15	Редактирование файла.	16
4.16	Запуск исполняемого файла.	16
4.17	Изменение программы.	17
4.18	Запуск исполняемого файла.	18
4.19	Создание файла.	18
4.20	Редактирование файл.	19
4.21	Запуск исполняемого файл.	19
5.1	Создание файла.	21
5.2	Написание программ.	22
5.3	Запуск исполняемого файла.	22
5.4	Запуск исполняемого файла.	23

Список таблиц

1 Цель работы

Цель данной лабораторной работы - освоение арифметических инструкций языка ассемблера NASM.

2 Задание

1. Символьные и численные данные в NASM
2. Выполнение арифметических операций в NASM
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.

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. 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. 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

4 Выполнение лабораторной работы

4.0.1 Символьные и численные данные в NASM

С помощью утилиты `mkdir` создаю директорию, в которой буду создавать файлы с программами для лабораторной работы №6

```
[nikta1382@fedora ~]$ mkdir -p ~/work/arch-pc/lab06
[nikta1382@fedora ~]$ cd ~/work/arch-pc/lab06
[nikta1382@fedora lab06]$ touch lab6-1.asm
[nikta1382@fedora lab06]$ ls
lab6-1.asm
[nikta1382@fedora lab06]$
```

Рис. 4.1: Перехожу в созданный каталог с помощью утилиты `cd`. С помощью утилиты `touch` создаю файл `lab6-1.asm`.

Копирую в текущий каталог файл `in_out.asm` с помощью утилиты `cp`, т.к. он будет использоваться в других программах

```
lab06]$ cp ~/Загрузки/in_out.asm in_out.asm
lab06]$
```

Рис. 4.2: Копирование `in_out.asm`.

Открываю созданный файл `lab6-1.asm`, вставляю в него программу вывода значения регистра `eax`

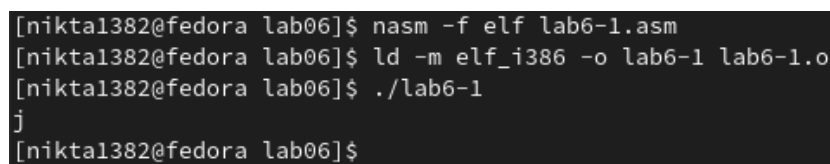


```
Открыть  lab6-1.asm
~/work/arch-pc/lab06

%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. 4.3: Открываю файл lab6-1.asm с помощью текстового редактора nano.

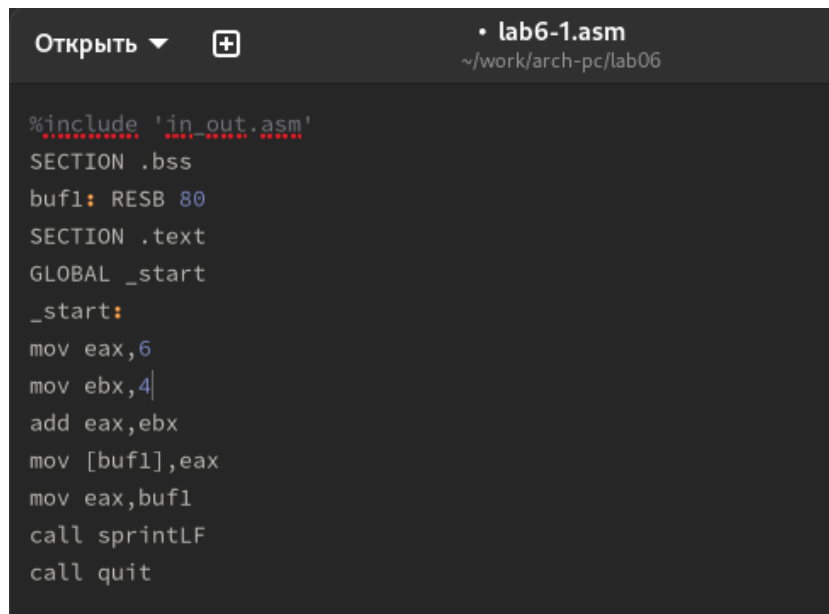
Создаю исполняемый файл программы и запускаю его. Вывод программы: символ j, потому что программа вывела символ, соответствующий по системе ASCII сумме двоичных кодов символов 4 и 6.



```
[nikta1382@fedora lab06]$ nasm -f elf lab6-1.asm
[nikta1382@fedora lab06]$ ld -m elf_i386 -o lab6-1 lab6-1.o
[nikta1382@fedora lab06]$ ./lab6-1
j
[nikta1382@fedora lab06]$
```

Рис. 4.4: Запуск исполняемого файла.

Изменяю в тексте программы символы “6” и “4” на цифры 6 и 4.

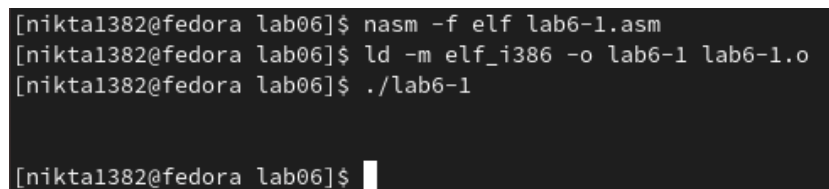


```
Открыть ▾ + • lab6-1.asm
~/work/arch-pc/lab06

%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. 4.5: Редактирование файла.

Создаю новый исполняемый файл программы и запускаю его. Теперь вывелся символ с кодом 10, это символ перевода строки, этот символ не отображается при выводе на экран.

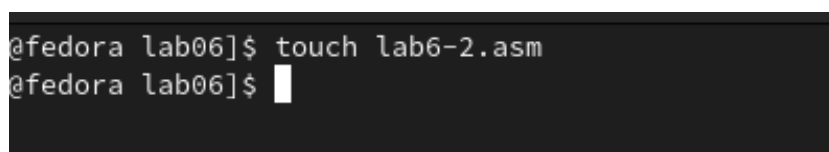


```
[nikta1382@fedora lab06]$ nasm -f elf lab6-1.asm
[nikta1382@fedora lab06]$ ld -m elf_i386 -o lab6-1 lab6-1.o
[nikta1382@fedora lab06]$ ./lab6-1

[nikta1382@fedora lab06]$
```

Рис. 4.6: Запуск исполняемого файла.

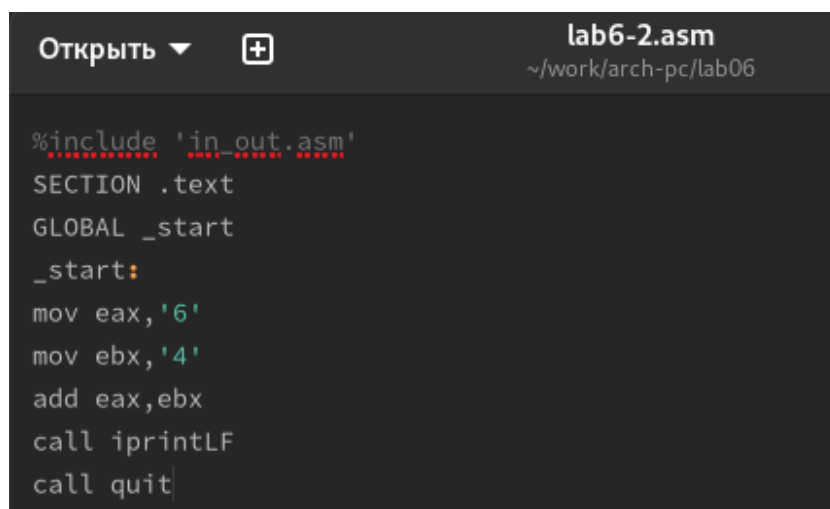
Создаю новый файл lab6-2.asm с помощью утилиты touch.



```
@fedora lab06]$ touch lab6-2.asm
@fedora lab06]$
```

Рис. 4.7: Создание файла.

Ввожу в файл текст другой программы для вывода значения регистра еах.

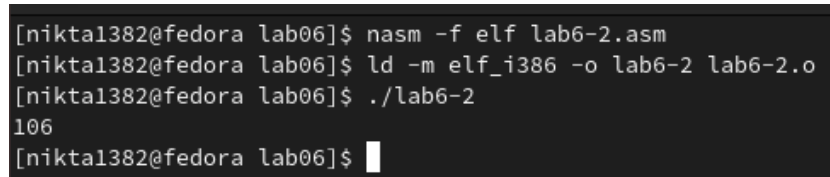


```
Открыть ▾ + lab6-2.asm
~/work/arch-пс/lab06

%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: Редактирование файла.

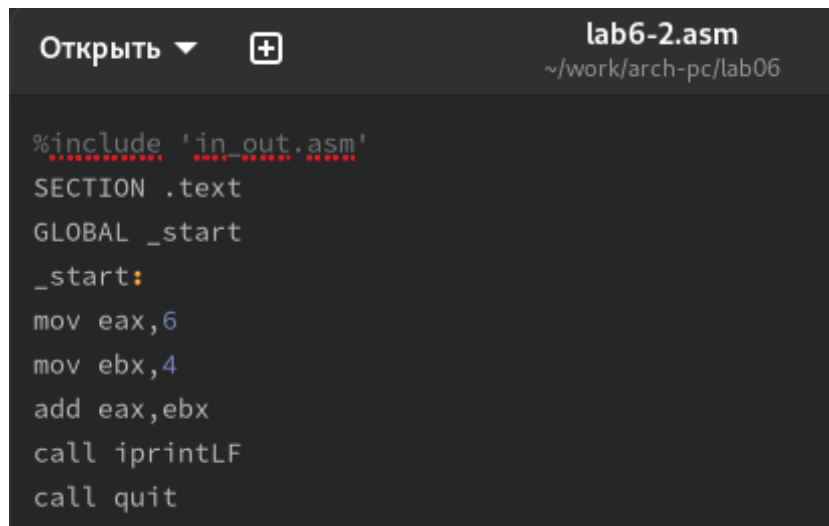
Создаю и запускаю исполняемый файл lab6-2. Теперь вывод число 106, потому что программа позволяет вывести именно число, а не символ, хотя все еще происходит именно сложение кодов символов “6” и “4”.



```
[nikta1382@fedora lab06]$ nasm -f elf lab6-2.asm
[nikta1382@fedora lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[nikta1382@fedora lab06]$ ./lab6-2
106
[nikta1382@fedora lab06]$
```

Рис. 4.9: Запуск исполняемого файла.

Заменяю в тексте программы в файле lab6-2.asm символы “6” и “4” на числа 6 и 4.




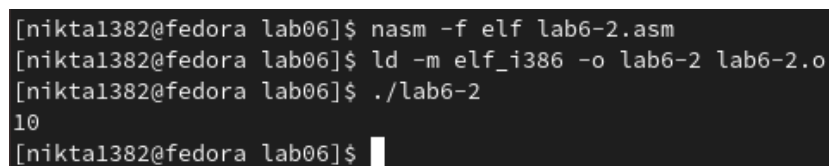
```
Открыть ▾  lab6-2.asm  
~/work/arch-pc/lab06  
  
%include 'in_out.asm'  
SECTION .text  
GLOBAL _start  
_start:  
mov eax,6  
mov ebx,4  
add eax,ebx  
call iprintLF  
call quit
```

Рис. 4.10: Редактирование файла.

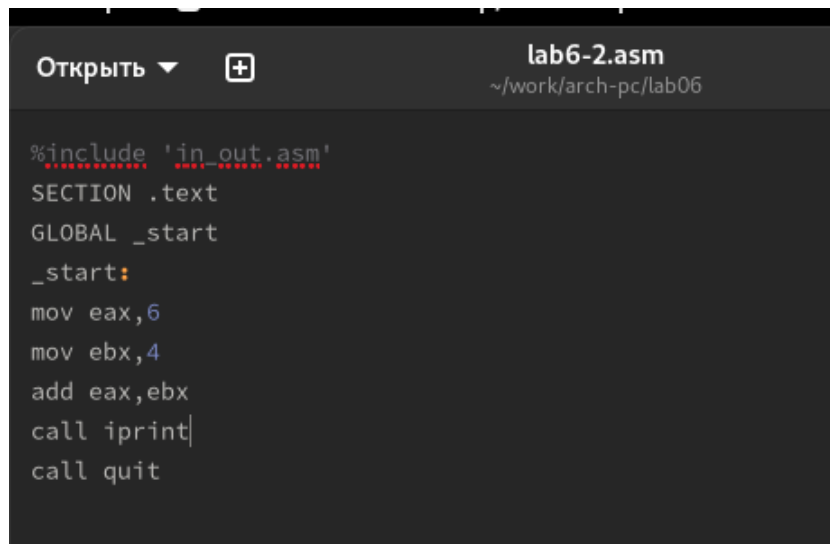
Создаю и запускаю новый исполняемый файл. Теперь программа складывает не соответствующие символам коды в системе ASCII, а сами числа, поэтому вывод 10.



```
[nikta1382@fedora lab06]$ nasm -f elf lab6-2.asm  
[nikta1382@fedora lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o  
[nikta1382@fedora lab06]$ ./lab6-2  
10  
[nikta1382@fedora lab06]$
```

Рис. 4.11: Запуск исполняемого файла.

Заменяю в тексте программы функцию iprintLF на iprint.

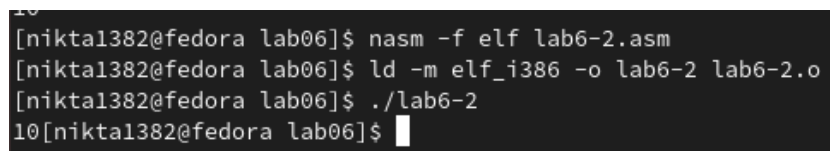


```
Открыть ▾ + lab6-2.asm
~/work/arch-pc/lab06

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

Рис. 4.12: Редактирование файла.

Создаю и запускаю новый исполняемый файл. Вывод не изменился, потому что символ переноса строки не отображался, когда программа исполнялась с функцией `iprintLF`, а `iprint` не добавляет к выводу символ переноса строки, в отличие от `iprintLF`.



```
10[nikta1382@fedora lab06]$ nasm -f elf lab6-2.asm
[nikta1382@fedora lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[nikta1382@fedora lab06]$ ./lab6-2
10[nikta1382@fedora lab06]$
```

Рис. 4.13: Запуск исполняемого файла.

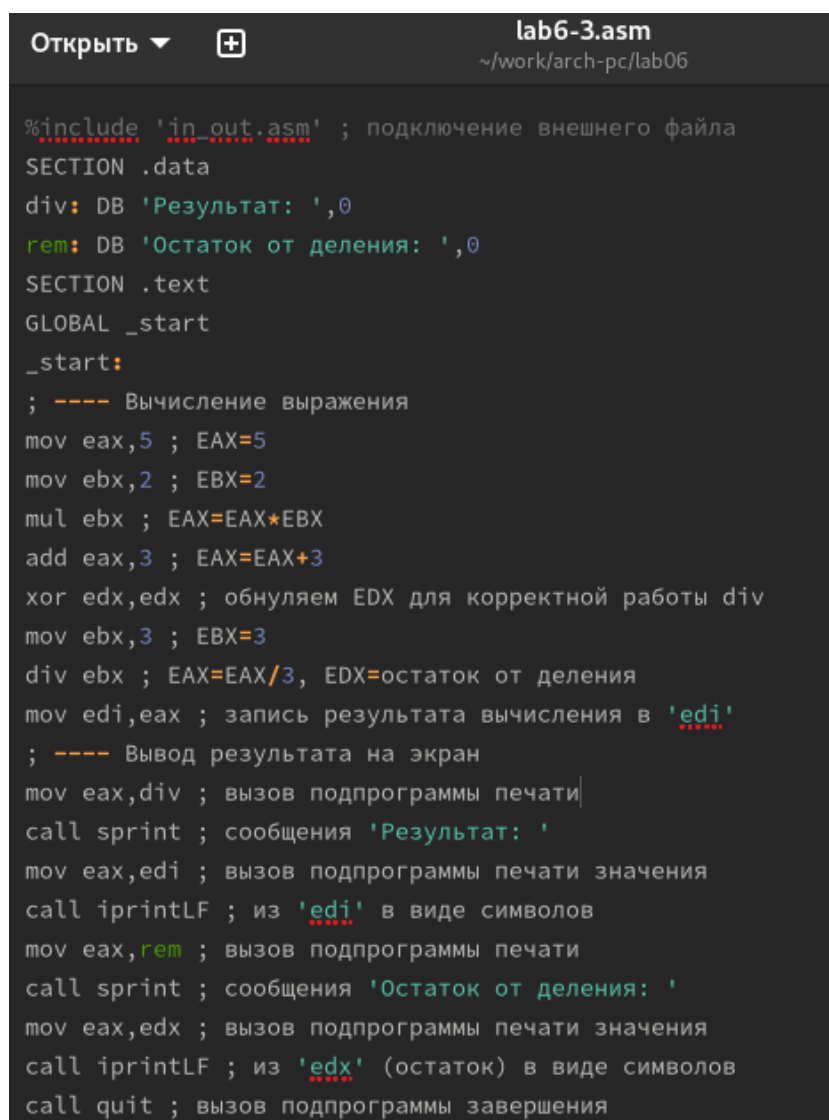
4.0.2 Выполнение арифметических операций в NASM

Создаю файл `lab6-3.asm` с помощью утилиты `touch`. И проверяю с помощью `ls`.

```
fedora lab06]$ touch lab6-3.asm
fedora lab06]$ ls
lab6-1.asm  lab6-2      lab6-2.o
lab6-1.o    lab6-2.asm  lab6-3.asm
fedora lab06]$
```

Рис. 4.14: Создание и проверка файла.

Ввожу в созданный файл текст программы для вычисления значения выражения $f(x) = (5 * 2 + 3)/3$

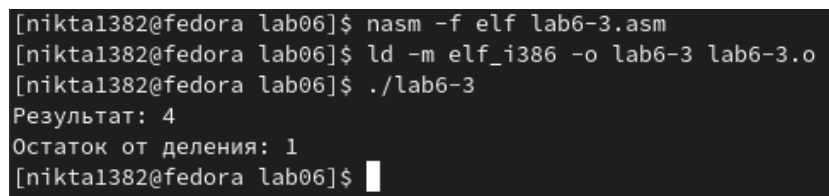


```
Открыть ▾ + lab6-3.asm
~/work/arch-pc/lab06

%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
```

Рис. 4.15: Редактирование файла.

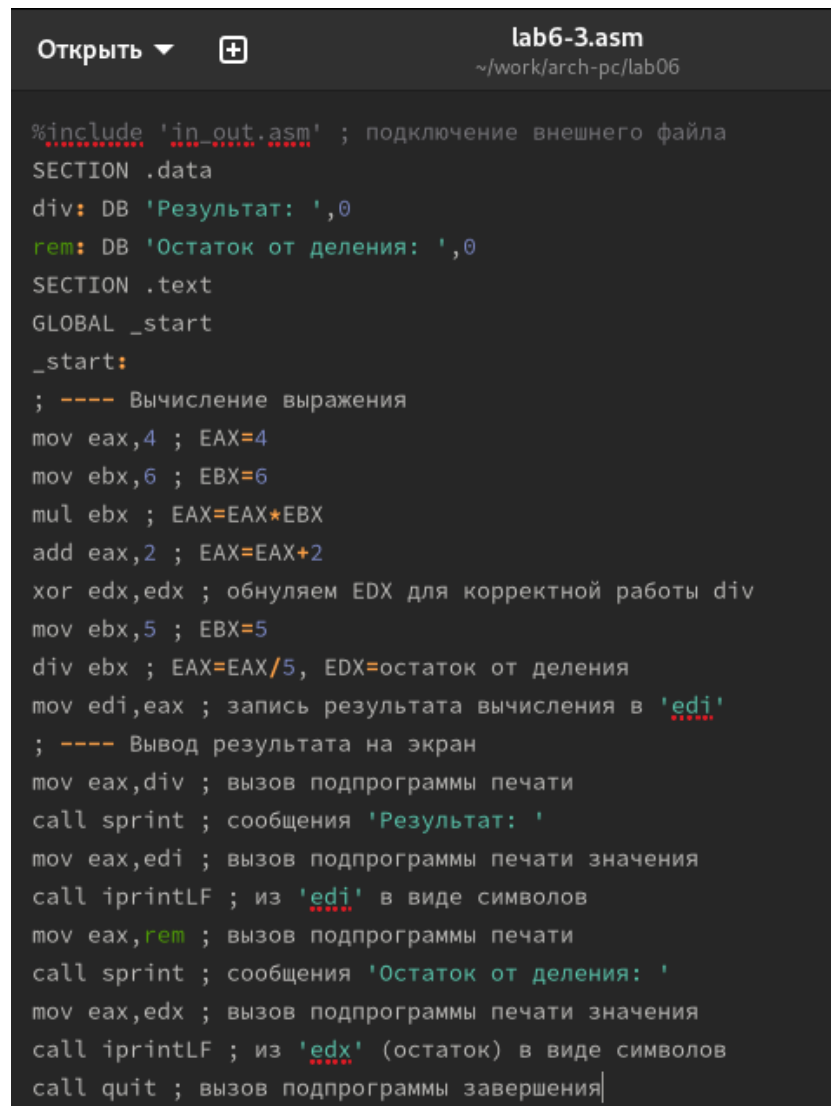
Создаю исполняемый файл и запускаю его.




```
[nikta1382@fedora lab06]$ nasm -f elf lab6-3.asm
[nikta1382@fedora lab06]$ ld -m elf_i386 -o lab6-3 lab6-3.o
[nikta1382@fedora lab06]$ ./lab6-3
Результат: 4
Остаток от деления: 1
[nikta1382@fedora lab06]$
```

Рис. 4.16: Запуск исполняемого файла.

Изменяю программу так, чтобы она вычисляла значение выражения $f(x) = (4 * 6 + 2) / 5$



The screenshot shows a text editor window titled 'lab6-3.asm' with the path '~/.work/arch-pc/lab06'. The code is written in assembly language and includes comments in Russian. It defines two data sections: '.data' for labels 'div' and 'rem', and '.text' for the main program logic. The logic calculates the expression (4 * 6 + 2) / 5 using x86 registers. It uses 'mov' to load constants, 'mul' for multiplication, 'add' for addition, 'xor' to clear the EDX register, and 'div' for division. The results are stored in 'edi' and 'edx' and printed using 'sprint' and 'iprintLF' subroutines. The program ends with a 'quit' call.

```
Открыть ▾  lab6-3.asm
~/.work/arch-pc/lab06

%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=4
mov ebx,6 ; EBX=6
mul ebx ; EAX=EAX*EBX
add eax,2 ; EAX=EAX+2
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,5 ; EBX=5
div ebx ; EAX=EAX/5, 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
```

Рис. 4.17: Изменение программы.

Создаю и запускаю новый исполняемый файл.

```
[nikta1382@fedora lab06]$ nasm -f elf lab6-3.asm
[nikta1382@fedora lab06]$ ld -m elf_i386 -o lab6-3 lab6-3.o
[nikta1382@fedora lab06]$ ./lab6-3
Результат: 5
Остаток от деления: 1
[nikta1382@fedora lab06]$
```

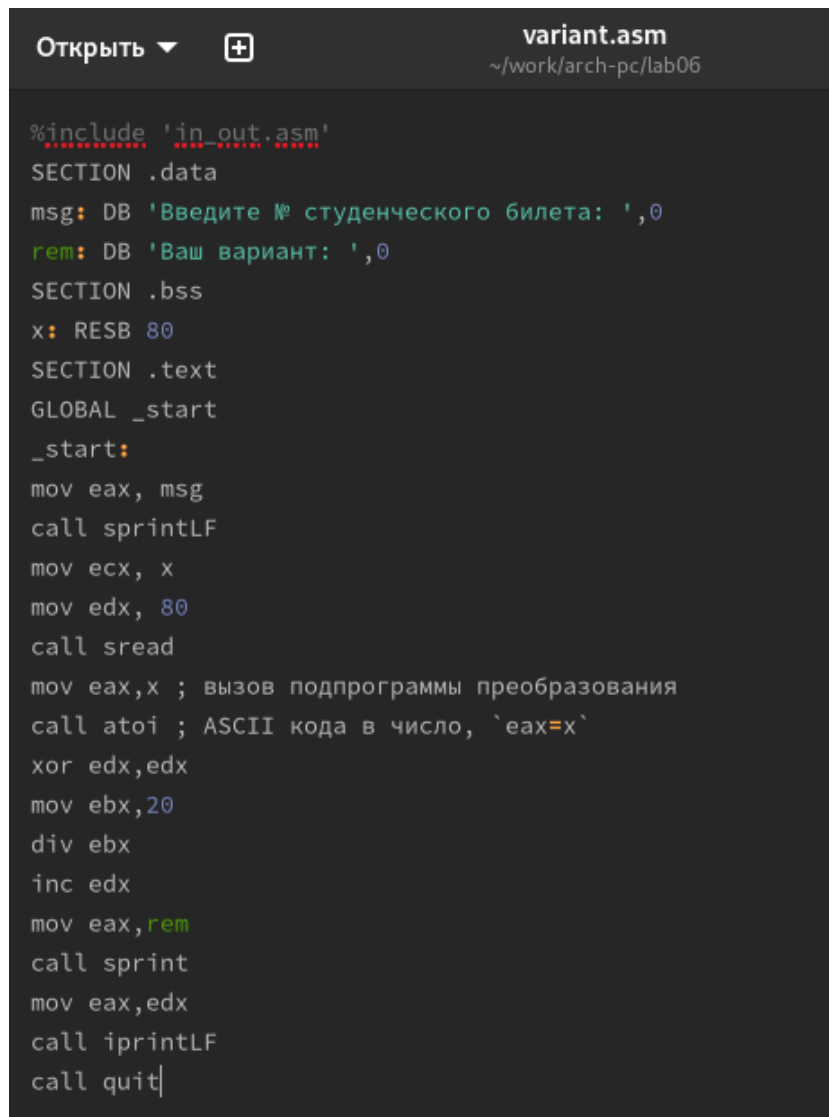
Рис. 4.18: Запуск исполняемого файла.

Создаю файл variant.asm с помощью утилиты touch.

```
edora lab06]$ touch variant.asm
edora lab06]$
```

Рис. 4.19: Создание файла.

Ввожу в файл текст программы для вычисления варианта задания по номеру студенческого билета.

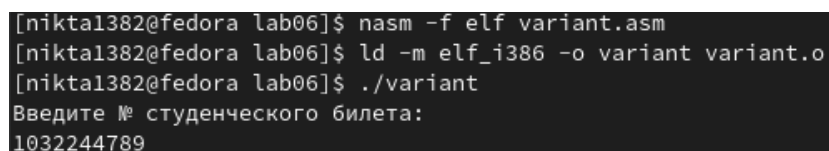


```
Открыть ▾ + variant.asm
~/work/arch-pc/lab06

%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 sprintLF
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 sprint
mov eax, edx
call iprintLF
call quit|
```

Рис. 4.20: Редактирование файл.

Создаю и запускаю исполняемый файл. Ввожу номер своего студенческого билета с клавиатуры, программа вывела, что мой вариант 20.



```
[nikta1382@fedora lab06]$ nasm -f elf variant.asm
[nikta1382@fedora lab06]$ ld -m elf_i386 -o variant variant.o
[nikta1382@fedora lab06]$ ./variant
Введите № студенческого билета:
1032244789
Ваш вариант: 20
```

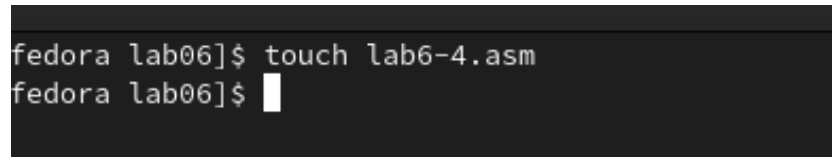
Рис. 4.21: Запуск исполняемого файл.

4.0.3 Ответы на вопросы по программе

1. За вывод сообщения “Ваш вариант” отвечают строки кода: `mov eax, ret` `call sprint`
2. Инструкция `mov ecx, x` используется, чтобы положить адрес вводимой строки `x` в регистр `ecx` `mov edx, 80` - запись в регистр `edx` длины вводимой строки `call sread` - вызов подпрограммы из внешнего файла, обеспечивающей ввод сообщения с клавиатуры.
3. `call atoi` используется для вызова подпрограммы из внешнего файла, которая преобразует `ascii`-код символа в целое число и записывает результат в регистр `eax`
4. За вычисления варианта отвечают строки: `xor edx,edx` ; обнуление `edx` для корректной работы `div` `mov ebx,20` ; `ebx = 20` `div ebx` ; `eax = eax/20`, `edx` - остаток от деления `inc edx` ; `edx = edx + 1`
5. При выполнении инструкции `div ebx` остаток от деления записывается в регистр `edx`
6. Инструкция `inc edx` увеличивает значение регистра `edx` на 1
7. За вывод на экран результатов вычислений отвечают строки: `mov eax,edx` `call iprintLF`

5 Выполнение заданий для самостоятельной работы

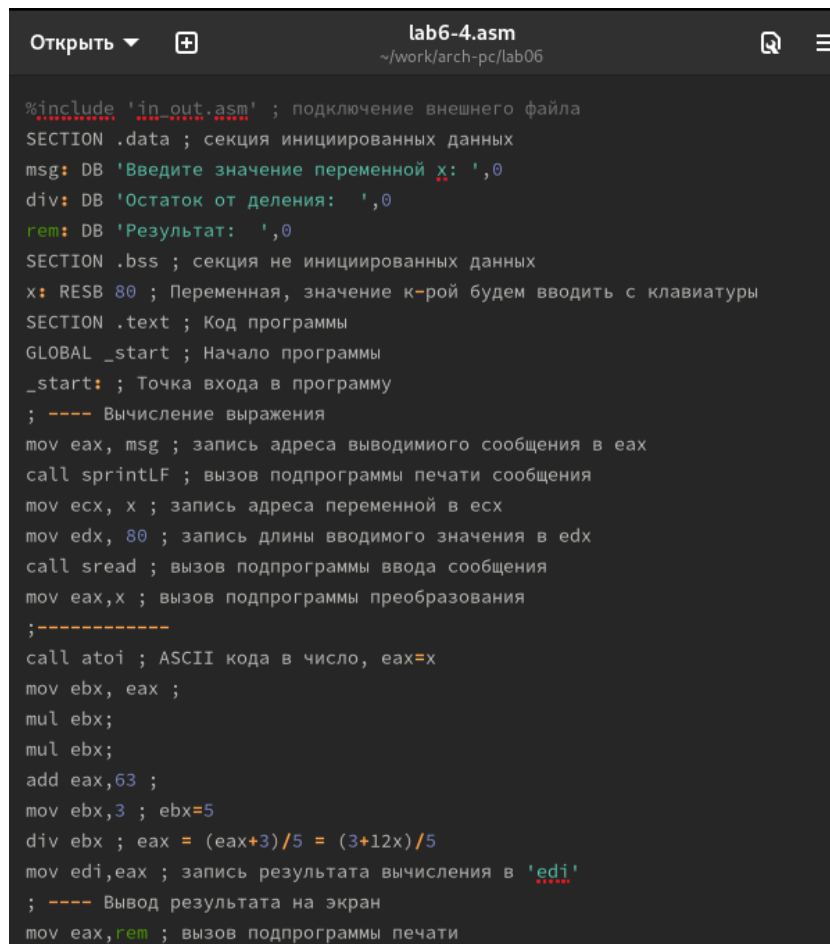
Создаю файл lab6-4.asm с помощью утилиты touch.



```
fedora lab06]$ touch lab6-4.asm
fedora lab06]$
```

Рис. 5.1: Создание файла.

Открываю созданный файл для редактирования, ввожу в него текст программы для вычисления значения выражения X в степени $3 * 1/3 + 21$. Это выражение было под вариантом 20.

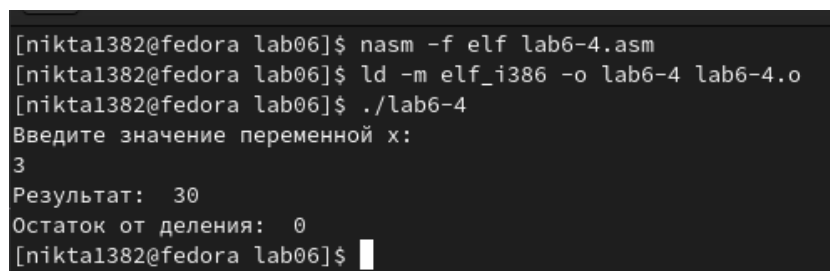


```
Открыть + lab6-4.asm ~/work/arch-pc/lab06

%include 'in_out.asm' ; подключение внешнего файла
SECTION .data ; секция иницированных данных
msg: DB 'Введите значение переменной x: ',0
div: DB 'Остаток от деления: ',0
rem: DB 'Результат: ',0
SECTION .bss ; секция не иницированных данных
x: RESB 80 ; Переменная, значение к-рой будем вводить с клавиатуры
SECTION .text ; Код программы
GLOBAL _start ; Начало программы
_start: ; Точка входа в программу
; ---- Вычисление выражения
mov eax, msg ; запись адреса выводимого сообщения в eax
call sprintf ; вызов подпрограммы печати сообщения
mov ecx, x ; запись адреса переменной в ecx
mov edx, 80 ; запись длины вводимого значения в edx
call sread ; вызов подпрограммы ввода сообщения
mov eax,x ; вызов подпрограммы преобразования
;-----
call atoi ; ASCII кода в число, eax=x
mov ebx, eax ;
mul ebx;
mul ebx;
add eax,63 ;
mov ebx,3 ; ebx=5
div ebx ; eax = (eax+3)/5 = (3+12x)/5
mov edi,eax ; запись результата вычисления в 'edi'
; ---- Вывод результата на экран
mov eax,rem ; вызов подпрограммы печати
```

Рис. 5.2: Написание программ.

Создаю и запускаю исполняемый файл. При вводе значения 1, вывод 21.



```
[nikta1382@fedora lab06]$ nasm -f elf lab6-4.asm
[nikta1382@fedora lab06]$ ld -m elf_i386 -o lab6-4 lab6-4.o
[nikta1382@fedora lab06]$ ./lab6-4
Введите значение переменной x:
3
Результат: 30
Остаток от деления: 0
[nikta1382@fedora lab06]$
```

Рис. 5.3: Запуск исполняемого файла.

Провожу еще один запуск исполняемого файла для проверки работы программы с другим значением на входе. Программа отработала верно.

```
[nikta1382@fedora lab06]$ nasm -f elf lab6-4.asm
[nikta1382@fedora lab06]$ ld -m elf_i386 -o lab6-4 lab6-4.o
[nikta1382@fedora lab06]$ ./lab6-4
Введите значение переменной x:
1
Результат: 21
Остаток от деления: 1
[nikta1382@fedora lab06]$
```

Рис. 5.4: Запуск исполняемого файла.

6 Выводы

При выполнении данной лабораторной работы я освоил арифметические инструкции языка ассемблера NASM.

Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learning-bash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ-Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).