

Лабораторная работа №6

Арифметические операции в NASM.

Дагделен Зейнап Реджеповна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
3.1	Адресация в NASM	7
3.2	Арифметические операции в NASM	7
3.2.1	Целочисленное сложение add.	7
3.2.2	Целочисленное вычитание sub.	8
3.2.3	Команды инкремента и декремента.	8
3.2.4	Команда изменения знака операнда neg.	8
3.2.5	Команды деления div и idiv.	9
3.3	Перевод символа числа в десятичную символьную запись	10
4	Выполнение лабораторной работы	11
4.1	Символьные и численные данные в NASM	11
4.2	Выполнение арифметических операций в NASM	17
4.3	Ответы на вопросы:	22
4.4	Задание для самостоятельной работы	23
5	Выводы	27
6	Список литературы	28

Список иллюстраций

4.1	Создание каталога и файла в нем	11
4.2	Текст программы	12
4.3	Создание исполняемого файла и его запуск	12
4.4	Текст программы	13
4.5	Создание исполняемого файла и его запуск	13
4.6	Создание файла lab-2.asm	14
4.7	Текст программы	14
4.8	Создание исполняемого файла и его запуск	14
4.9	Текст программы	15
4.10	Создание исполняемого файла и его запуск	15
4.11	Текст программы после изменений	16
4.12	Создание исполняемого файла и его запуск	16
4.13	Создание файла lab6-3.asm	17
4.14	Текст программы	18
4.15	Создание исполняемого файла и его запуск	18
4.16	Текст программы	19
4.17	Создание исполняемого файла и его запуск	19
4.18	Текст программы	21
4.19	Создание файла variant.asm, его исполняемого файла и его запуск	22
4.20	Текст программы	24
4.21	Создание исполняемого файла и его запуск	24

Список таблиц

1 Цель 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

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.

3.2 Арифметические операции в NASM

3.2.1 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 ,

Допустимые сочетания операндов для команды add аналогичны сочетаниям операндов для команды mov.

3.2.2 Целочисленное вычитание sub.

Команда целочисленного вычитания sub (от англ. subtraction – вычитание) работает аналогично команде add и выглядит следующим образом:

sub ,

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 (от англ. increment) и dec (от англ. decrement), 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.

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.

3.2.5 Команды деления `div` и `idiv`.

Для деления, как и для умножения, существует 2 команды `div` (от англ. divide – деление) 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. Согласно стандарту ASCII каждый символ кодируется одним байтом.

Среди инструкций NASM нет такой, которая выводит числа (не в символьном виде). Для решения этой проблемы необходимо проводить преобразование ASCII символов в числа и обратно.

4 Выполнение лабораторной работы

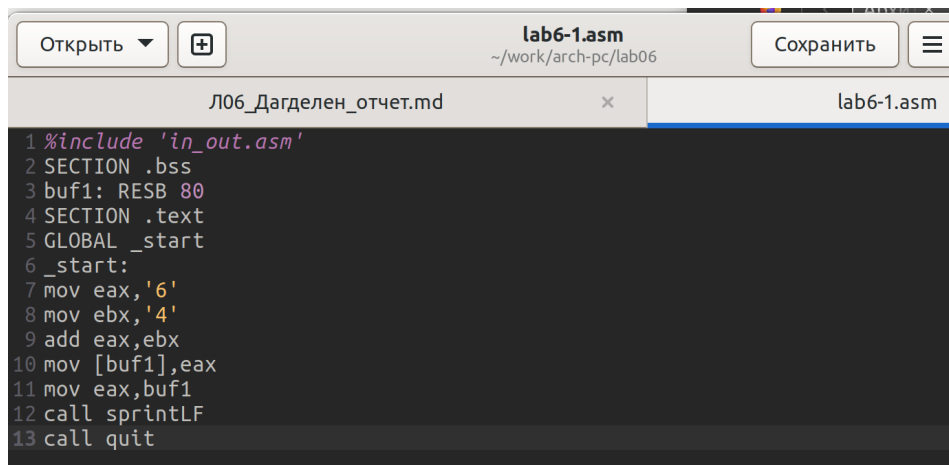
4.1 Символьные и численные данные в NASM

1. Создаю каталог для программ лабораторной работы № 6 с помощью команды `mkdir`, и, перейдя в него, создаю файл `lab6-1.asm` с помощью `touch` (рис. [4.1]).

```
zrdagdelen@zrdagdelen:~$ mkdir ~/work/arch-pc/lab06
zrdagdelen@zrdagdelen:~$ cd ~/work/arch-pc/lab06
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ls
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ touch lab6-1.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ls
lab6-1.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ □
```

Рис. 4.1: Создание каталога и файла в нем

2. Рассмотрю примеры программ вывода символьных и численных значений. Программы будут выводить значения записанные в регистр `eax`. Ввожу в файл `lab6-1.asm` текст программы из листинга 6.1 (рис. [4.2]).



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

Рис. 4.2: Текст программы

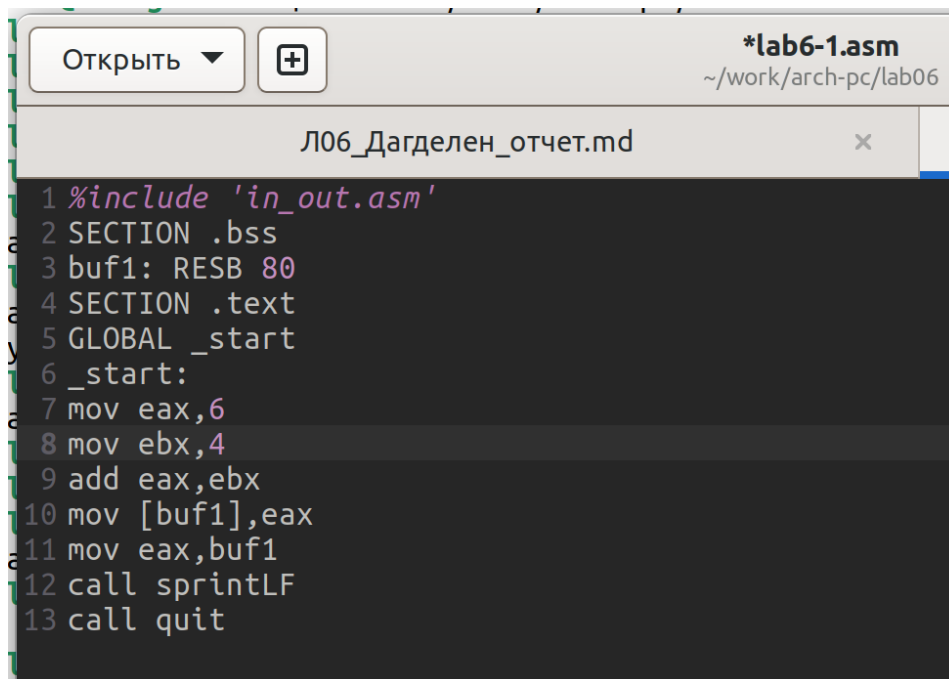
Создаю исполняемый файл и запускаю его(рис. [4.3]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ls
in_out.asm lab6-1.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ nasm -f elf lab6-1.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-1 lab6-1.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ls
in_out.asm lab6-1 lab6-1.asm lab6-1.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ./lab6-1
j
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$
```

Рис. 4.3: Создание исполняемого файла и его запуск

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

3. Далее изменяю текст программы и вместо символов, записываю в регистры числа. Изменяю текст программы (Листинг 6.1) следующим образом (рис. [4.4]).



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

Рис. 4.4: Текст программы

Создаю исполняемый файл и запускаю его(рис. [4.5]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ nasm -f elf lab6-1.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-1 lab6-1.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ./lab6-1

zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$
```

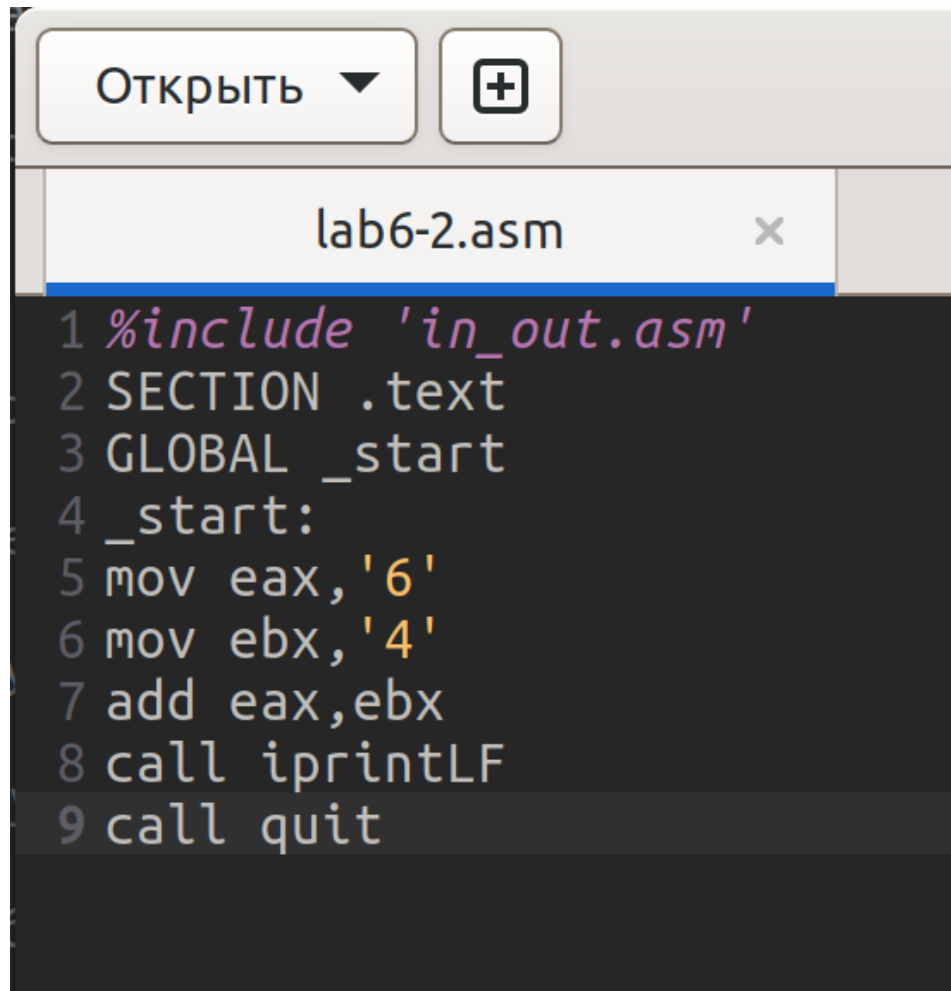
Рис. 4.5: Создание исполняемого файла и его запуск

Как и в предыдущем случае при исполнении программы я не получила число 10. В данном случае выводится символ с кодом 10. Это символ перевода строки, к сожалению, он не отображается при выводе на экран.

4. Создаю файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 с помощью touch и ввожу в него текст программы из листинга 6.2(рис. [4.6]- [4.7]). Создаю исполняемый файл и запускаю его (рис. [4.8]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ touch ~/work/arch-pc/lab06/lab6-2.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ls
in_out.asm lab6-1 lab6-1.asm lab6-1.o lab6-2.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$
```

Рис. 4.6: Создание файла lab-2.asm



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

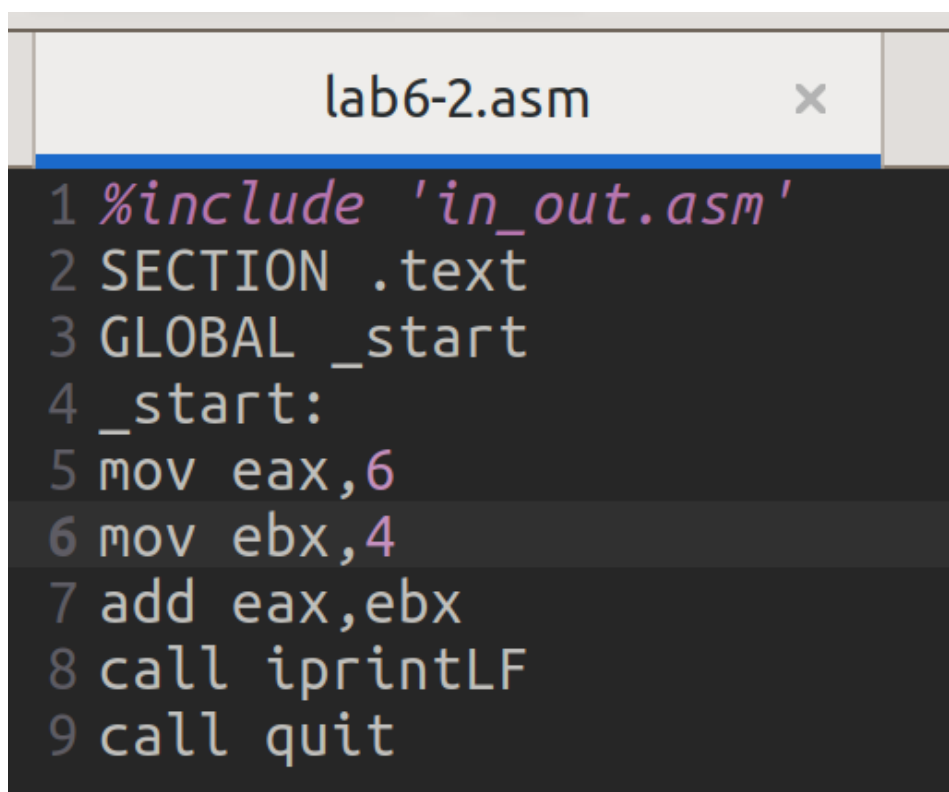
Рис. 4.7: Текст программы

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ls
in_out.asm lab6-1 lab6-1.asm lab6-1.o lab6-2 lab6-2.asm lab6-2.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ./lab6-2
106
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$
```

Рис. 4.8: Создание исполняемого файла и его запуск

В результате работы программы я получила число 106. В данном случае, как и в первом, команда `add` складывает коды символов '6' и '4' ($54+52=106$). Однако, в отличие от программы из листинга 6.1, функция `iprintLF` позволяет вывести число, а не символ, кодом которого является это число.

5. Аналогично предыдущему примеру изменю символы на числа. Заменяю строки (рис. [4.9]).



```
lab6-2.asm
1 %include 'in_out.asm'
2 SECTION .text
3 GLOBAL _start
4 _start:
5 mov eax,6
6 mov ebx,4
7 add eax,ebx
8 call iprintLF
9 call quit
```

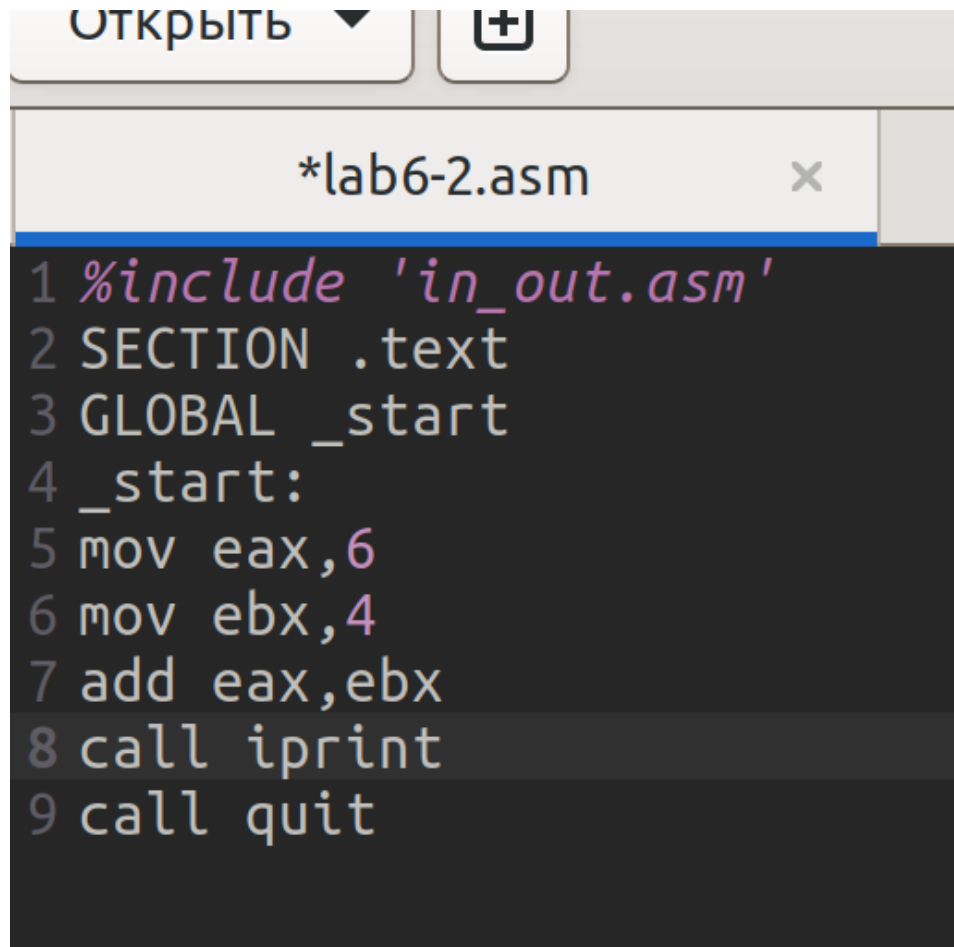
Рис. 4.9: Текст программы

Создаю исполняемый файл и запускаю его(рис. [4.10]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ./lab6-2
10
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$
```

Рис. 4.10: Создание исполняемого файла и его запуск

Теперь программа складывает не коды, соответствующие символам в системе ASCII, а сами числа, поэтому вывод 10. Заменяю функцию `iprintLF` на `iprint` (рис. [4.11]).



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

Рис. 4.11: Текст программы после изменений

Создаю исполняемый файл и запускаю его (рис. [4.12]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ./lab6-2
10zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$
```

Рис. 4.12: Создание исполняемого файла и его запуск

Чем отличается вывод функций `iprintLF` и `iprint`? 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`.

4.2 Выполнение арифметических операций в NASM

6. В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения $f(x) = (5 \times 2 + 3)/3$. Создаю файл `lab6-3.asm` в каталоге `~/work/arch-pc/lab06` с помощью `touch` (рис. [4.13]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ touch ~/work/arch-pc/lab06/lab6-3.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ls
in_out.asm lab6-1 lab6-1.asm lab6-1.o lab6-2 lab6-2.asm lab6-2.o lab6-3.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$
```

Рис. 4.13: Создание файла `lab6-3.asm`

Ввожу текст программы из листинга 6.3 в `lab6-3.asm` (рис. [4.14]).

```
lab6-3.asm x lab6-2.asm x Л06_Дагделен
1 ;-----
2 ; Программа вычисления выражения (5 * 2 + 3)/3
3 ;-----
4 %include 'in_out.asm' ; подключение внешнего файла
5 SECTION .data
6 div: DB 'Результат: ',0
7 rem: DB 'Остаток от деления: ',0
8 SECTION .text
9 GLOBAL _start
10 _start:
11 ; ---- Вычисление выражения
12 mov eax,5 ; EAX=5
13 mov ebx,2 ; EBX=2
14 mul ebx ; EAX=EAX*EBX
15 add eax,3 ; EAX=EAX+3
16 xor edx,edx ; обнуляем EDX для корректной работы div
17 mov ebx,3 ; EBX=3
18 div ebx ; EAX=EAX/3, EDX=остаток от деления
19 mov edi,eax ; запись результата вычисления в 'edi'
20 ; ---- Вывод результата на экран
21 mov eax,div ; вызов подпрограммы печати
22 call sprint ; сообщения 'Результат: '
23 mov eax,edi ; вызов подпрограммы печати значения
24 call iprintLF ; из 'edi' в виде символов
25 mov eax,rem ; вызов подпрограммы печати
26 call sprint ; сообщения 'Остаток от деления: '
27 mov eax,edx ; вызов подпрограммы печати значения
28 call iprintLF ; из 'edx' (остаток) в виде символов
29 call quit ; вызов подпрограммы завершения
```

Рис. 4.14: Текст программы

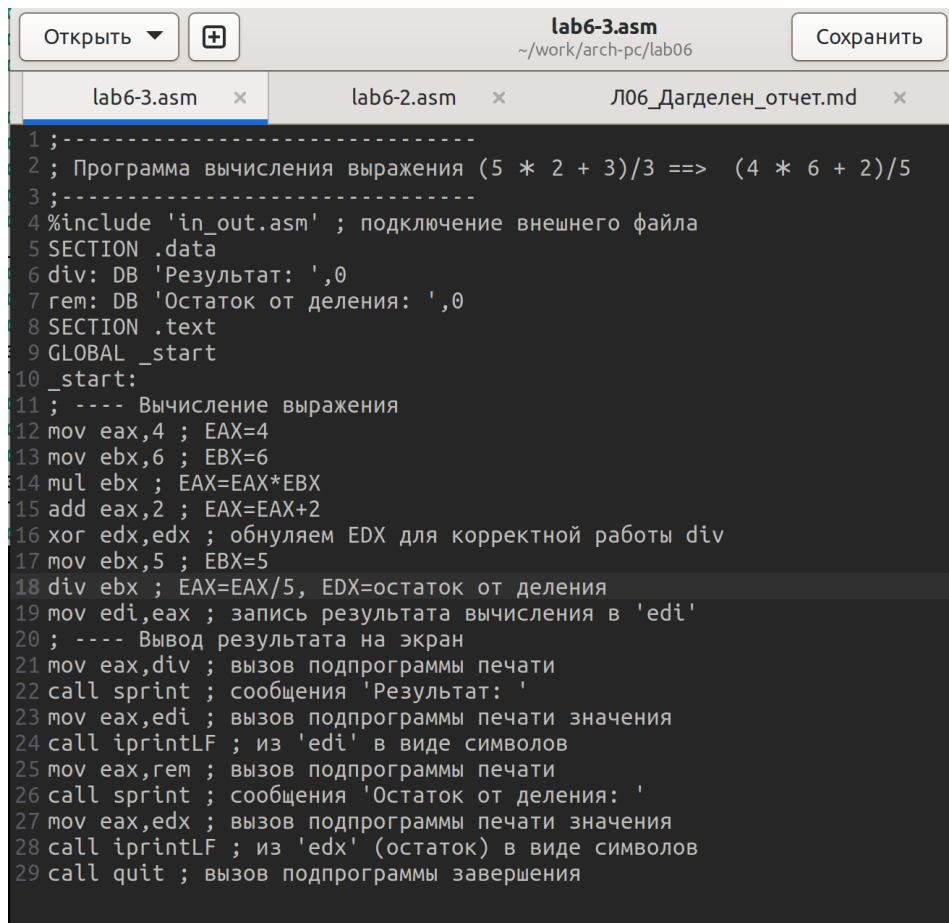
Создаю исполняемый файл и запускаю его(рис. [4.15]).

```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ./lab6-3
Результат: 4
Остаток от деления: 1
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$
```

Рис. 4.15: Создание исполняемого файла и его запуск

Результат работы программы следующий: результат = 4, остаток = 1.

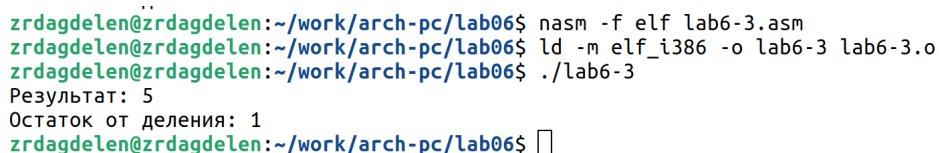
Изменяю текст программы для вычисления выражения $f(x) = (4 \times 6 + 2)/5$ (рис. [4.16]).



```
1 ;-----
2 ; Программа вычисления выражения (5 * 2 + 3)/3 ==> (4 * 6 + 2)/5
3 ;-----
4 %include 'in_out.asm' ; подключение внешнего файла
5 SECTION .data
6 div: DB 'Результат: ',0
7 rem: DB 'Остаток от деления: ',0
8 SECTION .text
9 GLOBAL _start
10 _start:
11 ; ---- Вычисление выражения
12 mov eax,4 ; EAX=4
13 mov ebx,6 ; EBX=6
14 mul ebx ; EAX=EAX*EBX
15 add eax,2 ; EAX=EAX+2
16 xor edx,edx ; обнуляем EDX для корректной работы div
17 mov ebx,5 ; EBX=5
18 div ebx ; EAX=EAX/5, EDX=остаток от деления
19 mov edi,eax ; запись результата вычисления в 'edi'
20 ; ---- Вывод результата на экран
21 mov eax,div ; вызов подпрограммы печати
22 call sprint ; сообщения 'Результат: '
23 mov eax,edi ; вызов подпрограммы печати значения
24 call iprintLF ; из 'edi' в виде символов
25 mov eax,rem ; вызов подпрограммы печати
26 call sprint ; сообщения 'Остаток от деления: '
27 mov eax,edx ; вызов подпрограммы печати значения
28 call iprintLF ; из 'edx' (остаток) в виде символов
29 call quit ; вызов подпрограммы завершения
```

Рис. 4.16: Текст программы

Создаю исполняемый файл и проверяю его работу(рис. [4.17]).



```
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ./lab6-3
Результат: 5
Остаток от деления: 1
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$
```

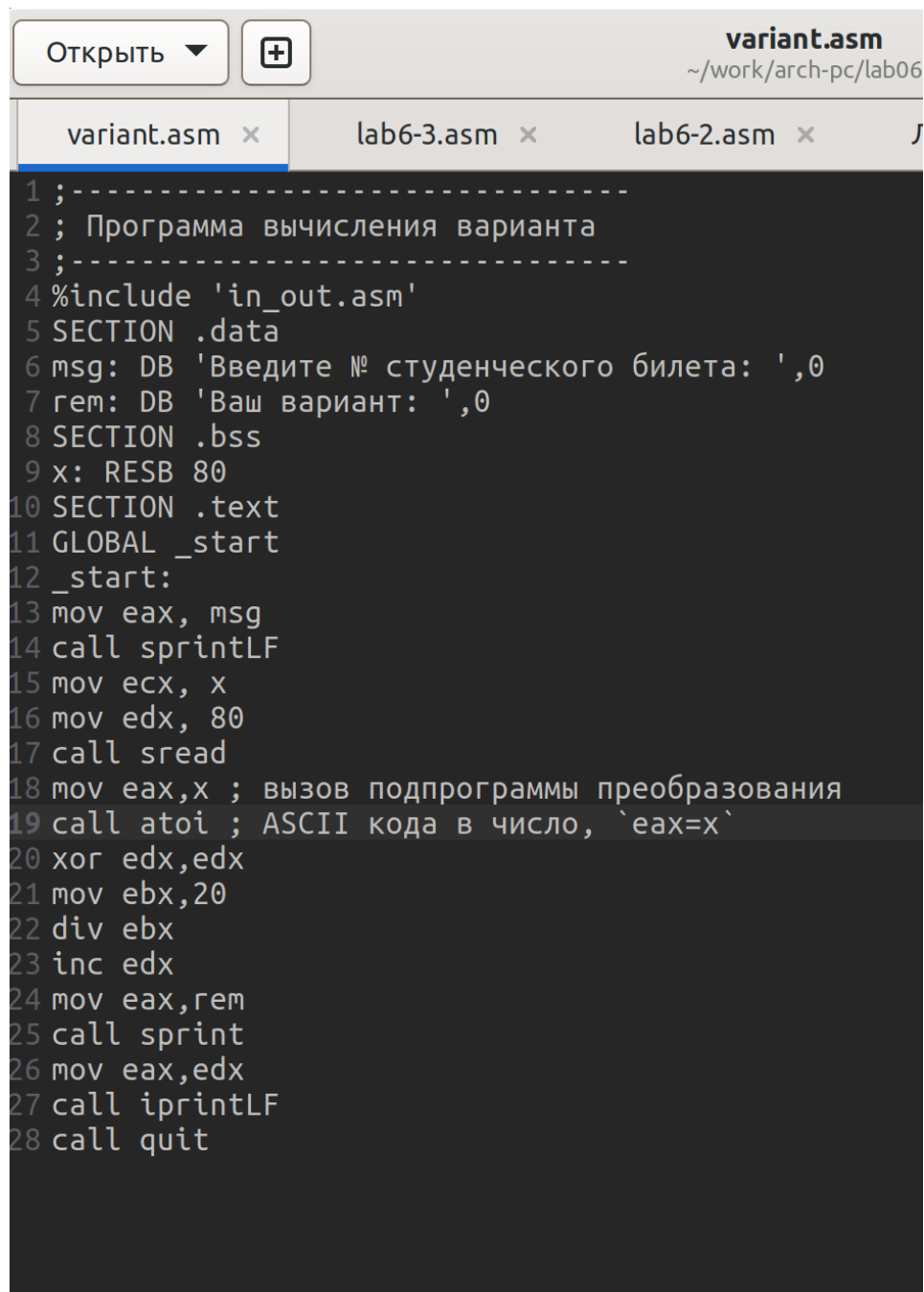
Рис. 4.17: Создание исполняемого файла и его запуск

7. В качестве другого примера рассмотрим программу вычисления варианта

задания по номеру студенческого билета, работающую по следующему алгоритму:

- вывести запрос на введение № студенческого билета
- вычислить номер варианта по формуле: $(\text{XX} \bmod 20) + 1$, где XX – номер студенческого билета (В данном случае $\text{X} \bmod \text{X}$ – это остаток от деления X на X).
- вывести на экран номер варианта.

Создаю файл variant.asm в каталоге ~/work/arch-pc/lab06 с помощью touch, заполняю его в соответствии с листингом 6.4(рис. [4.18]), создаю исполняемый файл и запускаю его(рис. [4.19]).



```
1 ;-----
2 ; Программа вычисления варианта
3 ;-----
4 %include 'in_out.asm'
5 SECTION .data
6 msg: DB 'Введите № студенческого билета: ',0
7 rem: DB 'Ваш вариант: ',0
8 SECTION .bss
9 x: RESB 80
10 SECTION .text
11 GLOBAL _start
12 _start:
13 mov eax, msg
14 call sprintLF
15 mov ecx, x
16 mov edx, 80
17 call sread
18 mov eax,x ; вызов подпрограммы преобразования
19 call atoi ; ASCII кода в число, `eax=x`
20 xor edx,edx
21 mov ebx,20
22 div ebx
23 inc edx
24 mov eax,rem
25 call sprint
26 mov eax,edx
27 call iprintLF
28 call quit
```

Рис. 4.18: Текст программы

```

zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ touch ~/work/arch-pc/lab06/variant.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ nasm -f elf variant.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ld -m elf_i386 -o variant variant.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1132236052
Ваш вариант: 13
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$

```

Рис. 4.19: Создание файла variant.asm, его исполняемого файла и его запуск

Я посчитала для проверки правильности работы программы значение выражения самостоятельно, программа отработала верно.

4.3 Ответы на вопросы:

1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’? Ответ:

```

mov eax,rem
call sprint

```

2. Для чего используются следующие инструкции? mov ecx, x mov edx, 80 call sread

Ответ: Инструкция mov ecx, x – 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. Для чего используется инструкция “call atoi”? Ответ: для вызова подпрограммы из внешнего файла, которая преобразует ascii-код символа в целое число и записывает результат в регистр eax

4. Какие строки листинга 6.4 отвечают за вычисления варианта? Ответ:

```

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. В какой регистр записывается остаток от деления при выполнении инструкции “div ebx”? Ответ: в регистр edx
6. Для чего используется инструкция “inc edx”? Ответ: для увеличения значения регистра edx на 1
7. Какие строки листинга 6.4 отвечают за вывод на экран результата вычисления? Ответ:

```
mov eax,edx
```

```
call iprintLF
```

4.4 Задание для самостоятельной работы

Напишу программу вычисления выражения $y=f(x)$.

Создаю файл variant-13.asm. Открываю созданный файл для редактирования, ввожу в него текст программы для вычисления значения выражения $(8x + 6) * 10$ (рис. [4.20]). Это выражение было под вариантом 13.

```

1 ;-----
2 ; Программа вычисления выражения (8x + 6) * 10
3 ;-----
4 %include 'in_out.asm' ; подключение внешнего файла
5 SECTION .data ; секция инициализированных данных
6 msg: DB 'Введите значение переменной x: ',0
7 res: DB 'Результат: ',0
8 SECTION .bss ; секция не инициализированных данных
9 x: RESB 80 ; Переменная, значение которой будем вводить с клавиатуры, выделенный размер -
   80 байт
10 SECTION .text ; Код программы
11 GLOBAL _start ; Начало программы
12 _start: ; Точка входа в программу
13 ; ---- Вычисление выражения
14 mov eax, msg ; запись адреса выводимого сообщения в eax
15 call sprint ; вызов подпрограммы печати сообщения
16 mov ecx, x ; запись адреса переменной в ecx
17 mov edx, 80 ; запись длины вводимого значения в edx
18 call sread ; вызов подпрограммы ввода сообщения
19 mov eax,x ; вызов подпрограммы преобразования
20 call atoi ; ASCII кода в число, 'eax=x'
21
22 mov ebx,8 ; EBX=8
23 mul ebx ; EAX=EAX*EBX
24
25 add eax,6; eax = eax+6 = 8x + 6
26 mov ebx,10 ; запись значения 10 в регистр ebx
27 mul ebx; EAX=EAX*EBX = (8x+6)*10
28 mov edi,eax ; запись результата вычисления в 'edi'
29 ; ---- Вывод результата на экран
30 mov eax,res ; вызов подпрограммы печати
31 call sprint ; сообщения 'Результат: '
32 mov eax,edi ; вызов подпрограммы печати значения
33 call iprintLF ; из 'edi' в виде символов
34 call quit ; вызов подпрограммы завершения
35
36

```

Рис. 4.20: Текст программы

Создаю и запускаю исполняемый файл (рис. [4.21]).

```

zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ nasm -f elf variant-13.asm
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ld -m elf_i386 -o variant-13 variant-13.o
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ./variant-13
Введите значение переменной x: 1
Результат: 140
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ ./variant-13
Введите значение переменной x: 4
Результат: 380
zrdagdelen@zrdagdelen:~/work/arch-pc/lab06$ 

```

Рис. 4.21: Создание исполняемого файла и его запуск

При вводе значения 1, вывод - 140. При вводе значения 4, вывод - 380
Программа отработала верно.

Программа для вычисления значения выражения $(8x + 6) * 10$.

; -----


```

; Программа вычисления выражения  $(8x + 6) * 10$ 
;-----
#include 'in_out.asm' ; подключение внешнего файла
SECTION .data ; секция инициированных данных
msg: DB 'Введите значение переменной x: ',0
rem: DB 'Результат: ',0
SECTION .bss ; секция не инициированных данных
x: RESB 80 ; Переменная, значение к-рой будем вводить с клавиатуры, выделенный ра
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 sprint ; 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,8 ; EBX=8
mul ebx ; EAX=EAX*EBX

add eax,6; eax = eax+6 =  $8x + 6$ 
mov ebx,10 ; запись значения 10 в регистр ebx
mul ebx; EAX=EAX*EBX =  $(8x+6)*10$ 
mov edi,eax ; запись результата вычисления в 'edi'
; ---- Вывод результата на экран
mov eax,rem ; вызов подпрограммы печати

```

```
call sprint ; сообщения 'Результат: '  
mov eax,edi ; вызов подпрограммы печати значения  
call iprintLF ; из 'edi' в виде символов  
call quit ; вызов подпрограммы завершения
```

5 Выводы

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

6 Список литературы

Архитектура ЭВМ