

Отчет по лабораторной работе №6

Дисциплина: архитектура компьютера

Комаров Владимир Артемович

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
4.1	Символьные и численные данные в NASM	8
4.2	Выполнение арифметических операций в NASM	12
4.2.1	Ответы на вопросы по программе	15
4.3	Выполнение заданий для самостоятельной работы	16
5	Выводы	19
6	Список литературы	20

Список иллюстраций

4.1	Создание директории	8
4.2	Создание файла	8
4.3	Создание копии файла	8
4.4	Редактирование файла	9
4.5	Запуск исполняемого файла	9
4.6	Редактирование файла	9
4.7	Запуск исполняемого файла	10
4.8	Создание файла	10
4.9	Редактирование файла	10
4.10	Запуск исполняемого файла	10
4.11	Редактирование файла	11
4.12	Запуск исполняемого файла	11
4.13	Редактирование файла	11
4.14	Запуск исполняемого файла	12
4.15	Создание файла	12
4.16	Редактирование файла	13
4.17	Запуск исполняемого файла	13
4.18	Изменение программы	14
4.19	Запуск исполняемого файла	14
4.20	Создание файла	14
4.21	Редактирование файла	15
4.22	Запуск исполняемого файла	15
4.23	Создание файла	16
4.24	Написание программы	17
4.25	Запуск исполняемого файла	17
4.26	Запуск исполняемого файла	17

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

4.1 Символьные и численные данные в NASM

С помощью утилиты `mkdir` создаю директорию, в которой буду создавать файлы с программами для лабораторной работы №6 (рис. 4.1). Перехожу в созданный каталог с помощью утилиты `cd`.

```
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ mkdir lab06  
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc$ cd lab06
```

Рис. 4.1: Создание директории

С помощью утилиты `touch` создаю файл `lab06-1.asm` (рис. 4.2).

```
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ touch lab06-1.asm  
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ls  
lab06-1.asm
```

Рис. 4.2: Создание файла

Копирую в текущий каталог файл `in_out.asm` с помощью утилиты `cp`, т.к. он будет использоваться в других программах (рис. 4.3).

```
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ cp ~/Downloads/in_out  
.asm in_out.asm  
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ls  
in_out.asm lab06-1.asm
```

Рис. 4.3: Создание копии файла

Открываю созданный файл `lab06-1.asm`, вставляю в него программу вывода значения регистра `eax` (рис. 4.4).

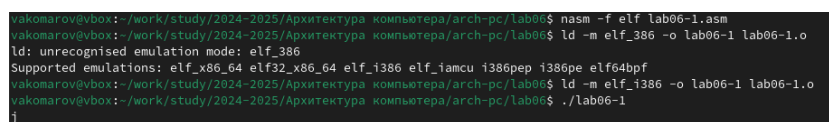


```
Open ▾  lab06-1.asm
~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06

#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 sprintf
call quit
```

Рис. 4.4: Редактирование файла

Создаю исполняемый файл программы и запускаю его (рис. 4.5). Вывод программы: символ j, потому что программа вывела символ, соответствующий по системе ASCII сумме двоичных кодов символов 4 и 6.



```
yakomarov@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ nasm -f elf lab06-1.asm
yakomarov@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ld -m elf_386 -o lab06-1 lab06-1.o
ld: unrecognized emulation mode: elf_386
Supported emulations: elf_x86_64 elf32_x86_64 elf_i386 elf_i386cpu i386pep i386pe elf64bpf
yakomarov@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ld -m elf_i386 -o lab06-1 lab06-1.o
yakomarov@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ./lab06-1
j
```

Рис. 4.5: Запуск исполняемого файла

Изменяю в тексте программы символы “6” и “4” на цифры 6 и 4 (рис. 4.6).



```
Open ▾  lab06-1.asm
~/work/study/2024-2025/Архитектура компьютера/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 sprintf
call quit
```

Рис. 4.6: Редактирование файла

Создаю новый исполняемый файл программы и запускаю его (рис. 4.7). Теперь вывелся символ с кодом 10, это символ перевода строки, этот символ не отображается при выводе на экран.

```
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ nasm -f elf lab06-1.asm
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ld -m elf_i386 -o lab06-1 lab06-1.o
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ./lab06-1
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$
```

Рис. 4.7: Запуск исполняемого файла

Создаю новый файл lab06-2.asm с помощью утилиты touch (рис. 4.8).

```
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ touch lab06-2.asm
```

Рис. 4.8: Создание файла

Ввожу в файл текст другой программы для вывода значения регистра eax (рис. 4.9).

```
Open ▾ + lab06-2.asm
~/work/study/2024-2025/Архитектура компьютера/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
```

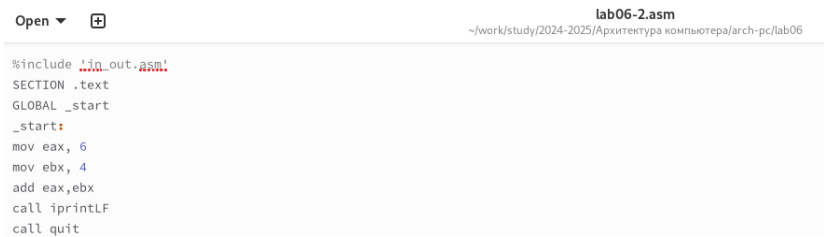
Рис. 4.9: Редактирование файла

Создаю и запускаю исполняемый файл lab06-2 (рис. 4.10). 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”.

```
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ nasm -f elf lab06-2.asm
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ld -m elf_i386 -o lab06-2 lab06-2.o
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ./lab06-2
106
```

Рис. 4.10: Запуск исполняемого файла

Заменяю в тексте программы в файле lab06-2.asm символы “6” и “4” на числа 6 и 4 (рис. 4.11).




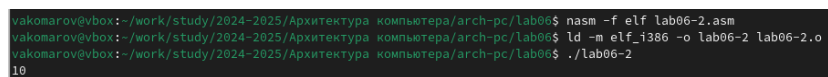
```
Open ▾  lab06-2.asm  
~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06  
%include "io_out.asm"  
SECTION .text  
GLOBAL _start  
_start:  
mov eax, 6  
mov ebx, 4  
add eax, ebx  
call iprintLF  
call quit
```

Рис. 4.11: Редактирование файла

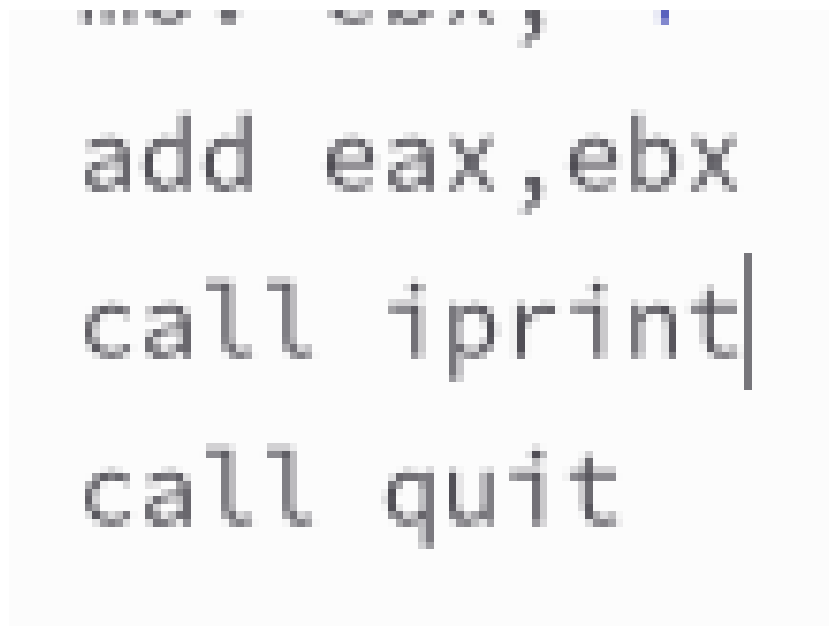
Создаю и запускаю новый исполняемый файл (рис. 4.12).. Теперь программа складывает не соответствующие символам коды в системе ASCII, а сами числа, поэтому вывод 10.



```
yakomarov@box:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ nasm -f elf lab06-2.asm  
yakomarov@box:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ld -m elf_i386 -o lab06-2 lab06-2.o  
yakomarov@box:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ./lab06-2  
10
```

Рис. 4.12: Запуск исполняемого файла

Заменяю в тексте программы функцию iprintLF на iprint (рис. 4.13).



```
add eax, ebx  
call iprint  
call quit
```

Рис. 4.13: Редактирование файла

Создаю и запускаю новый исполняемый файл (рис. 4.14). Вывод не изменился, потому что символ переноса строки не отображался, когда программа исполня-

лась с функцией `iprintLF`, а `iprint` не добавляет к выводу символ переноса строки, в отличие от `iprintLF`.

```
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ nasm -f elf lab06-2.asm
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ld -m elf_i386 -o lab06-2 lab06-2.o
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ./lab06-2
10vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$
```

Рис. 4.14: Запуск исполняемого файла

4.2 Выполнение арифметических операций в NASM

Создаю файл `lab06-3.asm` с помощью утилиты `touch` (рис. 4.15).

```
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ touch lab06-3.asm
```

Рис. 4.15: Создание файла

Ввожу в созданный файл текст программы для вычисления значения выражения $f(x) = (5 * 2 + 3)/3$ (рис. 4.16).

```

#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. 4.16: Редактирование файла

Создаю исполняемый файл и запускаю его (рис. 4.17).

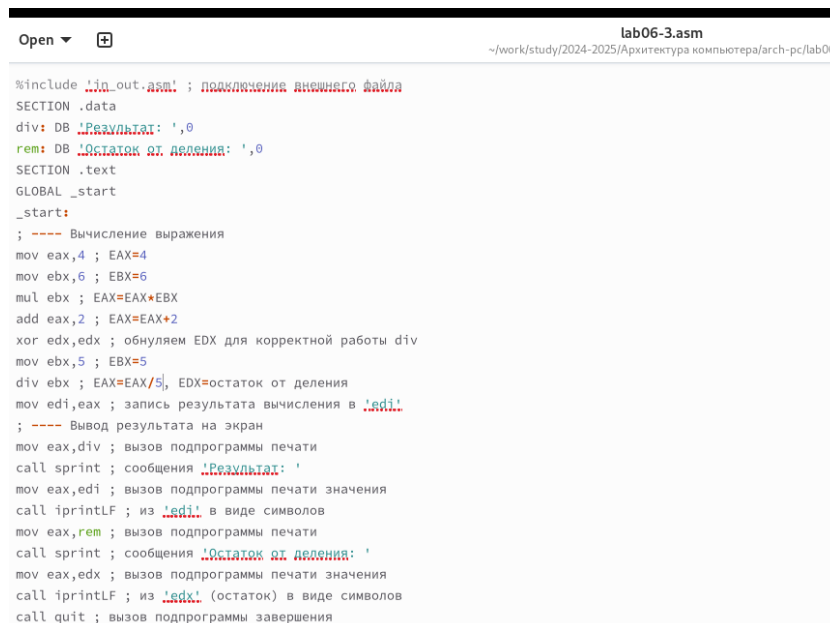
```


vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ nasm -f elf lab06-3.asm
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ld -m elf_i386 -o lab06-3 lab06-3.o
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ./lab06-3
Результат: 4
Остаток от деления: 1

```

Рис. 4.17: Запуск исполняемого файла

Изменяю программу так, чтобы она вычисляла значение выражения $f(x) = (4 * 6 + 2)/5$ (рис. 4.18).

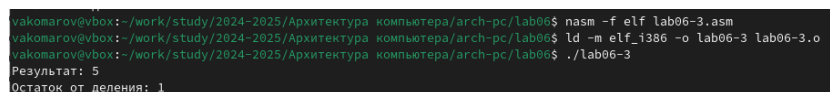


```
Open ▾  lab06-3.asm
~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06

%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. 4.18: Изменение программы

Создаю и запускаю новый исполняемый файл (рис. 4.19). Я посчитал для проверки правильности работы программы значение выражения самостоятельно, программа отработала верно.



```
vakomarov@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ nasm -f elf lab06-3.asm
vakomarov@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ld -m elf_i386 -o lab06-3 lab06-3.o
vakomarov@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ./lab06-3
Результат: 5
Остаток от деления: 1
```

Рис. 4.19: Запуск исполняемого файла

Создаю файл variant.asm с помощью утилиты touch (рис. 4.20).



```
vakomarov@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ touch variant.asm
```

Рис. 4.20: Создание файла

Ввожу в файл текст программы для вычисления варианта задания по номеру студенческого билета (рис. 4.21).

```

Open ▾ (+)
variant.asm
~/work/study/2024-2025/Архитектура компьютера/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. 4.21: Редактирование файла

Создаю и запускаю исполняемый файл (рис. 4.22). Ввожу номер своего студ. билета с клавиатуры, программа вывела, что мой вариант - 18.

```

vakomarov@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ nasm -f elf variant.asm
vakomarov@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ld -m elf_i386 -o variant variant.o
vakomarov@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1132246757
Ваш вариант: 18

```

Рис. 4.22: Запуск исполняемого файла

4.2.1 Ответы на вопросы по программе

1. За вывод сообщения “Ваш вариант” отвечают строки кода:

```

mov eax, rem
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
```

4.3 Выполнение заданий для самостоятельной работы

Создаю файл `lab06-4.asm` с помощью утилиты `touch` (рис. 4.23).



```
vakonarov@vbox: ~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ touch lab06-4.asm
```

Рис. 4.23: Создание файла

Открываю созданный файл для редактирования, ввожу в него текст программы для вычисления значения выражения $(x + 10) * 3 - 20$ (рис. 4.24). Это выражение было под вариантом 18.


```

%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg: DB 'Введите значение переменной x: ',0
rem: DB 'Результат: ',0
SECTION .bss
x: RESB 80 ; Переменная, значение к-рой будем вводить с клавиатуры
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x`
add eax, 10; eax = eax + 10 = x + 10
mov ebx, 3
mul ebx; EAX=EAX*EBX = (x+10)*3
add eax, -20; eax = eax - 20 = (x+10)*3 - 20
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

```

Рис. 4.24: Написание программы

Создаю и запускаю исполняемый файл (рис. 4.25). При вводе значения 1, вывод - 13.

```

vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ nasm -f elf lab06-4.asm
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ld -m elf_i386 -o lab06-4 lab06-4.o
vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ./lab06-4
Введите значение переменной x: 1
Результат: 13

```

Рис. 4.25: Запуск исполняемого файла

Провожу еще один запуск исполняемого файла для проверки работы программы с другим значением на входе (рис. 4.26). Программа отработала верно.

```

vakomarov@vbox:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab06$ ./lab06-4
Введите значение переменной x: 5
Результат: 25

```

Рис. 4.26: Запуск исполняемого файла

Листинг 4.1. Программа для вычисления значения выражения $(x + 10) * 3 - 20$.

```

%include 'in_out.asm' ; подключение внешнего файла
SECTION .data
msg: DB 'Введите значение переменной x: ',0
rem: DB 'Результат: ',0
SECTION .bss
x: RESB 80 ; Переменная, значение к-рой будем вводить с клавиатуры
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax,x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x`
add eax,10; eax = eax+10 = x + 10
mov ebx,3
mul ebx; EAX=EAX*EBX = (x+10)*3
add eax,-20; eax = eax-20 = (x+10)*3-20
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 освоил арифметические инструкции языка ассемблера NASM.

6 Список литературы

2. Таблица ASCII