

Отчёт по лабораторной работе №6

Дисциплина: архитектура компьютера

Бызова Мария Олеговна

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	8
3.1	Символьные и численные данные в NASM	8
3.2	Выполнение арифметических операций в NASM	13
3.2.1	Ответы на вопросы по программе	17
4	Выполнение заданий для самостоятельной работы	18
5	Выводы	21

Список иллюстраций

3.1	Создание необходимой директории и файла	8
3.2	Копирование файла	9
3.3	Редактирование файла	9
3.4	Создание исполняемого файла	9
3.5	Запуск исполняемого файла	9
3.6	Редактирование файла	10
3.7	Создание исполняемого файла	10
3.8	Запуск исполняемого файла	10
3.9	Создание файла	11
3.10	Редактирование файла	11
3.11	Создание исполняемого файла	11
3.12	Запуск исполняемого файла	11
3.13	Редактирование файла	12
3.14	Создание исполняемого файла	12
3.15	Запуск исполняемого файла	12
3.16	Редактирование файла	13
3.17	Создание исполняемого файла	13
3.18	Запуск исполняемого файла	13
3.19	Создание файла	14
3.20	Редактирование файла	14
3.21	Создание исполняемого файла	14
3.22	Запуск исполняемого файла	14
3.23	Изменение программы	15
3.24	Запуск исполняемого файла	15
3.25	Создание файла	15
3.26	Редактирование файла	16
3.27	Создание исполняемого файла	16
3.28	Запуск исполняемого файла	16
4.1	Создание файла	18
4.2	Написание программы	18
4.3	Создание исполняемого файла	19
4.4	Запуск исполняемого файла	19
4.5	Запуск исполняемого файла	19

Список таблиц

1 Цель работы

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

2 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.

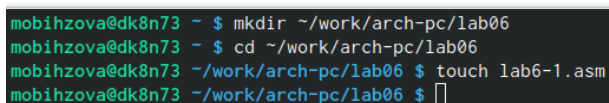
Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации производится согласно кодовой таблице символов ASCII. ASCII – сокращение от American Standard Code for Information Interchange (Американский стандартный код для обмена информацией). Согласно 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

3 Выполнение лабораторной работы

3.1 Символьные и численные данные в NASM

1. С помощью утилиты `mkdir` создаем директорию, в которой будем создавать файлы с программами для лабораторной работы №6. Переходим в созданный каталог с помощью утилиты `cd`. С помощью утилиты `touch` создаем файл `lab6-1.asm` (рис. [3.1]).



```
mobihzova@dk8n73 ~ $ mkdir ~/work/arch-pc/lab06
mobihzova@dk8n73 ~ $ cd ~/work/arch-pc/lab06
mobihzova@dk8n73 ~/work/arch-pc/lab06 $ touch lab6-1.asm
mobihzova@dk8n73 ~/work/arch-pc/lab06 $
```

Рис. 3.1: Создание необходимой директории и файла

2. При помощи Midnight Commander открываем созданный файл `lab6-1.asm`, вставляем в него программу вывода значения регистра `eax`, предварительно скопировав в текущий каталог файл `in_out.asm`, поскольку он необходим для правильной работы программы (рис. [3.2], [3.3]).

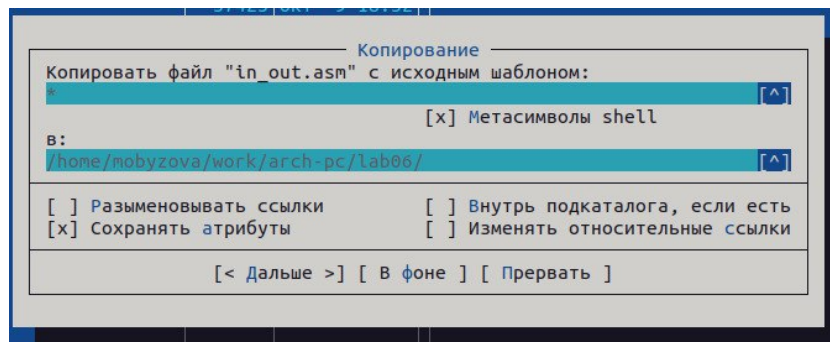


Рис. 3.2: Копирование файла

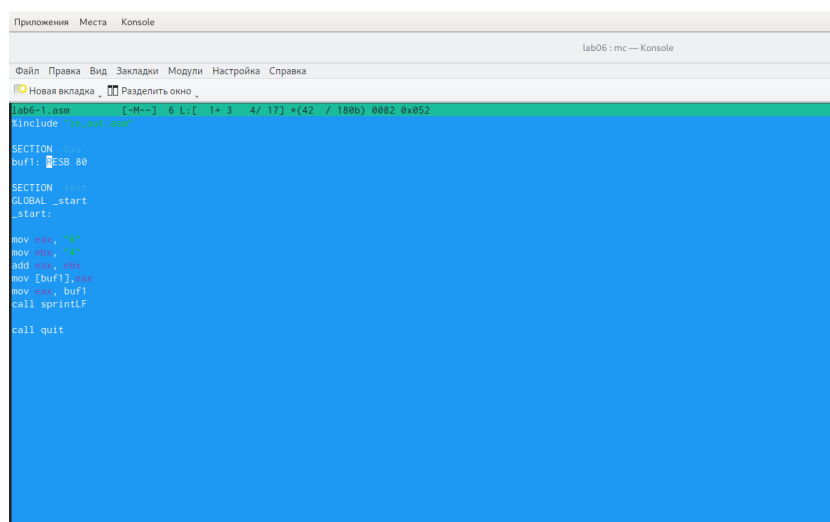


Рис. 3.3: Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [3.4], [3.5]).

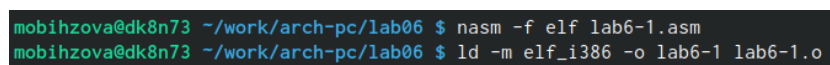


Рис. 3.4: Создание исполняемого файла

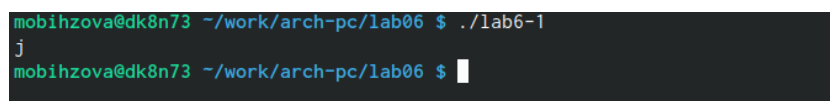


Рис. 3.5: Запуск исполняемого файла

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

3. Далее изменим текст программы и вместо символов, запишем в регистры числа. Исправим текст программы (рис. [3.6]).

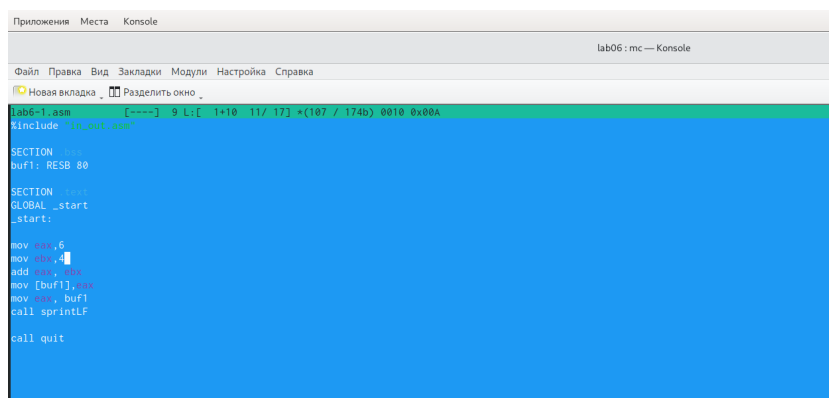


Рис. 3.6: Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [3.7], [3.8]).

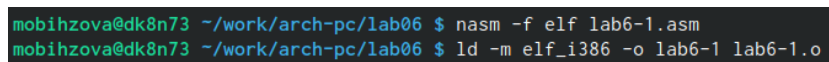


Рис. 3.7: Создание исполняемого файла

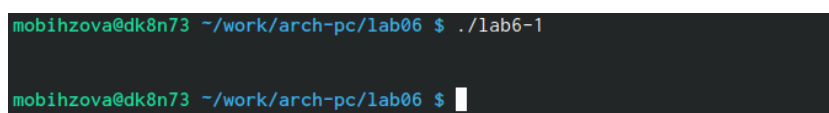


Рис. 3.8: Запуск исполняемого файла

Теперь выводится символ с кодом 10 - это символ перевода строки. Этот символ не отображается при выводе на экран.

4. Создадим новый файл lab6-2.asm с помощью утилиты touch (рис. [3.9]).

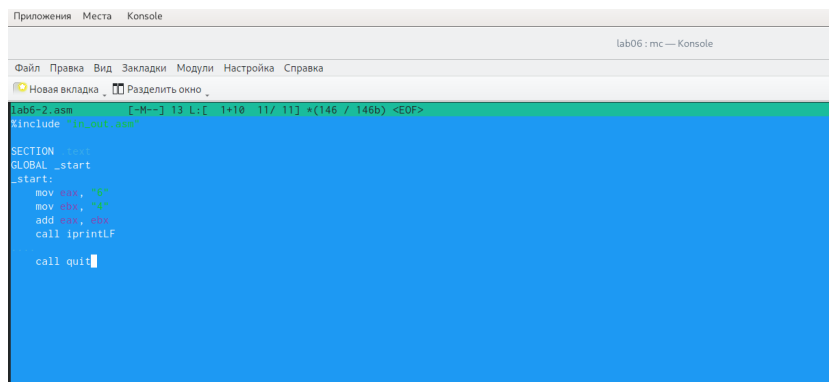
```

mobihzova@dk8n73 ~/work/arch-pc/lab06 $ touch lab6-2.asm
mobihzova@dk8n73 ~/work/arch-pc/lab06 $

```

Рис. 3.9: Создание файла

Вводим в файл текст другой программы для вывода значения регистра eax (рис. [3.10]).



```

lab6-2.asm [M--] 13 L: 1x10 11/ 113 *(146 / 146b) <EOF>
#include "fputc.asm"

SECTION .text
GLOBAL _start
_start:
    mov eax, 6
    mov ebx, 4
    add ebx, ebx
    call iprintLF
    call quit

```

Рис. 3.10: Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [3.14], [3.15]).

```

mobihzova@dk8n73 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
mobihzova@dk8n73 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o

```

Рис. 3.11: Создание исполняемого файла

```

mobihzova@dk8n73 ~/work/arch-pc/lab06 $ ./lab6-2
106
mobihzova@dk8n73 ~/work/arch-pc/lab06 $

```

Рис. 3.12: Запуск исполняемого файла

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). Однако, в отличие от программы из листинга 6.1, функция iprintLF позволяет вывести число, а не символ, кодом которого является это число.

5. Заменяем в тексте программы в файле lab6-2.asm символы “6” и “4” на числа 6 и 4 (рис. [3.13]).

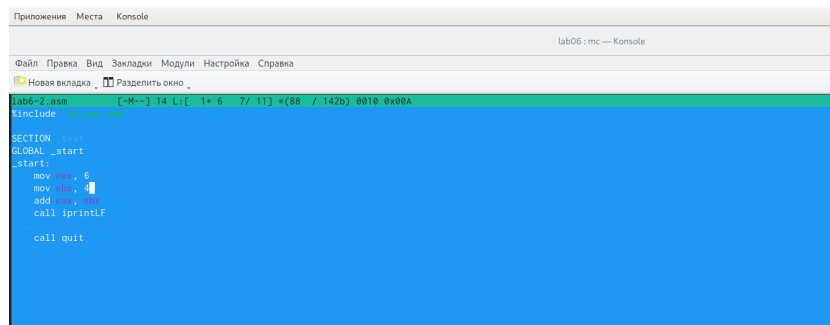


Рис. 3.13: Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [??], [??]).

```
mobihzova@dk8n73 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
mobihzova@dk8n73 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
```

Рис. 3.14: Создание исполняемого файла

```
mobihzova@dk8n73 ~/work/arch-pc/lab06 $ ./lab6-2
10
mobihzova@dk8n73 ~/work/arch-pc/lab06 $
```

Рис. 3.15: Запуск исполняемого файла

Теперь программа складывает не соответствующие символам коды в системе ASCII, а сами числа, поэтому вывод 10.

Заменяем в тексте программы функцию iprintLF на iprint (рис. [3.16]).

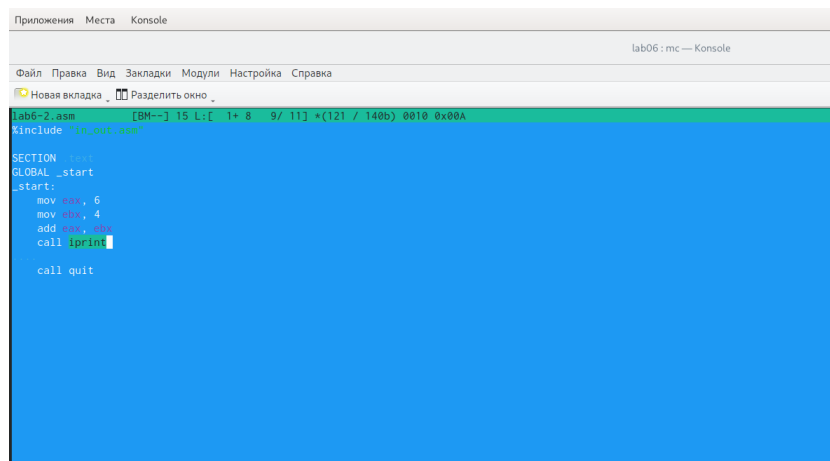


Рис. 3.16: Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [3.17], [3.18]).

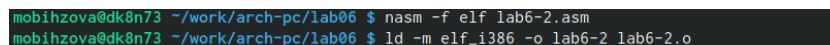


Рис. 3.17: Создание исполняемого файла

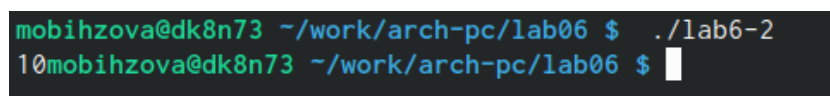


Рис. 3.18: Запуск исполняемого файла

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`.

3.2 Выполнение арифметических операций в NASM

6. Создаем файл `lab6-3.asm` с помощью утилиты `touch` (рис. [3.19]).

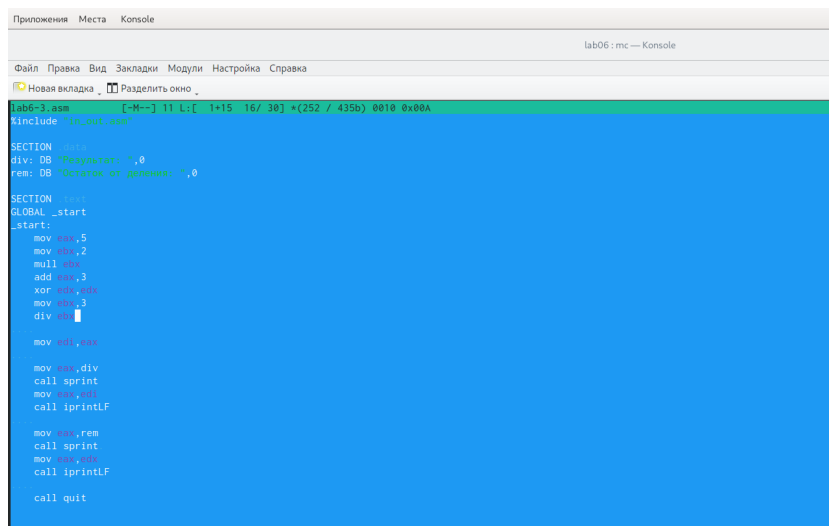
```

mobihzova@dk8n73 ~/work/arch-pc/lab06 $ ./lab6-3
10mobihzova@dk8n73 ~/work/arch-pc/lab06 $ touch lab6-3.asm
mobihzova@dk8n73 ~/work/arch-pc/lab06 $

```

Рис. 3.19: Создание файла

Вводим в созданный файл текст программы для вычисления значения выражения $f(x) = (5 * 2 + 3)/3$ (рис. [3.20]).



```

lab6-3.asm 11 L: [ 1*15 16/ 30] *(252 / 435b) 0010 0x00A
#include "in_out.asm"

SECTION .text
div: DB "Результат: ",0
rem: DB "Остаток от деления: ",0

SECTION .start
GLOBAL _start
_start:
    mov eax,5
    mov ebx,2
    mul ebx
    add eax,3
    xor ebx,ebx
    mov ebx,3
    div ebx

    mov edi,div
    mov eax,div
    call sprintf
    mov ebx,edi
    call iprintLF

    mov eax,rem
    call sprintf
    mov ebx,edi
    call iprintLF

    call quit

```

Рис. 3.20: Редактирование файла

Далее создаем исполняемый файл и запускаем его (рис. [3.21], [3.22]).

```

mobihzova@dk8n73 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
mobihzova@dk8n73 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o

```

Рис. 3.21: Создание исполняемого файла

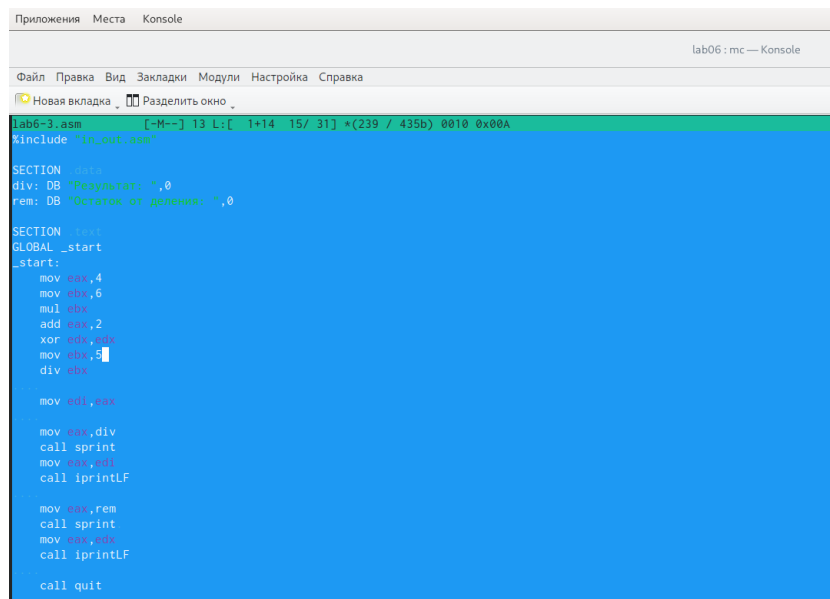
```

mobihzova@dk8n73 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 4
Остаток от деления: 1
mobihzova@dk8n73 ~/work/arch-pc/lab06 $

```

Рис. 3.22: Запуск исполняемого файла

Изменяем программу так, чтобы она вычисляла значение выражения $f(x) = (4 * 6 + 2)/5$ (рис. [3.23]).



```
lab6-3.asm [-M--] 13 L: [ 1+14 15/ 31] *(239 / 435b) 0010 0x00A
#include "in_out.asm"

SECTION .text
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

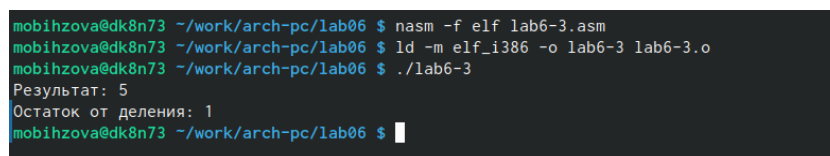
    mov edi,eax
    mov eax,div
    call sprint
    mov eax,rem
    call iprintLF

    mov eax,rem
    call sprint
    mov eax,edx
    call iprintLF

    call quit
```

Рис. 3.23: Изменение программы

Далее создаем исполняемый файл и запускаем его (рис. [3.24]).



```
mobihzova@dk8n73 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
mobihzova@dk8n73 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
mobihzova@dk8n73 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 5
Остаток от деления: 1
mobihzova@dk8n73 ~/work/arch-pc/lab06 $
```

Рис. 3.24: Запуск исполняемого файла

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

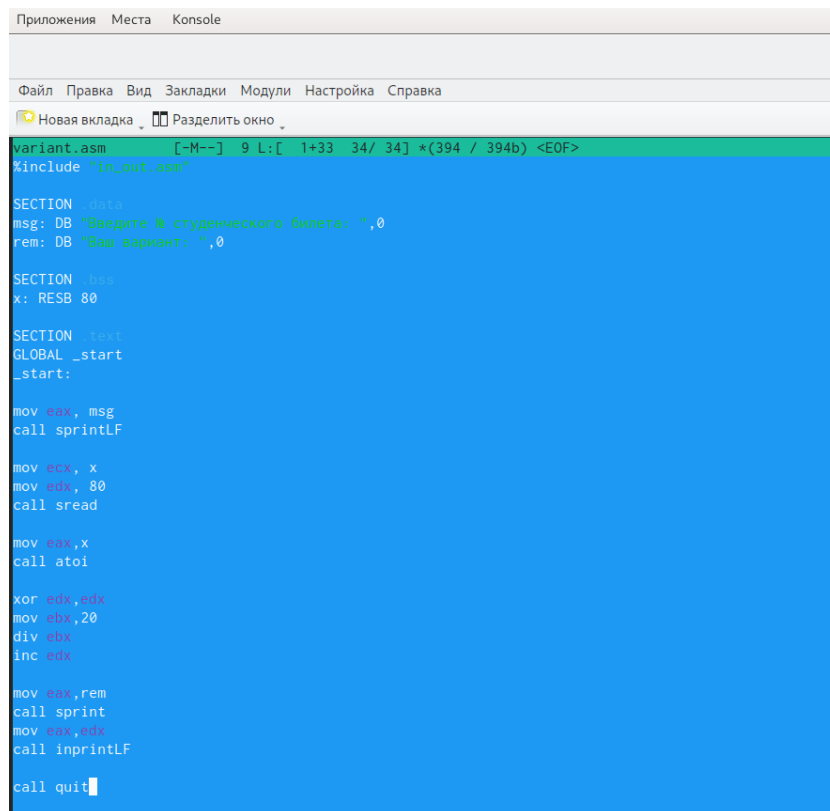
7. Создадим файл variant.asm с помощью утилиты touch (рис. [3.25]).



```
mobyzova@mobyzova-VirtualBox:~/work/arch-pc/lab06$ touch variant.asm
mobyzova@mobyzova-VirtualBox:~/work/arch-pc/lab06$
```

Рис. 3.25: Создание файла

Вводим в файл текст программы для вычисления варианта задания по номеру студенческого билета (рис. [3.26]).



```
variant.asm [-M--] 9 L:[ 1+33 34/ 34] *(394 / 394b) <EOF>
#include "in_out.asm"

SECTION .msg
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

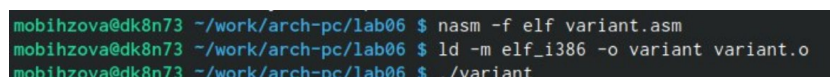
xor edx, edx
mov ebx, 20
div ebx
inc edx

mov eax, rem
call sprintf
mov ebx, edx
call inprintLF

call quit
```

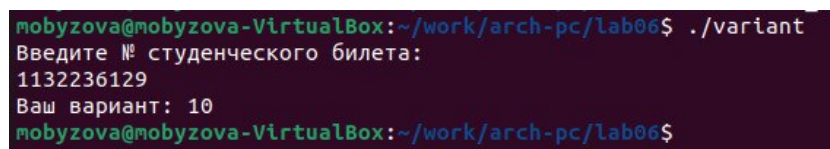
Рис. 3.26: Редактирование файла

Создаем и запускаем исполняемый файл (рис. [3.27]). Вводим номер своего студ. билета с клавиатуры, программа вывела, что мой вариант - 10.(рис. [3.28]).



```
mobihzova@dk8n73 ~/work/arch-pc/lab06 $ nasm -f elf variant.asm
mobihzova@dk8n73 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o variant variant.o
mobihzova@dk8n73 ~/work/arch-pc/lab06 $ ./variant
```

Рис. 3.27: Создание исполняемого файла



```
mobyzova@mobyzova-VirtualBox:~/work/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1132236129
Ваш вариант: 10
mobyzova@mobyzova-VirtualBox:~/work/arch-pc/lab06$
```

Рис. 3.28: Запуск исполняемого файла

3.2.1 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. Инструкция mov edx, 80 используется для записи в регистр edx длины вводимой строки. Инструкция call sread используется для вызова подпрограммы из внешнего файла, обеспечивающей ввод сообщения с клавиатуры.
3. Инструкция 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:

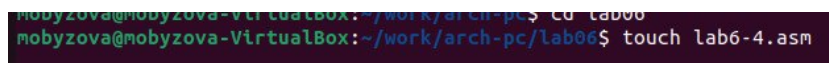
```
xor edx,edx ; обнуление edx для корректной работы div  
mov ebx,20 ; ebx = 20  
div ebx ; eax = eax/20, edx - остаток от деления  
inc edx ; edx = edx + 1
```

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 Выполнение заданий для самостоятельной работы

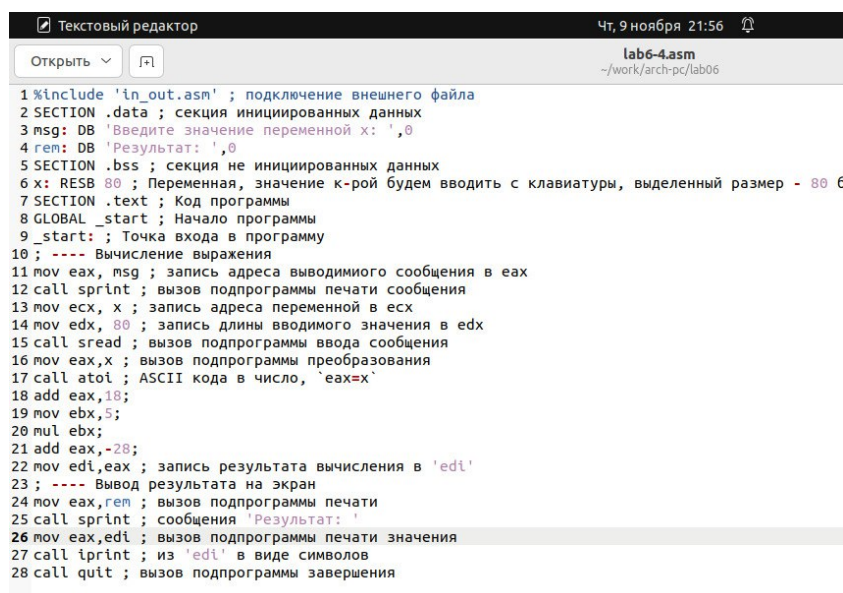
1. Создаем файл lab6-4.asm с помощью утилиты touch (рис. [4.1]).



```
mobyzoa@mobyzoa-VirtualBox: ~/.work/arch-pc$ cd lab06
mobyzoa@mobyzoa-VirtualBox: ~/.work/arch-pc/lab06$ touch lab6-4.asm
```

Рис. 4.1: Создание файла

Открываем созданный файл для редактирования, вводим в него текст программы для вычисления значения выражения $5 * (x + 18) - 28$ (рис. [4.2]). Это выражение было под вариантом 10.



```
Текстовый редактор Чт, 9 ноября 21:56
lab6-4.asm
~/.work/arch-pc/lab06

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

Рис. 4.2: Написание программы

Далее создаем исполняемый файл и запускаем его (рис. [4.3], [4.4]).

```
mobyzova@mobyzova-VirtualBox:~/work/arch-pc/lab06$ nasm -f elf lab6-4.asm
mobyzova@mobyzova-VirtualBox:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-4 lab6-4.o
```

Рис. 4.3: Создание исполняемого файла

```
mobyzova@mobyzova-VirtualBox:~/work/arch-pc/lab06$ ./lab6-4
Введите значение переменной x: 2
Результат: 72mobyzova@mobyzova-VirtualBox:~/work/arch-pc/lab06$
```

Рис. 4.4: Запуск исполняемого файла

При вводе значения 2, вывод - 72. Проводим еще один запуск исполняемого файла для проверки работы программы с другим значением на входе (рис. [4.5]). Программа отработала верно.

```
Результат: 72mobyzova@mobyzova-VirtualBox:~/work/arch-pc/lab06$ ./lab6-4
Введите значение переменной x: 3
Результат: 77mobyzova@mobyzova-VirtualBox:~/work/arch-pc/lab06$
```

Рис. 4.5: Запуск исполняемого файла

Листинг. Программа для вычисления значения выражения $5 * (x + 18) - 28$.

```
%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`
add eax,18;
mov ebx,5;
mul ebx;
add eax, -28;
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 iprint ; из 'edi' в виде символов
call quit ; вызов подпрограммы завершения

```

5 Выводы

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