

Отчет по лабораторной работе №7

Дисциплина: архитектура компьютера

Краснопер Данила Олегович

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Выполнение арифметических операций в NASM	15
4.1.1	Ответы на вопросы по программе	19
4.2	Выполнение заданий для самостоятельной работы	20
5	Выводы	22

Список иллюстраций

4.1	Создание директории	9
4.2	Создание файла	9
4.3	Создание копии файла	9
4.4	Редактирование файла	10
4.5	Запуск исполняемого файла	10
4.6	Редактирование файла	11
4.7	Запуск исполняемого файла	12
4.8	Создание файла	12
4.9	Редактирование файла	12
4.10	Запуск исполняемого файла	13
4.11	Редактирование файла	13
4.12	Запуск исполняемого файла	14
4.13	Редактирование файла	14
4.14	Запуск исполняемого файла	15
4.15	Создание файла	15
4.16	Редактирование файла	16
4.17	Запуск исполняемого файла	16
4.18	Изменение программы	17
4.19	Запуск исполняемого файла	17
4.20	Создание файла	17
4.21	Редактирование файла	18
4.22	Запуск исполняемого файла	18
4.23	Создание файла	20
4.24	Написание программы	20
4.25	Запуск исполняемого файла	20
4.26	Запуск исполняемого файла	21

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 используют 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

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`.

```
dokrasnoper@dk5n56 ~ $ mkdir ~/work/arch-pc
dokrasnoper@dk5n56 ~ $ mkdir ~/work/arch-pc/lab06
dokrasnoper@dk5n56 ~ $ cd ~/work/arch-pc/lab06
```

Рис. 4.1: Создание директории

С помощью утилиты `touch` создаю файл `lab6-1.asm` (рис. 4.2).

```
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ touch lab6-1.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).

```
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ cp ~/Загрузки/in_out.asm in_out.asm
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ls
in_out.asm  lab6-1.asm
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $
```

Рис. 4.3: Создание копии файла

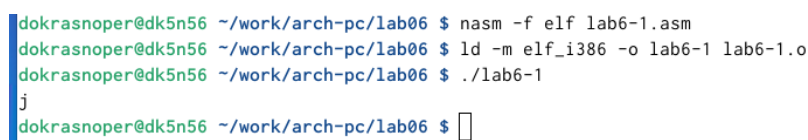
Открываю созданный файл `lab6-1.asm`, вставляю в него программу вывода значения регистра `eax` (рис. 4.4).



```
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 printf
13 call _exit
```

Рис. 4.4: Редактирование файла

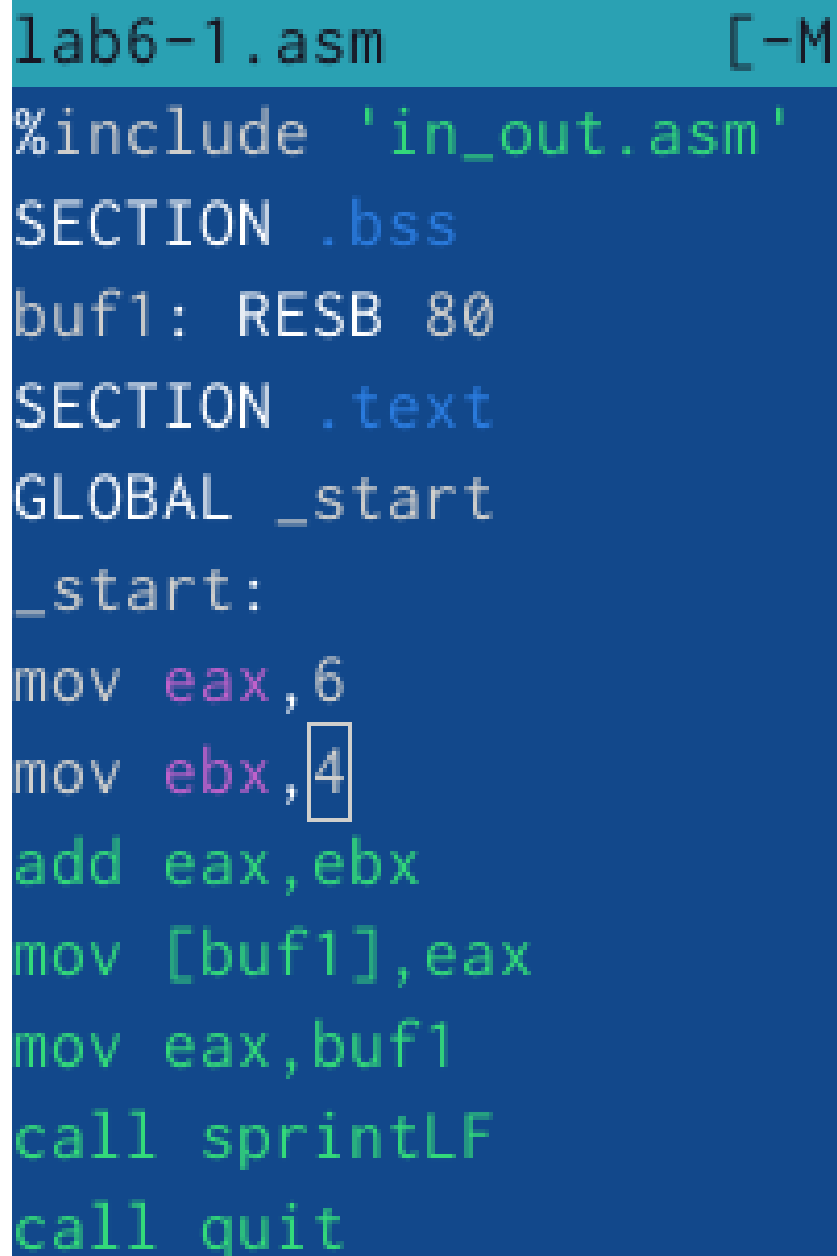
Создаю исполняемый файл программы и запускаю его (рис. 4.5). Вывод программы: символ j, потому что программа вывела символ, соответствующий по системе ASCII сумме двоичных кодов символов 4 и 6.



```
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ./lab6-1
j
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $
```

Рис. 4.5: Запуск исполняемого файла

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



```
lab6-1.asm [-M
#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.

```
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ./lab6-1

dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $
```

Рис. 4.7: Запуск исполняемого файла

Создаю новый файл lab6-2.asm с помощью утилиты touch (рис. 4.8).

```
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/lab6-2.asm
```

Рис. 4.8: Создание файла

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

```
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
```

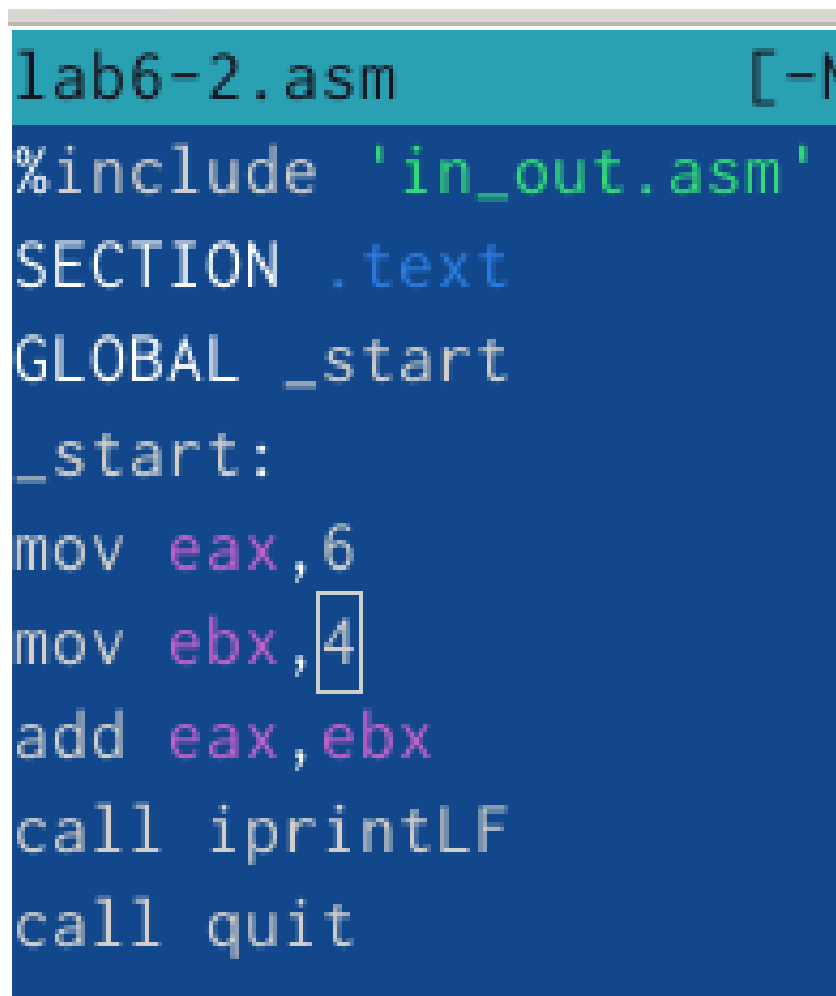
Рис. 4.9: Редактирование файла

Создаю и запускаю исполняемый файл lab6-2 (рис. 4.10). Теперь выводится число 106, потому что программа позволяет вывести именно число, а не символ, хотя все еще происходит именно сложение кодов символов “6” и “4”.

```
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ d -m elf_i386 -o lab6-2 lab6-2.o
bash: d: команда не найдена
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ./lab6-2
106
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $
```

Рис. 4.10: Запуск исполняемого файла

Заменяю в тексте программы в файле lab6-2.asm символы “6” и “4” на числа 6 и 4 (рис. 4.11).



```
lab6-2.asm [-M
#include 'in_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

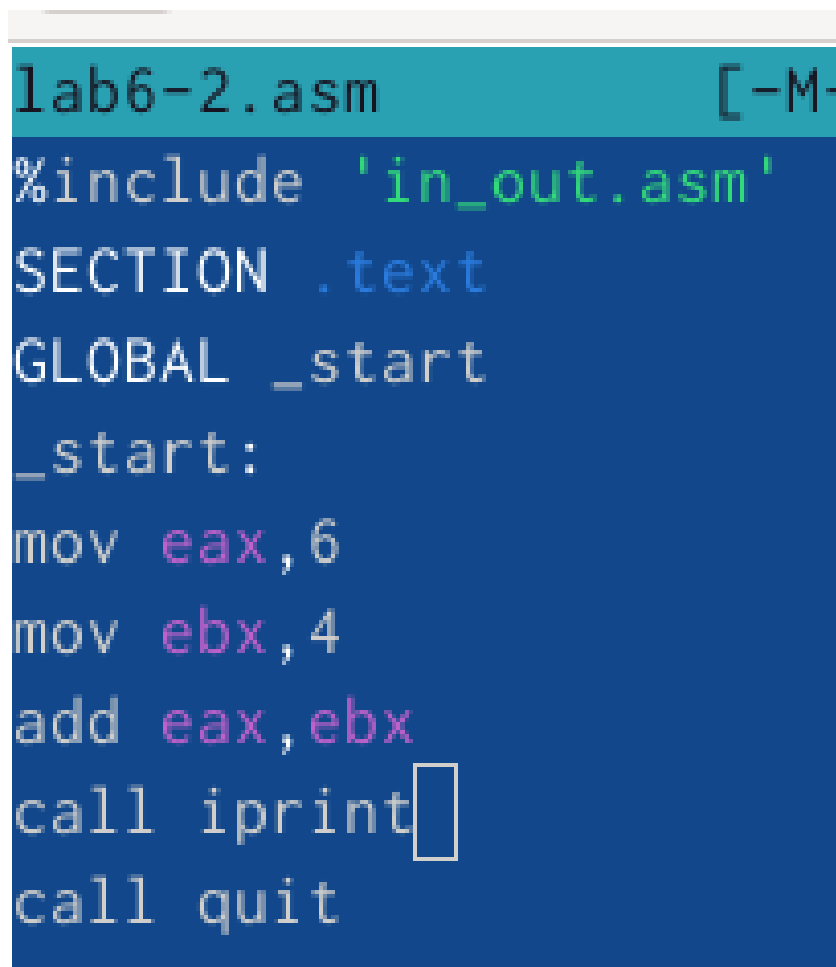
Создаю и запускаю новый исполняемый файл (рис. 4.12).. Теперь программа складывает не соответствующие символам коды в системе ASCII, а сами числа,

поэтому вывод 10.

```
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ./lab6-2
10
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $
```

Рис. 4.12: Запуск исполняемого файла

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



```
lab6-2.asm [-M-
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
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 исполнялась с функцией `iprintLF`, а `iprint` не добавляет к выводу символ переноса строки, в отличие от `iprintLF`.

```
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ./lab6-2
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ./lab6-2
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ./lab6-2
10dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ █
```

Рис. 4.14: Запуск исполняемого файла

4.1 Выполнение арифметических операций в NASM

Создаю файл lab6-3.asm с помощью утилиты touch (рис. 4.15).

```
noper@dk5n56 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/lab6-3.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).

```
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ d -m elf_i386 -o lab6-2 lab6-2.o
l:
Юр Открыть *lab6-3.asm
Юр ~/work/arch-pc/lab06
Юр 1 ;-----
Юр 2 ; Программа вычисления выражения
Юр 3 ;-----
Юр 4 %include 'in_out.asm' ; подключение внешнего файла
Юр 5 SECTION .data
Юр 6 div: DB 'Результат: ',0
Юр 7 rem: DB 'Остаток от деления: ',0
Юр 8 SECTION .text
Юр 9 GLOBAL _start
Юр 10 _start:
Юр 11 ; ---- Вычисление выражения
Юр 12 mov eax,5 ; EAX=5
Юр 13 mov ebx,2 ; EBX=2
Юр 14 mul ebx ; EAX=EAX*EBX
Юр 15 add eax,3 ; EAX=EAX+3
Юр 16 xor edx,edx ; обнуляем EDX для корректной работы div
Юр 17 mov ebx,3 ; EBX=3
Юр 18 div ebx ; EAX=EAX/3, EDX=остаток от деления
Юр 19 mov edi,eax ; запись результата вычисления в 'edi'
Юр 20 ; ---- Вывод результата на экран
Юр 21 mov eax,div ; вызов подпрограммы печати
Юр 22 call sprint ; сообщения 'Результат: '
Юр 23 mov eax,edi ; вызов подпрограммы печати значения
Юр 24 call iprintLF ; из 'edi' в виде символов
Юр 25 mov eax,rem ; вызов подпрограммы печати
Юр 26 call sprint ; сообщения 'Остаток от деления: '
Юр 27 mov eax,edx ; вызов подпрограммы печати значения
Юр 28 call iprintLF ; из 'edx' (остаток) в виде символов
Юр 29 call quit ; вызов подпрограммы завершения
```

Рис. 4.16: Редактирование файла

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

```
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 4
Остаток от деления: 1
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $
```

Рис. 4.17: Запуск исполняемого файла

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


```

;-----
; Программа вычисления выражения
;-----
#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.

```

dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 5
Остаток от деления: 1
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ █

```

Рис. 4.19: Запуск исполняемого файла

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

```

dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/variant.asm
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ █

```

Рис. 4.20: Создание файла

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

```
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
0 SECTION .text
1 GLOBAL _start
2 _start:
3 mov eax, msg
4 call sprintf
5 mov ecx, x
6 mov edx, 80
7 call sread
8 mov eax, x ; вызов подпрограммы преобразования
9 call atoi ; ASCII кода в число, 'eax=x'
0 xor edx, edx
1 mov ebx, 20
2 div ebx
3 inc edx
4 mov eax, rem
5 call sprintf
6 mov eax, edx
7 call iprintLF
8 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 - 9.



Ваш вариант: 9

Рис. 4.22: Запуск исполняемого файла

4.1.1 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
```

4.2 Выполнение заданий для самостоятельной работы

Создаю файл lab6-4.asm с помощью утилиты touch (рис. 4.23).

```
dokrasnoper@dk5n56 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/lab6-4.asm
```

Рис. 4.23: Создание файла

Открываю созданный файл для редактирования, ввожу в него текст программы для вычисления значения выражения $10 + (31x - 5)$ (рис. 4.24). Это выражение было под вариантом 9.

```
lab6-4.asm [-----] 0 L: [ 1+ 3 4/ 32] *(216 /1679b) 0114 0x072
#include 'in_out.asm' ; подключение внешнего файла
SECTION .data ; секция иницированных данных
msg: DB 'Введите значение переменной x: ',0
rem: DB 'Результат: ',0
SECTION .bss ; секция не иницированных 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,31
mul ebx
add eax,-5
add eax,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 iprint ; 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 3, вывод - 98.

```
dokrasnoper@dk3n57 ~/work/arch-pc/lab06 $ ./lab6-4
Введите значение переменной x: 3
Результат: 98dokrasnoper@dk3n57 ~/work/arch-pc/lab06 $ ./lab6-4
```

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.

```
Результат: 98dokrasnoper@dk3n57 ~/work/arch-pc/lab06 $ ./lab6-4
Введите значение переменной x: 1
Результат: 36dokrasnoper@dk3n57 ~/work/arch-pc/lab06 $
```

Рис. 4.26: Запуск исполняемого файла

5 Выводы

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