

# **Отчёт по лабораторной работе 6**

**Архитектура компьютеров**

Линь Хаоюнь

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Выполнение лабораторной работы</b>	<b>6</b>
2.1	Ответы на вопросы по программе variant.asm . . . . .	17
2.2	Самостоятельное задание . . . . .	18
<b>3</b>	<b>Выводы</b>	<b>21</b>

## Список иллюстраций

2.1	Подготовил каталог . . . . .	6
2.2	Программа в файле lab6-1.asm . . . . .	7
2.3	Запуск программы lab6-1.asm . . . . .	8
2.4	Программа в файле lab6-1.asm . . . . .	9
2.5	Запуск программы lab6-1.asm . . . . .	9
2.6	Программа в файле lab6-2.asm . . . . .	10
2.7	Запуск программы lab6-2.asm . . . . .	10
2.8	Программа в файле lab6-2.asm . . . . .	11
2.9	Запуск программы lab6-2.asm . . . . .	11
2.10	Запуск программы lab6-2.asm . . . . .	12
2.11	Программа в файле lab6-3.asm . . . . .	13
2.12	Запуск программы lab6-3.asm . . . . .	13
2.13	Программа в файле lab6-3.asm . . . . .	14
2.14	Запуск программы lab6-3.asm . . . . .	15
2.15	Программа в файле variant.asm . . . . .	16
2.16	Запуск программы variant.asm . . . . .	16
2.17	Программа в файле task.asm . . . . .	19
2.18	Запуск программы task.asm . . . . .	20

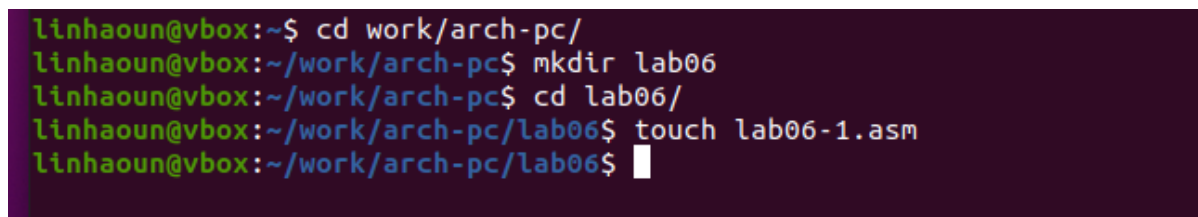
## **Список таблиц**

# 1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

## 2 Выполнение лабораторной работы

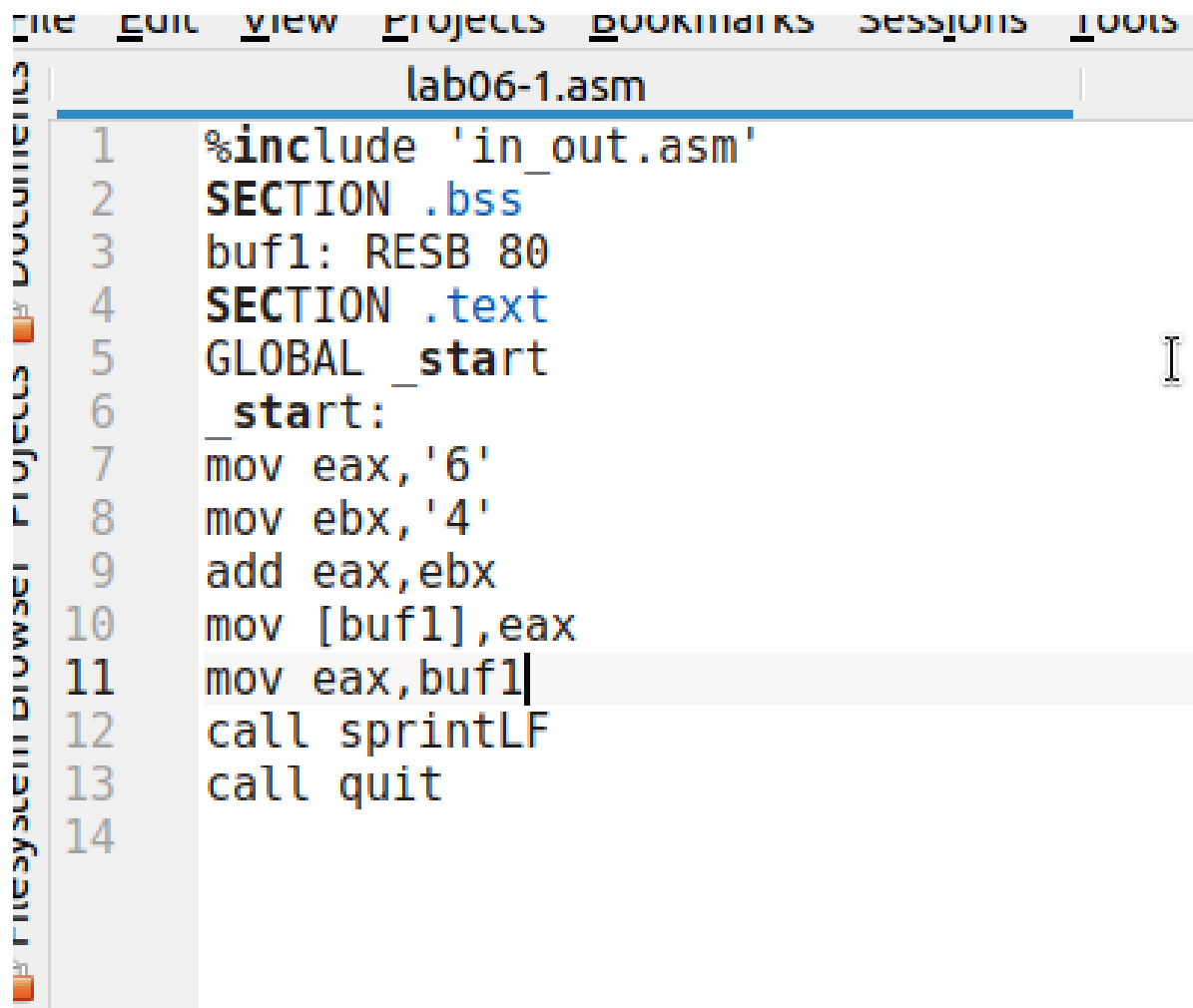
Создал каталог для программ лабораторной работы № 6, перешел в него и создал файл lab6-1.asm. (рис. 2.1)

A terminal window with a dark purple background and light green text. The text shows a series of commands being executed in a shell. The prompt is 'linhaoun@vbox:~\$'. The first command is 'cd work/arch-pc/'. The second prompt is 'linhaoun@vbox:~/work/arch-pc\$' and the command is 'mkdir lab06'. The third prompt is 'linhaoun@vbox:~/work/arch-pc\$' and the command is 'cd lab06/'. The fourth prompt is 'linhaoun@vbox:~/work/arch-pc/lab06\$' and the command is 'touch lab06-1.asm'. The fifth prompt is 'linhaoun@vbox:~/work/arch-pc/lab06\$' followed by a cursor.

```
linhaoun@vbox:~$ cd work/arch-pc/
linhaoun@vbox:~/work/arch-pc$ mkdir lab06
linhaoun@vbox:~/work/arch-pc$ cd lab06/
linhaoun@vbox:~/work/arch-pc/lab06$ touch lab06-1.asm
linhaoun@vbox:~/work/arch-pc/lab06$
```

Рис. 2.1: Подготовил каталог

Рассмотрим примеры программ вывода символьных и численных значений. Программы будут выводить значения, записанные в регистр eax.



```
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
14
```

Рис. 2.2: Программа в файле lab6-1.asm

В данной программе (рис. 2.2) мы записываем символ '6' в регистр `eax` (`mov eax, '6'`), а символ '4' в регистр `ebx` (`mov ebx, '4'`). Затем мы добавляем значение регистра `ebx` к значению в регистре `eax` (`add eax, ebx`, результат сложения записывается в регистр `eax`). После этого мы выводим результат. Однако, для использования функции `sprintf`, необходимо, чтобы в регистре `eax` был записан адрес, поэтому мы используем дополнительную переменную. Мы записываем значение регистра `eax` в переменную `buf1` (`mov [buf1], eax`), а затем записываем адрес переменной `buf1` в регистр `eax` (`mov eax, buf1`) и вызываем функцию `sprintf`.

```
linhaoun@vbox:~/work/arch-pc/lab06$ nasm -f elf lab06-1.asm
linhaoun@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-1.o -o lab06-1
linhaoun@vbox:~/work/arch-pc/lab06$ ./lab06-1
j
linhaoun@vbox:~/work/arch-pc/lab06$ □
```

Рис. 2.3: Запуск программы lab6-1.asm

В данном случае, когда мы ожидаем увидеть число 10 при выводе значения регистра `eax`, фактическим результатом будет символ 'j'. Это происходит из-за того, что код символа '6' равен 00110110 в двоичном представлении (или 54 в десятичном представлении), а код символа '4' равен 00110100 (или 52 в десятичном представлении). Когда мы выполняем команду `add eax, ebx`, результатом будет сумма кодов - 01101010 (или 106 в десятичном представлении), который соответствует символу 'j'. (рис. 2.3)

Далее изменяю текст программы и вместо символов, запишем в регистры числа. (рис. 2.4)



```

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
14

```

Рис. 2.4: Программа в файле lab6-1.asm

```

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

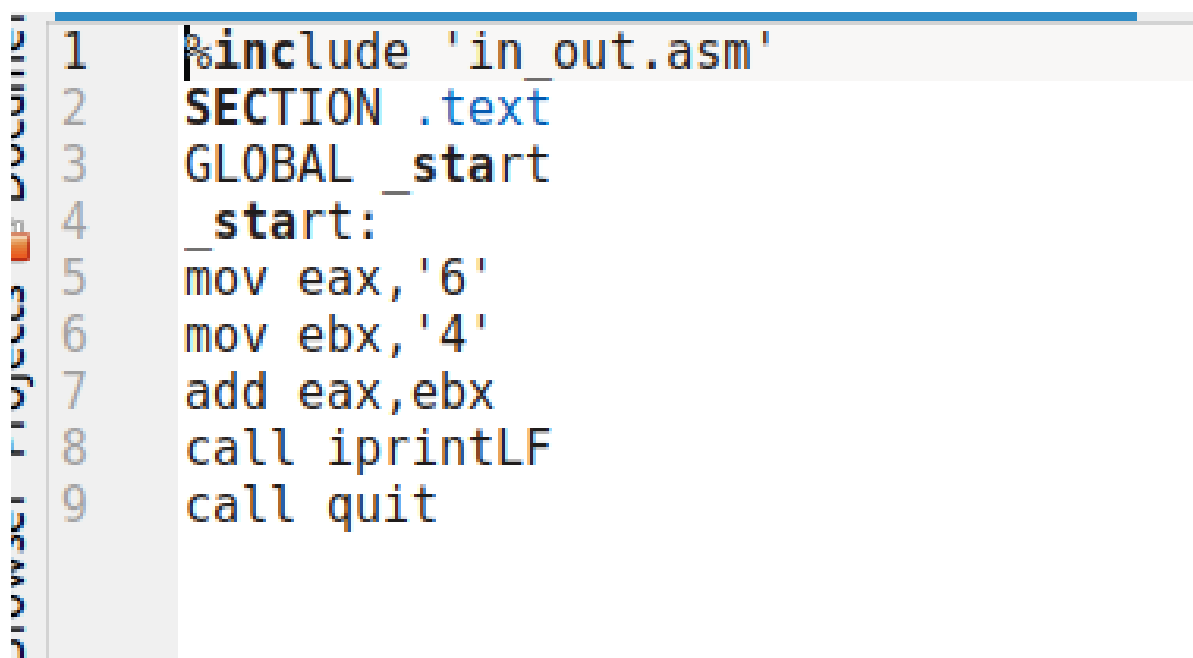
linhaoun@vbox:~/work/arch-pc/lab06$

```

Рис. 2.5: Запуск программы lab6-1.asm

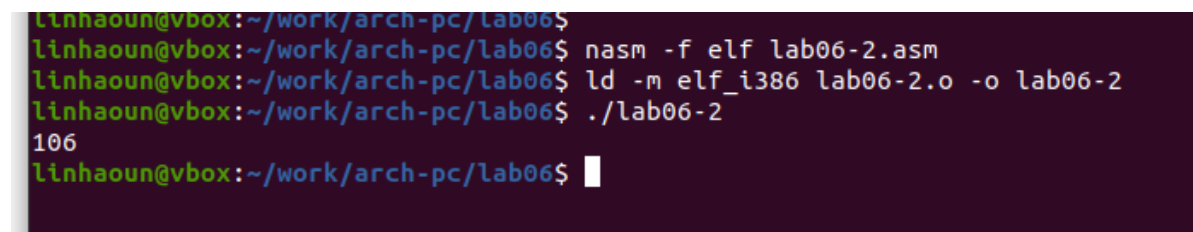
Как и в предыдущем случае, при выполнении программы мы не получим число 10. Вместо этого выводится символ с кодом 10, который представляет собой символ конца строки (возврат каретки). (рис. 2.5) Этот символ не отображается в консоли, но он добавляет пустую строку.

Как отмечалось выше, для работы с числами в файле in\_out.asm реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Преобразовал текст программы с использованием этих функций. (рис. 2.6)

A screenshot of a text editor showing an assembly program. The program is in the file lab6-2.asm. It includes 'in\_out.asm', sets the section to .text, and declares the global symbol \_start. The code starts at \_start, moves the ASCII value '6' into the EAX register, moves the ASCII value '4' into the EBX register, adds EBX to EAX, calls the iprintLF function to print the result, and then calls the quit function.

```
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. 2.6: Программа в файле lab6-2.asm

A screenshot of a terminal window showing the execution of the assembly program. The user is in the directory ~/work/arch-pc/lab06. They run 'nasm -f elf lab06-2.asm' to assemble the program, then 'ld -m elf\_i386 lab06-2.o -o lab06-2' to link it, and finally './lab06-2' to run it. The output of the program is the number 106.

```
linhaoun@vbox:~/work/arch-pc/lab06$
linhaoun@vbox:~/work/arch-pc/lab06$ nasm -f elf lab06-2.asm
linhaoun@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-2.o -o lab06-2
linhaoun@vbox:~/work/arch-pc/lab06$ ./lab06-2
106
linhaoun@vbox:~/work/arch-pc/lab06$
```

Рис. 2.7: Запуск программы lab6-2.asm

В результате выполнения программы мы получим число 106. (рис. 2.7) В данном случае, как и в первом случае, команда add складывает коды символов '6' и '4' ( $54+52=106$ ). Однако, в отличие от предыдущей программы, функция iprintLF позволяет вывести число, а не символ, кодом которого является это число.

Аналогично предыдущему примеру изменим символы на числа.(рис. 2.8)

```

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. 2.8: Программа в файле lab6-2.asm

Функция `iprintLF` позволяет вывести число и операндами были числа (а не коды символов). Поэтому получаем число 10.(рис. 2.9)

```

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

```

Рис. 2.9: Запуск программы lab6-2.asm

Заменяю функцию `iprintLF` на `iprint`. Создал исполняемый файл и запустил его. Вывод отличается тем, что нет переноса строки.(рис. 2.10)

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

Рис. 2.10: Запуск программы lab6-2.asm

В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения (рис. 2.11) (рис. 2.12)

$$f(x) = (5 * 2 + 3) / 3$$

.

```

1  %include 'in_out.asm'
2  SECTION .data
3  div: DB 'Результат: ',0
4  rem: DB 'Остаток от деления: ',0
5  SECTION .text
6  GLOBAL _start
7  _start:
8
9  mov eax,5
10 mov ebx,2
11 mul ebx
12 add eax,3
13 xor edx,edx
14 mov ebx,3
15 div ebx
16 mov edi,eax
17 mov eax,div
18 call sprint
19 mov eax,edi
20 call iprintLF
21 mov eax,rem
22 call sprint
23 mov eax,edx
24 call iprintLF
25 call quit
26

```

Рис. 2.11: Программа в файле lab6-3.asm

```

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

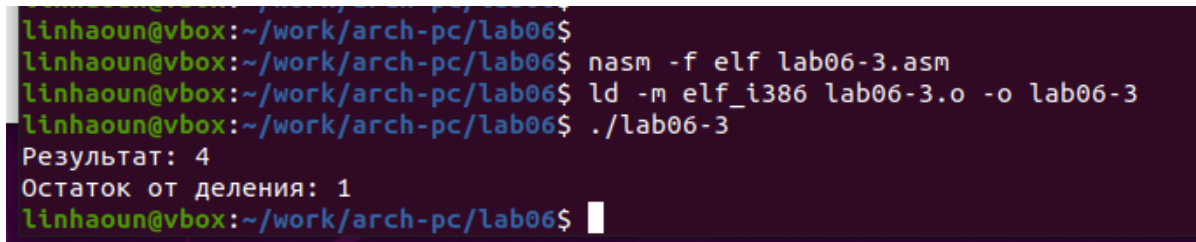
```

Рис. 2.12: Запуск программы lab6-3.asm

Изменил текст программы для вычисления выражения

$$f(x) = (4 * 6 + 2) / 5$$

. Создал исполняемый файл и проверил его работу. (рис. 2.13) (рис. 2.14)

A terminal window with a dark purple background and green text. It shows the compilation of an assembly file 'lab06-3.asm' using 'nasm' and 'ld' to create an executable 'lab06-3'. The program is then executed, displaying the result '4' and the remainder '1' from a division.

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

Рис. 2.13: Программа в файле lab6-3.asm

```
lab06-3.asm
1  %include 'in_out.asm'
2  SECTION .data
3  div: DB 'Результат: ',0
4  rem: DB 'Остаток от деления: ',0
5  SECTION .text
6  GLOBAL _start
7  _start:
8
9  mov eax,4
10 mov ebx,6
11 mul ebx
12 add eax,2
13 xor edx,edx
14 mov ebx,5
15 div ebx
16 mov edi,eax
17 mov eax,div
18 call sprint
19 mov eax,edi
20 call iprintLF
21 mov eax,rem
22 call sprint
23 mov eax,edx
24 call iprintLF
25 call quit
26
```

Рис. 2.14: Запуск программы lab6-3.asm

В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета. (рис. 2.15) (рис. 2.16)

В данном случае число, над которым необходимо проводить арифметические операции, вводится с клавиатуры. Как отмечалось выше ввод с клавиатуры осуществляется в символьном виде и для корректной работы арифметических операций в NASM символы необходимо преобразовать в числа. Для этого может быть использована функция `atoi` из файла `in_out.asm`.

```
variant.asm
1  %include 'in_out.asm'
2  |
3  SECTION .data
4  msg: DB 'Введите № студенческого билета: ',0
5  rem: DB 'Ваш вариант: ',0
6  SECTION .bss
7  x: RESB 80
8  SECTION .text
9  GLOBAL _start
10 _start:
11  mov eax, msg
12  call sprintf
13  mov ecx, x
14  mov edx, 80
15  call sread
16  mov eax,x
17  call atoi
18  xor edx,edx
19  mov ebx,20
20  div ebx
21  inc edx
22  mov eax,rem
23  call sprintf
24  mov eax,edx
25  call iprintLF
26  call quit
27
```

Рис. 2.15: Программа в файле variant.asm

```
linhaoun@vbox:~/work/arch-pc/lab06$
linhaoun@vbox:~/work/arch-pc/lab06$ nasm -f elf variant.asm
linhaoun@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 variant.o -o variant
linhaoun@vbox:~/work/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1032238264
Ваш вариант: 5
linhaoun@vbox:~/work/arch-pc/lab06$
```

Рис. 2.16: Запуск программы variant.asm



## 2.1 Ответы на вопросы по программе variant.asm

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

Строка “mov eax, mem” перекладывает в регистр значение переменной с фразой “Ваш вариант:”

Строка “call sprint” вызывает подпрограмму вывода строки

2. Для чего используются следующие инструкции?

Инструкция “pasm” используется для компиляции кода на языке ассемблера NASM

Инструкция “mov ecx, x” используется для перемещения значения переменной x в регистр ecx

Инструкция “mov edx, 80” используется для перемещения значения 80 в регистр edx

Инструкция “call sread” вызывает подпрограмму для считывания значения студенческого билета из консоли

3. Для чего используется инструкция “call atoi”?

Инструкция “call atoi” используется для преобразования введенных символов в числовой формат

4. Какие строки листинга отвечают за вычисления варианта?

Строка “xor edx, edx” обнуляет регистр edx

Строка “mov ebx, 20” записывает значение 20 в регистр ebx

Строка “div ebx” выполняет деление номера студенческого билета на 20

Строка “inc edx” увеличивает значение регистра edx на 1

5. В какой регистр записывается остаток от деления при выполнении инструкции “div ebx”?

Остаток от деления записывается в регистр `edx`

6. Для чего используется инструкция `inc edx`?

Инструкция `inc edx` используется для увеличения значения в регистре `edx` на 1, в соответствии с формулой вычисления варианта

7. Какие строки листинга отвечают за вывод на экран результата вычислений?

Строка `mov eax, edx` перекладывает результат вычислений в регистр `eax`

Строка `call iprintLF` вызывает подпрограмму для вывода значения на экран

## 2.2 Самостоятельное задание

Написать программу вычисления выражения  $y = f(x)$ . Программа должна выводить выражение для вычисления, выводить запрос на ввод значения  $x$ , вычислять заданное выражение в зависимости от введенного  $x$ , выводить результат вычислений. Вид функции  $f(x)$  выбрать из таблицы 6.3 вариантов заданий в соответствии с номером полученным при выполнении лабораторной работы. Создайте исполняемый файл и проверьте его работу для значений  $x_1$  и  $x_2$  из 6.3.

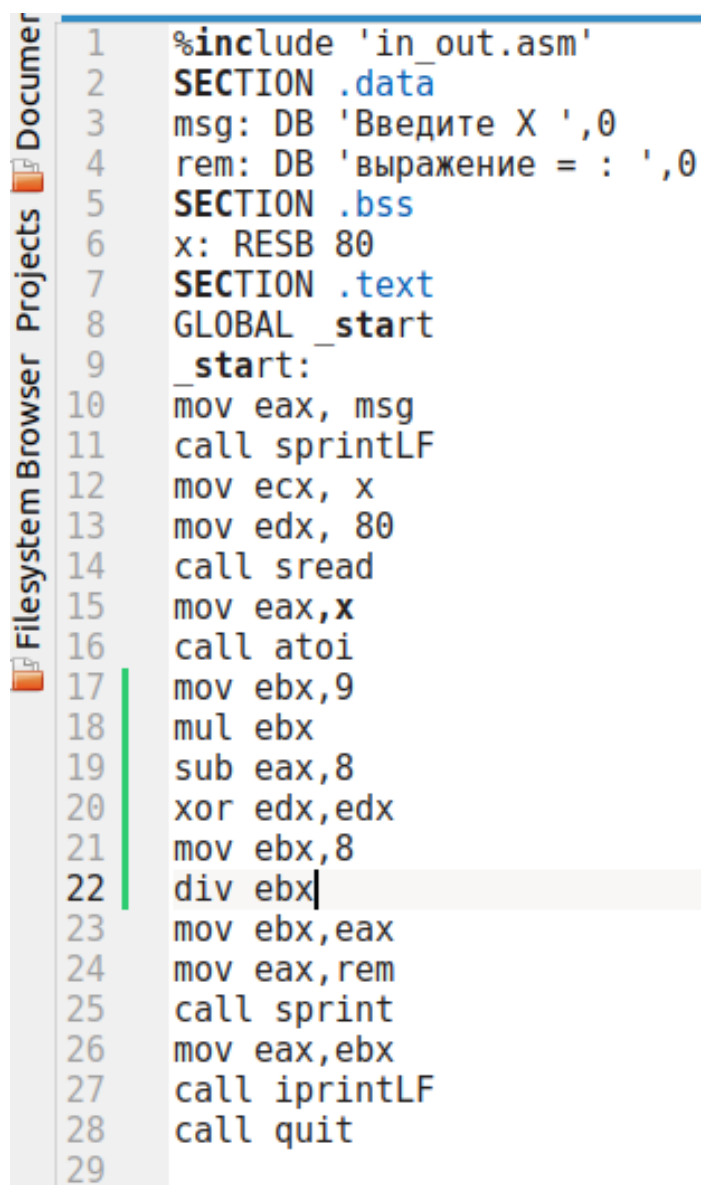
Получили вариант 5 -

$$(9x - 8)/8$$

для

$$x_1 = 8, x_2 = 64$$

(рис. 2.17) (рис. 2.18)



```

1  %include 'in_out.asm'
2  SECTION .data
3  msg: DB 'Введите X ',0
4  rem: DB 'выражение = : ',0
5  SECTION .bss
6  x: RESB 80
7  SECTION .text
8  GLOBAL _start
9  _start:
10 mov eax, msg
11 call sprintfLF
12 mov ecx, x
13 mov edx, 80
14 call sread
15 mov eax,x
16 call atoi
17 mov ebx,9
18 mul ebx
19 sub eax,8
20 xor edx,edx
21 mov ebx,8
22 div ebx
23 mov ebx,eax
24 mov eax,rem
25 call sprintf
26 mov eax,ebx
27 call iprintLF
28 call quit
29

```

Рис. 2.17: Программа в файле task.asm

```
linhaoun@vbox:~/work/arch-pc/lab06$  
linhaoun@vbox:~/work/arch-pc/lab06$ nasm -f elf task.asm  
linhaoun@vbox:~/work/arch-pc/lab06$ ld -m elf_i386 task.o -o task  
linhaoun@vbox:~/work/arch-pc/lab06$ ./task  
Введите X  
8  
выражение = : 8  
linhaoun@vbox:~/work/arch-pc/lab06$ ./task  
Введите X  
64  
выражение = : 71  
linhaoun@vbox:~/work/arch-pc/lab06$
```

Рис. 2.18: Запуск программы task.asm

## **3 Выводы**

Изучили работу с арифметическими операциями.