

Отчёт по лабораторной работе 6

Архитектура компьютеров

ТРАОРЕ АНРИ НОЭЛЬ

Содержание

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

Список иллюстраций

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

Список таблиц

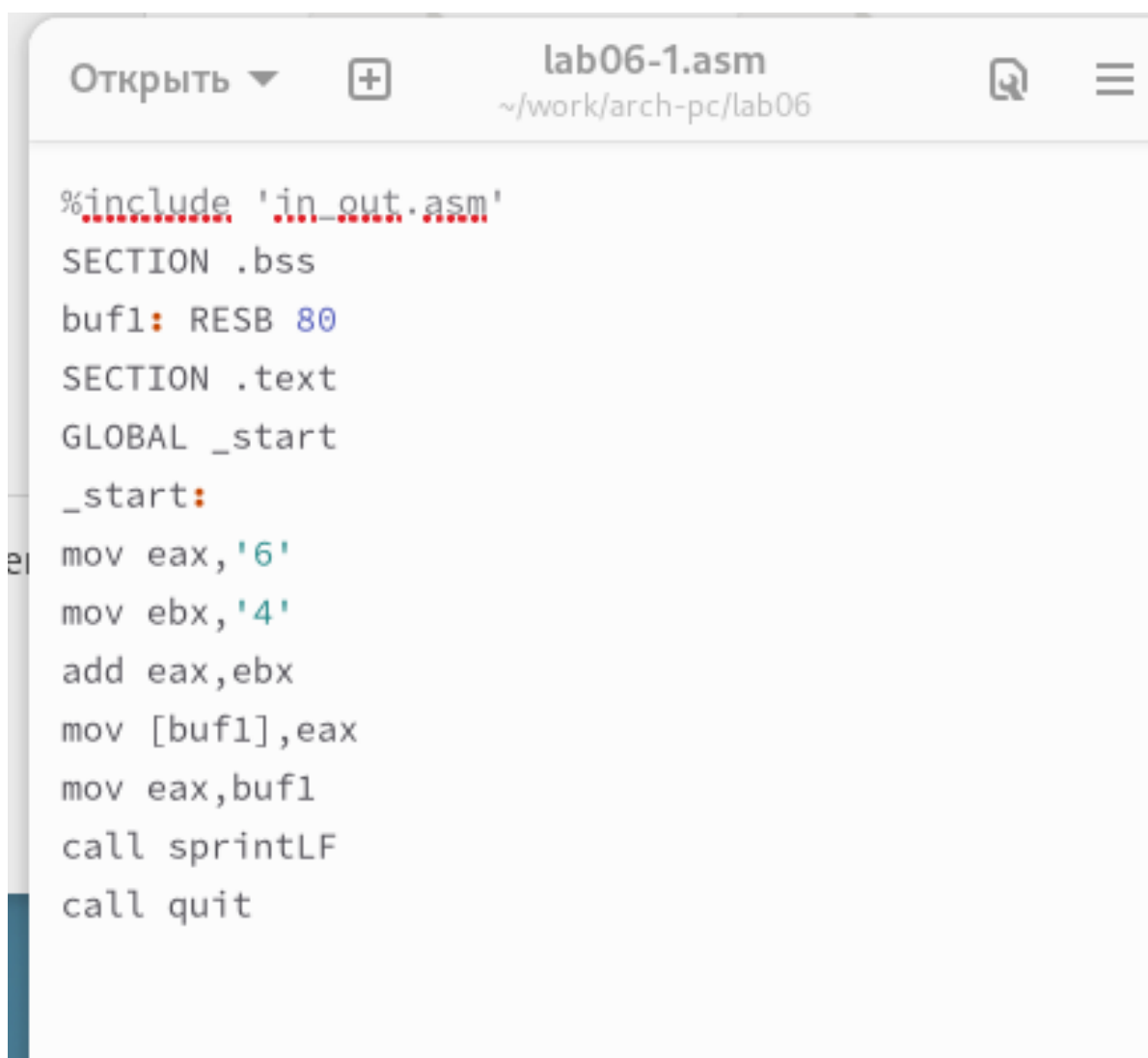
1 Цель работы

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.

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



```
Открыть ▾  +  lab06-1.asm  ~/work/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 printf
call quit
```

Рис. 2.1: Программа в файле lab6-1.asm

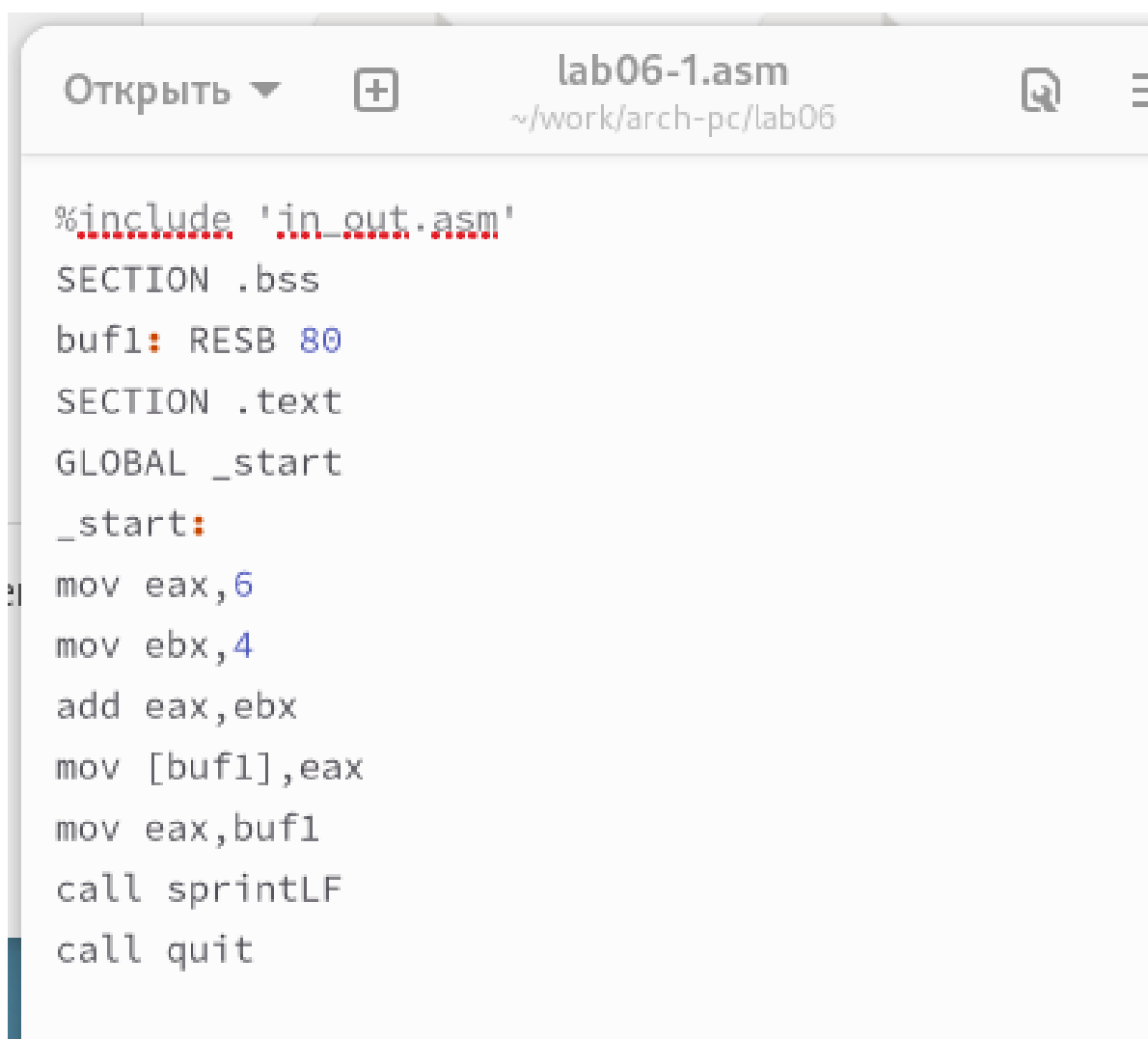
В данной программе (рис. 2.1) мы записываем символ '6' в регистр `eax` (`mov eax, '6'`), а символ '4' в регистр `ebx` (`mov ebx, '4'`). Затем мы добавляем значение регистра `ebx` к значению в регистре `eax` (`add eax, ebx`, результат сложения записывается в регистр `eax`). После этого мы выводим результат. 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`.

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

Рис. 2.2: Запуск программы lab6-1.asm

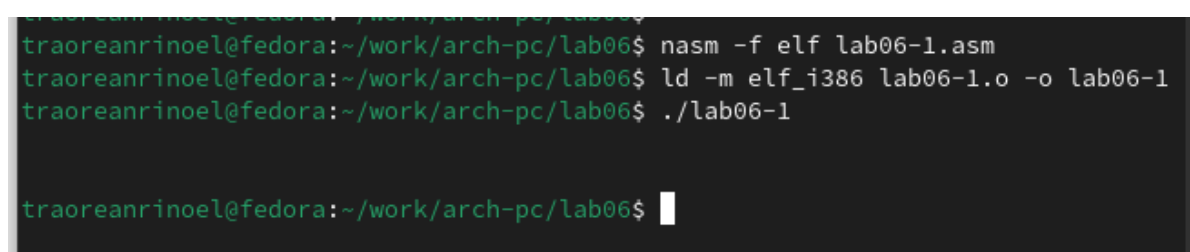
В данном случае, когда мы ожидаем увидеть число 10 при выводе значения регистра `eax`, фактическим результатом будет символ 'j'. Это происходит из-за того, что код символа '6' равен 00110110 в двоичном представлении (или 54 в десятичном представлении), а код символа '4' равен 00110100 (или 52 в десятичном представлении). Когда мы выполняем команду `add eax, ebx`, результатом будет сумма кодов - 01101010 (или 106 в десятичном представлении), который соответствует символу 'j'. (рис. 2.2)

Далее изменяю текст программы и вместо символов, запишем в регистры числа. (рис. 2.3)



```
%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. 2.3: Программа в файле lab6-1.asm



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

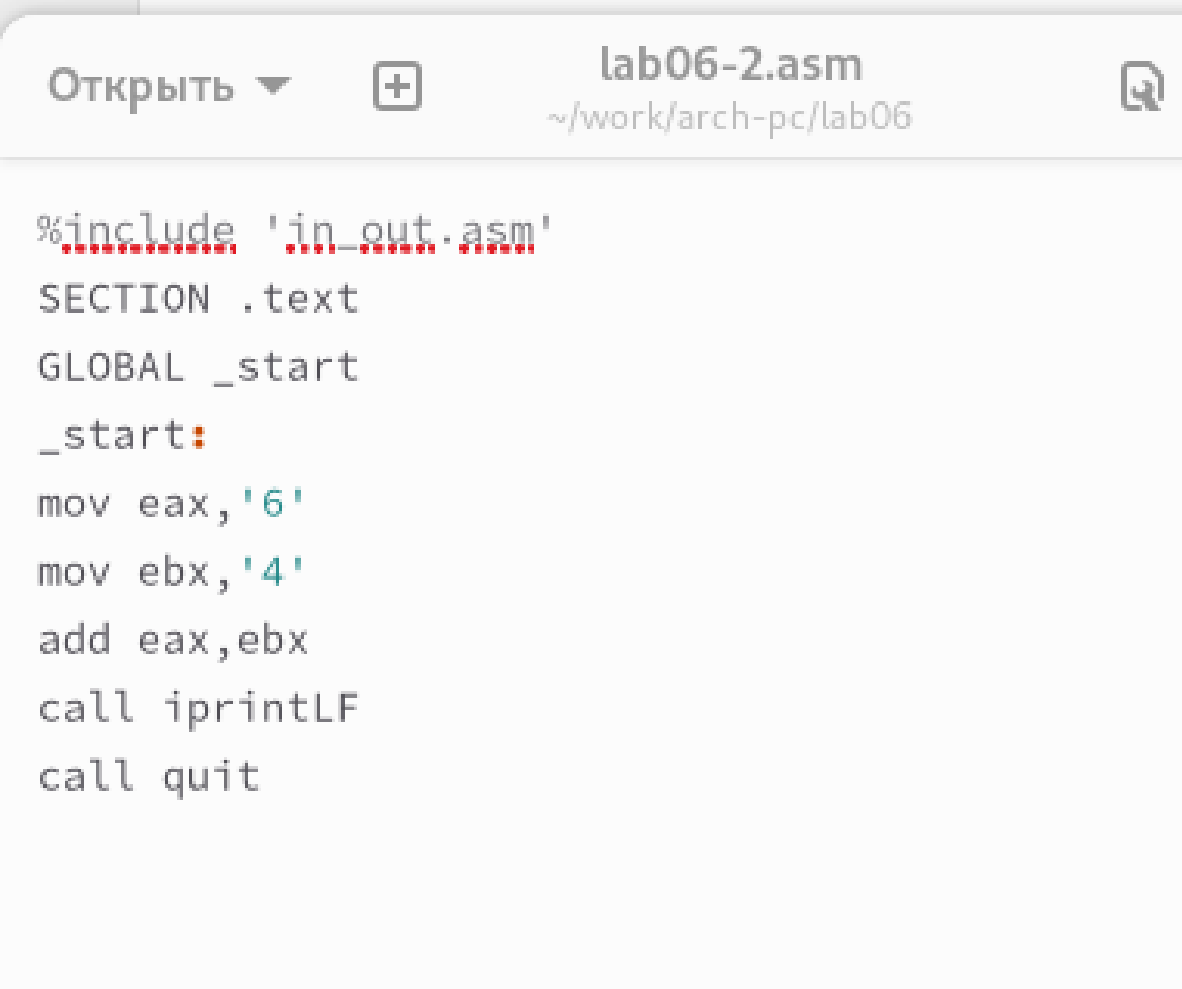
traoreanrinoel@fedora:~/work/arch-pc/lab06$
```

Рис. 2.4: 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. 2.4) Этот символ не отображается в консоли, но он добавляет пустую строку.

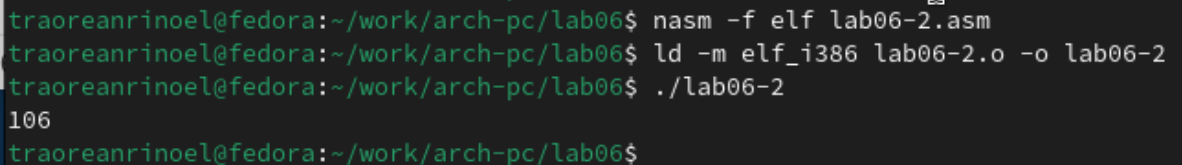
Как отмечалось выше, для работы с числами в файле `in_out.asm` реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Преобразовал текст программы с использованием этих функций. (рис. 2.5)



```
Открыть ▼ + lab06-2.asm ~/work/arch-pc/lab06

%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax, '6'
mov ebx, '4'
add eax, ebx
call iprintLF
call quit
```

Рис. 2.5: Программа в файле `lab6-2.asm`

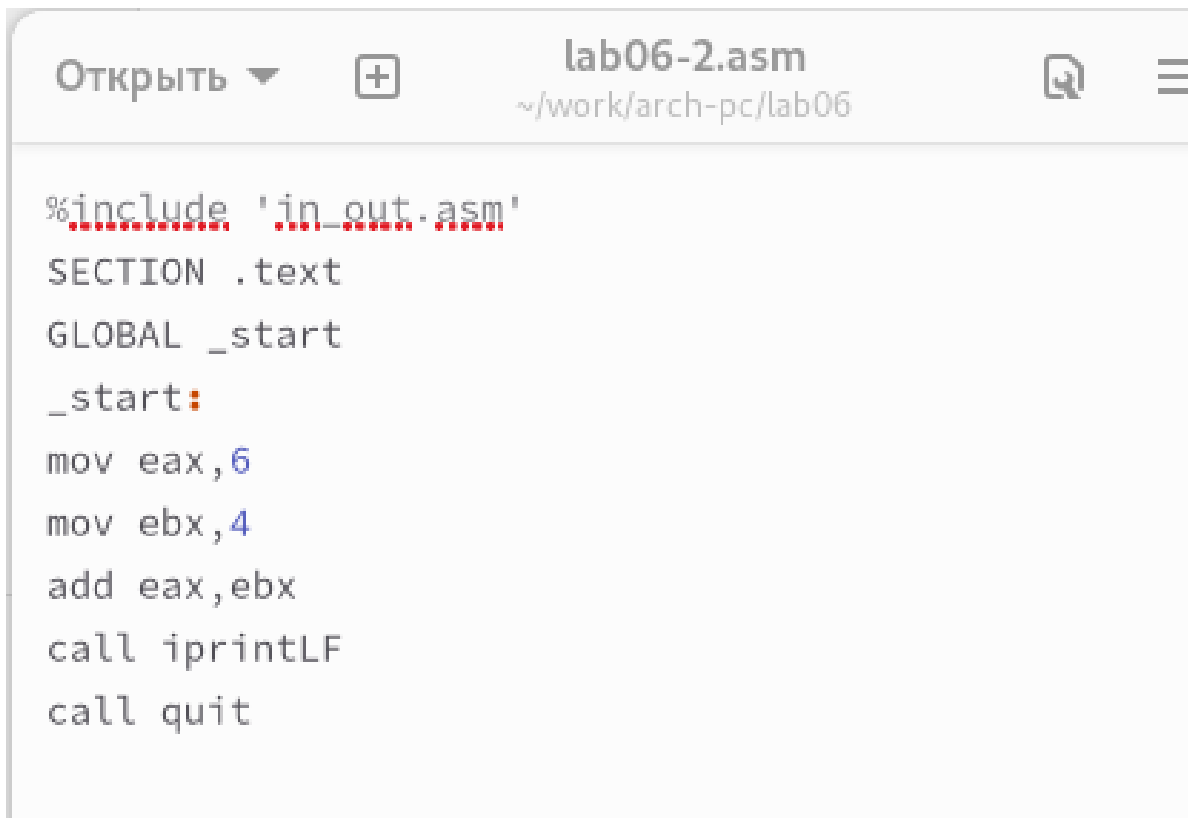


```
traoreanrinoel@fedora:~/work/arch-pc/lab06$ nasm -f elf lab06-2.asm
traoreanrinoel@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-2.o -o lab06-2
traoreanrinoel@fedora:~/work/arch-pc/lab06$ ./lab06-2
106
traoreanrinoel@fedora:~/work/arch-pc/lab06$
```

Рис. 2.6: Запуск программы `lab6-2.asm`

В результате выполнения программы мы получим число 106. (рис. 2.6) 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. 2.7)

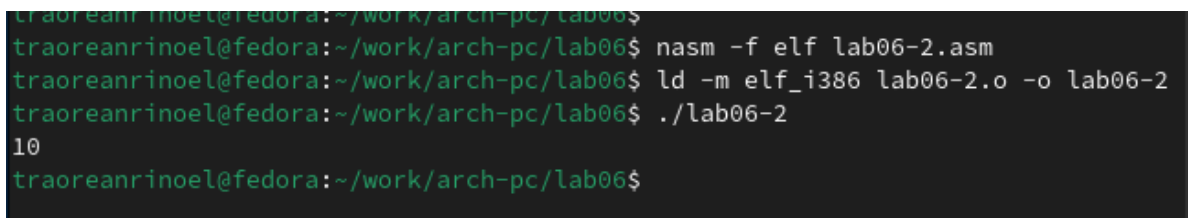


```
Открыть ▾  lab06-2.asm  ~/work/arch-pc/lab06

%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprintLF
call quit
```

Рис. 2.7: Программа в файле lab6-2.asm

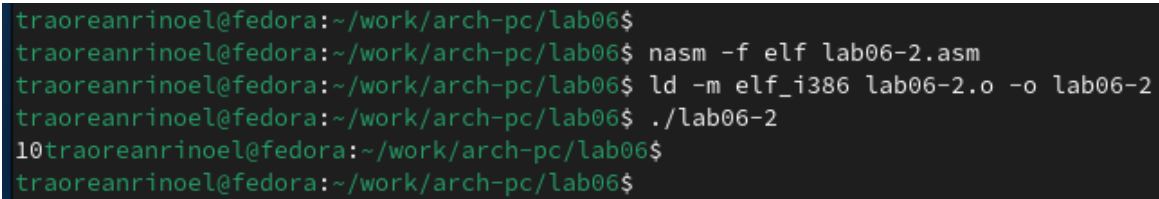
Функция iprintLF позволяет вывести число и операндами были числа (а не коды символов). Поэтому получаем число 10.(рис. 2.8)



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

Рис. 2.8: Запуск программы lab6-2.asm

Заменяю функцию `iprintLF` на `iprint`. Создал исполняемый файл и запустил его. Вывод отличается тем, что нет переноса строки. (рис. 2.9)



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

Рис. 2.9: Запуск программы `lab6-2.asm`

В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения (рис. 2.10) (рис. 2.11)

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

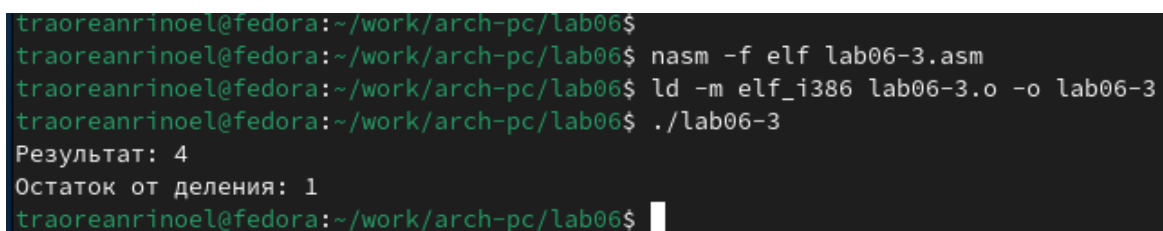
.



```
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0
SECTION .text
GLOBAL _start
_start:

mov eax,5
mov ebx,2
mul ebx
add eax,3
xor edx,edx
mov ebx,3
div ebx
mov edi,eax
mov eax,div
call sprint
mov eax,edi
call iprintLF
mov eax,rem
call sprint
mov eax,edx
call iprintLF
call quit
```

Рис. 2.10: Программа в файле lab6-3.asm



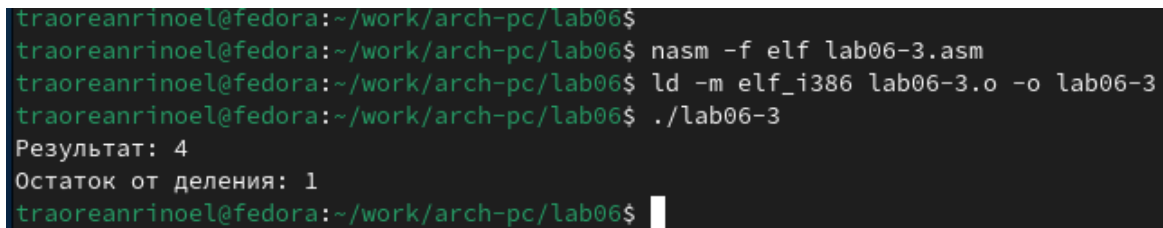
```
traoreanrinoel@fedora:~/work/arch-pc/lab06$
traoreanrinoel@fedora:~/work/arch-pc/lab06$ nasm -f elf lab06-3.asm
traoreanrinoel@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-3.o -o lab06-3
traoreanrinoel@fedora:~/work/arch-pc/lab06$ ./lab06-3
Результат: 4
Остаток от деления: 1
traoreanrinoel@fedora:~/work/arch-pc/lab06$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

Рис. 2.11: 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. 2.12) (рис. 2.13)



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

Рис. 2.12: Программа в файле lab6-3.asm



```
Открыть ▾ + lab06-3.asm
~/work/arch-pc/lab06

%include 'in_out.asm'
SECTION .data
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,edi
call iprintLF
mov eax,rem
call sprint
mov eax,edx
call iprintLF
call quit
```

Рис. 2.13: Запуск программы lab6-3.asm

В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета. (рис. 2.14) (рис. 2.15)

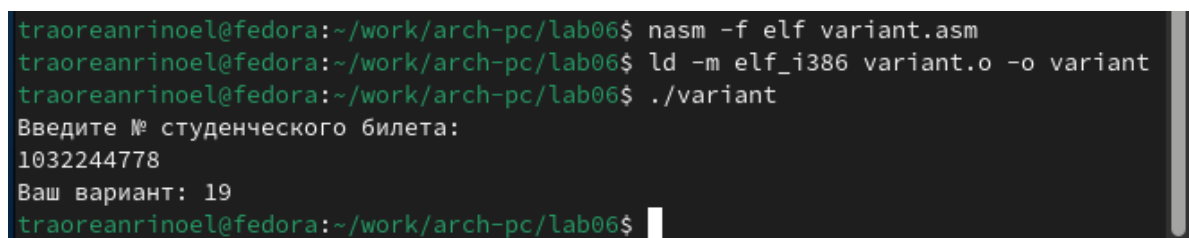
В данном случае число, над которым необходимо проводить арифметические операции, вводится с клавиатуры. Как отмечалось выше ввод с клавиатуры осуществляется в символьном виде и для корректной работы арифметических операций в NASM символы необходимо преобразовать в числа. Для этого может быть использована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



```
Открыть ▾ + variant.asm
~/work/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 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 eax, edx
call iprintLF
call quit
```

Рис. 2.14: Программа в файле variant.asm



```
traoreanrinoel@fedora:~/work/arch-pc/lab06$ nasm -f elf variant.asm
traoreanrinoel@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 variant.o -o variant
traoreanrinoel@fedora:~/work/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1032244778
Ваш вариант: 19
traoreanrinoel@fedora:~/work/arch-pc/lab06$
```

Рис. 2.15: 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 “nasm” 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.

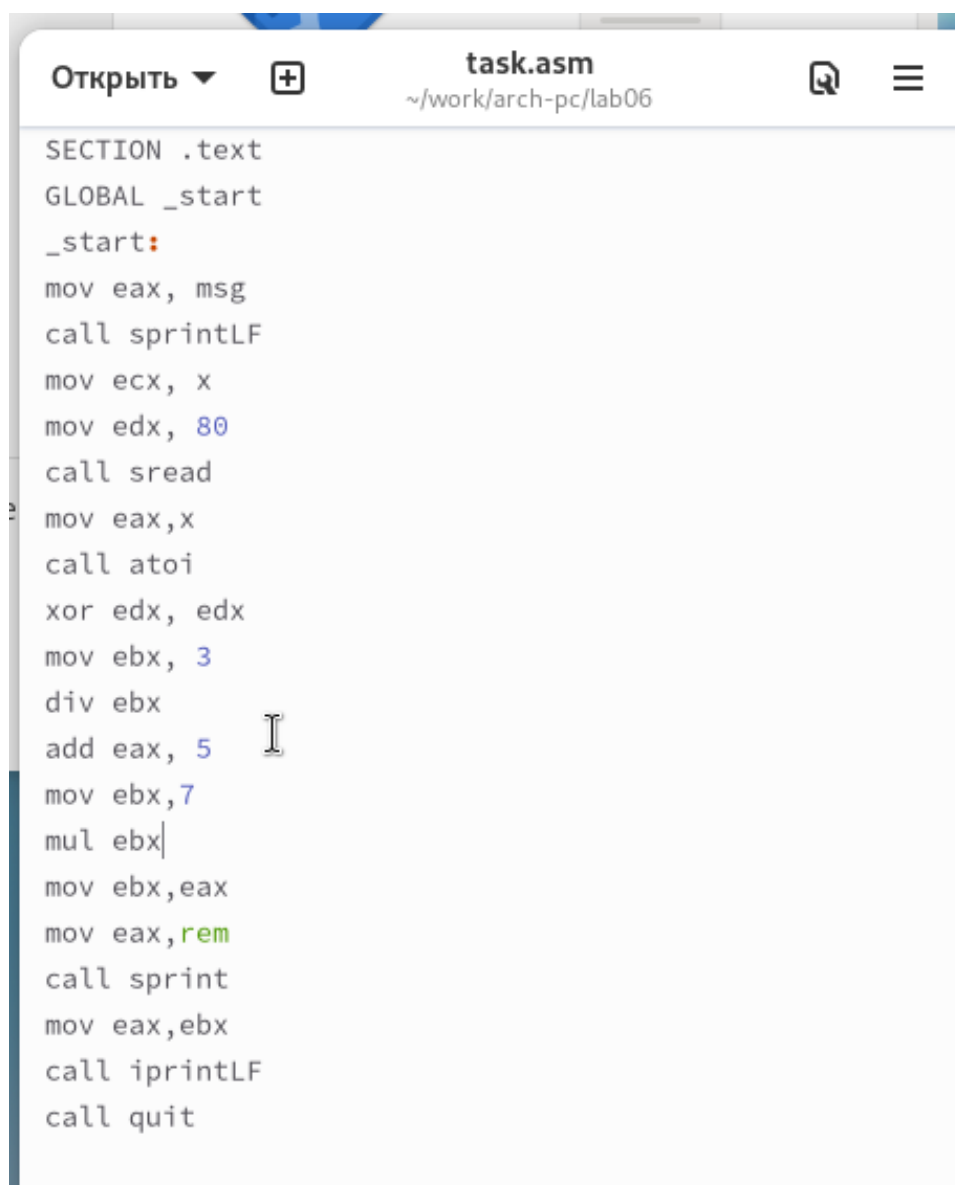
Получили вариант 19 -

$$(x/3 + 5) * 7$$

для

$$x_1 = 1, x_2 = 9$$

(рис. 2.16) (рис. 2.17)

A screenshot of a text editor window titled "task.asm" with a path "~/work/arch-pc/lab06". The editor contains assembly code for a program. The code starts with a section declaration ".text" and a global symbol "_start". It then defines "_start:" and proceeds with several instructions: moving "msg" to "eax", calling "sprintf", moving "x" to "ecx", moving "80" to "edx", calling "sread", moving "x" to "eax", calling "atoi", XORing "edx" with itself, moving "3" to "ebx", dividing "ebx" by "ebx", adding "5" to "eax", moving "7" to "ebx", multiplying "ebx" by "eax", moving the result back to "ebx", moving "rem" to "eax", calling "sprintf", moving "ebx" to "eax", calling "iprintLF", and finally calling "quit".

```
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, 3
div ebx
add eax, 5
mov ebx, 7
mul ebx
mov ebx, eax
mov eax, rem
call sprintf
mov eax, ebx
call iprintLF
call quit
```

Рис. 2.16: Программа в файле task.asm

```
traoreanrinoel@fedora:~/work/arch-pc/lab06$  
traoreanrinoel@fedora:~/work/arch-pc/lab06$ nasm -f elf task.asm  
traoreanrinoel@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 task.o -o task  
traoreanrinoel@fedora:~/work/arch-pc/lab06$ ./task  
Введите X  
3  
выражение = : 42  
traoreanrinoel@fedora:~/work/arch-pc/lab06$ ./task  
Введите X  
9  
выражение = : 56  
traoreanrinoel@fedora:~/work/arch-pc/lab06$
```

Рис. 2.17: Запуск программы task.asm

3 Выводы

Изучили работу с арифметическими операциями.