

# **Отчёта по лабораторной работе 6**

**Освоение арифметических инструкций языка ассемблера NASM.**

Исмагамбетов Владимир НБИбд-04-22

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
<b>5</b>	<b>Выводы</b>	<b>24</b>
	<b>Список литературы</b>	<b>25</b>

## Список иллюстраций

4.1	Пример программы . . . . .	10
4.2	Работа программы . . . . .	11
4.3	Пример программы . . . . .	12
4.4	Работа программы . . . . .	12
4.5	Пример программы . . . . .	13
4.6	Работа программы . . . . .	13
4.7	Пример программы . . . . .	14
4.8	Работа программы . . . . .	15
4.9	Работа программы . . . . .	15
4.10	Пример программы . . . . .	16
4.11	Работа программы . . . . .	17
4.12	Пример программы . . . . .	18
4.13	Работа программы . . . . .	19
4.14	Пример программы . . . . .	20
4.15	Работа программы . . . . .	20
4.16	Пример программы . . . . .	22
4.17	Работа программы . . . . .	23

## Список таблиц

# 1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

## 2 Задание

1. Изучите примеры программ.
2. Напишите программу вычисления выражения в соответствии с вариантом.
3. Загрузите файлы на GitHub.

### 3 Теоретическое введение

В основном наборе инструкций входят разные вариации четырех арифметических действий: сложение, вычитание, умножение, деление. Важно помнить, что в результате арифметических действий меняются некоторые биты регистра флагов, что позволяет выполнять команду условного перехода, т.е. разветвлять программу на основе результат операции. Замечу, что для команд сложения и вычитания справедливыми являются отмеченное выше для операндов команды `mov`. К командам сложения можно отнести: `add` – обычное сложение, `adc` – сложение с добавлением результату флага переноса в качестве единицы (если флаг равен нулю, то команда эквивалентна команде `add`), `xadd` – сложение, с предварительным обменом данных между операндами, `inc` – прибавление единицы к содержимому операнда. Несколько примеров: `add %rbx, dt` (или `addq, dt`, где четко указано, что складываются 64-битовые величины) – к содержимому области памяти `dt` добавляется содержимое регистра `rbx` и результат помещается в `dt`; `adc %rdx, %rdx` – удвоение содержимого регистра `rdx` плюс добавление значения флага переноса; `incl ll` – увеличение на единицу содержимого памяти по адресу `ll`. При этом явно указывается, что операнд имеет размер 32 бита (`dword`).

К командам вычитания можно отнести следующие инструкции процессора x86-64: `sub` – обычное вычитание, `sbb` – вычитание из результата флага переноса в качестве единицы (если флаг равен нулю, то команда эквивалентна `sub`), `dec` – вычитание единицы из результата, `neg` – вычитание значения операнда из 0. Несколько примеров: `sub %rax, ll` – из содержимого `ll` вычитается содержимое

регистра `rax` (или явно `subq %rax, ll`, где указывается, что операнды имеют 64-размер), и результат помещается в `ll`; `subw go, %ax` – вычитание из содержимого `ax` числа по адресу `go`, результат помещается в `ax`; `sbb %rdx, %rax` – вычитание с дополнительным вычитанием флага переноса (из числа в `rax` вычитается число в `rdx` и результат в `rax`); `decbl` – вычитание единицы из байта, расположенного по адресу `l`. Следует отметить еще специальную команду `cmpr`, которая во всем похожа на команду `sub`, кроме одного – результат вычитания никуда не помещается. Инструкция используется специально, для сравнения операндов.

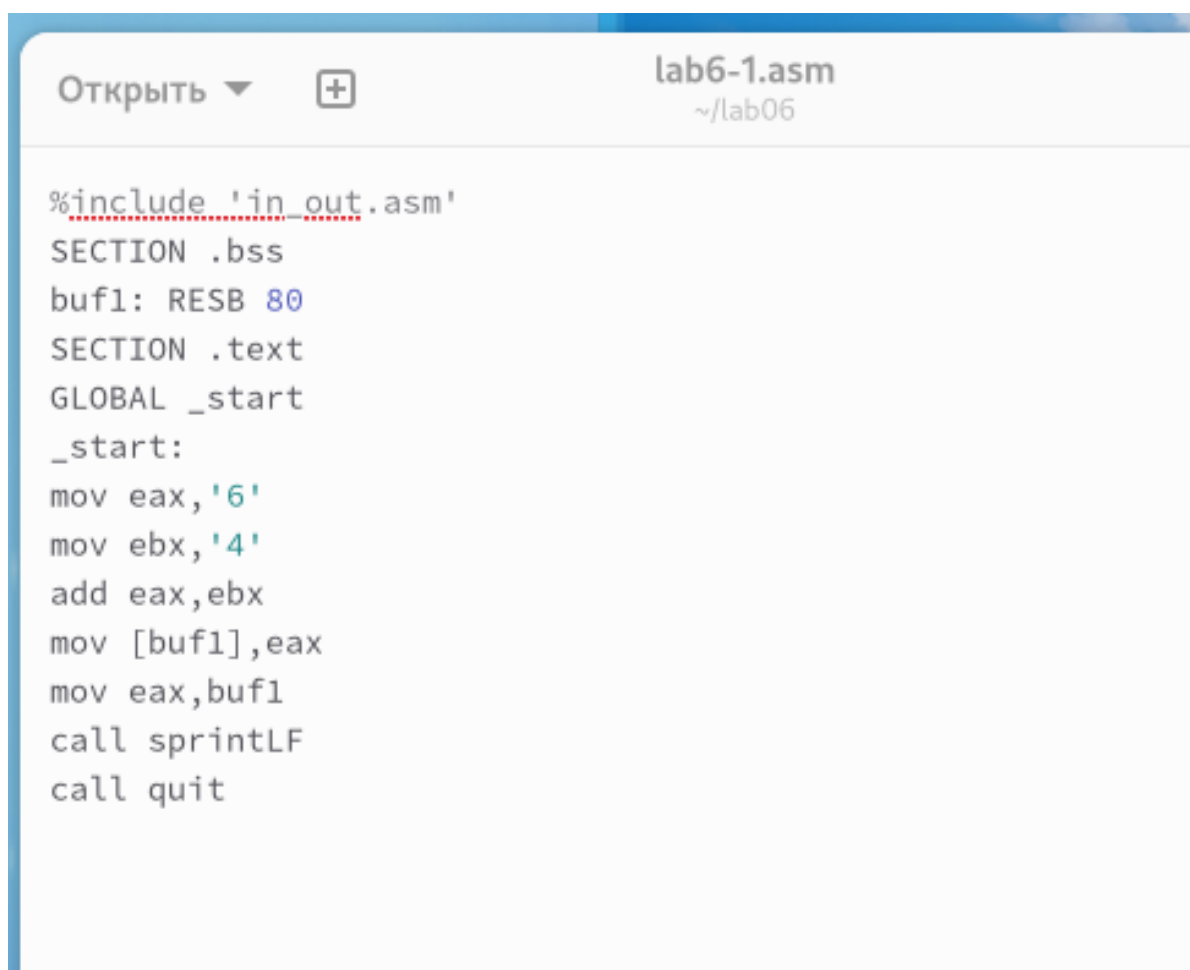
Две основные команды умножения: `mul` – умножение беззнаковых чисел, `imul` – умножение знаковых чисел. Команда содержит один операнд – регистр или адрес памяти. В зависимости от размера операнда данные помещаются: в `ax`, `dx : ax`, `edx : eax`, `rdx : rax`. Например: `mull ll` – содержимое памяти с адресом `ll` будет умножено на содержимое `eax` (не забываем о суффиксе `l`), а результат отправлен в пару регистров `edx : eax`; `mul %dl` – умножить содержимое регистра `dl` на содержимое регистра `al`, а результат положить в `ax`; `mul %r8` – умножить содержимое регистра `r8` на содержимое регистра `rax`, а результат положить в пару регистров `rdx : rax`.

Для деления (целого) также предусмотрены две команды: `div` – беззнаковое деление, `idiv` – знаковое деление. Инструкция также имеет один операнд – делитель. В зависимости от его размера результат помещается: `al` – результат деления, `ah` – остаток от деления; `ax` – результат деления, `dx` – остаток от деления; `eax` – результат деления, `edx` – остаток от деления; `rax` – результат деления, `rdx` – остаток от деления. Приведем примеры: `divl dv` – содержимое `edx : eax` делится на делитель, находящийся в памяти по адресу `dv` и результат деления помещается в `eax`, остаток в `edx`; `div %rsi` – содержимое `rdx : rax` делится на содержимое `rsi`, результат помещается в `rax`, остаток в `rdx`.



## 4 Выполнение лабораторной работы

1. Создайте каталог для программ лабораторной работы № 6, перейдите в него и создайте файл lab7-1.asm:
2. Рассмотрим примеры программ вывода символьных и численных значений. Программы будут выводить значения, записанные в регистр eax. (рис. 4.1, 4.2)

The image shows a screenshot of a code editor window. The title bar at the top contains the text "Открыть" (Open) with a dropdown arrow, a plus icon in a square, and the filename "lab6-1.asm" with the path "~/lab06" below it. The main area of the editor contains assembly code. The first line is "%include 'in\_out.asm'" with a red dotted underline. The code defines a .bss section with a buffer "buf1" of 80 bytes, and a .text section with a global "\_start" label. The program logic starts at "\_start:", moves the value '6' into the EAX register, moves the value '4' into the EBX register, adds EBX to EAX, stores the result in the memory location [buf1], moves the address of buf1 into EAX, calls the "sprintf" function, and finally calls the "quit" function.

```
Открыть ▾ + lab6-1.asm  
~/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. 4.1: Пример программы

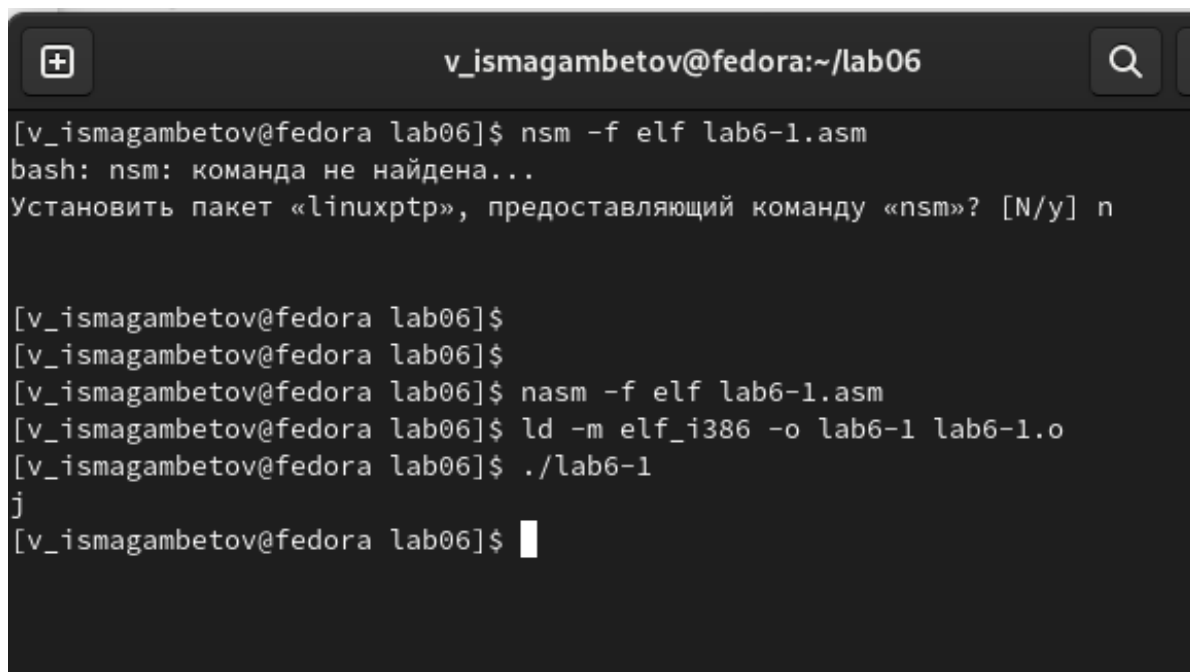
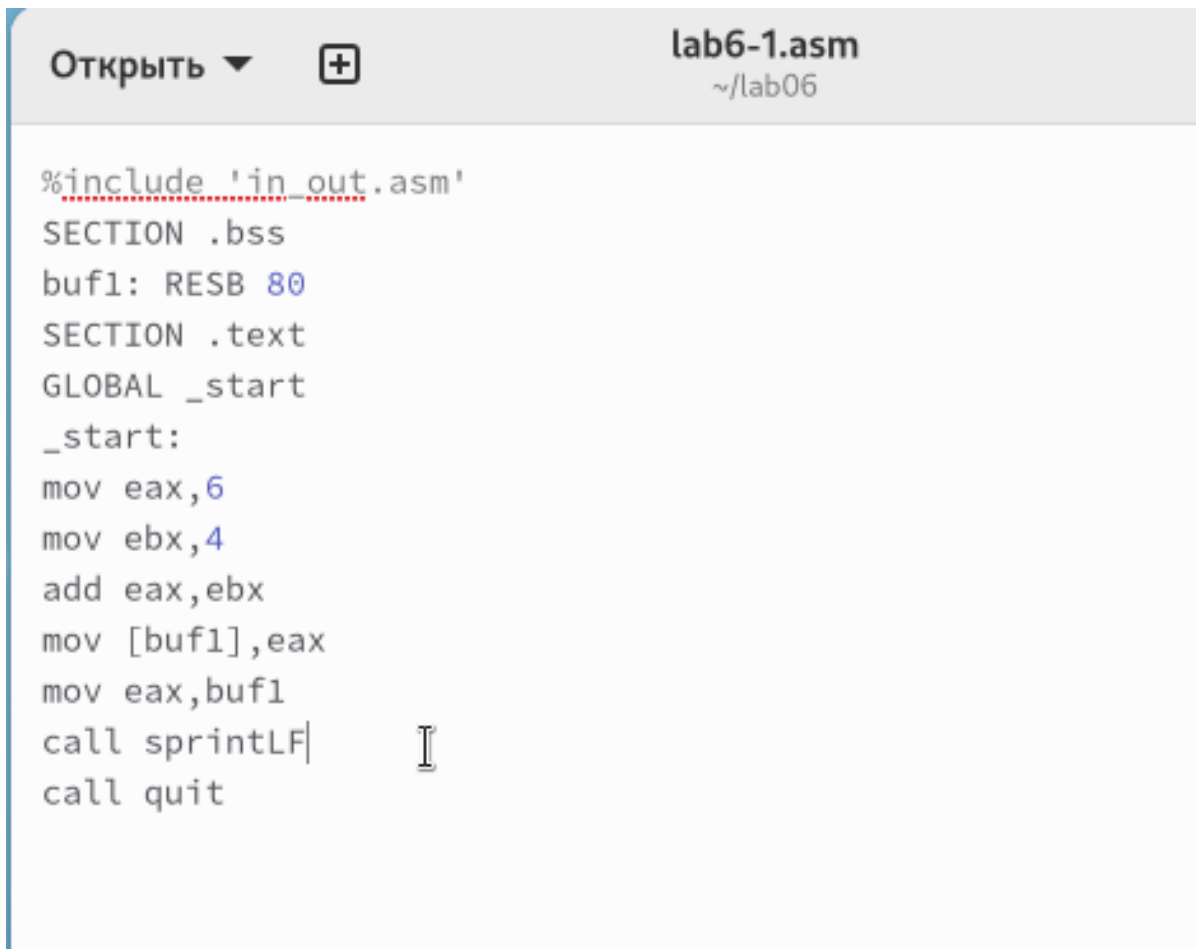
A terminal window with a dark background and light text. The title bar shows the user 'v\_ismagambetov' on a 'fedora' machine in the directory '~/lab06'. The terminal content shows a sequence of commands: 'nasm -f elf lab6-1.asm' which fails with a 'command not found' error, followed by a prompt to install the 'linuxptp' package which is declined. Then, 'nasm -f elf lab6-1.asm' is run successfully, followed by 'ld -m elf\_i386 -o lab6-1 lab6-1.o' to create an executable, and finally './lab6-1' to run it, which produces the output 'j'.

Рис. 4.2: Работа программы

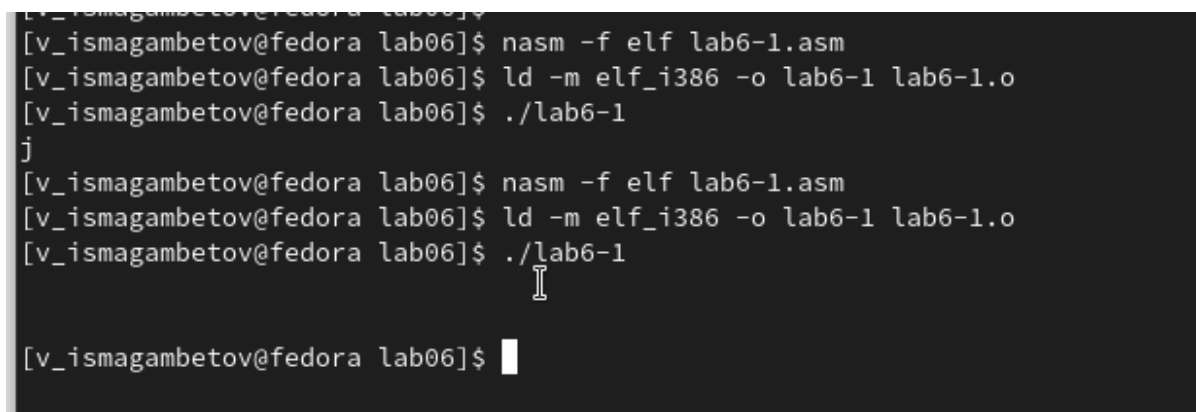
3. Далее изменим текст программы и вместо символов, запишем в регистры числа. Исправьте текст программы (Листинг 1) следующим образом: (рис. 4.3, 4.4)



```
Открыть ▾ + lab6-1.asm
~/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. 4.3: Пример программы

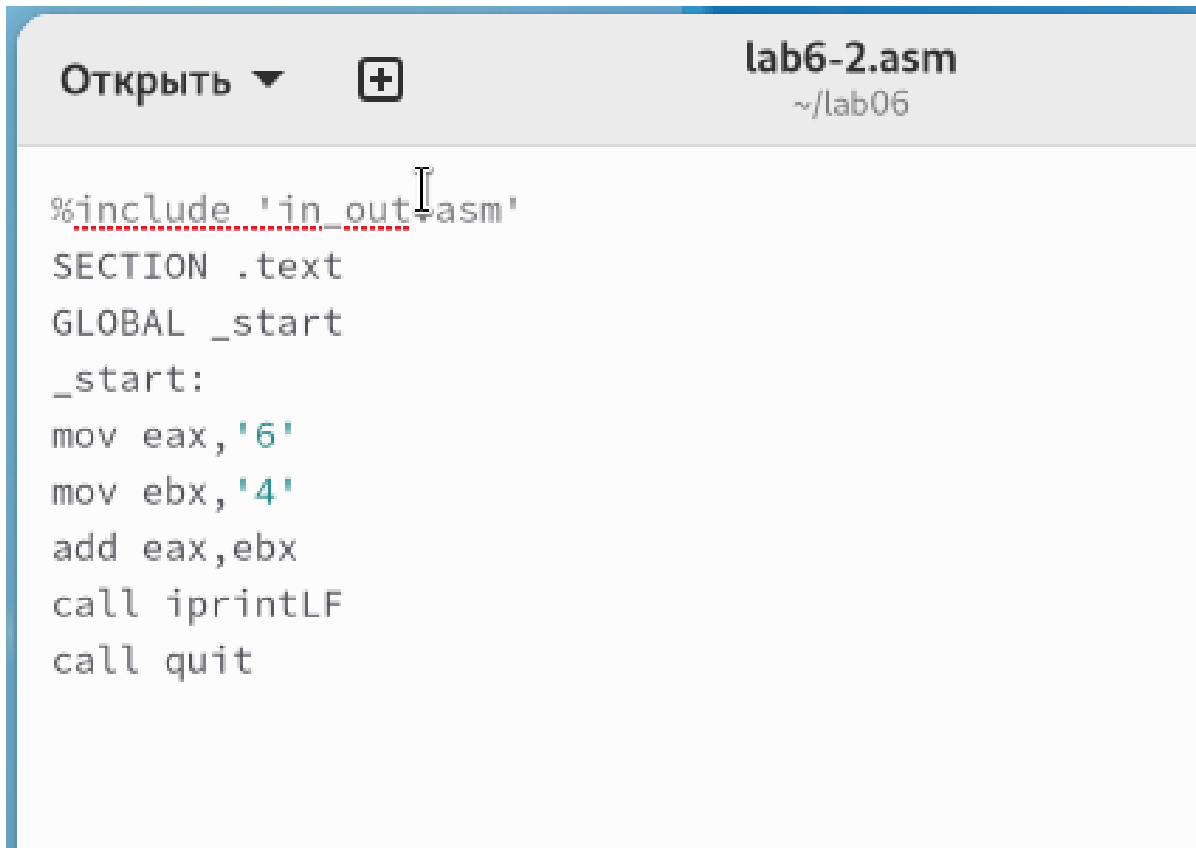


```
[v_ismagambetov@fedora lab06]$ nasm -f elf lab6-1.asm
[v_ismagambetov@fedora lab06]$ ld -m elf_i386 -o lab6-1 lab6-1.o
[v_ismagambetov@fedora lab06]$ ./lab6-1
j
[v_ismagambetov@fedora lab06]$ nasm -f elf lab6-1.asm
[v_ismagambetov@fedora lab06]$ ld -m elf_i386 -o lab6-1 lab6-1.o
[v_ismagambetov@fedora lab06]$ ./lab6-1
j
[v_ismagambetov@fedora lab06]$
```

Рис. 4.4: Работа программы

Никакой символ не виден, но он есть. Это возврат каретки LF.

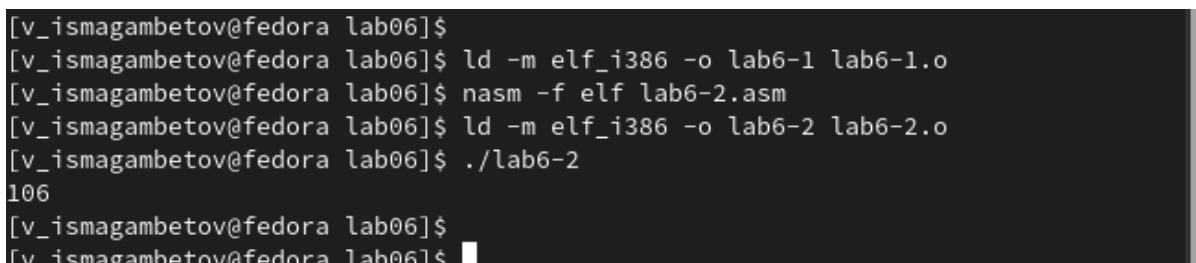
4. Как отмечалось выше, для работы с числами в файле `in_out.asm` реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Преобразуем текст программы из Листинга 7.1 с использованием этих функций. (рис. 4.5, 4.6)



```
Открыть ▼  +  lab6-2.asm
~ /lab06

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

Рис. 4.5: Пример программы



```
[v_ismagambetov@fedora lab06]$
[v_ismagambetov@fedora lab06]$ ld -m elf_i386 -o lab6-1 lab6-1.o
[v_ismagambetov@fedora lab06]$ nasm -f elf lab6-2.asm
[v_ismagambetov@fedora lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[v_ismagambetov@fedora lab06]$ ./lab6-2
106
[v_ismagambetov@fedora lab06]$
[v_ismagambetov@fedora lab06]$
```

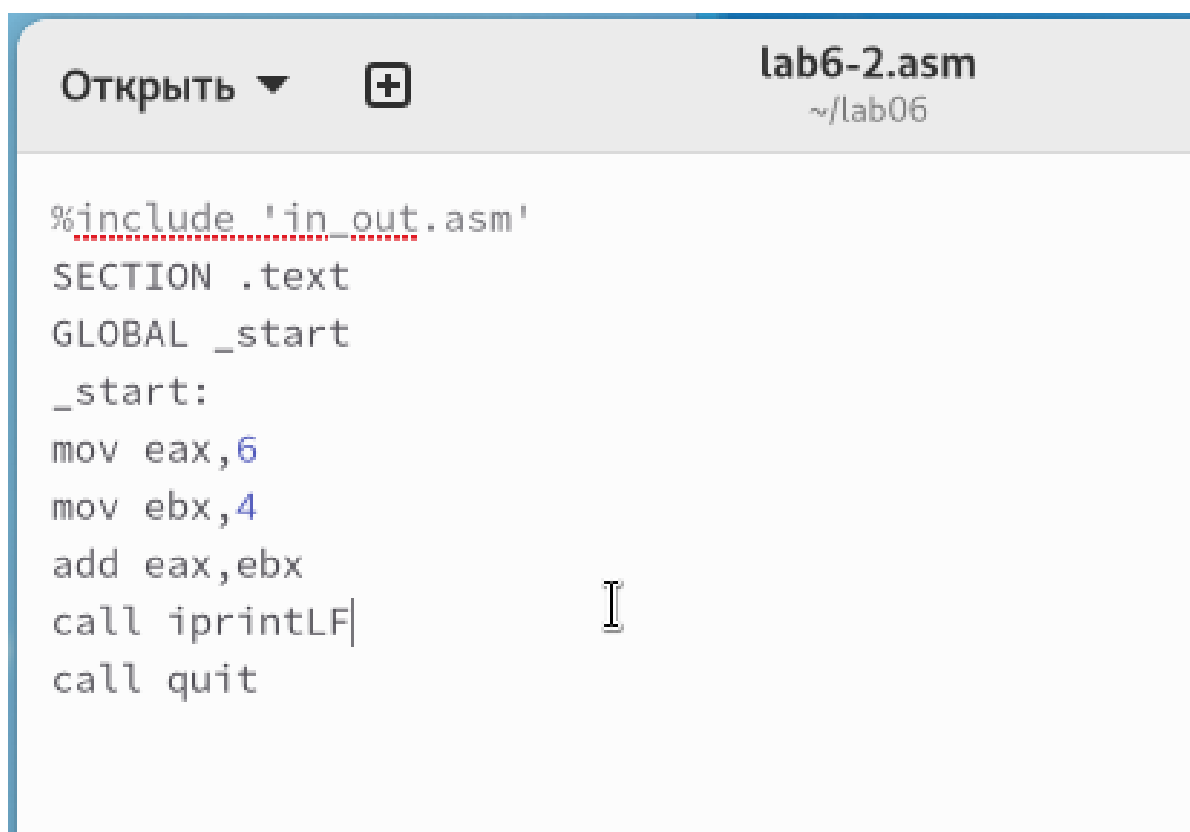
Рис. 4.6: Работа программы

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 7.1, функция `iprintLF` позволяет вывести число, а не символ, кодом которого является это число.

5. Аналогично предыдущему примеру изменим символы на числа. (рис. 4.7, 4.8)

Создайте исполняемый файл и запустите его. Какой результат будет получен при исполнении программы? – получили число 10



```
Открыть ▼  +  lab6-2.asm
~ /lab06

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

Рис. 4.7: Пример программы

```

[v_ismagambetov@fedora lab06]$
[v_ismagambetov@fedora lab06]$ ld -m elf_i386 -o lab6-1 lab6-1.o
[v_ismagambetov@fedora lab06]$ nasm -f elf lab6-2.asm
[v_ismagambetov@fedora lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[v_ismagambetov@fedora lab06]$ ./lab6-2
106
[v_ismagambetov@fedora lab06]$
[v_ismagambetov@fedora lab06]$ nasm -f elf lab6-2.asm
[v_ismagambetov@fedora lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[v_ismagambetov@fedora lab06]$ ./lab6-2
10
[v_ismagambetov@fedora lab06]$
[v_ismagambetov@fedora lab06]$

```

Рис. 4.8: Работа программы

Замените функцию `iprintLF` на `iprint`. Создайте исполняемый файл и запустите его. Чем отличается вывод функций `iprintLF` и `iprint`? - Вывод отличается что нет переноса строки. (рис. 4.9)

```

[v_ismagambetov@fedora lab06]$
[v_ismagambetov@fedora lab06]$ ld -m elf_i386 -o lab6-1 lab6-1.o
[v_ismagambetov@fedora lab06]$ nasm -f elf lab6-2.asm
[v_ismagambetov@fedora lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[v_ismagambetov@fedora lab06]$ ./lab6-2
106
[v_ismagambetov@fedora lab06]$
[v_ismagambetov@fedora lab06]$ nasm -f elf lab6-2.asm
[v_ismagambetov@fedora lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[v_ismagambetov@fedora lab06]$ ./lab6-2
10
[v_ismagambetov@fedora lab06]$
[v_ismagambetov@fedora lab06]$
[v_ismagambetov@fedora lab06]$ nasm -f elf lab6-2.asm
[v_ismagambetov@fedora lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[v_ismagambetov@fedora lab06]$ ./lab6-2
10[v_ismagambetov@fedora lab06]$
[v_ismagambetov@fedora lab06]$

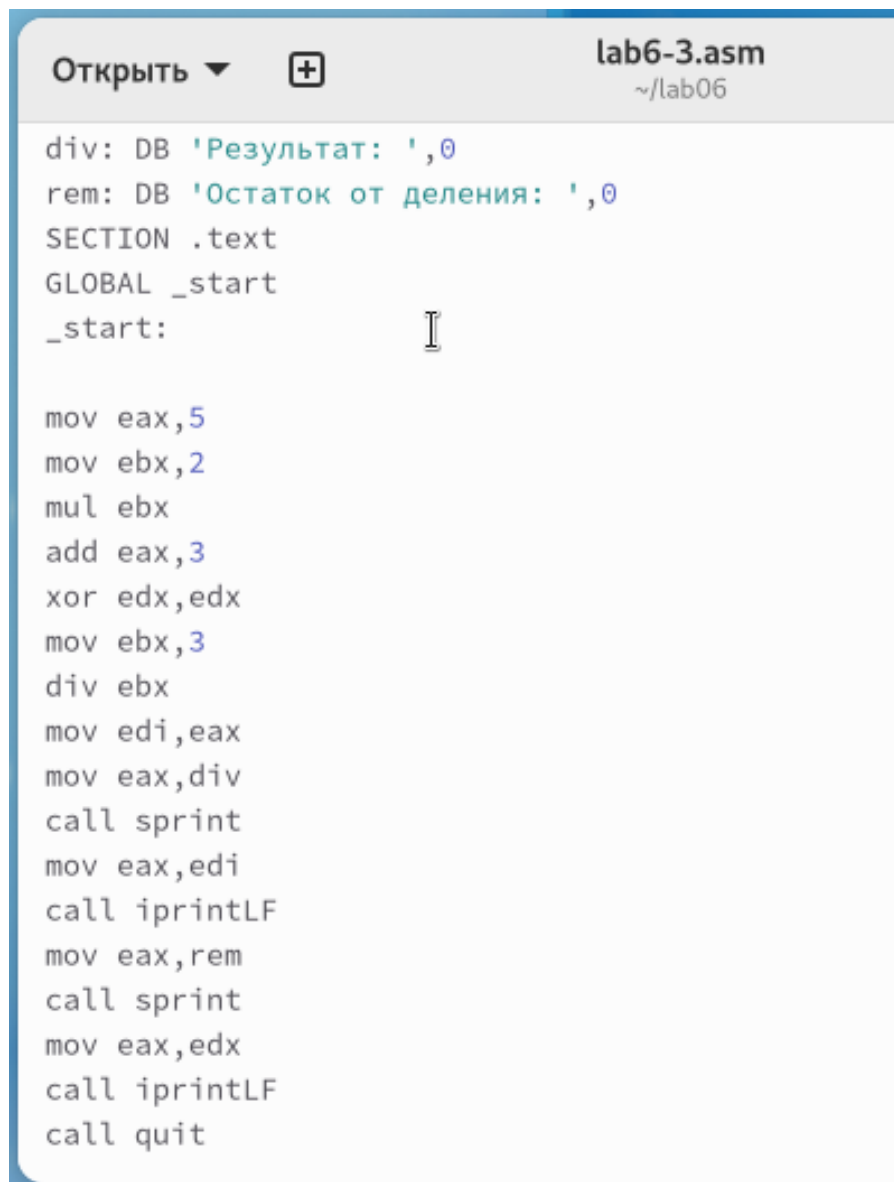
```

Рис. 4.9: Работа программы

- В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения

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

. (рис. 4.10, рис. 4.11)




```
Открыть ▾  lab6-3.asm  
~/lab06  
  
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. 4.10: Пример программы




```
[v_ismagambetov@fedora lab06]$  
[v_ismagambetov@fedora lab06]$  
[v_ismagambetov@fedora lab06]$ nasm -f elf lab6-3.asm  
[v_ismagambetov@fedora lab06]$ ld -m elf_i386 -o lab6-3 lab6-3.o  
[v_ismagambetov@fedora lab06]$ ./lab6-3  
Результат: 4  
Остаток от деления: 1  
[v_ismagambetov@fedora lab06]$  
[v_ismagambetov@fedora lab06]$
```

Рис. 4.11: Работа программы

Измените текст программы для вычисления выражения

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

. Создайте исполняемый файл и проверьте его работу. (рис. 4.12, рис. 4.13)

Открыть ▾  lab6-3.asm  
~/lab06

```
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. 4.12: Пример программы

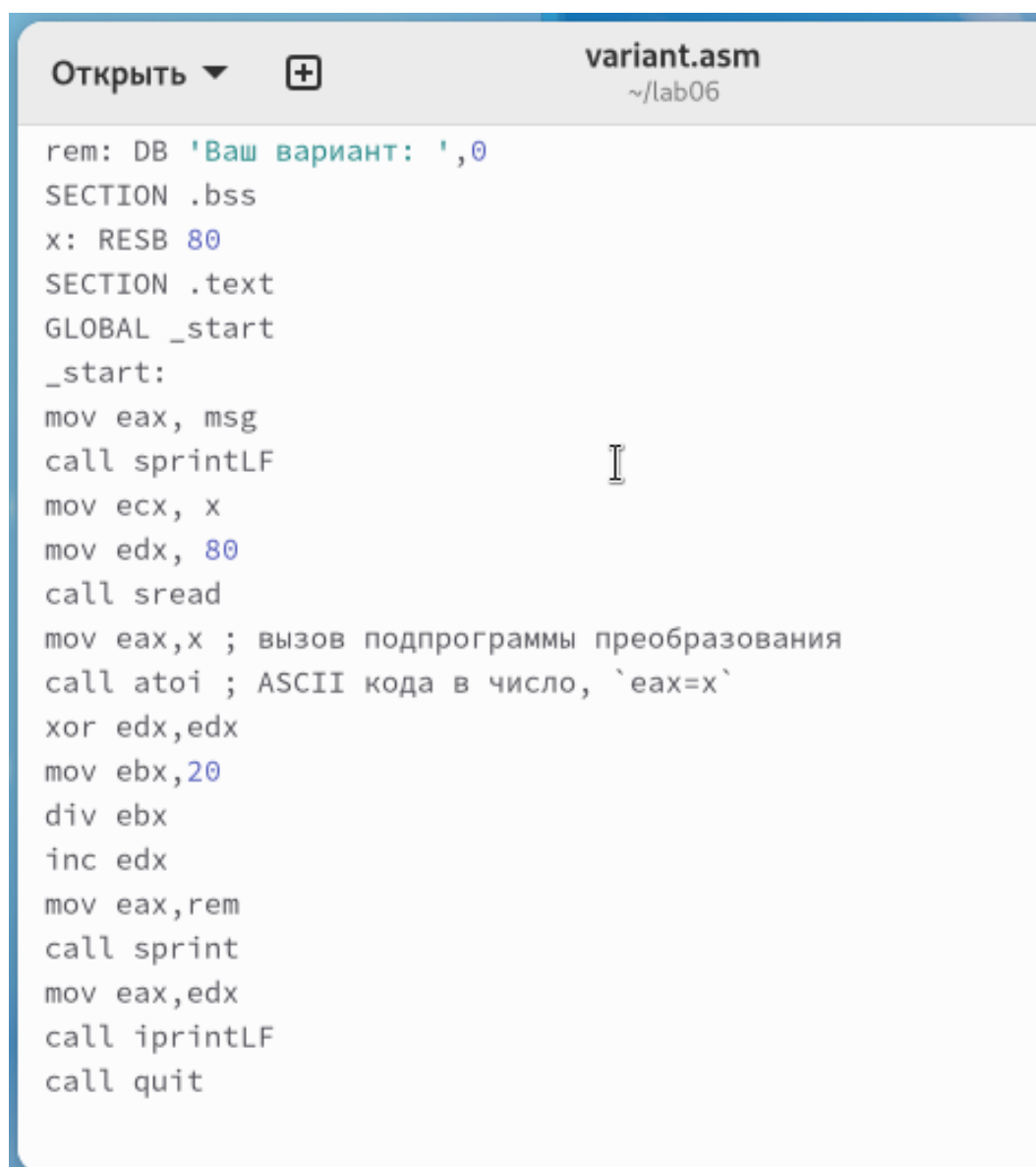
```

[v_ismagambetov@fedora lab06]$
[v_ismagambetov@fedora lab06]$
[v_ismagambetov@fedora lab06]$ nasm -f elf lab6-3.asm
[v_ismagambetov@fedora lab06]$ ld -m elf_i386 -o lab6-3 lab6-3.o
[v_ismagambetov@fedora lab06]$ ./lab6-3
Результат: 4
Остаток от деления: 1
[v_ismagambetov@fedora lab06]$
[v_ismagambetov@fedora lab06]$
[v_ismagambetov@fedora lab06]$ nasm -f elf lab6-3.asm
[v_ismagambetov@fedora lab06]$ ld -m elf_i386 -o lab6-3 lab6-3.o
[v_ismagambetov@fedora lab06]$ ./lab6-3
Результат: 5
Остаток от деления: 1
[v_ismagambetov@fedora lab06]$

```

Рис. 4.13: Работа программы

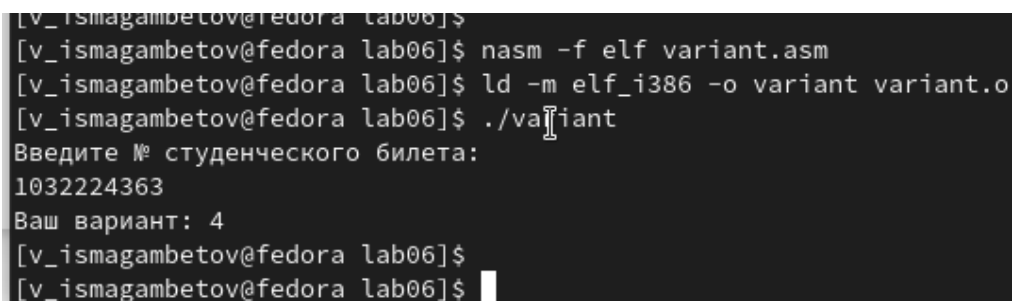
7. В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета, работающую по следующему алгоритму: (рис. 4.14, рис. 4.15)



```
Открыть ▾ + variant.asm
~/lab06

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. 4.14: Пример программы



```
[v_ismagambetov@fedora lab06]$
[v_ismagambetov@fedora lab06]$ nasm -f elf variant.asm
[v_ismagambetov@fedora lab06]$ ld -m elf_i386 -o variant variant.o
[v_ismagambetov@fedora lab06]$ ./variant
Введите № студенческого билета:
1032224363
Ваш вариант: 4
[v_ismagambetov@fedora lab06]$
[v_ismagambetov@fedora lab06]$
```

Рис. 4.15: Работа программы

- Какие строки листинга 7.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’? – `mov eax,rem` – перекладывает в регистр значение переменной с фразой ‘Ваш вариант:’ `call sprint` – вызов подпрограммы вывода строки
- Для чего используются следующие инструкции? `push ecx, x` `mov edx, 80` `call sread`

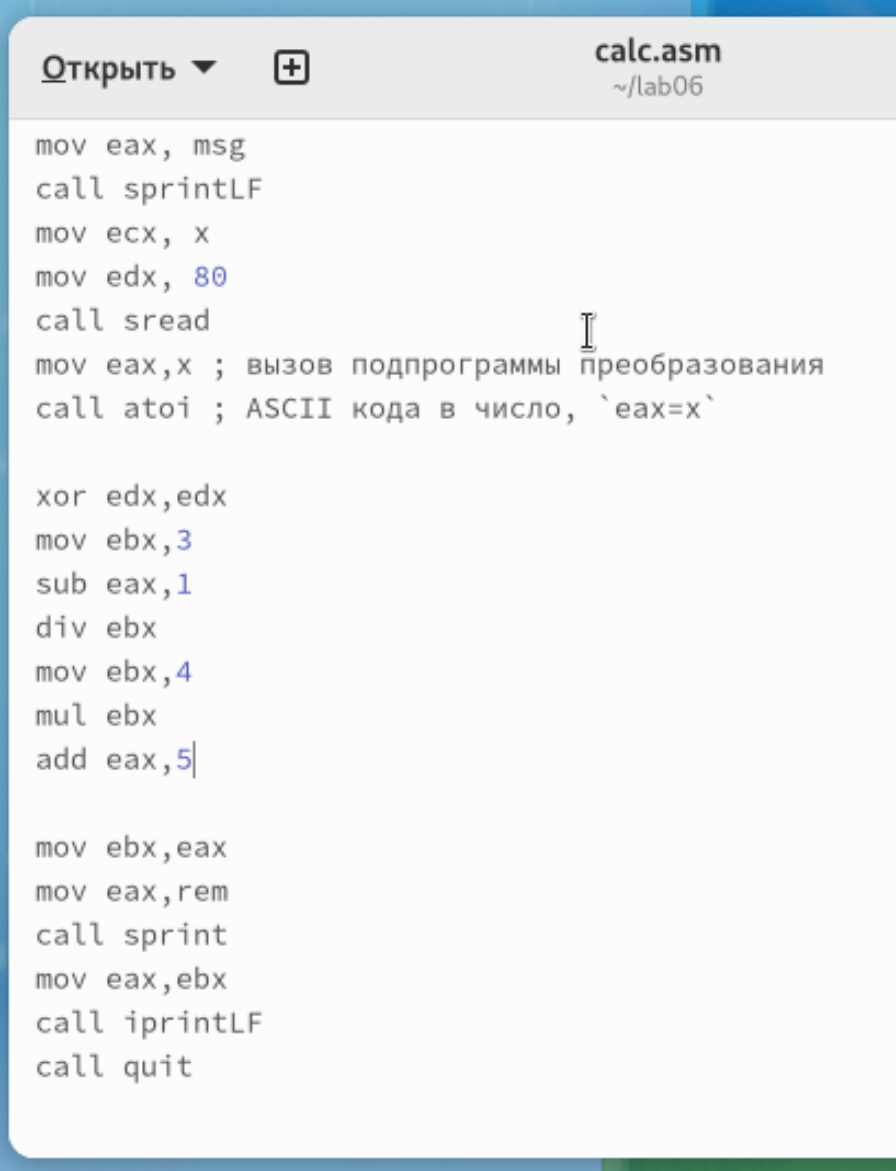
Считывает значение студбилета в переменную X из консоли

- Для чего используется инструкция “`call atoi`”? – эта подпрограмма переводит введенные символы в числовой формат
  - Какие строки листинга 7.4 отвечают за вычисления варианта? `xor edx,edx` `mov ebx,20` `div ebx`
  - В какой регистр записывается остаток от деления при выполнении инструкции “`div ebx`”? 1 байт AH 2 байта DX 4 байта EDX – наш случай
  - Для чего используется инструкция “`inc edx`”? по формуле вычисления варианта нужно прибавить единицу
  - Какие строки листинга 7.4 отвечают за вывод на экран результата вычисления? `mov eax,edx` – результат перекладывается в регистр `eax` `call iprintLF` – вызов подпрограммы вывода
8. 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. (рис. 4.16, рис. 4.17)

Получили вариант 4 -

$$(4/3)(x - 8) + 5$$

для  $x=4$  и 10



```
Открыть ▼  +  calc.asm
~\lab06

mov eax, msg
call sprintfLF
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, 3
sub eax, 1
div ebx
mov ebx, 4
mul ebx
add eax, 5

mov ebx, eax
mov eax, rem
call sprintf
mov eax, ebx
call iprintLF
call quit
```

Рис. 4.16: Пример программы

```
[v_ismagambetov@fedora lab06]$  
[v_ismagambetov@fedora lab06]$ nasm -f elf calc.asm  
[v_ismagambetov@fedora lab06]$ ld -m elf_i386 -o calc calc.o  
[v_ismagambetov@fedora lab06]$ ./calc  
Введите X  
4  
выражение = : 9  
[v_ismagambetov@fedora lab06]$ ./calc  
Введите X  
10  
выражение = : 17  
[v_ismagambetov@fedora lab06]$
```

Рис. 4.17: Работа программы

## **5 Выводы**

Изучили работу с арифметическими операциями



# Список литературы

1. Расширенный ассемблер: NASM
2. MASM, TASM, FASM, NASM под Windows и Linux