

Отчёт по лабораторной работе 6

дисциплина: Архитектура компьютера

Элиана Сулейманова

Содержание

1 Цель работы	5
2 Выполнение лабораторной работы	6
2.1 Символьные и численные данные в NASM	6
2.2 Выполнение арифметических операций в NASM	12
2.3 Ответы на вопросы	16
2.4 Задание для самостоятельной работы	17
3 Выводы	20

Список иллюстраций

2.1 Программа lab6-1.asm	7
2.2 Запуск программы lab6-1.asm	7
2.3 Программа lab6-1.asm с числами	8
2.4 Запуск программы lab6-1.asm с числами	9
2.5 Программа lab6-2.asm	10
2.6 Запуск программы lab6-2.asm	10
2.7 Программа lab6-2.asm с числами	11
2.8 Запуск программы lab6-2.asm с числами	12
2.9 Запуск программы lab6-2.asm без переноса строки	12
2.10 Программа lab6-3.asm	13
2.11 Запуск программы lab6-3.asm	13
2.12 Программа lab6-3.asm с другим выражением	14
2.13 Запуск программы lab6-3.asm с другим выражением	14
2.14 Программа variant.asm	15
2.15 Запуск программы variant.asm	16
2.16 Программа calc.asm	18
2.17 Запуск программы calc.asm	19

Список таблиц

1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

2 Выполнение лабораторной работы

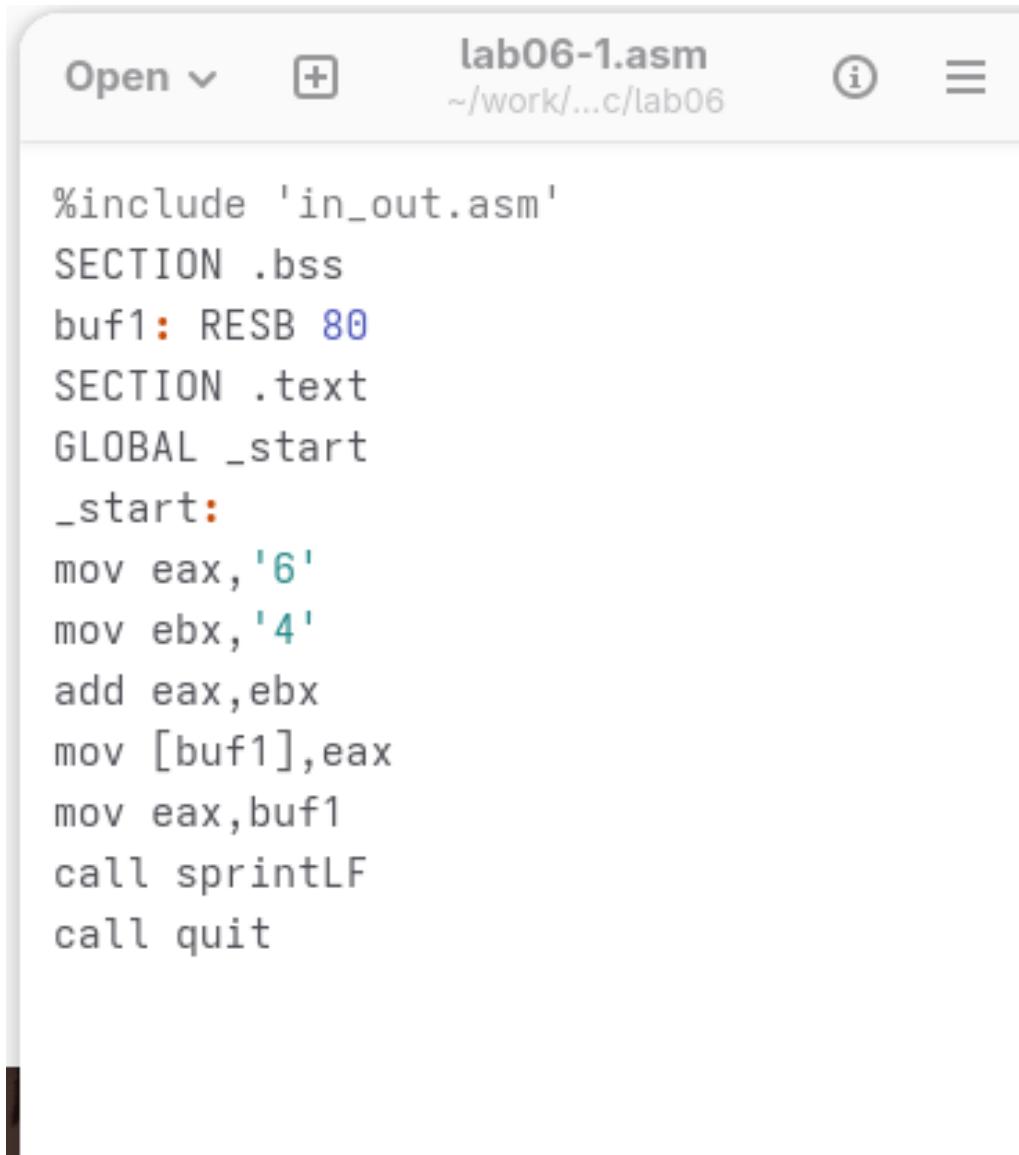
2.1 Символьные и численные данные в NASM

Я создаю каталог для программ лабораторной работы № 6, перехожу в него и создаю файл lab6-1.asm.

Программы, которые будут приведены далее, демонстрируют вывод символьных и численных значений, записанных в регистр eax.

В первой программе в регистр eax записывается символ 6 с помощью команды `mov eax, '6'`, а в регистр ebx – символ 4 с помощью команды `mov ebx, '4'`. Далее, к значению в регистре eax прибавляется значение регистра ebx командой `add eax, ebx`. Результат сложения записывается в регистр eax. После этого выводится результат.

Так как функция `sprintLF` требует, чтобы в регистр eax был записан адрес, используется дополнительная переменная. Для этого записываю значение из регистра eax в переменную `buf1` командой `mov [buf1], eax`, затем записываю адрес переменной `buf1` в регистр eax командой `mov eax, buf1` и вызываю функцию `sprintLF`.



The screenshot shows a code editor window with the following interface elements:

- Top bar: "Open ▾" (dropdown), "+" (new file), "lab06-1.asm" (current file), " ~/work/...c/lab06" (file path), "i" (info icon), and "≡" (menu icon).
- Code area:

```
%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 sprintLF
    call quit
```

Рисунок 2.1: Программа lab6-1.asm

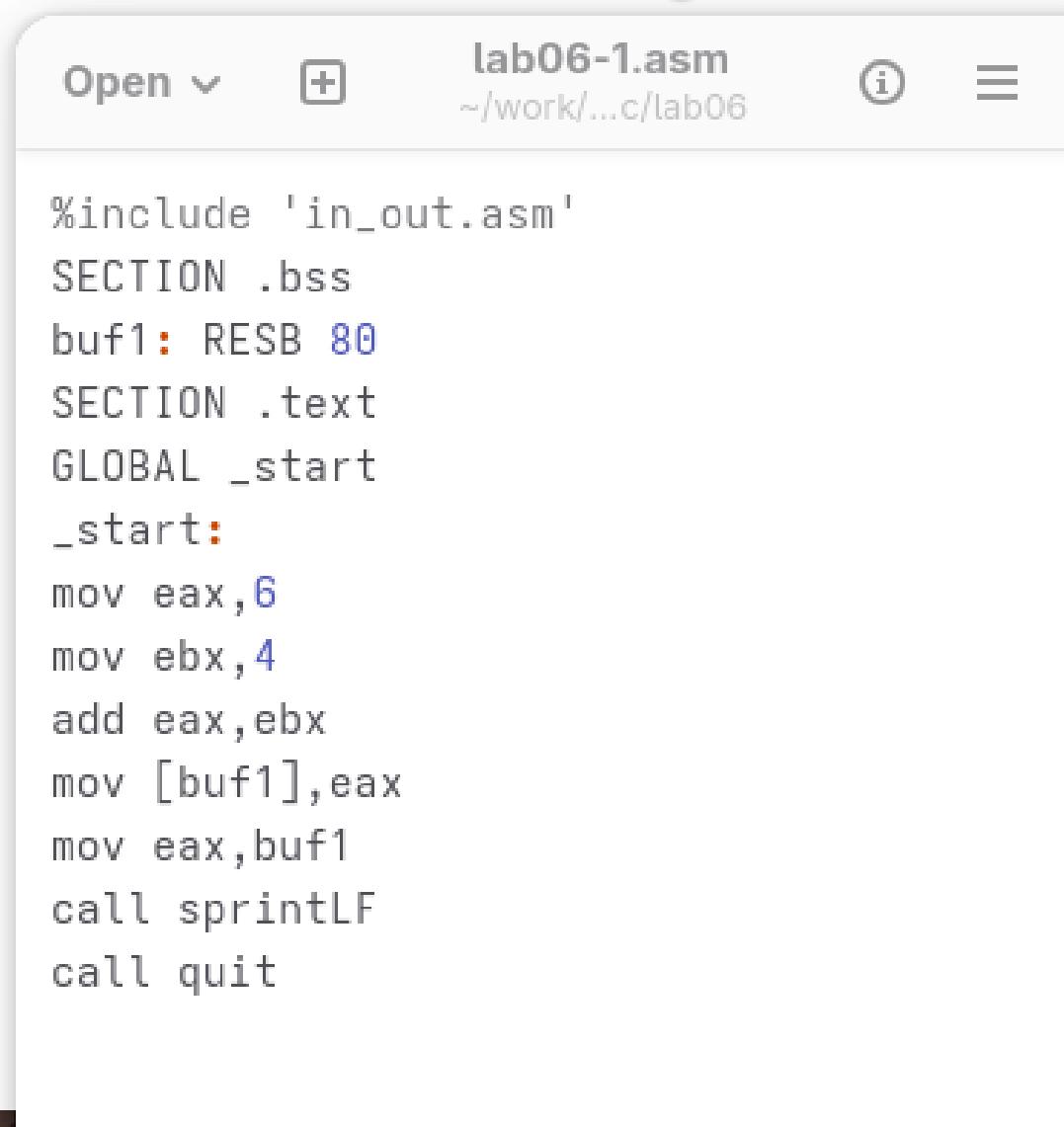
```
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ nasm -f elf lab06-1.asm
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-1.o -o lab06-1
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ ./lab06-1
j
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ █
```

Рисунок 2.2: Запуск программы lab6-1.asm

При выводе значения из регистра eax мы ожидаем увидеть число 10. Одна-

ко результатом будет символ j. Это происходит потому, что код символа 6 равен 54 в десятичной системе (или 00110110 в двоичной), а код символа 4 – 52 в десятичной системе (или 00110100 в двоичной). После выполнения команды add eax, ebx, в регистр eax записывается сумма кодов, равная 106, что соответствует символу j.

Далее, в программе вместо символов записываю в регистры числа.



The screenshot shows a text editor window with the following details:

- File menu: Open ▾
- New file icon: +
- File name: lab06-1.asm
- File path: ~/work/...c/lab06
- Information icon: ⓘ
- More options icon: Ⓜ

The assembly code listed is:

```
%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 sprintLF  
    call quit
```

Рисунок 2.3: Программа lab6-1.asm с числами

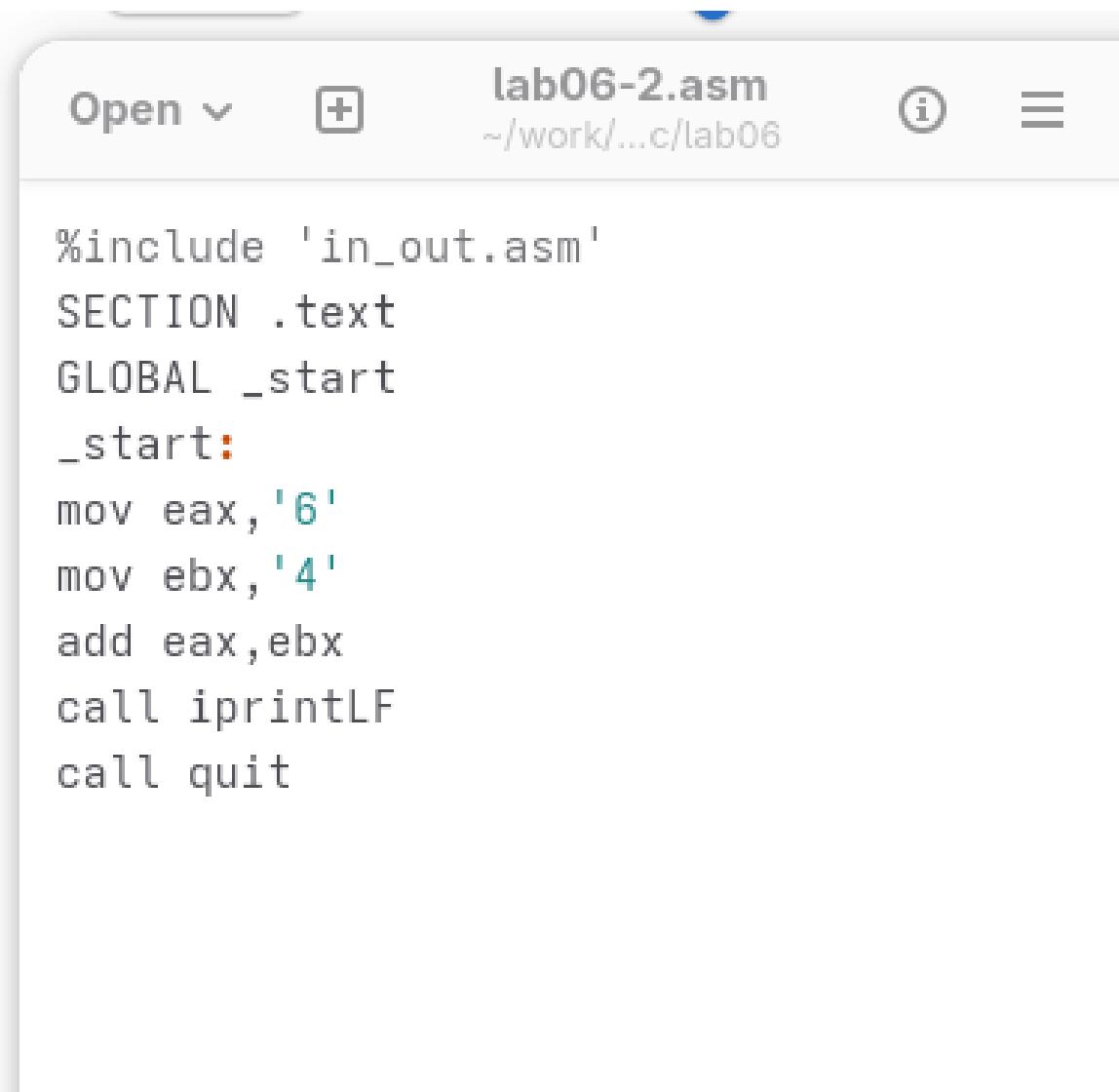
```
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ nasm -f elf lab06-1.asm
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-1.o -o lab06-1
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ ./lab06-1
```

```
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$
```

Рисунок 2.4: Запуск программы lab6-1.asm с числами

Как и в предыдущем примере, выводится не число 10, а символ с кодом 10, который представляет собой символ конца строки. Этот символ не отображается в консоли, но добавляет пустую строку.

Для работы с числами в файле `in_out.asm` реализованы функции для преобразования символов ASCII в числа и наоборот. Преобразую программу с использованием этих функций.



The screenshot shows a text editor window with the following interface elements:

- Top left: "Open" dropdown and "+" button.
- Top center: File name "lab06-2.asm" and path "~/work/...c/lab06".
- Top right: Help icon (i) and menu icon (three horizontal lines).

The code area contains the following assembly code:

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

Рисунок 2.5: Программа lab6-2.asm



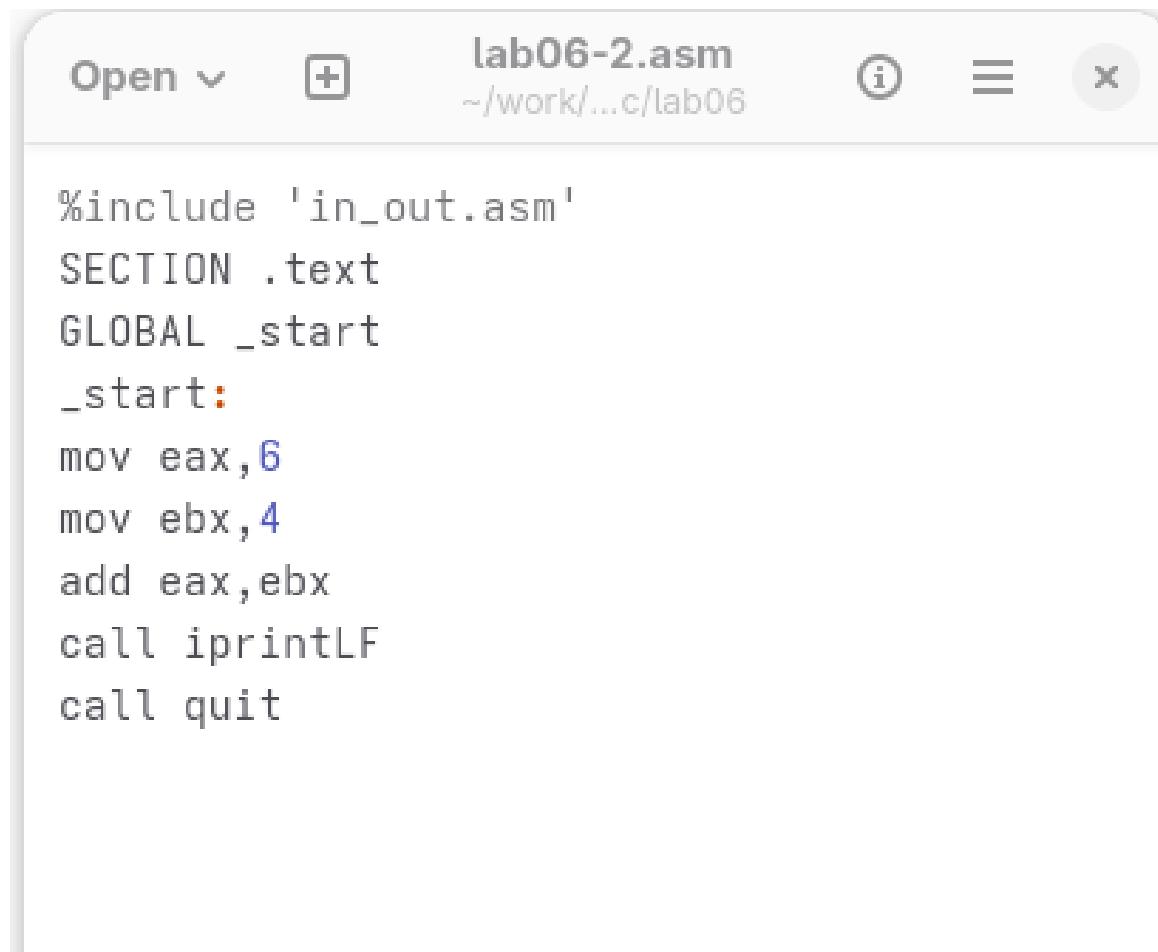
```
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ nasm -f elf lab06-2.asm
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-2.o -o lab06-2
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ ./lab06-2
106
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ █
```

Рисунок 2.6: Запуск программы lab6-2.asm

Результатом работы программы будет число 106. Здесь, как и в первом при-

мере, команда add складывает коды символов 6 и 4 ($54 + 52 = 106$). Однако, в отличие от предыдущей программы, функция iprintLF позволяет вывести именно число, а не символ, соответствующий данному коду.

Заменяю символы на числа.



The screenshot shows a text editor window with the following details:

- File menu: Open ▾
- Add file button: +
- File name: lab06-2.asm
- File path: ~/work/...c/lab06
- Help icon: ⓘ
- Mode icon: ⚒
- Close icon: ✘

The assembly code in the editor is:

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

Рисунок 2.7: Программа lab6-2.asm с числами

В данном случае, благодаря функции iprintLF, выводится число 10, так как операндами являются числа.

```
ersuleimanova@virtualbox:~/work/arch-pc/lab06$ 
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ nasm -f elf lab06-2.asm
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-2.o -o lab06-2
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ ./lab06-2
10
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ █
```

Рисунок 2.8: Запуск программы lab6-2.asm с числами

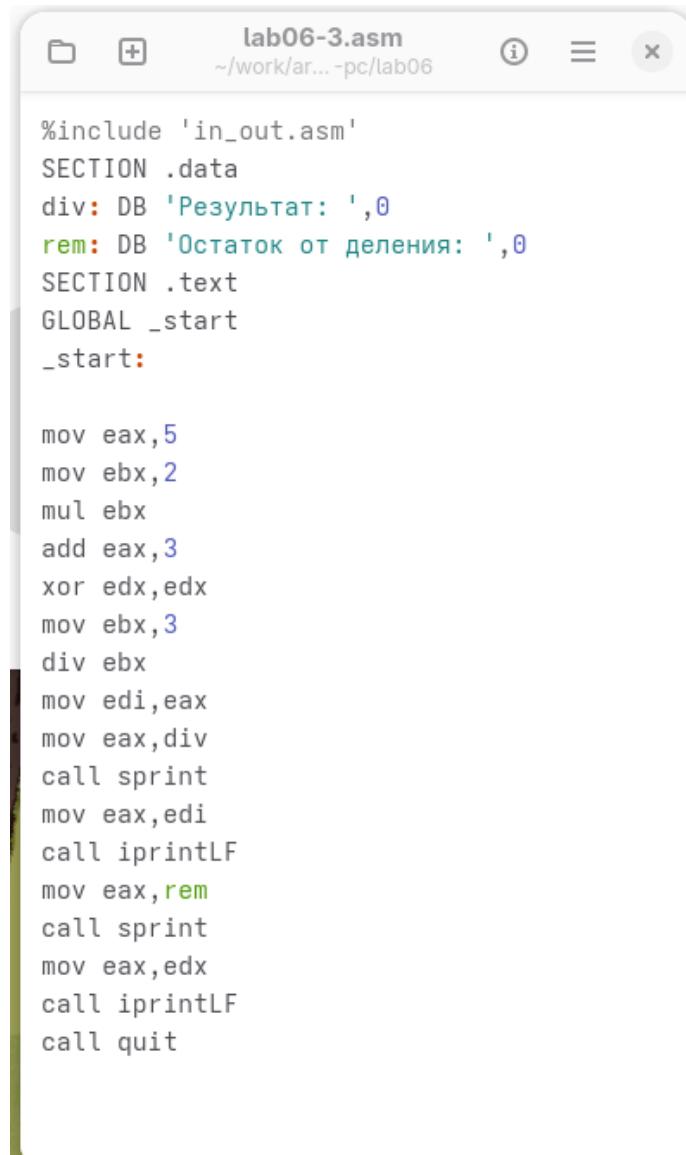
Заменяю функцию iprintLF на iprint и создаю исполняемый файл, затем запускаю программу. Вывод отличается тем, что теперь нет переноса строки.

```
ersuleimanova@virtualbox:~/work/arch-pc/lab06$ 
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ nasm -f elf lab06-2.asm
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-2.o -o lab06-2
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ ./lab06-2
10ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ 
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ █
```

Рисунок 2.9: Запуск программы lab6-2.asm без переноса строки

2.2 Выполнение арифметических операций в NASM

Примером арифметических операций в NASM будет программа для вычисления выражения $f(x) = (5 * 2 + 3)/3$.



```
lab06-3.asm
~/work/ar...-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,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
```

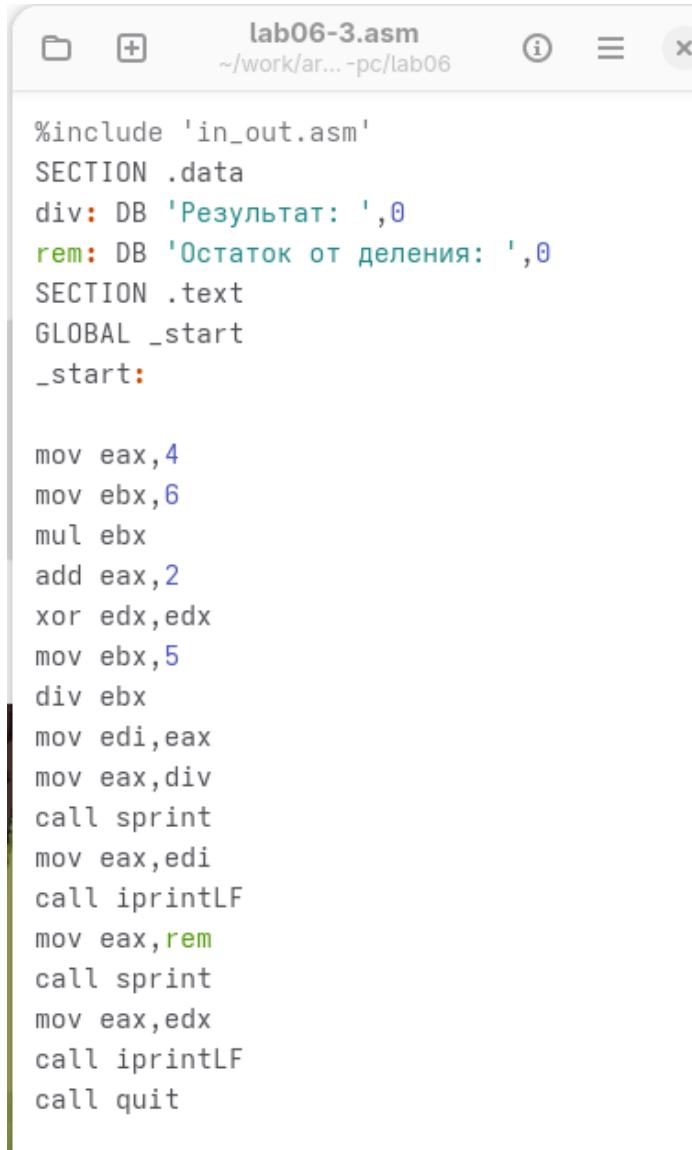
Рисунок 2.10: Программа lab6-3.asm

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

Рисунок 2.11: Запуск программы lab6-3.asm

Изменяю программу для вычисления выражения $f(x) = (4 * 6 + 2)/5$. Со-

здаю исполняемый файл и проверяю его работу.

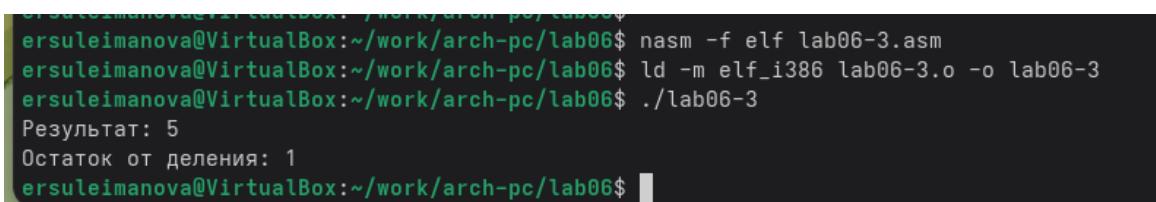


The screenshot shows a text editor window titled "lab06-3.asm" with the file path "~/work/arch-pc/lab06". The code is written in NASM assembly language. It includes sections for data and text, defines variables for result and remainder, and performs division of 4 by 6, printing both the result and remainder. The assembly code uses standard NASM syntax with labels like _start, div, rem, and various mov, add, xor, div, call, and mov instructions.

```
%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
```

Рисунок 2.12: Программа lab6-3.asm с другим выражением



The screenshot shows a terminal window with a black background and white text. It displays the command-line steps to build and run the assembly program: "nasm -f elf lab06-3.asm", "ld -m elf_i386 lab06-3.o -o lab06-3", and "./lab06-3". The output of the program is shown in red text: "Результат: 5" and "Остаток от деления: 1". The user's name "ersuleimanova" and the host "VirtualBox" are visible at the top of the terminal window.

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

Рисунок 2.13: Запуск программы lab6-3.asm с другим выражением

Другим примером будет программа для вычисления варианта задания по номеру студенческого билета. В этом случае число, с которым производятся арифметические операции, вводится с клавиатуры. Для корректной работы с числами, введенные символы необходимо преобразовать в числовый формат, для чего используется функция atoi из файла in_out.asm.



The screenshot shows a text editor window with the file 'variant.asm' open. The code is written in assembly language and includes comments in Russian. The code reads a student number from the user, converts it to an integer using the atoi function, performs some arithmetic operations (addition and division), and then prints the result back to the screen. The assembly code uses standard conventions like .data, .bss, .text sections, and various CPU instructions.

```
%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 sprintLF
    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 sprint
    mov eax,edx
    call iprintLF
    call quit
```

Рисунок 2.14: Программа variant.asm

```
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ nasm -f elf variant.asm
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ ld -m elf_i386 variant.o -o variant
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1132250423
Ваш вариант: 4
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$
```

Рисунок 2.15: Запуск программы variant.asm

2.3 Ответы на вопросы

1. Какие строки листинга отвечают за вывод на экран сообщения „Ваш вариант:“?

- Инструкция `mov eax, %em` записывает значение переменной с фразой „Ваш вариант:“ в регистр eax.
- Инструкция `call sprint` вызывает подпрограмму для вывода строки.

2. Для чего используются следующие инструкции?

- `mov ecx, x` — записывает значение переменной x в регистр ecx.
- `mov edx, 80` — записывает значение 80 в регистр edx.
- `call sread` — вызывает подпрограмму для считывания значения студенческого билета.

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?

- `xor edx, edx` — обнуляет регистр edx.
- `mov ebx, 20` — записывает значение 20 в регистр ebx.

- `div ebx` – выполняет деление номера студенческого билета на 20.
- `inc edx` – увеличивает значение в регистре `edx` на 1.

Здесь происходит деление номера студенческого билета на 20. Остаток записывается в регистр `edx`, к которому добавляется 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`» увеличивает значение в регистре `edx` на 1, согласно формуле вычисления варианта.

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

- Инструкция `mov eax, edx` записывает результат в регистр `eax`.
- Инструкция `call iprintLF` вызывает подпрограмму для вывода значения на экран.

2.4 Задание для самостоятельной работы

Необходимо написать программу для вычисления выражения $y = f(x)$, которая должна выводить выражение для вычисления, запросить ввод значения x , вычислить выражение в зависимости от введенного значения и вывести результат. Вид функции $f(x)$ должен быть выбран согласно таблице 6.3 вариантов заданий.

Мы получили вариант 4 для выражения $4/3(x - 1) + 5$ с $x=4, x=10$.

The screenshot shows a text editor window titled "calc.asm" with the file path "~/work/arch-pc/lab06". The code is written in assembly language:

```
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите X ',0
rem: DB 'выражение = : ',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
    call atoi
    sub eax,1
    mov ebx,4
    mul ebx
    mov ebx,3
    xor edx,edx
    div ebx
    add eax,5
    mov ebx,eax
    mov eax,rem
    call sprint
    mov eax,ebx
    call iprintLF
    call quit
```

Рисунок 2.16: Программа calc.asm

При \$ x=4 \$ результат равен 9.

При \$ x=10 \$ результат равен 17.

```
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ nasm -f elf calc.asm
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ ld -m elf_i386 calc.o -o calc
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ ./calc
Введите X
4
выражение = : 9
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ ./calc
Введите X
10
выражение = : 17
ersuleimanova@VirtualBox:~/work/arch-pc/lab06$ █
```

Рисунок 2.17: Запуск программы calc.asm

Программа работает правильно.

3 Выводы

Изучили работу с арифметическими операциями.