

Отчёт по лабораторной работе 6

Архитектура компьютеров

Сюй Хайфэн

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	11
2.9	Запуск программы lab6-2.asm без переноса строки	11
2.10	Программа lab6-3.asm	12
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 task.asm	18
2.17	Запуск программы task.asm	19

Список таблиц

1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

2 Выполнение лабораторной работы

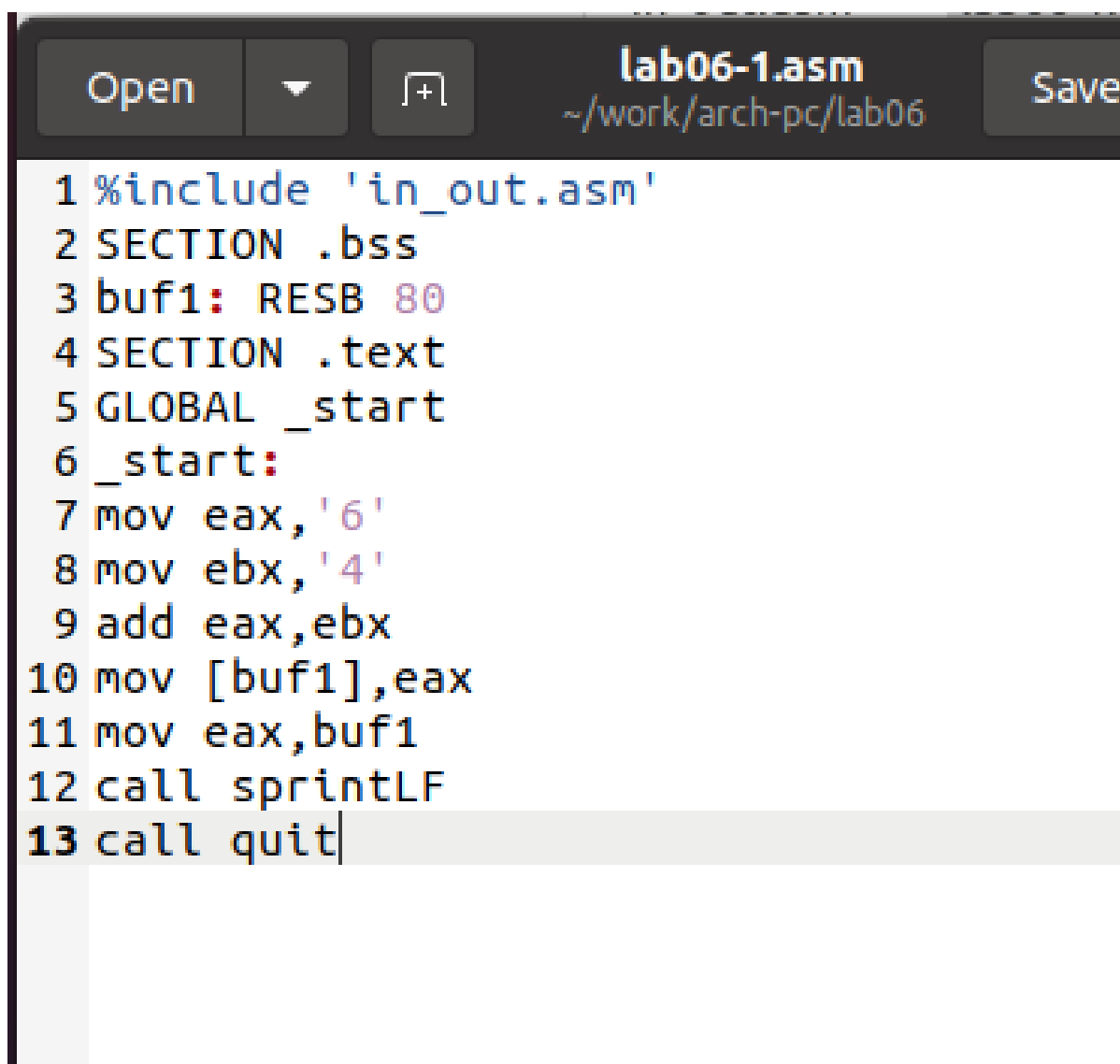
2.1 Символьные и численные данные в NASM

Я создал каталог для программ лабораторной работы № 6, перешел в него и создал файл `lab6-1.asm`.

В этом разделе рассмотрим примеры программ, которые выводят символьные и численные значения. Программы будут выводить данные, записанные в регистр `eax`.

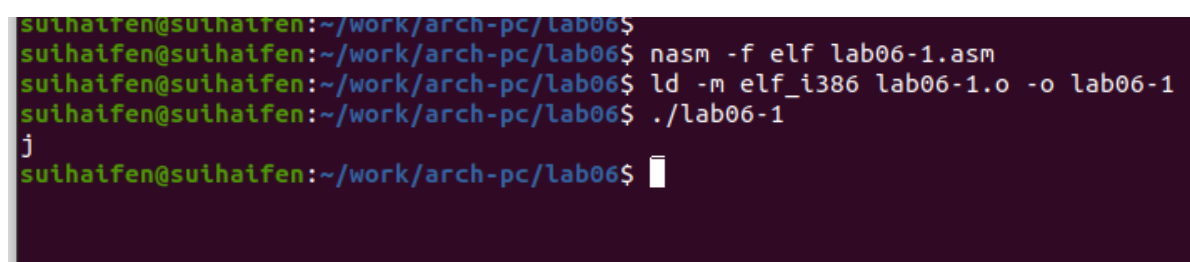
В данной программе в регистр `eax` записывается символ '6' (с помощью команды `mov eax, '6'`), в регистр `ebx` записывается символ '4' (с помощью команды `mov ebx, '4'`). Затем к значению в регистре `eax` прибавляется значение из регистра `ebx` (командой `add eax, ebx`), и результат сохраняется в `eax`. После этого выводим результат.

Так как для работы функции `sprintf` в регистр `eax` должен быть записан адрес, создаем дополнительную переменную. Сначала записываем значение из регистра `eax` в переменную `buf1` (команда `mov [buf1], eax`), затем записываем адрес этой переменной в регистр `eax` (команда `mov eax, buf1`) и вызываем функцию `sprintf`.



```
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 sprintLF
13 call quit
```

Рис. 2.1: Программа lab6-1.asm



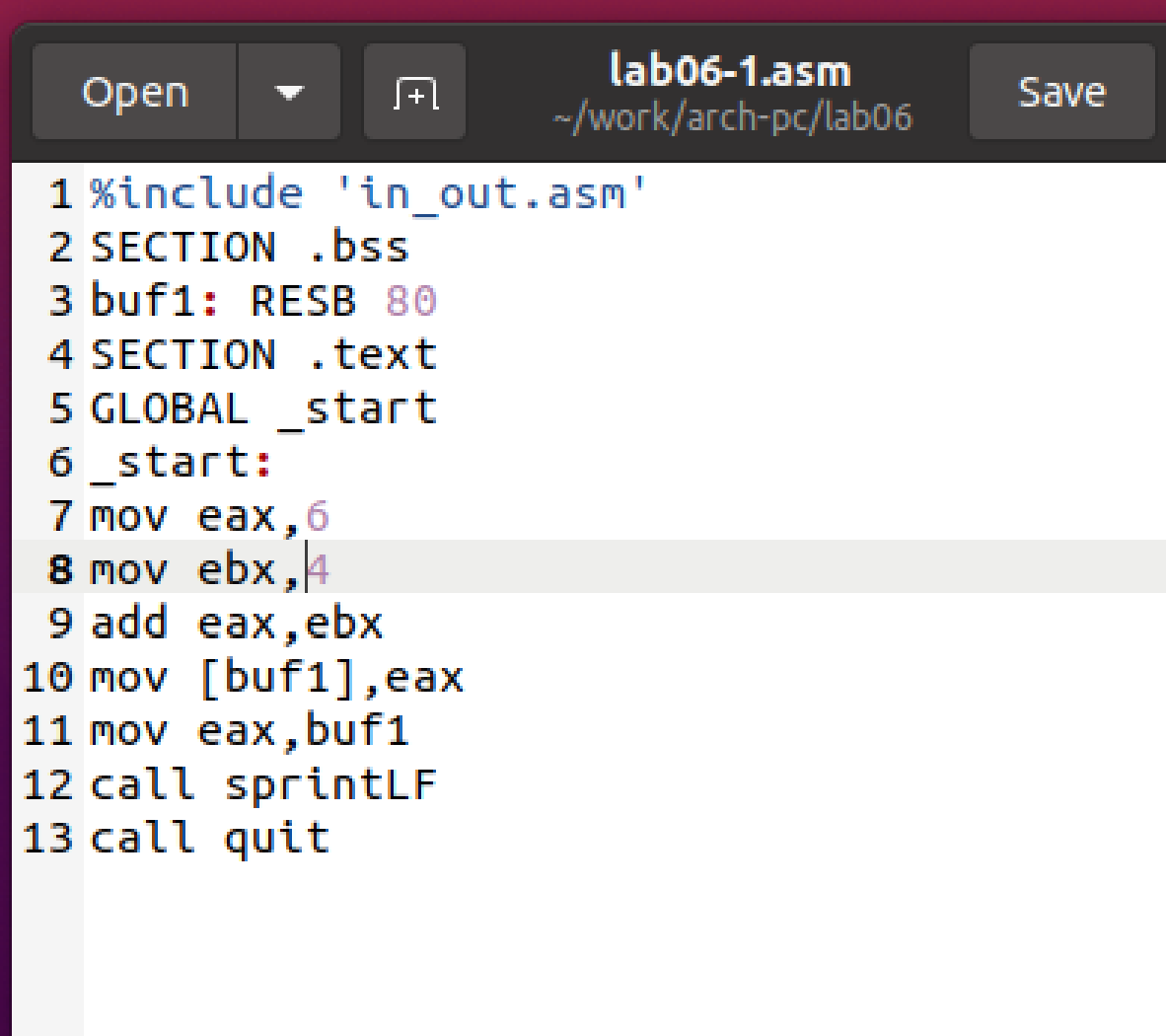
```
suihaifen@suihaifen:~/work/arch-pc/lab06$
suihaifen@suihaifen:~/work/arch-pc/lab06$ nasm -f elf lab06-1.asm
suihaifen@suihaifen:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-1.o -o lab06-1
suihaifen@suihaifen:~/work/arch-pc/lab06$ ./lab06-1
j
suihaifen@suihaifen:~/work/arch-pc/lab06$
```

Рис. 2.2: Запуск программы lab6-1.asm

При выводе значения из регистра `eax` мы ожидали увидеть число 10, но на

самом деле вывелся символ 'j'. Это связано с тем, что код символа '6' равен 00110110 (54 в десятичной системе), а код символа '4' — 00110100 (52). Когда эти значения сложились (с помощью команды `add eax, ebx`), результатом стало значение 106, что в свою очередь соответствует символу 'j' в таблице ASCII.

Затем я изменил программу, заменив символы на числа.



```
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 sprintLF
13 call quit
```

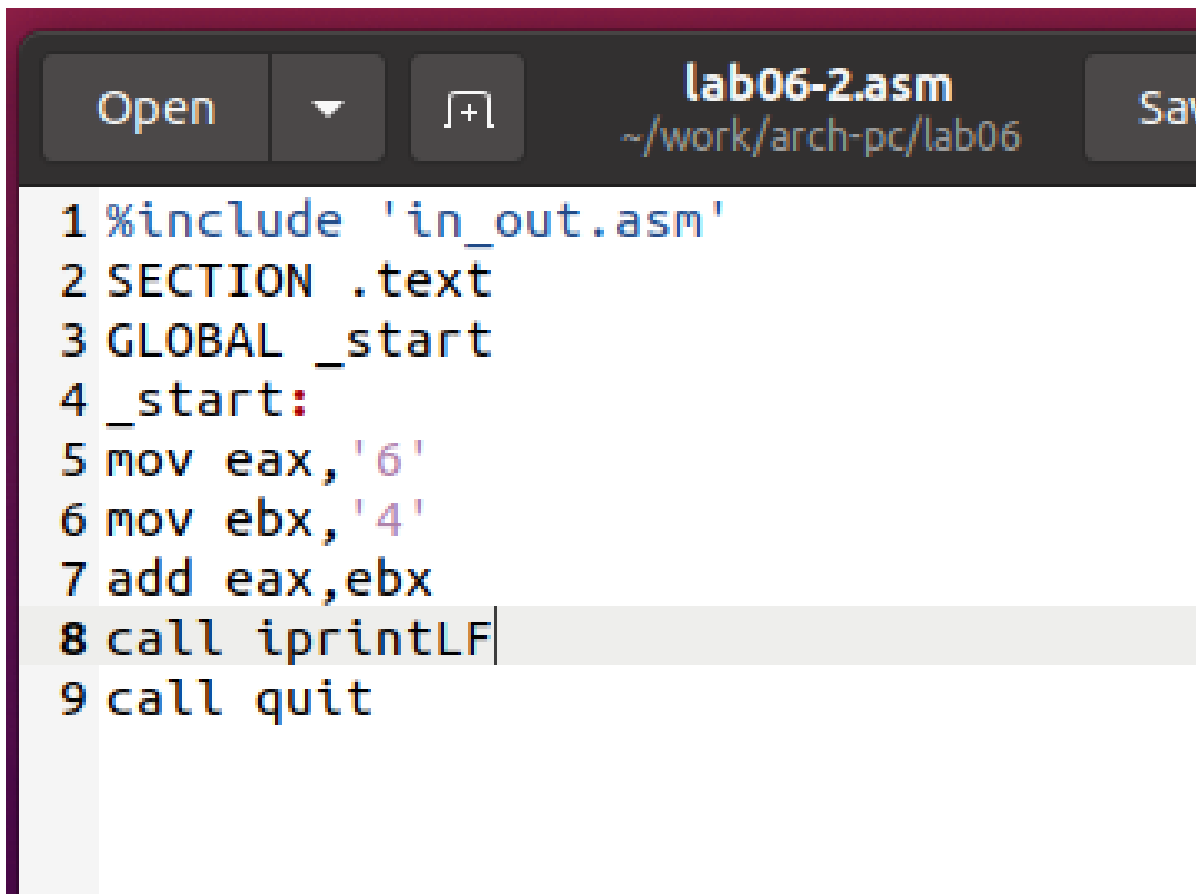
Рис. 2.3: Программа lab6-1.asm с числами


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

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

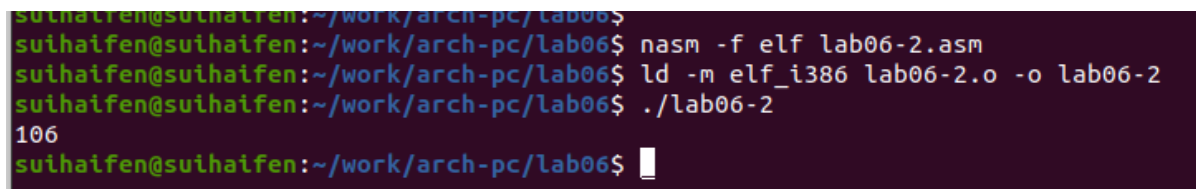
Как и в предыдущем примере, при исполнении программы мы не получим число 10, а на экране появится символ с кодом 10, который представляет собой символ конца строки (возврат каретки). В консоли он не отображается, но добавляет новую строку.

Для работы с числами в файле `in_out.asm` предусмотрены подпрограммы для преобразования символов в числа и наоборот. Я преобразовал программу, используя эти функции.



```
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. 2.5: Программа lab6-2.asm

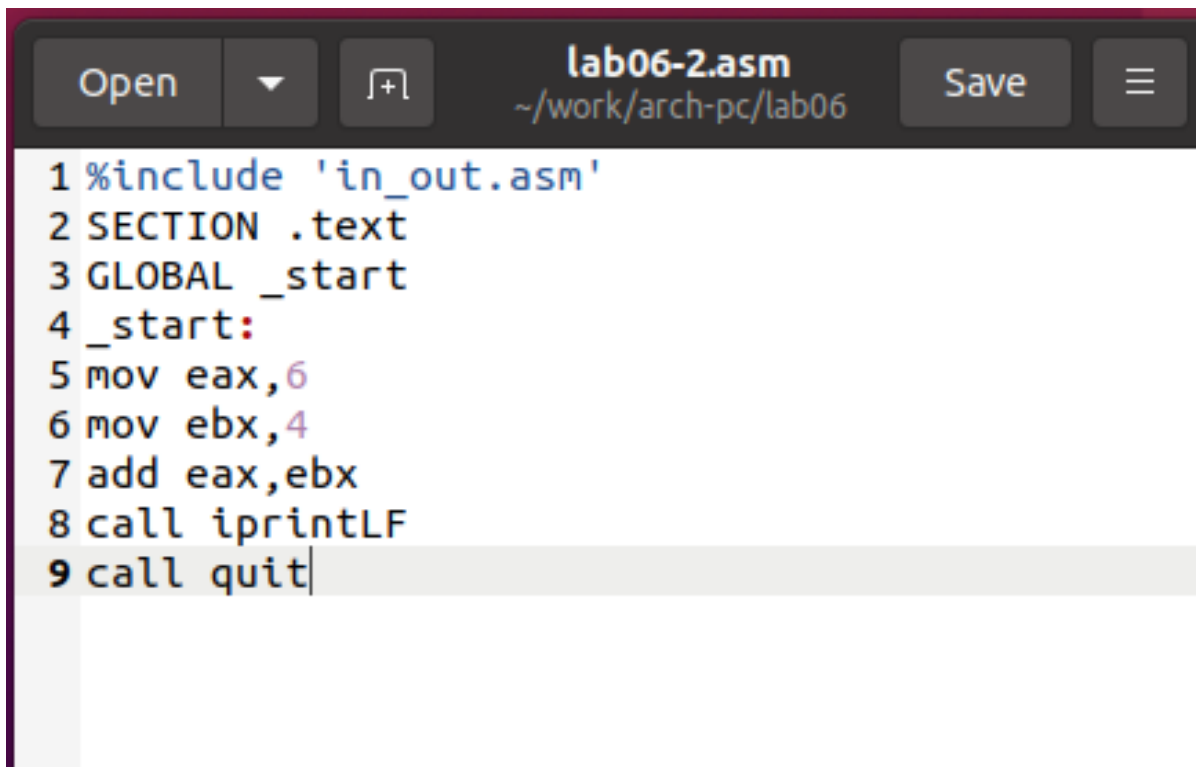


```
suihaifen@suihaifen:~/work/arch-pc/lab06$ nasm -f elf lab06-2.asm
suihaifen@suihaifen:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-2.o -o lab06-2
suihaifen@suihaifen:~/work/arch-pc/lab06$ ./lab06-2
106
suihaifen@suihaifen:~/work/arch-pc/lab06$
```

Рис. 2.6: Запуск программы lab6-2.asm

Результатом выполнения программы стало число 106. Здесь, как и в предыдущем примере, команда `add` складывает коды символов '6' и '4' ($54 + 52 = 106$). Однако теперь функция `iprintLF` позволяет вывести это число, а не символ, код которого равен 106.

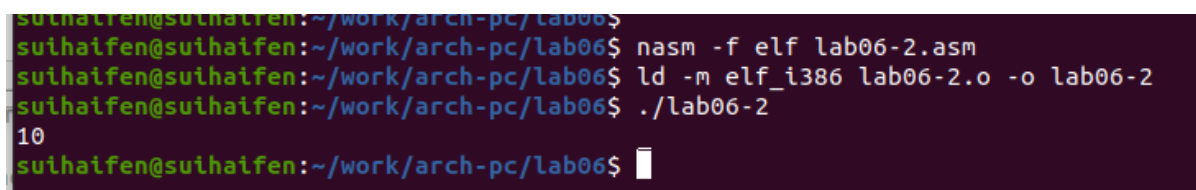
Заменяя символы на числа, и результат вывода — число 10, так как функции выводят именно числовые значения.



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

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
```

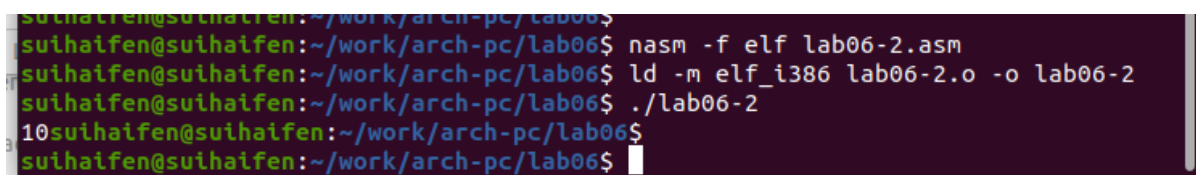
Рис. 2.7: Программа lab6-2.asm с числами



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

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

Позже заменил функцию `iprintLF` на `iprint`. Создал исполняемый файл и запустил его. Результат отличается тем, что теперь выводится значение без переноса строки.

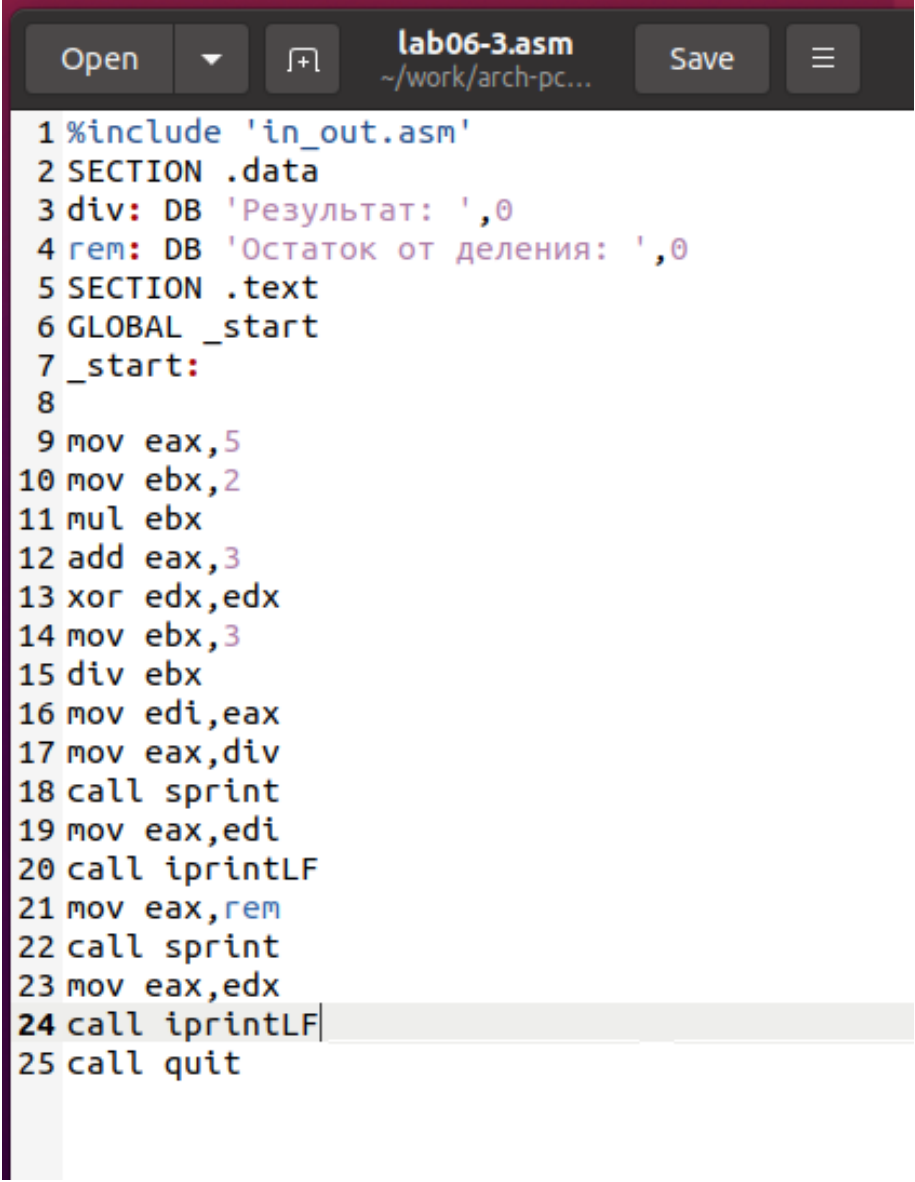


```
suihaifen@suihaifen:~/work/arch-pc/lab06$ nasm -f elf lab06-2.asm
suihaifen@suihaifen:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-2.o -o lab06-2
suihaifen@suihaifen:~/work/arch-pc/lab06$ ./lab06-2
10suihaifen@suihaifen:~/work/arch-pc/lab06$
```

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

2.2 Выполнение арифметических операций в NASM

Теперь рассмотрим пример программы, которая выполняет арифметические операции, используя выражение $f(x) = (5 * 2 + 3) / 3$.



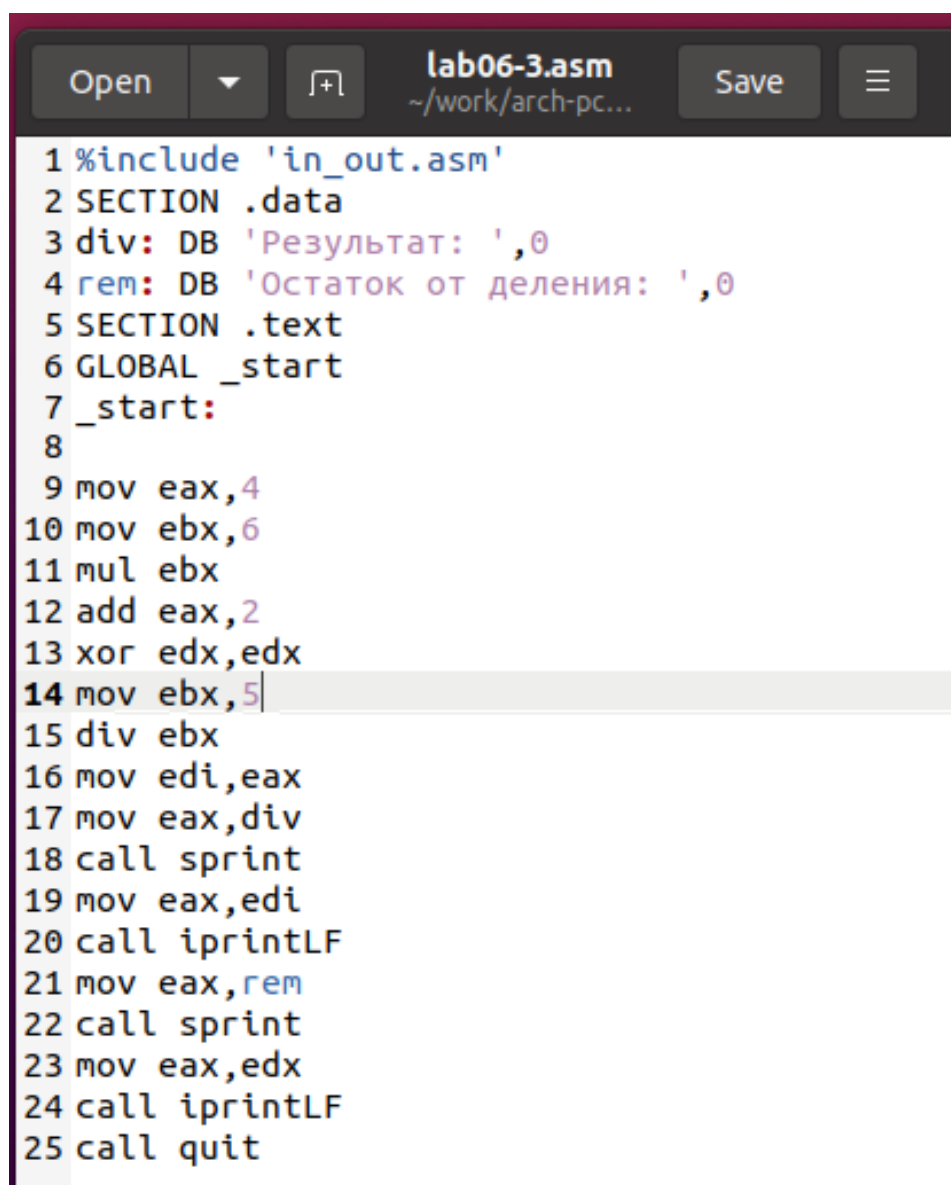
```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 rem: DB 'Остаток от деления: ',0
5 SECTION .text
6 GLOBAL _start
7 _start:
8
9 mov eax,5
10 mov ebx,2
11 mul ebx
12 add eax,3
13 xor edx,edx
14 mov ebx,3
15 div ebx
16 mov edi,eax
17 mov eax,div
18 call sprint
19 mov eax,edi
20 call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

Рис. 2.10: Программа lab6-3.asm

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

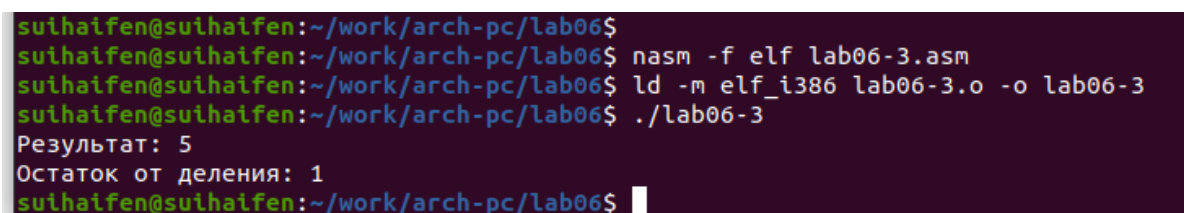
Рис. 2.11: Запуск программы lab6-3.asm

Я изменил программу для вычисления выражения $f(x) = (4 * 6 + 2)/5$, создал исполняемый файл и проверил его работу.



```
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
```

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



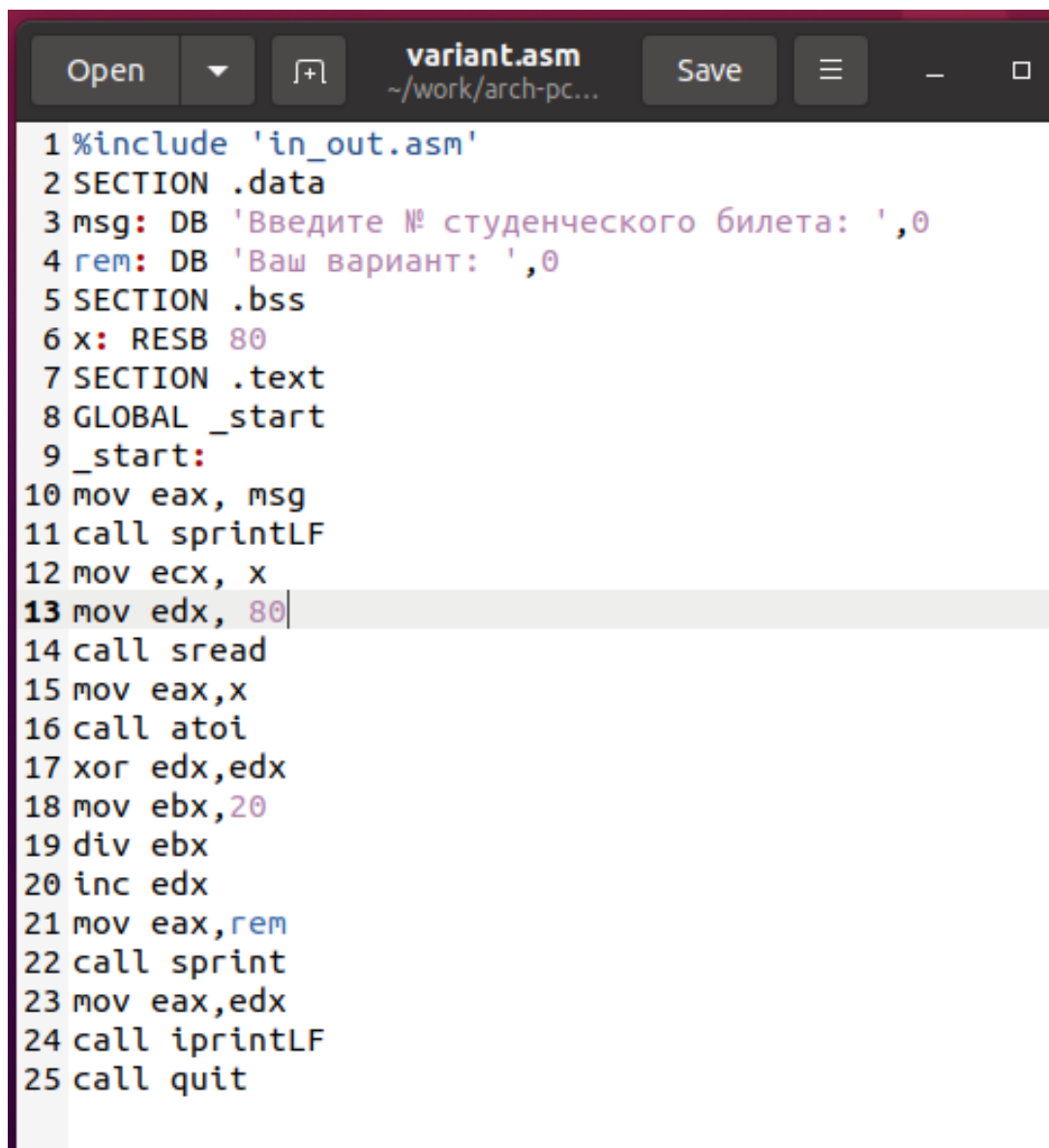
```
suihaifen@suihaifen:~/work/arch-pc/lab06$
suihaifen@suihaifen:~/work/arch-pc/lab06$ nasm -f elf lab06-3.asm
suihaifen@suihaifen:~/work/arch-pc/lab06$ ld -m elf_i386 lab06-3.o -o lab06-3
suihaifen@suihaifen:~/work/arch-pc/lab06$ ./lab06-3
Результат: 5
Остаток от деления: 1
suihaifen@suihaifen:~/work/arch-pc/lab06$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

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

Еще один пример — программа для вычисления варианта задания, используя

номер студенческого билета.

В этом случае значение для вычислений вводится с клавиатуры. Как я уже упоминал, ввод данных осуществляется в символьной форме, и для правильной работы арифметических операций необходимо преобразовать символы в числа. Для этого используется функция `atoi` из файла `in_out.asm`.



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

Рис. 2.14: Программа `variant.asm`

```
suihaifen@suihaifen:~/work/arch-pc/lab06$  
suihaifen@suihaifen:~/work/arch-pc/lab06$ nasm -f elf variant.asm  
suihaifen@suihaifen:~/work/arch-pc/lab06$ ld -m elf_i386 variant.o -o variant  
suihaifen@suihaifen:~/work/arch-pc/lab06$ ./variant  
Введите № студенческого билета:  
1032249608  
Ваш вариант: 9  
suihaifen@suihaifen:~/work/arch-pc/lab06$
```

Рис. 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, rem` переносит значение переменной с фразой ‘Ваш вариант:’ в регистр `eax`.
 - Инструкция `call sprint` вызывает подпрограмму для вывода строки.

2. Для чего используются следующие инструкции?

 - Инструкция `mov esx, x` — для помещения значения переменной `x` в регистр `esx`.
 - Инструкция `mov edx, 80` — для помещения значения `80` в регистр `edx`.
 - Инструкция `call sread` — для вызова подпрограммы 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?

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

- Инструкция `div ebx` выполняет деление номера студенческого билета на 20.
- Инструкция `inc edx` увеличивает значение регистра `edx` на 1.

В данном случае происходит деление номера студенческого билета на 20. Остаток от деления сохраняется в регистре `edx`, и к нему прибавляется 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, что необходимо для вычисления варианта.
7. Какие строки листинга отвечают за вывод на экран результата вычислений?
 - Инструкция `mov eax, edx` записывает результат вычислений в регистр `eax`.
 - Инструкция `call iprintLF` вызывает подпрограмму для вывода результата на экран.

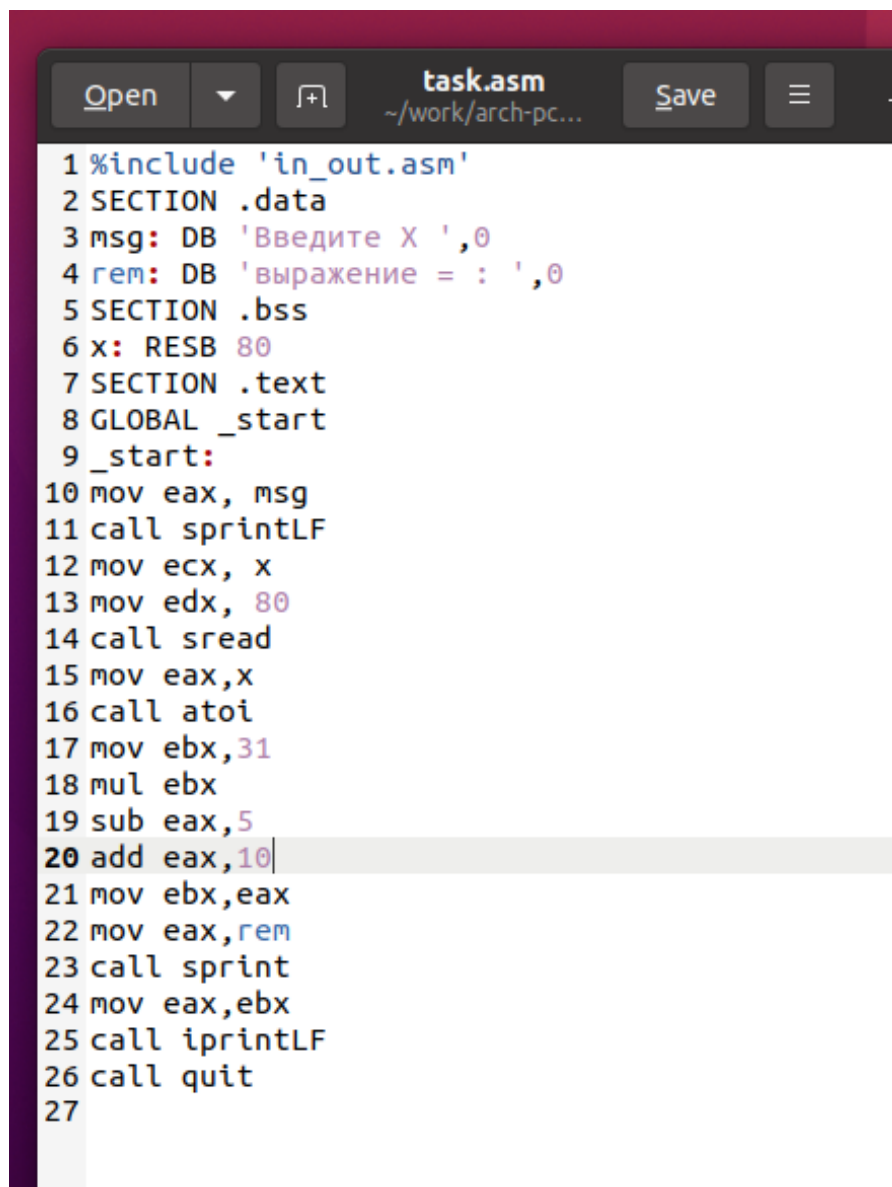
2.4 Задание для самостоятельной работы

Написал программу для вычисления выражения $y = f(x)$. Программа должна выводить выражение для вычисления, запросить ввод значения x , вычислить выражение в зависимости от введенного x и вывести результат. В функцию $f(x)$ выбрал вариант из таблицы 6.3 в соответствии с номером, полученным в лабораторной работе. Создал исполняемый файл и проверил его работу для значений x_1 и x_2 .

Получил вариант 9: $10 + (31x - 5)$ для $x=3$ и $x=1$.

При $x=3$ результат — 98.

При \$ x=1 \$ результат — 36.



```
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 sprintf
12 mov ecx, x
13 mov edx, 80
14 call sread
15 mov eax, x
16 call atoi
17 mov ebx, 31
18 mul ebx
19 sub eax, 5
20 add eax, 10
21 mov ebx, eax
22 mov eax, rem
23 call sprintf
24 mov eax, ebx
25 call iprintLF
26 call quit
27
```

Рис. 2.16: Программа task.asm

```
suihaifen@suihaifen:~/work/arch-pc/lab06$  
suihaifen@suihaifen:~/work/arch-pc/lab06$ nasm -f elf task.asm  
suihaifen@suihaifen:~/work/arch-pc/lab06$ ld -m elf_i386 task.o -o task  
suihaifen@suihaifen:~/work/arch-pc/lab06$ ./task  
Введите X  
3  
выражение = : 98  
suihaifen@suihaifen:~/work/arch-pc/lab06$ ./task  
Введите X  
1  
выражение = : 36  
suihaifen@suihaifen:~/work/arch-pc/lab06$
```

Рис. 2.17: Запуск программы task.asm

Программа работает корректно.

3 Выводы

Изучили работу с арифметическими операциями.