

Шаблон отчёта по лабораторной работе

Простейший вариант

Дмитрий Сергеевич Кулябов

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Выполнение задания для самостоятельной работы	20
4	Выводы	23

Список иллюстраций

2.1	Создание рабочей директории и файла lab6-1.nasm для записи кода на языке Ассемблера	6
2.2	Запуск Midnight commander	6
2.3	Запись кода из листинга в файл lab6-1.asm	7
2.4	Копирование файла in_out.asm в рабочую директорию	8
2.5	Вид каталога после перенесения файла in_out.asm	9
2.6	Сборка исполняемого файла из lab6-1.asm	9
2.7	Запуск исполняемого файла и результат вывода	10
2.8	Редактирование файла	10
2.9	Запуск исполняемого файла и результат вывода	11
2.10	Создание второго файла: lab6-2.asm	11
2.11	Запись кода из листинга в файл lab6-2.asm	12
2.12	Запуск исполняемого файла и результат вывода	12
2.13	Изменение файла lab6-2.asm	13
2.14	Сборка исполняемого файла и результат работы программы	13
2.15	Редактирование файла lab6-2.asm	14
2.16	Сборка и результат работы отредактированного файла	14
2.17	Создание третьего файла: lab6-3.asm	14
2.18	Вставка кода из листинга в созданный ранее файл	15
2.19	Сборка файла lab6-3.asm и результат его работы	15
2.20	Редактирование файла lab6-3.asm	16
2.21	Повторная сборка уже изменённого файла lab6-3.asm и результат его работы	16
2.22	Создание файла variant.asm для вычисления варианта для самостоятельной работы	17
2.23	Вставка кода из листинга в файл variant.asm	17
2.24	Сборка и запуск программы, а также результат выполнения	18
3.1	Код требуемой программы	21
3.2	Сборка исполняемого файла	22
3.3	Запуск программы и проверка её корректной работы	22

Список таблиц

1 Цель работы

Познакомиться с базовыми инструкциями языка Ассемблер, отвечающими за основные арифметические операции

2 Выполнение лабораторной работы

Для начала выполнения лабораторной работы необходимо создать папку рабочего каталога и файл lab6-1.asm (Рис. 2.1):

```
dsjakovleva@fedora:~$ mkdir ~/work/arch-pc/lab06
dsjakovleva@fedora:~$ cd ~/work/arch-pc/lab06
dsjakovleva@fedora:~/work/arch-pc/lab06$ touch lab6-1.asm
dsjakovleva@fedora:~/work/arch-pc/lab06$
```

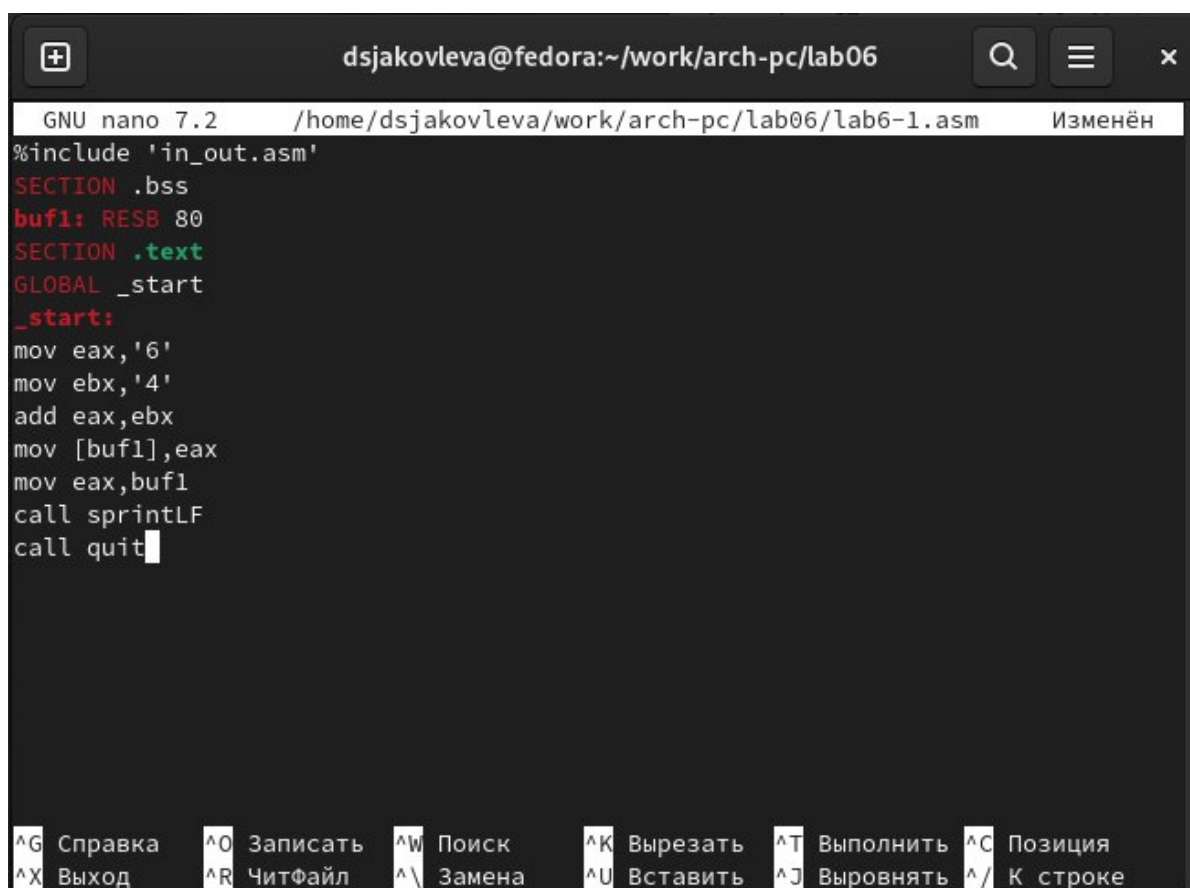
Рис. 2.1: Создание рабочей директории и файла lab6-1.asm для записи кода на языке Ассемблера

После этого, для более комфортной работы, запустим Midnight commander (Рис. 2.2):

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

Рис. 2.2: Запуск Midnight commander

Вставим в наш созданный файл код из листинга 6.1 с помощью команды F4 (редактор) в MC (Рис. 2.3):



```
GNU nano 7.2 /home/dsjakovleva/work/arch-pc/lab06/lab6-1.asm  Изменён
%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
```

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выровнять ^/ К строке

Рис. 2.3: Запись кода из листинга в файл lab6-1.asm

Перед сборкой файла стоит учесть, что он использует сторонний файл `in_out.asm`. С помощью команды `F5` скопируем его из каталога пятой лабораторной работы (Рис. 2.4):

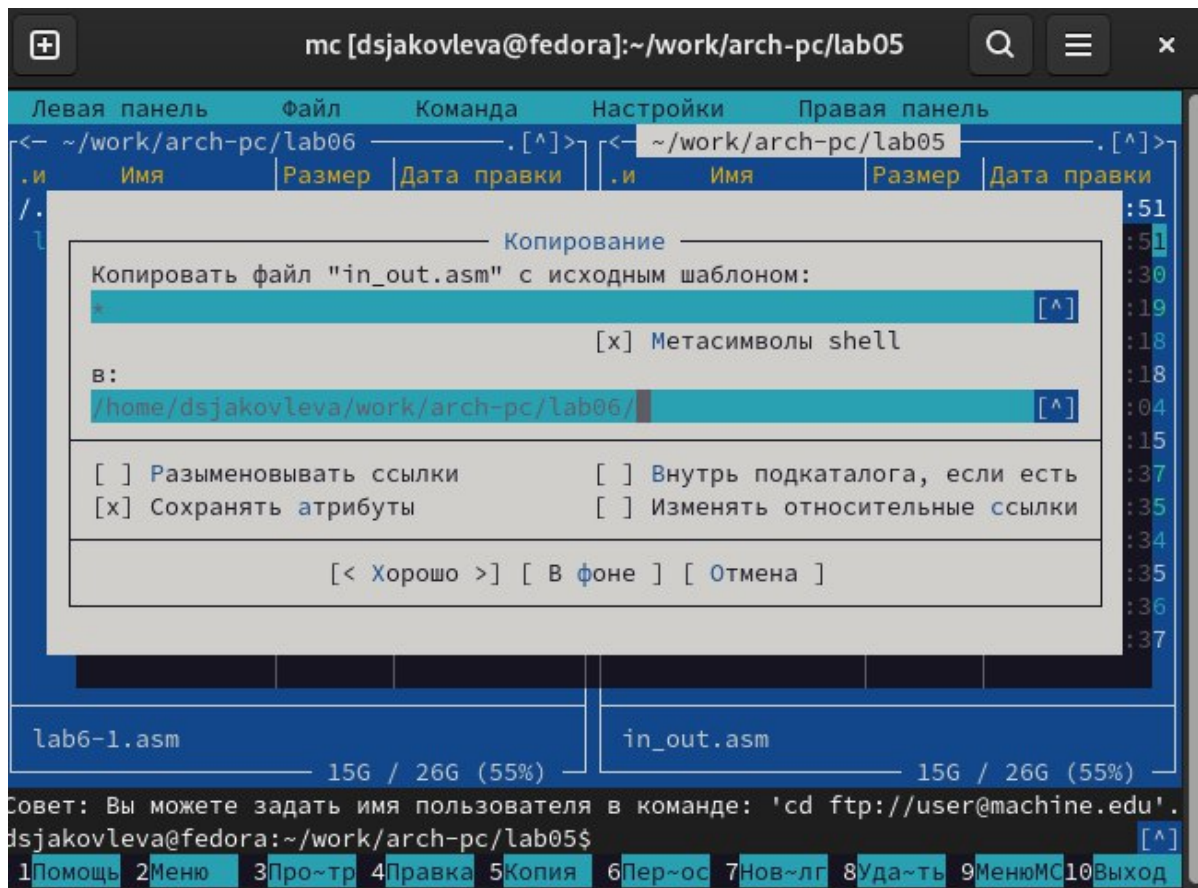


Рис. 2.4: Копирование файла in_out.asm в рабочую директорию

Так будет выглядеть наша рабочая директория (Рис. 2.5):

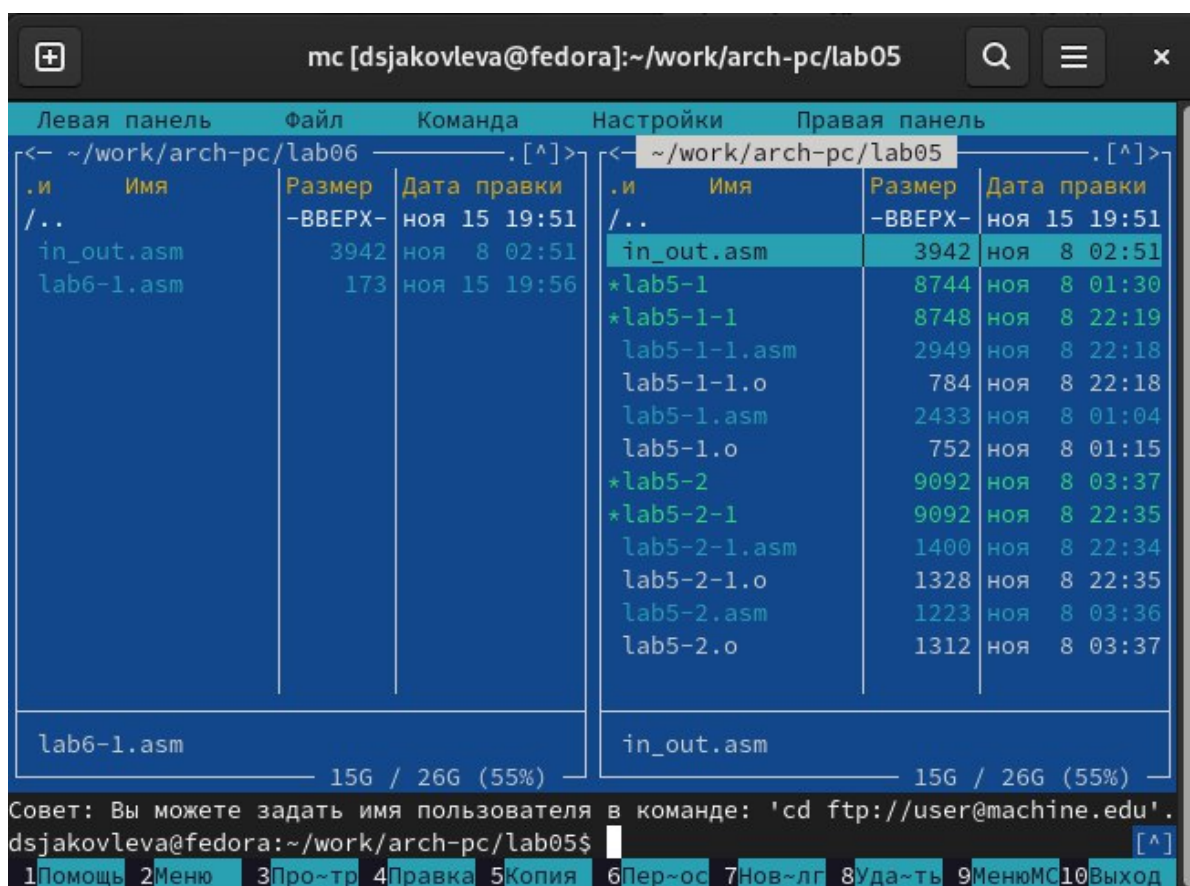


Рис. 2.5: Вид каталога после перенесения файла in_out.asm

Теперь соберём наш файл в исполняемое приложение уже знакомыми инструментами, nasm и ld (Рис. 2.6):

```
dsjakovleva@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-1.asm
dsjakovleva@fedora:~/work/arch-pc/lab06$ d -m elf_i386 -o lab6-1 lab6-1.o
bash: d: команда не найдена...
dsjakovleva@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-1 lab6-1.o
dsjakovleva@fedora:~/work/arch-pc/lab06$
```

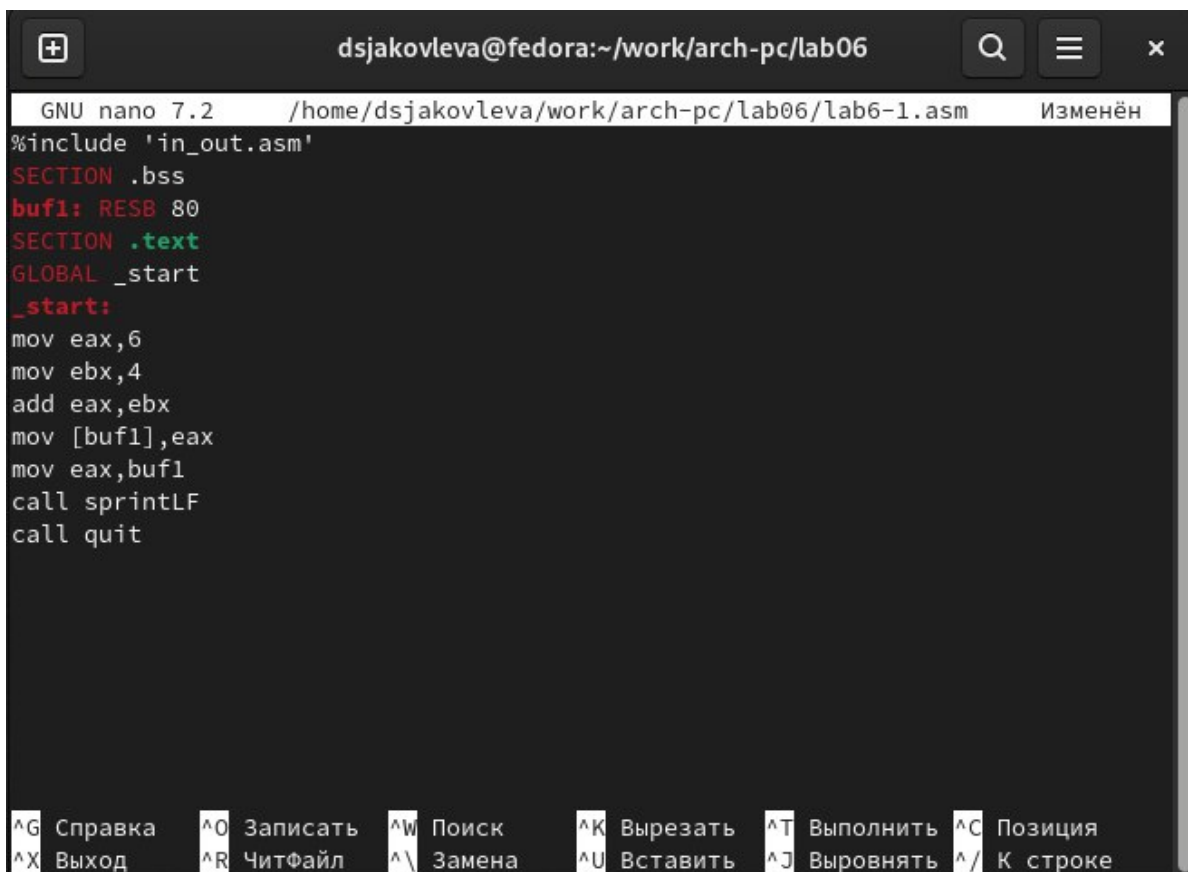
Рис. 2.6: Сборка исполняемого файла из lab6-1.asm

Теперь запустим файл и посмотрим на результат (Рис. 2.7):

```
dsjakovleva@fedora:~/work/arch-pc/lab06$ ./lab6-1
j
dsjakovleva@fedora:~/work/arch-pc/lab06$
```

Рис. 2.7: Запуск исполняемого файла и результат вывода

Нам выводит символ j, однако это неправильный вывод. Наша цель - сложить 6 и 4, и получить в выводе число 10. Попробуем изменить наш файл (Рис. 2.8):



```
GNU nano 7.2 /home/dsjakovleva/work/arch-pc/lab06/lab6-1.asm  Изменён
%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
```

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выровнять ^/ К строке

Рис. 2.8: Редактирование файла

Мы убрали кавычки у цифр, и теперь мы складываем уже не символы “6” и “4” (когда мы складываем символы, мы складываем их коды ASCII), а числа. Теперь попробуем собрать исполняемый файл также, как собирали до этого, и запустим (Рис. 2.9):

```
dsjakovleva@fedora:~/work/arch-pc/lab06$ ./lab6-1  
dsjakovleva@fedora:~/work/arch-pc/lab06$
```

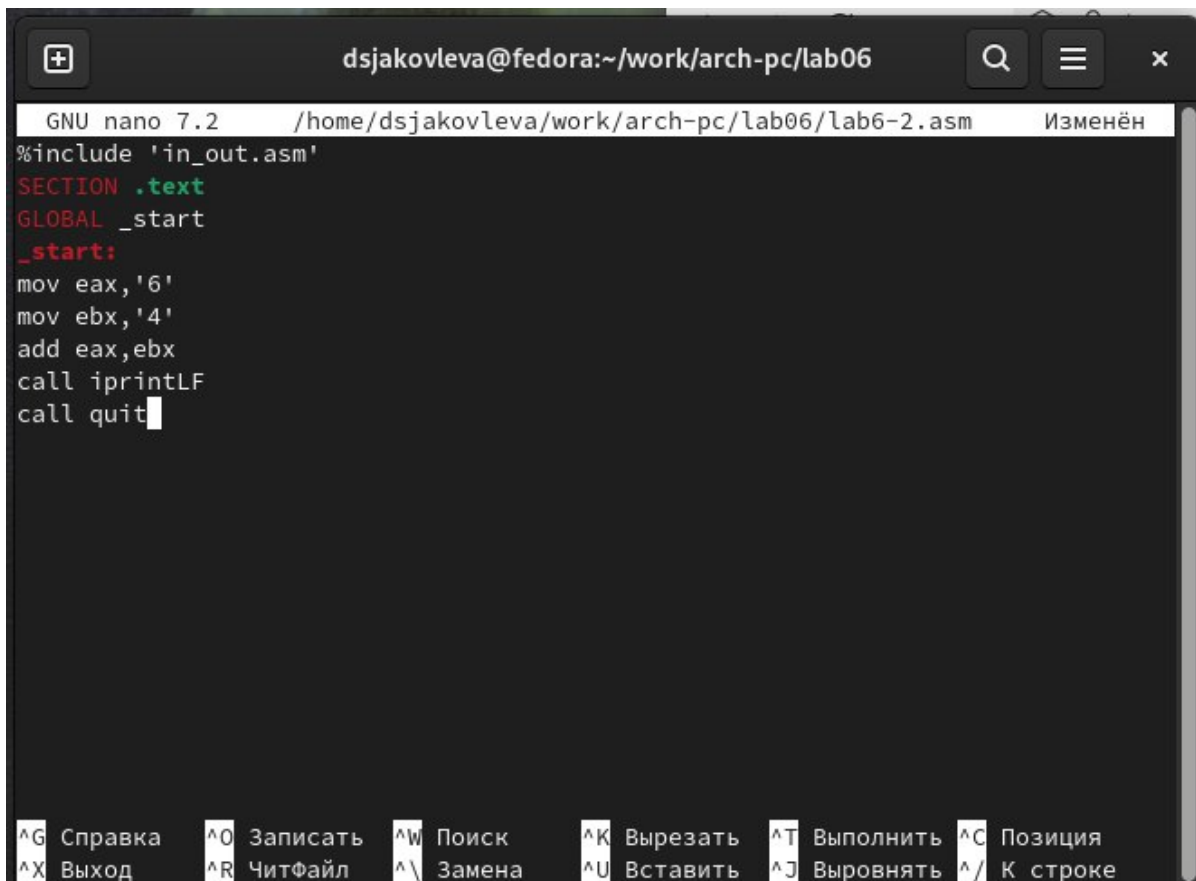
Рис. 2.9: Запуск исполняемого файла и результат вывода

Мы видим, что ничего не вывелось. Но так ли это? Когда мы вызываем команду `sprintf`, она выводит не число 10, а символ с номером 10. Посмотрим на таблицу ASCII и увидим, что символ под номером 10 это символ перевода строки. Именно поэтому мы его не видим, мы видим просто новую строку. Теперь создадим второй файл под названием `lab6-2.asm` (Рис. 2.10):

```
dsjakovleva@fedora:~/work/arch-pc/lab06$ touch ~/work/arch-pc/lab06/lab6-2.asm  
dsjakovleva@fedora:~/work/arch-pc/lab06$
```

Рис. 2.10: Создание второго файла: `lab6-2.asm`

Теперь вставим в него код из листинга 6.2 (Рис. 2.11):

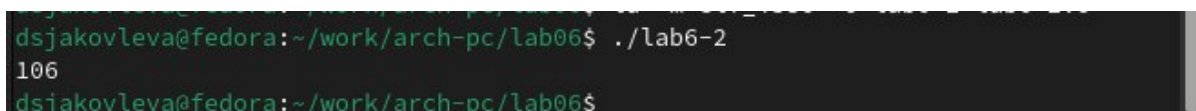


```
GNU nano 7.2 /home/dsjakovleva/work/arch-pc/lab06/lab6-2.asm Изменён
%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax, '6'
mov ebx, '4'
add eax, ebx
call iprintLF
call quit
```

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R ЧитФайл ^\ Замена ^U Вставить ^J Выворнять ^/_ К строке

Рис. 2.11: Запись кода из листинга в файл lab6-2.asm

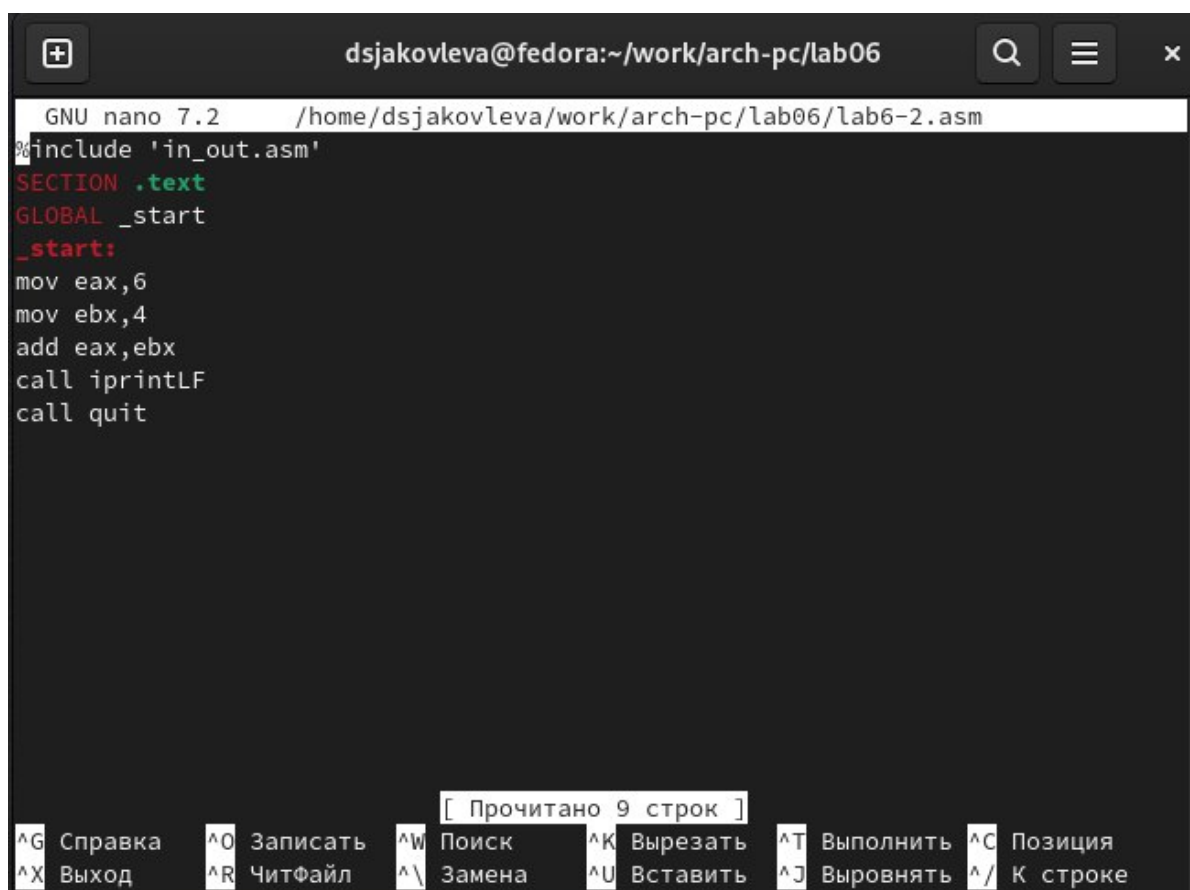
Как мы видим, основное отличие заключается в том, что вместо `sprintLF` используется `iprintLF`. Соберём файл и запустим его, чтобы посмотреть, как изменится вывод (Рис. 2.12):



```
dsjakovleva@fedora:~/work/arch-pc/lab06$ ./lab6-2
106
dsjakovleva@fedora:~/work/arch-pc/lab06$
```

Рис. 2.12: Запуск исполняемого файла и результат вывода

Мы видим число 106. Так как цифры в коде указаны в кавычках, мы складываем их коды (54 и 52 в сумме дают 106). Теперь программа способна вывести число, а не символ ASCII с соответствующим номером. Теперь, если мы уберём кавычки у цифр, программа должна вывести 10. Убедимся в этом, сделав соответствующие изменения в коде (Рис. 2.13):



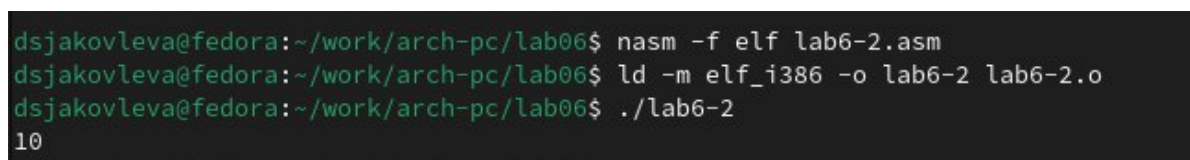
```
GNU nano 7.2 /home/dsjakovleva/work/arch-pc/lab06/lab6-2.asm
%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprintLF
call quit
```

[Прочитано 9 строк]

^G Справка	^O Записать	^W Поиск	^K Вырезать	^T Выполнить	^C Позиция
^X Выход	^R ЧитФайл	^\ Замена	^U Вставить	^J Выводить	^/ К строке

Рис. 2.13: Изменение файла lab6-2.asm

Соберём программу и запустим её (Рис. 2.14):



```
dsjakovleva@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
dsjakovleva@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
dsjakovleva@fedora:~/work/arch-pc/lab06$ ./lab6-2
10
```

Рис. 2.14: Сборка исполняемого файла и результат работы программы

Как видим, программа действительно вывела число 10. Кроме операции `iprintLF` в файле `in_out.asm` есть операция `iprint`. Посмотрим, чем они отличаются. Заменяем в коде `iprintLF` на `iprint` (Рис. 2.15):

```
GNU nano 7.2 /home/dsjakovleva/work/arch-pc/lab06/lab6-2.asm Изменён
%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprint
call quit
```

Рис. 2.15: Редактирование файла lab6-2.asm

Попробуем собрать программу и запустить её (Рис. 2.16):

```
dsjakovleva@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
dsjakovleva@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
dsjakovleva@fedora:~/work/arch-pc/lab06$ ./lab6-2
10dsjakovleva@fedora:~/work/arch-pc/lab06$
```

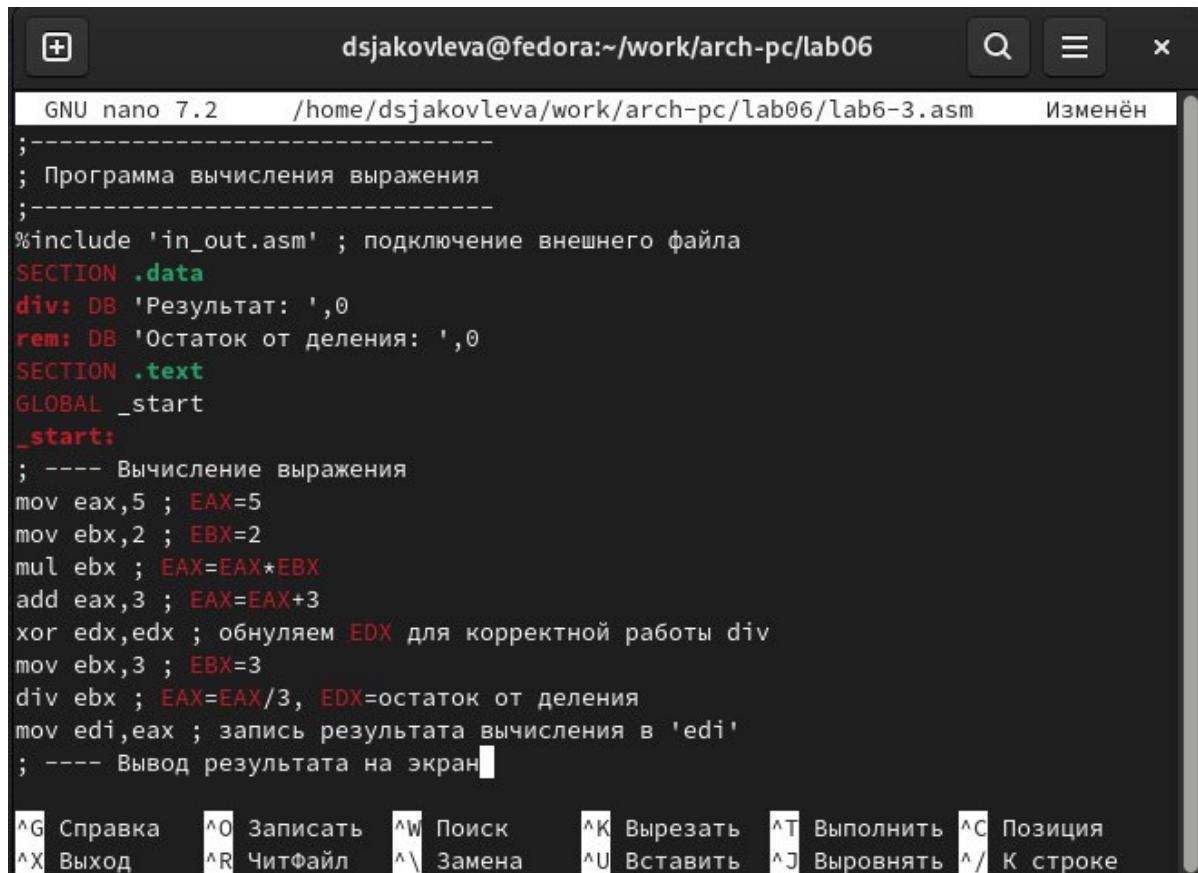
Рис. 2.16: Сборка и результат работы отредактированного файла

Как видим, операция `iprint` не переносит на следующую строку, в отличие от `iprintLF`. В этом их разница. Теперь создадим третий файл (Рис. 2.17):

```
10dsjakovleva@fedora:~/work/arch-pc/lab06$ touch ~/work/arch-pc/lab06/lab6-3.asm
dsjakovleva@fedora:~/work/arch-pc/lab06$
```

Рис. 2.17: Создание третьего файла: lab6-3.asm

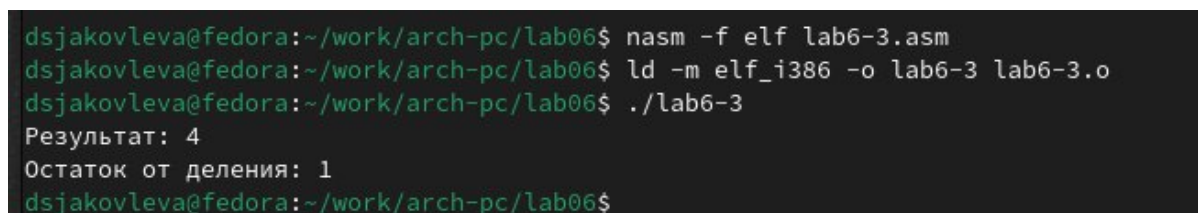
Он должен выводить значение функции $(5*2+3)/3$. Для этого вставим код из файла листинга 6.3 (Рис. 2.18):



```
GNU nano 7.2 /home/dsjakovleva/work/arch-pc/lab06/lab6-3.asm
;-----
; Программа вычисления выражения
;-----
#include 'in_out.asm' ; подключение внешнего файла
SECTION .data
div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения
mov eax,5 ; EAX=5
mov ebx,2 ; EBX=2
mul ebx ; EAX=EAX*EBX
add eax,3 ; EAX=EAX+3
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,3 ; EBX=3
div ebx ; EAX=EAX/3, EDX=остаток от деления
mov edi,eax ; запись результата вычисления в 'edi'
; ---- Вывод результата на экран
^G Справка    ^O Записать   ^W Поиск      ^K Вырезать   ^T Выполнить  ^C Позиция
^X Выход      ^R ЧитФайл   ^\ Замена     ^U Вставить   ^J Выровнять  ^/_ К строке
```

Рис. 2.18: Вставка кода из листинга в созданный ранее файл

Попробуем запустить эту программу, предварительно её собрав (Рис. 2.19):

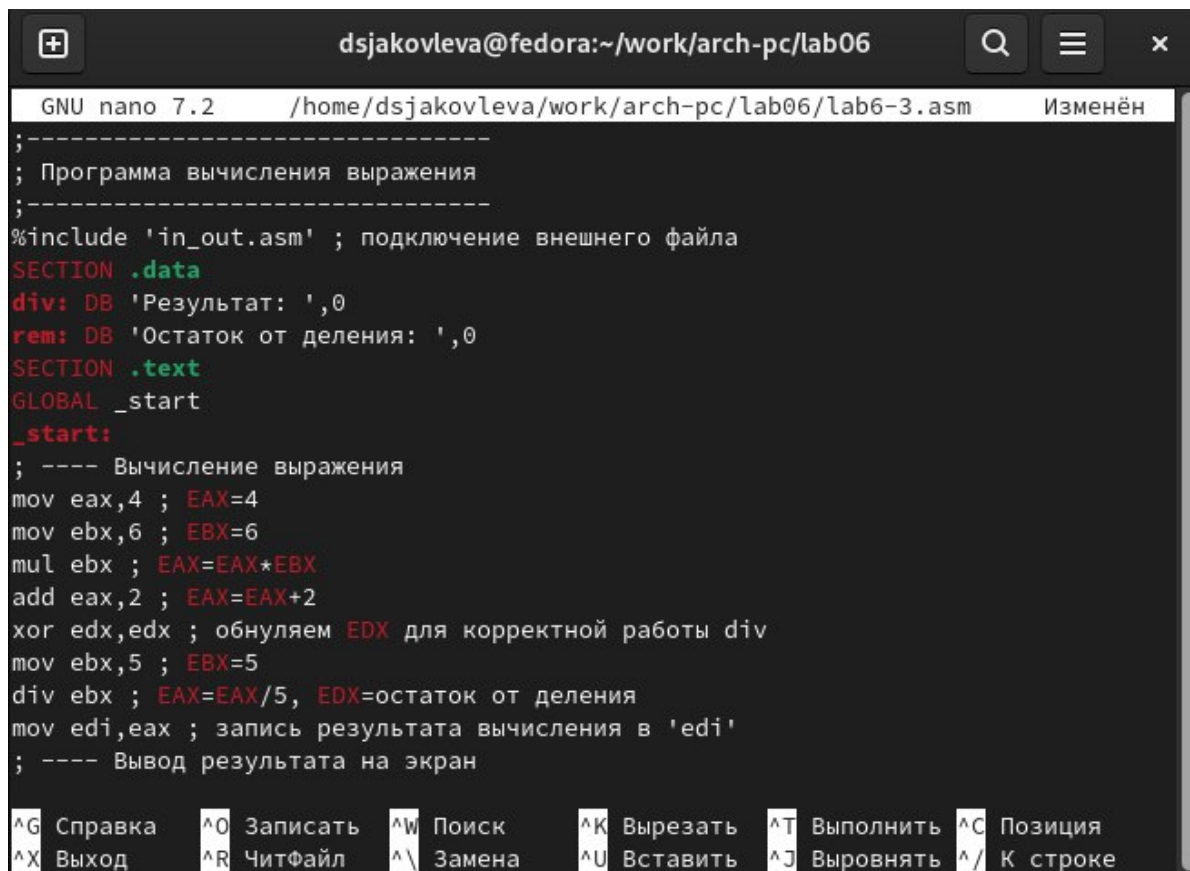


```
dsjakovleva@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
dsjakovleva@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
dsjakovleva@fedora:~/work/arch-pc/lab06$ ./lab6-3
Результат: 4
Остаток от деления: 1
dsjakovleva@fedora:~/work/arch-pc/lab06$
```

Рис. 2.19: Сборка файла lab6-3.asm и результат его работы

Полученный результат совпадает с результатом, указанным в лабораторной работе. Теперь изменим файл так, чтобы он вычислял значение выражения

$(4*6+2)/5$. Для этого в коде заменим число 5 на 4, число 2 на 6, число 3 на 2, и второе число 3 на 5 (Рис. 2.20):

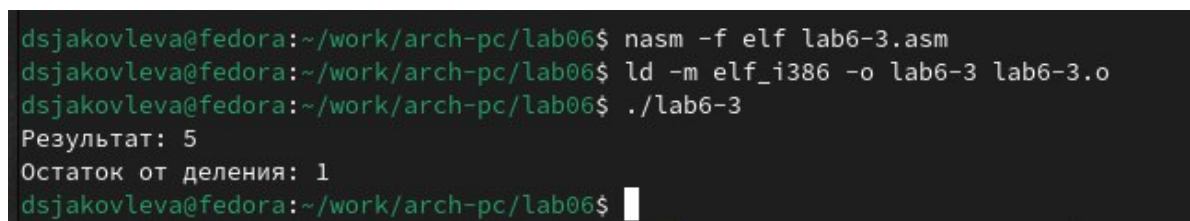


```
GNU nano 7.2 /home/dsjakovleva/work/arch-pc/lab06/lab6-3.asm Изменён
;-----
; Программа вычисления выражения
;-----
#include 'in_out.asm' ; подключение внешнего файла
SECTION .data
div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения
mov eax,4 ; EAX=4
mov ebx,6 ; EBX=6
mul ebx ; EAX=EAX*EBX
add eax,2 ; EAX=EAX+2
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,5 ; EBX=5
div ebx ; EAX=EAX/5, EDX=остаток от деления
mov edi,eax ; запись результата вычисления в 'edi'
; ---- Вывод результата на экран

^G Справка  ^O Записать  ^W Поиск    ^K Вырезать  ^T Выполнить ^C Позиция
^X Выход    ^R ЧитФайл  ^_ Замена   ^U Вставить  ^J Выводить   ^/ К строке
```

Рис. 2.20: Редактирование файла lab6-3.asm

Соберём программу и запустим её (Рис. 2.21):



```
dsjakovleva@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
dsjakovleva@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
dsjakovleva@fedora:~/work/arch-pc/lab06$ ./lab6-3
Результат: 5
Остаток от деления: 1
dsjakovleva@fedora:~/work/arch-pc/lab06$
```

Рис. 2.21: Повторная сборка уже изменённого файла lab6-3.asm и результат его работы

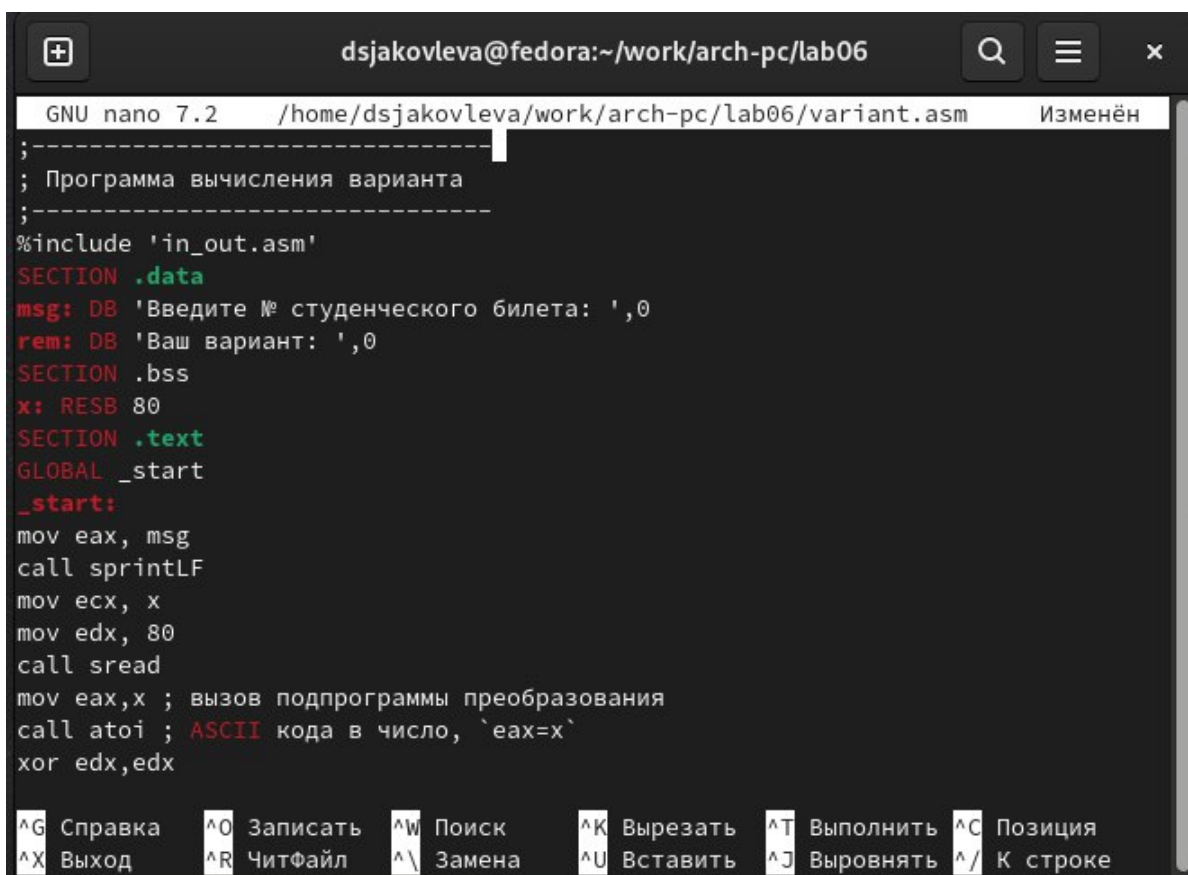
Пересчитав значение выражения вручную, убеждаемся, что вывод корректный. Теперь создадим файл variant.asm для вычисления варианта самостоятельной

работы (Рис. 2.22):

```
dsjakovleva@fedora:~/work/arch-pc/lab06$ touch ~/work/arch-pc/lab06/variant.asm
dsjakovleva@fedora:~/work/arch-pc/lab06$
```

Рис. 2.22: Создание файла variant.asm для вычисления варианта для самостоятельной работы

Вставим в файл код из листинга 6.4, который вычисляет номер варианта по формуле $(s \bmod 20) + 1$, где s - номер студенческого билета (Рис. 2.23):



```
GNU nano 7.2 /home/dsjakovleva/work/arch-pc/lab06/variant.asm Изменён
;-----
; Программа вычисления варианта
;-----
%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 ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x`
xor edx, edx

^G Справка      ^O Записать     ^W Поиск       ^K Вырезать    ^T Выполнить   ^C Позиция
^X Выход        ^R ЧитФайл     ^\ Замена     ^U Вставить    ^J Выводить    ^/ К строке
```

Рис. 2.23: Вставка кода из листинга в файл variant.asm

Соберём и запустим программу, указав номер студенческого билета. В моём случае это 1132243816 (Рис. 2.24):

```
dsjakovleva@fedora:~/work/arch-pc/lab06$ nasm -f elf variant.asm
dsjakovleva@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o variant variant.o
dsjakovleva@fedora:~/work/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1132243816
Ваш вариант: 17
dsjakovleva@fedora:~/work/arch-pc/lab06$
```

Рис. 2.24: Сборка и запуск программы, а также результат выполнения

Программа вывела число 17. Действительно, ведь остаток от деления числа 1132243816 на 20 равен 16. $16 + 1 = 17$, соответственно.

Разберём работу кода, ответив на предложенные в лабораторной работе вопросы:

1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’?

За это отвечает 25-ая строчка `call sprint`, перед которой идёт строка `mov eax,rem`, которая перемещает строку с фразой в регистр `eax`, из которого мы считываем данные для вывода

2. Для чего используются следующие инструкции?

```
mov ecx, x
mov edx, 80
call sread
```

Эти строки используются для того, чтобы записать данные в переменную `x`

3. Для чего используется инструкция “`call atoi`”?

Для преобразования ASCII кода в число

4. Какие строки листинга 6.4 отвечают за вычисления варианта?

Напрямую за вычисление отвечают следующие строки:

```
div ebx
inc edx
```

Первая делит число `x` в регистре `eax` на значение регистра `ebx` (в нашем случае 20), а вторая прибавляет к значению регистра `edx` (куда сохранился остаток от деления в прошлой операции) единицу

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

Как уже было сказано в ответе на предыдущий вопрос, в регистр edx

6. Для чего используется инструкция “inc edx”?

Для увеличения значения регистра edx на единицу

7. Какие строки листинга 6.4 отвечают за вывод на экран результата вычисления?

За это отвечают строки:

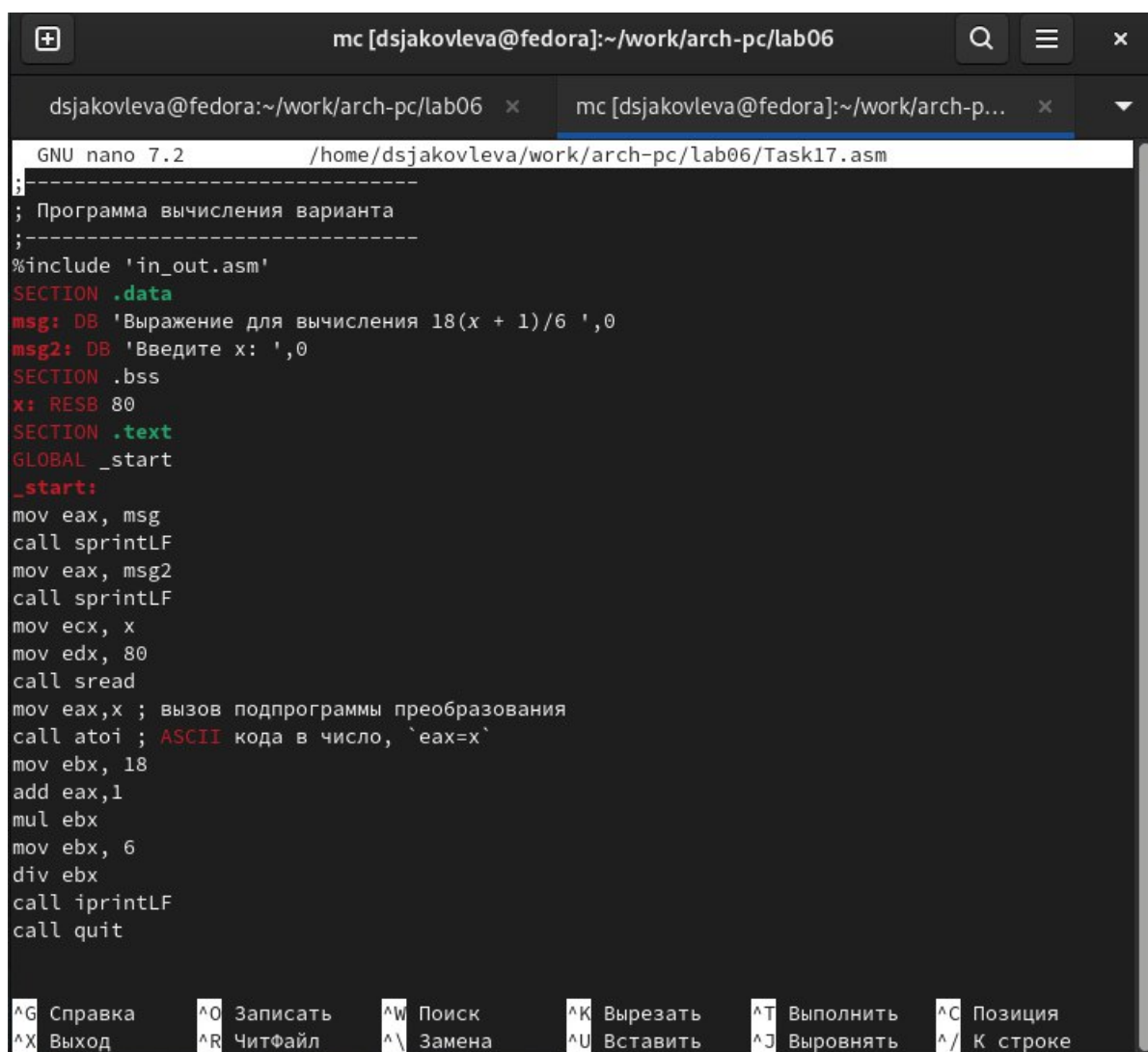
```
mov eax,edx
```

```
call iprintLF
```

первая строка переносит значение регистра edx в eax, а вторая вызывает операцию вывода значения регистра eax на экран

3 Выполнение задания для самостоятельной работы

Теперь в качестве самостоятельной работы напомним код программы для вычисления выражения в варианте 17: $18(\pi + 1)/6$. В предварительно созданном файле Task17.asm впишем следующий код (Рис. 3.1):



```
GNU nano 7.2 /home/dsjakovleva/work/arch-pc/lab06/Task17.asm
;-----
; Программа вычисления варианта
;-----
#include 'in_out.asm'
SECTION .data
msg: DB 'Выражение для вычисления 18(x + 1)/6 ',0
msg2: DB 'Введите x: ',0
SECTION .bss
x: RESB 80
SECTION .text
GLOBAL _start
_start:
mov eax, msg
call sprintf
mov eax, msg2
call sprintf
mov ecx, x
mov edx, 80
call sread
mov eax, x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x`
mov ebx, 18
add eax, 1
mul ebx
mov ebx, 6
div ebx
call iprintf
call quit
```

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить ^C Позиция
^X Выход ^R Читфайл ^\ Замена ^U Вставить ^J Выровнять ^/ К строке

Рис. 3.1: Код требуемой программы

Он немного отличается от предыдущих программ. За вычисление выражения отвечают следующие команды:

```
add eax,1
mul ebx
div ebx
```

Где первая отвечает за прибавление к регистру `eax` единицы.

Вторая отвечает за умножение значения регистра `eax` (где теперь находится $x+1$) на значение регистра `ebx` (куда записано значение 18).

Третья отвечает за деление значения регистра `eax` (где теперь находится $18 \cdot (x+1)$) на значение регистра `ebx` (которое мы перезаписали на 6).

Попробуем собрать нашу программу (Рис. 3.2):

```
dsjakovleva@fedora:~/work/arch-pc/lab06$ nasm -f elf Task17.asm
dsjakovleva@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o Task17 Task17.o
```

Рис. 3.2: Сборка исполняемого файла

И запустим код, указав в качестве `x` предложенные в лабораторной работе значения (Рис. 3.3):

```
dsjakovleva@fedora:~/work/arch-pc/lab06$ ./Task17
Выражение для вычисления 18(x + 1)/6
Введите x:
3
12
dsjakovleva@fedora:~/work/arch-pc/lab06$ ./Task17
Выражение для вычисления 18(x + 1)/6
Введите x:
1
6
dsjakovleva@fedora:~/work/arch-pc/lab06$
```

Рис. 3.3: Запуск программы и проверка её корректной работы

Как видим, программа выводит правильные значения выражения.

4 Выводы

В результате выполнения лабораторной работы было получено представление о том, какие арифметические операции есть в языке Ассемблера, и как они работают. Были написаны программы, использующие в себе операции сложения, вычитания, умножения и деления.