

Отчёта по лабораторной работе 6

Освоение арифметических инструкций языка ассемблера NASM.

Валиева Марина Русланбековна

Содержание

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

Список иллюстраций

4.1	Пример программы	10
4.2	Работа программы	11
4.3	Пример программы	12
4.4	Работа программы	13
4.5	Пример программы	14
4.6	Работа программы	14
4.7	Пример программы	15
4.8	Работа программы	15
4.9	Работа программы	16
4.10	Пример программы	17
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

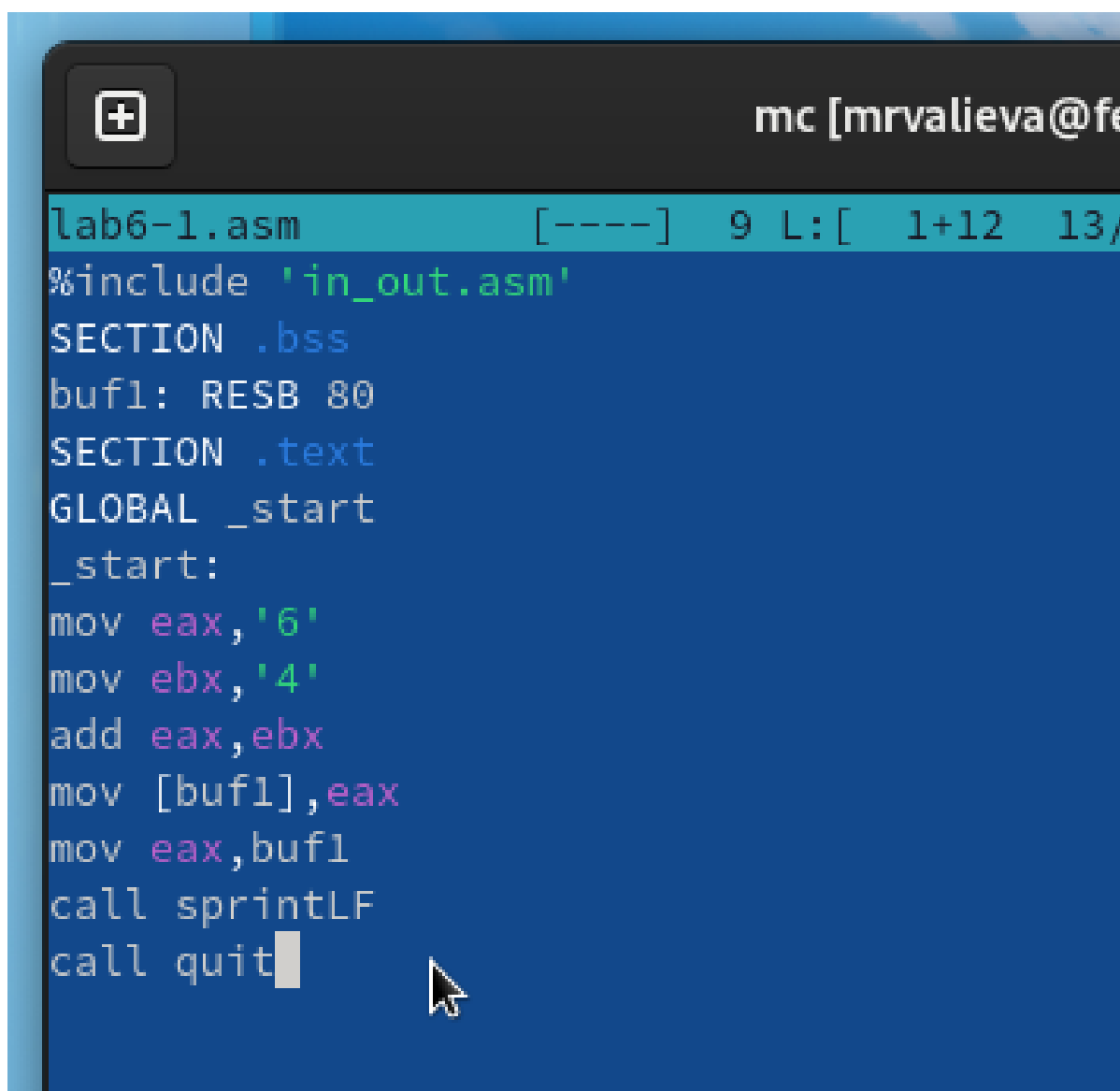
регистра `rax` (или явно `subq %rax, ll`, где указывается, что операнды имеют 64-размер), и результат помещается в `ll`; `subw go, %ax` – вычитание из содержимого `ax` числа по адресу `go`, результат помещается в `ax`; `sbb %rdx, %rax` – вычитание с дополнительным вычитанием флага переноса (из числа 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`.

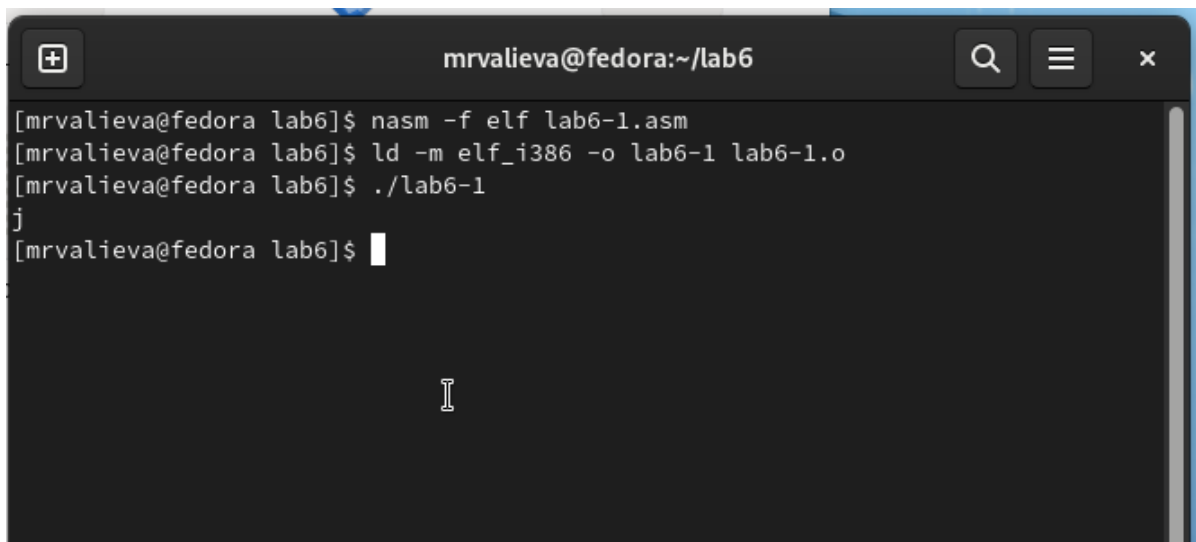
4 Выполнение лабораторной работы

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)



```
lab6-1.asm [----] 9 L: [ 1+12 13/
#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 _exit
```

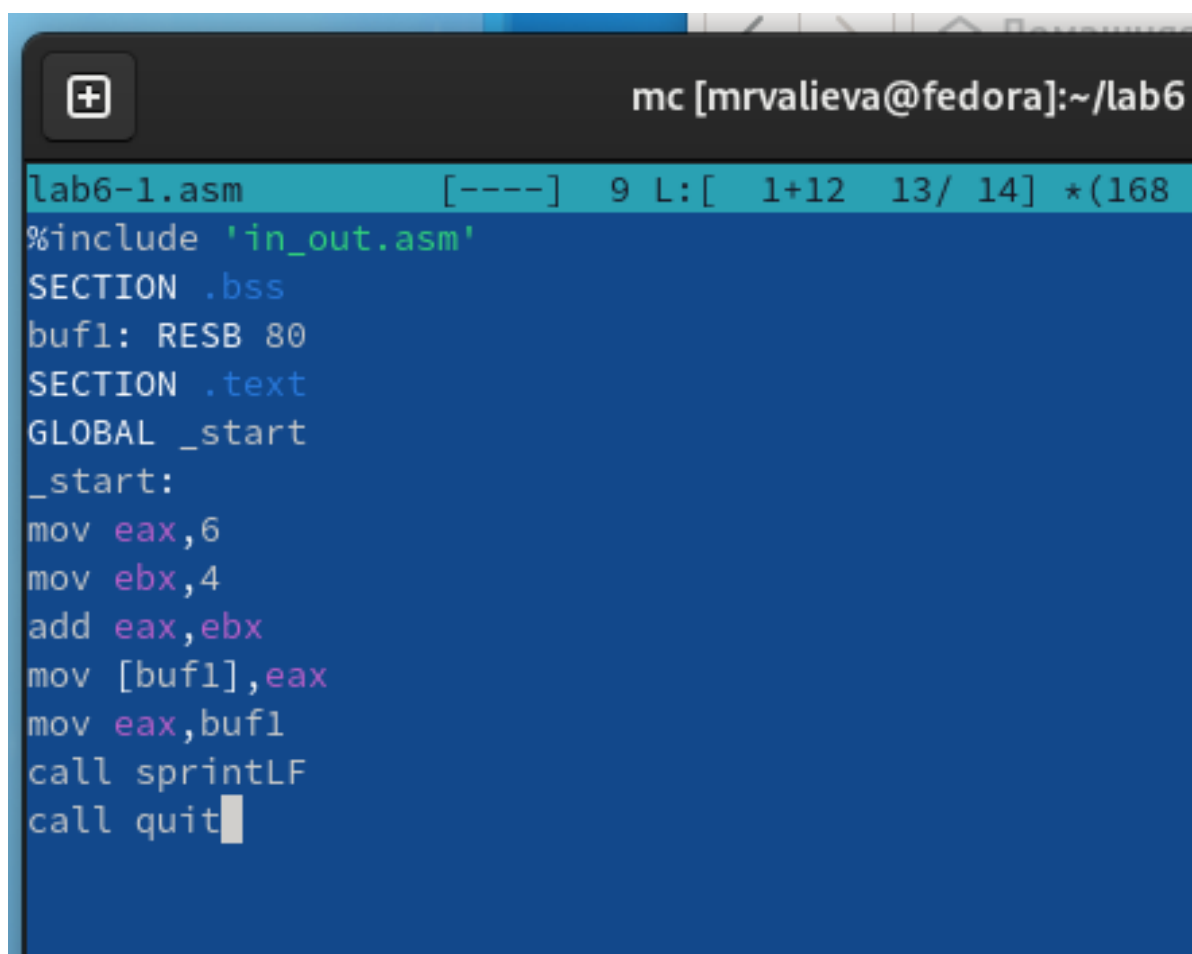
Рис. 4.1: Пример программы

A terminal window titled 'mrvalieva@fedora:~/lab6' with search, menu, and close buttons. The terminal shows the following commands and output:

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

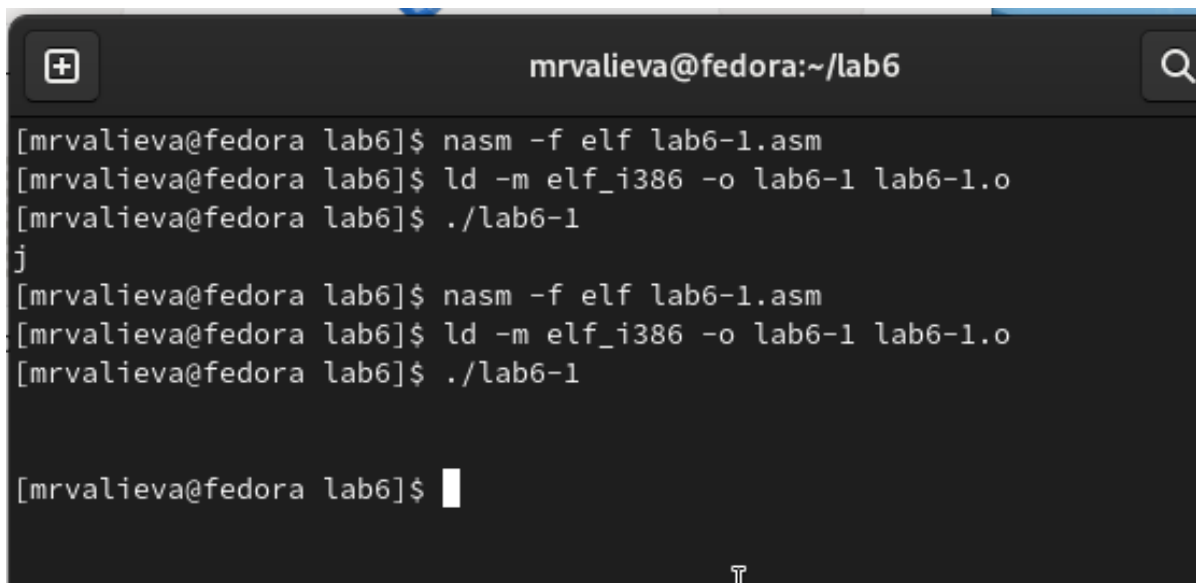
Рис. 4.2: Работа программы

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



```
lab6-1.asm      [----]  9 L:[ 1+12 13/ 14] *(168
#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 _exit
```

Рис. 4.3: Пример программы

A terminal window titled 'mrvalieva@fedora:~/lab6' with a search icon in the top right. The terminal shows the following commands and output:

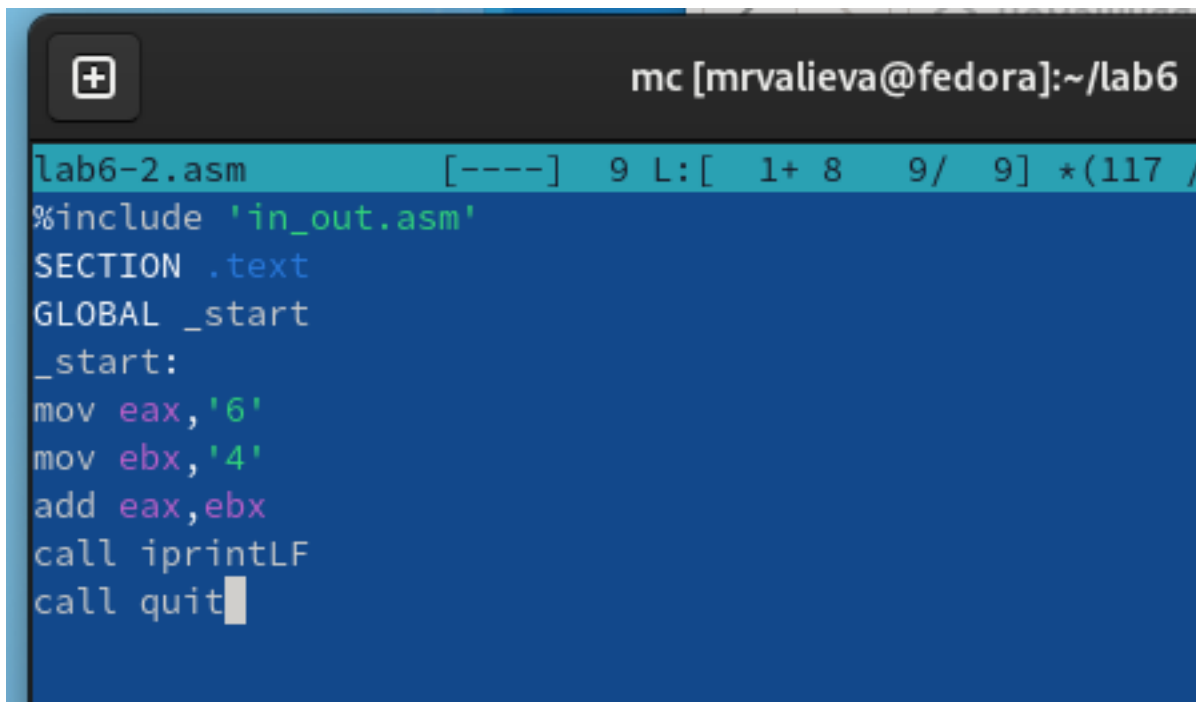
```
[mrvalieva@fedora lab6]$ nasm -f elf lab6-1.asm
[mrvalieva@fedora lab6]$ ld -m elf_i386 -o lab6-1 lab6-1.o
[mrvalieva@fedora lab6]$ ./lab6-1
j
[mrvalieva@fedora lab6]$ nasm -f elf lab6-1.asm
[mrvalieva@fedora lab6]$ ld -m elf_i386 -o lab6-1 lab6-1.o
[mrvalieva@fedora lab6]$ ./lab6-1

[mrvalieva@fedora lab6]$
```

Рис. 4.4: Работа программы

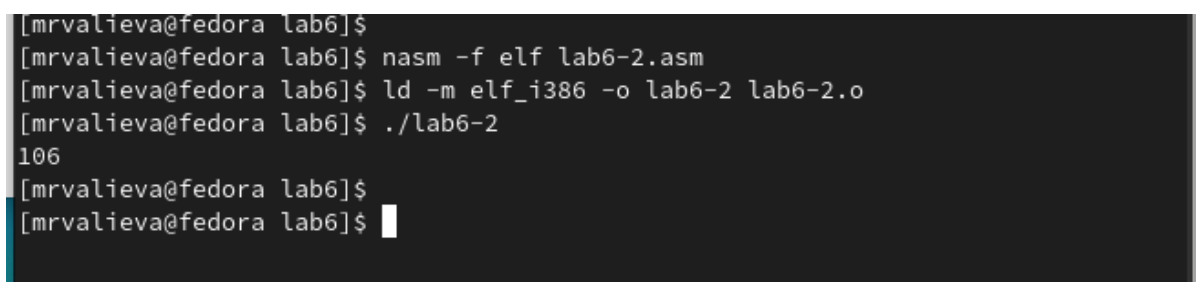
Никакой символ не виден, но он есть. Это возврат каретки LF.

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)



```
mc [mrvalieva@fedora]:~/lab6
lab6-2.asm [----] 9 L:[ 1+ 8 9/ 9] *(117 /
#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



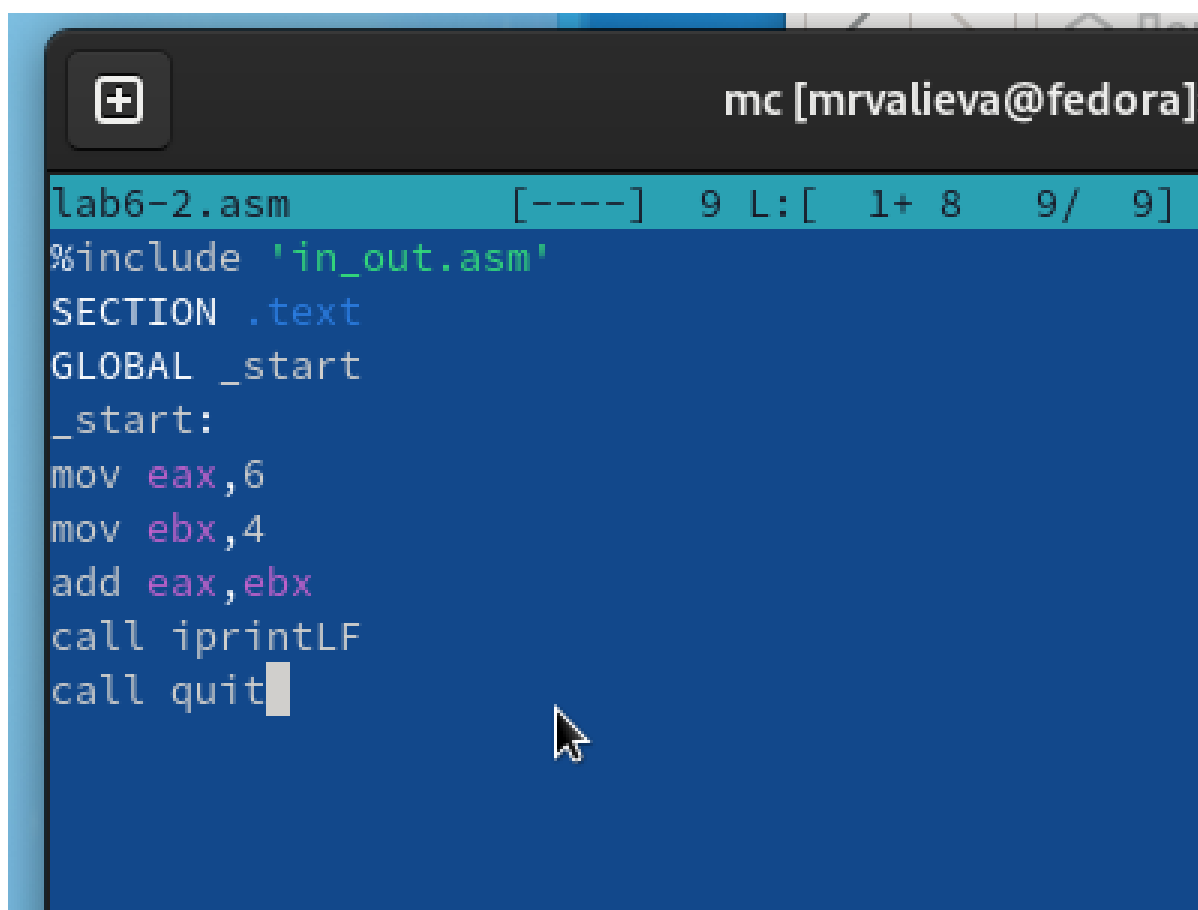
```
[mrvalieva@fedora lab6]$
[mrvalieva@fedora lab6]$ nasm -f elf lab6-2.asm
[mrvalieva@fedora lab6]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[mrvalieva@fedora lab6]$ ./lab6-2
106
[mrvalieva@fedora lab6]$
[mrvalieva@fedora lab6]$
```

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.

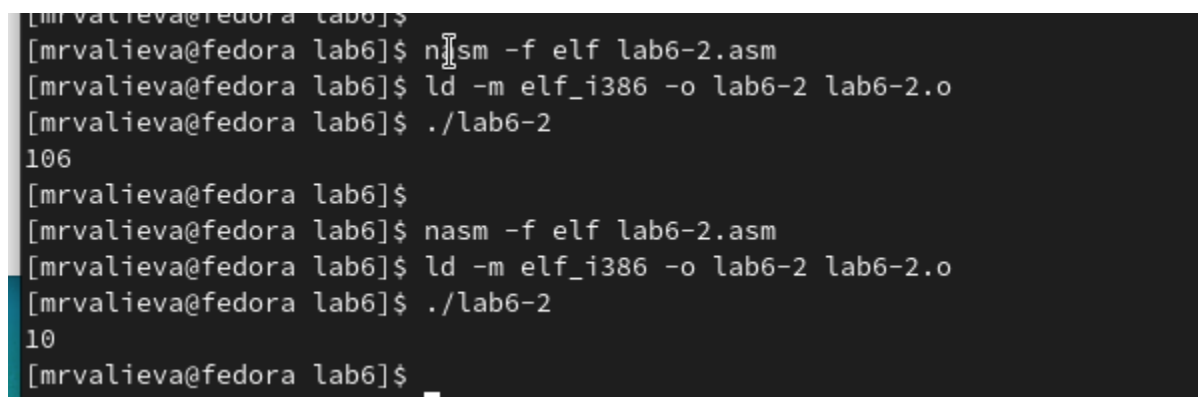
5. Аналогично предыдущему примеру изменим символы на числа. (рис. 4.7, 4.8)

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



```
mc [mrvalieva@fedora]
lab6-2.asm [----] 9 L:[ 1+ 8 9/ 9]
#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



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

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)

```

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

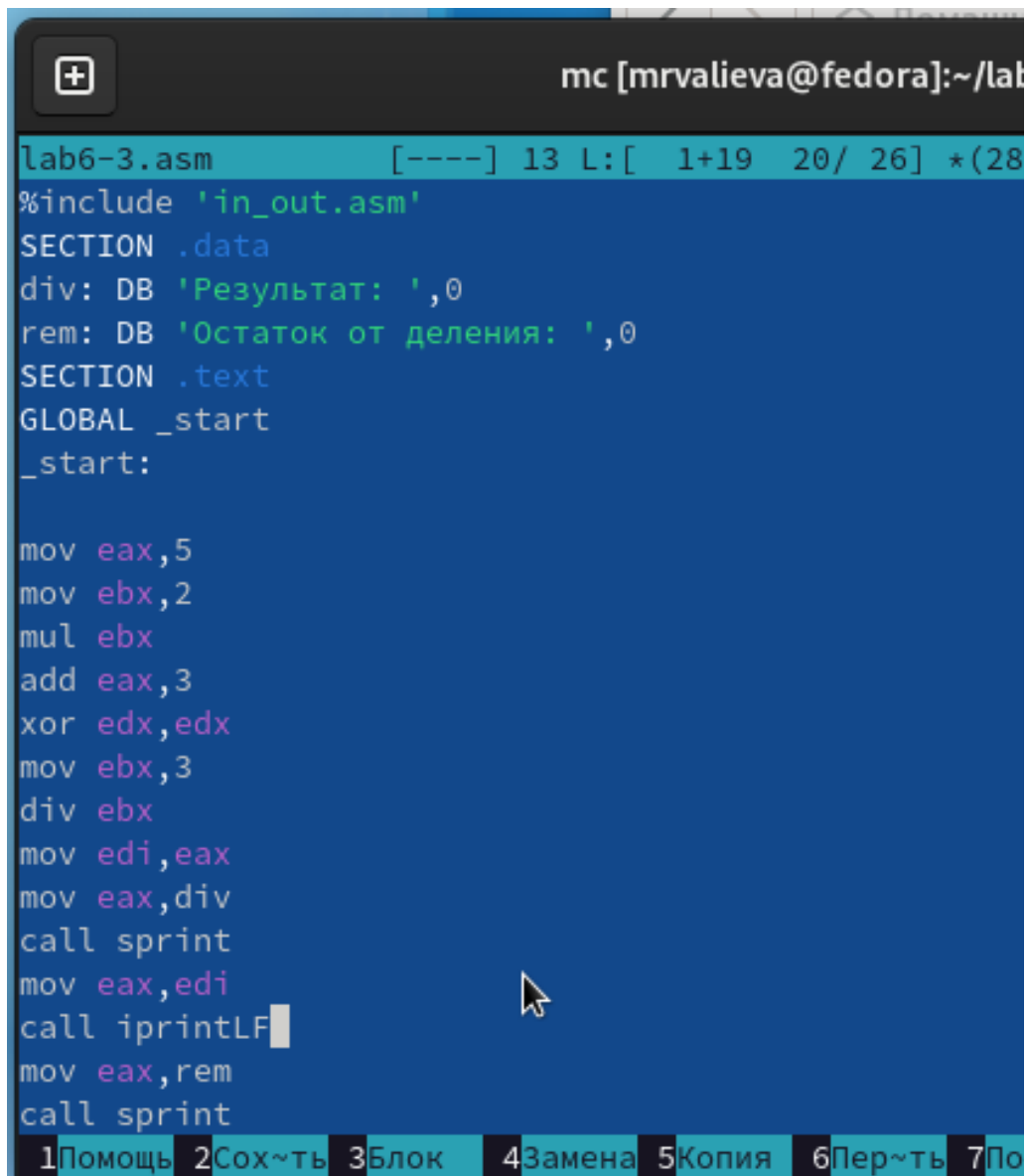
```

Рис. 4.9: Работа программы

6. 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)



```
mc [mrvalieva@fedora]:~/lab
lab6-3.asm [----] 13 L:[ 1+19 20/ 26] *(28
#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

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7По
```

Рис. 4.10: Пример программы

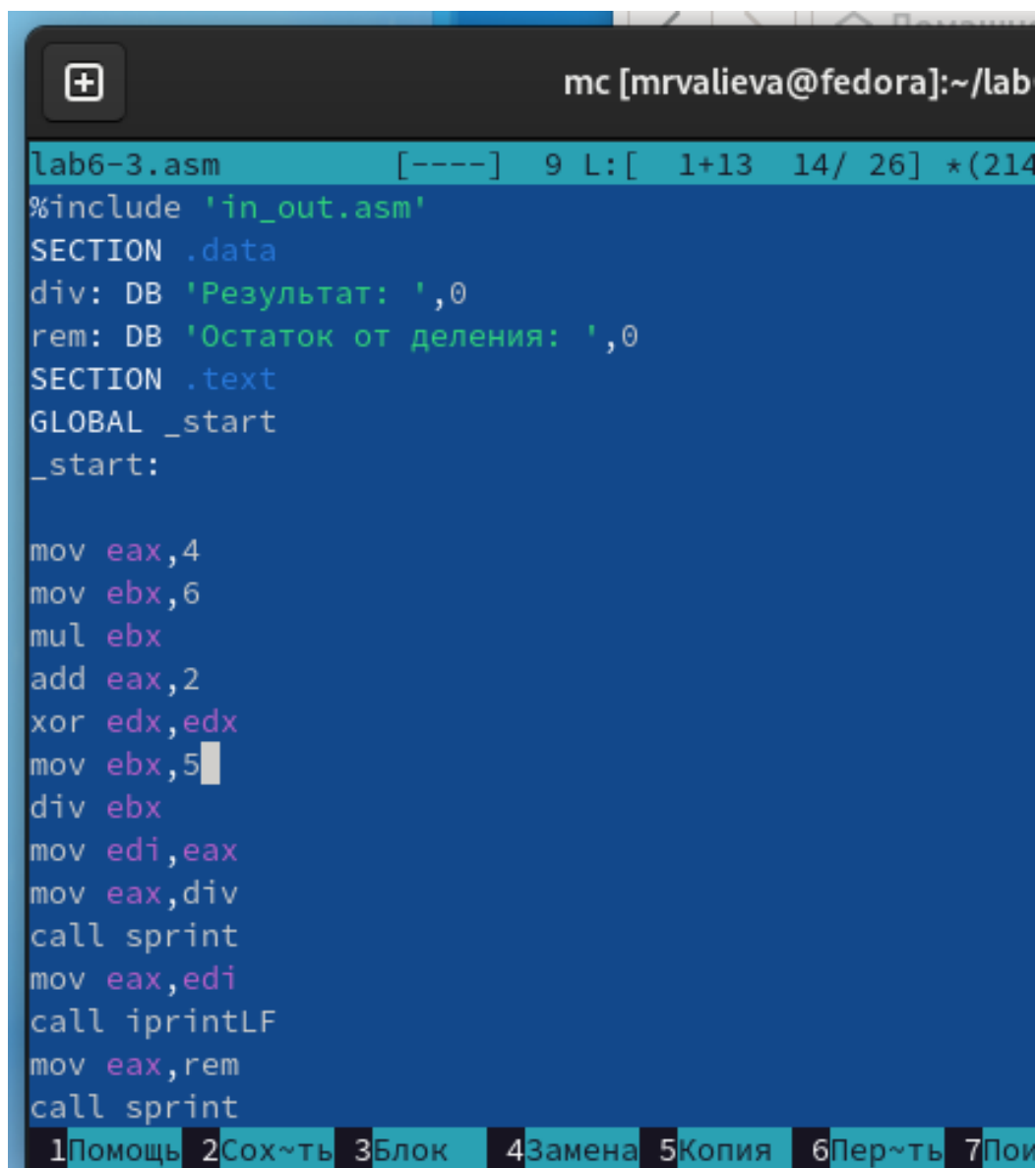
```
[mrvalieva@fedora lab6]$
[mrvalieva@fedora lab6]$ nasm -f elf lab6-3.asm
[mrvalieva@fedora lab6]$ ld -m elf_i386 -o lab6-3 lab6-3.o
[mrvalieva@fedora lab6]$ ./lab6-3
Результат: 4
Остаток от деления: 1
[mrvalieva@fedora lab6]$
[mrvalieva@fedora lab6]$
```

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)



```
mc [mrvalieva@fedora]:~/lab
lab6-3.asm [----] 9 L: [ 1+13 14/ 26] *(214
#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

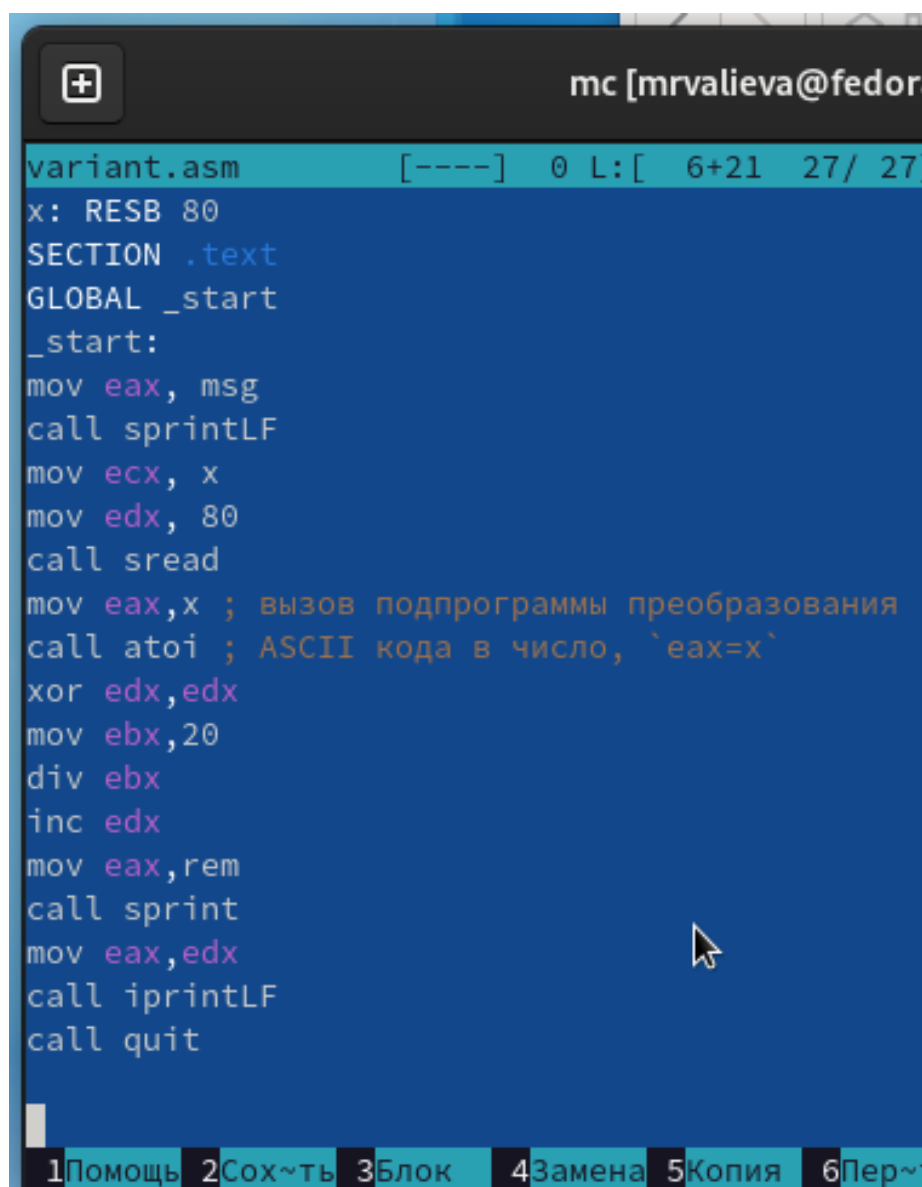
1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск
```

Рис. 4.12: Пример программы

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

Рис. 4.13: Работа программы

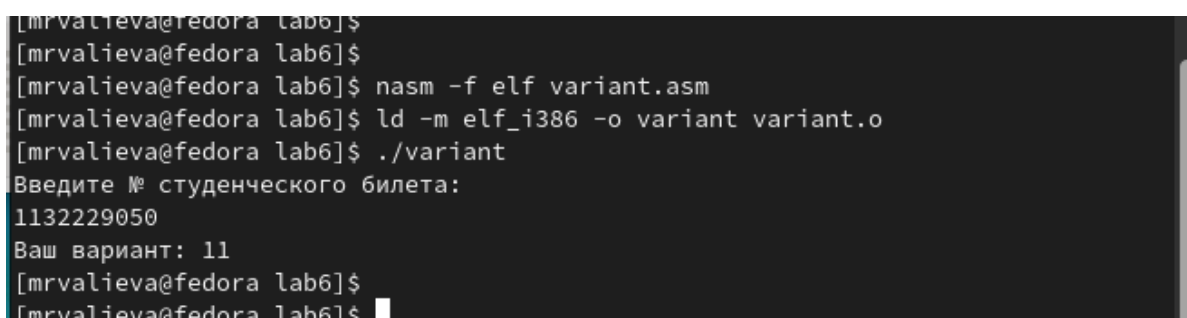
7. В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета, работающую по следующему алгоритму: (рис. 4.14, рис. 4.15)



```
variant.asm  [----]  0  L:[ 6+21  27/ 27
x: RESB 80
SECTION .text
GLOBAL _start
_start:
mov  eax, msg
call sprintf
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 sprintf
mov  eax,edx
call iprintLF
call quit
```

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Пер...

Рис. 4.14: Пример программы



```
[mrvalieva@fedora lab6]$
[mrvalieva@fedora lab6]$
[mrvalieva@fedora lab6]$ nasm -f elf variant.asm
[mrvalieva@fedora lab6]$ ld -m elf_i386 -o variant variant.o
[mrvalieva@fedora lab6]$ ./variant
Введите № студенческого билета:
1132229050
Ваш вариант: 11
[mrvalieva@fedora lab6]$
[mrvalieva@fedora lab6]$
```

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`

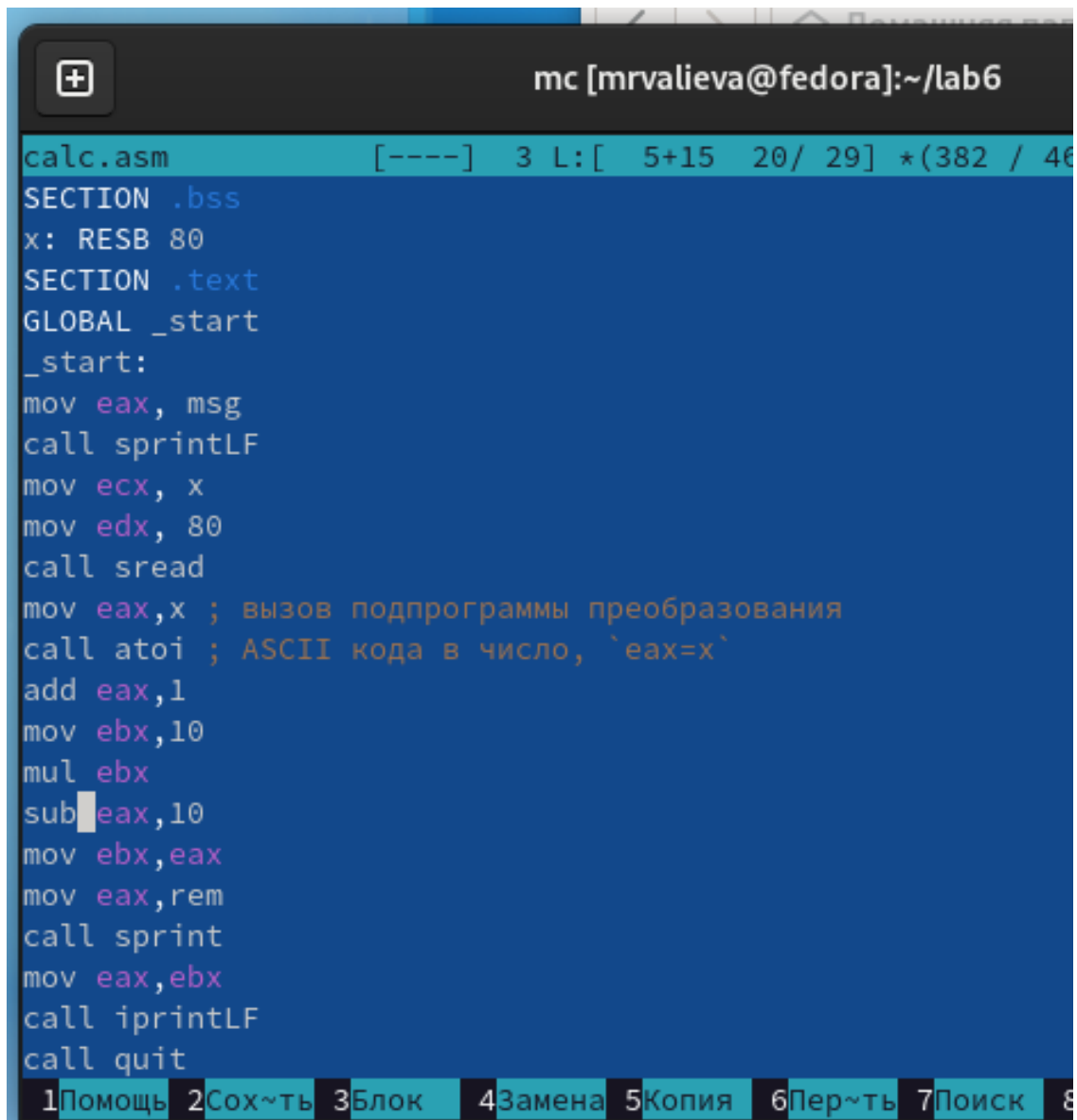
Считывает значение студбилета в переменную X из консоли

- 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 11 -

$$10(x + 1) - 10$$

для $x=1$ и 7



```
mc [mrvalieva@fedora]:~/lab6
calc.asm [----] 3 L:[ 5+15 20/ 29] *(382 / 46
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`
add eax, 1
mov ebx, 10
mul ebx
sub eax, 10
mov ebx, eax
mov eax, rem
call sprint
mov eax, ebx
call iprintLF
call quit
1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск 8
```

Рис. 4.16: Пример программы

```
[mrvalieva@fedora lab6]$  
[mrvalieva@fedora lab6]$ nasm -f elf calc.asm  
[mrvalieva@fedora lab6]$ ld -m elf_i386 -o calc calc.o  
[mrvalieva@fedora lab6]$ ./calc  
Введите X  
1  
выражение = : 10  
[mrvalieva@fedora lab6]$ ./calc  
Введите X  
7  
выражение = : 70  
[mrvalieva@fedora lab6]$  
[mrvalieva@fedora lab6]$
```

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