

Отчёта по лабораторной работе 7

Освоение арифметических инструкций языка ассемблера NASM.

Туем Гислен НКАбд-03-22

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Выводы	23
	Список литературы	24

Список иллюстраций

4.1	Пример программы	9
4.2	Работа программы	10
4.3	Пример программы	11
4.4	Работа программы	12
4.5	Пример программы	13
4.6	Работа программы	13
4.7	Пример программы	14
4.8	Работа программы	14
4.9	Работа программы	15
4.10	Пример программы	16
4.11	Работа программы	16
4.12	Пример программы	17
4.13	Работа программы	18
4.14	Пример программы	19
4.15	Работа программы	19
4.16	Пример программы	21
4.17	Работа программы	22

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

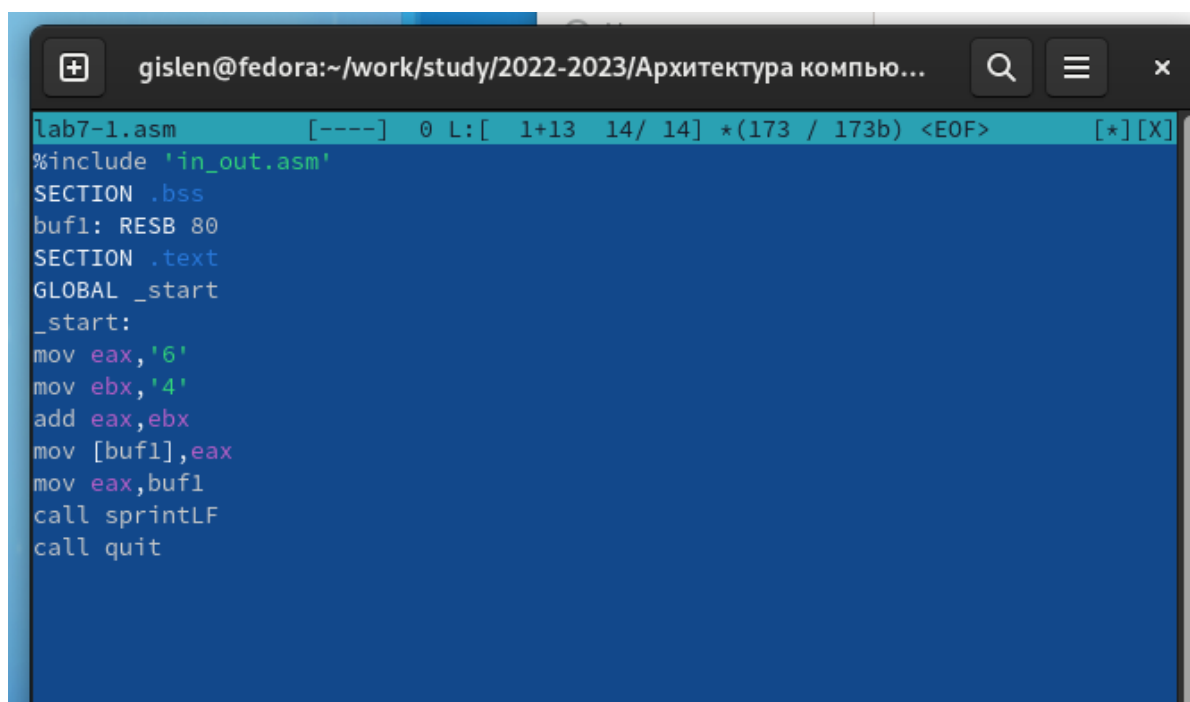
регистра `rax` (или явно `subq %rax, ll`, где указывается, что операнды имеют 64-размер), и результат помещается в `ll`; `subw go, %ax` – вычитание из содержимого `ax` числа по адресу `go`, результат помещается в `ax`; `sbb %rdx, %rax` – вычитание с дополнительным вычитанием флага переноса (из числа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

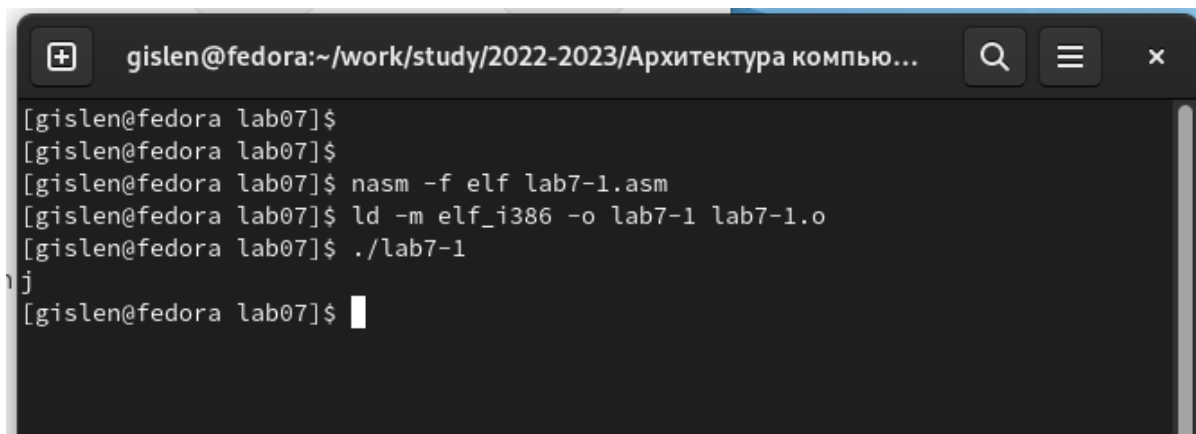
4 Выполнение лабораторной работы

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 еах. (рис. 4.1, 4.2)



```
lab7-1.asm [----] 0 L: [ 1+13 14/ 14] *(173 / 173b) <EOF> [*] [X]
#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 gray text. The window title is "gislen@fedora:~/work/study/2022-2023/Архитектура компью...". The terminal shows a series of commands and their outputs. The commands are: "nasm -f elf lab7-1.asm", "ld -m elf_i386 -o lab7-1 lab7-1.o", and "./lab7-1". The output of the first command is "j", and the output of the second command is "j". The terminal is currently at the prompt "[gislen@fedora lab07]\$".

```
gislen@fedora:~/work/study/2022-2023/Архитектура компью...
[gislen@fedora lab07]$
[gislen@fedora lab07]$ nasm -f elf lab7-1.asm
[gislen@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[gislen@fedora lab07]$ ./lab7-1
j
[gislen@fedora lab07]$
```

Рис. 4.2: Работа программы

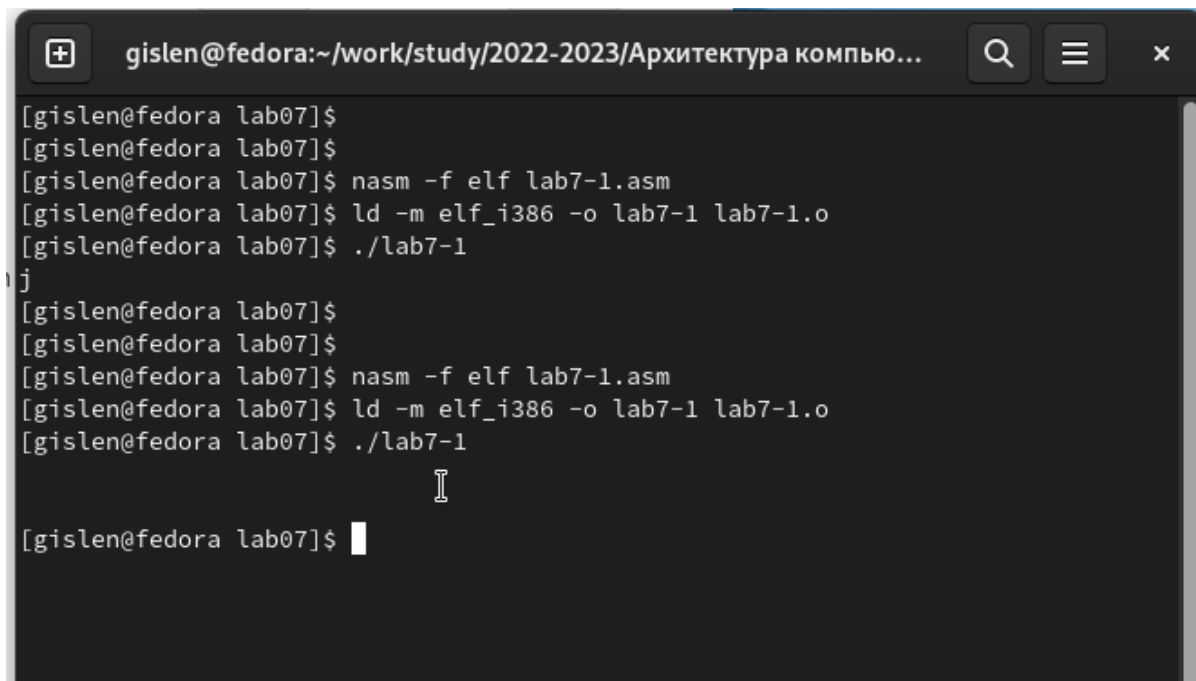
3. Далее изменим текст программы и вместо символов, запишем в регистры числа. Исправьте текст программы (Листинг 1) следующим образом: (рис. 4.3, 4.4)



```
lab7-1.asm [----] 9 L: [ 1+ 6 7/ 14
#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
```

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перезагрузить

Рис. 4.3: Пример программы

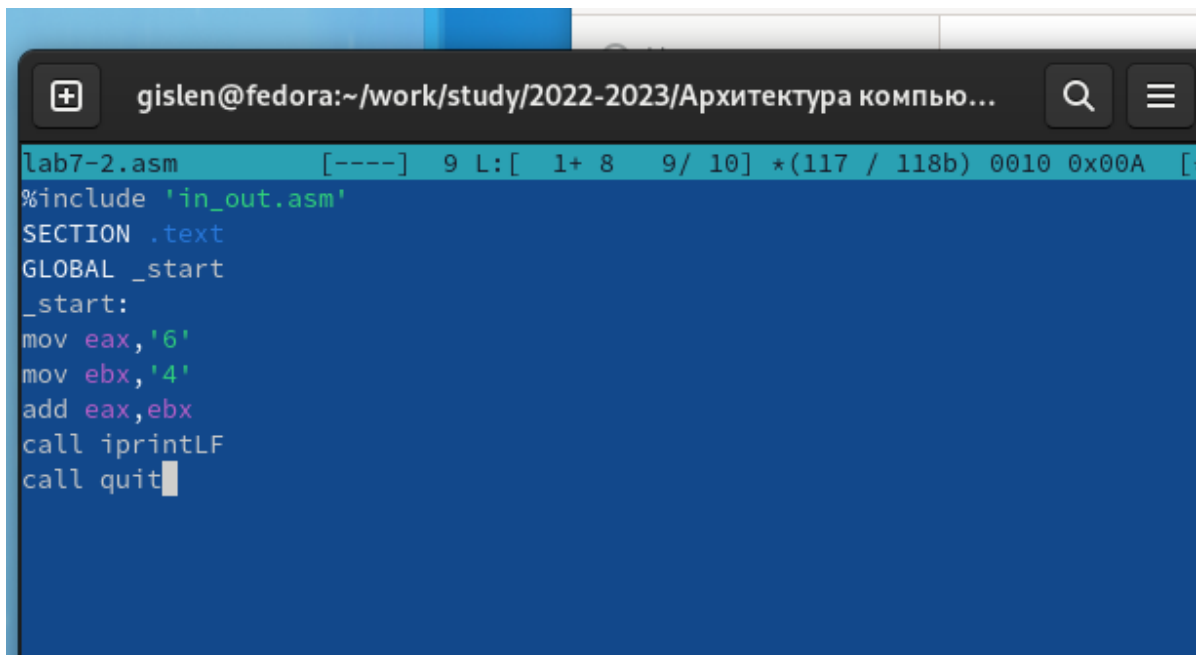
A terminal window with a dark background and light text. The window title is "gislen@fedora:~/work/study/2022-2023/Архитектура компью...". The terminal shows a sequence of commands: two empty prompts, then "nasm -f elf lab7-1.asm", then "ld -m elf_i386 -o lab7-1 lab7-1.o", then "./lab7-1". After the first execution, there is a faint "j" on the line. The commands are repeated. The cursor is at the end of the last prompt.

```
[gislen@fedora lab07]$  
[gislen@fedora lab07]$  
[gislen@fedora lab07]$ nasm -f elf lab7-1.asm  
[gislen@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o  
[gislen@fedora lab07]$ ./lab7-1  
j  
[gislen@fedora lab07]$  
[gislen@fedora lab07]$  
[gislen@fedora lab07]$ nasm -f elf lab7-1.asm  
[gislen@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o  
[gislen@fedora lab07]$ ./lab7-1  
[gislen@fedora lab07]$
```

Рис. 4.4: Работа программы

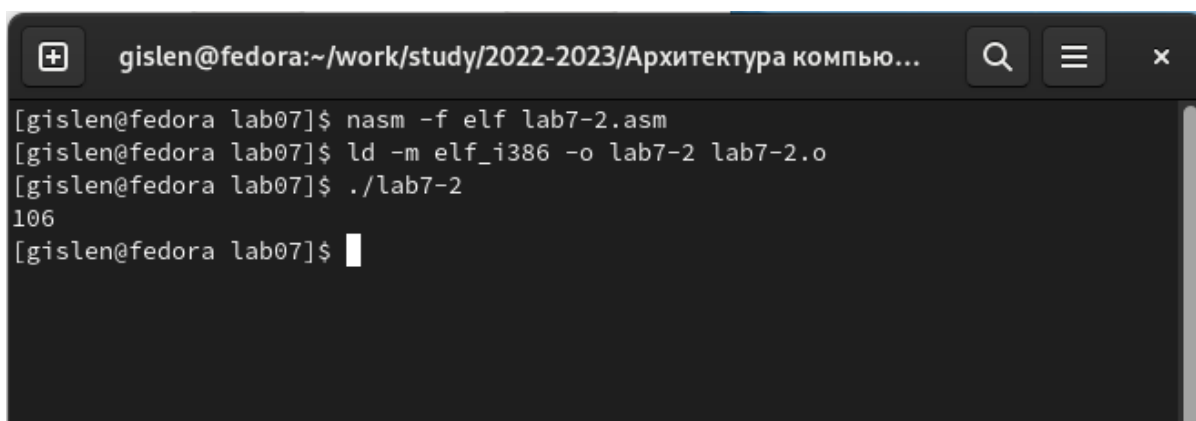
Никакой символ не виден, но он есть. Это возврат каретки LF.

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)

A screenshot of a code editor window. The title bar shows the user 'gislen@fedora' and the path '~/work/study/2022-2023/Архитектура компью...'. The code is in assembly format for x86-64. It includes 'in_out.asm', sets the section to '.text', and declares a global '_start'. The program starts at '_start:', moves the value 6 into 'eax', moves the value 4 into 'ebx', adds 'ebx' to 'eax', calls 'iprintLF' to print the result, and finally calls 'quit' to exit.

```
lab7-2.asm [----] 9 L: [ 1+ 8 9/ 10] *(117 / 118b) 0010 0x00A [
#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

A screenshot of a terminal window. The title bar shows the user 'gislen@fedora' and the path '~/work/study/2022-2023/Архитектура компью...'. The terminal shows the commands to assemble 'lab7-2.asm' using 'nasm' into 'lab7-2.o', link it with 'ld' into 'lab7-2', and then execute it. The output of the program is the number '106'.

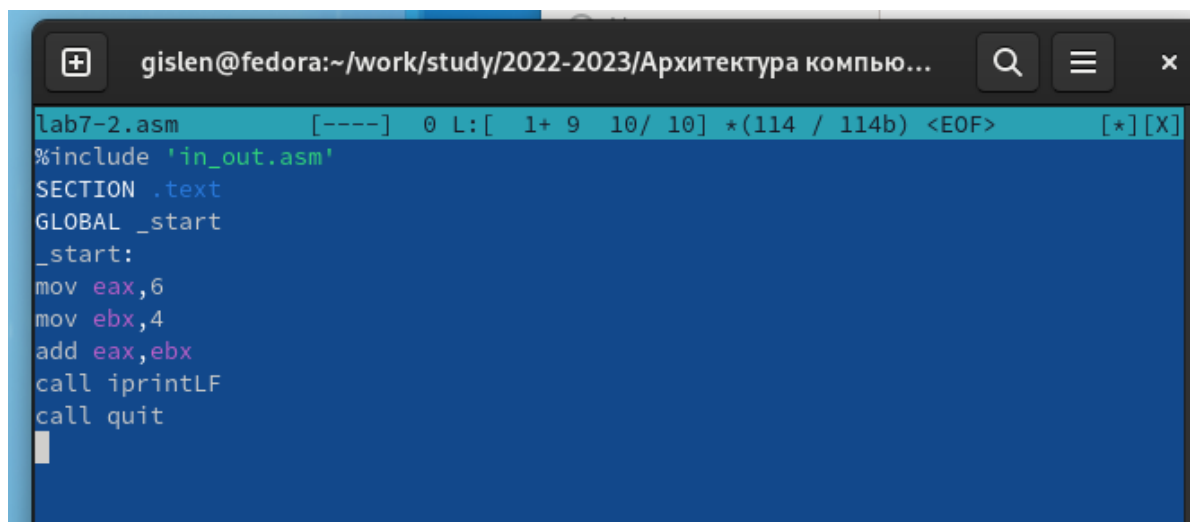
```
[gislen@fedora lab07]$ nasm -f elf lab7-2.asm
[gislen@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[gislen@fedora lab07]$ ./lab7-2
106
[gislen@fedora lab07]$
```

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.

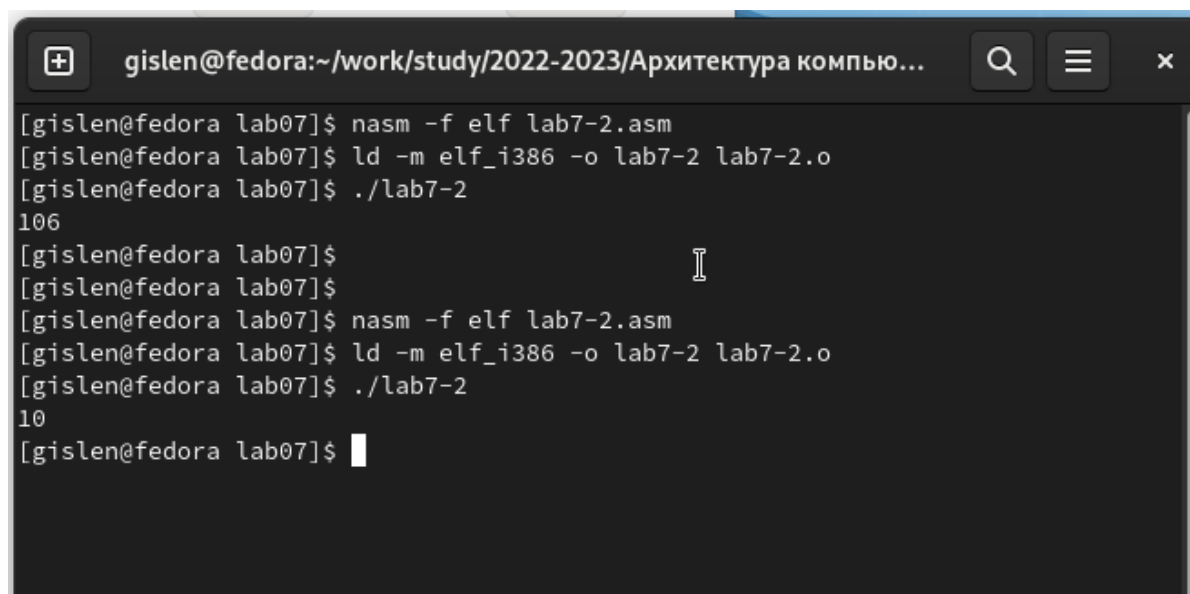
5. Аналогично предыдущему примеру изменим символы на числа. (рис. 4.7, 4.8)

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



```
lab7-2.asm [----] 0 L: [ 1+ 9 10/ 10] *(114 / 114b) <EOF> [*][X]
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprintLF
call quit
```

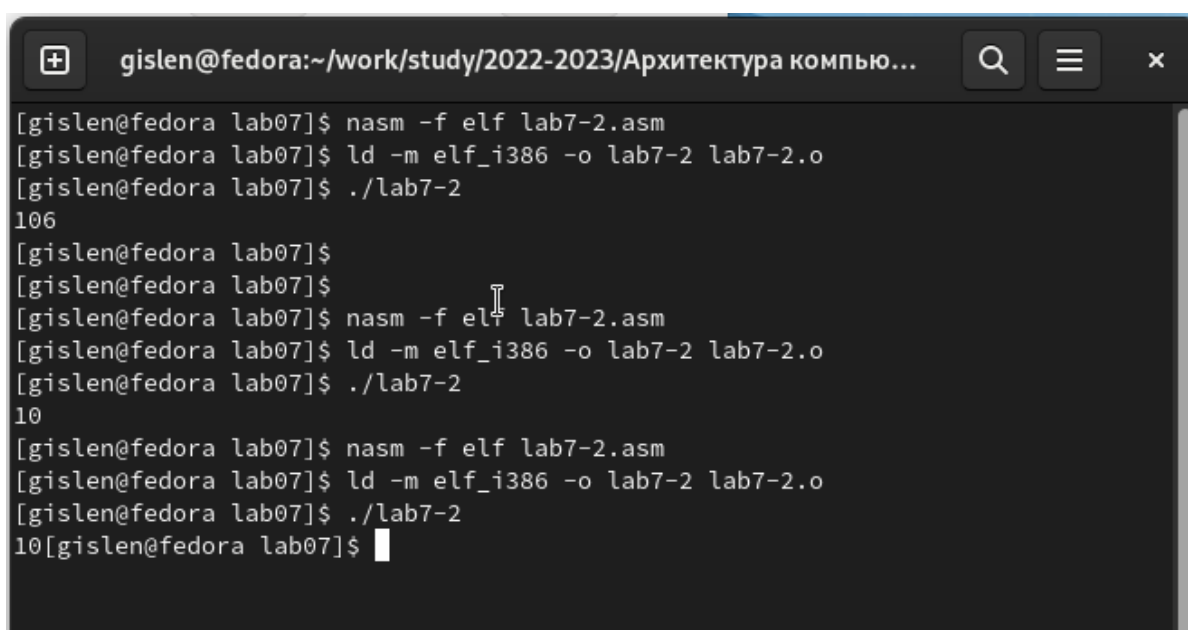
Рис. 4.7: Пример программы



```
[gislen@fedora lab07]$ nasm -f elf lab7-2.asm
[gislen@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[gislen@fedora lab07]$ ./lab7-2
106
[gislen@fedora lab07]$
[gislen@fedora lab07]$
[gislen@fedora lab07]$ nasm -f elf lab7-2.asm
[gislen@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[gislen@fedora lab07]$ ./lab7-2
10
[gislen@fedora lab07]$
```

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)

A terminal window with a dark background and light text. The window title is "gislen@fedora:~/work/study/2022-2023/Архитектура компью...". The terminal shows a series of commands and their outputs. The commands are: "nasm -f elf lab7-2.asm", "ld -m elf_i386 -o lab7-2 lab7-2.o", and "./lab7-2". The outputs are "106", "10", and "10" respectively. The terminal is in a directory named "lab07".

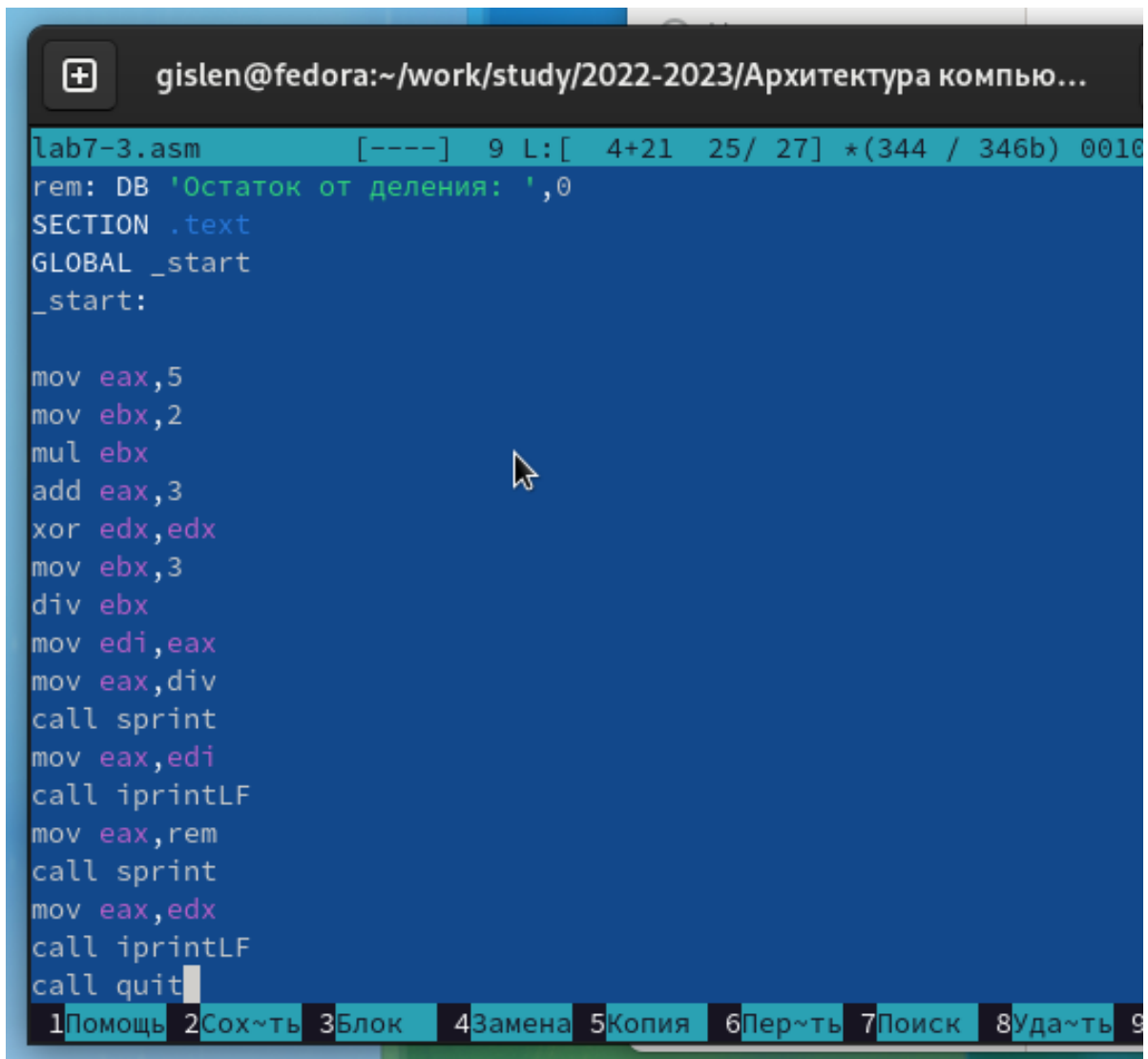
```
[gislen@fedora lab07]$ nasm -f elf lab7-2.asm
[gislen@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[gislen@fedora lab07]$ ./lab7-2
106
[gislen@fedora lab07]$
[gislen@fedora lab07]$
[gislen@fedora lab07]$ nasm -f elf lab7-2.asm
[gislen@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[gislen@fedora lab07]$ ./lab7-2
10
[gislen@fedora lab07]$ nasm -f elf lab7-2.asm
[gislen@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[gislen@fedora lab07]$ ./lab7-2
10
[gislen@fedora lab07]$
```

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)

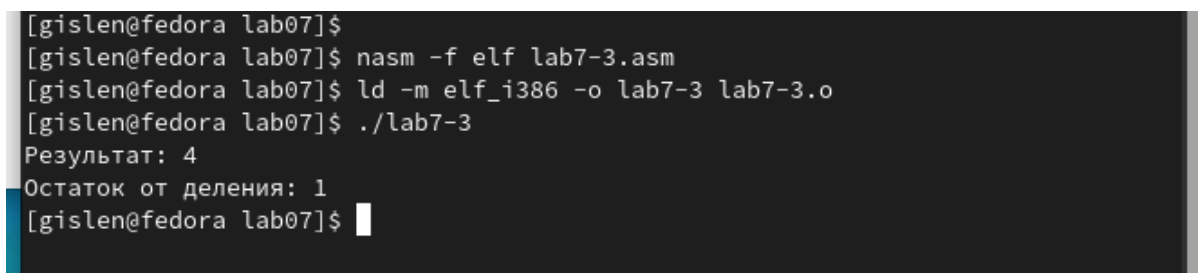


```
lab7-3.asm [----] 9 L: [ 4+21 25/ 27] *(344 / 346b) 0010
rem: DB 'Остаток от деления: ',0
SECTION .text
GLOBAL _start
_start:

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
```

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск 8Удалить 9

Рис. 4.10: Пример программы



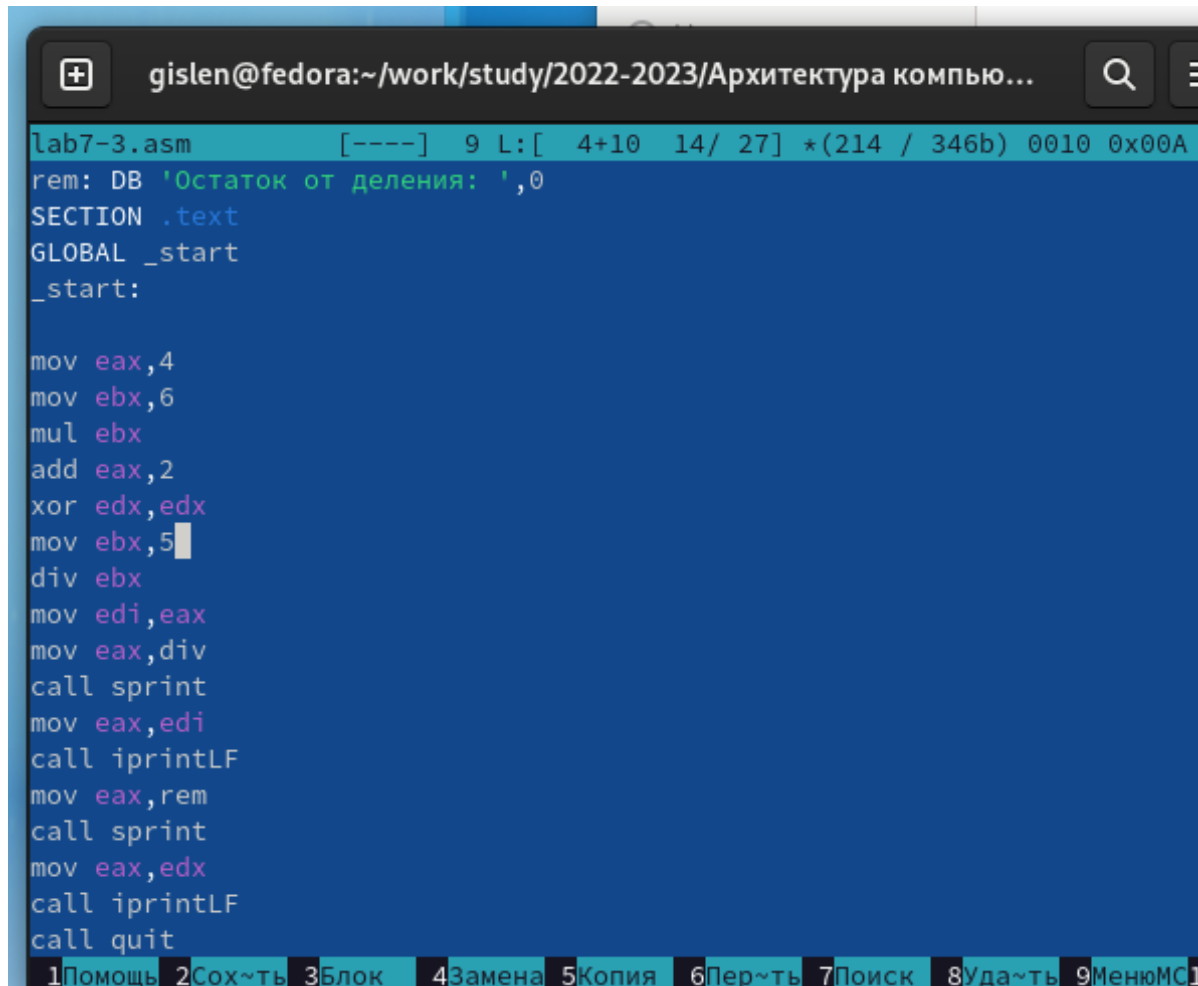
```
[gislen@fedora lab07]$
[gislen@fedora lab07]$ nasm -f elf lab7-3.asm
[gislen@fedora lab07]$ ld -m elf_i386 -o lab7-3 lab7-3.o
[gislen@fedora lab07]$ ./lab7-3
Результат: 4
Остаток от деления: 1
[gislen@fedora lab07]$
```

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)



```
gislen@fedora:~/work/study/2022-2023/Архитектура компью...
lab7-3.asm  [----]  9 L:[  4+10  14/ 27] *(214 / 346b) 0010 0x00A
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

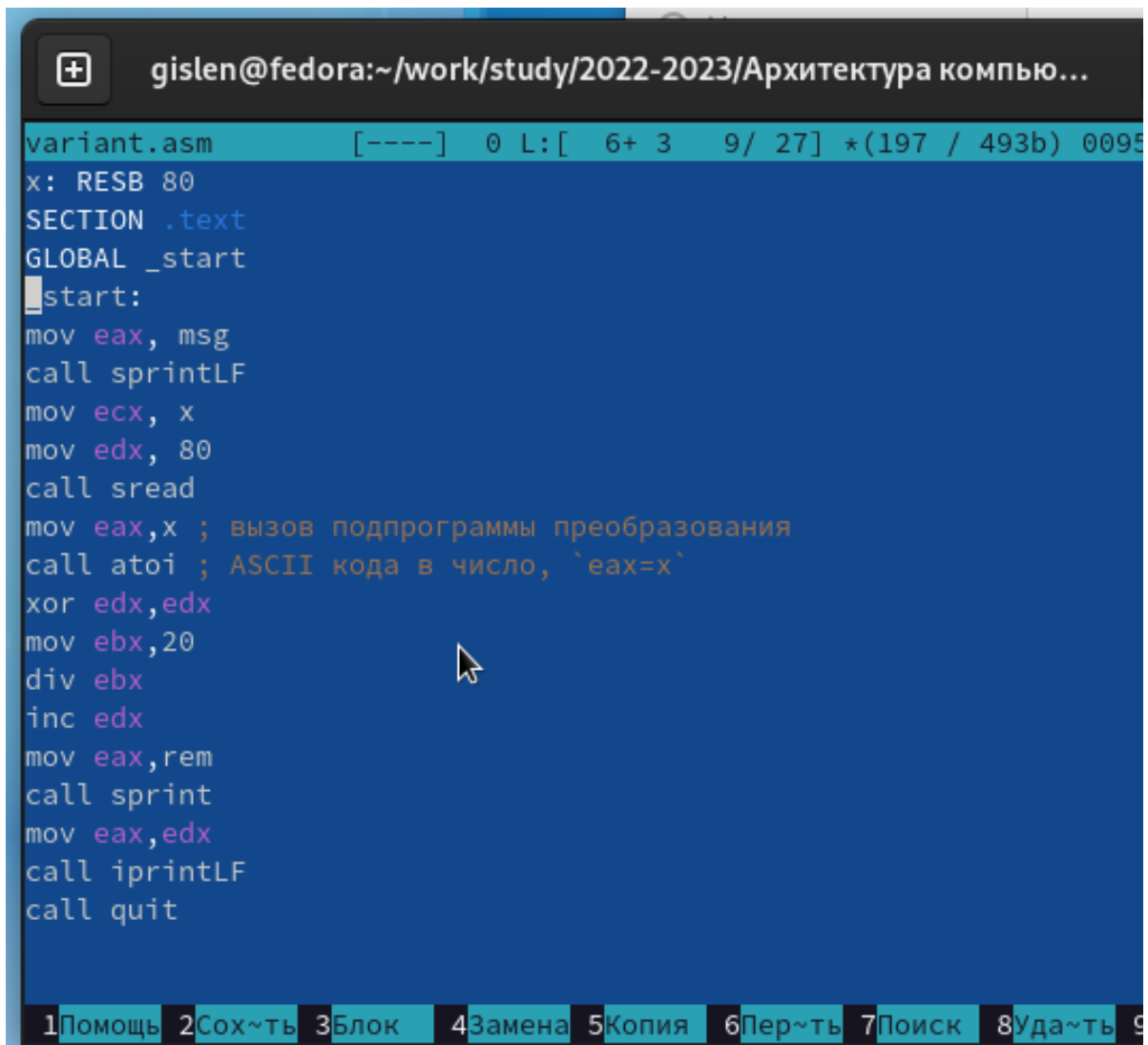
1Помощь 2Сох~ть 3Блок 4Замена 5Копия 6Пер~ть 7Поиск 8Уда~ть 9МенюМС1
```

Рис. 4.12: Пример программы

```
10[gislen@fedora lab07]$  
[gislen@fedora lab07]$  
[gislen@fedora lab07]$ nasm -f elf lab7-3.asm  
[gislen@fedora lab07]$ ld -m elf_i386 -o lab7-3 lab7-3.o  
[gislen@fedora lab07]$ ./lab7-3  
Результат: 4  
Остаток от деления: 1  
[gislen@fedora lab07]$ nasm -f elf lab7-3.asm  
[gislen@fedora lab07]$ ld -m elf_i386 -o lab7-3 lab7-3.o  
[gislen@fedora lab07]$ ./lab7-3  
Результат: 5  
Остаток от деления: 1  
[gislen@fedora lab07]$  
[gislen@fedora lab07]$  
[gislen@fedora lab07]$  
[gislen@fedora lab07]$
```

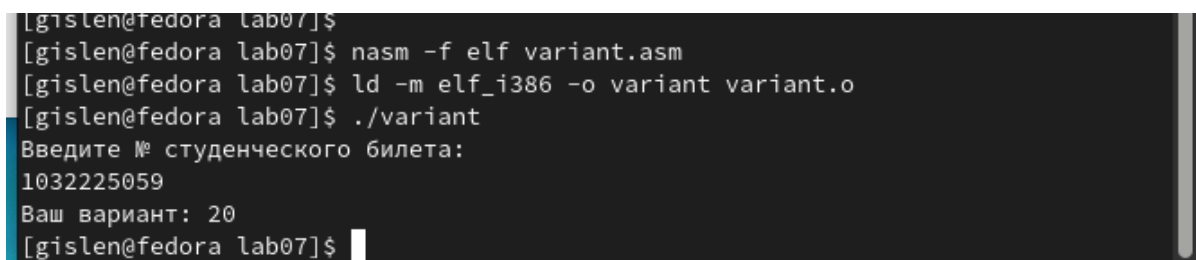
Рис. 4.13: Работа программы

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+ 3  9/ 27] *(197 / 493b) 0095
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
```

Рис. 4.14: Пример программы



```
[gislen@fedora lab07]$
[gislen@fedora lab07]$ nasm -f elf variant.asm
[gislen@fedora lab07]$ ld -m elf_i386 -o variant variant.o
[gislen@fedora lab07]$ ./variant
Введите № студенческого билета:
1032225059
Ваш вариант: 20
[gislen@fedora lab07]$
```

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

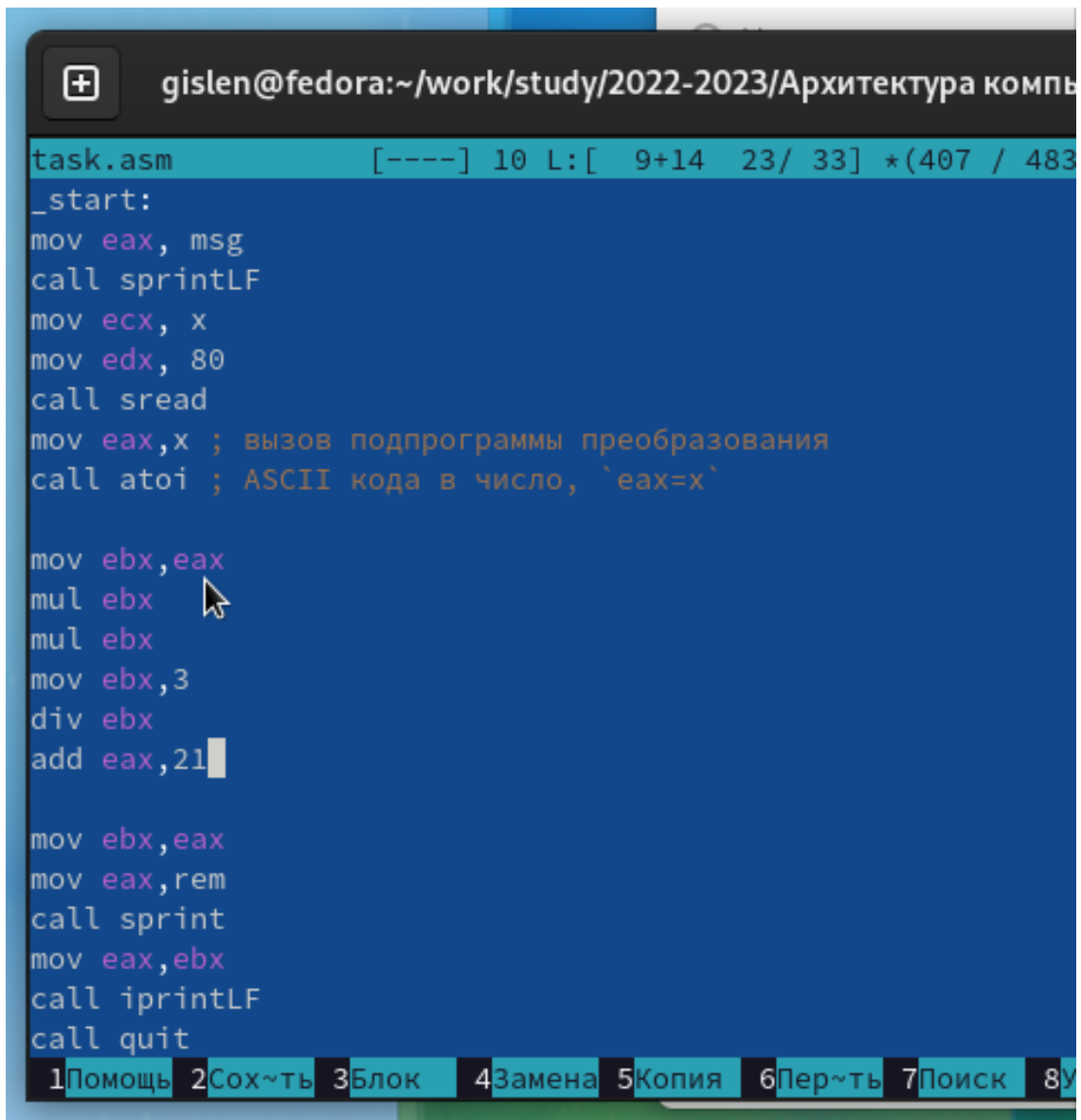
- Для чего используется инструкция “`call atoi`”? – эта подпрограмма переводит введенные символы в числовой формат
- Какие строки листинга 7.4 отвечают за вычисления варианта? `xor edx, edx` `mov ebx, 20` `div ebx`
- В какой регистр записывается остаток от деления при выполнении инструкции “`div ebx`”? 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 вычислений `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 x1 и x2 из 6.3. (рис. 4.16, рис. 4.17)

Получили вариант 20 -

$$(x^3/3) + 21$$

для x=1 и 3



```
gislen@fedora:~/work/study/2022-2023/Архитектура компь
task.asm [----] 10 L:[ 9+14 23/ 33] *(407 / 483
_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`

mov ebx,eax
mul ebx
mul ebx
mov ebx,3
div ebx
add eax,21

mov ebx,eax
mov eax,rem
call sprint
mov eax,ebx
call iprintLF
call quit

1Помощь 2Сохранить 3Блок 4Замена 5Копия 6Перейти 7Поиск 8У
```

Рис. 4.16: Пример программы

```
[gislen@fedora lab07]$  
[gislen@fedora lab07]$ nasm -f elf task.asm  
[gislen@fedora lab07]$ ld -m elf_i386 -o task task.o  
[gislen@fedora lab07]$ ./task  
Введите X  
1  
выражение = : 21  
[gislen@fedora lab07]$ ./task  
Введите X  
3  
выражение = : 30  
[gislen@fedora lab07]$
```

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