

Отчёта по лабораторной работе 7

Освоение арифметических инструкций языка ассемблера NASM.

Мужига Кармел чибангу

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Выводы	19
	Список литературы	20

Список иллюстраций

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

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 с ложения 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

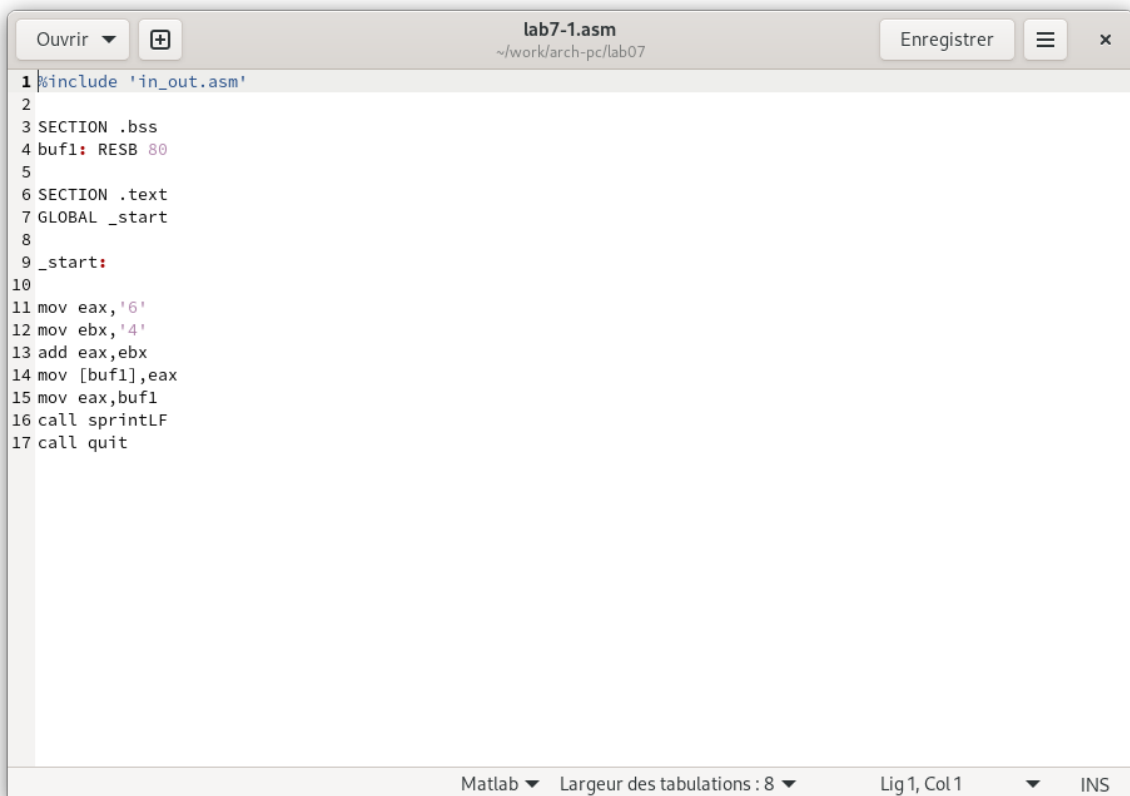
регистра `rax` (или явно `subq %rax, ll`, где указывается, что операнды имеют 64-размер), и результат помещается в `ll`; `subw go, %ax` – вычитание из содержимого `ax` числа по адресу `go`, результат помещается в `ax`; `sbb %rdx, %rax` – вычитание с дополнительным вычитанием флага переноса (из числа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



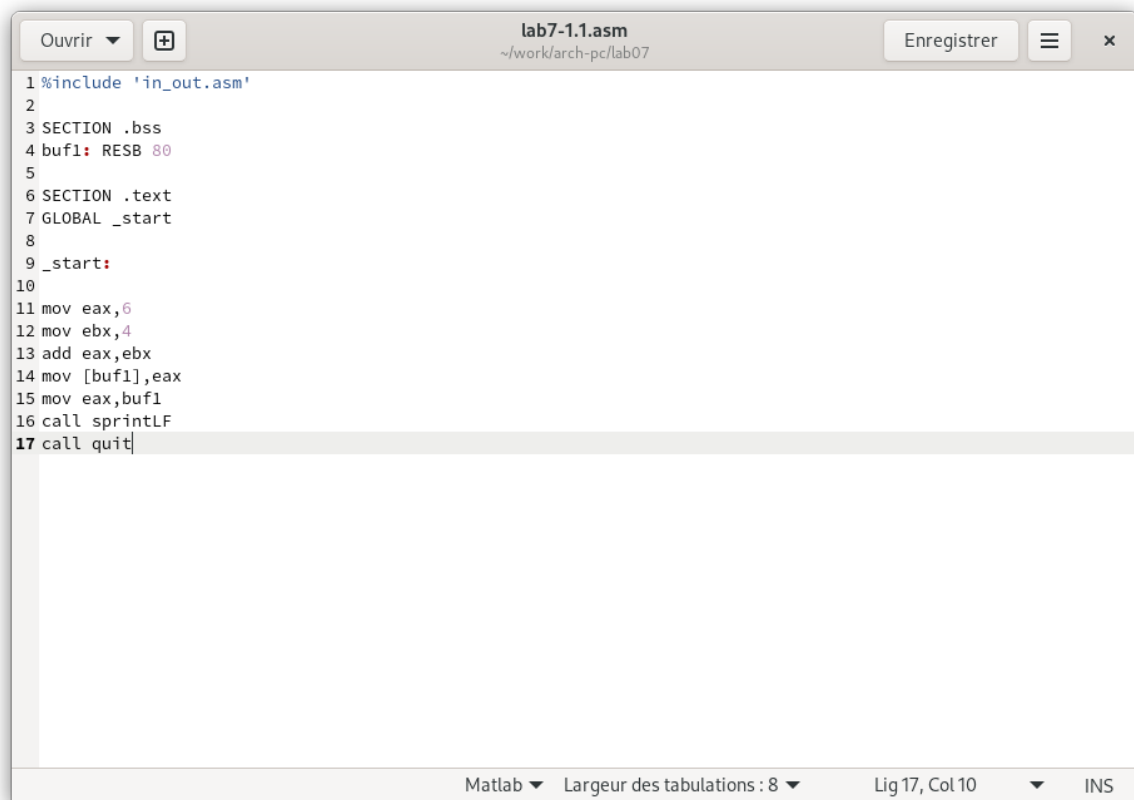
```
1 %include 'in_out.asm'
2
3 SECTION .bss
4 buf1: RESB 80
5
6 SECTION .text
7 GLOBAL _start
8
9 _start:
10
11 mov eax, '6'
12 mov ebx, '4'
13 add eax, ebx
14 mov [buf1], eax
15 mov eax, buf1
16 call sprintf
17 call quit
```

Рис. 4.1: Пример программы

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



```
1 %include 'in_out.asm'
2
3 SECTION .bss
4 buf1: RESB 80
5
6 SECTION .text
7 GLOBAL _start
8
9 _start:
10
11 mov eax,6
12 mov ebx,4
13 add eax,ebx
14 mov [buf1],eax
15 mov eax,buf1
16 call sprintLF
17 call quit
```

Рис. 4.3: Пример программы

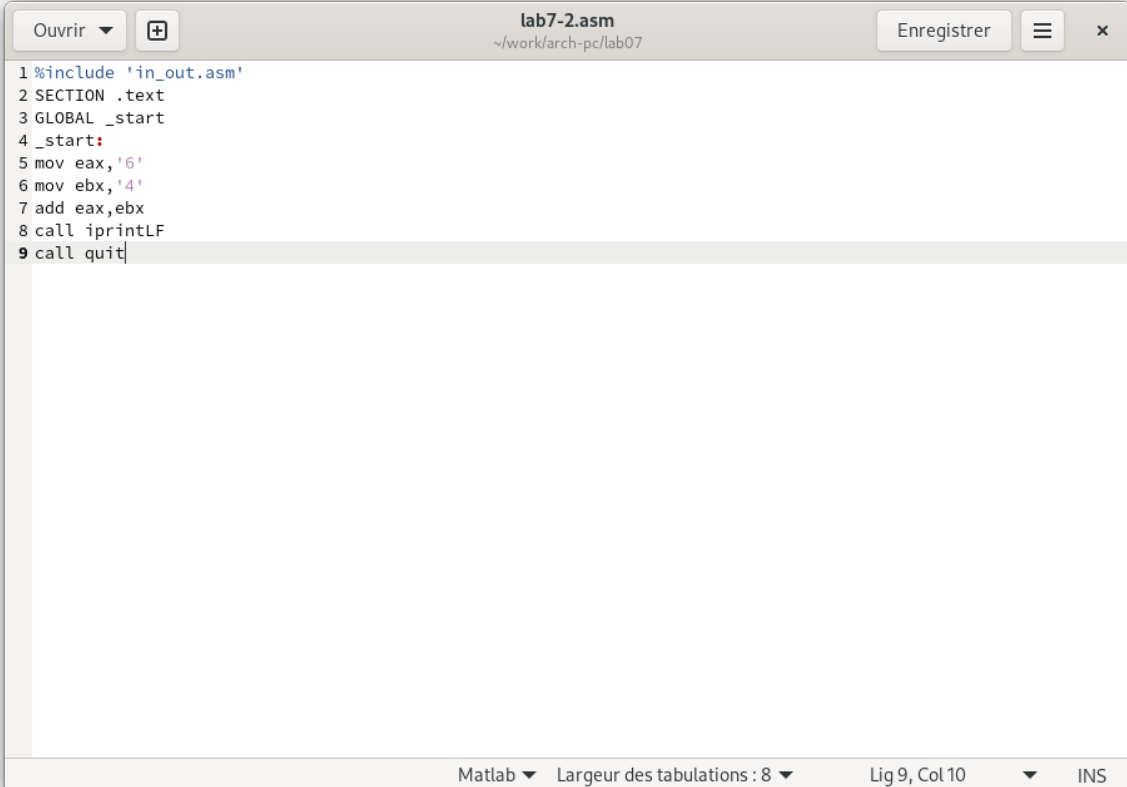
```
[kcmuzhinga@fedora lab07]$ nasm -f elf lab7-1.1.asm
[kcmuzhinga@fedora lab07]$ ld -m elf_i386 -o lab7-1.1 lab7-1.1.o
[kcmuzhinga@fedora lab07]$ ./lab7-1.1

[kcmuzhinga@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)



```
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. 4.5: Пример программы

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

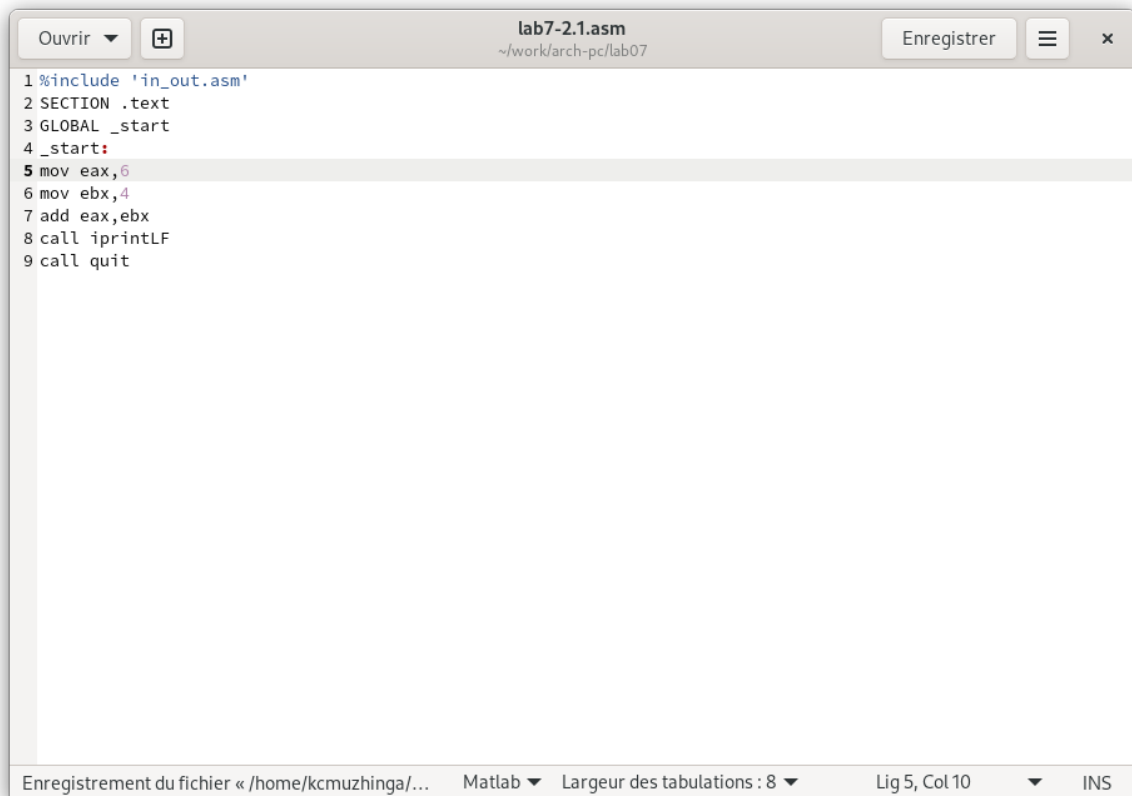
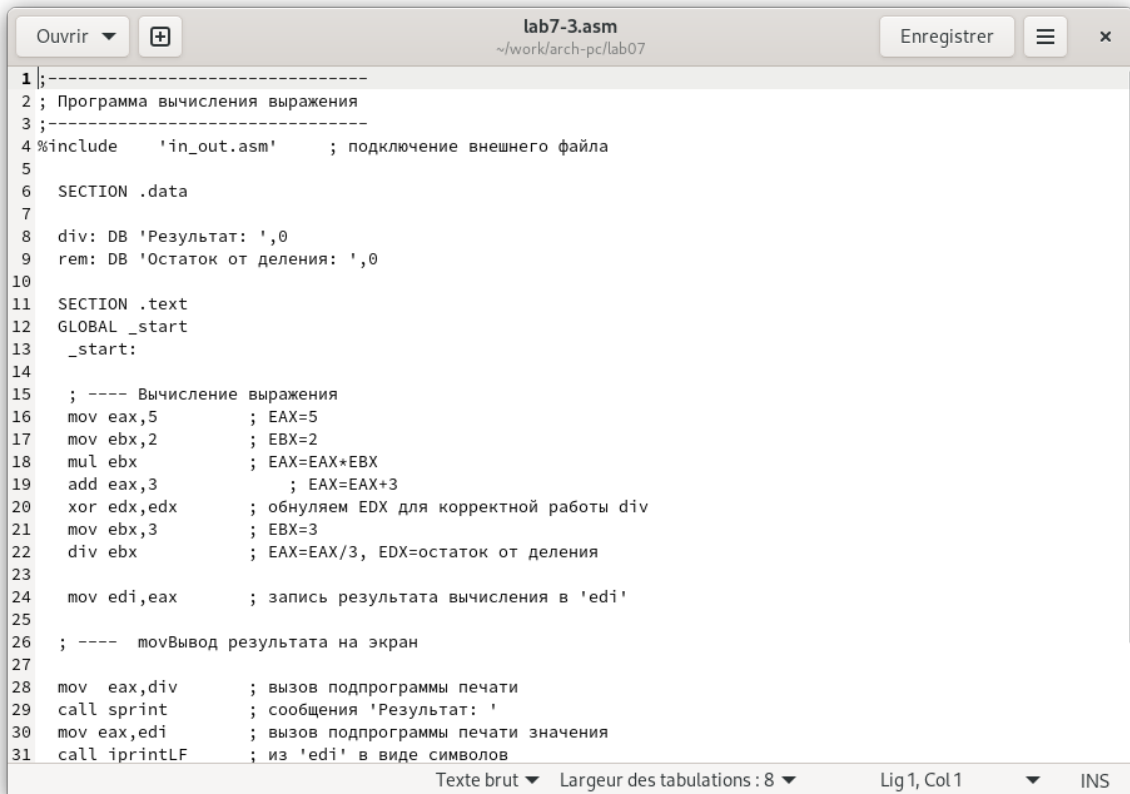


Рис. 4.7: Пример программы

```
[kcmuzhinga@fedora lab07]$ touch lab7-2.1.asm
[kcmuzhinga@fedora lab07]$ nasm -f elf lab7-2.1.asm
[kcmuzhinga@fedora lab07]$ ld -m elf_i386 -o lab7-2.1 lab7-2.1.o
[kcmuzhinga@fedora lab07]$ ./lab7-2.1
10
[kcmuzhinga@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)



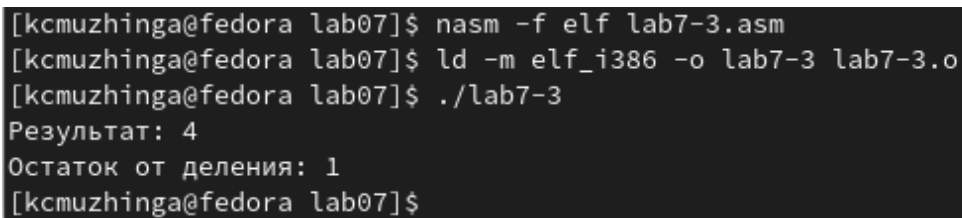
```
1;-----
2; Программа вычисления выражения
3;-----
4%include 'in_out.asm' ; подключение внешнего файла
5
6SECTION .data
7
8div: DB 'Результат: ',0
9rem: DB 'Остаток от деления: ',0
10
11SECTION .text
12GLOBAL _start
13_start:
14
15; ---- Вычисление выражения
16mov eax,5 ; EAX=5
17mov ebx,2 ; EBX=2
18mul ebx ; EAX=EAX*EBX
19add eax,3 ; EAX=EAX+3
20xor edx,edx ; обнуляем EDX для корректной работы div
21mov ebx,3 ; EBX=3
22div ebx ; EAX=EAX/3, EDX=остаток от деления
23
24mov edi,eax ; запись результата вычисления в 'edi'
25
26; ---- вывод результата на экран
27
28mov eax,div ; вызов подпрограммы печати
29call sprint ; сообщения 'Результат: '
30mov eax,edi ; вызов подпрограммы печати значения
31call iprintLF ; из 'edi' в виде символов
```

Рис. 4.9: Работа программы

6. В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения

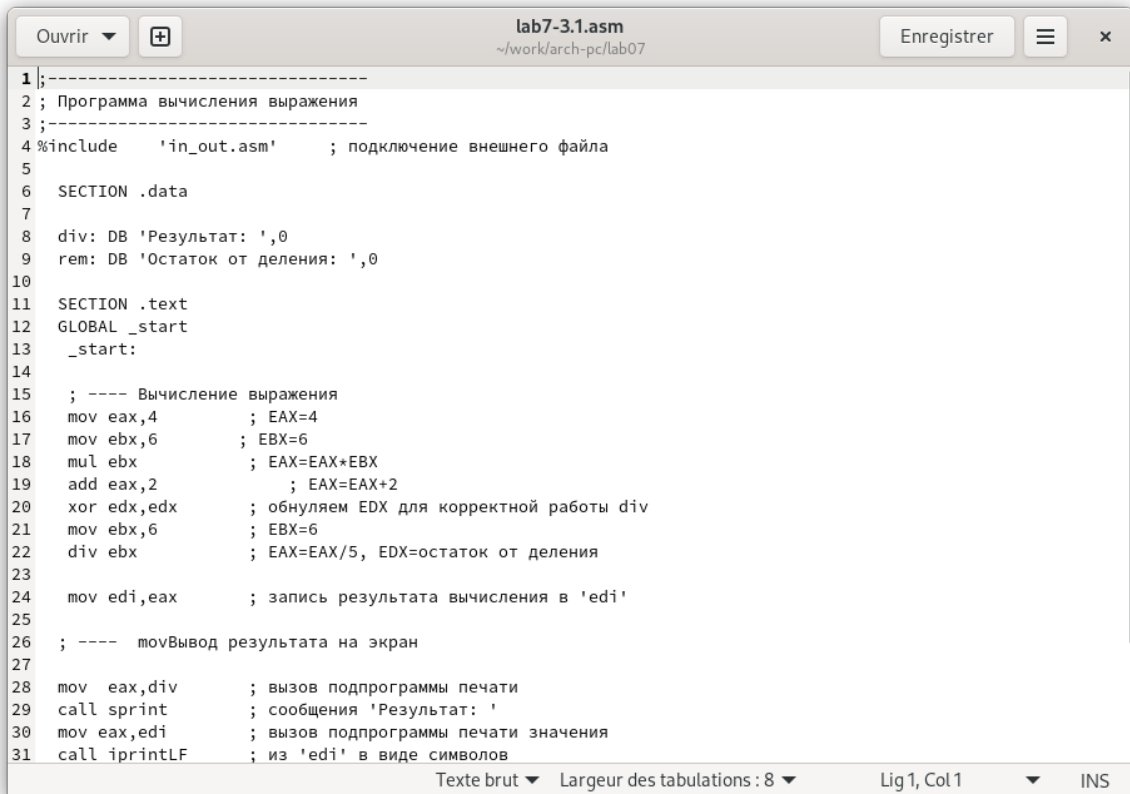
$$f(x) = (5 * 2 + 3) / 3.$$

. (рис. 4.10, рис. 4.11)



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

Рис. 4.10: Пример программы



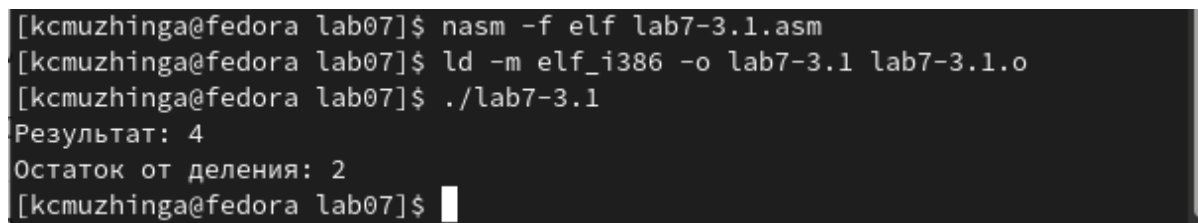
```
1 ;-----
2 ; Программа вычисления выражения
3 ;-----
4 %include 'in_out.asm' ; подключение внешнего файла
5
6 SECTION .data
7
8 div: DB 'Результат: ',0
9 rem: DB 'Остаток от деления: ',0
10
11 SECTION .text
12 GLOBAL _start
13 _start:
14
15 ; ---- Вычисление выражения
16 mov eax,4 ; EAX=4
17 mov ebx,6 ; EBX=6
18 mul ebx ; EAX=EAX*EBX
19 add eax,2 ; EAX=EAX+2
20 xor edx,edx ; обнуляем EDX для корректной работы div
21 mov ebx,6 ; EBX=6
22 div ebx ; EAX=EAX/5, EDX=остаток от деления
23
24 mov edi,eax ; запись результата вычисления в 'edi'
25
26 ; ---- вывод результата на экран
27
28 mov eax,div ; вызов подпрограммы печати
29 call sprint ; сообщения 'Результат: '
30 mov eax,edi ; вызов подпрограммы печати значения
31 call iprintLF ; из 'edi' в виде символов
```

Рис. 4.11: Работа программы

Измените текст программы для вычисления выражения

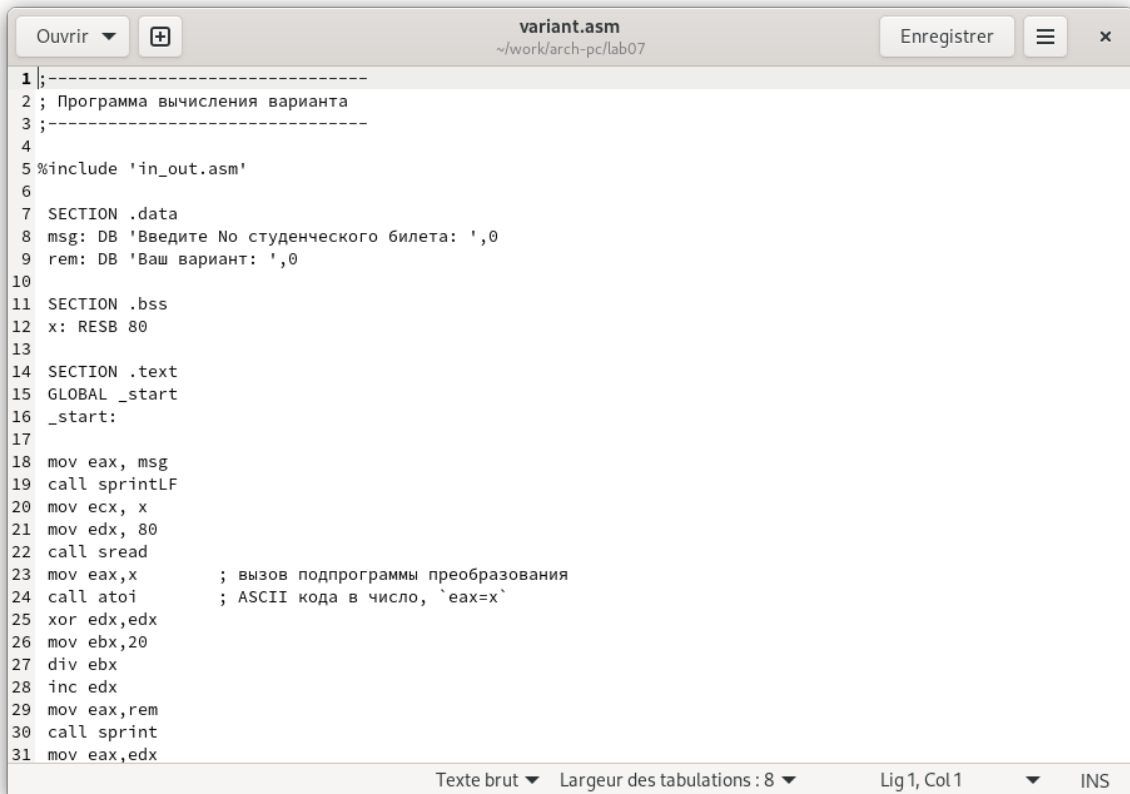
$$f(x) = (4 * 6 + 2)/5$$

. Создайте исполняемый файл и проверьте его работу. (рис. 4.12, рис. 4.13)



```
[kcmuzhinga@fedora lab07]$ nasm -f elf lab7-3.1.asm
[kcmuzhinga@fedora lab07]$ ld -m elf_i386 -o lab7-3.1 lab7-3.1.o
[kcmuzhinga@fedora lab07]$ ./lab7-3.1
Результат: 4
Остаток от деления: 2
[kcmuzhinga@fedora lab07]$
```

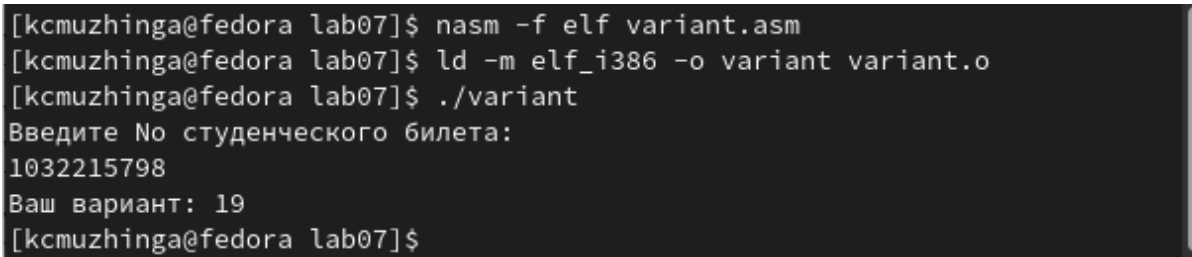
Рис. 4.12: Пример программы



```
1 ;-----
2 ; Программа вычисления варианта
3 ;-----
4
5 %include 'in_out.asm'
6
7 SECTION .data
8 msg: DB 'Введите No студенческого билета: ',0
9 rem: DB 'Ваш вариант: ',0
10
11 SECTION .bss
12 x: RESB 80
13
14 SECTION .text
15 GLOBAL _start
16 _start:
17
18 mov eax, msg
19 call printf
20 mov ecx, x
21 mov edx, 80
22 call read
23 mov eax, x      ; вызов подпрограммы преобразования
24 call atoi      ; ASCII кода в число, 'eax=x'
25 xor edx, edx
26 mov ebx, 20
27 div ebx
28 inc edx
29 mov eax, rem
30 call printf
31 mov eax, edx
```

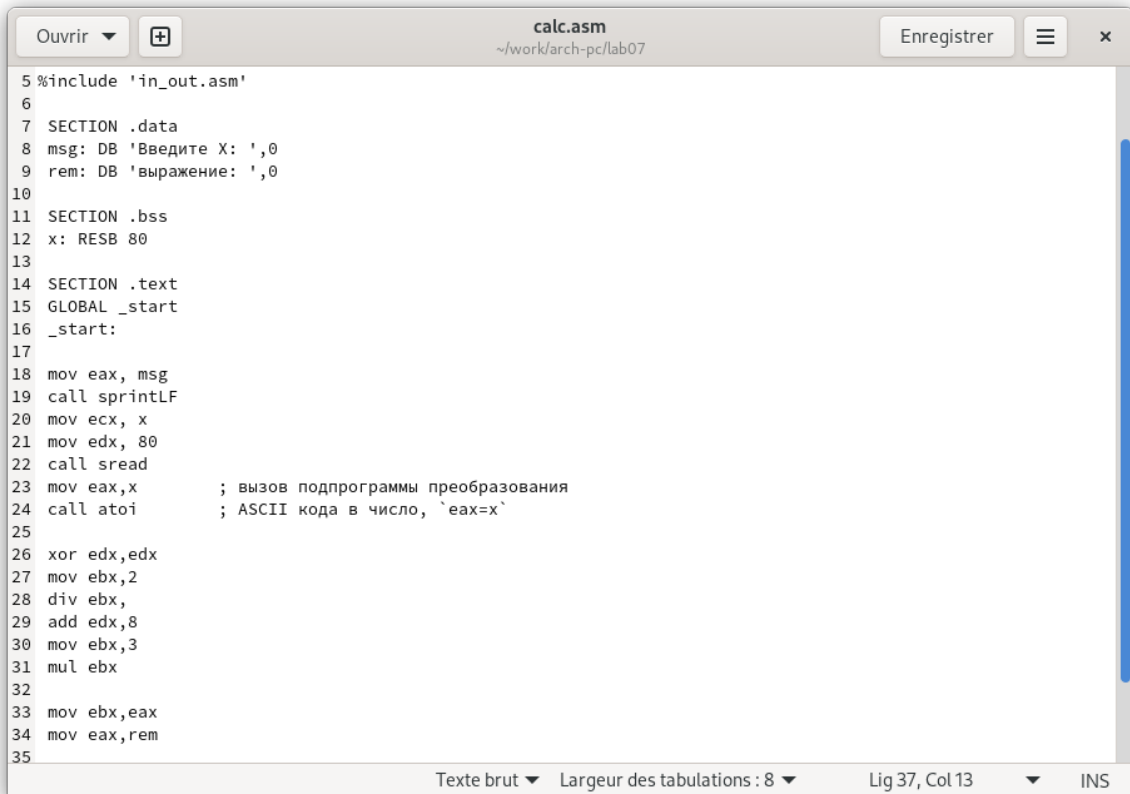
Рис. 4.13: Работа программы

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



```
[kcmuzhinga@fedora lab07]$ nasm -f elf variant.asm
[kcmuzhinga@fedora lab07]$ ld -m elf_i386 -o variant variant.o
[kcmuzhinga@fedora lab07]$ ./variant
Введите No студенческого билета:
1032215798
Ваш вариант: 19
[kcmuzhinga@fedora lab07]$
```

Рис. 4.14: Пример программы



```
5 %include 'in_out.asm'
6
7 SECTION .data
8 msg: DB 'Введите X: ',0
9 rem: DB 'выражение: ',0
10
11 SECTION .bss
12 x: RESB 80
13
14 SECTION .text
15 GLOBAL _start
16 _start:
17
18 mov eax, msg
19 call sprintLF
20 mov ecx, x
21 mov edx, 80
22 call sread
23 mov eax,x           ; вызов подпрограммы преобразования
24 call atoi           ; ASCII кода в число, `eax=x`
25
26 xor edx,edx
27 mov ebx,2
28 div ebx,
29 add edx,8
30 mov ebx,3
31 mul ebx
32
33 mov ebx,eax
34 mov eax,rem
35
```

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? `mov 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 x_1 и x_2 из 6.3. (рис. 4.16, рис. ??)

Получили вариант 15 -

$$(5 + x)^2 - 3$$

для $x=5$ и 1

```

calc.asm:25: error: invalid combination of opcode and operands
[kcmuzhinga@fedora lab07]$ nasm -f elf calc.asm
[kcmuzhinga@fedora lab07]$ ld -m elf_i386 -o calc calc.o
[kcmuzhinga@fedora lab07]$ ./calc
Введите X:
3
выражение: 3
[kcmuzhinga@fedora lab07]$

```

Рис. 4.16: Пример программы

5 Выводы

Изучили работу с арифметическими операциями

Список литературы

1. Расширенный ассемблер: NASM
2. MASM, TASM, FASM, NASM под Windows и Linux