

Лабораторная работа №6

Арифметические операции в NASM

Архитектура компьютера и Операционные системы

Ван Сихэм Франклин О Нил Джон (Миша)

Содержание

1	Цель работы	5
2	Теоретическое введение	6
2.1	Адресация в NASM	6
2.2	Арифметические операции в NASM	7
2.2.1	Целочисленное сложение add.	7
2.2.2	Целочисленное вычитание sub.	8
2.2.3	Команды инкремента и декремента.	8
2.2.4	Команда изменения знака операнда neg.	8
2.2.5	Команды умножения mul и imul.	9
2.2.6	Команды деления div и idiv.	10
2.2.7	Перевод символа числа в десятичную символьную запись	11
3	Порядок выполнения лабораторной работы	13
3.1	Символьные и численные данные в NASM	13
4	Задание для самостоятельной работы	30
5	Заключение	35

Список иллюстраций

3.1	Картинка 1 создание каталога для лабораторной работы №6 .	13
3.2	Картика 2 Программа вывода значения регистра еах	15
3.3	Картика 3 Программа вывода значения регистра еах	16
3.4	Картинка 4 Изменен символов на числа в регистры	17
3.5	Картинка 5 Результат не отображается	17
3.6	Картинка 6 Создан файл lab6-2.asm	18
3.7	Картинка 7 Вывод из листинга 6.2	19
3.8	Картинка 8 Результат программы из листинга 6.2	19
3.9	Картинка 9 Изменен символы на числа	20
3.10	Картинка 10 Программа вычисления выражения $f(x) = (5 \times 2 + 3)/3$	22
3.11	Картинка 11 Вывод результата выражения $f(x) = (5 \times 2 + 3)/3$	23
3.12	Картинка 12 Программа вычисления изменённого выражения $f(x) = (4 \times 6 + 2)/5$	24
3.13	Картинка 13 Создан файл 'variant'	25
3.14	Картинка 14 Выполнения листинга 6.4 в файле variant.asm	27
3.15	Картинка 15 Результат программы вычисления варианта задания по номеру студенческого билета	28
3.16	Картинка 18 Аналитическая проверка работы программы . . .	28
3.17	Картинка 17 Аналитическая проверка работы программы . . .	28
4.1	Картинка 18 Создан файл для само-работы	30
4.2	Картинка 19 Выполнения программы вычисления выражения $y = (8x - 6)/2$	31
4.3	Картинка 19 Результат программы вычисления выражения $y = (8x - 6)/2$	32
4.4	Картинка 20 Обновление в GitHub	33
4.5	Картинка 21 Обновление в GitHub	34

Список таблиц

1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

2 Теоретическое введение

2.1 Адресация в NASM

Большинство инструкций на языке ассемблера требуют обработки операндов. Адрес операнда предоставляет место, где хранятся данные, подлежащие обработке. Это могут быть данные хранящиеся в регистре или в ячейке памяти. Далее рассмотрены все существующие способы задания адреса хранения операндов – способы адресации. Существует три основных способа адресации:

1. Регистровая адресация – операнды хранятся в регистрах и в команде используются имена этих регистров, например: `mov ax, bx`.
2. Непосредственная адресация – значение операнда задается непосредственно в команде, Например: `mov ax, 2`.
3. Адресация памяти – операнд задает адрес в памяти. В команде указывается символическое обозначение ячейки памяти, над содержимым которой требуется выполнить операцию.

Например, определим переменную `intg DD 3` – это означает, что задается область памяти размером 4 байта, адрес которой обозначен меткой `intg`. В таком случае, команда

```
mov eax, [intg]
```

копирует из памяти по адресу `intg` данные в регистр `eax`. В свою очередь команда

```
mov [intg],eax
```

запишет в память по адресу intg данные из регистра eax. Также рассмотрим команду

```
mov eax,intg
```

В этом случае в регистр eax запишется адрес intg. Допустим, для intg выделена память начиная с ячейки с адресом 0x600144, тогда команда `mov eax,intg` аналогична команде `mov eax,0x600144` – т.е. эта команда запишет в регистр eax число 0x600144.

2.2 Арифметические операции в NASM

2.2.1 Целочисленное сложение add.

Схема команды целочисленного сложения add (от англ. addition - добавление) выполняет сложение двух операндов и записывает результат по адресу первого операнда. Команда add работает как с числами со знаком, так и без знака и выглядит следующим образом:

```
add <операнд_1>, <операнд_2>
```

Допустимые сочетания операндов для команды add аналогичны сочетаниям операндов для команды mov. Так, например, команда `add eax,ebx` прибавит значение из регистра eax к значению из регистра ebx и запишет результат в регистр eax. Примеры:

```
add ax,5 ; AX = AX + 5
```

```
add dx,cx ; DX = DX + CX
```

```
add dx,cl ; Ошибка: разный размер операндов.
```

2.2.2 Целочисленное вычитание **sub**.

Команда целочисленного вычитания **sub** (от англ. subtraction – вычитание) работает аналогично команде **add** и выглядит следующим образом:

sub <операнд_1>, <операнд_2>

Так, например, команда **sub ebx, 5** уменьшает значение регистра **ebx** на 5 и записывает результат в регистр **ebx**.

2.2.3 Команды инкремента и декремента.

Довольно часто при написании программ встречается операция прибавления или вычитания единицы. Прибавление единицы называется инкрементом, а вычитание — декрементом. Для этих операций существуют специальные команды: **inc** (от англ. increment) и **dec** (от англ. decrement), которые увеличивают и уменьшают на 1 свой операнд. Эти команды содержат один операнд и имеют следующий вид:

inc<операнд>

dec<операнд>

Операндом может быть регистр или ячейка памяти любого размера. Команды инкремента и декремента выгодны тем, что они занимают меньше места, чем соответствующие команды сложения и вычитания. Так, например, команда **inc ebx** увеличивает значение регистра **ebx** на 1, а команда **dec ax** уменьшает значение регистра **ax** на 1.

2.2.4 Команда изменения знака операнда **neg**.

Еще одна команда, которую можно отнести к арифметическим командам это команда изменения знака **neg**:

neg <операнд>

Команда **neg** рассматривает свой операнд как число со знаком и меняет знак операнда на противоположный. Операндом может быть регистр или ячейка памяти любого размера.

mov ax,1 ; AX = 1

neg ax ; AX = -1

2.2.5 Команды умножения **mul** и **imul**.

Умножение и деление, в отличие от сложения и вычитания, для знаковых и беззнаковых чисел производиться по-разному, поэтому существуют различные команды. Для беззнакового умножения используется команда **mul** (от англ. multiply – умножение):

mul <операнд>

Для знакового умножения используется команда **imul**:

imul <операнд>

Для команд умножения один из сомножителей указывается в команде и должен находиться в регистре или в памяти, но не может быть непосредственным операндом. Второй сомножитель в команде явно не указывается и должен находиться в регистре EAX, AX или AL, а результат помещается в регистры EDX:EAX, DX:AX или AX, в зависимости от размера операнда 6.1.

Таблица 6.1. Регистры используемые командами умножения в Nasm

Размер операнда	Неявный множитель	Результат умножения
1 байт	AL	AX
2 байта	AX	DX:AX
4 байта	EAX	EDX:EAX

Пример использования инструкции `mul`:

```
a dw 270
```

```
mov ax, 100 ; AX = 100
mul a      ; AX = AX*a
mul bl     ; AX = AL*BL
mul ax     ; DX:AX = AX*AX
```

2.2.6 Команды деления `div` и `idiv`.

Для деления, как и для умножения, существует 2 команды `div` (от англ. divide - деление) и `idiv`:

```
div <делитель> ; Беззнаковое деление
idiv <делитель> ; Знаковое деление
```

В командах указывается только один операнд – делитель, который может быть регистром или ячейкой памяти, но не может быть непосредственным операндом. Местоположение делимого и результата для команд деления зависит от размера делителя. Кроме того, так как в результате деления получается два числа – частное и остаток, то эти числа помещаются в определённые регистры 6.2.

Таблица 6.2. Регистры используемые командами деления в Nasm

Размер операнда (делителя)	Делимое	Частное	Остаток
1 байт	AX	AL	AH
2 байта	DX:AX	AX	DX
4 байта	EDX:EAX	EAX	EDX

Например, после выполнения инструкций

```
mov ax,31
mov dl,15
div dl
```

результат 2 (31/15) будет записан в регистр al, а остаток 1 (остаток от деления 31/15) — в регистр ah. Если делитель — это слово (16-бит), то делимое должно записываться в регистрах dx:ax. Так в результате выполнения инструкций

```
mov ax,2          ; загрузить в регистровую
mov dx,1          ; пары `dx:ax` значение 10002h
mov bx,10h
div bx
```

в регистр ax запишется частное 1000h (результат деления 10002h на 10h), а в регистр dx — 2 (остаток от деления).

2.2.7 Перевод символа числа в десятичную символьную запись

Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации производится согласно кодовой таблице символов ASCII. ASCII – сокращение от American Standard Code for Information Interchange (Американский стандартный код для обмена информацией). Согласно стандарту ASCII каждый символ кодируется одним байтом. Расширенная таблица ASCII состоит из двух частей. Первая (символы с кодами 0-127) является универсальной (см. Приложение.), а вторая (коды 128-255) предназначена для специальных символов и букв национальных алфавитов и на компьютерах разных типов может меняться. Среди инструкций NASM нет такой, которая выводит числа (не в символьном виде). Поэтому, например, чтобы вывести число, надо предварительно преобразовать его цифры в ASCII-коды этих цифр и выводить на экран эти коды, а не само

число. Если же выводить число на экран непосредственно, то экран воспримет его не как число, а как последовательность ASCII-символов – каждый байт числа будет воспринят как один ASCII-символ – и выведет на экран эти символы. Аналогичная ситуация происходит и при вводе данных с клавиатуры. Введенные данные будут представлять собой символы, что сделает невозможным получение корректного результата при выполнении над ними арифметических операций. Для решения этой проблемы необходимо проводить преобразование ASCII символов в числа и обратно. Для выполнения лабораторных работ в файле `in_out.asm` реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Это:

- **iprint** – вывод на экран чисел в формате ASCII, перед вызовом `iprint` в регистр `eax` необходимо записать выводимое число (`mov eax,<int>`).
- **iprintLF** – работает аналогично `iprint`, но при выводе на экран после числа добавляет к символ перевода строки.
- **atoi** – функция преобразует `ascii`-код символа в целое число и записывает результат в регистр `eax`, перед вызовом `atoi` в регистр `eax` необходимо записать число (`mov eax,<int>`).

3 Порядок выполнения лабораторной работы

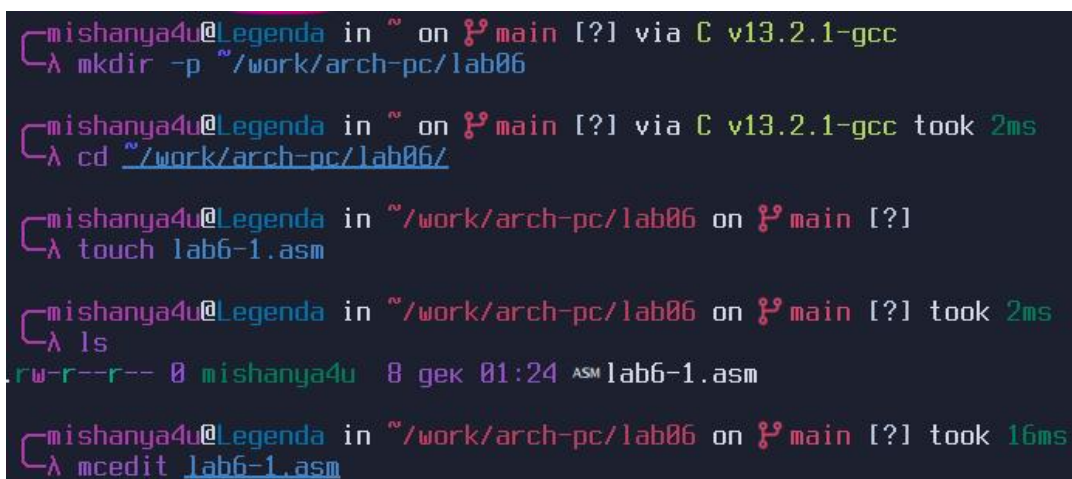
3.1 Символьные и численные данные в NASM

1. Создайте каталог для программ лабораторной работы No 6, перейдите в него и создайте файл lab6-1.asm:

```
mkdir ~/work/arch-pc/lab06
```

```
cd ~/work/arch-pc/lab06
```

```
touch lab6-1.asm
```



```
mishanya4u@Legenda in ~ on 1 main [?] via C v13.2.1-gcc
λ mkdir -p ~/work/arch-pc/lab06

mishanya4u@Legenda in ~ on 1 main [?] via C v13.2.1-gcc took 2ms
λ cd ~/work/arch-pc/lab06/

mishanya4u@Legenda in ~/work/arch-pc/lab06 on 1 main [?]
λ touch lab6-1.asm

mishanya4u@Legenda in ~/work/arch-pc/lab06 on 1 main [?] took 2ms
λ ls
-rw-r--r-- 0 mishanya4u 8 gek 01:24 ASM lab6-1.asm

mishanya4u@Legenda in ~/work/arch-pc/lab06 on 1 main [?] took 16ms
λ mcedit lab6-1.asm
```

Рис. 3.1: Картинка 1 создание каталога для лабораторной работы №6

2. Рассмотрим примеры программ вывода символьных и численных значений. Программы будут выводить значения записанные в регистр `eax`.

Введите в файл `lab6-1.asm` текст программы из листинга 6.1. В данной программе в регистр `eax` записывается символ 6 (`mov eax, '6'`), в регистр `ebx` символ 4 (`mov ebx, '4'`). Далее к значению в регистре `eax` прибавляем значение регистра `ebx` (`add eax, ebx`, результат сложения запишется в регистр `eax`). Далее выводим результат. Так как для работы функции `sprintf` в регистр `eax` должен быть записан адрес, необходимо использовать дополнительную переменную. Для этого запишем значение регистра `eax` в переменную `buf1` (`mov [buf1], eax`), а затем запишем адрес переменной `buf1` в регистр `eax` (`mov eax, buf1`) и вызовем функцию `sprintf`.

Листинг 6.1. Программа вывода значения регистра `eax`

```
%include 'in_out.asm'
```

```
SECTION .bss
```

```
buf1:  RESB 80
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov eax, '6'
```

```
mov ebx, '4'
```

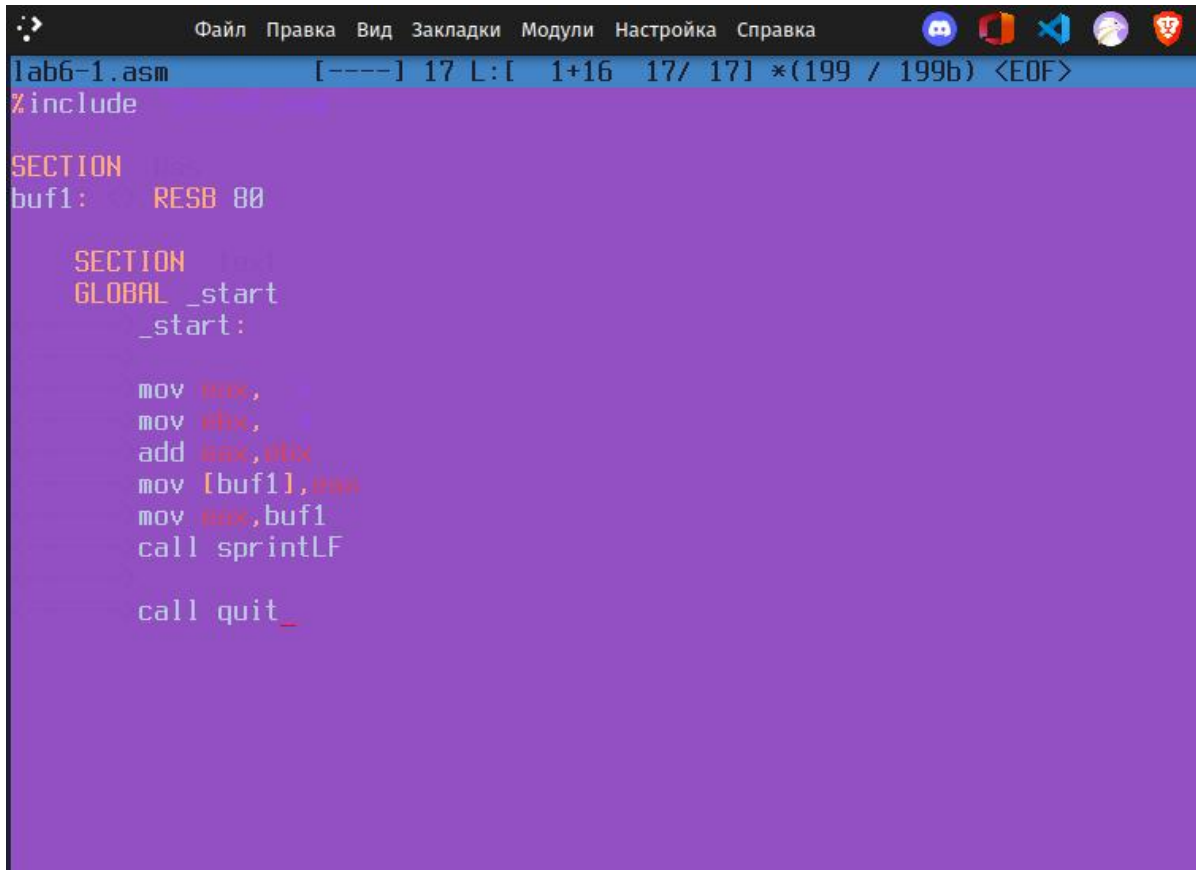
```
add eax, ebx
```

```
mov [buf1], eax
```

```
mov eax, buf1
```

```
call sprintLF
```

```
call quit
```



```
lab6-1.asm [-----] 17 L: [ 1+16 17/ 17] *(199 / 199b) <EOF>
%include "in_out.asm"

SECTION .text
buf1:    RESB 80

SECTION .text
GLOBAL _start
_start:

    mov eax, 1
    mov ebx, 2
    add eax, ebx
    mov [buf1], eax
    mov eax, buf1
    call sprintLF

    call quit_
```

Рис. 3.2: Картика 2 Программа вывода значения регистра eax

Создайте исполняемый файл и запустите его.

```
nasm -f elf lab6-1.asm
```

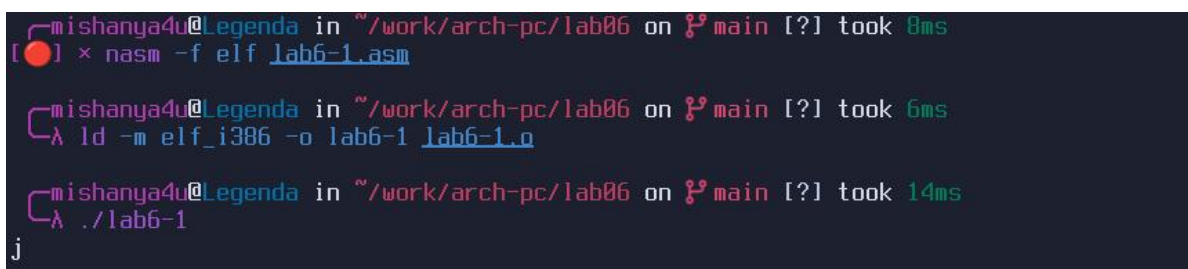
```
ld -m elf_i386 -o lab6-1 lab6-1.o
```

```
./lab6-1
```

ВАЖНО! Для корректной работы программы подключаемый файл `in_out.asm` должен лежать в том же каталоге, что и файл

с текстом программы. Перед созданием исполняемого файла создайте копию файла in_out.asm в каталоге ~/work/arch-pc/lab06.

В данном случае при выводе значения регистра еах мы ожидаем увидеть число 10. Однако результатом будет символ j. Это происходит потому, что код символа 6 равен 00110110 в двоичном представлении (или 54 в десятичном представлении), а код символа 4 – 00110100. (52). Команда add еах,ебх запишет в регистр еах сумму кодов – 01101010 (106), что в свою очередь является кодом символа j (см. таблицу ASCII в приложении).



```
mishanya4u@Legenda in ~/work/arch-pc/lab06 on 3 main [?] took 8ms  
[?] x nasm -f elf lab6-1.asm  
  
mishanya4u@Legenda in ~/work/arch-pc/lab06 on 3 main [?] took 6ms  
λ ld -m elf_i386 -o lab6-1 lab6-1.o  
  
mishanya4u@Legenda in ~/work/arch-pc/lab06 on 3 main [?] took 14ms  
λ ./lab6-1  
j
```

Рис. 3.3: Картика 3 Программа вывода значения регистра еах

3. Далее изменим текст программы и вместо символов, запишем в регистры числа. Исправьте текст программы (Листинг 6.1) следующим образом: замените строки

```
mov еах, '6'  
mov ебх, '4'
```

на строки

```
mov еах, 6  
mov ебх, 4
```



```
lab6-1.asm [----] 16 L: [ 1+10 11/ 17] *(121 / 193b) 0052 0x034 [*][X]
%include

SECTION
buf1: RESB 80

SECTION
GLOBAL _start
_start:

mov ecx,6
mov ebx,4
add ebx,ecx
mov [buf1],ecx
mov ecx,buf1
call sprintLF

call quit
```

Рис. 3.4: Картинка 4 Изменен символов на числа в регистры

Создайте исполняемый файл и запустите его. Как и в предыдущем случае при исполнении программы мы не получим число 10. В данном случае выводится символ с кодом 10. Пользуясь таблицей ASCII определите какому символу соответствует код 10. Отображается ли этот символ при выводе на экран?

Нет символ не отображается.

```
mishanya4u@Legenda in ~/work/arch-pc/lab06 on 19 main [?] took 1ms
λ mcedit lab6-1.asm

mishanya4u@Legenda in ~/work/arch-pc/lab06 on 19 main [?] took 37s
λ nasm -f elf lab6-1.asm

mishanya4u@Legenda in ~/work/arch-pc/lab06 on 19 main [?] took 5ms
λ ld -m elf_i386 -o lab6-1 lab6-1.o

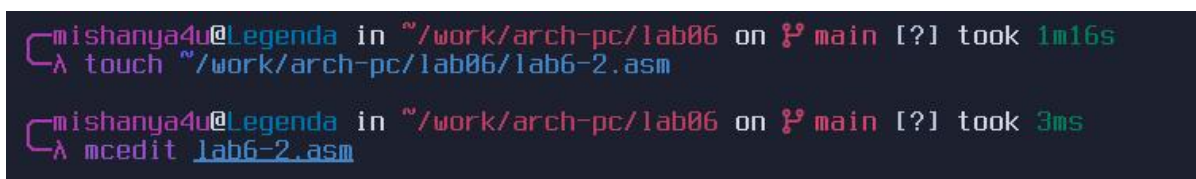
mishanya4u@Legenda in ~/work/arch-pc/lab06 on 19 main [?] took 5ms
λ ./lab6-1
```

Рис. 3.5: Картинка 5 Результат не отображается

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 6.1 с использованием этих функций.

Создайте файл `lab6-2.asm` в каталоге `~/work/arch-pc/lab06` и введите в него текст программы из листинга 6.2.

```
touch ~/work/arch-pc/lab06/lab6-2.asm
```



```
mishanya4u@Legenda in ~/work/arch-pc/lab06 on ? main [?] took 1m16s
λ touch ~/work/arch-pc/lab06/lab6-2.asm

mishanya4u@Legenda in ~/work/arch-pc/lab06 on ? main [?] took 3ms
λ mcedit lab6-2.asm
```

Рис. 3.6: Картинка 6 Создан файл `lab6-2.asm`

Листинг 6.2. Программа вывода значения регистра `eax`

```
%include 'in_out.asm'

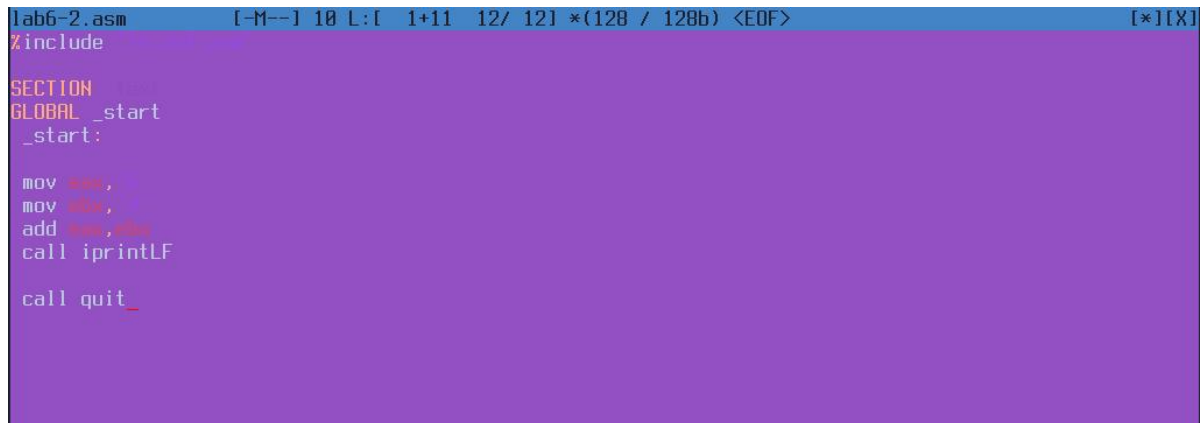
SECTION .text
GLOBAL _start

_start:
mov eax, '6'
mov ebx, '4'
add eax, ebx

call iprintLF
call quit
```

Создайте исполняемый файл и запустите его.

```
nasm -f elf lab6-2.asm
ld -m elf_i386 -o lab6-2 lab6-2.o
./lab6-2
```



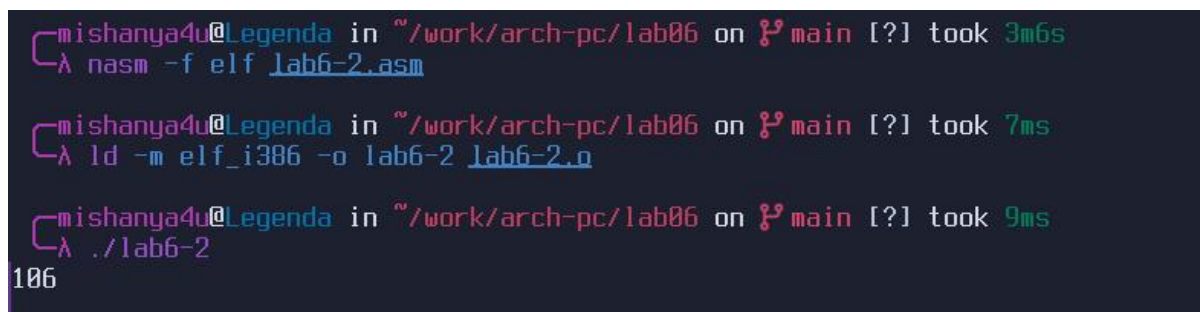
```
lab6-2.asm  [-M--] 10 L: 1+11 12/ 12] *(128 / 128b) <EOF>  [*][X]
#include
SECTION
GLOBAL _start
_start:

mov eax,
mov ebx,
add eax,ebx
call iprintLF

call quit_
```

Рис. 3.7: Картинка 7 Вывод из листинга 6.2

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 6.1, функция `iprintLF` позволяет вывести число, а не символ, кодом которого является это число.



```
mishanya4u@Legenda in ~/work/arch-pc/lab06 on P main [?] took 3ms
└─ nasm -f elf lab6-2.asm

mishanya4u@Legenda in ~/work/arch-pc/lab06 on P main [?] took 7ms
└─ ld -m elf_i386 -o lab6-2 lab6-2.o

mishanya4u@Legenda in ~/work/arch-pc/lab06 on P main [?] took 9ms
└─ ./lab6-2
106
```

Рис. 3.8: Картинка 8 Результат программы из листинга 6.2

5. Аналогично предыдущему примеру изменим символы на числа. Замените строки

```
mov eax, '6'
mov ebx, '4'
```

на строки

```
mov eax, 6
mov ebx, 4
```

Создайте исполняемый файл и запустите его. Какой результат будет получен при исполнении программы? Замените функцию `iprintLF` на `iprint`. Создайте исполняемый файл и запустите его. Чем отличается вывод функций `iprintLF` и `iprint`?

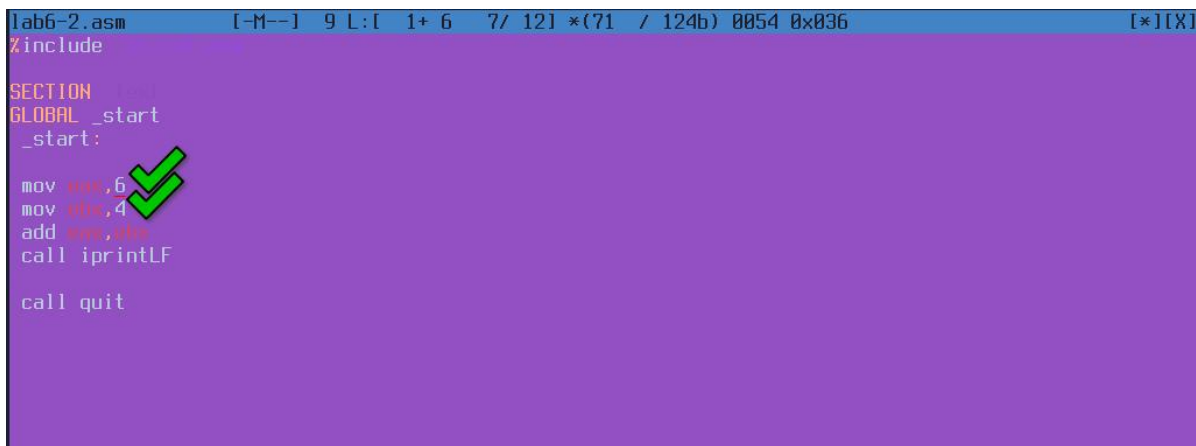


Рис. 3.9: Картинка 9 Изменен символы на числа

Внимательно изучите текст программы из листинга 6.3 и введите в `lab6-3.asm`.

Листинг 6.3. Программа вычисления выражения $f(x) = (5 * 2 + 3)/3$

```
;-----
; 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
```

```
mov eax,div       ; вызов подпрограммы печати
```

```
call sprint       ; сообщения 'Результат: '
```

```
mov eax,edi       ; вызов подпрограммы печати значения
```

```
call iprintLF     ; из 'edi' в виде символов
```

```

mov eax,rem      ; вызов подпрограммы печати
call sprint      ; сообщения 'Остаток от деления: '
mov eax,edx      ; вызов подпрограммы печати значения
call iprintLF    ; из 'edx' (остаток) в виде символов

call quit       ; вызов подпрограммы завершения

```

```

lab6-3.asm      [-M--]  2 L: [  1+ 7   8/ 34] *(231 /1143b) 0010 0x000
;
; Программа (вычисления) вычисления
;
%include "asm.inc" ; подключение файла

SECTION .data
div: DB "Результат: ",0
rem: DB "Остаток от деления: ",0

SECTION .text
GLOBAL _start
_start:
; ----- Программа (вычисления)
mov eax,5      ; EAX=5
mov ebx,2      ; EBX=2
mul ebx        ; EAX=EBX*EBX
add eax,3      ; EAX=EBX*3
xor ebx,ebx    ; обнулим EBX для корректной работы div
mov ebx,3      ; EBX=3
div ebx        ; EAX=EBX/3

mov edi,edx    ; запись результата вычисления в 'edi'

; ----- Вывод результата на экран
mov ebx,div    ; вызов подпрограммы печати
call sprint    ; сообщение 'Результат: '
mov ebx,edi    ; вызов подпрограммы печати значения
call iprintLF  ; из 'edi' в виде символов

call quit     ; вызов подпрограммы завершения

```

Рис. 3.10: Картинка 10 Программа вычисления выражения $f(x) = (5 \times 2 + 3)/3$

Создайте исполняемый файл и запустите его. Результат работы про-

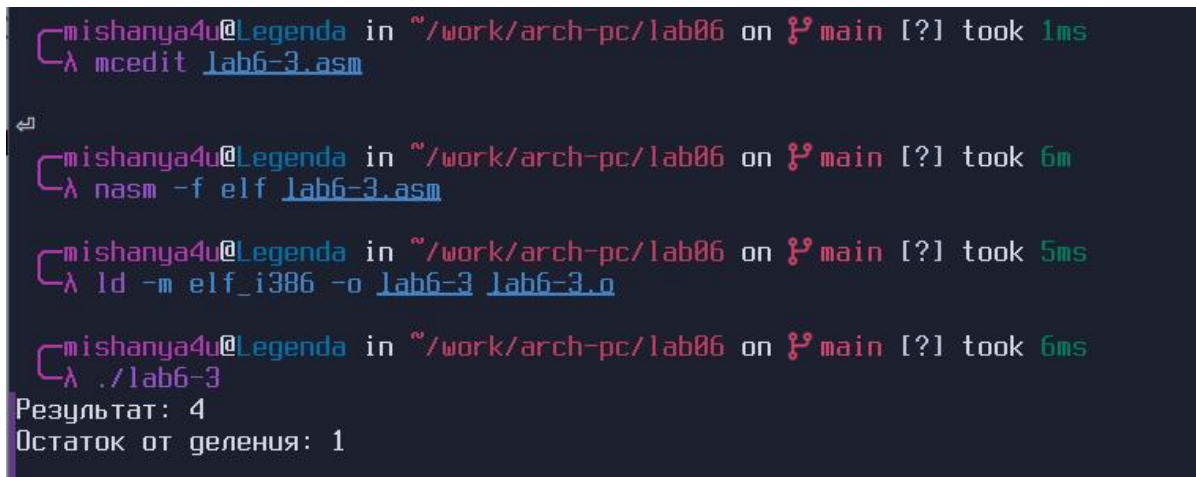
граммы должен быть следующим:

```
user@dk4n31:~$ ./lab6-3
```

Результат: 4

Остаток от деления: 1

```
user@dk4n31:~$
```



```
mishanya4u@Legenda in ~/work/arch-pc/lab06 on P main [?] took 1ms
λ mcedit lab6-3.asm

mishanya4u@Legenda in ~/work/arch-pc/lab06 on P main [?] took 6ms
λ nasm -f elf lab6-3.asm

mishanya4u@Legenda in ~/work/arch-pc/lab06 on P main [?] took 5ms
λ ld -m elf_i386 -o lab6-3 lab6-3.o

mishanya4u@Legenda in ~/work/arch-pc/lab06 on P main [?] took 6ms
λ ./lab6-3
Результат: 4
Остаток от деления: 1
```

Рис. 3.11: Картинка 11 Вывод результата выражения $f(x) = (5 \times 2 + 3)/3$

Измените текст программы для вычисления выражения $f(x) = (4 * 6 + 2)/5$. Создайте исполняемый файл и проверьте его работу.

```

lab6-3.asm      [-M--] 12 L:1 1+20 21/ 391 *(526 /1450b) 0100 0x064
;
; Программа вычисления выражения
;
%include "lab6-3.inc" ; подключение внешнего файла

SECTION .data

div: DB 0 ; делитель
rem: DB 0 ; остаток от деления

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 ecx,ecx ; обнуляем ECX для корректной работы div
mov ebx,5 ; EBX=5
div ebx ; EAX=EAX/5

mov edi,eax ; запись результата вычисления в "edi"

; ----- Вывод результата на экран

mov eax,div ; вызов подпрограммы печати
call sprint ; сообщение "Результат: "
mov ecx,edi ; вызов подпрограммы печати значения
call iprintLF ; из "edi" в базе символов

mov eax,rem ; вызов подпрограммы печати
call sprint ; сообщение "Остаток от деления: "
mov ecx,edi ; вызов подпрограммы печати значения
call iprintLF ; из "edi" (остаток) в базе символов

call quit ; вызов подпрограммы завершения

```

Рис. 3.12: Картинка 12 Программа вычисления изменённого выражения

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

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

- вывести запрос на введение No студенческого билета
- вычислить номер варианта по формуле: $(S_n \bmod 20) + 1$, где S_n –

номер

- студенческого билета (В данном случае $a \bmod b$ – это остаток от деления a на b).
- вывести на экран номер варианта.

В данном случае число, над которым необходимо проводить арифметические операции, вводится с клавиатуры. Как отмечалось выше ввод с клавиатуры осуществляется в символь- ном виде и для корректной работы арифметических операций в NASM символы необходимо преобразовать в числа. Для этого может быть использована функция `atoi` из файла `in_out.asm`.

Создайте файл `variant.asm` в каталоге `~/work/arch-pc/lab06`:

```
touch ~/work/arch-pc/lab06/variant.asm
```

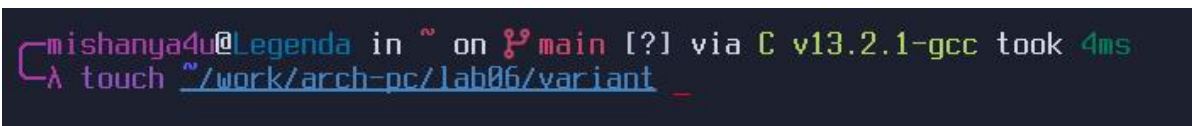


Рис. 3.13: Картинка 13 Создан файл 'variant'

Внимательно изучите текст программы из листинга 6.4 и введите в файл `variant.asm`.

Листинг 6.4. Программа вычисления варианта задания по номеру студенческого билета

```
;-----  
; Программа вычисления варианта  
;-----  
  
%include 'in_out.asm'  
  
SECTION .data
```

```
msg:          DB 'Введите No студенческого билета: ',0
rem:          DB 'Ваш вариант: ',0
```

```
SECTION .bss
```

```
x:    RESB 80
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov eax, msg
call sprintf
```

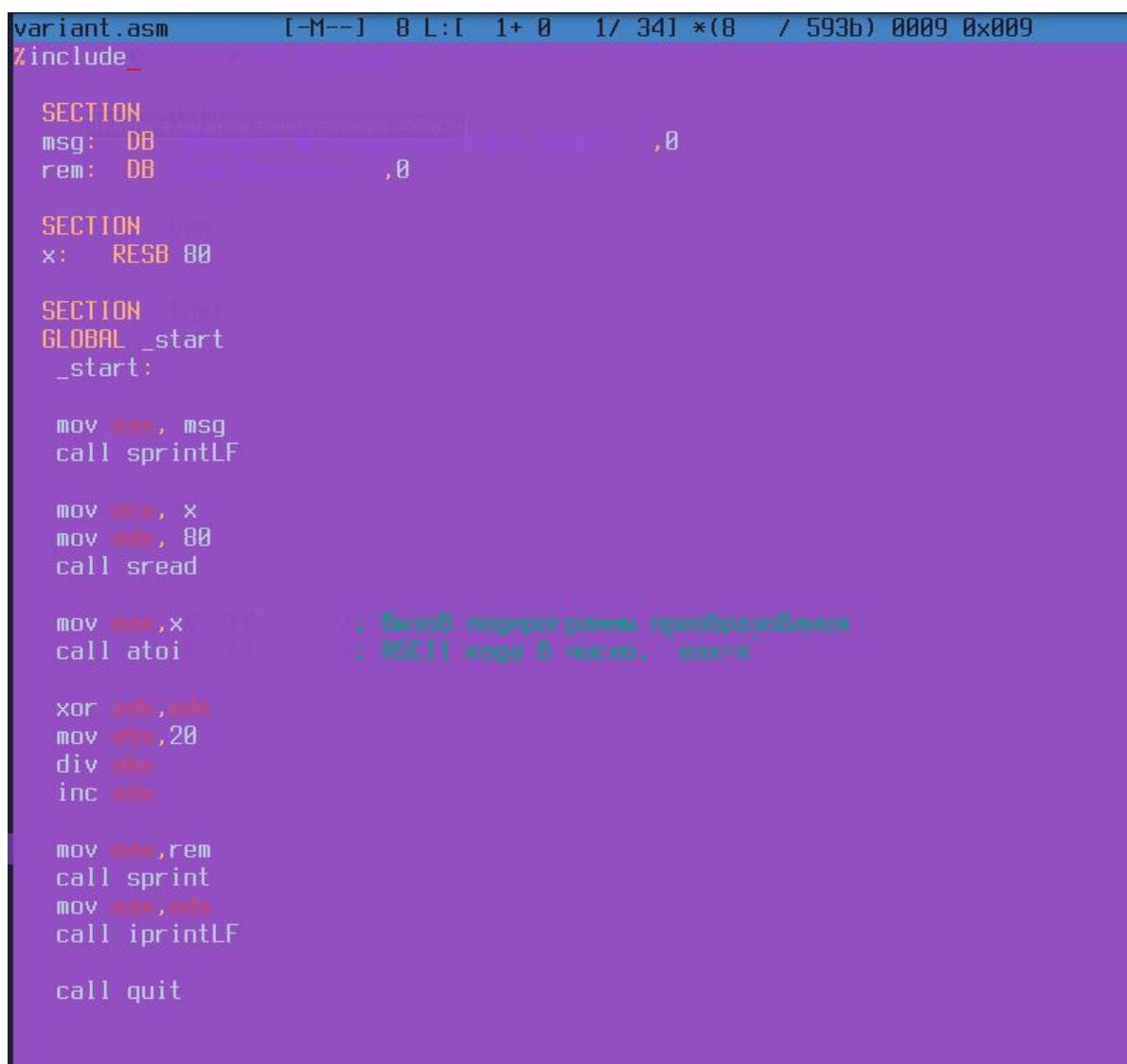
```
mov ecx, x
mov edx, 80
call sread
```

```
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`



```
variant.asm      [-M--]  8 L:[ 1+ 0  1/ 34] *(8  / 593b) 0009 0x009
%include _

SECTION .data
msg: DB "Введите номер варианта: ",0
rem: DB "Результат: ",0

SECTION .bss
x: RESB 80

SECTION .text
GLOBAL _start
_start:

    mov esi, msg
    call sprintf

    mov esi, x
    mov edi, 80
    call sread

    mov esi, x
    call atoi
    ; Введите номер варианта преобразовать
    ; ASCII коды в число, eax=x

    xor edi,edi
    mov edi,20
    div edi
    inc edi

    mov esi,rem
    call sprintf
    mov esi,edi
    call iprintfLF

    call quit
```

Рис. 3.14: Картинка 14 Выполнения листинга 6.4 в файле variant.asm

Создайте исполняемый файл и запустите его. Проверьте результат работы программы вычислив номер варианта аналитически.

```

mishanya4u@Legenda in ~ on 19 main [?] via C v13.2.1-gcc took 4ms
[●] × cd ~/work/arch-pc/lab06/

mishanya4u@Legenda in ~/work/arch-pc/lab06 on 19 main [?] took 1ms
λ nasm -f elf variant.asm

mishanya4u@Legenda in ~/work/arch-pc/lab06 on 19 main [?] took 7ms
λ ld -m elf_i386 -o variant variant.o

mishanya4u@Legenda in ~/work/arch-pc/lab06 on 19 main [?] took 19ms
λ ./variant
Введите № студенческого билета:
1032189251
Ваш вариант: 12

```

Рис. 3.15: Картинка 15 Результат программы вычисления варианта задания по номеру студенческого билета

Operation	Result
1032189251 mod 20	11

Рис. 3.16: Картинка 18 Аналитическая проверка работы программы

$$11 + 1 = 12$$

Рис. 3.17: Картинка 17 Аналитическая проверка работы программы

Включите в отчет по выполнению лабораторной работы ответы на следующие вопросы: 1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения 'Ваш вариант: '?

```

mov eax, rem
call sprint

```

2. Для чего используются следующие инструкции?

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

3. Для чего используется инструкция “call atoi”?

Инструкция call atoi используется для передачи строки в функцию atoi и получения соответствующего целого числа

4. Какие строки листинга 6.4 отвечают за вычисления варианта?

```
xor edx, edx
mov ebx, 20
```

```
div ebx
inc edx
```

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

Остаток от деления при выполнении инструкции ``div ebx`` записывается в edx

6. Для чего используется инструкция inc edx?

Инструкция inc edx используется для увеличения значения регистра edx на 1.

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

```
mov eax, edx
call iprintLF
```

4 Задание для самостоятельной работы

1. 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.

A terminal window with a dark background. The prompt is 'mishanya4u@Legenda in ~/work/arch-pc/lab06 on 8 main [?] took 2ms'. The user has entered the command 'touch ~/work/arch-pc/lab06/self-work.asm' and the prompt is now on the next line.

```
mishanya4u@Legenda in ~/work/arch-pc/lab06 on 8 main [?] took 2ms
λ touch ~/work/arch-pc/lab06/self-work.asm
```

Рис. 4.1: Картинка 18 Создан файл для само-работы

```

self-work.asm      [-M--] 17 L:[ 1+11 12/ 49] *(339 /1171b) 0010 0x00A
%include "asm.h" ; включение файла

SECTION .data

formula: DB "Введите значение x: ",0
msg: DB "Введите значение остатка от деления на 8: ",0
rem: DB "Остаток от деления на 8: ",0

SECTION .bss
x: RESB 80

SECTION .text
GLOBAL _start

_start:

; ----- Выходные данные -----

mov eax, formula
call sprintf

mov eax, msg
call sprintf

mov ebx, x
mov ecx, 80
call sread

mov eax, x
call atoi

mov ebx, 8
mul ebx
sub eax, 6
mov ebx, 2
div ebx

mov ebx, eax

; ----- Вывод результата на экран -----

mov eax, rem
call sprintf ; Вывод информации о начале
; сообщения "Результат:"

mov eax, ebx
call iprintLF ; Вывод информации о начале вычисления
; из 'ebx' (остаток) в базе счисления

call quit ; Вывод информации о завершении

```

Рис. 4.2: Картинка 19 Выполнения программы вычисления выражения $y = (8x - 6)/2$

```
mishanya4u@Legenda in ~/work/arch-pc/lab06 on 19 main [?] took 2s
λ mcedit self-work.asm

mishanya4u@Legenda in ~/work/arch-pc/lab06 on 19 main [?] took 45s
λ nasm -f elf self-work.asm

mishanya4u@Legenda in ~/work/arch-pc/lab06 on 19 main [?] took 6ms
λ ld -m elf_i386 -o self-work self-work.o

mishanya4u@Legenda in ~/work/arch-pc/lab06 on 19 main [?] took 6ms
λ ./self-work
Формула Вычисление: (8x - 6)/2
Введите значения переменной X:
5
Результат: 17
```

Рис. 4.3: Картинка 19 Результат программы вычисления выражения $y = (8x - 6)/2$


```

...nya4u@Legenda in repo: arch-pc/labs/lab06/report on master [x!?] took 30s
└─ cd ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/

└─ mishanya4u@Legenda in repo: arch-pc on master [x!?] took 1ms
└─ git pull
Уже актуально.

└─ mishanya4u@Legenda in repo: arch-pc on master [x!?] took 3s
└─ git add .

└─ mishanya4u@Legenda in repo: arch-pc on master [x+] took 1s
└─ git commit -am 'feat(main): add files lab-6'
[main e9a7e83] feat(main): add files lab-6
78 files changed, 1081 insertions(+), 49 deletions(-)
create mode 100644 labs/lab03/lab3re.tar.gz
create mode 100644 labs/lab03/report/[Content_Types].xml
create mode 100644 labs/lab03/report/_rels/.rels
create mode 100644 labs/lab03/report/docProps/app.xml
create mode 100644 labs/lab03/report/docProps/core.xml
create mode 100644 labs/lab03/report/docProps/custom.xml
delete mode 100644 labs/lab05/report/.~lock.report.docx#
create mode 100644 labs/lab05/report/lab5.ZIP/15.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-1.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-10.1.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-10.2.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-10.3.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-10.4.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-10.5.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-10.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-11.1.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-11.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-12.1.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-12.2.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-12.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-13.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-14.1.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-14.2.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-14.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-16.1.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-16.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-17.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-18.1.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-18.2.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-18.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-19.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-2.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-3.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-4.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-5.png

```

Рис. 4.4: Картинка 20 Обновление в GitHub

```

create mode 100644 labs/lab05/report/lab5.ZIP/lab5-7.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-8.1.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-8.png
create mode 100644 labs/lab05/report/lab5.ZIP/lab5-9.png
create mode 100644 labs/lab05/report/lab5.ZIP/placeimg_800_600_tech.jpg
create mode 100644 labs/lab05/report/lab5.ZIP/report.docx
create mode 100644 labs/lab05/report/lab5.ZIP/report.md
create mode 100644 labs/lab05/report/lab5.ZIP/report.pdf
create mode 100644 labs/lab05/report/lab5.ZIP/symlab5.desktop
create mode 100644 labs/lab05/report/lab5.tar.gz
create mode 120000 labs/lab05/report/symlab5
create mode 100644 labs/lab06/report/image/lab6-1.jpg
create mode 100644 labs/lab06/report/image/lab6-10*.jpg
create mode 100644 labs/lab06/report/image/lab6-10-1.jpg
create mode 100644 labs/lab06/report/image/lab6-10.jpg
create mode 100644 labs/lab06/report/image/lab6-12-1.jpg
create mode 100644 labs/lab06/report/image/lab6-12.jpg
create mode 100644 labs/lab06/report/image/lab6-13-1.jpg
create mode 100644 labs/lab06/report/image/lab6-13.jpg
create mode 100644 labs/lab06/report/image/lab6-2.jpg
create mode 100644 labs/lab06/report/image/lab6-3.jpg
create mode 100644 labs/lab06/report/image/lab6-4.jpg
create mode 100644 labs/lab06/report/image/lab6-5.1.jpg
create mode 100644 labs/lab06/report/image/lab6-5.jpg
create mode 100644 labs/lab06/report/image/lab6-6.jpg
create mode 100644 labs/lab06/report/image/lab6-7.jpg
create mode 100644 labs/lab06/report/image/lab6-8*.jpg
create mode 100644 labs/lab06/report/image/lab6-8.jpg
create mode 100644 labs/lab06/report/image/lab6-9-1.jpg
create mode 100644 labs/lab06/report/image/lab6-9.jpg
create mode 100644 labs/lab06/report/image/lab6-check1.jpg
create mode 100644 labs/lab06/report/image/lab6-check2.jpg
create mode 100644 labs/lab06/report/image/lab6.jpg
create mode 100644 labs/lab06/report/image/let1.jpg
create mode 100644 labs/lab06/report/image/pip.jpg
create mode 100644 labs/lab06/report/report.docx
create mode 100644 labs/lab06/report/report.pdf

[mishanya4u@Legenda in repo: arch-pc on master [!1] took 85ms]
λ git push
Перечисление объектов: 98, готово.
Подсчет объектов: 100% (98/98), готово.
При сжатии изменений используется до 8 потоков
Сжатие объектов: 100% (84/84), готово.
Запись объектов: 100% (86/86), 17.02 Мб | 2.63 Мб/с, готово.
Всего 86 (изменений 6), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (6/6), completed with 6 local objects.
To github.com:Mishanya4/study_2023-2024_arh-pc.git
 a30f8c8..e9a7e83 master -> master

```

Рис. 4.5: Картинка 21 Обновление в GitHub

5 Заключение

После изучения арифметических операций, я могу выполнять базовые математические операции (сложение, вычитание, умножение, деление) на уровне ассемблера, а также обращаться к различным ячейкам памяти для чтения и записи данных. Также я познакомился с использованием регистров процессора для выполнения операций и хранения промежуточных результатов.