

# **Отчёт по лабораторной работе №6**

**2023**

Просина Ксения Максимовна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Теоретическое введение</b>	<b>6</b>
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. . . . .	7
2.2.3	Команды инкремента и декремента. . . . .	7
2.2.4	Команда изменения знака операнда neg. . . . .	8
2.2.5	Команды умножения mul и imul. . . . .	8
2.2.6	Команды деления div и idiv. . . . .	9
2.3	Перевод символа числа в десятичную символьную запись . . . . .	10
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>12</b>
3.1	Символьные и численные данные в NASM . . . . .	12
3.2	Выполнение арифметических операций в NASM . . . . .	15
3.3	Задание для самостоятельной работы . . . . .	18
<b>4</b>	<b>Выводы</b>	<b>24</b>
	<b>Список литературы</b>	<b>25</b>

## Список иллюстраций

2.1	Регистры используемые командами умножения в Nasm . . . . .	9
2.2	Регистры используемые командами деления в Nasm . . . . .	10
3.1	Создание каталога и файла . . . . .	12
3.2	Создание исполняемого файла и проверка работы . . . . .	13
3.3	Рис 5 . . . . .	14
3.4	Создание файла, исполняемого файла и проверка . . . . .	14
3.5	Создание исполняемого файла и проверка . . . . .	15
3.6	Создание исполняемого файла и проверка . . . . .	15
3.7	Создание файла . . . . .	16
3.8	Создание исполняемого файла и проверка . . . . .	16
3.9	Изменение кода, создание исполняемого файла и проверка . . . .	16
3.10	Создание исполняемого файла и проверка . . . . .	17
3.11	Выражения для $\boxtimes(\boxtimes)$ для задания №1 . . . . .	19
3.12	Код для вычисления функции . . . . .	22
3.13	Компоновка и проверка . . . . .	23

## 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:

- Регистровая адресация – операнды хранятся в регистрах и в команде используются имена этих регистров, например: `mov ax,bx`.
- Непосредственная адресация – значение операнда задается непосредственно в команде, Например: `mov ax,2`.
- 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

## 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 ,

Допустимые сочетания операндов для команды add аналогичны сочетаниям операндов для команды mov. Так, например, команда add eax,ebx прибавит значение из регистра ebx к значению из регистра eax 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 ,  
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 коман-

ды: `inc` (от англ. *increment*) и `dec` (от англ. *decrement*), которые увеличивают и уменьшают на 1 свой операнд. Эти команды содержат один операнд и имеет следующий вид: `inc`

`dec`

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`

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`

Для знакового умножения используется команда `imul`: `imul`

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 нахо-



даться в регистре EAX, AX или AL, а результат помещается в регистры EDX:EAX, DX:AX или AX, в зависимости от размера операнда.

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

Рис. 2.1: Регистры используемые командами умножения в Nasm

Пример использования инструкции `mul`: `a dw 270`

`mov ax, 100 ; AX = 100`

`mul a ; AX = AXa,`

`mul bl ; AX = ALBL`

`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` ; Беззнаковое деление

`idiv` ; 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.

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

Рис. 2.2: Регистры используемые командами деления в Nasm

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` ; пару `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).

## 2.3 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) явля-

ется 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 делает 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,` ).
- `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 запишет результат в регистр `eax`, перед вызовом `atoi` в регистр `eax` необходимо записать число (`mov eax,` ).

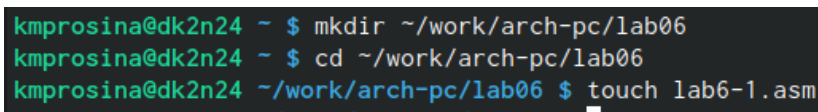
## 3 Выполнение лабораторной работы

### 3.1 Символьные и численные данные в NASM

1. Создайте каталог для программ лабораторной работы № 6, перейдите в него и создайте файл lab6-1.asm: `mkdir ~/work/arch-pc/lab06`

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

`touch lab6-1.asm`



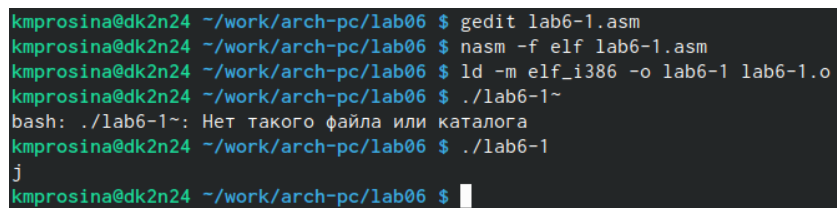
```
kmprosina@dk2n24 ~ $ mkdir ~/work/arch-pc/lab06
kmprosina@dk2n24 ~ $ cd ~/work/arch-pc/lab06
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ touch lab6-1.asm
```

Рис. 3.1: Создание каталога и файла

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 printf. 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.



```
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ gedit lab6-1.asm
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ ./lab6-1~
bash: ./lab6-1~: Нет такого файла или каталога
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ ./lab6-1
j
kmprosina@dk2n24 ~/work/arch-pc/lab06 $
```

Рис. 3.2: Создание исполняемого файла и проверка работы

В данном случае при выводе значения регистра eax 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 eax,ebx запишет в регистр eax сумму кодов – 01101010 (106), что в свою очередь является кодом символа j (см. таблицу ASCII в приложении). 3. Далее изменим текст программы и вместо символов, запишем в регистры числа. Исправьте текст программы (Листинг 6.1) следующим образом: замените строки mov eax,'6'

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

```
на строки mov eax,6
```

```
mov ebx,4
```

Создайте исполняемый файл и запустите его. Как и в предыдущем случае при исполнении программы мы не получим число 10. В данном случае выводит-

ся символ с кодом 10. Пользуясь таблицей ASCII определите какому символу соответствует код 10. Отображается ли этот символ при выводе на экран?

```
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ gedit lab6-1.asm
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ ./lab6-1
```

Рис. 3.3: Рис 5

Код 10 соответствует символу LF, который не может отображаться в консоли

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`

Создайте исполняемый файл и запустите его. `nasm -f elf lab6-2.asm`  
`ld -m elf_i386 -o lab6-2 lab6-2.o`  
`./lab6-2`

```
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/lab6-2.asm
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ gedit lab6-2.asm
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ ./lab6-2
106
kmprosina@dk2n24 ~/work/arch-pc/lab06 $
```

Рис. 3.4: Создание файла, исполняемого файла и проверка

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.

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?

```
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ gedit lab6-2.asm
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
kmprosina@dk2n24 ~/work/arch-pc/lab06 $ ./lab6-2
10
```

Рис. 3.5: Создание исполняемого файла и проверка

После изменений код сложил 6 и 4, в итоге получилось 10  
Замените функцию `iprintLF` на `iprint`. Создайте исполняемый файл и запустите его. Чем отличается вывод функций `iprintLF` и `iprint`?

```
kmprosina@dk3n33 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
kmprosina@dk3n33 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
kmprosina@dk3n33 ~/work/arch-pc/lab06 $ ./lab6-2
10kmprosina@dk3n33 ~/work/arch-pc/lab06 $
```

Рис. 3.6: Создание исполняемого файла и проверка

`iprintLF` выводит ответ и переводит следующую информацию на новой строке. Поскольку мы заменили его на `iprint`, новая строка консоли не вывелась на следующую и последовала сразу после выведенного числа

## 3.2 Выполнение арифметических операций в NASM

6. В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения  $\boxtimes(\boxtimes) = (5 \boxtimes$

2 + 3)/3. Создайте файл lab6-3.asm в каталоге ~/work/arch-pc/lab06: touch  
~/work/arch-pc/lab06/lab6-3.asm

```
0kmprosina@dk3n33 ~/work/arch-pc/lab06 touch lab6-3.asm
```

Рис. 3.7: Создание файла

Внимательно изучите текст программы из листинга 6.3 и введите в lab6-3.asm. Создайте исполняемый файл и запустите его. Результат работы программы должен быть следующим: user@dk4n31:~\$ ./lab6-3 Результат: 4 Остаток от деления: 1 user@dk4n31:~\$

```
kmprosina@dk3n33 ~/work/arch-pc/lab06 $ gedit lab6-3.asm
kmprosina@dk3n33 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
kmprosina@dk3n33 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
kmprosina@dk3n33 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 4
Остаток от деления: 1
```

Рис. 3.8: Создание исполняемого файла и проверка

Измените текст программы для вычисления выражения  $\boxed{x}(\boxed{x}) = (4 \times 6 + 2)/5$ . Создайте исполняемый файл и проверьте его работу.

<pre>1;----- 2; Программа вычисления выражения 3;----- 4#include 'in_out.asm' ; подключение внешнего файла 5SECTION .data 6div: DB 'Результат: ',0 7rem: DB 'Остаток от деления: ',0 8SECTION .text 9GLOBAL _start 0_start: 1; ---- Вычисление выражения 2mov eax,4 ; EAX=4 3mov ebx,6 ; EBX=6 4mul ebx ; EAX=EAX*EBX 5add eax,2 ; EAX=EAX+2 6xor edx,edx ; обнуляем EDX для корректной работы div 7mov ebx,5 ; EBX=5 8div ebx ; EAX=EAX/3, EDX=остаток от деления 9mov edi,eax ; запись результата вычисления в 'edi' 10; ---- Вывод результата на экран 11mov eax,div ; вызов подпрограммы печати 12call sprint ; сообщения 'Результат: ' 13mov eax,edi ; вызов подпрограммы печати значения 14call iprintLF ; из 'edi' в виде символов 15mov eax,rem ; вызов подпрограммы печати 16call sprint ; сообщения 'Остаток от деления: ' 17mov eax,edx ; вызов подпрограммы печати значения 18call iprintLF ; из 'edx' (остаток) в виде символов 19call quit ; вызов подпрограммы завершения</pre>	<pre>10kmprosina@dk3n33 ~/work/arch-pc/lab06 touch lab6-3.asm kmprosina@dk3n33 ~/work/arch-pc/lab06 \$ gedit lab6-3.asm kmprosina@dk3n33 ~/work/arch-pc/lab06 \$ nasm -f elf lab6-3.asm kmprosina@dk3n33 ~/work/arch-pc/lab06 \$ ld -m elf_i386 -o lab6-3 lab6-3.o kmprosina@dk3n33 ~/work/arch-pc/lab06 \$ ./lab6-3 Результат: 4 Остаток от деления: 1 kmprosina@dk3n33 ~/work/arch-pc/lab06 \$ gedit lab6-3.asm kmprosina@dk3n33 ~/work/arch-pc/lab06 \$ nasm -f elf lab6-3.asm kmprosina@dk3n33 ~/work/arch-pc/lab06 \$ ld -m elf_i386 -o lab6-3 lab6-3.o kmprosina@dk3n33 ~/work/arch-pc/lab06 \$ ./lab6-3 Результат: 5 Остаток от деления: 1 kmprosina@dk3n33 ~/work/arch-pc/lab06 \$</pre>
--	--

Рис. 3.9: Изменение кода, создание исполняемого файла и проверка



7. В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета, работающую по следующему алгоритму:
- вывести запрос на введение № студенческого билета
  - вычислить номер варианта по формуле:  $(\text{номер} \bmod 20) + 1$ , где  $\text{номер}$  – номер студенческого билета (В данном случае  $\text{номер} \bmod 20$  – это остаток от деления  $\text{номер}$  на 20).
  - 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 символьном 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` Внимательно изучите текст программы из листинга 6.4 и введите в файл `variant.asm`. Создайте исполняемый файл и запустите его. Проверьте результат работы программы вычислив номер варианта аналитически.

```
kmprosina@dk3n33 ~/work/arch-pc/lab06 $ gedit variant.asm
kmprosina@dk3n33 ~/work/arch-pc/lab06 $ nasm -f elf variant.asm
kmprosina@dk3n33 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o variant variant.o
kmprosina@dk3n33 ~/work/arch-pc/lab06 $ ./variant
Введите № студенческого билета:
1132231938
Ваш вариант: 19
```

Рис. 3.10: Создание исполняемого файла и проверка

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: '? `rem: 'DB 'Ваш вариант: ',0`  
`и mov eax,rem`
2. Для чего используются следующие инструкции? `mov ecx, x`  
`mov edx, 80`  
`call sread`

3. Для чего используется инструкция “call atoi”?

Для корректной работы арифметических операций в NASM символы необходимо преобразовать в 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”? edx

6. Для чего используется инструкция “inc edx”?

Для добавления к числу единицы

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

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

### 3.3 Задание для самостоятельной работы

1. Написать программу вычисления выражения  $x = x(x)$ . 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  $x(x)$  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  $x_2$ .

Номер варианта	Выражение для $f(x)$	$x_1$	$x_2$
1	$(10 + 2x)/3$	1	10
2	$(12x + 3)5$	1	6
3	$(2 + x)^2$	2	8
4	$\frac{4}{3}(x - 1) + 5$	4	10
5	$(9x - 8)/8$	8	64
6	$x^3/2 + 1$	2	5
7	$5(x - 1)^2$	3	5
8	$(11 + x) \cdot 2 - 6$	1	9
9	$10 + (31x - 5)$	3	1
10	$5(x + 18) - 28$	2	3
11	$10(x + 1) - 10$	1	7
12	$(8x - 6)/2$	1	5
13	$(8x + 6) \cdot 10$	1	4
14	$(\frac{x}{2} + 8) \cdot 3$	1	4
15	$(5 + x)^2 - 3$	5	1
16	$(10x - 5)^2$	3	1
17	$18(x + 1)/6$	3	1
18	$3(x + 10) - 20$	1	5
19	$(\frac{1}{3}x + 5) \cdot 7$	3	9
20	$x^3 \cdot \frac{1}{3} + 21$	1	3

Рис. 3.11: Выражения для  $f(x)$  для задания №1

При выполнении задания преобразовывать (упрощать) выражения для  $\otimes(\otimes)$  нельзя. При выполнении деления в качестве результата можно использовать только целую часть от деления и не учитывать остаток (т.е.  $5 \otimes 2 = 2$ ).

Мне попался номер 19, это  $(x/3 + 5) * 7$

Получившийся код:

```
;----- ; Программа вычисления функции ;-----  
%include 'in_out.asm'  
SECTION .data  
msg: DB 'Введите x:',0  
rem: DB 'F(x) =',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  
mov ebx,3  
div ebx  
add eax,5  
mov edx,7
```

```
mul edx  
mov edx,eax  
;_____
```

```
mov eax,rem  
call sprint  
mov eax,edx  
call iprintLF  
call quit
```

```

1 ;-----
2 ; Программа вычисления функции
3 ;-----
4 %include 'in_out.asm'
5 SECTION .data
6 msg: DB 'Введите x: ',0
7 rem: DB 'F(x) = ',0
8 SECTION .bss
9 x: RESB 80
10 SECTION .text
11 GLOBAL _start
12 _start:
13 mov eax, msg
14 call sprintf
15 mov ecx, x
16 mov edx, 80
17 call sread
18 mov eax,x ; вызов подпрограммы преобразования
19 call atoi ; ASCII кода в число, 'eax=x'
20 ;-----
21
22 xor edx,edx
23 mov ebx,3
24 div ebx
25 add eax,5
26 mov edx,7
27 mul edx
28 mov edx,eax
29 ;-----
30
31 mov eax,rem
32 call sprintf
33 mov eax,edx
34 call iprintLF
35 call quit

```

Рис. 3.12: Код для вычисления функции

mov ebx,3 присвоил ebx 3 div ebx разделил x на 3 add eax,5 прибавил к x 5 mov  
edx,7 присвоил 7 mul edx умножил на 7 mov edx,eax присвоил ebx значение x

```
kmprosina@dk4n68 ~/work/arch-pc/lab06 $ nasm -f elf meow.asm
kmprosina@dk4n68 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o meow meow.o
kmprosina@dk4n68 ~/work/arch-pc/lab06 $ ./meow
Введите x:
3
F(x) = 42
kmprosina@dk4n68 ~/work/arch-pc/lab06 $ ./meow
Введите x:
9
F(x) = 56
```

Рис. 3.13: Компоновка и проверка

Посчитав уравнение самостоятельно, я сверилась с ответами, которые оказались верными

## 4 Выводы

Были освоены арифметические функции ассемблера NASM и применены знания на практике



## Список литературы

::: {#refs} :::  
https://esystem.rudn.ru/pluginfile.php/2089662/mod\_resource/content/0/%D0%9B%D