

Отчёт по лабораторной работе №6  
Дисциплина: Архитектура компьютеров  
и операционные системы  
Борисова Ксения Михайловна

# 1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM

1 Цель работы	2
2 Задание	3
3 Теоретическое введение	3
4. Выполнение лабораторной работы	7
Ответы на вопросы по программе	11
Выполнение самостоятельной работы	12
Выводы	12
Список литературы	12

## 2 Задание

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

## 3 Теоретическое введение

### 6.2.1. 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 0x600144.

## **6.2.2. Арифметические операции в NASM**

### **6.2.2.1. Целочисленное сложение add.**

Схема команды целочисленного сложения add (от англ. addition – добавление) выполняет сложение двух operandов 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 operandов для команды add аналогичны сочетаниям operandов 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 operandов.

### **6.2.2.2. 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.

### **6.2.2.3. 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 operand. Эти команды содержат один operand и имеет следующий вид: inc <операнд> dec <операнд> Operandом 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 inc ax уменьшает значение регистра ax на 1.

#### **6.2.2.4. 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 operand как число со знаком и меняет знак операнда на противоположный. Operandом может быть регистр или ячейка памяти любого размера. mov ax,1 ; AX = 1 neg ax ; AX = -1

#### **6.2.2.5. Команды умножения mul и imul.**

Умножение и деление, в отличии 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 operandом. 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. Регистры используемые командами умножения в Nasm Размер operand'a 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

#### **6.2.2.5. 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 operand – делитель, который может быть регистром или ячейкой памяти, но не может быть непосредственным operandом. 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. Регистры используемые командами деления в Nasm Размер operand'a (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 ; 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).

### **6.2.3. 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) 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 перевод 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>)

## 4. Выполнение лабораторной работы

Создаю каталог для программ лабораторной работы № 6, перехожу в него и создаю файл lab6-1.asm

```
kmborisova@dk3n55 ~ $ mkdir ~/work/arch-pc/lab06
kmborisova@dk3n55 ~ $ cd ~/work/arch-pc/lab06
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ touch lab6-1.asm
kmborisova@dk3n55 ~/work/arch-pc/lab06 $
```

Рис.1

Ввожу в файл lab6-1.asm текст программы из листинга 6.1.

```
Открыть ▼ + lab6-1.asm
~/work/arch-pc/lab06
1 %include 'in_out.asm'
2 SECTION .bss
3 buf1: RESB 80
4 SECTION .text
5 GLOBAL _start
6 _start:
7 mov eax,'6'
8 mov ebx,'4'
9 add eax,ebx
10 mov [buf1],eax
11 mov eax,buf1
12 call sprintLF
13 call quit
```

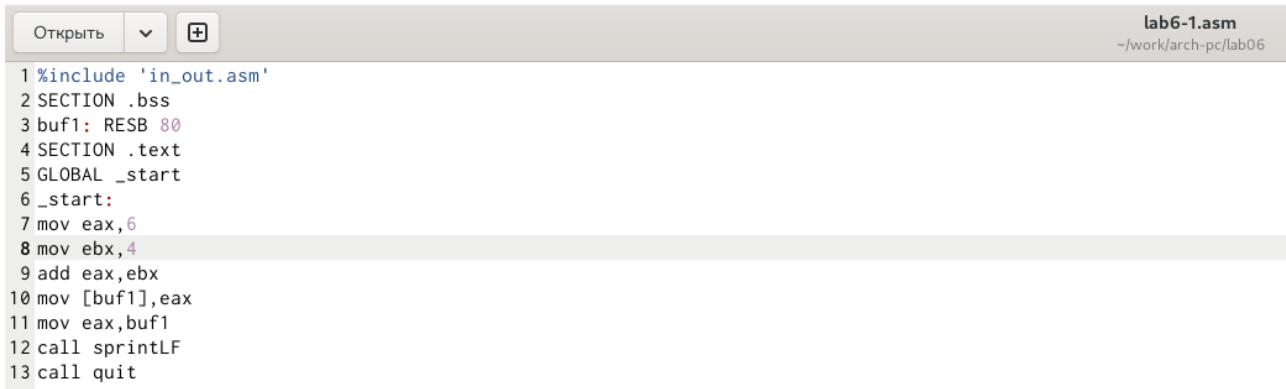
Рис.2

Создаю исполняемый файл и запускаю его

```
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-1
j
kmborisova@dk3n55 ~/work/arch-pc/lab06 $
```

Рис.3

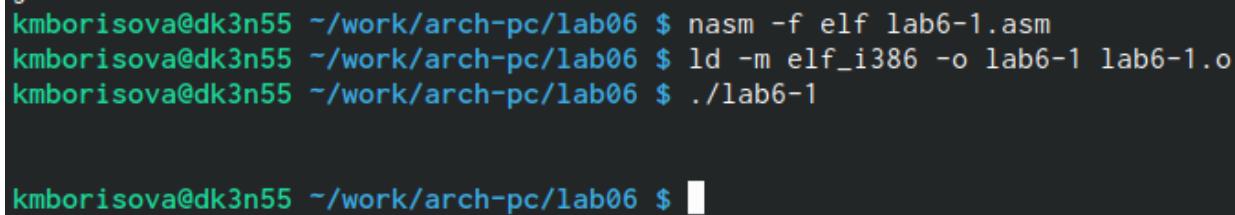
Изменяю текст программы и вместо символов записываю в регистры числа. Исправляю текст программы (Листинг 6.1) следующим образом: заменяю строки mov eax,'6' mov ebx,'4' на строки mov eax,6 mov ebx,4



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

Рис.4

Создайте исполняемый файл и запустите его.

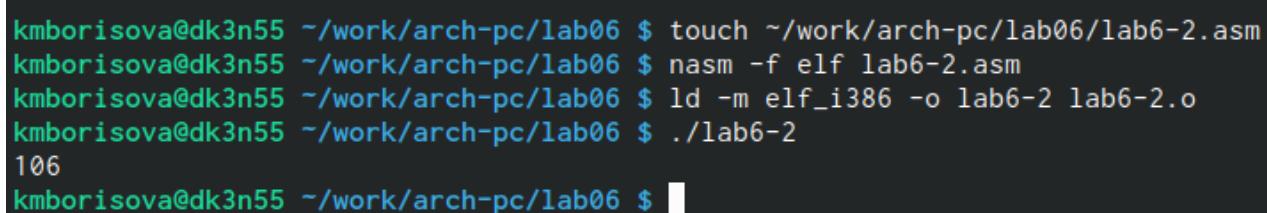


```
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-1

kmborisova@dk3n55 ~/work/arch-pc/lab06 $ █
```

Рис.5

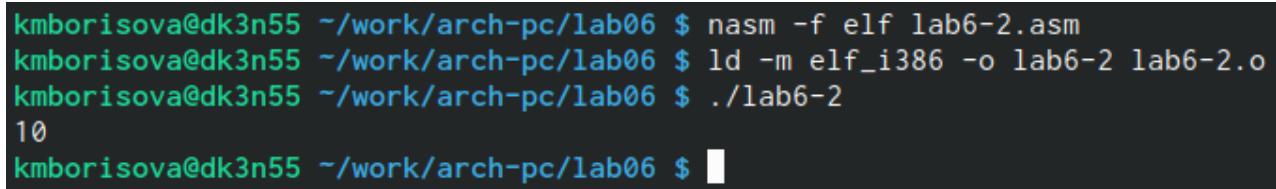
Создаю файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 и ввожу в него текст программы из листинга 6.2.



```
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/lab6-2.asm
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-2
106
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ █
```

Рис.6

Изменяю символы на числа. Создаю исполняемый файл и запускаю его.



```
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-2
10
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ █
```

Рис.7

Заменяю функцию iprintLF на iprint. Создаю исполняемый файл и запускаю его.

```

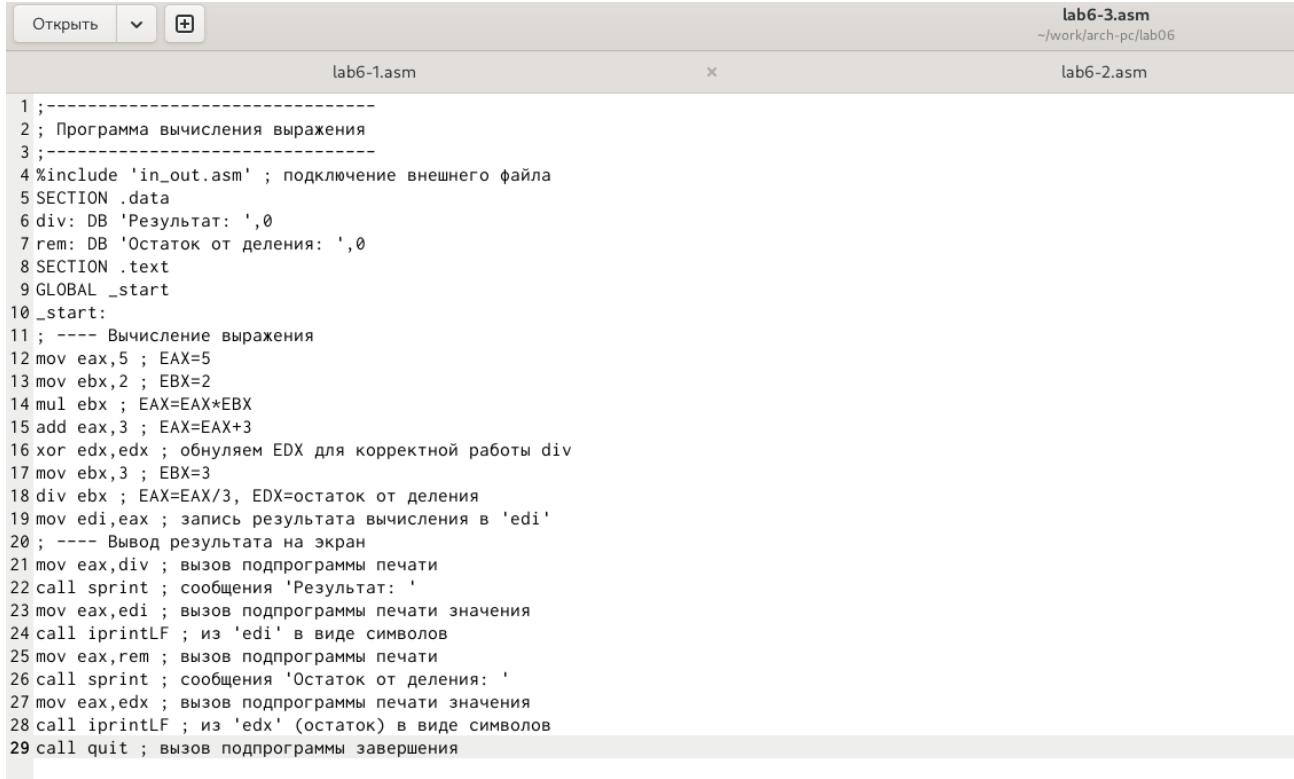
16
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-2
10kmborisova@dk3n55 ~/work/arch-pc/lab06 $ █

```

Рис.8

Создаю файл lab6-3.asm в каталоге ~/work/arch-pc/lab06.

Внимательно изучаю текст программы из листинга 6.3 и ввожу в lab6-3.asm.



```

Открыть ▾ + lab6-3.asm
~/work/arch-pc/lab06
lab6-1.asm × lab6-2.asm
-----+
1 ;-----
2 ; Программа вычисления выражения
3 ;-----
4 %include 'in_out.asm' ; подключение внешнего файла
5 SECTION .data
6 div: DB 'Результат: ',0
7 rem: DB 'Остаток от деления: ',0
8 SECTION .text
9 GLOBAL _start
10 _start:
11 ; ---- Вычисление выражения
12 mov eax,5 ; EAX=5
13 mov ebx,2 ; EBX=2
14 mul ebx ; EAX=EAX*EBX
15 add eax,3 ; EAX=EAX+3
16 xor edx,edx ; обнуляем EDX для корректной работы div
17 mov ebx,3 ; EBX=3
18 div ebx ; EAX=EAX/3, EDX=остаток от деления
19 mov edi,eax ; запись результата вычисления в 'edi'
20 ; ---- Вывод результата на экран
21 mov eax,div ; вызов подпрограммы печати
22 call sprint ; сообщения 'Результат: '
23 mov eax,edi ; вызов подпрограммы печати значения
24 call iprintLF ; из 'edi' в виде символов
25 mov eax,rem ; вызов подпрограммы печати
26 call sprint ; сообщения 'Остаток от деления: '
27 mov eax,edx ; вызов подпрограммы печати значения
28 call iprintLF ; из 'edx' (остаток) в виде символов
29 call quit ; вызов подпрограммы завершения

```

Рис.9

Создаю исполняемый файл и запускаю его.

```

10kmborisova@dk3n55 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/lab6-3.asm
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 4
Остаток от деления: 1
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ █

```

Рис.10

Изменяю текст программы для вычисления выражения  $f(x) = (4 * 6 + 2)/5$ .

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

Рис.11

Создаю исполняемый файл и запускаю его.

```
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm  
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o  
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-3  
Результат: 5  
Остаток от деления: 1  
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ █
```

Рис.12

Создаю файл variant.asm в каталоге ~/work/arch-pc/lab06.

Внимательно изучаю текст программы из листинга 6.4 и ввожу в файл variant.asm. Создаю исполняемый файл и запускаю его.

```
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ touch ~/work/arch-pc/lab06/variant.asm
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf variant.asm
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o variant variant.o
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ./variant
Введите № студенческого билета:
1032253847
Ваш вариант: 8
```

Рис.13

## Ответы на вопросы по программе

1. За вывод сообщения “Ваш вариант” отвечают строки кода:

```
mov eax,rem
call sprint
```

2. Инструкция mov ecx, x используется, чтобы положить адрес вводимой строки x в регистр ecx mov edx, 80 - запись в регистр edx длины вводимой строки call sread - вызов подпрограммы из внешнего файла, обеспечивающей ввод сообщения с клавиатуры
2. call atoi используется для вызова подпрограммы из внешнего файла, которая преобразует ascii-код символа в целое число и записывает результат в регистр eax
3. За вычисления варианта отвечают строки:

```
xor edx,edx ; обнуление edx для корректной работы div
mov ebx,20 ; ebx = 20
div ebx ; eax = eax/20, edx - остаток от деления
inc edx ; edx = edx + 1
```

5. При выполнении инструкции div ebx остаток от деления записывается в регистр edx

5. Инструкция inc edx увеличивает значение регистра edx на 1

6. За вывод на экран результатов вычислений отвечают строки:

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

## Выполнение самостоятельной работы

Пишу программу для вычисления выражения под номером своего варианта, который я узнала при выполнении прошлого задания (8)

```
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ nasm -f elf lab6-4.asm  
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-4 lab6-4.o  
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-4  
y = (11 + x) * 2 - 6  
x = 1  
y = 18  
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ ./lab6-4  
y = (11 + x) * 2 - 6  
x = 9  
y = 34  
kmborisova@dk3n55 ~/work/arch-pc/lab06 $ █
```

Рис.14

## Выводы

При выполнении данной лабораторной работы я освоила арифметические инструкции языка ассемблера NASM.

## Список литературы

1. Лабораторная работа №6

1. Таблица ASCII

