

# **Отчет по лабораторной работе №6**

**Архитектура компьютера и операционные системы**

Александр Дмитриевич Собко

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>10</b>
4.1	Символьные и численные данные в NASM . . . . .	10
4.2	Выполнение арифметических операций в NASM . . . . .	14
4.3	Ответы на вопросы . . . . .	17
<b>5</b>	<b>Задание для самостоятельной работы</b>	<b>19</b>
<b>6</b>	<b>Выводы</b>	<b>21</b>
	<b>Список литературы</b>	<b>22</b>

## Список иллюстраций

4.1	Рисунок1	11
4.2	Рисунок2	11
4.3	Рисунок3	11
4.4	Рисунок4	12
4.5	Рисунок5	12
4.6	Рисунок6	13
4.7	Рисунок7	13
4.8	Рисунок8	13
4.9	Рисунок9	14
4.10	Рисунок10	14
4.11	Рисунок11	15
4.12	Рисунок12	15
4.13	Рисунок13	15
4.14	Рисунок14	16
4.15	Рисунок15	16
5.1	Рисунок16	20
5.2	Рисунок17	20

## Список таблиц

# 1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM

## 2 Задание

Изменить несколько программ с вводом, выводом и вычислением, затем написать свою

### 3 Теоретическое введение

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.

Схема команды целочисленного сложения `add` (от англ. addition - добавление) выполняет сложение двух операндов и записывает результат по адресу первого операнда

Команда целочисленного вычитания `sub` (от англ. subtraction – вычитание) работает аналогично команде `add`

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.

Еще одна команда, которую можно отнести к арифметическим командам это

команда изменения знака `neg`

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

Для знакового умножения используется команда `imul`

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

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, 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, .`). • `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, .`)

## 4 Выполнение лабораторной работы

### 4.1 Символьные и численные данные в NASM

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

```

GNU nano 6.2 lab6-1.asm *
#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

```

Рис. 4.1: Рисунок1

Создайте исполняемый файл и запустите его.

```

alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура_компьютер
a/arch-pc/lab06$ nasm -f elf lab6-1.asm
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура_компьютер
a/arch-pc/lab06$ ld -m elf_i386 -o lab6-1 lab6-1.o
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура_компьютер
a/arch-pc/lab06$ ./lab6-1
j
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура_компьютер
a/arch-pc/lab06$

```

Рис. 4.2: Рисунок2

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

```

GNU nano 6.2 lab6-1.asm
#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

```

Прочитано 13 строк

<sup>^</sup>G Справка    <sup>^</sup>O Записать    <sup>^</sup>M Поиск    <sup>^</sup>K Вырезать    <sup>^</sup>T Выполнить    <sup>^</sup>C Позиция  
<sup>^</sup>X Выход    <sup>^</sup>R ЧитФайл    <sup>^</sup>\ Замена    <sup>^</sup>U Вставить    <sup>^</sup>J Выровнять    <sup>^</sup>/ К строке

Рис. 4.3: Рисунок3

Создайте исполняемый файл и запустите его

```
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера
a/arch-pc/lab06$ nasm -f elf lab6-1.asm
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера
a/arch-pc/lab06$ ld -m elf_i386 -o lab6-1 lab6-1.o
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера
a/arch-pc/lab06$ ./lab6-1

alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютера
a/arch-pc/lab06$
```

Рис. 4.4: Рисунок4

Да, отображается. Это символ перевода строки (line feed)

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.

```
GNU nano 6.2                                lab6-2.asm *
%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax, '6'
mov ebx, '4'
add eax, ebx
call iprintLF
call quit
```

^G Справка	^O Записать	^M Поиск	^K Вырезать	^J Выполнить	^C Позиция
^X Выход	^R ЧитФайл	^N Замена	^U Вставить	^_ Выровнять	^/ К строке

Рис. 4.5: Рисунок5

Создайте и запустите исполняемый файл.

```
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютер
a/arch-pc/lab06$ touch lab6-2.asm
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютер
a/arch-pc/lab06$ nano lab6-2.asm
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютер
a/arch-pc/lab06$ nasm -f elf lab6-2.asm
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютер
a/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютер
a/arch-pc/lab06$ ./lab6-2
106
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютер
a/arch-pc/lab06$
```

Рис. 4.6: Рисунок6

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

```
GNU nano 6.2 lab6-2.asm *
%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprintLF
call quit
```

Рис. 4.7: Рисунок7

Создайте исполняемый файл и запустите его

```
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютер
a/arch-pc/lab06$ nano lab6-2.asm
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютер
a/arch-pc/lab06$ nasm -f elf lab6-2.asm
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютер
a/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютер
a/arch-pc/lab06$ ./lab6-2
10
alexander@alexander-Swift-SF315-S2G: ~/work/study/2023-2024/Архитектура компьютер
a/arch-pc/lab06$
```

Рис. 4.8: Рисунок8

Замените функцию iprintLF на iprint. Создайте исполняемый файл и запустите его. Чем отличается вывод функций iprintLF и iprint?

```
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
a/arch-pc/lab06$ nano lab6-2.asm
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
a/arch-pc/lab06$ nasm -f elf lab6-2.asm
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
a/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьютера
a/arch-pc/lab06$ ./lab6-2
10alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьют
ер
a/arch-pc/lab06$
```

Рис. 4.9: Рисунок9

iprint не добавляет символ перевода строки.

## 4.2 Выполнение арифметических операций в NASM

- В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения  $f(x) = (5 * 2 + 3)/3$ .

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

```
a/arch-pc/lab06$ cat lab6-3.asm
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0
SECTION .text
GLOBAL _start
_start:

mov eax,5
mov ebx,2
mul ebx
add eax,3
xor edx,edx
mov ebx,3
div ebx
mov edi,eax
mov eax,div
call sprint
mov eax,edi
call iprintLF
mov eax,rem
call sprint
mov eax,edx
call iprintLF
call quit
```

Рис. 4.10: Рисунок10

Создайте исполняемый файл и запустите его

```

a/arch-pc/lab06$ touch lab6-3.asm
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура компьютер
a/arch-pc/lab06$ nano lab6-3.asm
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура компьютер
a/arch-pc/lab06$ nasm -f elf lab6-3.asm
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура компьютер
a/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура компьютер
a/arch-pc/lab06$ ./lab6-3
Результат: 4
Остаток от деления: 1

```

Рис. 4.11: Рисунок11

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

```

мьютера/arch-pc/lab06$ cat lab6-3.asm
#include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0
SECTION .text
GLOBAL _start
_start:

mov eax,4
mov ebx,6
mul ebx
add eax,2
xor edx,edx
mov ebx,5
div ebx
mov edi,eax
mov eax,div
call sprint
mov eax,edi
call iprintLF
mov eax,rem
call sprint
mov eax,edx
call iprintLF
call quit

```

Рис. 4.12: Рисунок12

```

alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура ко
мьютера/arch-pc/lab06$ nasm -f elf lab6-3.asm
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура ко
мьютера/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
alexander@alexander-Swift-SF315-S2G:~/work/study/2023-2024/Архитектура ко
мьютера/arch-pc/lab06$ ./lab6-3
Результат: 5
Остаток от деления: 1

```

Рис. 4.13: Рисунок13

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:  $(S_n \bmod 20) + 1$ , 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 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`:

```
компьютера/arch-pc/lab06$ cat variant.asm
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите № студенческого билета: ',0
rem: DB 'Ваш вариант: ',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
call atoi
xor edx, edx
mov ebx, 20
div ebx
inc edx
mov eax, rem
call sprint
mov eax, edx
call iprintLF
call quit
```

Рис. 4.14: Рисунок14

Создайте и запустите исполняемый файл

```
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура ко
мпьютера/arch-pc/lab06$ nasm -f elf variant.asm
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура ко
мпьютера/arch-pc/lab06$ ld -m elf_i386 -o variant variant.o
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура ко
мпьютера/arch-pc/lab06$ ./variant
Введите № студенческого билета:
12412351
Ваш вариант: 12
```

Рис. 4.15: Рисунок15



## 4.3 Ответы на 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
```

Для ввода строки с клавиатуры

3. Для чего используется инструкция “call atoi”?

Для вызова подпрограммы преобразования ASCII кода в число

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”?

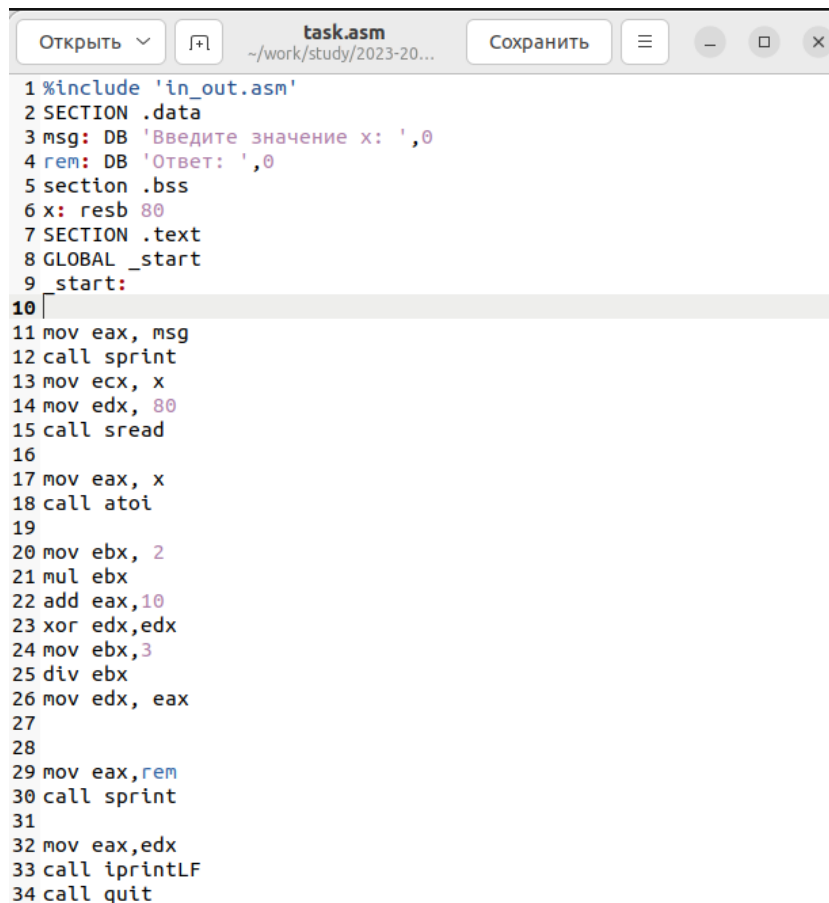
Для увеличения edx на 1

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

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

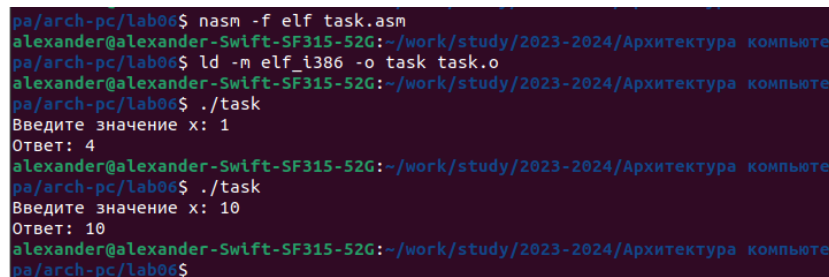
## 5 Задание для самостоятельной работы

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



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите значение x: ',0
4 rem: DB 'Ответ: ',0
5 section .bss
6 x: resb 80
7 SECTION .text
8 GLOBAL _start
9 _start:
10
11 mov eax, msg
12 call sprint
13 mov ecx, x
14 mov edx, 80
15 call sread
16
17 mov eax, x
18 call atoi
19
20 mov ebx, 2
21 mul ebx
22 add eax,10
23 xor edx,edx
24 mov ebx,3
25 div ebx
26 mov edx, eax
27
28
29 mov eax,rem
30 call sprint
31
32 mov eax,edx
33 call iprintLF
34 call quit
```

Рис. 5.1: Рисунок16



```
pa/arch-pc/lab06$ nasm -f elf task.asm
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab06$ ld -m elf_i386 -o task task.o
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab06$ ./task
Введите значение x: 1
Ответ: 4
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab06$ ./task
Введите значение x: 10
Ответ: 10
alexander@alexander-Swift-SF315-52G:~/work/study/2023-2024/Архитектура компьюте
pa/arch-pc/lab06$
```

Рис. 5.2: Рисунок17

Я взял пример под номером 1

## 6 Выводы

Мы научились выделять память под переменные, также проводить арифметические операции с ними и в конце написали свою программу, считающую функцию от значения переменной  $x$ .

## **Список литературы**