

# **Лабораторная работа №6**

Хорошева Алёна Евгеньевна

# 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>12</b>
<b>5</b>	<b>Выводы</b>	<b>20</b>
	<b>Список литературы</b>	<b>21</b>

## Список иллюстраций

4.1	“Текст программы lab6-1.asm” . . . . .	13
4.2	“Запуск исполняемого файла” . . . . .	13
4.3	“Регистр с числами” . . . . .	14
4.4	“Символ с кодом 10” . . . . .	14
4.5	“Запуск исполняемого файла” . . . . .	14
4.6	“Запуск программы lab6-2.asm” . . . . .	14
4.7	“Запуск программы с изменением символов на числа” . . . . .	15
4.8	“Замена функции <code>iPrintLF</code> на <code>iPrint</code> ” . . . . .	15
4.9	“Вычисление выражения №1” . . . . .	15
4.10	“Программа вычисления варианта” . . . . .	16
4.11	“Программа из самостоятельной работы” . . . . .	19

## **Список таблиц**

# 1 Цель работы

Освоить навыки работы с арифметическими операциями на языке ассемблера NASM.

## 2 Задание

1. Написать программу вычисления выражения  $\square = \square(\square)$ . Программа должна выводить выражение для вычисления, выводить запрос на ввод значения  $\square$ , вычислять заданное выражение в зависимости от введенного  $\square$ , выводить результат вычислений. Вид функции  $\square(\square)$  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  $\square_1$  и  $\square_2$  из таблицы.

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

## 3 Теоретическое введение

### 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`.

## 2. Арифметические операции

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 `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
```

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`.



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 имеет следующий вид:

```
inc 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.

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

```
neg <операнд>
```

Команда `neg` рассматривает свой операнд как число со знаком и меняет знак операнда на противоположный

```
mov ax,1 ; AX = 1
```

```
neg ax ; AX = -1
```

Умножение и деление, в отличие от сложения и вычитания, для знаковых и беззнаковых чисел имеют разный результат, поэтому существуют различные команды. Для беззнакового умножения используется команда `mul`

```
mul <операнд>
```

Для знакового умножения используется команда `imul`:

```
imul <операнд>
```

Для команд умножения один из сомножителей указывается в команде и должен находиться в регистре `eax`

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

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

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

В командах указывается только один операнд – делитель, который может быть регистром или константой

### 3. Перевод символа числа в десятичную 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, 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.

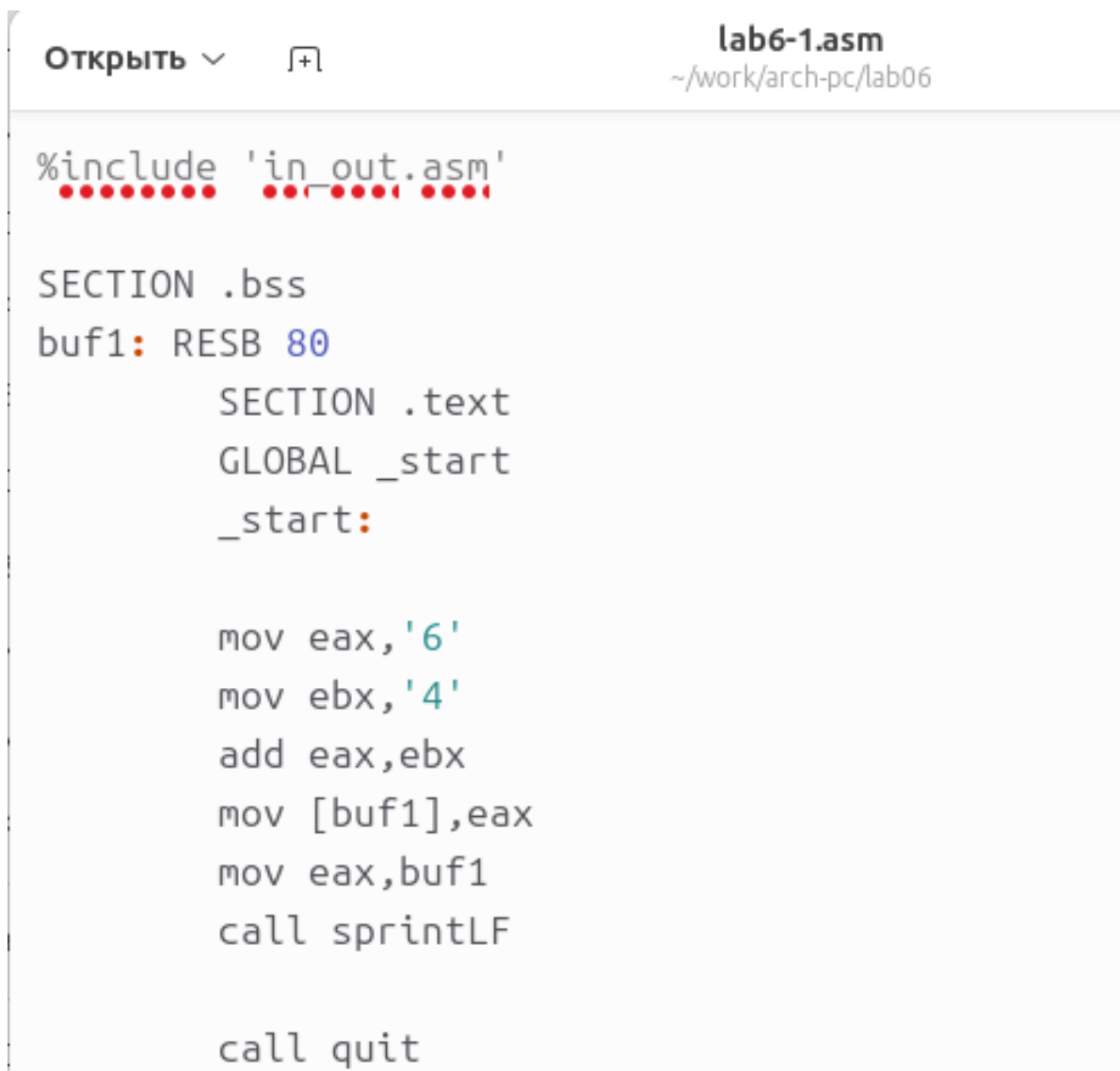
Для выполнения лабораторных работ в файле `in_out.asm` реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Это:

- `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

1. Создаю каталог для подпрограмм лабораторной работы и в нём командой *touch* создаю файл *lab6-1.asm*. Текст файла - это листинг 6.1: *Программа вывода значения регистра eax*. В программе используется внешний файл - *in\_out.asm*, предварительно скопированный в каталог *lab06*.



```
lab6-1.asm
~/work/arch-pc/lab06

%include 'in_out.asm'

SECTION .bss
buf1: RESB 80

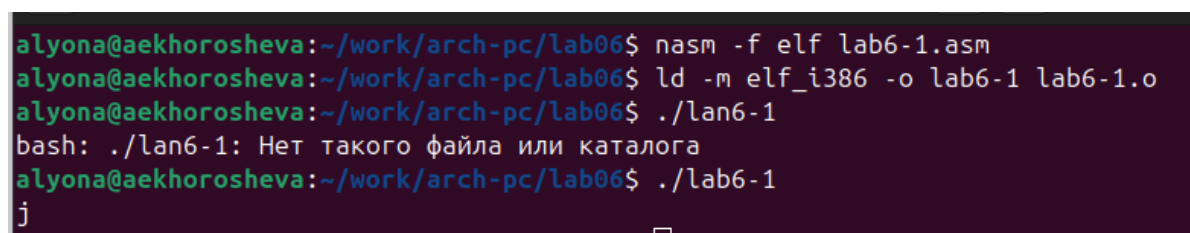
SECTION .text
GLOBAL _start
_start:

    mov eax, '6'
    mov ebx, '4'
    add eax, ebx
    mov [buf1], eax
    mov eax, buf1
    call sprintf

    call quit
```

Рис. 4.1: “Текст программы lab6-1.asm”

2. После создания и запуска исполняемого файла получаем результат - символ j, потому что его код - это сумма кодов 6 и 4(106).



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

Рис. 4.2: “Запуск исполняемого файла”

3. Изменим текст программы, вводя в регистр именно числа, а не числа в виде символа, то есть без кавычек. Создаём исполняемый файл и запускаем.

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

Рис. 4.3: “Регистр с числами”

После данного изменения программа должна вывести символ с кодом 10(а не само число 10)

10	12	0x0A	1010	LF, \n
----	----	------	------	--------

Рис. 4.4: “Символ с кодом 10”

Посмотрим на результат программы:

```
alyona@aeckhorosheva:~/work/arch-pc/lab06$ ./lab6-1  
alyona@aeckhorosheva:~/work/arch-pc/lab06$
```

Рис. 4.5: “Запуск исполняемого файла”

Мы видим, что после запуска вывелось две пустые строки, то есть сработал перенос строки

4. Создаём новый файл - lab6-2.asm для написания программы из листинга 6.2 с использованием функций из внешнего файла. На этот раз программа будет выводить значение регистра eax(106), а не символ, код которого программа получит в результате вычислений.

```
alyona@aeckhorosheva:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm  
alyona@aeckhorosheva:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o  
alyona@aeckhorosheva:~/work/arch-pc/lab06$ ./lab6-2  
106
```

Рис. 4.6: “Запуск программы lab6-2.asm”

5. Аналогично предыдущей программе изменим символы на числа и запустим исполняемый файл. Получаем результат: 10.

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

Рис. 4.7: “Запуск программы с изменением символов на числа”

6. В тексте программы меняем функцию `iprintLF` на `iprint` и запускаем исполняемый файл. Сразу заметно отличие в выводах этих функций - теперь после числа 10 *нет переноса строки*.

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

Рис. 4.8: “Замена функции `iprintLF` на `iprint`”

7. Создаю новый файл в исходном каталоге - `lab6-3.asm`. Запишем текст программы из листинга 6.3 для вычисления выражения  $\lfloor (5 \cdot 2 + 3) / 3 \rfloor$ . Запускаем исполняемый файл и получаем корректный результат:

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

Рис. 4.9: “Вычисление выражения №1”

8. Изменяю текст программы для вычисления другого выражения:  $\lfloor (4 \cdot 6 + 2) / 5 \rfloor$ . Программа должна вывести результат 5 с остатком от деления 1.

Проверим корректность работы программы запуском исполняемого файла:

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

9. Рассмотрим программу вычисления варианта задания по номеру студ. билета. В новом файле `variant.asm` запишем программу из листинга 6.4. Вариант будет вычисляться по формуле - *остаток от деления(<номер студ.билета> / 20) + 1*.

В моём случае номер билета делится на 20 без остатка, прибавляем 1 и получаем первый вариант. Запускаем программу:

```
alyona@aeckhorosheva:~/work/arch-pc/lab06$ nasm -f elf variant.asm
alyona@aeckhorosheva:~/work/arch-pc/lab06$ ld -m elf_i386 -o variant variant.o
alyona@aeckhorosheva:~/work/arch-pc/lab06$ ./variant
Введите № студенческого билета:
1132246820
Ваш вариант: 1
```

Рис. 4.10: “Программа вычисления варианта”

#### Ответы на вопросы:

1. *rem: DB 'Ваш вариант: ',0*

*mov eax,rem*

*call sprint*

Эти строки отвечают за вывод на экран текста “Ваш вариант:”

2. *mov esx, x* - инструкция перемещает адрес, хранящийся в *x*, в регистр *esx*. Тут будет строка для чтения её функцией *sread*.

*mov edx, 80* - устанавливает максимальное количество символов (т.е. 80) для чтения в регистре *edx*.

*call sread* - вызывает функцию *sread*, которая считывает строку из ввода в терминале и сохраняет её в области памяти, на которую указывает регистр *esx*.

3. *call atoi* - вызывает функцию из внешнего файла для того, чтобы преобразовать



ASCII код в число и записать его в регистр *eax*.

4. За вычисление варианта, то есть деления на 20 и прибавления единицы к остатку. Это инструкция *div* - для деления и *inc* - для прибавления 1. Ниже строки из листинга, отвечающие за эти вычисления:

- *xor edx,edx*
- *mov ebx,20*
- *div ebx*
- *inc edx*

5. Инструкция *div ebx* запишет остаток от деления на 20 в регистр *edx*.

6. Инструкция *inc edx* используется для прибавления единицы к остатку от деления, записанного в регистр *edx*.

7. За вывод на экран результата вычислений отвечают следующие строки:

- *mov eax,rem*
- *call sprint*
- *mov eax,edx*
- *call iprintLF*

### **Задание для самостоятельной работы**

Мой номер варианта = 1. Выражение  $(10 + 2\sqrt{2})/3$  и значения для  $x_1, x_2 = 1, 10$ .

Для написания программы вычисления выражения создаём новый файл - lab6-4.asm. Необходимо вычислить выражение. Для этого зададим переменную *x*:

- *SECTION .bss*
- *x: RESB 80*

и поля для ввода значений *x* и вывода результатов вычислений:

- *SECTION .data*
- *msg: DB 'Введите значение x:',0*
- *div: DB 'Результат:',0*
- *rem: DB 'Остаток от деления:',0*

Теперь начинается основной текст программы:

- *SECTION .text*

- *\*GLOBAL \_start\**

- *\*\_start:\**

- *mov eax, msg*

- *call sprintLF*

В этом фрагменте кода выводится текст на запрос значения *x*, введённого с клавиатуры.

Далее мы считываем введённое значение и преобразуем *x* из кода в число, помещаем его в регистр *eax*:

- *mov ecx, x*

- *mov edx, 80*

- *call sread*

- *mov eax,ecx*

- *call atoi*

Теперь вычислим выражение с помощью инструкций *mul*, *div*, *add*, *xor*, подставляя туда значения из заданной функции.

- *mov ebx,2*

- *mul ebx* ; умножили *x* на 2

- *add eax,10* ; прибавили 10

- *xor edx,edx* ; обнулили регистр *edx*

- *mov ebx,3* ;

- *div ebx*; поделили полученное ранее на 3

- *mov edi,eax* ; запись результата вычисления в 'edi'

Осталось вывести результат на экран и завершить программу.

- *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* ; завершение программы

После создания и запуска данной программы получаем верные результаты:

```

alyona@aeckhorosheva:~/work/arch-pc/lab06$ ./lab6-4
Введите значение x:
1
Результат: 4
Остаток от деления: 0
alyona@aeckhorosheva:~/work/arch-pc/lab06$ ./lab6-4
Введите значение x:
10
Результат: 10
Остаток от деления: 0

```

Рис. 4.11: “Программа из самостоятельной работы”

## 5 Выводы

В результате лабораторной работы были освоены практические навыки написания программы, которая вычисляет значение выражения, используя арифметические операции языка ассемблера.

Были изучены три способа адресации: регистровая, непосредственная и адресация памяти.

Также освоена теория по кодированию информации согласно ASCII.

## **Список литературы**