Лабораторная работа №7

Россохин Олег

Содержание

# 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:

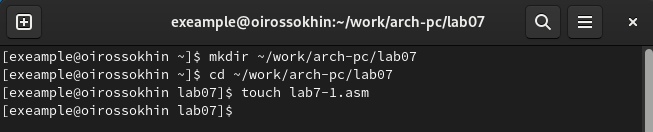
• Регистровая адресация – операнды хранятся в регистрах и в команде используются имена этих регистров, например: mov ax,bx.

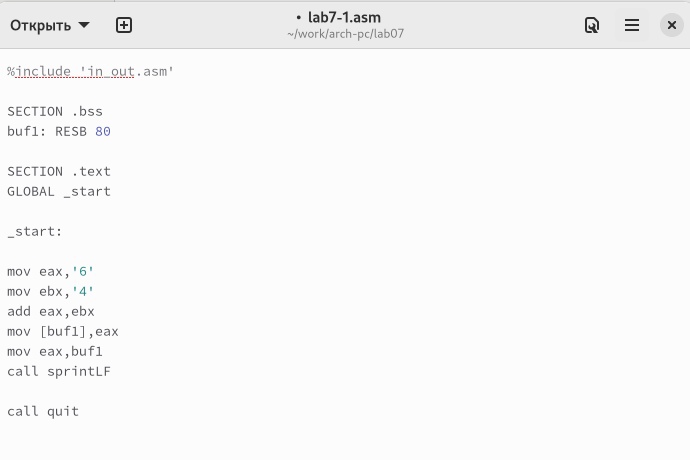
• Непосредственная адресация – значение операнда задается непосредственно в команде, Например: mov ax,2.

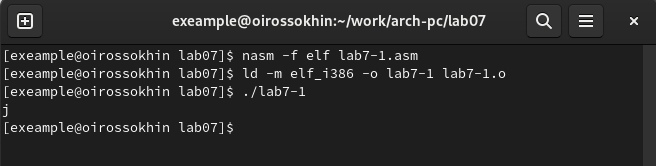
• Адресация памяти – операнд задает адрес в памяти. В команде указывается символическое обозначение ячейки памяти, над содержимым которой требуется выполнить операцию.

# 3 Выполнение лабораторной работы

1. Создадим каталог для программам лабораторной работы № 7, перейдем в него и создадим файл **lab7-1.asm**:



1. Рассмотрим примеры программ вывода символьных и численных значений. Программы будут выводить значения записанные в регистр eax. Введем в файл lab7-1.asm текст программы:  
   

Создадим исполняемый файл и запустим его: 

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, а код символа 4 – 00110100. Команда add eax,ebx запишет в регистр eax сумму кодов – 01101010, что в свою очередь является кодом символа **j**.

1. Далее изменим текст программы и вместо символов запишем в регистры числа. Исправим текст программы следующим образом:

заменим строки

mov eax,‘6’

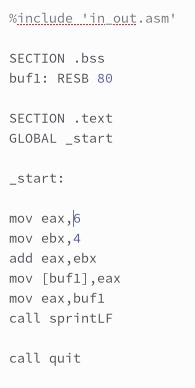
mov ebx,‘4’



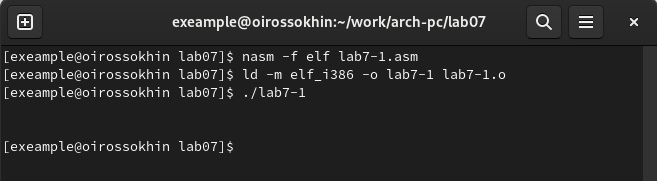
на строки

mov eax,6

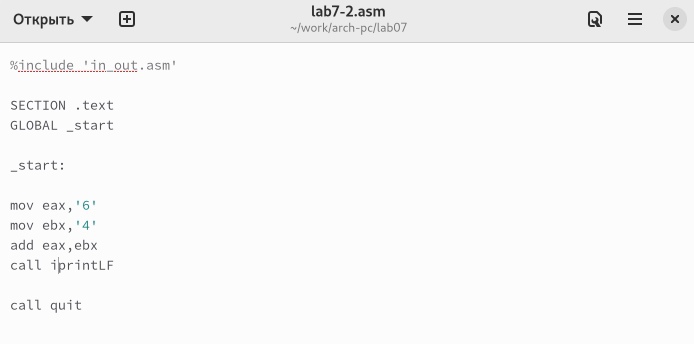
mov ebx,4

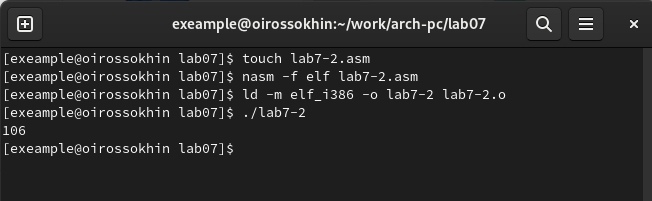


Создадим исполняемый файл и запустим его.



Как и в предыдущем случае при исполнении программы мы не получим число 10. В данном случае выводится символ с кодом 10.

1. 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 с использованием этих функций. Создадим файл **lab7-2.asm** в каталоге ~/work/arch-pc/lab07 и введем в него текст программы: 

Создадим исполняемый файл и запустите его. 

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 отличии от прошлой программы, функция **iprintLF** позволяет вывести число, а не символ, кодом которого является это число.

1. Аналогично предыдущему примеру изменим символы на числа. Заменим строки

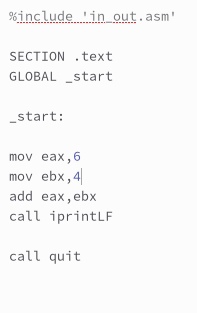
mov eax,‘6’

mov ebx,‘4’

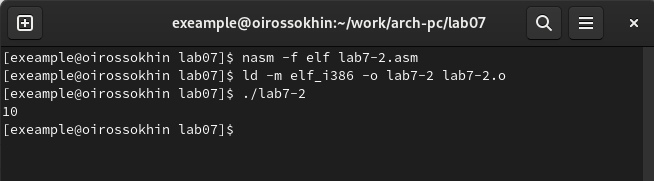
на строки

mov eax,6

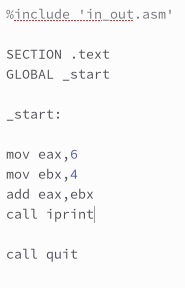
mov ebx,4

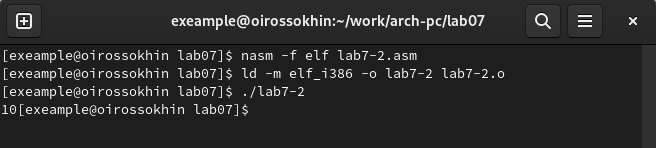


Создадим исполняемый файл и запустим его.



Далее заменим функцию iprintLF на iprint. Создадим исполняемый файл и запустим его.

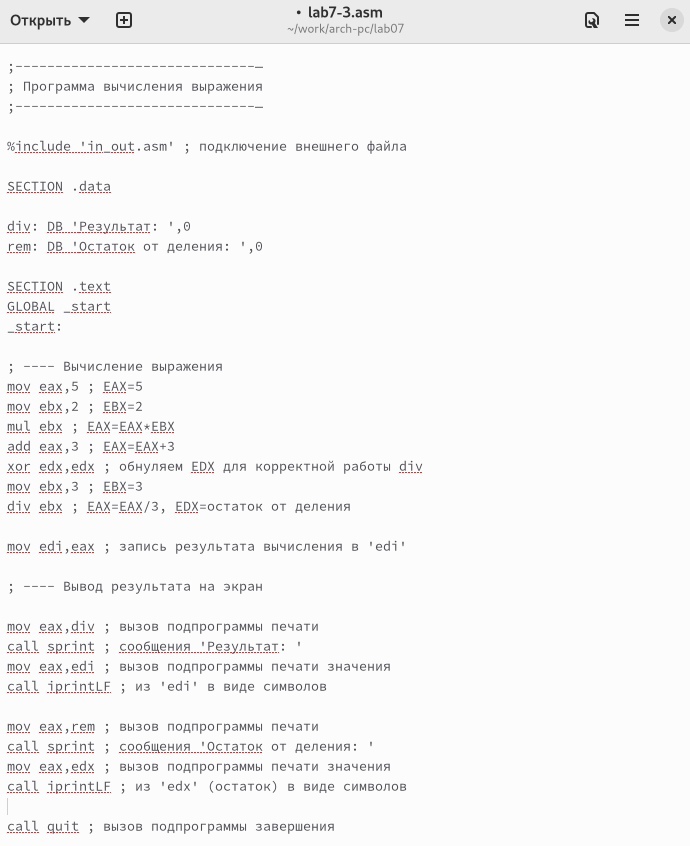


 наблюдается явное отличие двух программ в виде переноса строки результата вывода.

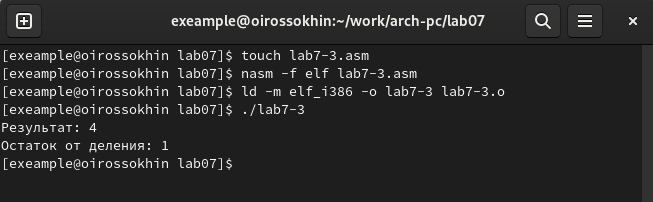
## 3.1 Выполнение арифметических операций в NASM

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

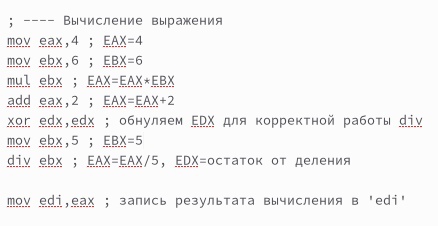
Создадим файл **lab7-3.asm** в каталоге ~/work/arch-pc/lab07 и запишем в него следующую программу:

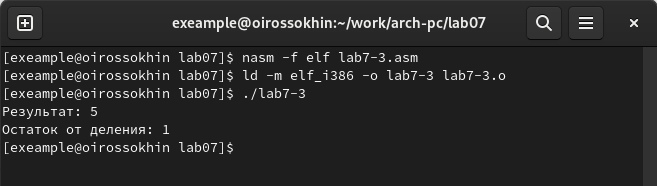


После чего сгенерируем исполняемый файл и запустим его:



Измениим текст программы для вычисления выражения 𝑓(𝑥) = (4 ∗ 6 + 2)/5.





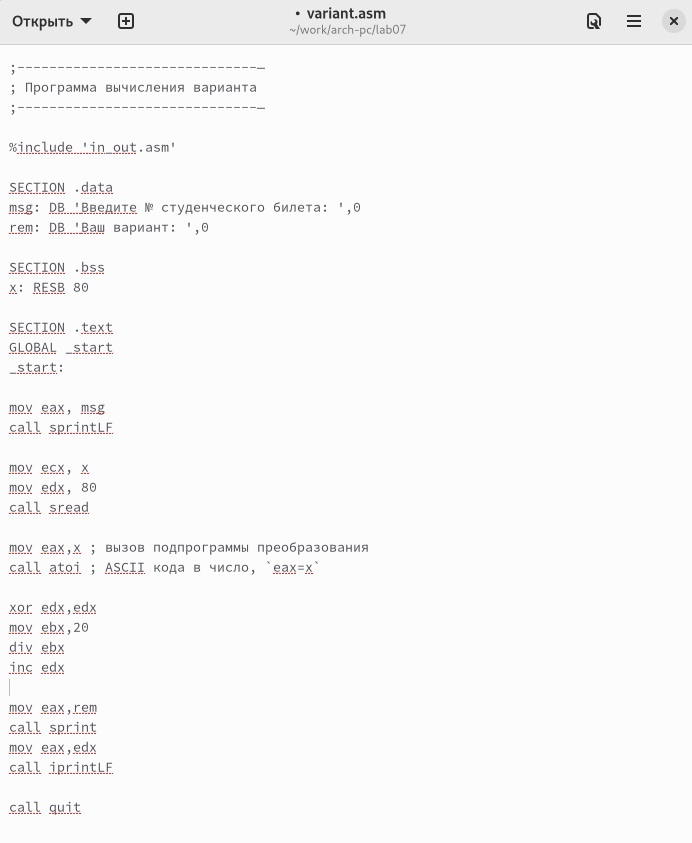
1. В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета, работающую по следующему алгоритму:

• вывести запрос на введение № студенческого билета

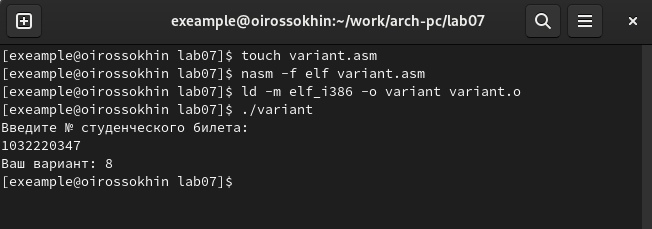
• вычислить номер варианта по формуле: (𝑆𝑛 mod 20) + 1, где 𝑆𝑛 – номер студенческого билета (В данном случае 𝑎 mod 𝑏 – это остаток от деления 𝑎 на 𝑏).

• вывести на экран номер варианта.

Создадим файл **variant.asm** в каталоге ~/work/arch-pc/lab07 и запишем в него следующую программу для вычисления номера варианта:



После мы создадим и запустим исполняемый файл, введя свой номер студенческого билета:

 Путем вычислений программы я получил 8 вариант.

# 4 Теоретические вопросы

#### Какие строки листинга 7.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’?

rem: DB ‘Ваш вариант:’,0 call sprintLF

#### Для чего используется следующие инструкции? nasm mov ecx, x

* mov edx, 80 call sread

Функция nasm определяет язык исполняемого файла Команда mov задает значение переменной ecx - регистр, позволяющий обращаться к данным mov edx, 80 - длина вводимой строки в байтах.

#### Для чего используется инструкция “call atoi”?

Команда call atoi позволяет преобразовать ASCII код в число и записать его в регистр eax, перед вызовом atoi в регистр eax необходимо записать число

#### Какие строки листинга 7.4 отвечают за вычисления варианта?

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

#### В какой регистр записывается остаток от деления при выполнении инструкции “div ebx”?

Остаток от деления при выполнении инструкции div ebx записывается в регистр edx.

#### Для чего используется инструкция “inc edx”?

Инструкция inc edx увеличивает значение регистра edx на единицу.

#### Какие строки листинга 7.4 отвечают за вывод на экран результата вычислений?

call sprint

call iprintLF

# 5 Выводы

По итогам проделанной работы мы освоили некоторые инструкции языка ассемблера NASM.

# 6 Список литературы

https://esystem.rudn.ru/pluginfile.php/1584637/mod\_resource/content/1/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%967.pdf