

Лабораторная работа №7

Россохин Олег

Содержание

1	Цель работы	5
2	Теоретическое введение	6
2.1	Адресация в NASM	6
3	Выполнение лабораторной работы	7
3.1	Выполнение арифметических операций в NASM	13
4	Теоретические вопросы	19
5	Выводы	21
6	Список литературы	22

Список иллюстраций

Список таблиц

1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

2 Теоретическое введение

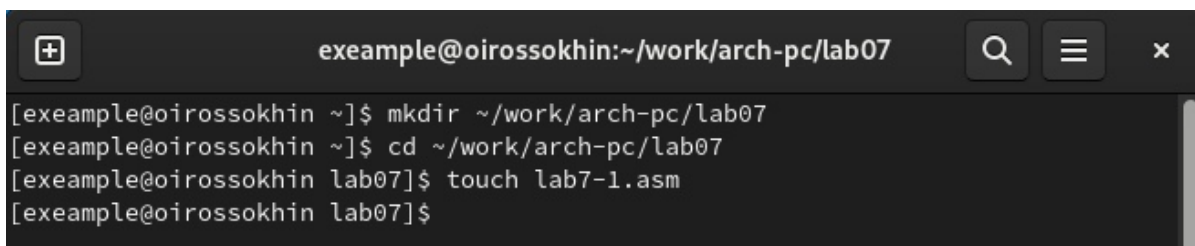
2.1 Адресация в NASM

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.

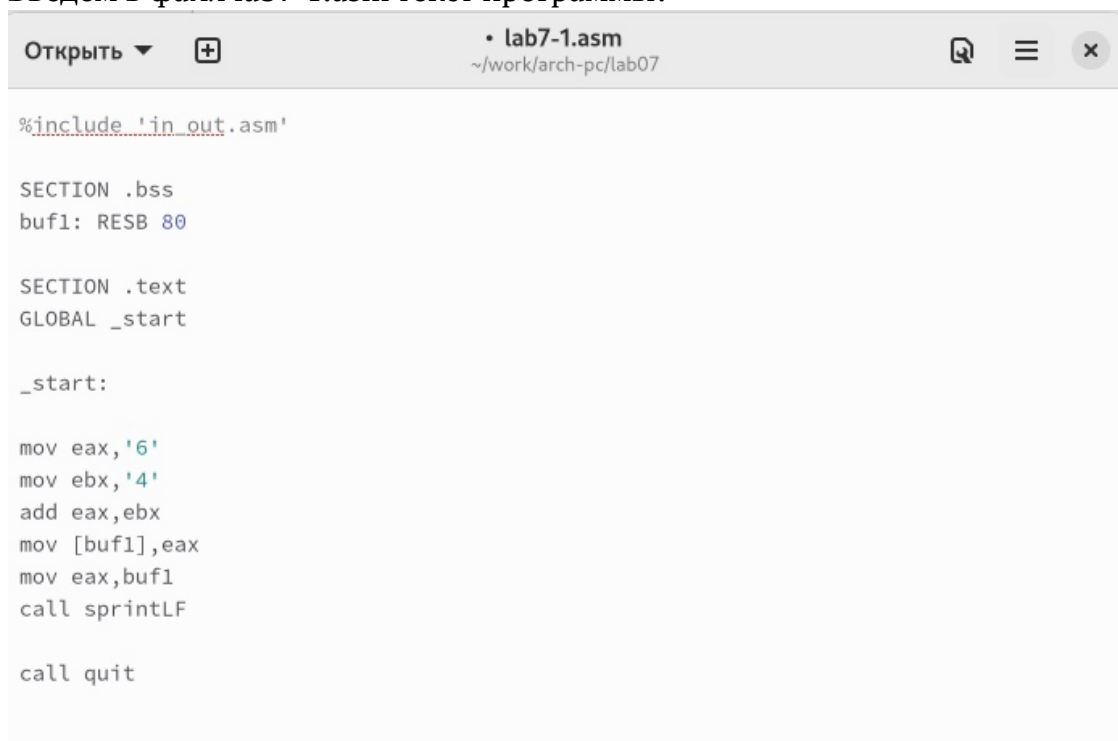
3 Выполнение лабораторной работы

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



```
exeample@oirossokhin:~/work/arch-pc/lab07
[exeample@oirossokhin ~]$ mkdir ~/work/arch-pc/lab07
[exeample@oirossokhin ~]$ cd ~/work/arch-pc/lab07
[exeample@oirossokhin lab07]$ touch lab7-1.asm
[exeample@oirossokhin lab07]$
```

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



```
Открыть ▾ + • lab7-1.asm ~/work/arch-pc/lab07
%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
```

```
example@oirossokhin:~/worl
[example@oirossokhin lab07]$ nasm -f elf lab7-1.o
[example@oirossokhin lab07]$ ld -m elf_i386 -o lab7-1
[example@oirossokhin lab07]$ ./lab7-1
j
[example@oirossokhin lab07]$
```

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**.

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

заменим строки

```
mov eax,'6'
```

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



```
%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
```

на строки

mov eax,6

mov ebx,4

```

%include 'in_out.asm'

SECTION .bss
buf1: RESB 80

SECTION .text
GLOBAL _start

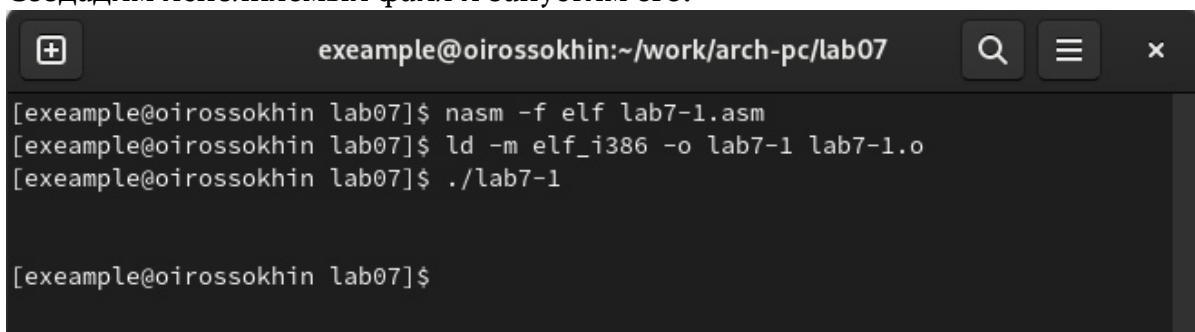
_start:

mov eax,6
mov ebx,4
add eax,ebx
mov [buf1],eax
mov eax,buf1
call sprintLF

call quit

```

Создадим исполняемый файл и запустим его.



```

example@oirossokhin:~/work/arch-pc/lab07
[example@oirossokhin lab07]$ nasm -f elf lab7-1.asm
[example@oirossokhin lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[example@oirossokhin lab07]$ ./lab7-1

[example@oirossokhin lab07]$

```

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

4. Как отмечалось выше, для работы с числами в файле `in_out.asm` реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Пре-

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

```
Открыть + lab7-2.asm ~/work/arch-pc/lab07

%include 'in_out.asm'

SECTION .text
GLOBAL _start

_start:

mov eax, '6'
mov ebx, '4'
add eax, ebx
call iprintLF

call quit
```

```
example@oirossokhin:~/work/arch-pc/lab07$ touch lab7-2.asm
example@oirossokhin:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
example@oirossokhin:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
example@oirossokhin:~/work/arch-pc/lab07$ ./lab7-2
106
example@oirossokhin:~/work/arch-pc/lab07$
```

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

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

```
mov eax, '6'
mov ebx, '4'
на строки
mov eax, 6
```

mov ebx,4

```
%include 'in_out.asm'

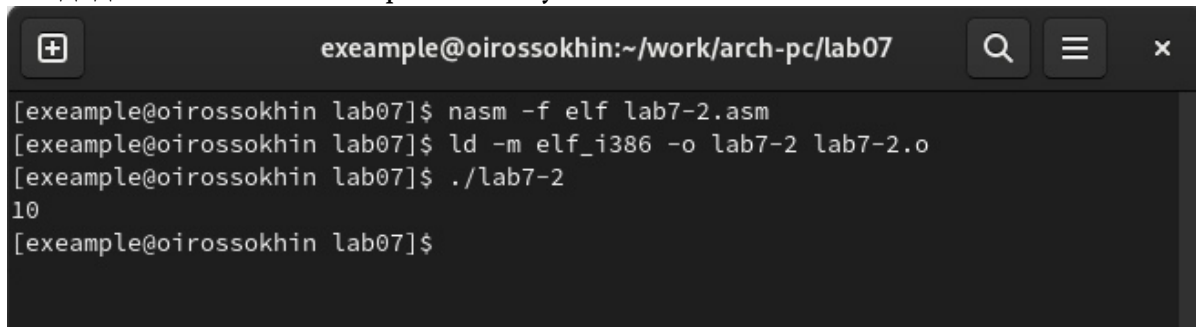
SECTION .text
GLOBAL _start

_start:

mov eax,6
mov ebx,4
add eax,ebx
call iprintLF

call quit
```

Создадим исполняемый файл и запустим его.

A terminal window with a dark background. The title bar shows the user 'example@oirossokhin' and the directory '~/work/arch-pc/lab07'. The terminal contains the following commands and output:

```
[example@oirossokhin lab07]$ nasm -f elf lab7-2.asm
[example@oirossokhin lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[example@oirossokhin lab07]$ ./lab7-2
10
[example@oirossokhin lab07]$
```

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

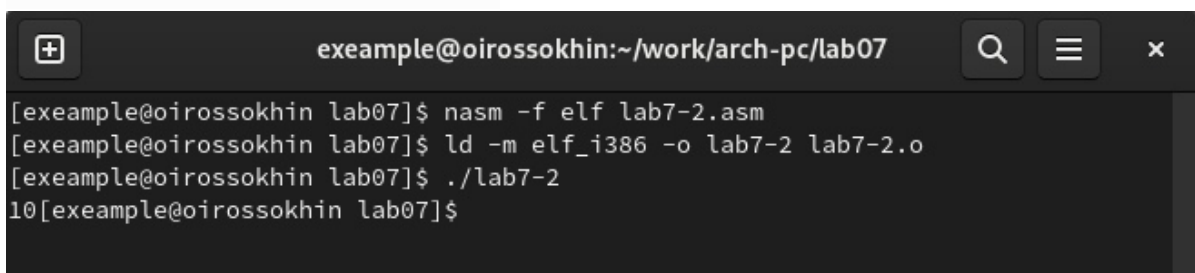
```
%include 'in_out.asm'

SECTION .text
GLOBAL _start

_start:

mov eax,6
mov ebx,4
add eax,ebx
call iprint

call quit
```



```
example@oirossokhin:~/work/arch-pc/lab07
[example@oirossokhin lab07]$ nasm -f elf lab7-2.asm
[example@oirossokhin lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[example@oirossokhin lab07]$ ./lab7-2
10[example@oirossokhin lab07]$
```

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 $5 \times 2 + 3$ / 3

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

```
Открыть ▾  • lab7-3.asm
~/work/arch-pc/lab07   

;-----
; Программа вычисления выражения
;-----

%include 'in_out.asm' ; подключение внешнего файла

SECTION .data

div: DB 'Результат: ',0
rem: DB 'Остаток от деления: ',0

SECTION .text
GLOBAL _start
_start:

; ---- Вычисление выражения
mov eax,5 ; EAX=5
mov ebx,2 ; EBX=2
mul ebx ; EAX=EAX*EBX
add eax,3 ; EAX=EAX+3
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,3 ; EBX=3
div ebx ; EAX=EAX/3, EDX=остаток от деления

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

; ---- Вывод результата на экран

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 ; вызов подпрограммы завершения
```

После чего сгенерируем исполняемый файл и запустим его:

```
example@oirossokhin:~/work/arch-pc/lab07
[example@oirossokhin lab07]$ touch lab7-3.asm
[example@oirossokhin lab07]$ nasm -f elf lab7-3.asm
[example@oirossokhin lab07]$ ld -m elf_i386 -o lab7-3 lab7-3.o
[example@oirossokhin lab07]$ ./lab7-3
Результат: 4
Остаток от деления: 1
[example@oirossokhin lab07]$
```

Изменим текст программы для вычисления выражения $\text{X}(\text{X}) = (4 \times 6 + 2)/5$.

```
; ---- Вычисление выражения
mov eax,4 ; EAX=4
mov ebx,6 ; EBX=6
mul ebx ; EAX=EAX*EBX
add eax,2 ; EAX=EAX+2
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,5 ; EBX=5
div ebx ; EAX=EAX/5, EDX=остаток от деления

mov edi,eax ; запись результата вычисления в 'edi'
```

```
example@oirossokhin:~/work/arch-pc/lab07
[example@oirossokhin lab07]$ nasm -f elf lab7-3.asm
[example@oirossokhin lab07]$ ld -m elf_i386 -o lab7-3 lab7-3.o
[example@oirossokhin lab07]$ ./lab7-3
Результат: 5
Остаток от деления: 1
[example@oirossokhin lab07]$
```

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: $(\text{студ. номер} \bmod 20) + 1$, где студ. номер – номер студенческого билета (В данном случае $\text{студ. номер} \bmod 20$ – это остаток от деления студ. номера на 20).
- 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:


```
Открыть ▾ + • variant.asm
~/work/arch-pc/lab07

;-----
; Программа вычисления варианта
;-----

%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 sprintf

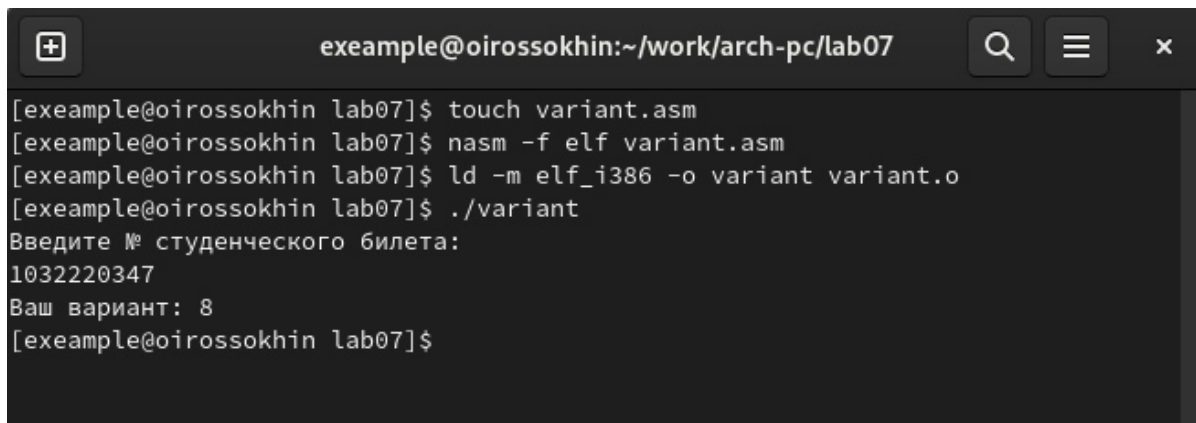
mov ecx, x
mov edx, 80
call sread

mov eax, x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x`

xor edx, edx
mov ebx, 20
div ebx
inc edx
|
mov eax, rem
call sprintf
mov eax, edx
call iprintLF

call quit
```

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



```
example@oirossokhin:~/work/arch-pc/lab07
[example@oirossokhin lab07]$ touch variant.asm
[example@oirossokhin lab07]$ nasm -f elf variant.asm
[example@oirossokhin lab07]$ ld -m elf_i386 -o variant variant.o
[example@oirossokhin lab07]$ ./variant
Введите № студенческого билета:
1032220347
Ваш вариант: 8
[example@oirossokhin lab07]$
```

Путем вычислений программы я получил 8 вариант.

4 Теоретические вопросы

1.

4.0.0.1 Какие строки листинга 7.4 отвечают за вывод на экран сообщения 'Ваш вариант:'?

```
rem: DB 'Ваш вариант:',0 call sprintLF
```

2.

4.0.0.2 Для чего используются следующие инструкции? `push mov ecx, x`

```
mov edx, 80 call sread
```

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

3.

4.0.0.3 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

4.

4.0.0.4 Какие строки листинга 7.4 отвечают за вычисления варианта?

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

5.

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

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

6.

4.0.0.6 Для чего используется инструкция “inc edx”?

Инструкция inc edx увеличивает значение регистра edx на единицу.

7.

4.0.0.7 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%