

Лабораторная работа №7

Арифметические операции в NASM

Вершинина Ангелина Алексеевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Символьные и численные данные в NASM	8
4.2	Выполнение арифметических операций в NASM	12
4.3	Задание для самостоятельной работы	15
5	Выводы	18
	Список литературы	19

Список иллюстраций

4.1	Создание каталога и файла	8
4.2	Создание и запуск исполняемого файла программы 1	9
4.3	Замена в тексте программы символов на числа	9
4.4	Создание и запуск исполняемого файла программы 2	10
4.5	Создание файла программы 3	10
4.6	Создание и запуск исполняемого файла программы 3	10
4.7	Замены строк	11
4.8	Создание и запуск исполняемого файла программы	11
4.9	Замена функции	12
4.10	Создание и запуск исполняемого файла программы	12
4.11	Создание файла программы	13
4.12	Создание и запуск исполняемого файла программы	13
4.13	Изменение текста программы	13
4.14	Создание и запуск исполняемого файла программы	14
4.15	Работа программы	14
4.16	Создание файла	16
4.17	Проверка работы программы на x1	16
4.18	Проверка работы программы на x2	16

Список таблиц

1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM

2 Задание

Написание программ с использование арифметических действий

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 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:

1. Регистровая адресация – операнды хранятся в регистрах и в команде используются имена этих регистров, например: `mov ax,bx`.
2. Непосредственная адресация – значение операнда задается непосредственно в команде, Например: `mov ax,2`.
3. Адресация памяти – операнд задает адрес в памяти. В команде указывается символическое обозначение ячейки памяти, над содержимым которой требуется выполнить операцию.

4 Выполнение лабораторной работы

4.1 Символьные и численные данные в NASM

Создам каталог для программ лабораторной работы №7, перейду в него и создам файл lab7-1.asm (рис. 4.1)



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

Рис. 4.1: Создание каталога и файла

Пример программы 1

Программы буду выводить значения записанные в регистр еах.

Введу в файл lab7-1.asm текст программы из листинга 7.1. В данной программе в регистр еах записывается символ 6, в регистр ебх символ 4. Далее к значению в регистре еах прибавляю значение регистра ебх и результат сложения запишу в регистр еа. Далее вывожу результат. Так как для работы функции sprintLF в регистр еах должен быть записан адрес, необходимо использовать дополнительную переменную. Для этого запишу значение регистра еах в переменную buf1, а затем запишу адрес переменной buf1 в регистр еах и вызову функцию sprintLF.

Далее создам исполняемый файл и запущу его. (рис. 4.2) Результатом является символ j. Это происходит потому, что код символа 6 равен 00110110 в двоичном представлении (или 54 в десятичном представлении), а код символа 4 – 00110100 (52). Команда add еах,ебх запишет в регистр еах сумму кодов – 01101010 (106),

что в свою очередь является кодом символа j.

```
[aavershinina@fedora lab07]$ nasm -f elf lab7-1.asm
[aavershinina@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[aavershinina@fedora lab07]$ ./lab7-1
j
[aavershinina@fedora lab07]$
```

Рис. 4.2: Создание и запуск исполняемого файла программы 1

Пример программы 2

Далее изменю текст программы и вместо символов, запишу в регистры числа.

Исправлю текст программы 1 следующим образом: (рис. 4.3)

заменю строки

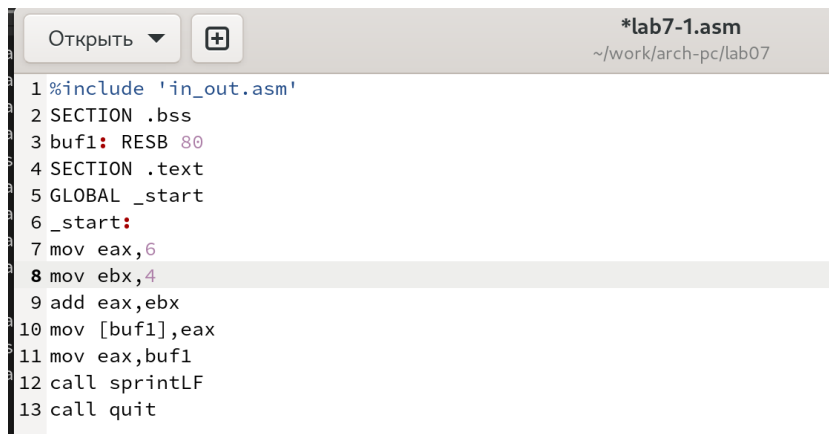
```
mov eax, '6'
```

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

на строки

```
mov eax, 6
```

```
mov ebx, 4
```



```
Открыть + *lab7-1.asm
~/work/arch-pc/lab07
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. 4.3: Замена в тексте программы символов на числа

Создам исполняемый файл и запущу его. (рис. 4.4) Как и в предыдущем случае при исполнении программы мы не получили число 10. В данном случае выводится символ с кодом 10. Пользуясь таблицей ASCII определю какому символу

соответствует код 10. Это символ перевода строки и он не отображается при выводе на экран.

```
[aavershinina@fedora lab07]$ nasm -f elf lab7-1.asm
[aavershinina@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[aavershinina@fedora lab07]$ ./lab7-1

[aavershinina@fedora lab07]$
```

Рис. 4.4: Создание и запуск исполняемого файла программы 2

Пример программы 3

Создам файл lab7-2.asm в каталоге ~/work/arch-pc/lab07 (рис. 4.5) и введу в него текст программы из листинга 7.2. Создам исполняемый файл и запущу его.(рис. 4.6) В результате работы программы получилось число 106. В данном случае, как и в первом, команда add складывает коды символов '6' и '4' (54+52=106). Однако, в отличии от программы из листинга 7.1, функция iprintLF позволяет вывести число, а не символ, кодом которого является это число.

```
[aavershinina@fedora lab07]$ touch ~/work/arch-pc/lab07/lab7-2.asm
```

Рис. 4.5: Создание файла программы 3

```
[aavershinina@fedora lab07]$ nasm -f elf lab7-2.asm
[aavershinina@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[aavershinina@fedora lab07]$ ./lab7-2
106
[aavershinina@fedora lab07]$
```

Рис. 4.6: Создание и запуск исполняемого файла программы 3

Аналогично предыдущему примеру изменим символы на числа.

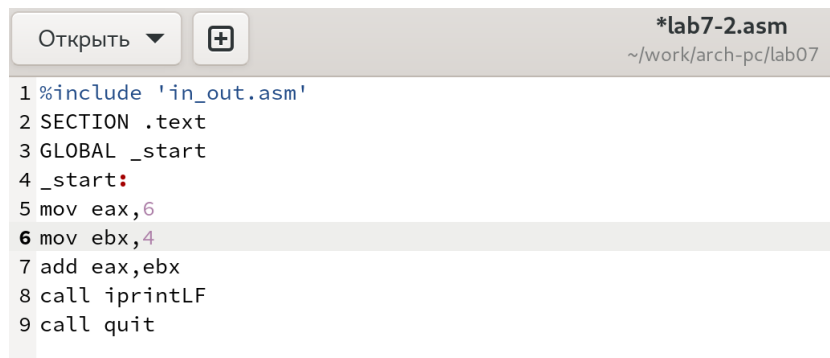
Заменю строки

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

на строки (рис. 4.7)

```
mov eax,6
```

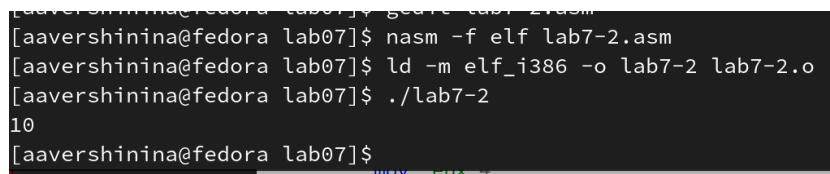
```
mov ebx,4
```



```
Открыть ▼ [+]  
*lab7-2.asm  
~/work/arch-pc/lab07  
1 %include 'in_out.asm'  
2 SECTION .text  
3 GLOBAL _start  
4 _start:  
5 mov eax,6  
6 mov ebx,4  
7 add eax,ebx  
8 call iprintLF  
9 call quit
```

Рис. 4.7: Замены строк

Создам исполняемый файл и запущу его. (рис. 4.8) результат выполнения программы - 10.



```
[aavershinina@fedora lab07]~$ gcc -c lab7-2.asm  
[aavershinina@fedora lab07]$ nasm -f elf lab7-2.asm  
[aavershinina@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o  
[aavershinina@fedora lab07]$ ./lab7-2  
10  
[aavershinina@fedora lab07]$
```

Рис. 4.8: Создание и запуск исполняемого файла программы

Заменю функцию iprintLF на iprint. (рис. 4.9) Создам исполняемый файл и запущу его. (рис. 4.10) Отличие вывода iprintLF и iprint: в первом случае после вывода происходит перенос строки, а втором нет переноса на новую строку

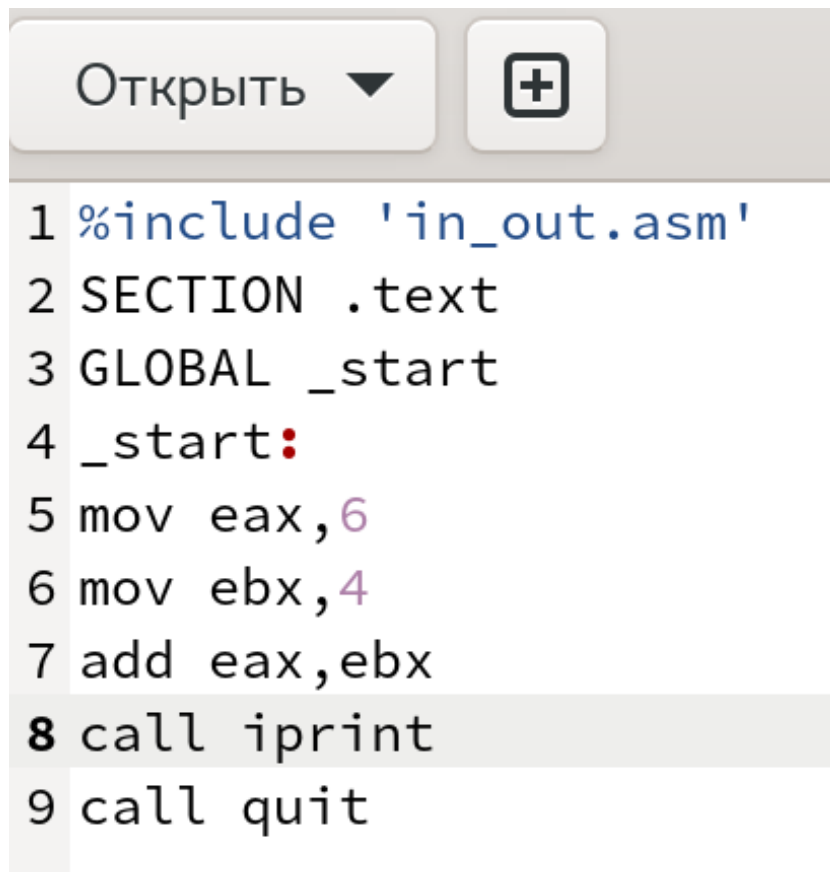


Рис. 4.9: Замена функции

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

Рис. 4.10: Создание и запуск исполняемого файла программы

4.2 Выполнение арифметических операций в NASM

В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения $\text{f}(x) = (5 \cdot x + 3)/3$

Создам файл lab7-3.asm в каталоге ~/work/arch-pc/lab07 (рис. 4.11)

```
[aavershinina@fedora lab07]$ ./lab7-3
10[aavershinina@fedora lab07]$ touch ~/work/arch-pc/lab07/lab7-3.asm
[aavershinina@fedora lab07]$
```

Рис. 4.11: Создание файла программы

Внимательно изучу текст программы из листинга 7.3 и введу в lab7-3.asm. Создам исполняемый файл и запущу его (рис. 4.12)

```
[aavershinina@fedora lab07]$ nasm -f elf lab7-3.asm
[aavershinina@fedora lab07]$ ld -m elf_i386 -o lab7-3 lab7-3.o
[aavershinina@fedora lab07]$ ./lab7-3
Результат: 4
Остаток от деления: 1
[aavershinina@fedora lab07]$
```

Рис. 4.12: Создание и запуск исполняемого файла программы

Изменяю текст программы для вычисления выражения $(4 \times 6 + 2)/5$. (рис. 4.13) Создам исполняемый файл и проверю его работу. (рис. 4.14)

```
Открыть ▼ + *lab7-3.asm ~/work/arch-pc/l
4 rem: DB 'Остаток от деления: ',0
5 SECTION .text
6 GLOBAL _start
7 _start:
8 ; ---- Вычисление выражения
9 mov eax,4 ; EAX=4
10 mov ebx,6 ; EBX=6
11 mul ebx ; EAX=EAX*EBX
12 add eax,2 ; EAX=EAX+2
13 xor edx,edx ; обнуляем EDX для корректной работы div
14 mov ebx,5 ; EBX=5
15 div ebx ; EAX=EAX/5, EDX=остаток от деления
16 mov edi,eax ; запись результата вычисления в 'edi'
17 ; ---- Вывод результата на экран
18 mov eax,div ; вызов подпрограммы печати
19 call sprint ; сообщения 'Результат: '
20 mov eax,edi ; вызов подпрограммы печати значения
21 call iprintLF ; из 'edi' в виде символов
22 mov eax,rem ; вызов подпрограммы печати
23 call sprint ; сообщения 'Остаток от деления: '
24 mov eax,edx ; вызов подпрограммы печати значения
```

Рис. 4.13: Изменение текста программы

```
[aavershinina@fedora lab07]$ nasm -f elf lab7-3.asm
[aavershinina@fedora lab07]$ ld -m elf_i386 -o lab7-3 lab7-3.o
[aavershinina@fedora lab07]$ ./lab7-3
Результат: 5
Остаток от деления: 1
[aavershinina@fedora lab07]$
```

Рис. 4.14: Создание и запуск исполняемого файла программы

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

1. вывести запрос на введение No студенческого билета
2. вычислить номер варианта по формуле: $(\text{№} \bmod 20) + 1$, где № – номер студенческого билета (В данном случае $\text{№} \bmod \text{№}$ – это остаток от деления № на №).
3. 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`.

Создам файл `variant.asm` в каталоге `~/work/arch-pc/lab07`. Внимательно изучу текст программы из листинга 7.4 и введу в файл `variant.asm`. Создам исполняемый файл и запущу его. (рис. 4.15)

```
[aavershinina@fedora lab07]$ nasm -f elf variant.asm
[aavershinina@fedora lab07]$ ld -m elf_i386 -o variant variant.o
[aavershinina@fedora lab07]$ ./variant
Введите No студенческого билета:
1132221891
Ваш вариант: 12
[aavershinina@fedora lab07]$
```

Рис. 4.15: Работа программы

Ответы на вопросы

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
3. `call atoi` используется для вызова подпрограммы из внешнего файла, которая преобразует ASCII-код символа в целое число и записывает результат в регистр `eax`
4. 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`
6. Инструкция `inc edx` увеличивает значение регистра `edx` на 1
7. За вывод на экран результатов вычислений отвечают строки

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

4.3 Задание для самостоятельной работы

Необходимо написать программу вычисления выражения $x = x(x)$. Программа должна выводить выражение для вычисления, выводить запрос на ввод значения

✘, вычислять заданное выражение в зависимости от введенного ✘, выводить результат вычислений. Вид функции $\text{✘}(\text{✘}) = (8x-6)/2$

Создам файл для выполнения задания (рис. 4.16)

```
[aavershinina@fedora lab07]$ touch ~/work/arch-pc/lab07/sr.asm  
[aavershinina@fedora lab07]$ gedit sr.asm
```

Рис. 4.16: Создание файла

Создам исполняемый файл и проверю его работу для значений $\text{✘}1 = 1$ и $\text{✘}2 = 5$.
(рис. 4.17 и 4.18)

```
[aavershinina@fedora lab07]$ nasm -f elf sr.asm  
[aavershinina@fedora lab07]$ ld -m elf_i386 -o sr sr.o  
[aavershinina@fedora lab07]$ ./sr  
Введите число:  
1  
Результат: 1  
[aavershinina@fedora lab07]$
```

Рис. 4.17: Проверка работы программы на x1

```
[aavershinina@fedora lab07]$ ./sr  
Введите число:  
5  
Результат: 17  
[aavershinina@fedora lab07]$
```

Рис. 4.18: Проверка работы программы на x2

Листинг программы

```
%include 'in_out.asm'  
  
SECTION .data  
fun: DB 'F(x)=(8x-6)/2',0  
msg: DB 'Введите число: ',0  
rem: DB 'Результат: ',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 ; eax=x
call atoi ; ASCII кода в число, `eax=x`
mov ebx, 8 ; ebx = 8
mul ebx ; eax = eax*ebx
sub eax, 6 ; eax - 6
mov ebx, 2 ; ebx = 2
div ebx ; eax = eax/2
mov edi, eax ; запись результата в edi
mov eax, rem
call sprint
mov eax, edi
call iprintLF
call quit

```

5 Выводы

В результате проделанной работы я освоила арифметических инструкций языка ассемблера NASM.

Список литературы