

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 7

дисциплина: *Архитектура компьютера*

Студент: Замбалова Дина Владимировна

Группа: НПИбд-01-22

МОСКВА

2022 г.

Содержание

1 Цель работы	5
2 Задание	6
3 Теоретическое введение.....	7
4 Выполнение лабораторной работы	13
5 Выводы	23
Список литературы	24

Список иллюстраций

4.1 Создание файла	13
4.2 Текст программы из листинга 7.1	13
4.3 Создание и запуск исполняемого файла.....	14
4.4 Замена строк	14
4.5 Создание и запуск исполняемого файла.....	14
4.6 Создание файла	15
4.7 Текст программы из листинга 7.2.....	15
4.8 Создание и запуск исполняемого файла.....	15
4.9 Замена строк	16
4.10 Создание и запуск исполняемого файла	16
4.11 Создание и запуск исполняемого файла	16
4.12 Создание файла	16
4.13 Текст программы из листинга 7.3	17
4.14 Текст программы из листинга 7.3	17
4.15 Создание и запуск исполняемого файла	17
4.16 Замена строк.....	18
4.17 Создание и запуск исполняемого файла.....	18
4.18 Создание файла	18
4.19 Текст программы из листинга 7.4	19
4.20 Текст программы из листинга 7.4	19
4.21 Создание и запуск исполняемого файла	20
4.22 Создание файла	21
4.23 Текст программы.....	21
4.24 Текст программы.....	22
4.25 Создание и запуск исполняемого файла	22

Список таблиц

3.1 Регистры используемые командами умножения в Nasm	10
3.2 Регистры используемые командами деления в Nasm.....	11

1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

2 Задание

Порядок выполнения работы.

Задание для самостоятельной работы:

1. 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 7.3 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 7.3.

3 Теоретическое введение

3.1 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`.

3.2 Арифметические операции в NASM

3.2.1 Целочисленное сложение add

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 <опреанд_1>,<операнд_2>
```

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.
```

3.2.2 Целочисленное вычитание sub

Команда целочисленного вычитания sub (от англ. subtraction – вычитание) работает аналогично команде add и выглядит следующим образом:

```
sub <опреанд_1>,<операнд_2>
```

Так, например, команда sub ebx,5 уменьшает значение регистра ebx на 5 и записывает результат в регистр ebx.

3.2.3 Команды инкремента и декремента

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 <операнд>
```


`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.

3.2.4 Команда изменения знака операнда `neg`

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

`neg <операнд>`

Команда `neg` рассматривает свой операнд как число со знаком и меняет знак операнда на противоположный. Операндом может быть регистр или ячейка памяти любого размера.

`mov ax,1 ; AX = 1`

`neg ax ; AX = -1`

3.2.5 Команды умножения `mul` и `imul`

Умножение и деление, в отличие от сложения и вычитания, для знаковых и беззнаковых чисел производиться по-разному, поэтому существуют различные команды.

Для беззнакового умножения используется команда `mul` (от англ. multiply – умножение):

`mul <операнд>`

Для знакового умножения используется команда `imul`:

`imul <операнд>`

Для команд умножения один из сомножителей указывается в команде и должен находиться в регистре или в памяти, но не может быть непосредственным операндом. Второй сомножитель в команде явно не указывается и должен находиться в регистре `EAX, AX` или `AL`, а результат помещается в регистры `EDX:EAX`, `DX:AX` или `AX`, в зависимости от размера операнда 3.1.

Таблица 3.1. Регистры используемые командами умножения в Nasm

Размер операнда	Неявный множитель	Результат умножения
1 байт	AL	AX
2 байт	AX	DX:AX
4 байт	EAX	EDX:EAX

Пример использования инструкции mul:

a dw 270

mov ax, 100 ; AX = 100

mul a ; AX = AX*a,

mul bl ; AX = AL*BL

mul ax ; DX:AX = AX*AX

3.2.6 Команды деления div и idiv.

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

div <делитель> ; Беззнаковое деление

idiv <делитель> ; Знаковое деление

В командах указывается только один операнд – делитель, который может быть регистром или ячейкой памяти, но не может быть непосредственным операндом. Местоположение делимого и результата для команд деления зависит от размера делителя. Кроме того, так как в результате деления получается два числа – частное и остаток, то эти числа помещаются в определённые регистры 3.2.

Таблица 3.2. Регистры используемые командами деления в Nasm

Размер операнда (делителя)	Делимое	Частное	Остаток
1 байт	AX	AL	AH
2 байт	DX:AX	AX	DX
4 байт	EDX:EAX	EAX	EDX

Например, после выполнения инструкций

```
mov ax,31
```

```
mov dl,15
```

```
div dl
```

результат 2 (31/15) будет записан в регистр al, а остаток 1 (остаток от деления 31/15) — в регистр ah.

Если делитель — это слово (16-бит), то делимое должно записываться в регистрах dx:ax. Так в результате выполнения инструкций

```
mov ax,2 ; загрузить в регистровую
```

```
mov dx,1 ; пару `dx:ax` значение 10002h
```

```
mov bx,10h
```

```
div bx
```

в регистр ax запишется частное 1000h (результат деления 10002h на 10h), а в регистр dx — 2 (остаток от деления).

3.3 Перевод символа числа в десятичную символьную 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 (см. Приложение.), 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

Порядок выполнения работы:

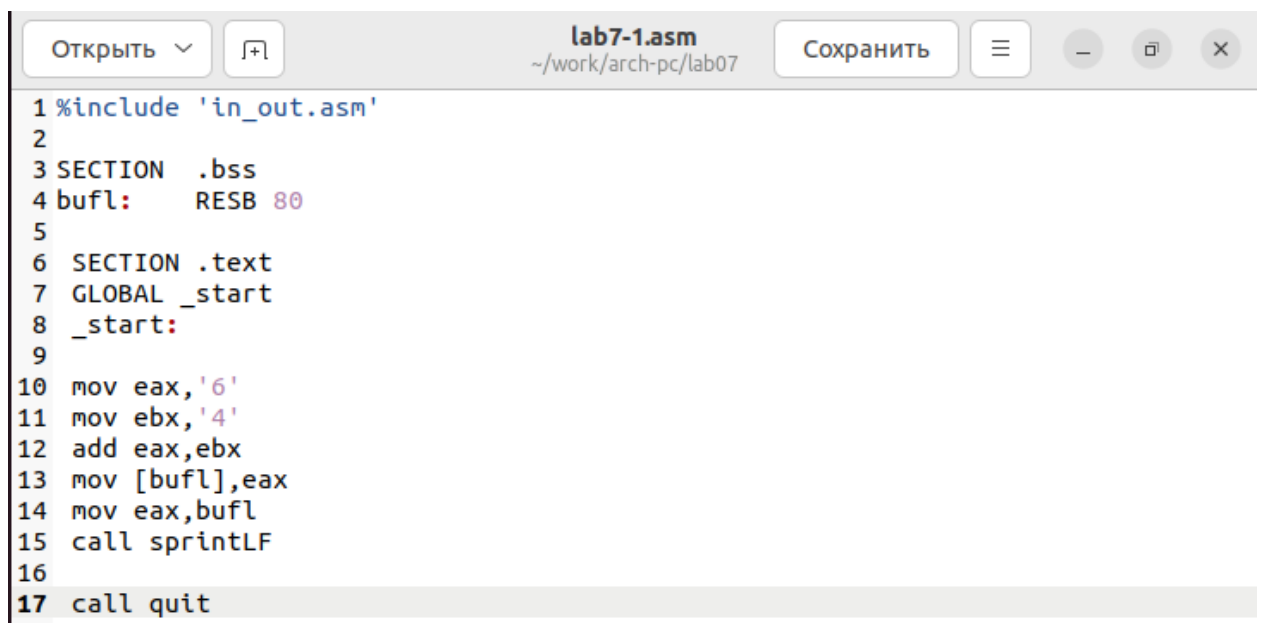
1. Создаю каталог для программ лабораторной работы № 7, перехожу в него и создаю файл lab7-1.asm (рис. 4.1).

```
dvzambalova@dvzambalova-VirtualBox:~$ mkdir ~/work/arch-pc/lab07
dvzambalova@dvzambalova-VirtualBox:~$ cd ~/work/arch-pc/lab07
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ touch lab7-1.asm
```

Рис. 4.1: Создание файла

2. Рассматриваю примеры программ вывода символьных и численных значений. Программы будут выводить значения, записанные в регистр eax.

Ввожу в файл lab7-1.asm текст программы из листинга 7.1 (рис. 4.2). 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.



```
lab7-1.asm
~/work/arch-pc/lab07

1 %include 'in_out.asm'
2
3 SECTION .bss
4 buf1:   RESB 80
5
6 SECTION .text
7 GLOBAL _start
8 _start:
9
10 mov eax,'6'
11 mov ebx,'4'
12 add eax,ebx
13 mov [buf1],eax
14 mov eax,buf1
15 call sprintf
16
17 call quit
```

Рис. 4.2: Текст программы из листинга 7.1

Создаю исполняемый файл и запускаю его (рис. 4.3). Перед созданием исполняемого файла создаю копию файла `in_out.asm` в каталоге `~/work/arch-pc/lab07`.

```
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ ./lab7-1
j
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$
```

Рис. 4.3: Создание и запуск исполняемого файла

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 (или 54 в десятичном представлении), а код символа 4 – 00110100 (52). Команда `add eax,ebx` запишет в регистр `eax` сумму кодов – 01101010 (106), что в свою очередь является кодом символа `j`.

3. Далее изменяю текст программы и вместо символов, записываю в регистры числа. Исправляю текст программы (Листинг 1) следующим образом: заменяю строки (рис. 4.4).

```
10 mov eax,6
11 mov ebx,4
```

Рис. 4.4: Замена строк

Создаю исполняемый файл и запускаю его (рис. 4.5).

```
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf lab7-1.asm
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-1 lab7-1.o
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ ./lab7-1

dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$
```

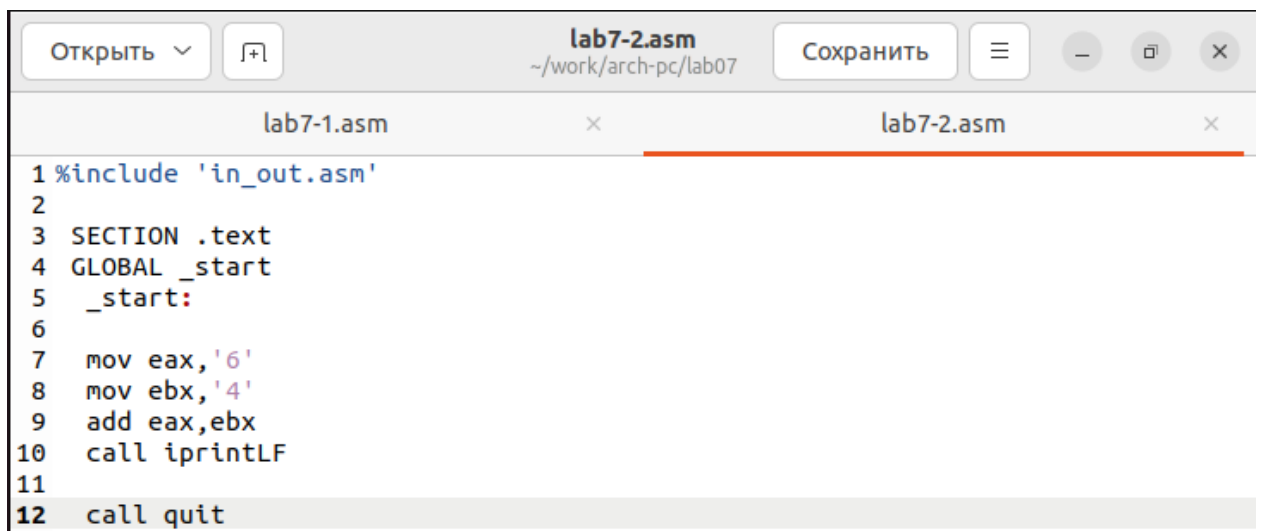
Рис. 4.5: Создание и запуск исполняемого файла

Как и в предыдущем случае при исполнении программы мы не получим число 10. В данном случае выводится символ с кодом 10. Пользуясь таблицей ASCII определяю, что символу STX соответствует код 10. Не отображается этот символ при выводе на экран.

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 7.1 с использованием этих функций. Создаю файл `lab7-2.asm` в каталоге `~/work/arch-pc/lab07` (рис. 4.6) и ввожу в него текст программы из листинга 7.2 (рис. 4.7).

```
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ touch lab7-2.asm
```

Рис. 4.6: Создание файла



```
lab7-2.asm
~/work/arch-pc/lab07
Открыть  [icon] Сохранить [icon] [icon] [icon] [icon]
lab7-1.asm x lab7-2.asm x
1 %include 'in_out.asm'
2
3 SECTION .text
4 GLOBAL _start
5 _start:
6
7 mov eax, '6'
8 mov ebx, '4'
9 add eax, ebx
10 call iprintLF
11
12 call quit
```

Рис. 4.7: Текст программы из листинга 7.2

Создаю исполняемый файл и запускаю его (рис. 4.8).

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

Рис. 4.8: Создание и запуск исполняемого файла

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 программы из листинга 7.1, функция `iprintLF` позволяет вывести число, а не символ, кодом которого является это число.

5. Аналогично предыдущему примеру изменяю символы на числа (рис. 4.9).

```
7  mov eax,6
8  mov ebx,4
```

Рис. 4.9: Замена строк

Создаю исполняемый файл и запускаю его (рис. 4.10).

```
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ ./lab7-2
10
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$
```

Рис. 4.10: Создание и запуск исполняемого файла

Результат при исполнении программы: 10.

Заменяю функцию `iprintLF` на `iprint`. Создаю исполняемый файл и запускаю его (рис. 4.11).

```
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf lab7-2.asm
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o lab7-2 lab7-2.o
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ ./lab7-2
10dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$
```

Рис. 4.11: Создание и запуск исполняемого файла

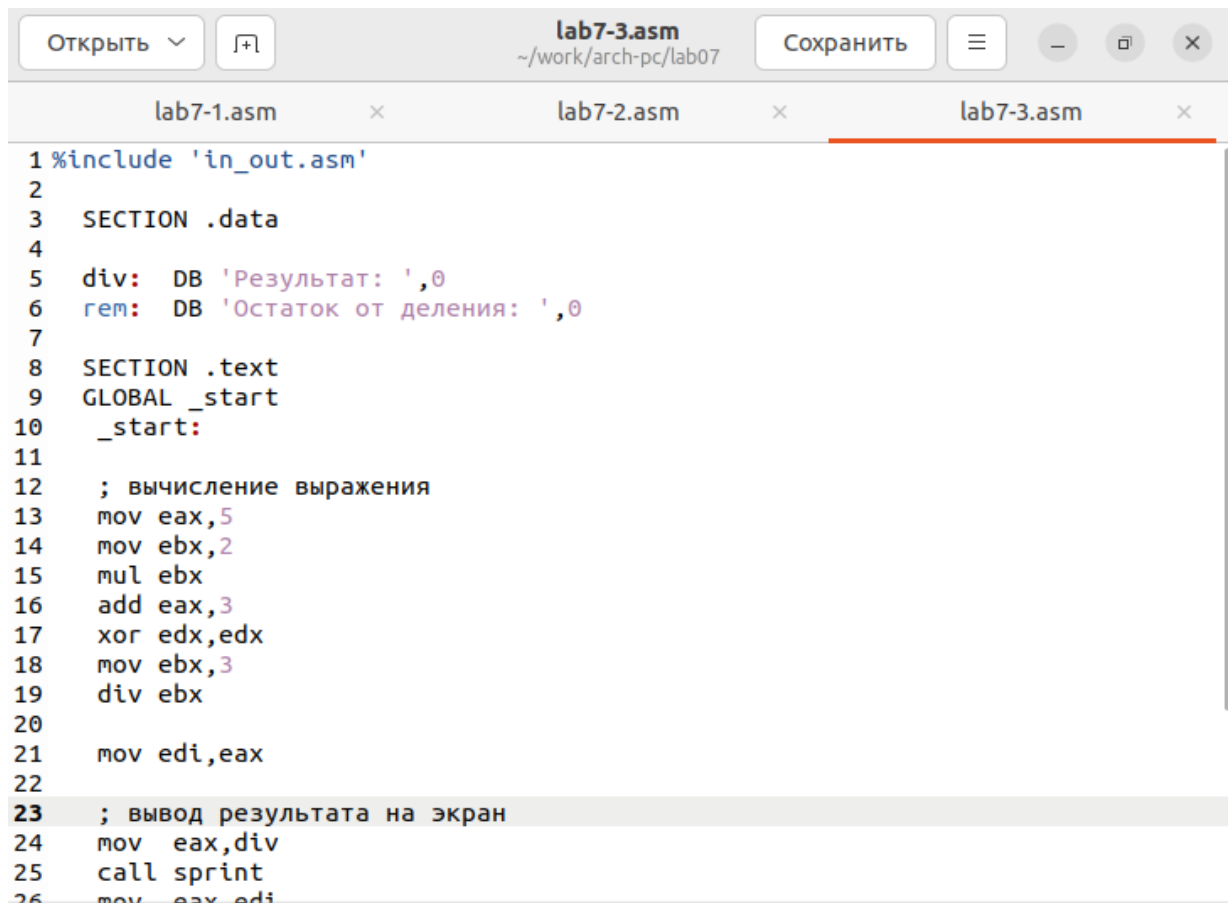
При `iprint` результат и имя пользователя находятся на одной строке, а при `iprintLF` нет.

6. В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения $f(x) = (5 * 2 + 3)/3$. Создаю файл `lab7-3.asm` в каталоге `~/work/arch-pc/lab07` (рис. 4.12).

```
10dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ touch lab7-3.asm
```

Рис. 4.12: Создание файла

Внимательно изучаю текст программы из листинга 7.3 и ввожу в `lab7-3.asm` (рис. 4.13, 4.14).



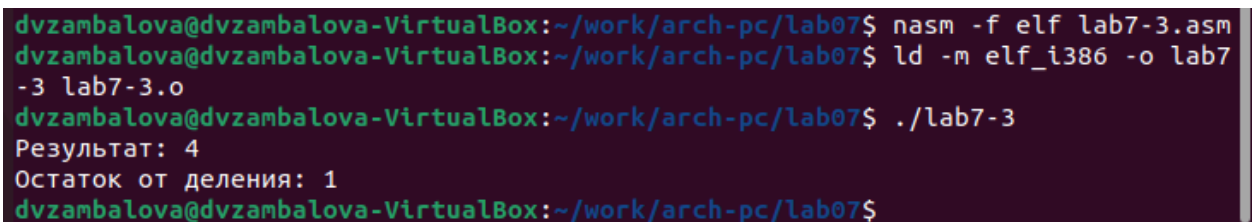
```
1 %include 'in_out.asm'
2
3 SECTION .data
4
5 div: DB 'Результат: ',0
6 rem: DB 'Остаток от деления: ',0
7
8 SECTION .text
9 GLOBAL _start
10 _start:
11
12 ; вычисление выражения
13 mov eax,5
14 mov ebx,2
15 mul ebx
16 add eax,3
17 xor edx,edx
18 mov ebx,3
19 div ebx
20
21 mov edi,eax
22
23 ; вывод результата на экран
24 mov eax,div
25 call sprint
26 mov eax,edi
```

Рис. 4.13 Текст программы из листинга 7.3

```
25 call sprint
26 mov eax,edi
27 call iprintLF
28
29 mov eax,rem
30 call sprint
31 mov eax,edx
32 call iprintLF
33
34 call quit
```

Рис. 4.14: Текст программы из листинга 7.3

Создаю исполняемый файл и запускаю его (рис. 4.15).



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

Рис. 4.15: Создание и запуск исполняемого файла

Изменяю текст программы для вычисления выражения $f(x) = (4 * 6 + 2)/5$ (рис. 4.16).

```
12 ; вычисление выражения
13 mov eax,4
14 mov ebx,6
15 mul ebx
16 add eax,2
17 xor edx,edx
18 mov ebx,5
19 div ebx
```

Рис. 4.16: Замена строк

Создаю исполняемый файл и запускаю его (рис. 4.17).

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

Рис. 4.17: Создание и запуск исполняемого файла

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: $(Sn \bmod 20) + 1$, где Sn – номер студенческого билета (В данном случае $a \bmod b$ – это остаток от деления a на b).
- вывести на экран номер варианта.

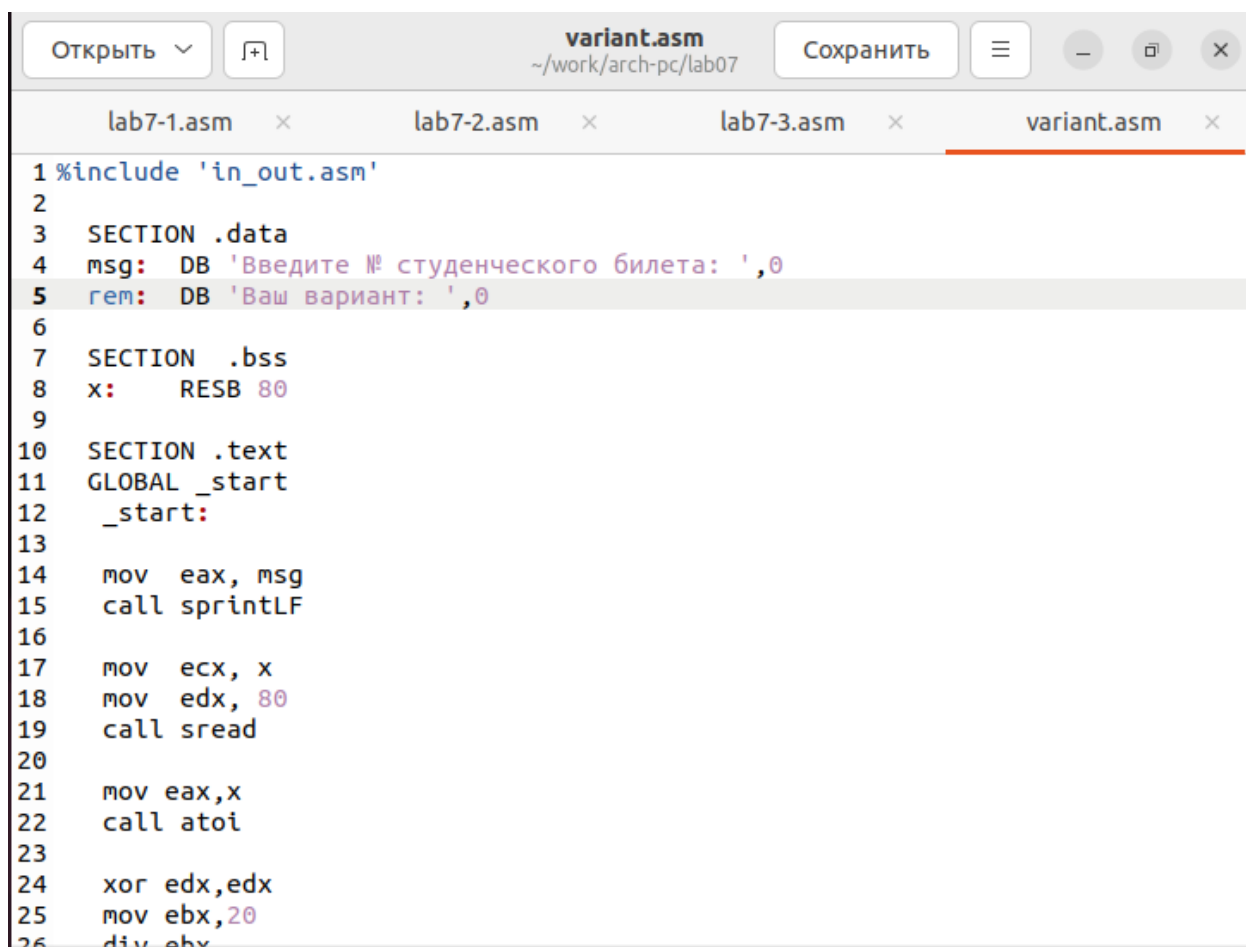
В данном случае число, над которым необходимо проводить арифметические операции, вводится с клавиатуры. Как отмечалось выше ввод с клавиатуры осуществляется в символьном виде и для корректной работы арифметических операций в NASM символы необходимо преобразовать в числа. Для этого может быть использована функция `atoi` из файла `in_out.asm`.

Создаю файл `variant.asm` в каталоге `~/work/arch-pc/lab07` (рис. 4.18).

```
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ touch variant.asm
```

Рис. 4.18: Создание файла

Внимательно изучаю текст программы из листинга 7.4 и ввожу в файл `variant.asm` (рис. 4.19, 4.20).



```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg: DB 'Введите № студенческого билета: ',0
5 rem: DB 'Ваш вариант: ',0
6
7 SECTION .bss
8 x: RESB 80
9
10 SECTION .text
11 GLOBAL _start
12 _start:
13
14 mov eax, msg
15 call sprintf
16
17 mov ecx, x
18 mov edx, 80
19 call sread
20
21 mov eax, x
22 call atoi
23
24 xor edx, edx
25 mov ebx, 20
26 div ebx
```

Рис. 4.19: Текст программы из листинга 7.4



```
25 mov ebx, 20
26 div ebx
27 inc edx
28
29 mov eax, rem
30 call sprint
31 mov eax, edx
32 call iprintLF
33
34 call quit
```

Рис. 4.20: Текст программы из листинга 7.4

Создаю исполняемый файл и запускаю его (рис. 4.21).

```

dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf variant.asm
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o variant variant.o
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ ./variant
Введите № студенческого билета:
1132226536
Ваш вариант: 17
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$

```

Рис. 4.21: Создание и запуск исполняемого файла

Проверяю результат работы программы, вычислив номер варианта аналитически.
Мой вариант – 17. $18(x + 1)/6$.

Включаю в отчет по выполнению лабораторной работы ответы на вопросы:

1. `mov eax,rem`

`call sprint`

`mov eax,edx`

`call iprintLF`

2. `nasm` превращает текст программы в объектный код,

`mov ecx, x` - в этом случае в регистр `ecx` запишется адрес `x`,

`mov edx, 80` - длина вводимой строки,

`call sread` - вызов подпрограммы ввода сообщения.

3. ASCII кода в число

4. `xor edx,edx`

`mov ebx,20`

`div ebx`

`inc edx`

5. `edx`

6. инкремент `edx`

7. `mov eax,rem`

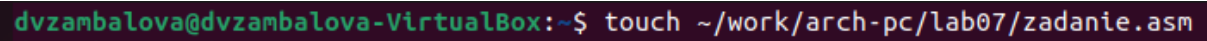
`call sprint`

```
mov eax,edx
```

```
call iprintLF
```

Задание для самостоятельной работы:

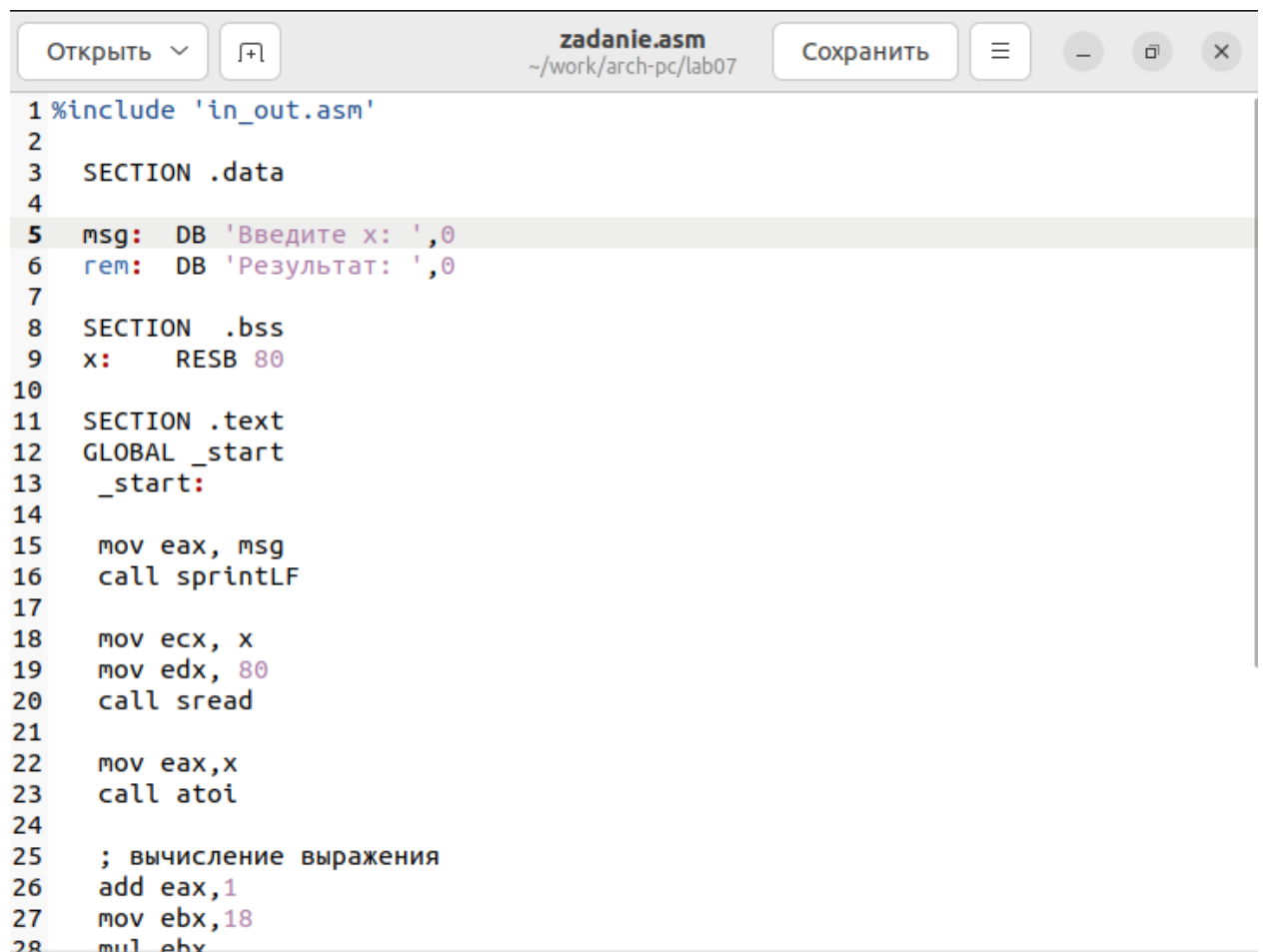
1. Создаю файл `zadanie.asm` в каталоге `~/work/arch-pc/lab07` (рис. 4.22).



```
dvzambalova@dvzambalova-VirtualBox:~$ touch ~/work/arch-pc/lab07/zadanie.asm
```

Рис. 4.22: Создание файла

Ввожу текст программы в файл `zadanie.asm` (рис. 4.23, 4.24).



```
1 %include 'in_out.asm'
2
3 SECTION .data
4
5 msg: DB 'Введите x: ',0
6 rem: DB 'Результат: ',0
7
8 SECTION .bss
9 x: RESB 80
10
11 SECTION .text
12 GLOBAL _start
13 _start:
14
15 mov eax, msg
16 call sprintLF
17
18 mov ecx, x
19 mov edx, 80
20 call sread
21
22 mov eax, x
23 call atoi
24
25 ; вычисление выражения
26 add eax, 1
27 mov ebx, 18
28 mul ebx
```

Рис. 4.23: Текст программы

```

26  add eax,1
27  mov ebx,18
28  mul ebx
29  xor edx,edx
30  mov ecx,6
31  div ecx
32
33  mov edi,eax
34
35  ; вывод результата на экран
36  mov eax,rem
37  call sprint
38  mov eax,edi
39  call iprintLF
40
41  call quit

```

Рис. 4.24: Текст программы

Создаю исполняемый файл и проверьте его работу для значений $x_1 = 3$ и $x_2 = 1$ (рис. 4.25).

```

dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ nasm -f elf zadanie.asm
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ ld -m elf_i386 -o zadanie.o zadanie.o
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ ./zadanie
Введите x:
3
Результат: 12
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$ ./zadanie
Введите x:
1
Результат: 6
dvzambalova@dvzambalova-VirtualBox:~/work/arch-pc/lab07$

```

Рис. 4.25: Создание и запуск исполняемого файла

5 Выводы

Я освоила арифметические инструкции языка ассемблера NASM.

Список литературы

1. Расширенный ассемблер: NASM. URL: https://www.opennet.ru/docs/RUS/nasm/nasm_ru3.html. Дата обращения (26.11.2022).