

Отчёт по лабораторной работе №6

Дисциплина: Архитектура Компьютера

Егор Витальевич Кузьмин

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
5	Выводы	17
	Список литературы	18

Список иллюстраций

4.1	Работа с директориями и создание файла	8
4.2	Создание копии файла и проверка	8
4.3	Редактирование файла	9
4.4	Подготовка и исполнение файла	9
4.5	Редактирование файла	9
4.6	Трансляция, компоновка, запуск исполняемого файла	10
4.7	Создание, редактирование файла	10
4.8	Компиляция, обработка и запуск исполняемого файла	10
4.9	Редактирование, компиляция, обработка и запуск исполняемого файла	11
4.10	Редактирование файла	11
4.11	Компиляция, обработка и запуск исполняемого файла	12
4.12	Создание и редактирование файла	12
4.13	Компиляция, обработка и запуск исполняемого файла	12
4.14	Редактирование, компиляция, обработка и запуск исполняемого файла	13
4.15	Работа с файлом variant.asm	14
4.16	Создание и редактирование файла	15
4.17	Компиляция, обработка и запуск исполняемого файла	15

1 Цель работы

Целью данной работы является освоение арифметических инструкций языка ассемблера NASM и приобретение навыков работы с ними.

2 Задание

- 0. Общее ознакомление с арифметическими инструкциями в NASM.
- 1. Символьные и численные данные в NASM.
- 2. Выполнение арифметических операций в NASM.
- 3. Выполнение заданий для самостоятельной работы

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 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.

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 является универсальной, а вторая 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.

4 Выполнение лабораторной работы

4.1) Символьные и численные данные в NASM.

С помощью утилиты `mkdir` создаю директорию `lab06` для выполнения соответствующей лабораторной работы. Перехожу в созданный каталог с помощью утилиты `cd`. С помощью `touch` создаю файл `lab6-1.asm`. (рис. 4.1).

```
[evkuzmin@fedora ~]$ mkdir ~/work/arch-pc/lab06
[evkuzmin@fedora ~]$ cd ~/work/arch-pc/lab06
[evkuzmin@fedora lab06]$ touch lab6-1.asm
[evkuzmin@fedora lab06]$
```

Рис. 4.1: Работа с директориями и создание файла

Копирую в текущий каталог файл `in_out.asm` с помощью утилиты `cp`, ибо он будет использоваться в дальнейшем. (рис. 4.2).

```
[evkuzmin@fedora lab06]$ cp ~/Загрузки/in_out.asm in_out.asm
[evkuzmin@fedora lab06]$ ls
in_out.asm  lab6-1.asm
```

Рис. 4.2: Создание копии файла и проверка

Открываю созданный файл `lab6-1.asm`, вставляю в него следующую программу: (рис. 4.3).

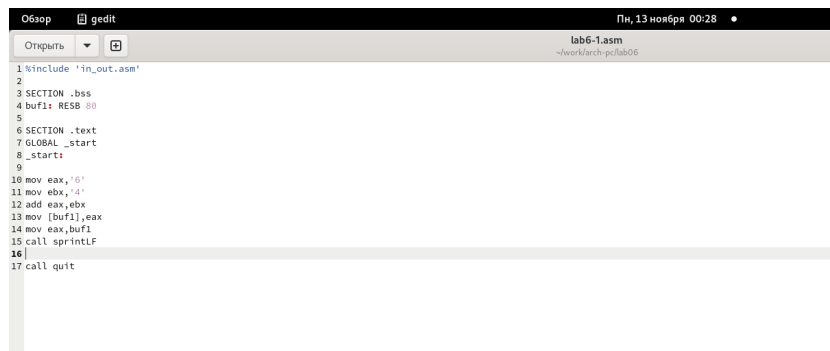


Рис. 4.3: Редактирование файла

Выполняю компиляцию, компоновку файла и запускаю его. Программа выводит символ j, потому что он соответствует сумме двоичных кодов символов 4 и 6 по системе ASCII. (рис. 4.4).

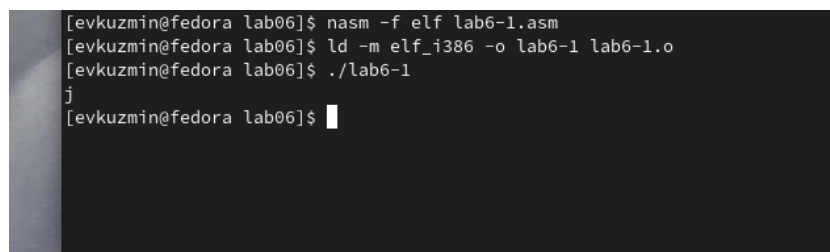


Рис. 4.4: Подготовка и исполнение файла

Изменяю в тексте программы символы “6” и “4” на цифры 6 и 4 (рис. 4.5).

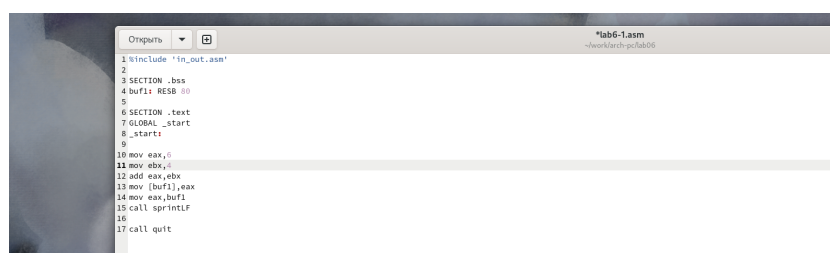


Рис. 4.5: Редактирование файла

Создаю новый исполняемый файл программы и запускаю его. На экран вывелся символ с кодом 10 и он не отображается, т.к. это символ перевода строки. (рис.

4.6).

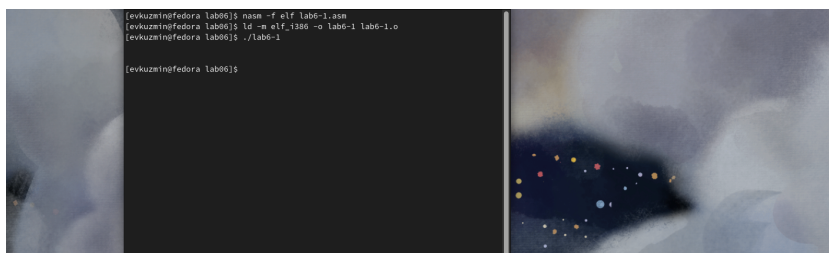


Рис. 4.6: Трансляция, компоновка, запуск исполняемого файла

Создаю новый файл lab6-2.asm с помощью утилиты touch. Открываю файл lab6-2.asm для редактирования, ввожу следующую программу. (рис. 4.7).

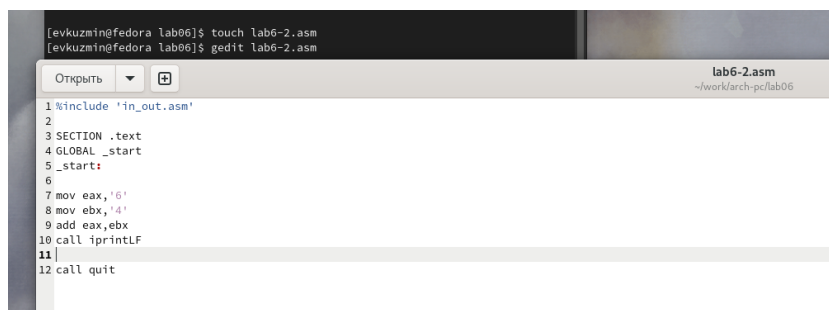


Рис. 4.7: Создание, редактирование файла

Выполняю компиляцию и компоновку, и запускаю исполняемый файл lab6-2. Теперь на экран выводится число 106, т.к. происходит сложение кодов символов “6” и “4”, но при этом программа позволяет вывести именно число, а не символ, кодом которого является это число (рис. 4.8).

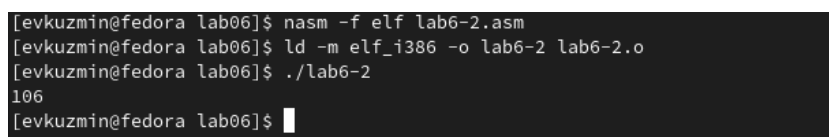


Рис. 4.8: Компиляция, обработка и запуск исполняемого файла

Следом также заменяю в тексте программы в файле lab6-2.asm символы “6” и “4” на числа 6 и 4. Затем провожу компиляцию и компоновку, после этого

запускаю исполняемый файл. Теперь программа складывает не соответствующие символам коды в системе ASCII, а сами числа, поэтому на экране мы видим число 10. (рис. 4.9).



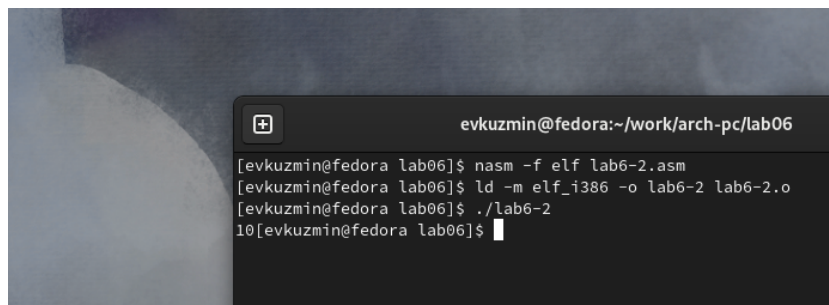
Рис. 4.9: Редактирование, компиляция, обработка и запуск исполняемого файла

Заменяю в тексте программы функцию iprintLF на iprint. (рис. 4.10).



Рис. 4.10: Редактирование файла

Проводим с файлом привычные операции. Вывод изменился (см. рисунок), потому что функция iprint не добавляет к выводу символ переноса строки в отличие от функции iprintLF. (рис. 4.11).

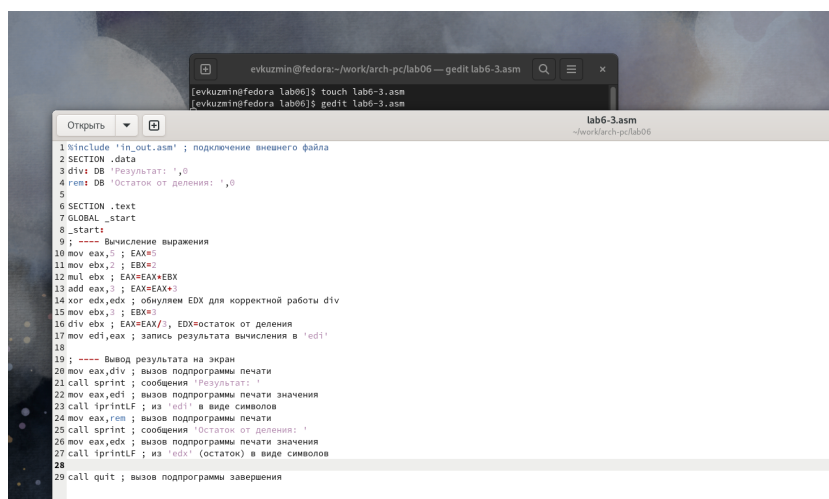


```
evkuzmin@fedora:~/work/arch-pc/lab06
[evkuzmin@fedora lab06]$ nasm -f elf lab6-2.asm
[evkuzmin@fedora lab06]$ ld -m elf_i386 -o lab6-2 lab6-2.o
[evkuzmin@fedora lab06]$ ./lab6-2
10[evkuzmin@fedora lab06]$
```

Рис. 4.11: Компиляция, обработка и запуск исполняемого файла

4.2) Выполнение арифметических операций в NASM.

Создаю файл lab6-3.asm с помощью утилиты touch. Ввожу в созданный файл текст программы для вычисления значения выражения: $f(x) = (5*2+3)/3$ (рис. 4.12).



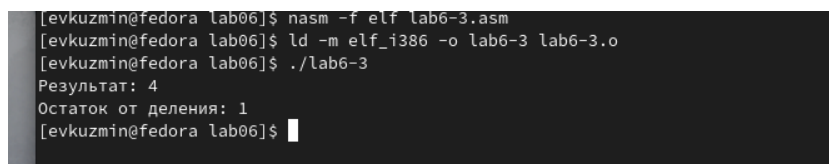
```
evkuzmin@fedora:~/work/arch-pc/lab06 --- gedit lab6-3.asm
[evkuzmin@fedora lab06]$ touch lab6-3.asm
[evkuzmin@fedora lab06]$ gedit lab6-3.asm

lab6-3.asm
~/work/arch-pc/lab06

1 %include 'in_out.asm' ; подключение внешнего файла
2 SECTION .data
3 drive DB 'Результат: ',0
4 rem: DB 'Остаток от деления: ',0
5
6 SECTION .text
7 GLOBAL _start
8 _start:
9 ; ---- Вычисление выражения
10 mov eax,2 ; EAX=2
11 mov ebx,2 ; EBX=2
12 mul ebx ; EAX=EAX*EBX
13 add eax,3 ; EAX=EAX+3
14 xor edx,edx ; обнуляем EDI для корректной работы div
15 mov ebx,3 ; EBX=3
16 div ebx ; EAX=EAX/3, EDI=остаток от деления
17 mov edi,eax ; запись результата вычисления в 'edi'
18
19 ; ---- Вывод результата на экран
20 mov eax,div ; вызов подпрограммы печати
21 call sprint ; сообщения 'Результат: '
22 mov eax,edi ; вызов подпрограммы печати значения
23 call iprintf ; из 'edi' в виде символов
24 mov eax,rem ; вызов подпрограммы печати
25 call sprint ; сообщения 'Остаток от деления: '
26 mov eax,edx ; вызов подпрограммы печати значения
27 call iprintf ; из 'edx' (остаток) в виде символов
28
29 call quit ; вызов подпрограммы завершения
```

Рис. 4.12: Создание и редактирование файла

Реализовываю исполняемый файл и запускаю его (рис. 4.13).



```
[evkuzmin@fedora lab06]$ nasm -f elf lab6-3.asm
[evkuzmin@fedora lab06]$ ld -m elf_i386 -o lab6-3 lab6-3.o
[evkuzmin@fedora lab06]$ ./lab6-3
Результат: 4
Остаток от деления: 1
[evkuzmin@fedora lab06]$
```

Рис. 4.13: Компиляция, обработка и запуск исполняемого файла

Изменяю программу так, чтобы она вычисляла значение выражения $f(x) = (4*6+2)/5$. Выполняю компиляцию и компоновку файла, далее запускаю новый исполняемый файл. Если проведем устную проверку, то убедимся в правильности работы программы. (рис. 4.14).

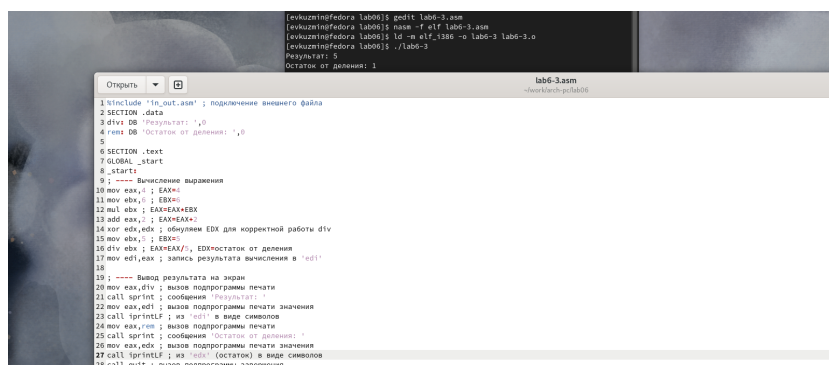


Рис. 4.14: Редактирование, компиляция, обработка и запуск исполняемого файла

Создаю файл variant.asm с помощью утилиты touch. Ввожу в файл текст программы для вычисления варианта задания по номеру студенческого билета. Провожу компиляцию и компоновку, запускаю исполняемый файл. Ввожу номер своего студенческого билета. Программа выводит, что мой вариант - 7 (рис. 4.15).

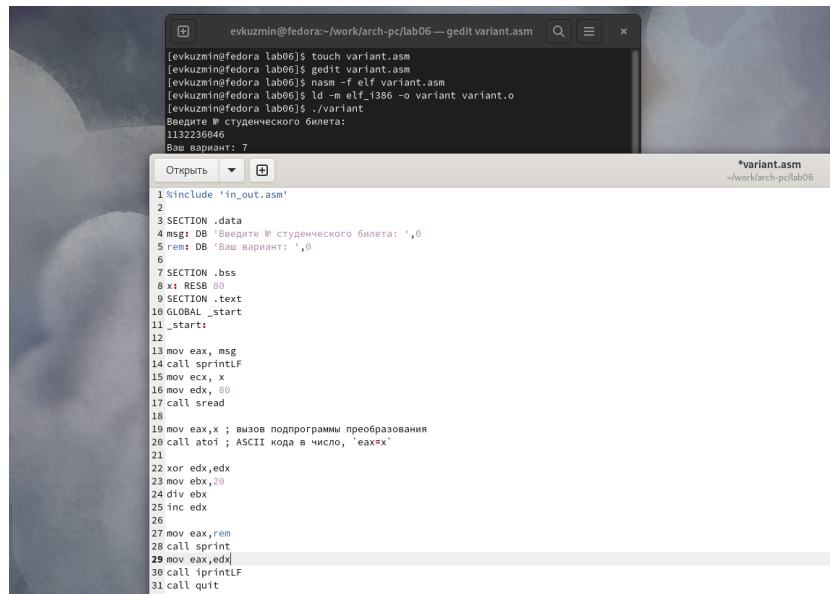


Рис. 4.15: Работа с файлом variant.asm

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` `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) Выполнение заданий для самостоятельной работы

Создаю файл `sr.asm` с помощью утилиты `touch`. Открываю созданный файл для редактирования, ввожу в него текст программы для вычисления значения выражения своего, 7-го варианта, $5*(x-1)^2$. (рис. 4.16)

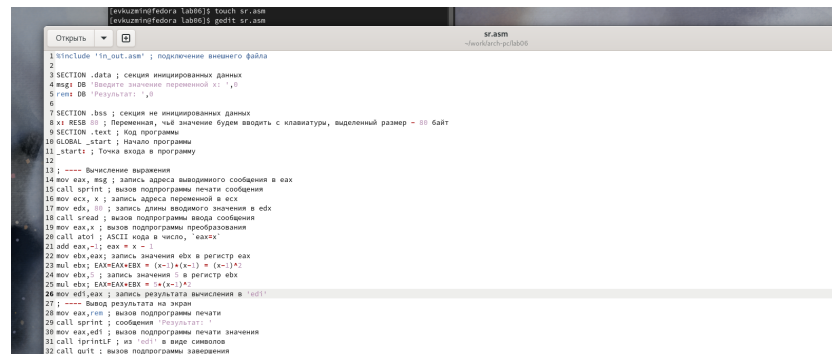


Рис. 4.16: Создание и редактирование файла

Проводим привычные операции и запускаем исполняемый файл, выполняем устную проверку и убеждаемся в правильности работы программы.(рис. 4.17)

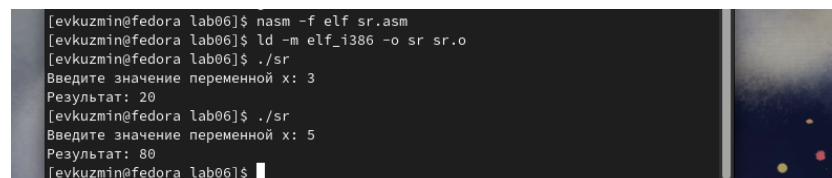


Рис. 4.17: Компиляция, обработка и запуск исполняемого файла

Листинг 4.1 - Программа для вычисления значения выражения из варианта 7.

```
%include 'in_out.asm' ; подключение внешнего файла
SECTION .data ;
секция иницированных данных msg: DB 'Введите значение переменной x:
',0 rem: DB 'Результат: ',0 SECTION .bss ; секция неиницированных
данных x: RESB 80 ; Переменная, значение которой мы будем вводить с
клавиатуры, выделенный размер - 80 байт SECTION .text ; Код программы
```

```

GLOBAL _start ; Начало программы _start: ; Точка входа в программу
; ---- Вычисление выражения mov eax, msg ; запись адреса выводимого
сообщения в eax call sprint ; вызов подпрограммы печати сообщения
mov ecx, x ; запись адреса переменной в ecx mov edx, 80 ; запись
длины вводимого значения в edx call sread ; вызов подпрограммы ввода
сообщения mov eax,x ; вызов подпрограммы преобразования call atoi
; ASCII кода в число, `eax=x` add eax,-1;  $eax = x - 1$  mov ebx,eax;
запись значения ebx в регистр eax mul ebx;  $EAX=EAX*EBX = (x-1)*(x-1) = (x-1)^2$ 
mov ebx,5 ; запись значения 5 в регистр ebx mul ebx;
 $EAX=EAX*EBX = 5*(x-1)^2$  mov edi,eax ; запись результата вычисления в
'edi' ; ---- Вывод результата на экран mov eax,tem ; вызов подпрограммы
печати call sprint ; сообщения 'Результат: ' mov eax,edi ; вызов
подпрограммы печати значения call iprintLF ; из 'edi' в виде символов
call quit ; вызов подпрограммы завершения

```


5 Выводы

При выполнении лабораторной работы я освоил арифметические инструкции языка ассемблера NASM и приобрел практический опыт работы с ними.

Список литературы

Архитектура компьютера и ЭВМ