РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6

дисциплина: Архитектура компьютера

Студент: Володина Алиса Алексеевна

Группа: НКАбд-01-25

МОСКВА

2025 г.

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Приобретение практических навыков работы в Midnight Commander. Освоение инструкций языка ассемблера mov и int.

2 Задание

Символьные и численные данные в NASM

Выполнение арифметических операций в NASM

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. Регистровая адресация–операнды хранятся в регистрах и в команде используются имена этих регистров, например: movax, bx. Непосредственная адресация–значениеоперандазадаетсянепосредственновкоманде,Например:movax,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 Codefor Information Interchange (Американский стандартный код для обмена информацией). Согласно стандарту ASCII каждый символ кодируется одним байтом. 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

4 Выполнение лабораторной работы

Символьные и численные данные в NASM

1. Создадим каталог для программам лабораторной работы № 6, перейдем в него и создадим файл lab6-1.asm(рисунок 0.1-0.3)

Рисунок 0.1 Создание нового каталога

Рисунок 0.2 Переход в новый каталог

Рисунок 0.3 Создание файла

2. Рассмотрим примеры программ вывода символьных и численных значений. Программы будут выводить значения, записанные в регистр eax.(рисунок 0.4-0.7)

Открыть *lab6-1.asm ~/work/arch-pc/lab06

```
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 sprintLF
16
17        call quit
```

Рисунок 0.4 Запись текста программы

Создадим исполняемый файл и запустим его

```
aavolodina@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-1.asm
aavolodina@fedora:~/work/arch-pc/lab06$
```

Рисунок 0.5 Создание исполняемого файла

```
aavolodina@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-1 lab6-1.o
aavolodina@fedora:~/work/arch-pc/lab06$
```

Рисунок 0.6 Создание исполняемого файла

```
avolodina@fedora:~/work/arch-pc/lab06$ ./lab6-1

avolodina@fedora:~/work/arch-pc/lab06$
```

Рисунок 0.7 Запуск исполняемого файла

Вывод программы отличается от предполагаемого, поскольку по таблице ASCII коды символов в сумме дают j

3. Далее изменим текст программы и вместо символов, запишем в регистры числа. Исправим текст программы следующим образом (рисунок 0.8)

```
%include 'in_out.asm'

SECTION .text
GLOBAL _start
_start:

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

call quit
```

Рисунок 0.8 Запись программы с изменением

Создадим исполняемый файл и запустим его (рисунок 0.9-0.10)

```
aavolodina@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-1.asm
aavolodina@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-1 lab6-1.o
aavolodina@fedora:~/work/arch-pc/lab06$
```

Рисунок 0.9 Создание исполняемого файла

```
aavolodina@fedora:~/work/arch-pc/lab06$ ./lab6-1


aavolodina@fedora:~/work/arch-pc/lab06$
```

Рисунок 0.10 Запуск исполняемого файла

На этот раз программа выдала пустую строку, поскольку символ 10 означает переход на новую строку

4. Создадим файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 и введем в него текст программы из листинга (рисунок 0.11-0.13)

```
aavolodina@fedora:~/work/arch-pc/lab06$ touch lab6-2.asm
aavolodina@fedora:~/work/arch-pc/lab06$
```

Рисунок 0.11 Создание файла

```
aavolodina@fedora:~/work/arch-pc/lab06$ gedit lab6-2.asm
```

Рисунок 0.12 Заход в файл

```
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
```

Рисунок 0.13 Запись программы

Создадим исполняемый файл и запустим его (рисунок 0.14-0.16)

```
aavolodina@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-2.asm
aavolodina@fedora:~/work/arch-pc/lab06$
```

Рисунок 0.14 Создание исполняемого файла

```
aavolodina@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
aavolodina@fedora:~/work/arch-pc/lab06$
```

Рисунок 0.15 Создание исполняемого файла

```
aavolodina@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-2 lab6-2.o
aavolodina@fedora:~/work/arch-pc/lab06$ ./lab6-2
106
```

Рисунок 0.16 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 отличии от программы из листинга, функция siprintLF позволяет вывести число, а не символ, кодом которого является это число

5.Аналогично предыдущему примеру изменим символы на числа. Заменим строки (рисунок 0.17)

```
%include 'in_out.asm'

SECTION .text
GLOBAL _start
_start:

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

call quit
```

Рисунок 0.17 Измененный текст программы

Создадим исполняемый файл и запустим его (рисунок 0.18-0.19)

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

Рисунок 0.18 Создание исполняемого файла

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

Рисунок 0.19 Запуск файла

При исполнении программы мы получим ожидаемый результат – 10

Заменим функцию iprintLF на iprint (рисунок 0.20)

```
%include 'in_out.asm'

SECTION .text
GLOBAL _start
_start:

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

call quit
```

Рисунок 0.20 Измененный текст программы

Создадим исполняемый файл и запустим его (рисунок 0.21-0.22)

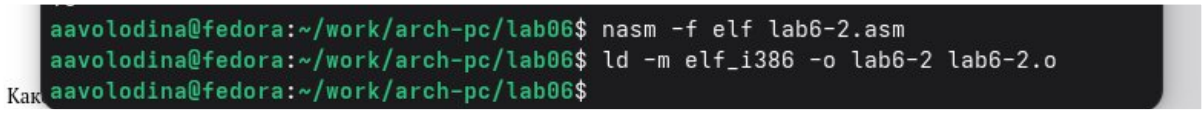


Рисунок 0.21 Создание исполняемого файла

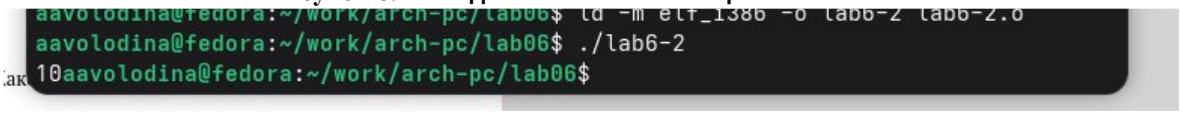


Рисунок 0.22 Запуск файла

При замене функции iprintLF на iprint мы получаем тот же результат, но без переноса строки

Выполнение арифметических операций в NASM

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

Создадим файл lab6-3.asm в каталоге ~/work/arch-pc/lab06 (рисунок 0.23)

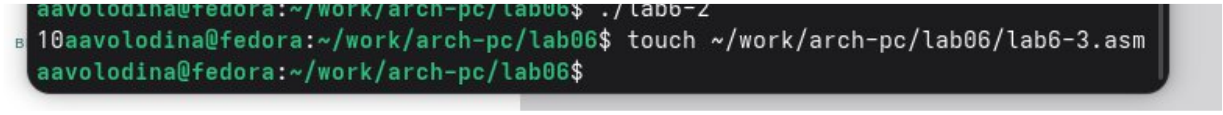
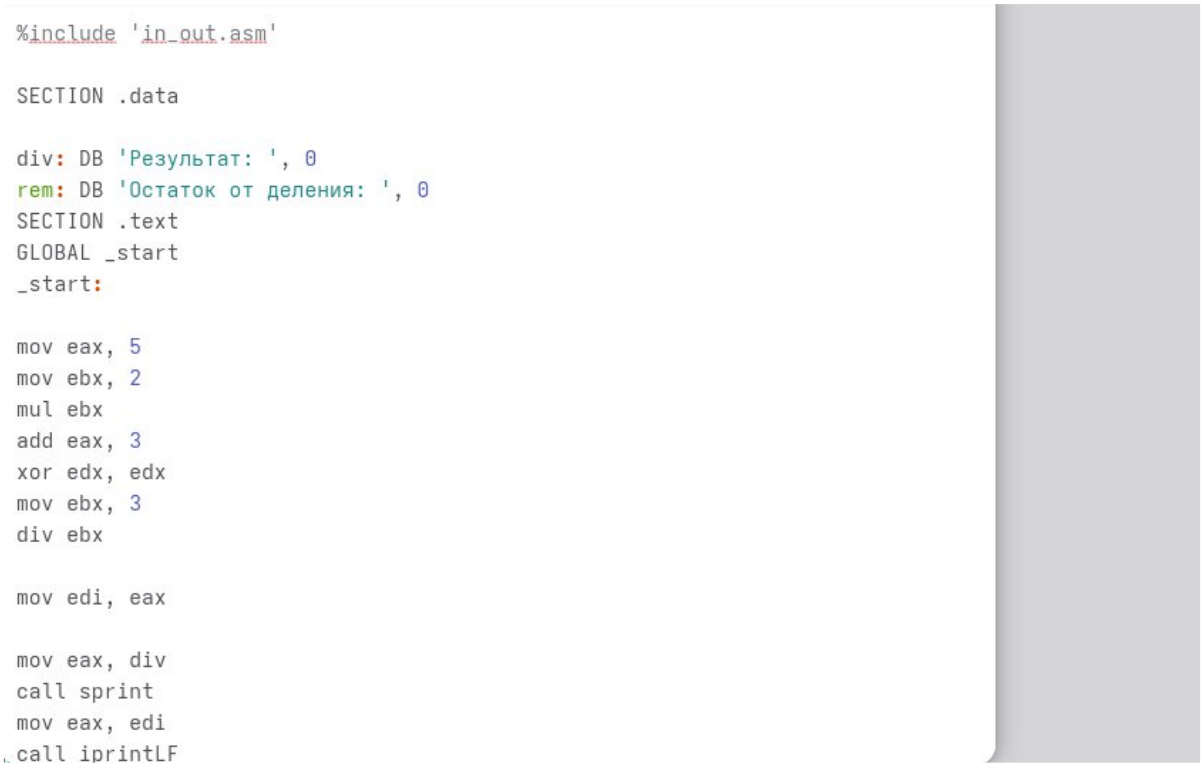


Рисунок 0.23 Создание файла

Создадим исполняемый файл и запустим его (рисунок 0.24-0.27)



```
%include 'in_out.asm'

SECTION .data

div: DB 'Результат: ', 0
rem: DB 'Остаток от деления: ', 0
SECTION .text
GLOBAL _start
_start:

mov eax, 5
mov ebx, 2
mul ebx
add eax, 3
xor edx, edx
mov ebx, 3
div ebx

mov edi, eax

mov eax, div
call sprint
mov eax, edi
call iprintLF
```

Рисунок 0.24 Запись текста программы

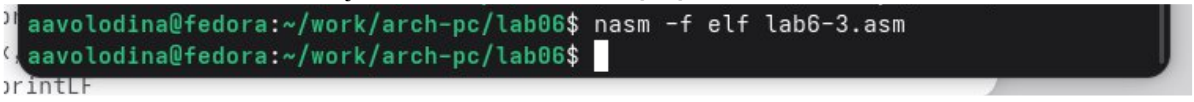


Рисунок 0.25 Создание исполняемого файла

```
aavolodina@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
aavolodina@fedora:~/work/arch-pc/lab06$
```

Рисунок 0.26 Создание исполняемого файла

```
aavolodina@fedora:~/work/arch-pc/lab06$ ./lab6-3
Результат: 4
Остаток от деления: 1
aavolodina@fedora:~/work/arch-pc/lab06$
```

Рисунок 0.27 Запуск исполняемого файла

Изменим текст программы для вычисления выражения $f(x) = (4 * 6 + 2)/5$. (рисунок 0.28)

```
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ', 0
rem: DB 'Остаток от деления: ', 0
SECTION .text
GLOBAL _start
_start:
mov eax, 4
mov ebx, 6
mul ebx
add eax, 2
xor edx, edx
mov ebx, 5
div ebx
mov edi, eax
mov eax, div
call sprint
mov eax, edi
call iprintLF
mov eax, rem
call sprint
mov eax, edx
call iprintLF
```

Рисунок 0.28 Измененный текст программы

Создадим исполняемый файл и проверим его работу. (рисунок 0.29-0.31)

```
aavolodina@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
aavolodina@fedora:~/work/arch-pc/lab06$
```

Рисунок 0.29 Создаем исполняемый файл

```
aavolodina@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
aavolodina@fedora:~/work/arch-pc/lab06$
```

Рисунок 0.30 Создаем исполняемый файл

```
aavolodina@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-3 lab6-3.o
aavolodina@fedora:~/work/arch-pc/lab06$ ./lab6-3
Результат: 5
Остаток от деления: 1
```

Рисунок 0.31 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* mod 20)+1, где *Sn*–номер студенческого билета (В данном случае *a* mod *b*–это остаток отделения *a* на *b*).
- вывести на экран номер варианта.

Создадим файл variant.asm в каталоге ~/work/arch-pc/lab06 (рисунок 0.32-0.33)

```
aavolodina@fedora:~/work/arch-pc/lab06$ touch ~/work/arch-pc/lab06/variant.asm
aavolodina@fedora:~/work/arch-pc/lab06$
```

Рисунок 0.32 Создание файла

```
%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 sprintLF

mov ecx, x
mov edx, 80
call sread

mov eax, x
call atoi

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

mov eax, rem
call sprint
mov eax, edx
call iprintLF

call quit
```

Рисунок 0.33 Запись текста программы

Создадим исполняемый файл и запустим его (рисунок 0.34-0.37)

```
aavolodina@fedora:~/work/arch-pc/lab06$ nasm -f elf variant.asm
aavolodina@fedora:~/work/arch-pc/lab06$
```

Рисунок 0.34 Создание исполняемого файла

```
aavolodina@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o variant variant.o
aavolodina@fedora:~/work/arch-pc/lab06$
```

Рисунок 0.35 Создание исполняемого файла

Рисунок 0.36 Запуск исполняемого файла

Рисунок 0.37 Проверка программы

Ответы на вопросы:

1) За вывод на экран сообщения «Ваш вариант:» отвечают строки:

mov eax,rem

call sprint

2) Данные инструкции используются для:

- mov ecx, x – помещает в ecx адрес буфера x для ввода
- mov edx, 80 — задаёт максимальную длину вводимой строки.
- call sread — вызов функции чтения строки из стандартного ввода.

3) Инструкция «call atoi» используется для:

Вывоза подпрограммы из внешнего файла, которая преобразует ascii-код символа в целое число и записывает результат в регистр eax

4) За вычисление варианта отвечают следующие строки:

xor edx,

edx mov ebx,20

div ebx

inc edx

5) Остаток от деления при выполнении инструкции «div ebx» записываетя в регистр edx

6) Инструкция «» используется для увеличения значения регистра edx на 1

7) За вывод на экран результата вычислений отвечают следующие строки:

mov eax,edx

call iprintLF

Задания для самостоятельной работы

1 В соответствии с выбранным вариантом по номеру студенческого билета (вариант номер 2), реализуем программу для подсчета функции f(x)=(12*x + 3)*5 (рисунок 0.38-0.40)

Рисунок 0.38 Создание файла

```
%include 'in_out.asm'

SECTION .data
msg: DB 'Введите значение переменной x: ',0
rem: DB 'Результат: ',0

SECTION .bss
x: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax, msg
call sprint

mov ecx, x
mov edx, 80
call sread

mov eax, x
call atoi

mov ebx, 31
mul ebx
sub eax, 5
add eax, 10
mov edi, eax
mov eax, rem
call sprint
mov eax, edi
call iprint
call quit
```

Рисунок 0.39 Запись текста программы

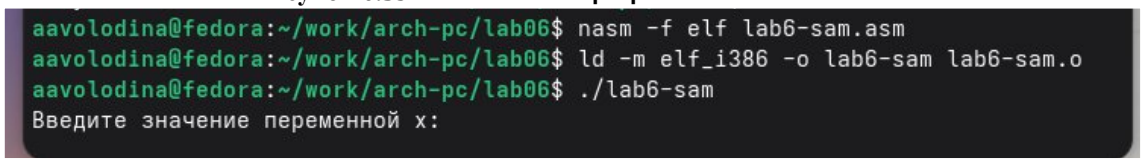


Рисунок 0.40 Создание исполняемого файла

Проверим работу программы используя переменные x1=1, x2=6

Для x1=1 (рисунок 0.41-0.42)

```
aavolodina@fedora:~/work/arch-pc/lab06$ nasm -f elf lab6-sam.asm
aavolodina@fedora:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-sam lab6-sam.o
aavolodina@fedora:~/work/arch-pc/lab06$ ./lab6-sam
Введите значение переменной x: 1
Результат: 75
```

Рисунок 0.41 Проверка работы программы с x1

Для x2=6:

```
aavolodina@fedora:~/work/arch-pc/lab06$ ./lab6-sam
Введите значение переменной x: 6
Результат: 375
aavolodina@fedora:~/work/arch-pc/lab06$
```

Рисунок 0.42 Проверка работы программы с x2

5 Вывод

При выполнении данной лабораторной работы я освоила арифметические инструкции языка ассемблера NASM

6 Список литературы

1. GDB:TheGNUProjectDebugger.—URL:https://www.gnu.org/software/gdb/.

2. GNUBashManual.—2016.—URL:https://www.gnu.org/software/bash/manual/.

3. Midnight CommanderDevelopment Center.—2021.—URL: https://midnight-commander. org/.

4. NASMAssemblyLanguageTutorials.—2021.—URL: https://asmtutor.com/.

5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 c. —(In a Nutshell). —ISBN 0596009658.—URL: http://www.amazon.com/Learning bash-Shell-Programming-Nutshell/dp/0596009658.

6. RobbinsA. Bash Pocket Reference.—O'Reilly Media,2016.—156 c.—ISBN 978-1491941591.

7. TheNASMdocumentation.—2021.—URL:https://www.nasm.us/docs.php.

8. Zarrelli G. Mastering Bash.—Packt Publishing,2017.—502 c.—ISBN 9781784396879.

9. Колдаев В. Д.,Лупин С. А.Архитектура ЭВМ.—М. : Форум,2018.

10. Куляс О. Л.,Никитин К. А. Курс программирования наASSEMBLER.—М. : Солон-Пресс, 2017.

11. НовожиловО.П.Архитектура ЭВМисистем.—М.:Юрайт,2016.

12. Расширенныйассемблер:NASM.—2021.—URL:https://www.opennet.ru/docs/RUS/nasm/.

13. Робачевский А.,Немнюгин С.,Стесик О. Операционная система UNIX.—2-е изд.—БХВ Петербург, 2010.—656 с.—ISBN 978-5-94157-538-1.

14. СтоляровА.Программированиенаязыкеассемблера NASMдляОСUnix.—2-еизд.— М. : МАКСПресс,2011.—URL:http://www.stolyarov.info/books/asm_unix.

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).

16. Таненбаум Э.,Бос Х. Современные операционные системы.—4-е изд.—СПб.: Питер, 2015. —1120 с.—(Классика Computer Science)