

Лабораторная работа №6

Архитектура вычислительных систем

Заболотная Кристина Александровна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Ответы на вопросы:	20
6	Выводы	21
	Список литературы	22

Список иллюстраций

4.1	61.png	9
4.2	62.png	9
4.3	63.png	10
4.4	64.png	11
4.5	65.png	11
4.6	66.png	12
4.7	67.png	12
4.8	68.png	12
4.9	69.png	13
4.10	610.png	13
4.11	611.png	14
4.12	612.png	15
4.13	613.png	15
4.14	614.png	15
4.15	615.png	16
4.16	616.png	16
4.17	617.png	16
4.18	618png	17
4.19	619png	17
4.20	620png	18
4.21	621png	19

Список таблиц

1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

2 Задание

Написать программу вычисления выражения. 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 выбрать из таблицы 6.3 вариантов заданий в соответствии с номером полученным при выполнении лабораторной работы. Создать исполняемый файл и проверить его работу для значений из 6.3.

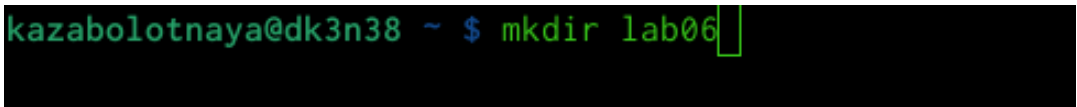
3 Теоретическое введение

1. Адресация в NASM 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.
2. Арифметические операции в NASM 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.
3. Целочисленное вычитание `sub` Команда целочисленного вычитания `sub` (от англ. subtraction – вычитание) работает аналогично команде `add`.
4. 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.
5. Команда изменения знака операнда `neg` Команда рассматривает свой операнд как число со знаком и меняет знак операнда на противоположный. Операндом может быть регистр или ячейка памяти любого размера.

6. 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). Для знакового умножения используется команда `imul`.
7. Команды деления `div` и `idiv` Для деления, как и для умножения, существует 2 команды `div` (от англ. `divide` - деление) и `idiv`. Для беззнакового умножения используется команда `div`. Для знакового умножения используется команда `idiv`.

4 Выполнение лабораторной работы

1. Создадим директорию для лабораторной работы №6.



```
kazaboltnaya@dk3n38 ~ $ mkdir lab06
```

Рис. 4.1: 61.png

2. Перейдем в нее и создадим файл lab6-1.asm.



```
kazaboltnaya@dk3n38 ~/work/arch-pc/lab06 $ touch lab06-1.asm  
kazaboltnaya@dk3n38 ~/work/arch-pc/lab06 $ nano lab6-1.asm
```

Рис. 4.2: 62.png

3. Введем в файл lab6-1.asm текст программы из листинга 6.1.

```
%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
```

Рис. 4.3: 63.png

4. Создадим копию файла in_out.asm в каталоге.

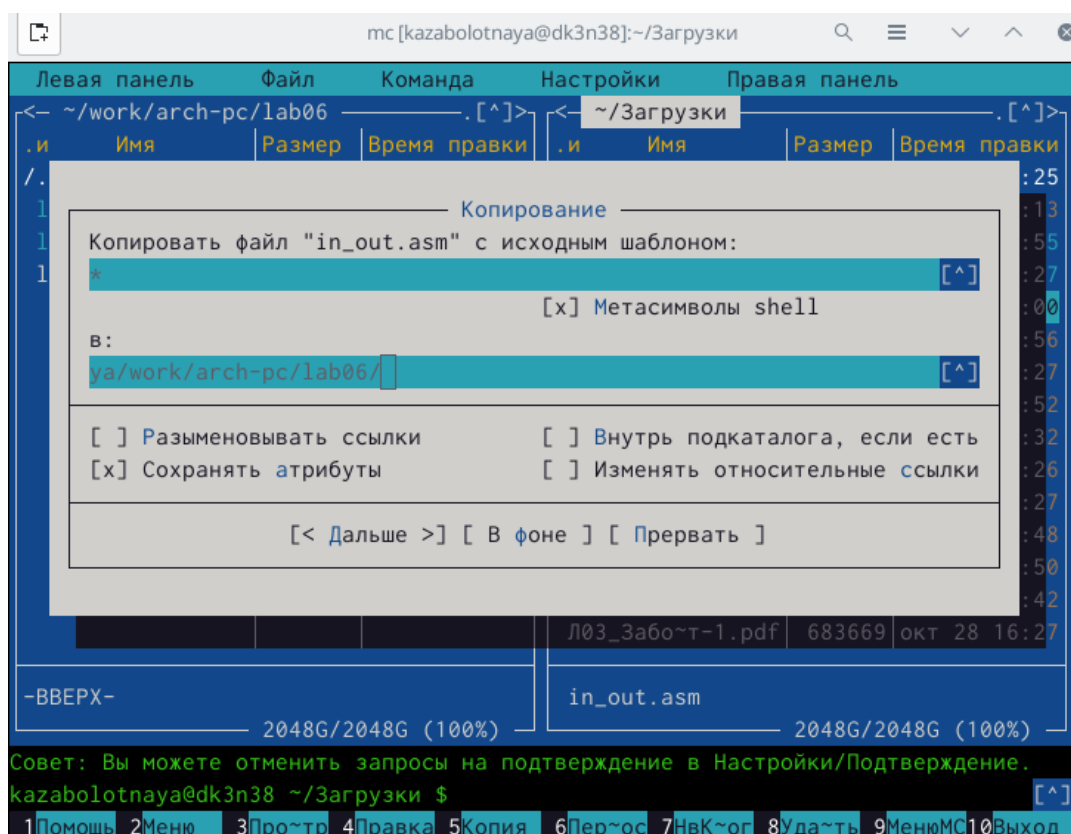


Рис. 4.4: 64.png

5. Создадим исполняемый файл и запустим его.

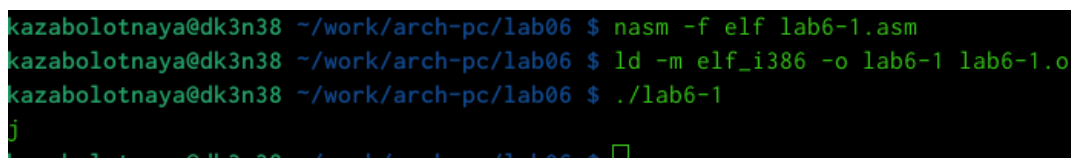


Рис. 4.5: 65.png

6. Далее изменим текст программы и вместо символов, запишем в регистры числа. Исправим текст программы.

```
%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
```

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить M-U Отмена

Рис. 4.6: 66.png

7. Создадим исполняемый файл и запустим его (6-1).

```
kazaboltnaya@dk3n38 ~/work/arch-pc/lab06 $ nano lab6-1.asm
kazaboltnaya@dk3n38 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
kazaboltnaya@dk3n38 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
kazaboltnaya@dk3n38 ~/work/arch-pc/lab06 $ ./lab6-1
```

Рис. 4.7: 67.png

8. Создадим файл lab6-2.asm в каталоге.

```
kazaboltnaya@dk3n38 ~/work/arch-pc/lab06 $ touch lab6-2.asm
kazaboltnaya@dk3n38 ~/work/arch-pc/lab06 $ nano lab6-2.asm
```

Рис. 4.8: 68.png

9. Введем в него текст программы из листинга 6.2.

```
%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,'6'
mov ebx,'4'
add eax,ebx
call iprintLF
call quit
```

Сохранить изменённый буфер? |

Y Да

N Нет ^C Отмена

Рис. 4.9: 69.png

10. Создадим исполняемый файл и запустим его (6-2).

```
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ ./lab6-2
106
```

Рис. 4.10: 610.png

11. Изменим символы на числа в lab6-2.

```
GNU nano 0.3
#include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprintLF
call quit
```

Рис. 4.11: 611.png

12. Создадим исполняемый файл и запустим его.

```
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ ./lab6-2
10
```

Рис. 4.12: 612.png

13. Создадим файл lab6-3.asm в каталоге.

```
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ touch lab6-3.asm
```

Рис. 4.13: 613.png

14. Введем в файл lab6-3.asm текст программы из листинга 6.3.

```
; ---- 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
```

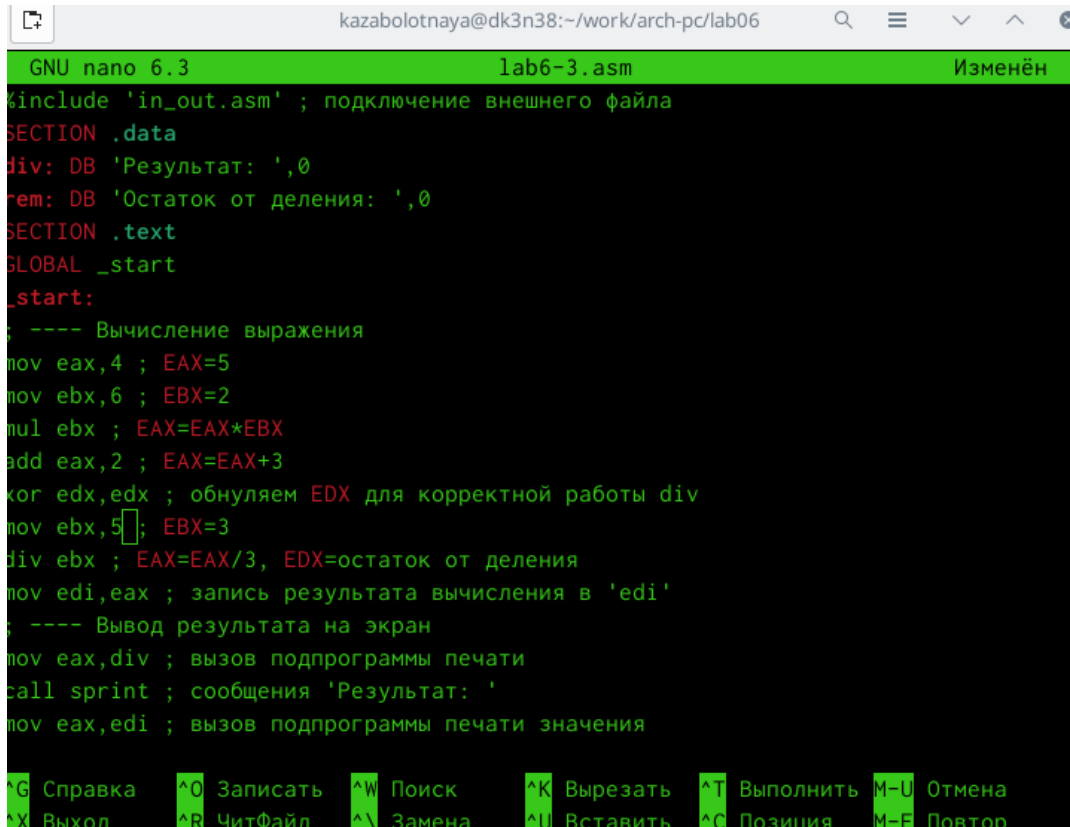
Рис. 4.14: 614.png

15. Создадим исполняемый файл и запустим его.

```
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 4
Остаток от деления: 1
```

Рис. 4.15: 615.png

16. Введем в файл lab6-3 программу вычисления выражения $\boxed{x}(\boxed{x}) = (5 \times 2 + 3)/3$.



```
GNU nano 6.3 lab6-3.asm Изменён
#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,4 ; EAX=5
mov ebx,6 ; EBX=2
mul ebx ; EAX=EAX*EBX
add eax,2 ; EAX=EAX+3
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,5 ; 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

^G Справка ^O Записать ^W Поиск ^K Вырезать ^T Выполнить M-U Отмена
^X Выход ^R ЧитФайл ^N Замена ^U Вставить ^C Позиция M-E Повтор
```

Рис. 4.16: 616.png

17. Создадим исполняемый файл и запустим его для вычисления выражения $\boxed{x}(\boxed{x}) = (5 \times 2 + 3)/3$.

```
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 5
Остаток от деления: 1
```

Рис. 4.17: 617.png

18. Создадим файл variant.asm в каталоге ~/work/arch-pc/lab06:

```
kazaboltnaya@dk3n38 ~/work/arch-pc/lab06 $ touch variant.asm
kazaboltnaya@dk3n38 ~/work/arch-pc/lab06 $ nano variant.asm
```

Рис. 4.18: 618png

19. Изменение содержимого variant.asm.

```
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите No 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
```

Рис. 4.19: 619png

20. Изменение кода.

```

GNU nano 6.3 lab6-4.asm
#include 'in_out.asm' ; подключение внешнего файла
SECTION .data
div: DB 'Результат: ',0
rem: DB 'Задайте переменную x=: ',0
rem1: DB 'x= : ',0
SECTION .bss
x: RESB 80
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения
mov eax, rem
call sprintLF
mov eax, rem1
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax,x
call atoi
mov ebx,9
mul ebx
sub eax,8
xor edx,edx
mov ebx,8
div ebx
mov edi,eax
mov eax,div
call sprint
    mov eax,edi
call iprintLF
call quit

```

Рис. 4.20: 620png

21. Вывод lab6-4 (Самостоятельная работа).

```
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ nano lab6-4.asm
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ nasm -f elf lab6-4.asm
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-4 lab6-4.o
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ ./lab6-4
Задайте переменную x=:
x= : 8
Результат: 8
kazabolotnaya@dk3n38 ~/work/arch-pc/lab06 $ ./lab6-4
Задайте переменную x=:
x= : 64
Результат: 71
```

Рис. 4.21: 621png

5 Ответы на вопросы:

1. `mov eax, x` и `rem call sprint`;
2. `mov ecx, x` - запись входной переменной в регистр `ecx`; `mov edx, 80` - запись размера переменной в регистр `edx`; `call sread` - вызов процедуры чтения данных;
3. `call atoi` - функция преобразующая ASCII код символа в целое число и записывающая результат в регистр `eax`;
4. `xor edx, edx` `mov ebx, 20` `div ebx`, `inc edx`;
5. `div ebx` - `ebx`;
6. `inc` - используется для увеличения операнда на единицу;
7. `mov eax, x` `rem call sprint` `mov eax, edx` `call iprintLF`.

6 Выводы

В ходе выполнения данной лабораторной работы были освоены арифметические инструкции языка ассемблера NASM.

Список литературы