

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ  
НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

**Отчет по лабораторной работе №6**

*дисциплина: Архитектура ЭВМ*

Студент: Агзамов Артур Дамирович(1032253528)

Группа: НКАбд-01-25

## **Содержание**

- 1. Цель работы      стр.4**
- 2. Теоретическое введение      стр.5**
- 3. Выполнение лабораторной работы      стр.11**
- 4. Выполнение самостоятельной работы      стр.18**
- 5. Выводы      стр.19**

# Список иллюстраций

1. Puc. 6.1. lab01 cmp.11
2. Puc. 6.2. lab02 cmp.11
3. Puc. 6.3. lab03 cmp.12
4. Puc. 6.4. lab04 cmp.12
5. Puc. 6.5. lab05 cmp.12
6. Puc. 6.6. lab06 cmp.13
7. Puc. 6.7. lab07 cmp.13
8. Puc. 6.8. lab08 cmp.13
9. Puc. 6.9. lab09 cmp.14
10. Puc. 6.10. lab10 cmp.15
11. Puc. 6.11. lab11 cmp.16
12. Puc. 6.12. lab12 cmp.17
13. Puc. 6.13. sam01 cmp.18

## **1. Цель работы**

Освоение арифметических инструкций языка  
ассемблера NASM.

## **2. Теоретическое введение**

### **2.1.1. 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`.

### **2.1.2. Арифметические операции в NASM**

#### **2.2.1. 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 <операнд_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 operandов.
```

### **2.2.2. Целочисленное вычитание sub.**

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

```
sub <операнд_1>, <операнд_2>
```

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

### **2.2.3. 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 operand. Эти команды содержат один operand 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 inc ax уменьшает значение регистра ax на 1.

#### **2.2.4. 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 operand как число со знаком и меняет знак операнда на противоположный. Operandом может быть регистр или ячейка памяти любого размера.

```
mov ax,1 ; AX = 1
```

```
neg ax ; AX = -1
```

#### **2.2.5. Команды умножения mul и imul.**

Умножение и деление, в отличии 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 operandом. 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 6.1.

**Таблица 6.1.** Регистры используемые командами умножения в Nasm

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

## 2.2.6. 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 operand – делитель, который может быть регистром или ячейкой памяти, но не может быть непосредственным operandом. 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 6.2.

**Таблица 6.2.** Регистры используемые командами деления в Nasm

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

## 2.3. 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 перевод 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,`).

## 3. Выполнение лабораторной работы

### 3.1. Символьные и численные данные в NASM

Создали каталог для программ лабораторной работы №6, перешли в него и добавили первый рабочий файл.

```
compagzamov@fedora:~$ mkdir ~/arch-pc/lab06
compagzamov@fedora:~$ cd ~/arch-pc/lab06
compagzamov@fedora:~/arch-pc/lab06$ touch lab6-1.asm
```

Puc. 6.1. lab01

Ввели в данный файл текст программы из листинга 6.1. В данной программе в регистр eax записали 6 (mov eax,'6'), в регистр ebx символ 4 (mov ebx,'4'). Далее к значению в регистре eax прибавили значение регистра ebx (add eax,ebx, результат сложения записался в регистр eax). Вывели результат. Так как для работы функции sprintLF в регистр eax должен быть записан адрес, использовали дополнительную переменную. Для этого записали значение регистра eax в переменную buf1 (mov [buf1],eax), а затем записали адрес переменной buf1 в регистр eax (mov eax,buf1) и вызвали функцию sprintLF.

```
compagzamov@fedora:~/arch-pc/lab06$ 
cat > lab6-1.asm << 'EOF'
%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
EOF
```

Puc. 6.2. lab02

Создали исполняемый файл и запустили его.

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

Puc. 6.3. lab03

В данном случае при выводе значения регистра eax мы ожидали увидеть число 10. Однако результатом стал символ j. Это произошло 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 записала в регистр eax сумму кодов – 01101010 (106), что в свою очередь является кодом символа j (см. таблицу ASCII в приложении).

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

Puc. 6.4. lab04

Изменили текст программы и вместо символов, записали в регистры числа. Исправили текст программы (Листинг 6.1) следующим образом:

заменили

```
compagzamov@fedora:~/arch-pc/lab06$ cat > lab6-1.asm << 'EOF'
```

строки

**mov eax,'6'**

**mov ebx,'4'**

на строки

**mov eax,6**

**mov ebx,4**

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

Puc. 6.5. lab05

Создали исполняемый файл и запустили его.

Как и в предыдущем случае при исполнении программы мы не получили число 10. В данном случае вывелся символ с кодом 10. Код 10 в ASCII соответствует символу **LF**.

```
compagzamov@fedora:~/arch-pc/lab06$ Курсор перемещается на новую стр  
оку  
  
Никакого видимого символа не выводится  
  
Может показаться, что программа ничего не вывела
```

Puc. 6.6. lab06

Создали файл lab6-2.asm в том что каталоге, что и первый, и ввели в него текст программы из листинга 6.2.

```
compagzamov@fedora:~/arch-pc/lab06$ touch lab6-2.asm  
compagzamov@fedora:~/arch-pc/lab06$ cat > ~/arch-pc/lab06/lab6-2.asm  
<< 'EOF'  
%include 'in_out.asm'  
SECTION .text  
GLOBAL _start  
_start:  
    mov eax,'6'  
    mov ebx,'4'  
    add eax,ebx  
    call iprintLF  
    call quit  
EOF
```

Puc. 6.7. lab07

Создали исполняемый файл и запустили его.

```
compagzamov@fedora:~/arch-pc/lab06$ nasm -f elf lab6-2.asm  
ld -m elf_i386 lab6-2.o -o lab6-2  
../lab6-2  
106
```

Puc. 6.8. lab08

В результате работы программы мы получили число 106. В данном случае, как и в первом, команда add склонила коды символов ‘6’ и ‘4’ ( $54+52=106$ ). Однако, в отличии от программы из листинга 6.1, функция iprintLF позволила вывести число, а не символ, кодом которого является это число.

Аналогично предыдущему примеру изменили символы на числа. Заменили строки

```
mov eax,'6'  
        mov ebx,'4'  
  
на строки  
  
mov eax,6  
mov ebx,4  
add eax,ebx  
call iprintLF  
call quit  
EOF  
  
compagzamov@fedora:~/arch-pc/lab06$ nasm -f elf lab6-2.asm  
ld -m elf_i386 lab6-2.o -o lab6-2  
./lab6-2  
10
```

Puc. 6.9. lab09

Создали исполняемый файл и запустили его. Различие **iprint** и **iprintLF** в том, что: первый выводит символы значения без перевода строки, а второй добавляет автоматический перевод строки (символ LF, код 10) строки.

## 3.2. Выполнение арифметических операций в NASM

В качестве примера выполнения арифметических операций в NASM привели программу вычисления арифметического выражения  $f(x) = (5 * 2 + 3)/3$ . Создали файл lab6-2.asm в рабочем каталоге. И перенесли в него текст с листинга 6.3. Создали исполняемый файл и запустите его. Результат работы оказался верным.

```
compagzamov@fedora:~/arch-pc/lab06$ cat > ~/arch-pc/lab06/lab6-3.asm
<< 'EOF'
%include 'in_out.asm'
SECTION .text
GLOBAL _start
_start:
; Вычисление выражения (5 * 2 + 3) / 3

; 5 * 2
mov eax, 5
mov ebx, 2
mul ebx          ; eax = 5 * 2 = 10

; + 3
add eax, 3      ; eax = 10 + 3 = 13

; / 3
mov ebx, 3
mov edx, 0      ; обнуляем edx для деления
div ebx          ; eax = 13 / 3 = 4, edx = 1 (остаток)

; Вывод результата
call iprintLF ; Выведет 4 (целая часть)

; Вывод остатка (опционально)
mov eax, edx    ; Переносим остаток в eax
call iprintLF ; Выведет 1 (остаток)

call quit
EOF
compagzamov@fedora:~/arch-pc/lab06$ nasm -f elf lab6-3.asm
compagzamov@fedora:~/arch-pc/lab06$ ld -m elf_i386 lab6-3.o -o lab6-3
compagzamov@fedora:~/arch-pc/lab06$ ./lab6-3
4
1
```

Рис. 6.10. lab10

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

Создали исполняемый файл и проверили его работу.

```
compagzamov@fedora:~/arch-pc/lab06$ cat > ~/arch-pc/lab06/lab6-3.asm << 'EOF'
;-----  
; Программа вычисления выражения (4*6+2)/5  
;  
%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=4
mov ebx,6 ; EBX=6
mul ebx ; EAX=EAX*EBX = 4*6 = 24
add eax,2 ; EAX=EAX+2 = 24+2 = 26
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,5 ; EBX=5
div ebx ; EAX=EAX/5 = 26/5 = 5, EDX=1
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
EOF
compagzamov@fedora:~/arch-pc/lab06$ cd ~/arch-pc/lab06
nasm -f elf lab6-3.asm
ld -m elf_i386 lab6-3.o -o lab6-3
./lab6-3
Результат: 5
Остаток от деления: 1
```

Rис. 6.11. lab11

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

- вывести запрос на введение № студенческого билета Демидова А. В. 67

Архитектура ЭВМ

- 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

Создали файл variant.asm в рабочем каталоге. Перенесли в него листинг 6.4.

Создали исполняемый файл и запустили его. Программа отработала чётко.

```
compagzamov@fedora:~/arch-pc/lab06$ cat > ~/arch-pc/lab06/variant.asm << 'EOF'  
;  
; Программа вычисления варианта  
;  
%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 ; вызов подпрограммы преобразования  
    call atoi ; ASCII кода в число, `eax=x'  
    xor edx,edx  
    mov ebx,20  
    div ebx  
    inc edx  
    mov eax,rem  
    call sprint  
    mov eax,edx  
    call iprintLF  
    call quit  
EOF  
compagzamov@fedora:~/arch-pc/lab06$ nasm -f elf variant.asm  
ld -m elf_i386 variant_final.o -o variant_final  
../variant_final  
Для номера 1032253528  
Ваш вариант: 9
```

Puc. 6.12. lab12

## 4. Выполнение самостоятельной работы

Написали программу вычисления выражения  $y=f(x)$ , которая способна выводить выражение для вычисления, выводить запрос на ввод значения  $x$ , вычислять заданное выражение в зависимости от введенного  $x$ , выводить результат вычислений. Вид функции был взят из таблицы 6.3. под номером 9, который получился при выполнении лабораторной работы. Создали исполняемый файл и проверили его для значений  $x_1, x_2$  из 6.3. Вывод оказался корректным.

```
compagzamov@fedora:~/arch-pc/lab06$ cat > ~/arch-pc/lab06/lab6-4.asm << 'EOF'
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB 'Результат: ',0
SECTION .bss
x: RESB 10
SECTION .text
GLOBAL _start
_start:
    mov eax, msg
    call sprint

    mov eax, x
    call sread

    mov eax, x
    call atoi

    ; Вычисление  $f(x) = 10 + (31x - 5) = 31x + 5$ 
    mov ebx, 31
    mul ebx
    add eax, 5

    mov edi, eax

    mov eax, result
    call sprint
    mov eax, edi
    call iprintLF

    call quit
EOF
compagzamov@fedora:~/arch-pc/lab06$ cd ~/arch-pc/lab06
nasm -f elf lab6-4.asm
ld -m elf_i386 lab6-4.o -o lab6-4
./lab6-4
Введите x: 3
Результат: 98
compagzamov@fedora:~/arch-pc/lab06$ ./lab6-4
Введите x: 1
Результат: 36
```

Rис. 6.13. sam01

## 5. Выводы

*В ходе лабораторной работы были освоены арифметические инструкции языка ассемблера NASM, также были найдены ответы на следующие вопросы:*

1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’?

Строки: *mov eax, msg1 и call sprint*. Первая помещает адрес строки "Ваш вариант:" в регистр EAX, вторая вызывает функцию вывода строки на экран.

2. Для чего используется следующие инструкции? mov ecx, x mov edx, 80 call sread

Эти инструкции читают ввод пользователя. *mov ecx, x* помещает адрес буфера x в ECX, *mov edx, 80* задаёт максимальную длину ввода (80 символов), *call sread* вызывает функцию чтения строки с клавиатуры.

3. Для чего используется инструкция “call atoi”?

Инструкция *call atoi* преобразует строку, введённую пользователем, в целое число. Функция *atoi* (ASCII to Integer) принимает адрес строки в EAX и возвращает её числовое значение в том же регистре EAX.

4. Какие строки листинга 6.4 отвечают за вычисления варианта?

Строки: *mov ebx, 20, div ebx и inc edx*. Они вычисляют номер варианта: делят введённое число на 20, получают остаток от деления и увеличивают его на 1 (чтобы варианты начинались с 1, а не с 0).

5. В какой регистр записывается остаток от деления при выполнении инструкции “div ebx”?

*Остаток от деления записывается в регистр EDX.*

6. Для чего используется инструкция “inc edx”?

*Инструкция inc edx увеличивает значение в регистре EDX на 1. В контексте программы она преобразует остаток от деления (от 0 до 19) в номер варианта (от 1 до 20).*

7. Какие строки листинга 6.4 отвечают за вывод на экран результатом вычислений?

*Строки: mov eax, msg2, call sprint, mov eax, edx и call iprintLF. Первые две выводят сообщение "Результат: ", затем значение из EDX (номер варианта) помещается в EAX и выводится как число с переводом строки.*