

Лабораторная работа №6

НКАБд-06-23

Улитина Мария Максимовна

Содержание

1 Цель работы.....	1
2 Задание	2
3 Теоретическое введение.....	2
3.1 Адресация в NASM	2
3.2 Арифметические операции в NASM	2
3.2.1 Целочисленное сложение add.....	2
3.2.2 Целочисленное вычитание sub.....	2
3.2.3 Команды инкремента и декремента.....	2
3.2.4 Команда изменения знака операнда neg.	3
3.2.5 Команды умножения mul и imul.....	3
3.2.6 Команды деления div и idiv.	3
3.3 Перевод символа числа в десятичную символьную запись.....	3
4 Выполнение лабораторной работы.....	4
4.1 Выполнение арифметических операций в NASM.	8
4.2 Задания для самостоятельной работы.	11
5 Выводы.....	12
Список литературы	12

1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

2 Задание

1. Работа с символьными и численными данными в NASM.
2. Выполнение арифметических операций в NASM.
3. Написание программы для вычисления заданного выражения.

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.

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 <операнд_1>, <операнд_2>`

3.2.2 Целочисленное вычитание `sub`.

Команда целочисленного вычитания `sub` (от англ. subtraction – вычитание) работает аналогично команде `add` и выглядит следующим образом: `sub <операнд_1>, <операнд_2>`

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_1>`, `dec <операнд_1>`,

3.2.4 Команда изменения знака операнда `neg`.

Еще одна команда, которую можно отнести к арифметическим командам это команда изменения знака `neg`: `neg` Команда `neg` рассматривает свой операнд как число со знаком и меняет знак операнда на противоположный.

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_1>`.

Для знакового умножения используется команда `imul`: `imul <операнд_1>`.

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.

3.2.6 Команды деления `div` и `idiv`.

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

`div` - Беззнаковое деление,

`idiv` - 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.

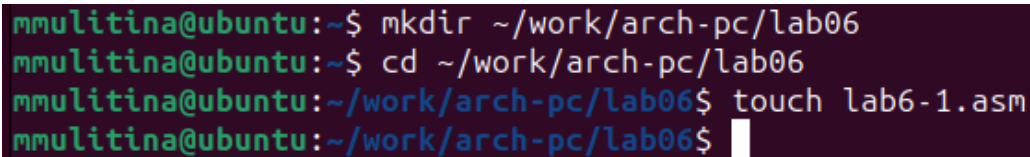
3.3 Перевод символа числа в десятичную символьную запись.

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, 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.

4 Выполнение лабораторной работы

Создадим каталог для программ, перейдём в него и создадим файл lab6-1.asm (рис. 1).



```
mmulitina@ubuntu:~$ mkdir ~/work/arch-pc/lab06
mmulitina@ubuntu:~$ cd ~/work/arch-pc/lab06
mmulitina@ubuntu:~/work/arch-pc/lab06$ touch lab6-1.asm
mmulitina@ubuntu:~/work/arch-pc/lab06$
```

Figure 1: Создание файла

Введём необходимый листинг в файл (рис. 2).

```
%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
```

Figure 2: Листинг

Создадим исполняемый файл и запустим его (рис. 3).

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

Figure 3: Исполнение

Изменим текст программы, заменим `mov eax, '6'; mov ebx, '4'` на `mov eax, 6; mov ebx, 4`. Код 10 соответствует переносу строки, не отображается (рис. 4).

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

Figure 4: Замена

Создадим файл lab6-2.asm (рис. 5).

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

Figure 5: Файл

Введём в файл предложенный листинг (рис. 6).

```
%include 'in_out.asm'

SECTION .text
GLOBAL _start
_start:

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

    call quit
```

Figure 6: Листинг

Создадим исполняемый файл и запустим его (рис. 7).

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

Figure 7: Листинг

Получаем число 106, сложив коды символов '6' и '4'. Аналогично предыдущему файлу заменим строки (рис. 8).

```
%include 'in_out.asm'
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov eax, 6
```

```
mov ebx, 4
```

```
add eax, ebx
```

```
call iprintLF
```

```
call quit
```

Figure 8: Замена

Создадим исполняемый файл и запустим его (рис. 9).

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

Figure 9: Запуск файла

Получим значение 10.

Заменим iprintLF на iprint, создадим исполняемый файл и запустим (рис. 10).

```
%include 'in_out.asm'

SECTION .text
GLOBAL _start
_start:

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

    call quit
```

Figure 10: Замена

(рис. 11).

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

Figure 11: Исполнение

С iprint не будет переноса строки.

4.1 Выполнение арифметических операций в NASM.

Создадим файл lab6-3.asm (рис. 12).

```
mmulitina@ubuntu:~/work/arch-pc/lab06$ touch ~/work/arch-pc/lab06/lab6-3.asm
```

Figure 12: Создание файла

Введём в созданный файл необходимый листинг (рис. 13).


```

;-----
; Программа вычисления выражения
;-----
%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,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'

```

Figure 13: Листинг

Создадим исполняемый файл и запустим его (рис. 14).

```

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

```

Figure 14: Работа файла

Изменим текст программы для вычисления выражения $f(x) = (4*6+2)/5$ (рис. 15).

```

%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
add eax,2 ; EAX=EAX+2
xor edx,edx ; обнуляем EDX для корректной работы div
mov ebx,5 ; EBX=5
div ebx ; EAX=EAX/3, EDX=остаток от деления

```

Figure 15: Измененный файл

Запустим исполняемый файл и проверим его работу (рис. 16).

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

Figure 16: Запуск файла

Создадим файл `variant.asm`, введём в него предложенный листинг (рис. 17).

```
;-----
; Программа вычисления варианта
;-----
%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:
```

Figure 17: Листинг

Создадим исполняемый файл и проверим его работу (рис. 18).

```
mmulitina@ubuntu:~/work/arch-pc/lab06$ nasm -f elf variant.asm
mmulitina@ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 -o variant variant.o
mmulitina@ubuntu:~/work/arch-pc/lab06$ ./variant
Введите No студенческого билета:
1132236002
Ваш вариант: 3
```

Figure 18: Вычисление номера студенческого

Ответы на вопросы:

1. За вывод на экран сообщения “Ваш вариант” отвечают строки `mov eax, msg` и `call sprintf`
2. Инструкции используются для считывания вводимого пользователем значения и записи его в переменную `x`.
3. Функция `atoi` преобразует `ascii` - код символа в целое число и записывает результат в регистр `eax`, перед вызовом `atoi` в регистр `eax` необходимо записать число.

4. За вычисление варианта отвечают строки `xor edx, edx mov ebx, 20 div ebx inc edx`
5. В `edx`.
6. Для увеличения остатка на 1.
7. `mov eax, edx call iprintLF`

4.2 Задания для самостоятельной работы.

Напишем программу для варианта 3 (рис. 19).

```
%include 'in_out.asm'
SECTION .bss
x: RESB 80
SECTION .text
GLOBAL _start
_start:
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi

add eax, 2
mov ebx, eax
mul ebx

call iprintLF

call quit
```

Figure 19: Вычисление по варианту 3

Запустим её и проверим на предложенных значениях (рис. 20).

```
mmulitina@ubuntu:~/work/arch-pc/lab06$ nasm -f elf variant1.asm
mmulitina@ubuntu:~/work/arch-pc/lab06$ ld -m elf_i386 -o variant1 variant1.o
mmulitina@ubuntu:~/work/arch-pc/lab06$ ./variant1
2
16
mmulitina@ubuntu:~/work/arch-pc/lab06$ ./variant1
8
100
```

Figure 20: Проверка

5 Выводы

В процессе выполнения работы я освоила арифметические инструкции языка ассемблера NASM.

Список литературы

Архитектура ЭВМ. Лабораторная работа №6.