# Лабораторная работа №6

Арифметические операции NASM

Сырцева Анастасия Романовна

## Содержание

## 1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

## 2 Теоретическое введение

### 2.1 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

### 2.2 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. Допустимые сочетания операндов для команды add аналогичны сочетаниям операндовдля команды mov. Так, например, команда add eax,ebx прибавит значение из регистра eax к значению из регистра ebx и запишет результат в регистр eax.

### 2.3 Целовисленное вычитание sub

Команда целочисленного вычитания sub (от англ. subtraction – вычитание) работает аналогично команде add и выглядит следующим образом:(рис. 1).

```
sub <операнд_1>, <операнд_2>
```

*Рис. 1: Целочисленное вычитание*

## 2.4 Команды инкремента и декремента

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. 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.

## 2.5 Команда изменения знака операнда neg

Еще одна команда, которую можно отнести к арифметическим командам это команда изменения знака neg. Она рассматривает свой операнд как число со знаком и меняет знак операнда на противоположный. Операндом может быть регистр или ячейка памяти любого размера.

## 2.6 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), а для знакового умножения используется команда imul. 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.1

*Таблица 1: Регистры, используемые командами умножения Nasm*

| Размер операнда | Неявный множитель | Результат умножения |
|---|---|---|
| 1 байт | AL | AX |
| 2 байта | AX | DX:AX |
| 4 байта | EAX | ADX:EAX |

## 2.7 Команды деления div и idiv

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

*Таблица 2: Регистры, используемые командами деления Nasm*

| Размер операнда(делителя) | Делимое | Частное | Остаток |
|---|---|---|---|
| 1 байт | AX | AL | AH |
| 2 байта | DX:AX | AX | DX |
| 4 байта | EDX:EAX | EAX | EDX |

## 2.8 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 перевода строки. - atoi – функция преобразует ascii-код символа в целое число и записает результат в регистр eax, перед вызовом atoi в регистр eax необходимо записать число (mov eax,).

# 3 Выполнение лабораторной работы

Создаю каталог для данной лабораторной работы аналогично 5 и 4 лаюораторным работам. Перехожу в него и создаю файл lab6-1.asm (рис. 2).

```
arsihrceva@dk2n21 ~/work $ mkdir ~/work/arch-pc/lab06
arsihrceva@dk2n21 ~/work $ cd arch-pc/lab06
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ touch lab06-1.asm
```

*Рис. 2: Создание каталога и файла*

Открываю созданный файл для редактирования и перепечатываю программу из условия лабораторной работы (рис. 3).

```
%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. 3: Программа сложения двух чисел*

Для работы программы копирую файл in_out.asm в каталог lab06 (рис. 4).

```
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ cp ~/work/in_out.asm ~/work/arch-pc/lab06/in_out.asm
```

*Рис. 4: Копирование файла in_out.asm*

Создаю исполняемый файл и запускаю его (рис. 5).

```
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ nasm -f elf lab06-1.asm
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab06-1.o
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ./lab6-1
j
```

*Рис. 5: Работа исполняемогоо файла*

В данном случае при выводе значения регистра eax ожидаю увидеть число 10. Однако результатом - символ j. Это происходит 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 запишет в регистр eax сумму кодов – 01101010 (106), что в свою очередь является кодом символа j Изменяю текст программы так, чтобы вместо символов были регистры числа (рис. 6).

```
%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. 6: Изменённый текст программы*

Создаю и запускаю исполняемый файл изменённой программы (рис. 7).

```
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ nasm -f elf lab06-1.asm
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab06-1.o
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ./lab6-1
```

*Рис. 7: Запуск изменённой программы*

Как и в предыдущем случае при исполнении программы не получаю число 10. В данном случае выводится символ с кодом 10 (LF, в соответствии с таблицей ASCII ) Создаю файл lab6-2.asm и открываю для редактирования (рис. 8).

```
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ touch lab6-2.asm
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ gedit lab6-2.asm
```

*Рис. 8: Создание нового файла*

Ввожу текст программы сложения с использованием iprintLF, который позволяет вывести число, а не символ, кодом которого является число (рис. 9).

```
 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
```

*Рис. 9: Ввод текста программы*

Создаю исполняемый файл и запускаю его (рис. 10).

```
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ./lab6-2
106
```

*Рис. 10: Запуск программы*

В результате работы программы мы получим число 106. Аналогично предыдущему примеру изменяю символы на числа. (рис. 11).

```
%include 'in_out.asm'

SECTION .text
GLOBAL _start
        _start:

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

        call quit
```

*Рис. 11: Изменённый текст программы*

Создаю и запускаю исполняемый файл (рис. 12).

```
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ./lab6-2
10
```

*Рис. 12: Запуск исполняемого файла*

Создаю новый файл lab6-3.asm в каталоге ~/work/arch-pc/lab06 и открываю его (рис. **¿fig:012?**).

```
10arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ touch lab6-3.asm
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ gedit lab6-3.asm
```

*Рис. 13: Название рисунка*

Ввожу текст программы, которая будет вычислять следующий пример:

$$(5 * 2 + 3)/3$$

(рис. **¿fig:013?**).

```
%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

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

                call quit
```

*Рис. 14: Текст программы для вычисления примера*

Создаю и запускаю исполняемый файл (рис. 13).

```
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 4
Остаток от деления: 1
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $
```

*Рис. 15: Результат программы*

Результат совпадает с правильным ответом, который указан в условии работы. Изменяю тест программы для решения примера:

$$(4 * 6 + 2)/5$$

(рис. 14).

```
%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

                call quit
```

*Рис. 16: Изменённый текст программы*

Создаю и запускаю исполняемый файл (рис. 15).

```
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 5
Остаток от деления: 1
```

*Рис. 17: Результат работы файла*

Создаю новый файл variant.asm (рис. 16).

```
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ touch variant.asm
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ gedit variant.asm
```

*Рис. 18: Создание файла*

Вожу текст программы, которая будет вычислять № варианта с помощью № студенческого билета (рис. 17).

```
%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
```

*Рис. 19: Текст программы для вычисления варианта*

Создаю и запускаю исполняемый файл (рис. 18).

```
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ nasm -f elf variant.asm
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o variant variant.o
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ./variant
Введите № студенческого билета:
1132246780
Ваш вариант: 1
```

*Рис. 20: Результат работы программы*

Вычисляя вариант по формуле:

$$(1132246780 mod 20) + 1$$

получаю так же 1 вариант.

## 3.1 Ответы на вопросы

- За вывод на экран сообщения ‘Ваш вариант:’ отвечают строки mov eax,rem и call sprint;
- Инструкция mov ecx, x используется, чтобы положить адрес вводимой строки в регистр ecx, mov edx, 80 - для записи в регистр edx длины вводимой строки, а call sread отвечает за вызов программы их внешнего файла, обезпечивающий ввод с клавиатуры.
- call atoi используется для вызова программы из in_out.asm, которая преобразует символы в число и запишет результат в регистр eax
- За вычисление варианта отвечают строки 34-37 (рис. 18)
- Остаток от деления при выполнении команды div ebx записывается в регистр edx
- Инструкция inc edx используется для увеличения значения регистра edx на один
- За вывод результата вычислений отвечают строки 41-42 (рис. 18)

# 4 Самостоятельная работа

В соответствии с вычисленным вариантом (рис. 18) нахожу нужное мне выражение для выполнения работы (рис. 19).

| Номер варианта | Выражение для $f(x)$ | $x_1$ | $x_2$ |
|---|---|---|---|
| 1 | $(10+2x)/3$ | 1 | 10 |

*Рис. 21: Мой вариант*

Создаю и открываю файл для выполнения работы (рис. 20).

```
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ touch lab6-4.asm
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ gedit lab6-4.asm
```

*Рис. 22: Создание файла*

Пишу текст программы для решения уравнения моего варинта (рис. 21).

```
%include 'in_out.asm'

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

                mov ebx,2
                mul ebx
                add eax,10
                xor edx,edx
                mov ebx,3
                div ebx

                mov edi, eax

                mov eax,rem
                call sprint
                mov eax,edi
                call iprintLF

                call quit
```

*Рис. 23: Текст программы*

Создаю и запускаю исполняемый файл, ввожу значение x1 (рис. 22).

```
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ nasm -f elf lab6-4.asm
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-4 lab6-4.o
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ./lab6-4
Введите значение переменной x:
1
Ответ: 4
```

*Рис. 24: Вывод результата программы со значением x1*

Вместо значения x1 ввожу значение x2 (рис. 23).

```
arsihrceva@dk2n21 ~/work/arch-pc/lab06 $ ./lab6-4
Введите значение переменной x:
10
Ответ: 10
```

*Рис. 25: Вывод результата со значением x2*

# 5 Выводы

Получаены навыки работы с арифметическими инструкциями языка ассемблера NASM.