

Отчёт по лабораторной работе №6

Арифметические операции в NASM

Прокопьева Марина Евгеньевна

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выводы	23
	Список литературы	24

Список иллюстраций

2.1	Создание каталога	9
2.2	Ввод текста в программу	10
2.3	Создание файла	11
2.4	Измена текста	12
2.5	Создание файла	13
2.6	Создание файла	13
2.7	Ввод текста	14
2.8	Создание файла	15
2.9	Измена текста	15
2.10	Создание файла	16
2.11	Измена текста	16
2.12	Создание файла	17
2.13	Создание файла	18
2.14	Измена текста	19
2.15	Создание файла	20
2.16	Создание файла	21
2.17	Запуск	21
2.18	Создание файла	22

Список таблиц

1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

2 Теоретическое введение

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`.

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` , 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 операндов.

Целочисленное вычитание `sub`.

Команда целочисленного вычитания `sub` (от англ. subtraction – вычитание) работает аналогично команде `add` и выглядит следующим образом: `sub` , Так, например, команда `sub ebx,5` уменьшает значение регистра `ebx` на 5 и записывает результат в регистр `ebx`.

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 dec` 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.

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. Операндом может быть регистр или ячейка памяти любого размера. `mov ax,1`; $AX = 1$ `neg ax`; $AX = -1$

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` Для знакового умножения используется команда `imul`: `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

Перевод символа числа в десятичную символьную запись

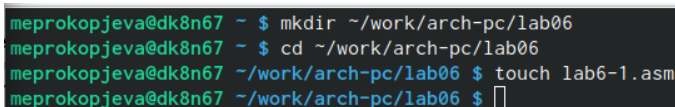
Ввод информации с клавиатуры и вывод её на экран осуществляется в символьном виде. Кодирование этой информации производится согласно кодовой таблице символов ASCII. ASCII – сокращение от `American Standard Code for Information Interchange` (Американский стандартный код для обмена информацией). Согласно 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 записывает результат в регистр `eax`, перед вызовом `atoi` в регистр `eax` необходимо записать число (`mov eax, ...`).

Выполнение лабораторной работы

1. Создание каталог для программ лабораторной работы № 6, перейдите в него и создайте файл `lab6-1.asm`:



```
meprokopjeva@dk8n67 ~ $ mkdir ~/work/arch-pc/lab06
meprokopjeva@dk8n67 ~ $ cd ~/work/arch-pc/lab06
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ touch lab6-1.asm
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $
```

Рис. 2.1: Создание каталога

2. Ввод в файл теста программы

```
/afs/.dk.sci.pfu.edu.ru/home
%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. 2.2: Ввод текста в программу

3. Создание исполняемого файла и запуск его.

```
meprokopjeva@dk8n67 ~ $ cd ~/work/arch-pc/lab06
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ ./lab6-1
j
```

Рис. 2.3: Создание файла

4. Далее изменю текст программы и вместо символов, запишу в регистры числа

```
lab6-1.asm [-M--]
#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. 2.4: Измена текста

Создание исполняемого файла и его запуск

```
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ ./lab6-1

meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ █
```

Рис. 2.5: Создание файла

5. Создание файла lab6-2 и ввод текста программы

```
meprokopjeva@dk8n67 ~ $ touch ~/work/arch-pc/lab06/lab6-2.asm
meprokopjeva@dk8n67 ~ $ cd ~/work/arch-pc/lab06
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ █
```

Рис. 2.6: Создание файла

```
lab6-1.asm [-M--]
#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. 2.7: Ввод текста

Создаю исполняемый файл и его запуск

```

meprokojjeva@dk8n67 ~$ cd ~/work/arch-pc/lab06
meprokojjeva@dk8n67 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
meprokojjeva@dk8n67 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
meprokojjeva@dk8n67 ~/work/arch-pc/lab06 $ ./lab6-2
106
meprokojjeva@dk8n67 ~/work/arch-pc/lab06 $

```

Рис. 2.8: Создание файла

6. Изменяем символы на числа

```

lab6-2.asm [-M--]
#include 'in_out.asm'

SECTION .text
GLOBAL _start
_start:
mov eax,6
mov ebx,4
add eax,ebx
call iprintLF

call quit

```

Рис. 2.9: Измена текста

Создаю исполняемый файл и его запускаю

```

meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ ./lab6-2
10
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $

```

Рис. 2.10: Создание файла

Заменяю функцию `iprintLF` на `iprint`.

```

meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ ./lab6-2
10meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $

```

Рис. 2.11: Измена текста

7. Создание файла lab6-3 и ввод текста программы


```
/afs/.dk.sci.pfu.edu.ru/home/m/e/mepr~opj
%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
```

1Помощь 2Разверн 3Выход 4Нех

Рис. 2.12: Создание файла

Создание исполняемого файла и его запускаяю

```
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 4
Остаток от деления: 1
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $
```

Рис. 2.13: Создание файла

Изменяю текст программы ждя вычисления жругой функции

```

lab6-3.asm      [----]  0 L:[ 1+16
%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. 2.14: Измена текста

Создаю исполняемый файл и запускаю его

```
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 5
Остаток от деления: 1
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $
```

Рис. 2.15: Создание файла

8. Создаю файл для вычисления варианта задания по номеру студенческого билета

```

#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 sprintf

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 sprintf
mov eax, edx

```

Рис. 2.16: Создание файла

Запускаю его

```

meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ nasm -f elf variant.asm
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o variant variant.o
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $ ./variant
Введите № студенческого билета:
1132237370
Ваш вариант: 11
meprokopjeva@dk8n67 ~/work/arch-pc/lab06 $

```

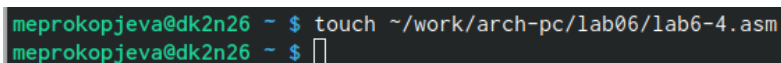
Рис. 2.17: Запуск

Ответы на вопросы:

1. За вывод сообщения “Ваш вариант” отвечают строки кода: `mov eax,rem` `call sprint`
2. Инструкция `mov ecx, x` используется, чтобы положить адрес вводимой строки `x` в регистр `ecx` `mov edx, 80` - запись в регистр `edx` длины вводимой строки `call sread` - вызов подпрограммы из внешнего файла, обеспечивающей ввод сообщения с клавиатуры
3. `call atoi` используется для вызова подпрограммы из внешнего файла, которая преобразует `ascii`-код символа в целое число и записывает результат в регистр `eax`
4. За вычисления варианта отвечают строки: `xor edx,edx` ; обнуление `edx` для корректной работы `div` `mov ebx,20` ; `ebx = 20` `div ebx` ; `eax = eax/20`, `edx` - остаток от деления `inc edx` ; `edx = edx + 1`
5. При выполнении инструкции `div ebx` остаток от деления записывается в регистр `edx`
6. Инструкция `inc edx` увеличивает значение регистра `edx` на 1
7. За вывод на экран результатов вычислений отвечают строки: `mov eax,edx` `call iprintLF`

Самостоятельная работа

1. Создаю файл для создания самостоятельной работы



```
meprokopjeva@dk2n26 ~ $ touch ~/work/arch-pc/lab06/lab6-4.asm
meprokopjeva@dk2n26 ~ $
```

Рис. 2.18: Создание файла

3 Выводы

Освоила арифметических инструкций языка ассемблера NASM.

Список литературы