

Лабораторная работа №6

Арифметические операции в NASM

Карпова Есения Алексеевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
5	Выводы	23
	Список литературы	24

Список иллюстраций

4.1	Создание директории	9
4.2	Создание копии файла	9
4.3	Редактирование файла	10
4.4	Запуск исполняемого файла	10
4.5	Замена символов в файле	11
4.6	Вывод исполняемого файла	11
4.7	Редактирование файла	12
4.8	Запуск исполняемого файла	12
4.9	Замена символов на числа	12
4.10	Запуск исполняемого файла	13
4.11	Редактирование файла	14
4.12	Запуск исполняемого файла	15
4.13	Ввод текста программы	16
4.14	Запуск исполняемого файла	16
4.15	Изменение программы	17
4.16	Запуск исполняемого файла	18
4.17	Редактирование файла	18
4.18	Запуск исполняемого файла	19
4.19	Создание нового файла	20
4.20	Текст программы	21
4.21	Текст программы	22

Список таблиц

1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

2 Задание

1. Символные и численные данные в NASM
2. Выполнение арифметических операций в NASM
3. Выполнение заданий для самостоятельной работы

3 Теоретическое введение

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. 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.

4 Выполнение лабораторной работы

1. Символьные и численные данные в NASM Создаю директорию, в которой буду создавать файлы с программами для лабораторной работы №6. Перехожу в созданный каталог с помощью утилиты `cd` и создаю файл `lab6-1.asm` (рис. 4.1).

```
eakarpova@dk4n62 ~ $ mkdir ~/work/arch-pc/lab06
eakarpova@dk4n62 ~ $ cd ~/work/arch-pc/lab06
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ touch lab6-1.asm
eakarpova@dk4n62 ~/work/arch-pc/lab06 $
```

Рис. 4.1: Создание директории

Копирую в текущий каталог файл `in_out.asm` с помощью утилиты `cp`, т.к. он будет использоваться в других программах (рис. 4.2).

```
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ cp ~/work/arch-pc/lab05/in_out.asm in_out.asm
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ls
in_out.asm  lab6-1.asm
eakarpova@dk4n62 ~/work/arch-pc/lab06 $
```

Рис. 4.2: Создание копии файла

Открываю созданный файл и вставляю в него программу вывода значения регистра `eax` (рис. 4.3).

```

GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/e/a/eakarpova/work/arch-pc/lab06/lab6-1.asm
#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: Редактирование файла

Создаю исполняемый файл программы и запускаю его (рис. 4.4). Программа выводит “j”, так как этот символ соответствует сумме двоичных кодов символов 4 и 6 по системе ASCII

```

eakarpova@dk4n62 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ./lab6-1
j
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ █

```

Рис. 4.4: Запуск исполняемого файла

Изменяю в тексте программы символы “6” и “4” на цифры 6 и 4(рис. 4.5).

```
GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/e/a/eakarpova/work/arch-pc/lab06/lab6-1.asm
%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. 4.5: Замена символов в файле

Создаю новый исполняемый файл программы и запускаю его. Теперь вывелся символ с кодом 10, это символ перевода строки, он не отображается при выводе на экран (рис. 4.6).

```
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ nasm -f elf lab6-1.asm
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-1 lab6-1.o
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ./lab6-1
```

Рис. 4.6: Вывод исполняемого файла

Создаю новый файл lab6-2.asm и ввожу в него текст другой программы для вывода значения регистра eax (рис. 4.7).

```

GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/e/a/eakarpova/work/arch-pc/lab06/lab6-2.asm
#include 'in_out.asm'

SECTION .text
GLOBAL _start
_start:

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

call quit

```

Рис. 4.7: Редактирование файла

Запускаю исполняемый файл lab6-2. Теперь вывод: число 106, так как программа позволяет ввести именно число, а не символ, хотя все еще происходит сложение кодов символов “6” и “4”(рис. 4.8).

```

eakarpova@dk4n62 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ./lab6-2
106

```

Рис. 4.8: Запуск исполняемого файла

Заменяю в тексте программы символы “6” и “4” на числа 6 и 4(рис. 4.9).

```

GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/e/a/eakarpova/work/arch-pc/lab06/lab6-2.asm
#include 'in_out.asm'

SECTION .text
GLOBAL _start
_start:

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

call quit

```

Рис. 4.9: Замена символов на числа

Создаю и запускаю новый исполняемый файл. теперь программа складывает

не соответствующие символам коды в системе ASCII, а сами числа, поэтому вывод 10(рис. 4.10).

```
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ./lab6-2
10
```

Рис. 4.10: Запуск исполняемого файла

Заменяю в текст программы функцию `iprintLF` на `iprint`(рис. 4.11).

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

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov eax, 6
```

```
mov ebx, 4
```

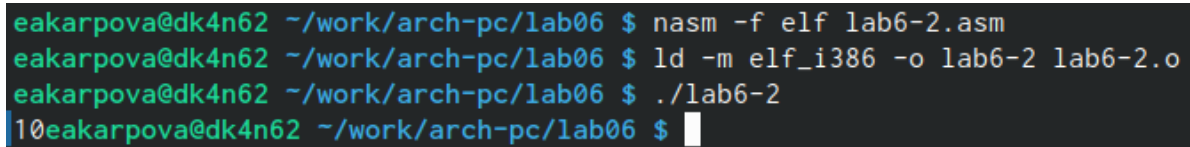
```
add eax, ebx
```

```
call iprint
```

```
call quit
```

Рис. 4.11: Редактирование файла

Запускаю исполняемый файл. Вывод не изменился, но `iprint` не добавляет к выводу символ переноса строки в отличие от `iprintLF` (рис. 4.12).



```
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ nasm -f elf lab6-2.asm
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-2 lab6-2.o
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ./lab6-2
10eakarpova@dk4n62 ~/work/arch-pc/lab06 $
```

Рис. 4.12: Запуск исполняемого файла

2. Выполнение арифметических операций в NASM

Создаю файл `lab6-3.asm` и ввожу в него текст программы для вычисления значения выражения $f(x) = (5 \cdot 2 + 3) / 3$ (рис. 4.13).

```

GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/e/a/eakarpova/work/arch-pc/lab06/lab6-3.asm
;-----
; 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'

; ---- 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. 4.13: Ввод текста программы

Создаю исполняемый файл и запускаю его(рис. 4.14).

```

eakarpova@dk4n62 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 4
Остаток от деления: 1

```

Рис. 4.14: Запуск исполняемого файла

Изменяю программу так, чтобы она вычисляла значение выражения $f(x) = (4*6$

+ 2)/5(рис. 4.14).

```
GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/e/a/eakarpova/work/arch-pc
;-----
; 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,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/5, 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 exit; завершение программы
```

Рис. 4.15: Изменение программы

Создаю и запускаю исполняемый файл. Программа отработала верно(рис. 4.16).

```
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ nasm -f elf lab6-3.asm
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-3 lab6-3.o
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ./lab6-3
Результат: 5
Остаток от деления: 1
```

Рис. 4.16: Запуск исполняемого файла

Создаю файл variant.asm и ввожу в него текст программы для вычисления задания по номеру студенческого билета(рис. 4.17).

```
GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/e/a/eakarpova/work/arch-pc/lab06/variant.asm
%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
call iprintLF

call quit
```

Рис. 4.17: Редактирование файла

Создаю и запускаю исполняемый файл. ввожу номер своего студенческого билета, программа вывела, что мой вариант - 9(рис. 4.18).

```
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ nasm -f elf variant.asm
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o variant variant.o
eakarpova@dk4n62 ~/work/arch-pc/lab06 $ ./variant
Введите № студенческого билета:
1132236008
Ваш вариант: 9
```

Рис. 4.18: Запуск исполняемого файла

Ответы на вопросы: 1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’? Ответ: `mov eax,rem call sprint` 2. Для чего используются следующие инструкции? 1) `mov ecx, x` 2) `mov edx, 80` 3) `call sread` Ответ: 1) Инструкция `mov ecx, x` используется, чтобы положить в адрес вводимой строки `x` в регистр `ecx` 2) `mov edx, 80` - запись в регистр `edx` длины вводимой строки 3) `call sread` - вызов подпрограммы из внешнего файла, обеспечивающей ввод сообщения с клавиатуры 3. Для чего используется инструкция “`call atoi`”? Ответ: `call atoi` используется для вызова подпрограммы из внешнего файла, который преобразует ASCII-код символа в целое число и записывает результат в регистр `eax` 4. Какие строки листинга 6.4 отвечают за вычисления варианта? Ответ: за вычисление варианта отвечают строки: `xor edx,edx mov ebx,20 div ebx inc edx` 5. В какой регистр записывается остаток от деления при выполнении инструкции “`div ebx`”? Ответ: при выполнении инструкции `div ebx` остаток от деления записывается в регистр `edx` 6. Для чего используется инструкция “`inc edx`”? Ответ: инструкция `inc edx` увеличивает значение регистра `edx` на 1 7. Какие строки листинга 6.4 отвечают за вывод на экран результата вычислений? Ответ: за вывод экран результатов вычисления отвечают строки `mov eax,edx call iprintLF` 3. Выполнение заданий для самостоятельной работы

Создаю файл `lab6-5.asm` с помощью утилиты `touch`(рис. 4.19).

```
eakarpova@dk5n60 ~/work/arch-pc/lab06 $ touch lab6-5.asm  
eakarpova@dk5n60 ~/work/arch-pc/lab06 $ mc
```

Рис. 4.19: Создание нового файла

Открываю созданный файл для редактирования, ввожу текст программы для вычисления значения выражения №9: $10 + (31x - 5)$ (рис. 4.20).

```

%include          'in_out.asm'

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

SECTION .bss
x:      RESB 80

SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
mov ebx, 31
mul ebx ; x*31
add eax, -5 ; x*31 - 5
add eax, 10; (x*31 - 5) + 10
mov edi, eax
; ---- Вывод результата на экран
mov eax, rem
call sprint
mov eax, edi
call iprintLF
call quit

```

Рис. 4.20: Текст программы

Создаю и запускаю исполняемый файл с разными значениями на входе. Убеждаюсь, что программа работает правильно(рис. 4.21).

```
eakarpova@dk5n60 ~/work/arch-pc/lab06 $ nasm -f elf lab6-5.asm
eakarpova@dk5n60 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-5 lab6-5.o
eakarpova@dk5n60 ~/work/arch-pc/lab06 $ ./lab6-5
Введите значение переменной x: 3
Результат: 98
eakarpova@dk5n60 ~/work/arch-pc/lab06 $ nasm -f elf lab6-5.asm
eakarpova@dk5n60 ~/work/arch-pc/lab06 $ ld -m elf_i386 -o lab6-5 lab6-5.o
eakarpova@dk5n60 ~/work/arch-pc/lab06 $ ./lab6-5
Введите значение переменной x: 1
Результат: 36
eakarpova@dk5n60 ~/work/arch-pc/lab06 $
```

Рис. 4.21: Текст программы

5 Выводы

При выполнении данной лабораторной работы я освоила арифметические инструкции языка NASM

Список литературы

1, Лабораторная работа№6 - Демидова А.В.