РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6

дисциплина: Архитектура компьютера

Студент: Ким Илья

Группа: НКАбд-03-25

МОСКВА

2025 г.

Содержание

Список иллюстраций

1. Цель работа

Цель данной лабораторной работы - освоение арифметических инструкций я

2. Задание

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

3. Теоретическое ведение

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. 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. 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. 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 это 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.

4.1. Символьные и численные данные в NASM

Создаю каталог для программ лабораторной работы №6 и перехожу в него, создаю там файл (рис.4.1.).

```
iekim@LINUXUBUNTU:~$ mkdir ~/work/arch-pc/lab06
iekim@LINUXUBUNTU:~$ cd ~ ~/work/arch-pc/lab06
bash: cd: слишком много аргументов
iekim@LINUXUBUNTU:~$ cd ~ ~/work/arch-pc/lab06
bash: cd: слишком много аргументов
iekim@LINUXUBUNTU:~$ cd ~/work/arch-pc/lab06
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ touch lab6-1.asm
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$
```

Рис. 4.1: Создание нового каталога

В созданном файле ввожу программу из листинга (рис.4.2).

```
%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 sprintLF
    call quit
```

Рис. 4.2: Сохранение новой программы

Создаю исполняемый файл и запускаю его, вывод программы отличается от предполагаемого изначально, ибо коды символов в сумме дают символ j по таблице ASCII. {#fig:003 width=70%}

```
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ ld 0m elf_i386 -o lab6-1 lab6-1.o
ld: невозможно найти 0m: Нет такого файла или каталога
ld: невозможно найти elf_i386: Нет такого файла или каталога
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-1 lab6-1.o
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ ./lab6-1
j
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$
```

Рис. 4.3: Запуск изначальной программы

Изменяю текст изначальной программы, убрав кавычки (рис. 4.4).

```
%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
```

Рис. 4.4: Измененная программа

На этот раз программа выдала пустую строчку, это связано с тем, что символ 10 означает переход на новую строку (рис. 4.5).

```
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ nasm -f elf lab6-1.asm
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-11 lab6-1.o
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ ./lab6-11


iekim@LINUXUBUNTU:~/work/arch-pc/lab06$
```

Рис. 4.5: Запуск измененной программы

Создаю новый файл для будущей программы и записываю в нее код из листинга (рис. 4.6).

```
%include 'in_out.asm'

SECTION .text
GLOBAL _start

_start:
    mov eax, '6'
    mov ebx, '4'
    add eax, ebx
    call iprintLF
    call quit
```

Рис. 4.6: Вторая программа

Создаю исполняемый файл и запускаю его, теперь отображается результат 106, программа, как и в первый раз, сложила коды символов, но вывела само число, а не его символ, благодаря замене функции вывода на iprintLF (рис. 4.7).

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

Рис. 4.7: Вывод второй программы

Убрав кавычки в программе, я снова ее запускаю и получаю предполагаемый изначально результат. (рис. 4.8).

Рис. 4.8: Вывод измененной второй программы

Заменив функцию вывода на iprint, я получаю тот же результат, но без переноса строки (рис. 4.9).

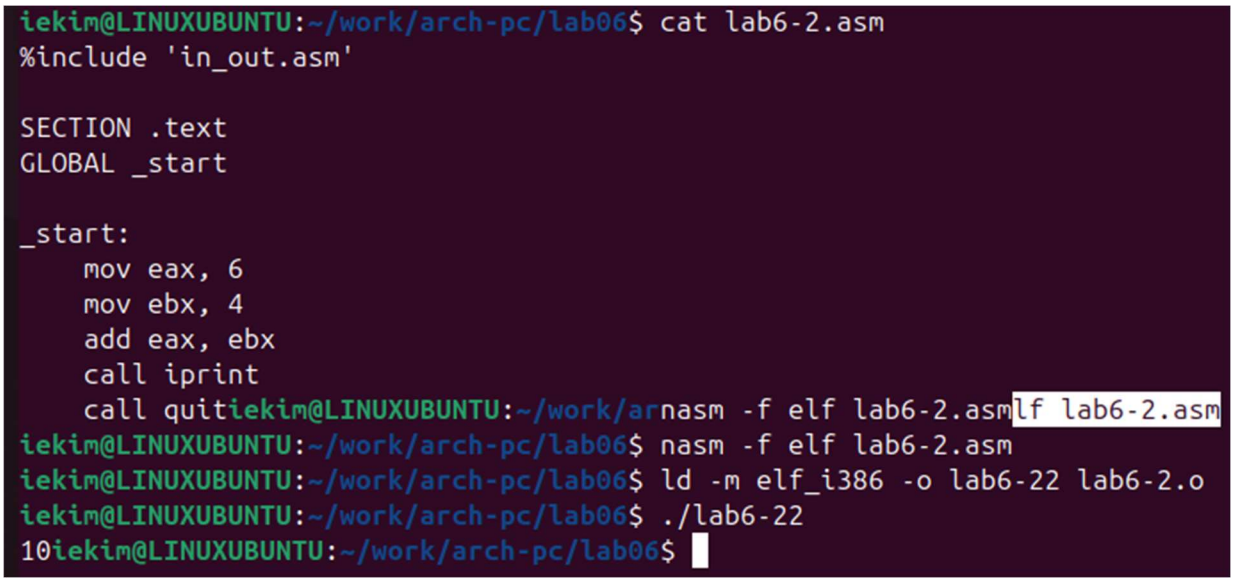

Рис. 4.9: Замена функции вывода во второй программе

4.2. Выполнение арифметических операций в NASM.

Создаю новый файл и копирую в него содержимое листинга (рис. 4.10).

```
%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 sprintdd
    mov eax, edx
    call iprintLF

    call quit
```

Рис. 4.10: Третья программа

Программа выполняет арифметические вычисления, на вывод идет результирующее выражения и его остаток от деления (рис. 4.11).

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

Рис. 4.11: Запуск третьей программы

Заменив переменные в программе для выражения f(x) = (4*6+2)/5 (рис. 4.12).

```
%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 sprintdd
    mov eax, edx
    call iprintLF

    call quit
```

Рис. 4.12: Изменение третьей программы

Запуск программы дает корректный результат (рис. 4.13).

```
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ nasm -f elf lab6-3.asm
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-31 lab6-3.o
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ ./lab6-31
Результат: 5
Остаток от деления: Остаток от деления: 1
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$
```

Рис. 4.13: Запуск измененной третьей программы

Создаю новый файл и помещаю текст из листинга (рис. 4.14).

```
%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. 4.14: Программа для подсчета варианта

Запустив программу и указав свой номер студенческого билета, я получил свой вариант для дальнейшей работы. (рис. 4.15).

```
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ nasm -f elf variants.asm
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ ld -m elf_i386 -o variants variants.o
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ ./variants
Введите № студенческого билета:
1032253496
Ваш вариант: 17
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$
```

Рис. 4.15: Запуск программы для подсчета варианта

4.3. Ответы на контрольные 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

4.4. Задание для самостоятельной работы

В соответсвии с выбранным вариантом, я реализую программу для подсчета функции f(x) = 10+(31x-5), проверка на нескольких переменных показывает корректное выполнение программы (рис. 4.16).

```
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ cat lab6-4.asm
%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:
    mov eax, msg
    call sprint

    mov ecx, x
    mov edx, 80
    call sread

    mov eax, x
    call atoi

    mov ebx, 31
    mul ebx
    sub eax, 5
    add eax, 10

    mov edi, eax

    mov eax, rem
    call sprint

    mov eax, edi
    call iprint

    call quitiekim@LINUXUBUNTU:~/work/arch-pc/lab06$ nasm -f elf lab6-4.asm
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ ld -m elf_i386 -o lab6-4 lab6-4.o
iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ ./lab6-4
Введите значение переменной x: 3
Результат: 98iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ ./lab6-4
Введите значение переменной x: 2
Результат: 67iekim@LINUXUBUNTU:~/work/arch-pc/lab06$ ./lab6-4
Введите значение переменной x: 1
Результат: 36iekim@LINUXUBUNTU:~/work/arch-pc/lab06$
```

Рис. 4.16: Запуск и проверка программы

Прилагаю код своей программы:

```
%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:
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
mov ebx, 31
mul ebx
sub eax, 5
add eax, 10
mov edi, eax
mov eax, rem
call sprint
mov eax, edi
call iprint
```

5. Выводы

При выполнении данной лабораторной работы я освоил арифметические инструкции языка ассемблера NASM.

6. Список литературы

1. Пример выполнения лабораторной работы
2. Курс на ТУИС
3. Лабораторная работа №6
4. Программирование на языке ассемблера NASM Столяров А. В.