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: Мурылев Иван Валерьевич

Группа: НПИбд-03-25

МОСКВА

2025 г.

Содержание

Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

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 ячейки памяти, над содержимым которой требуется выполнить операцию.

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>

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

sub <операнд_1>, <операнд_2>

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>

dec <операнд>

Вычитание:

Еще одна команда, которую можно отнести к арифметическим командам это команда изменения знака neg:

neg <операнд>

Для беззнакового умножения используется команда mul (от англ. multiply умножение):

mul <операнд>

Для знакового умножения используется команда imul:

imul <операнд>

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

div <делитель> ; Беззнаковое деление

idiv <делитель> ; Знаковое деление

Выполнение работы

Лабараторая часть

0. Так как в структуре репозитория уже создана папка отчета, первый этап пропускается.

1.

Создадим файл lab06-1.asm и заполним его листингом:

```
[ivmurihlev@personal report]$ touch lab06-1.asm
```

Затем создаем исполняемый файл и запускаем:

```
[ivmurihlev@personal report]$ ld -m elf_i386 lab06-1.o -o lab06-1
[ivmurihlev@personal report]$ ./lab06-1
j
```

После смены кода он выводит:

```
ivmurihlev@personal report]$ nasm -f elf lab06-1.asm
ivmurihlev@personal report]$ ld -m elf_i386 lab06-1.o -o lab06-1
ivmurihlev@personal report]$ ./lab06-1
```

Символ соответствует таблице (переход на новую строку):

10	12	0x0A	1010	LF, \n

2.

То же самое для lab06-2.asm

```
[ivmurihlev@personal report]$ touch lab06-2.asm
[ivmurihlev@personal report]$ nasm -f elf lab06-2.asm
[ivmurihlev@personal report]$ ld -m elf_i386 lab06-2.o -o lab06-2
[ivmurihlev@personal report]$ ./lab06-2
10
```

Выводит десять.

После замены функции строчку не выводит.

```
[ivmurihlev@personal report]$ nasm -f elf lab06-2.asm
[ivmurihlev@personal report]$ ld -m elf_i386 lab06-2.o -o lab06-2
[ivmurihlev@personal report]$ ./lab06-2
10[ivmurihlev@personal report]$
```

3.

Тоже самое для lab06-3.asm:

```
[ivmurihlev@personal report]$ touch lab06-3.asm
[ivmurihlev@personal report]$ nasm -f elf lab06-3.asm
[ivmurihlev@personal report]$ ld -m elf_i386 lab06-3.o -o lab06-3
[ivmurihlev@personal report]$ ./lab06-3
Результат: 4
Остаток от деления: 1
```

Потом изменяем файл и проверяем:

```
[ivmurihlev@personal report]$ nasm -f elf lab06-3.asm
[ivmurihlev@personal report]$ ld -m elf_i386 lab06-3.o -o lab06-3
[ivmurihlev@personal report]$ ./lab06-3
Результат: 5
Остаток от деления: 1
```

26%5==1

26//5 == 5

Все верно.

4.

Создаем variant.asm:

```
[ivmurihlev@personal report]$ touch variant.asm
[ivmurihlev@personal report]$ nasm -f elf variant.asm
[ivmurihlev@personal report]$ ld -m elf_i386 variant.o -o variant
[ivmurihlev@personal report]$ ./variant
Введите № студенческого билета:
1032251966
Ваш вариант: 7
```

Мой вариант 7

1.Функция sread -читает ввод с консоли.

2.Atoi преобразует символы в число.

3.Эти строки вычисляют вариант:

```
xor edx,edx
mov ebx,20
div ebx
inc edx
```

4.inc edx , прибавляет 1 к значению edx.

5.Эти строки выводят вариант.Первая переносит значение edx в eax, затем iprintLF выводит eax.

```
mov  eax,edx
call iprintLF
```

Самостоятельная часть

Смотрим вариант 7 $5(x-1)^2$ 3 5

Потом файл samostoyatelnaya.asm заполняем, а затем запускаем исполнительный файл:

```
[ivmurihlev@personal report]$ ./samostaatelnaya
Введите x:
3
2
Ответ: 20
```

Как видно он работает (5(3-1)^2 = 20).*

Вывод

Были изучены особенности работы с операторами на языке ассамблер и важность размеров (операндов) регистров.