**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: Волчкова Елизавета Дмитриевна

Группа: НКАбд-01-24

**МОСКВА**

2024 г.

|  |  |  |
| --- | --- | --- |
| 1. | Цель работы | 3 |
| 2. | Теоретическое введение | 4 |
| 3. | Задания  для самостоятельной работы | 5 |
| 4. | Выполнение лабораторной и самостоятельной работ | 6 |
| 5. | Вывод | 14 |
| 6. | Список литературы. | 15 |
| . |  |  |

Цель работы.

Узнать и ознакомиться с арифметическими инструкциями языка ассемблера NASM.

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.

Задание для самостоятельной работы.

1. Написать программу вычисления выражения 𝑦 = 𝑓(𝑥).
2. Программа должна выводить выражение для вычисления, выводить запрос на ввод значения 𝑥, вычислять заданное выражение в зависимости от введенного 𝑥, выводить результат вычислений.
3. Вид функции 𝑓(𝑥) выбрать из таблицы 6.3 вариантов заданий в соответствии с номером полученным при выполнении лабораторной работы.
4. Создайте исполняемый файл и проверьте его работу для значений 𝑥1 и 𝑥2 из 6.3

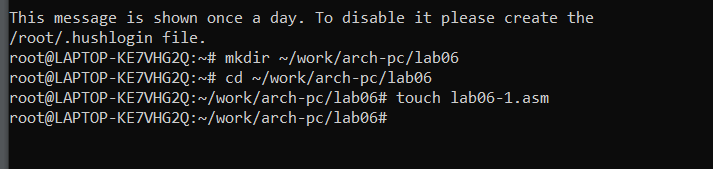
Выполнение лабораторной работы.

Создала каталог для программам лабораторной работы № 6, перешла в него и создала файл lab6-1.asm:

mkdir ~/work/arch-pc/lab06

cd ~/work/arch-pc/lab06

touch lab6-1.asm



2. Рассмотрела примеры программ вывода символьных и численных значений. Программы будут выводить значения записанные в регистр eax.

Затем ввела в файл lab6-1.asm текст программы из листинга 6.1. В данной программе в регистр eax записала символ 6 (mov eax,'6'), в регистр ebx символ 4 (mov ebx,'4').

Далее к значению в регистре eax прибавила значение регистра ebx (add eax,ebx, результат сложения запишется в регистр eax).

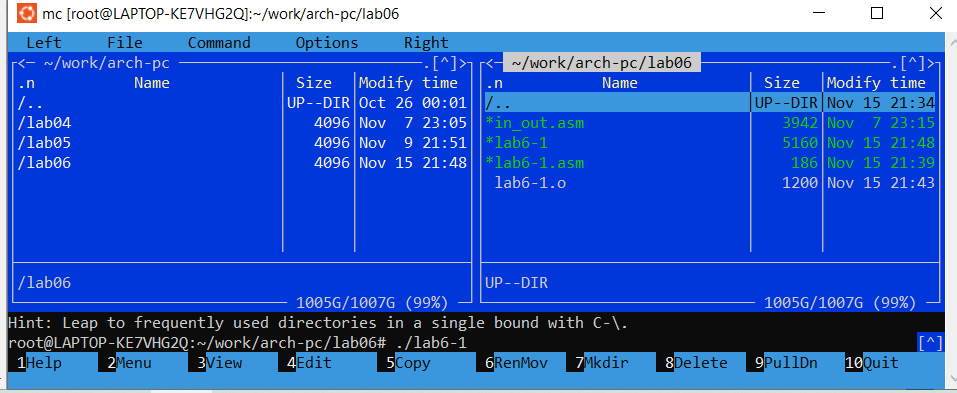
Потом вывела результат. Записала значение регистра eax в переменную buf1 (mov [buf1],eax), а потом адрес переменной buf1 в регистр eax (mov eax,buf1) и вызовем функцию sprintLF.

Ввела программу вывода значения регистра eax, создала исполняемый файл и запустила его.

nasm -f elf lab6-1.asm

ld -m elf\_i386 -o lab6-1 lab6-1.o

./lab6-1



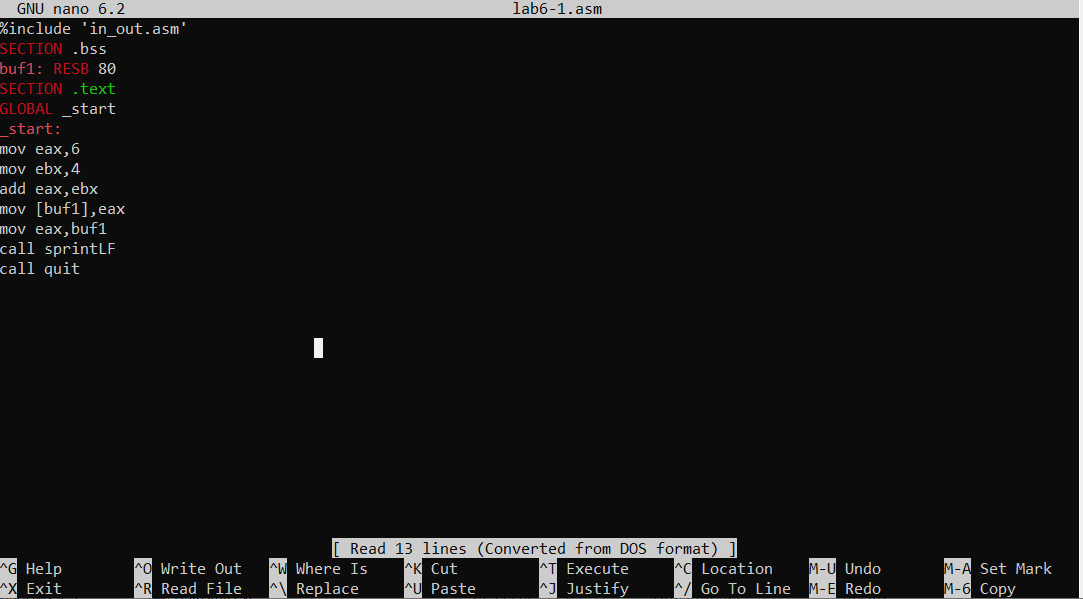
После изменила текст программы и вместо символов, записала в регистры числа. Исправила текст программы (Листинг 6.1) следующим образом: заменила строки

mov eax,'6'

mov ebx,'4' на строки

mov eax,6

mov ebx,4

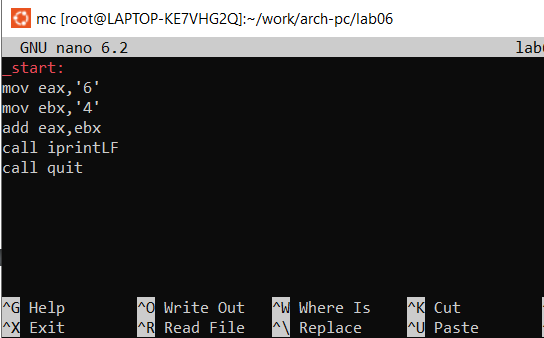


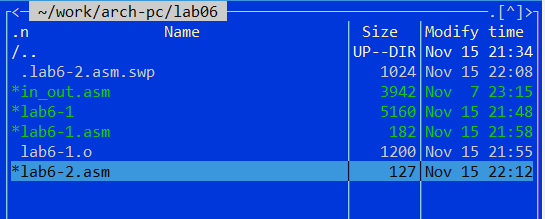
Создала исполняемый файл и запустила его.

Преобразовала текст программы из Листинга 6.1 с использованием этих функций. Создала файл lab6-2.asm в каталоге ~/work/arch-pc/lab06 и ввела в него текст программы из листинга 6.2.

touch ~/work/arch-pc/lab06/lab6-2.asm

Создала исполняемый файл и запустила его.

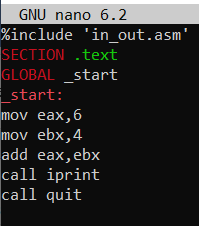




В результате работы программы я получила число 106

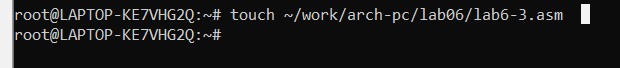
Аналогично предыдущему примеру изменила символы на числа. Потом заменила строки mov eax,'6' mov ebx,'4' на строки mov eax,6 mov ebx,4 Создала исполняемый файл и запустила его

Заменила функцию iprintLF на iprint, создайла исполняемый файл и запустила его.



Создала файл lab6-3.asm в каталоге ~/work/arch-pc/lab06:

touch ~/work/arch-pc/lab06/lab6-3.asm



Изучила текст программы из листинга 6.3 и ввела в lab6-3.asm. Создала исполняемый файл и запустила его.

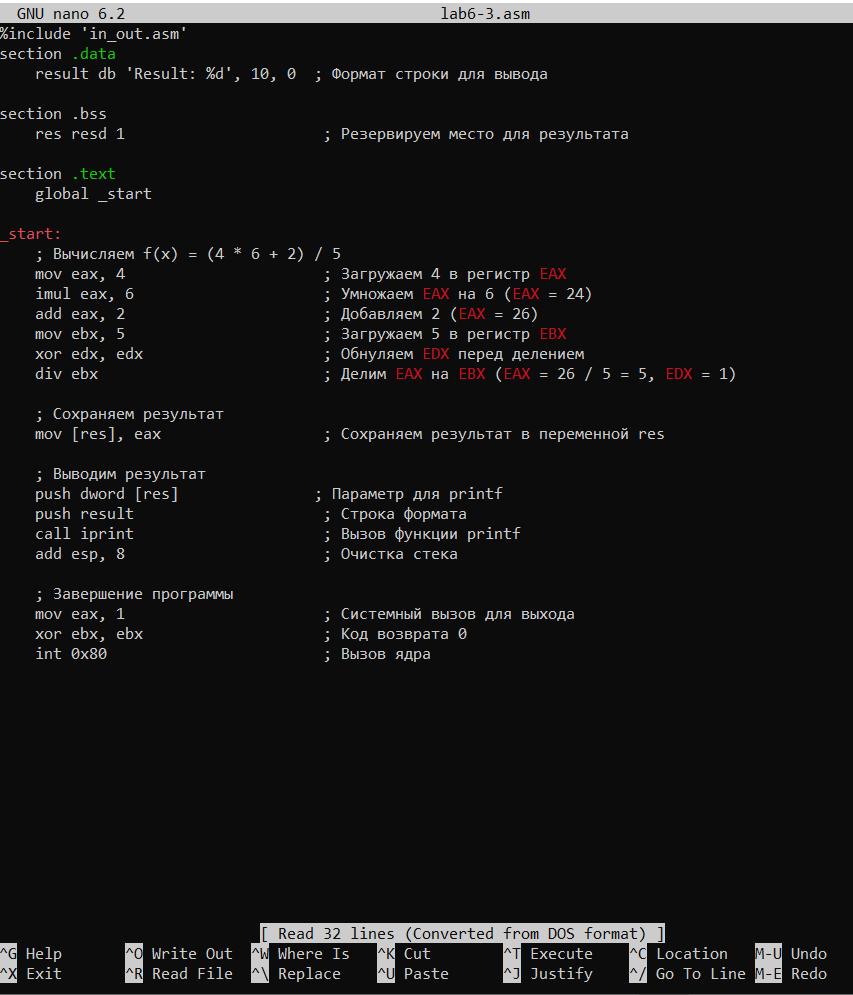
Результат работы программы должен быть следующим: user@dk4n31:~$ ./lab6-3

Результат: 4

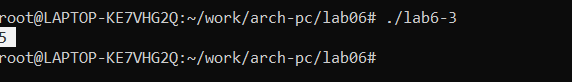
Остаток от деления: 1

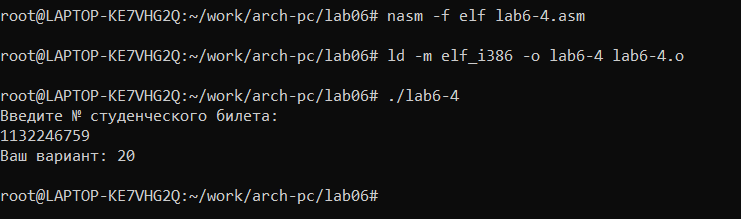
user@dk4n31:~$

Изменила текст программы для вычисления выражения



𝑓(𝑥) = (4 ∗ 6 + 2)/5. Создала исполняемый файл и проверила его работу. Ответ: 5!



Далее вывела запрос на введение № студенческого билета, потом вычислила номер варианта по формуле: 

(𝑆𝑛 mod 20) + 1, где 𝑆𝑛 – номер студенческого билета (В данном случае 𝑎 mod 𝑏 – это остаток от деления 𝑎 на 𝑏), потом вывела на экран номер варианта.

Создала файл variant.asm в каталоге ~/work/arch-pc/lab06:

touch ~/work/arch-pc/lab06/variant.asm

Изучила текст программы из листинга 6.4 и ввела в файл variant.asm.

Потом создла исполняемый файл и запустила его, далее проверила результат работы программы, вычислив номер варианта аналитически.

Ответы на вопросы:

1. Какие строки листинга 6.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’?

20

2. Для чего используется следующие инструкции?

mov ecx, x помещаем в регистр ecx число х

mov edx, 80 длина строки 80 символов

call sread считывает из строки длиной 80 символов значение переменной х

3. Для чего используется инструкция “call atoi”?

4. Какие строки листинга 6.4 отвечают за вычисления варианта?

5. В какой регистр записывается остаток от деления при выполнении инструкции “div ebx”?

При этом остаток от целочисленного деления помещается в регистр AH, DX или EDX соответственно.

6. Для чего используется инструкция “inc edx”? -Инструкция

inc edx используется для увеличения значения регистра EDX на 1

7. Какие строки листинга 6.4 отвечают за вывод на экран результата вычислений?

mov eax,edx

call iprintLF

Выполнение самостоятельной работы.

Написала программу вычисления выражения 𝑦 = 𝑓(𝑥).

(Программа должна выводить выражение для вычисления, выводить запрос на ввод значения 𝑥, вычислять заданное выражение в зависимости от введенного 𝑥, выводить результат вычислений.)

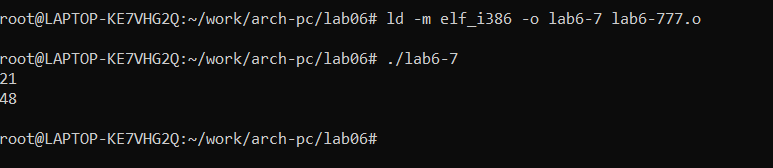


Вид функции 𝑓(𝑥) выбрала из таблицы 6.3 вариантов заданий в соответствии с номером полученным при выполнении лабораторной работы.



Затем создала исполняемый файл и проверьте его работу для значений 𝑥1 и 𝑥2 из 6.3.

𝑥1 = 1, ответ = 21 и 𝑥2 = 3, ответ = 48



Вывод.

Я узнала и ознакомилась с арифметическими инструкциями языка ассемблера NASM. Решая задания, я поняла, как работать с арифметическими действиями.

Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.

2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.

3. Midnight Commander Development Center. — 2021. — URL: https://midnight-commander. org/.

4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.

5. Newham C. Learning the bash Shell: Unix Shell Programming. — O’Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learningbash-Shell-Programming-Nutshell/dp/0596009658>.

6. Robbins A. Bash Pocket Reference. — O’Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.

7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.

8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.

9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.

10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.

11. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.

12. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.

13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВПетербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.

14. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: <http://www.stolyarov.info/books/asm_unix>.

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).

16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science