Отчёт по лабораторной работе №6

Архитектура компьютера

Алехин Давид Андреевич

Содержание

# 1 Цель работы

Освоение арифметических инструкций языка ассемблера NASM.

# 2 Задание

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

# 3 Теоретическое введение

**Адресация в NASM**

Большинство инструкций на языке ассемблера требуют обработки операндов. Адрес опе- ранда 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 ко- манде, Например: mov ax,2. • Адресация памяти – операнд задает адрес в памяти. В команде указывается символи- ческое 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 анало- гично 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 вычита- ния 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 отличии 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 нахо- диться 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.

**Команды деления div и idiv**

Для деления, как и для умножения, существует 2 команды div (от англ. divide - деление) и idiv: div ; Беззнаковое деление idiv ; Знаковое деление В командах указывается только один операнд – делитель, который может быть регистром или ячейкой памяти, но не может быть непосредственным операндом. Местоположение делимого и результата для команд деления зависит от размера делителя. Кроме того, так как в результате деления получается два числа – частное и остаток, то эти числа помещаются в определённые регистры.

**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). По- этому, 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 дан- ные 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 записает результат в регистр eax, перед вызовом atoi в регистр eax необходимо записать число (mov eax,).

# 4 Выполнение лабораторной работы

Создаю папкую lab06 и в ней файл lab6-1.asm. (рис. 1).

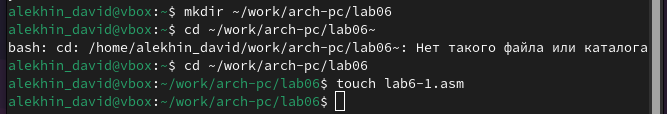


Рис. 1: lab06/lab6-1.asm

Ввожу в lab6-1.asm текст программы 6.1. Программа вывода значения регистра eax. (рис. 2).

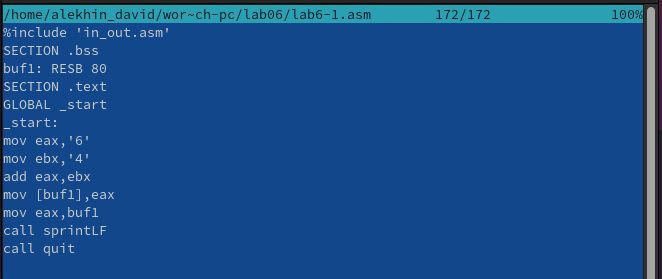


Рис. 2: lab6-1.asm

Переношу файл in\_out.asm в папку lab06. (рис. 3).

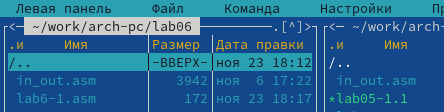


Рис. 3: lab06/in\_out.asm

Компеллирую и запускаю файл lab6-1.asm. (рис. 4).

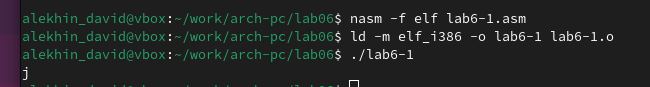


Рис. 4: Запуск lab6-1.asm

Меняю символы на числовые значения. (рис. 5).

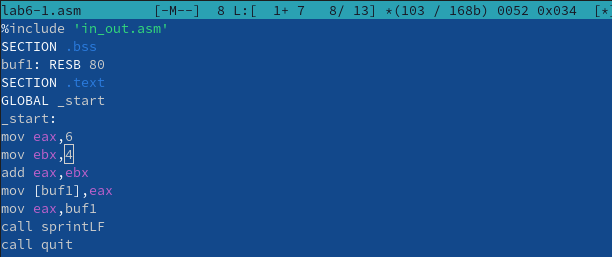


Рис. 5: Изменённый lab6-1.asm

Компеллирую и запускаю исправленный файл. В результате получается невидимвый символ. (рис. 6).

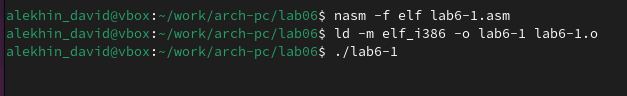


Рис. 6: Запуск изменённого lab6-1.asm

Создаю файл lab6-2.asm и вписываю туда текст команды 6.2. Программа вывода значения регистра eax. (рис. 7).

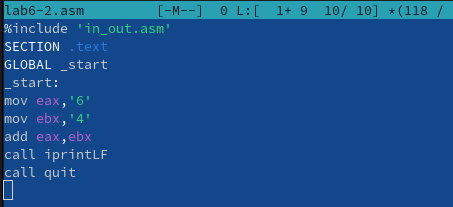


Рис. 7: Код lab6-2.asm

Запускаю файл lab6-2.asm. (рис. 8).

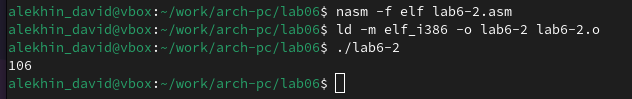


Рис. 8: Запуск lab6-2.asm

Меняю символы на числовые значения. (рис. 9).

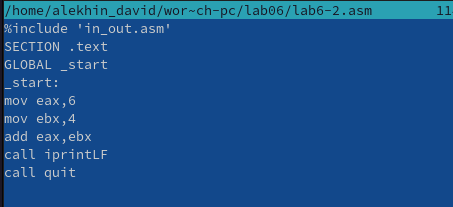


Рис. 9: Изменённый lab6-2.asm

Компеллирую и запускаю исправленный файл. Врезультате получаем 10. (рис. 10).

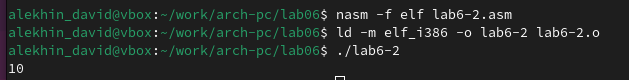


Рис. 10: Запуск изменённого lab6-2.asm

Меняю print на printLF и запускаю. Различие в том что print не переносит на следующую строку. (рис. 11).

Рис. 11: print на printLF

Рис. 11: print на printLF

Создаю файл lab6-3.asm и вписываю туда текст команды 6.3. Программа вычисления выражения. (рис. 12).

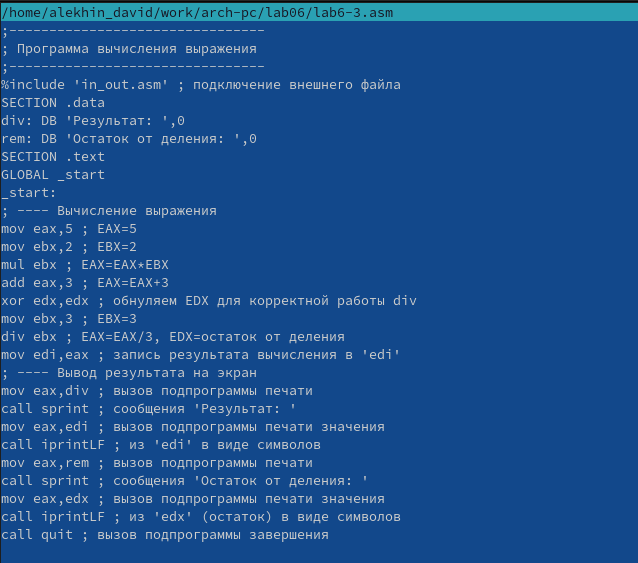


Рис. 12: Текст lab6-3.asm

Запускаю lab6-3.asm. (рис. 13).

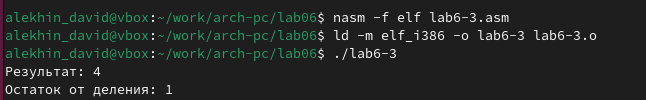


Рис. 13: Запуск lab6-3.asm

Меняю значения переменных в lab6-3.asm. (рис. 14).

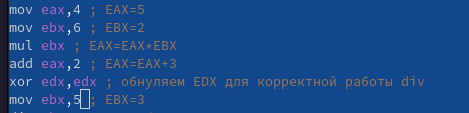


Рис. 14: Текст изменённого lab6-3.asm

Запускаю изменённый файл. (рис. 15).

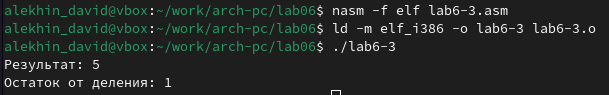


Рис. 15: Запуск изменённого lab6-3.asm

Создаю файл variant.asm и вписываю текст команды 6.4. Программа вычисления вычисления варианта задания по номеру студенческого билета. (рис. 16).

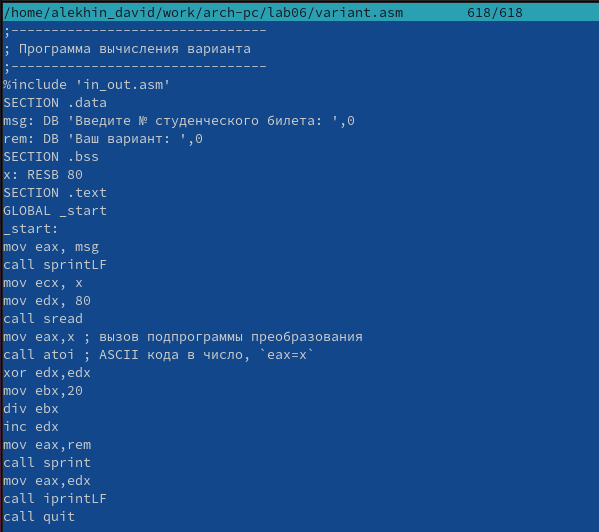


Рис. 16: Текст variant.asm

Запускаю команду, вписываю свой студ билет, получаю 1. (рис. 17).

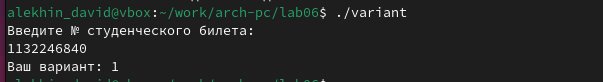


Рис. 17: Запуск variant.asm

Ответ на вопросы: 1)Какие строки листинга 7.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’? mov eax,msg call sprintLF 2) Для чего используется следующие инструкции? nasm mov ecx, x mov edx, 80 call sread Эти инструкции используются для ввода переменной Х с клавиатуры и сохранения введенных данных. 3) Для чего используется инструкция “call atoi”? Эта инструкция используется для преобразования Кода переменной ASCII в число. 4) Какие строки листинга 7.4 отвечают за вычисления варианта? mov ebx,20 div ebx inc edx 5)В какой регистр записывается остаток от деления при выполнении инструкции “div ebx”? В регистре ebx. 6) Для чего используется инструкция “inc edx”? Для увеличения значения edx на 1. 7)Какие строки листинга 7.4 отвечают за вывод на экран результата вычислений? mov eax,edx call iprintLF

Приступаю к самостоятельному заданию. Создаю файл lab6-1s.asm и вписываю туда текст комманды для выполнения свмостоятельного задания. (рис. 18).

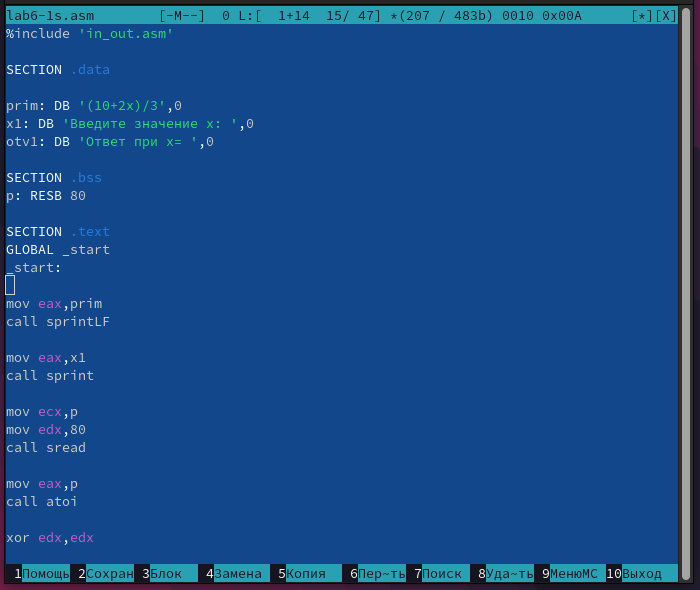


Рис. 18: lab6-1s.asm

Компеллирую файл и запускаю вписывая разные значения х. (рис. 19).

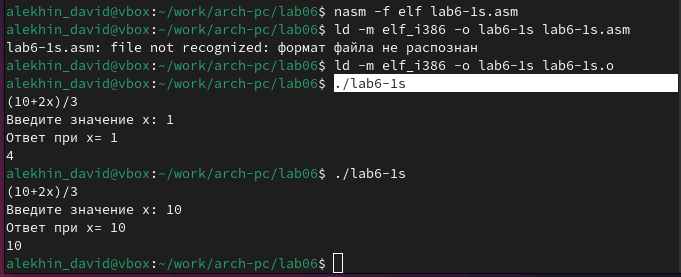


Рис. 19: Запуск lab6-1s.asm

# 5 Выводы

Выполнив лабораторную я изучил арифметические инструкции языкв nasm.

# Список литературы

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/Learning- bash-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. — БХВ- Петербург, 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).