Отчёта по лабораторной работе 6

Освоение арифметических инструкций языка ассемблера NASM.

Мельников Максим

Содержание

# 1 Цель работы

Целью работы является освоение арифметических инструкций языка ассемблера NASM.

# 2 Задание

1. Изучите примеры программ.
2. Напишите программу вычисления выражения в соответсвии с вариантом.
3. Загрузите файлы на GitHub.

# 3 Теоретическое введение

В основном наборе инструкций входят разные вариации четырех арифметических действий: сложение, вычитание, умножение, деление. Важно помнить, что в результате арифметических действий меняются некоторые биты регистра флагов, что позволяет выполнять команду условного перехода, т.е. разветвлять программу на основе результат операции. Замечу, что для команд с ложения и вычитания справедливыми являются отмеченное выше для операндов команды mov . К командам сложения можно отнести: add – обычное сложение, adc – сложение с добавлением результату флага переноса в качестве единицы (если флаг равен нулю, то команда эквивалентна команде add ), xadd – сложение, с предварительным обменом данных между операндами, inc – прибавление единицы к содержимому операнда. Несколько примеров: add %rbx , dt (или addq, dt, где четко указано, что складываются 64-битовые величины) – к содержимому области памяти dt добавляется содержимое регистра rbx и результат помещается в dt ; adc %rdx , %rdx – удвоение содержимого регистра rdx плюс добавление значения флага переноса; incl ll – увеличение на единицу содержимого памяти по адресу ll. При этом явно указывается, что операнд имеет размер 32 бита (d - dword).

К командам вычитания можно отнести следующие инструкции процессора x86-64: sub – обычное вычитание, sbb - вычитание из результата флага переноса в качестве единицы (если флаг равен нулю, то команда эквивалентна sub ), dec – вычитание единицы из результата, neg – вычитание значения операнда из 0 . Несколько примеров: sub %rax , ll - из содержимого ll вычитается содержимое регистра rax (или явно subq %rax , ll, где указывается, что операнды имеют 64-размер), и результат помещается в ll; subw go, %ax – вычитание из содержимого ax числа по адресу go, результат помещается в ax ; sbb %rdx , %rax – вычитание с дополнительным вычитанием флага переноса (из числа в rax вычитается число в rdx и результат в rax); decb l – вычитание единицы из байта, расположенного по адресу l . Следует отметить еще специальную команду cmp , которая во всем похожа на команду sub, кроме одного – результат вычитания никуда не помещается. Инструкция используется специально, для сравнения операндов.

Две основные команды умножения: mul – умножение беззнаковых чисел, imul – умножение знаковых чисел. Команда содержит один операнд – регистр или адрес памяти. В зависимости от размера операнда данные помещаются: в ax , dx : ax , edx : eax , rdx : rax . Например: mull ll – содержимое памяти с адресом ll будет умножено на содержимое eax (не забываем о суффиксе l), а результат отправлен в пару регистров edx : eax; mul %dl – умножить содержимое регистра dl на содержимое регистра al , а результат положить в ax ; mul %r8 – умножить содержимое регистра r8 на содержимое регистра rax , а результат положить в пару регистров rdx : rax.

Для деления (целого) также предусмотрены две команды: div – беззнаковое деление, idiv – знаковое деление. Инструкция также имеет один операнд - делитель. В зависимости от его размера результат помещается: al – результат деления, ah – остаток от деления; ax – результат деления, dx – остаток от деления; eax – результат деления, edx – остаток от деления; rax – результат деления, rdx – остаток от деления. Приведем примеры: divl dv – содержимое edx : eax делится на делитель, находящийся в памяти по адресу dv и результат деления помещается в eax , остаток в edx ; div %rsi – содержимое rdx : rax делится на содержимое rsi , результат помещается в rax , остаток в rdx .

# 4 Выполнение лабораторной работы

1. Создайте каталог для программам лабораторной работы № 6, перейдите в него и создайте файл lab7-1.asm:
2. Рассмотрим примеры программ вывода символьных и численных значений. Программы будут выводить значения, записанные в регистр eax. (рис. 1, 2)

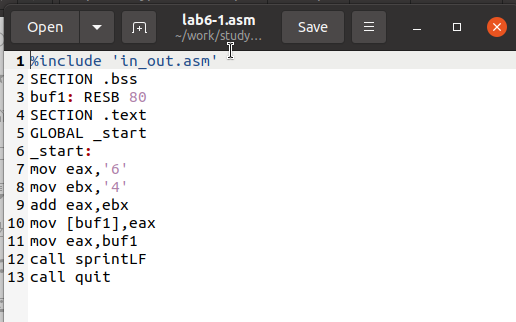


Рис. 1: Пример программы

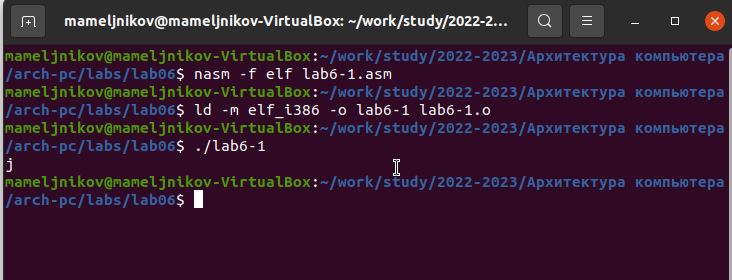


Рис. 2: Работа программы

1. Далее изменим текст программы и вместо символов, запишем в регистры числа. Исправьте текст программы (Листинг 1) следующим образом: (рис. 3, 4)

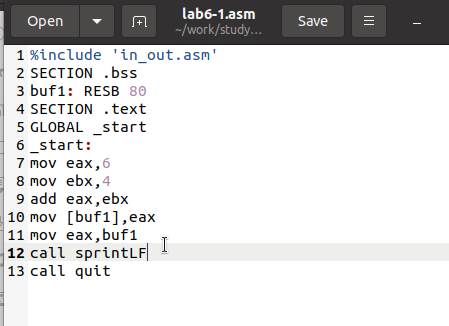


Рис. 3: Пример программы

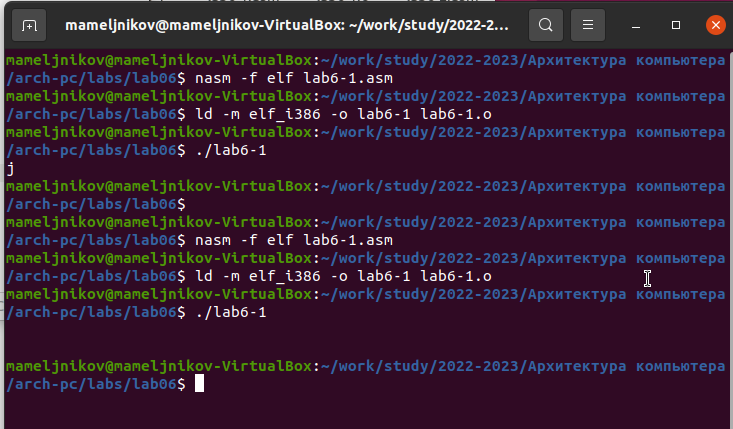


Рис. 4: Работа программы

Никакой символ не виден, но он есть. Это возврат каретки LF.

1. Как отмечалось выше,для работы с числами в файле in\_out.asm реализованы подпрограммы для преобразования ASCII символов в числа и обратно. Преобразуем текст программы из Листинга 7.1 с использованием этих функций. (рис. 5, 6)

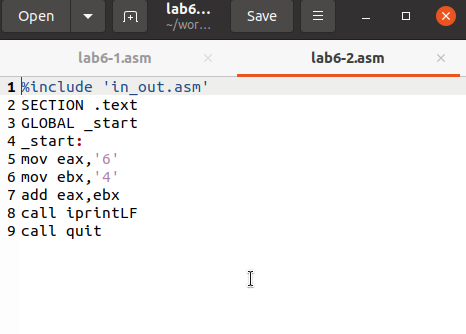


Рис. 5: Пример программы

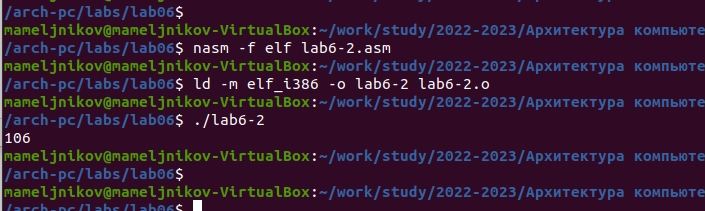


Рис. 6: Работа программы

В результате работы программы мы получим число 106. В данном случае, как и в первом, команда add складывает коды символов ‘6’ и ‘4’ (54+52=106). Однако, в отличии от программы из листинга 7.1, функция iprintLF позволяет вывести число, а не символ, кодом которого является это число.

1. Аналогично предыдущему примеру изменим символы на числа. (рис. 7, 8)

Создайте исполняемый файл и запустите его. Какой результат будет получен при исполнении программы? – получили число 10

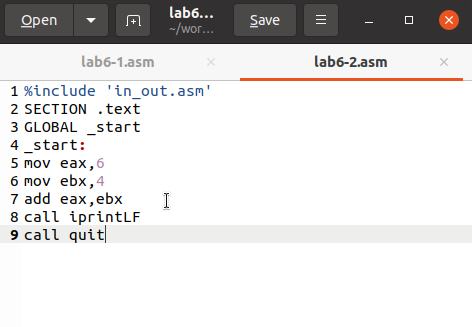


Рис. 7: Пример программы

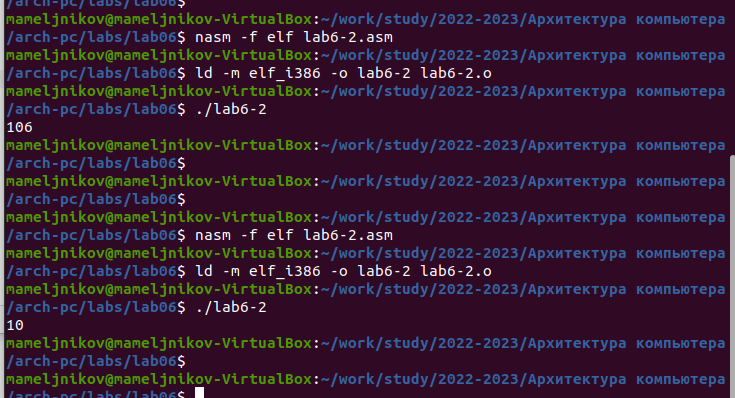


Рис. 8: Работа программы

Замените функцию iprintLF на iprint. Создайте исполняемый файл и запустите его. Чем отличается вывод функций iprintLF и iprint? - Вывод отличается что нет переноса строки. (рис. 9)

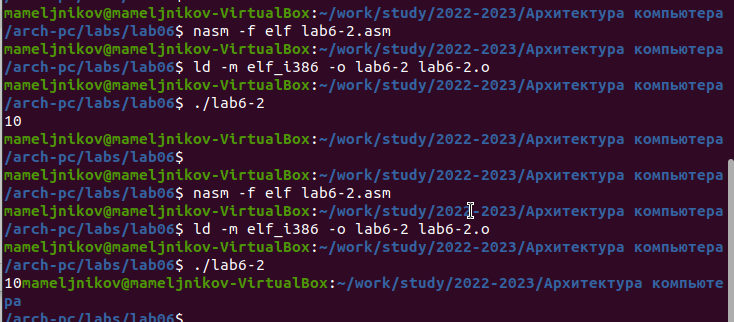


Рис. 9: Работа программы

1. В качестве примера выполнения арифметических операций в NASM приведем программу вычисления арифметического выражения

* . (рис. 10, рис. 11)

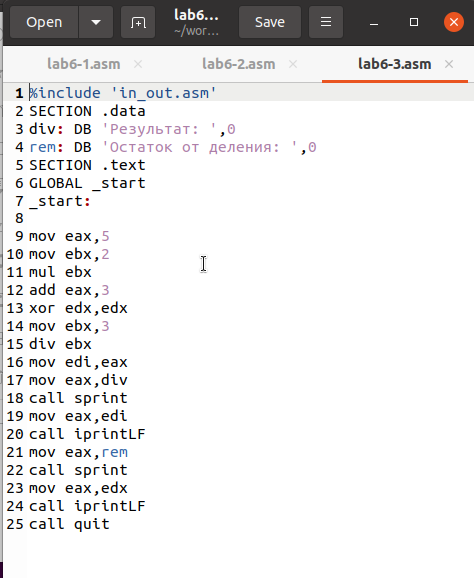


Рис. 10: Пример программы

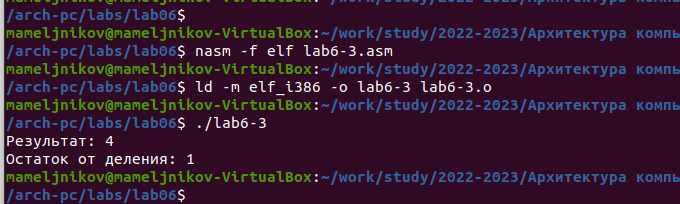


Рис. 11: Работа программы

Измените текст программы для вычисления выражения

. Создайте исполняемый файл и проверьте его работу. (рис. 12, рис. 13)

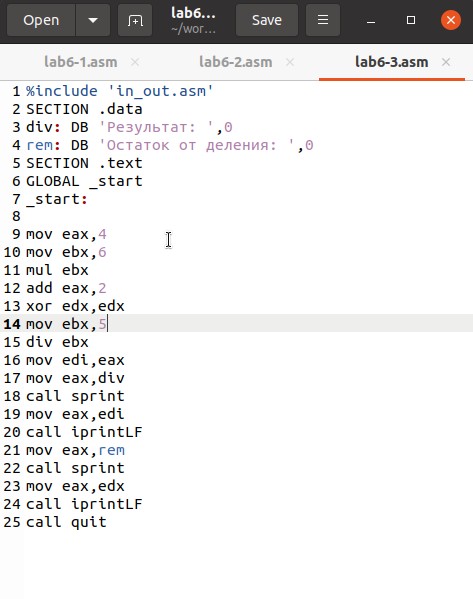


Рис. 12: Пример программы

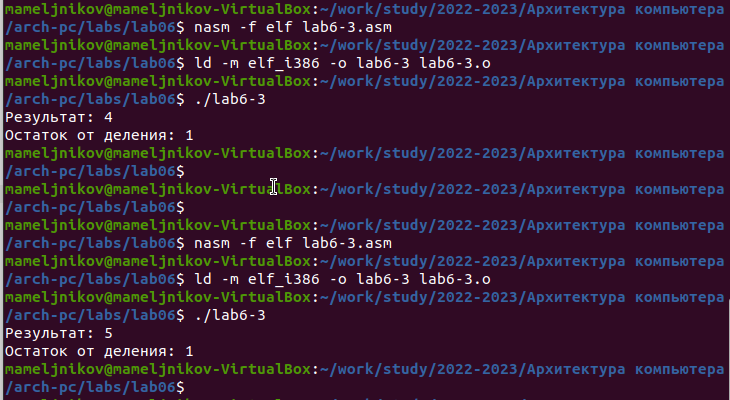


Рис. 13: Работа программы

1. В качестве другого примера рассмотрим программу вычисления варианта задания по номеру студенческого билета, работающую по следующему алгоритму: (рис. 14, рис. 15)

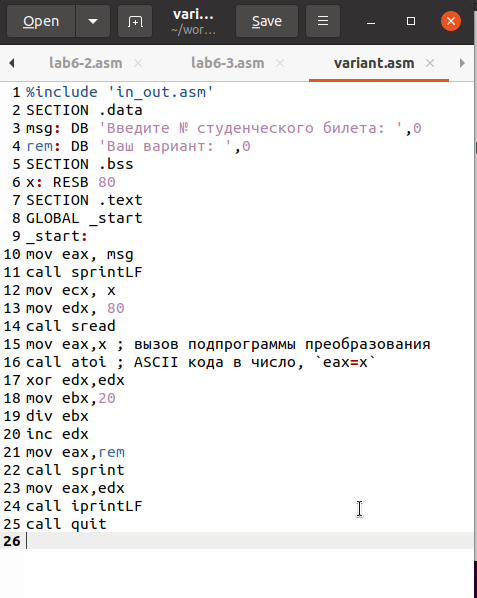


Рис. 14: Пример программы

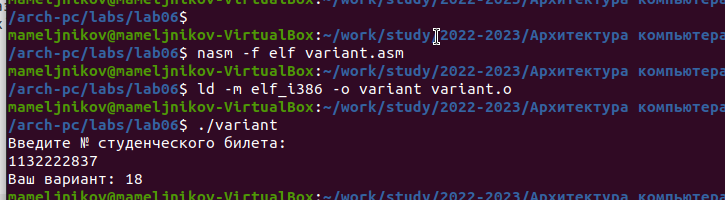


Рис. 15: Работа программы

* Какие строки листинга 7.4 отвечают за вывод на экран сообщения ‘Ваш вариант:’? – mov eax,rem – перекладывает в регистр значение переменной с фразой ‘Ваш вариант:’ call sprint – вызов подпрограммы вывода строки
* Для чего используется следующие инструкции? nasm mov ecx, x mov edx, 80 call sread

Считывает значение студбилета в переменную Х из консоли

* Для чего используется инструкция “call atoi”? - эта подпрограмма переводит введенные символы в числовой формат
* Какие строки листинга 7.4 отвечают за вычисления варианта? xor edx,edx mov ebx,20 div ebx
* В какой регистр записывается остаток от деления при выполнении инструкции “div ebx”? 1 байт AH 2 байта DX 4 байта EDX – наш случай
* Для чего используется инструкция “inc edx”? по формуле вычисления варианта нужно прибавить единицу
* Какие строки листинга 7.4 отвечают за вывод на экран результата вычислений mov eax,edx – результат перекладывается в регистр eax call iprintLF – вызов подпрограммы вывода

1. Написать 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) выбрать из таблицы 6.3 вариантов заданий в соответствии с номером полученным при выполнении лабораторной работы. Создайте исполняемый файл и проверьте его работу для значений x1 и x2 из 6.3. (рис. 16, рис. 17)

Получили вариант 18 -

для х=1 и 5

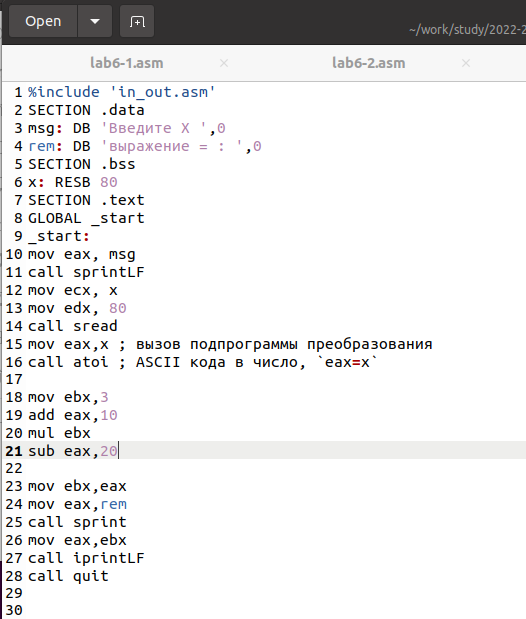


Рис. 16: Пример программы

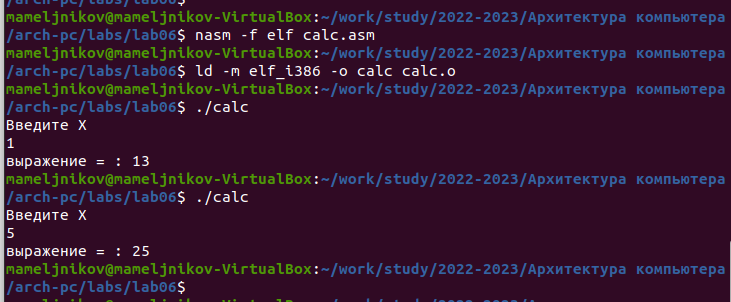


Рис. 17: Работа программы

# 5 Выводы

Изучили работу с арифметическими операциями

# Список литературы

1. [Расширенный ассемблер: NASM](https://www.opennet.ru/docs/RUS/nasm/)
2. [MASM, TASM, FASM, NASM под Windows и Linux](https://habr.com/ru/post/326078/)