Лабораторная работа №6

Арифметические операции в NASM

Борисенкова София Павловна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Познакомиться с базовыми инструкциями языка Ассемблер, отвечающими за основные арифметические операции

2 Выполнение лабораторной работы

Для начала выполнения лабораторной работы необходимо создать папку рабочего каталога и файл lab6-1.asm (Рис. 2.1):

Создание рабочей директории и файла lab6-1.nasm для записи кода на языке Ассемблера

Рис. 2.1: Создание рабочей директории и файла lab6-1.nasm для записи кода на языке Ассемблера

После этого, для более комфортной работы, запустим Midnight commander (Рис. 2.2):

Запуск Midnight commander

Рис. 2.2: Запуск Midnight commander

Вставим в наш созданный файл код из листинга 6.1 с помощью команды F4 (редактор) в MC (Рис. 2.3):

Запись кода из листинга в файл lab6-1.asm

Рис. 2.3: Запись кода из листинга в файл lab6-1.asm

Перед сборкой файла стоит учесть, что он использует сторонний файл in_out.asm. С помощью команды F5 скопируем его из каталога пятой лабораторной работы (Рис. 2.4):

Копирование файла in_out.asm в рабочую директорию

Рис. 2.4: Копирование файла in_out.asm в рабочую директорию

Так будет выглядеть наша рабочая директория (Рис. 2.5):

Вид каталога после перенесения файла in_out.asm

Рис. 2.5: Вид каталога после перенесения файла in_out.asm

Теперь соберём наш файл в исполняемое приложение уже знакомыми инструментами, nasm и ld (Рис. 2.6):

Сборка исполняемого файла из lab6-1.asm

Рис. 2.6: Сборка исполняемого файла из lab6-1.asm

Теперь запустим файл и посмотрим на результат (Рис. 2.7):

Запуск исполняемого файла и результат вывода

Рис. 2.7: Запуск исполняемого файла и результат вывода

Нам выводит символ j, однако это неправильный вывод. Наша цель - сложить 6 и 4, и получить в выводе число 10. Попробуем изменить наш файл (Рис. 2.8):

Редактирование файла

Рис. 2.8: Редактирование файла

Мы убрали кавычки у цифр, и теперь мы складываем уже не символы “6” и “4” (когда мы складываем символы, мы складываем их коды ASCII), а числа. Теперь попробуем собрать исполняемый исполняемый файл также, как собирали до этого, и запустим (Рис. 2.9):

Запуск исполняемого файла и результат вывода

Рис. 2.9: Запуск исполняемого файла и результат вывода

Мы видим, что ничего не вывелось. Но так ли это? Когда мы вызываем команду sprintLF, она выводит не число 10, а символ с номером 10. Посмотрим на таблицу ASCII и увидим, что символ под номером 10 это символ перевода строки. Именно поэтому мы его не видим, мы видим просто новую строку. Теперь создадим второй файл под названием lab6-2.asm (Рис. 2.10):

Создание второго файла: lab6-2.asm

Рис. 2.10: Создание второго файла: lab6-2.asm

Теперь вставим в него код из листинга 6.2 (Рис. 2.11):

Запись кода из листинга в файл lab6-2.asm

Рис. 2.11: Запись кода из листинга в файл lab6-2.asm

Как мы видим, основное отличие заключается в том, что вместо sprintLF используется iprintLF. Соберём файл и запустим его, чтобы посмотреть, как изменится вывод (Рис. 2.12):

Запуск исполняемого файла и результат вывода

Рис. 2.12: Запуск исполняемого файла и результат вывода

Мы видим число 106. Так как цифры в коде указаны в кавычках, мы складываем их коды (54 и 52 в сумме дают 106). Теперь программа способна вывести число, а не символ ASCII с соответствующим номером. Теперь, если мы уберём кавычки у цифр, программа должна вывести 10. Убедимся в этом, сделав соответствующие изменения в коде (Рис. 2.13):

Изменение файла lab6-2.asm

Рис. 2.13: Изменение файла lab6-2.asm

Соберём программу и запустим её (Рис. 2.14):

Сборка исполняемого файла и результат работы программы

Рис. 2.14: Сборка исполняемого файла и результат работы программы

Как видим, программа действительно вывела число 10. Кроме операции iprintLF в файле in_out.asm есть операция iprint. Посмотрим, чем они отличаются. Заменим в коде iprintLF на iprint (Рис. 2.15):

Редактирование файла lab6-2.asm

Рис. 2.15: Редактирование файла lab6-2.asm

Попробуем собрать программу и запустить её (Рис. 2.16):

Сборка и результат работы отредактированного файла

Рис. 2.16: Сборка и результат работы отредактированного файла

Как видим, операция iprint не переносит на следующую строку, в отличие от iprintLF. В этом их разница. Теперь создадим третий файл (Рис. 2.17):

Создание третьего файла: lab6-3.asm

Рис. 2.17: Создание третьего файла: lab6-3.asm

Он должен выводить значение функции (5*2+3)/3. Для этого вставим код из файла листинга 6.3 (Рис. 2.18):

Вставка кода из листинга в созданный ранее файл

Рис. 2.18: Вставка кода из листинга в созданный ранее файл

Попробуем запустить этот программу, предварительно её собрав (Рис. 2.19):

Сборка файла lab6-3.asm и результат его работы

Рис. 2.19: Сборка файла lab6-3.asm и результат его работы

Полученный результат совпадает с результатом, указанным в лабораторной работе. Теперь изменим файл так, чтобы он вычислял значение выражения (4*6+2)/5. Для этого в коде заменим число 5 на 4, число 2 на 6, число 3 на 2, и второе число 3 на 5 (Рис. 2.20):

Редактирование файла lab6-3.asm

Рис. 2.20: Редактирование файла lab6-3.asm

Соберём программу и запустим её (Рис. 2.21):

Повторная сборка уже изменённого файла lab6-3.asm и результат его работы

Рис. 2.21: Повторная сборка уже изменённого файла lab6-3.asm и результат его работы

Пересчитав значение выражения вручную, убеждаемся, что вывод корректный. Теперь создадим файл variant.asm для вычисления варианта самостоятельной работы (Рис. 2.22):

Создание файла variant.asm для вычисления варианта для самостоятельной работы

Рис. 2.22: Создание файла variant.asm для вычисления варианта для самостоятельной работы

Вставим в файл код из листинга 6.4, который вычисляет номер варианта по формуле (s mod 20) + 1, где s - номер студенческого билета (Рис. 2.23):

Вставка кода из листинга в файл variant.asm

Рис. 2.23: Вставка кода из листинга в файл variant.asm

Соберём и запустим программу, указав номер студенческого билета. В моём случае это 1132243816 (Рис. 2.24):

Сборка и запуск программы, а также результат выполнения

Рис. 2.24: Сборка и запуск программы, а также результат выполнения

Программа вывела число 17. Действительно, ведь остаток от деления числа 1132243816 на 20 равен 16. 16 + 1 = 17, соответственно.

Разберём работу кода, ответив на предложенные в лабораторной работе вопросы:

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

За это отвечает 25-ая строчка call sprint, перед которой идёт строка mov eax,rem, которая перемещает строку с фразой в регистр eax, из которого мы считываем данные для вывода

2. Для чего используется следующие инструкции?

mov ecx, x

mov edx, 80

call sread

Эти строки используются для того, чтобы записать данные в переменную x

3. Для чего используется инструкция “call atoi”?

Для преобразования ASCII кода в число

4. Какие строки листинга 6.4 отвечают за вычисления варианта?

Напрямую за вычисление отвечают следующие строки:

div ebx

inc edx

Первая делит число x в регистре eax на значение регистра ebx (в нашем случае 20), а вторая прибавляет к значению регистра edx (куда сохранился остаток от деления в прошлой операции) единицу

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

Как уже было сказано в ответе на предыдущий вопрос, в регистр edx

6. Для чего используется инструкция “inc edx”?

Для увеличения значения регистра edx на единицу

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

За это отвечают строки:

mov eax,edx

call iprintLF

первая строка переносит значение регистра edx в eax, а вторая вызывает операцию вывода значения регистра eax на экран

3 Выполнение задания для самостоятельной работы

Теперь в качестве самостоятельной работы напишем код программы для вычисления выражения в варианте 17: 18(+ 1)/6 . В предварительно созданном файле Task17.asm впишем следующий код (Рис. 3.1):

Код требуемой программы

Рис. 3.1: Код требуемой программы

Он немного отличается от предыдущих программ. За вычисление выражения отвечают следующие команды:

add eax,1

mul ebx

div ebx

Где первая отвечает за прибавление к регистру eax единицы.

Вторая отвечает за умножение значения регистра eax (где теперь находится x+1) на значение регистра ebx (куда записано значение 18).

Третья отвечает за деление значения регистра eax (где теперь находится 18*(x+1)) на значение регистра ebx (которое мы перезаписали на 6).

Попробуем собрать нашу программу (Рис. 3.2):

Сборка исполняемого файла

Рис. 3.2: Сборка исполняемого файла

И запустим код, указав в качестве x предложенные в лабораторной работе значения (Рис. 3.3):

Запуск программы и проверка её корректной работы

Рис. 3.3: Запуск программы и проверка её корректной работы

Как видим, программа выводит правильные значения выражения.

4 Выводы

В результате выполнения лабораторной работы было получено представление о том, какие арифметические операции есть в языке Ассемблера, и как они работают. Были написаны программы, использующие в себе операции сложения, вычитания, умножения и деления.