Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра дискретной математики и
информационных технологий

# Разработка методических рекомендаций для дисциплины "Вычислительные системы, сети и телекоммуникации"

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 5 курса 521 группы
направления 09.03.01 – Информатика и вычислительная техника
факультета компьютерных наук и информационных технологий
Сычева Романа Ивановича

Научный руководитель
к. ф.-м.н., доцент _____ В.А. Поздняков
подпись, дата

Зав. кафедрой
к. ф.-м.н., доцент _____ Л.Б. Тяпаев
подпись, дата

Саратов 2016

# ВВЕДЕНИЕ

Главным преимуществом ассемблера, в отличие от других языков программирования, является то, что на нём можно написать более эффективную программу, чем если бы она была бы сгенерирована транслятором с языка высокого уровня. То есть для программ на ассемблере характерно использование меньшего количества команд и обращений в память, что позволяет увеличить скорость и уменьшить размер программы. Но так как языки низкого уровня сложнее для чтения и понимания так как состоят непосредственно из машинных команд, по этой причине увеличивается вероятность возникновения ошибки в программе и усложняется возможность совместной разработки.

Актуальность работы состоит в подготовке методических рекомендаций, которые позволят выполнять лабораторные работы непосредственно в современной ОС в защищённом режиме работы процессора. В то время как большинство существующих методических рекомендаций предлагает использовать реальный режим и устаревшую операционную систему DOS.

Целью данной бакалаврской работы является разработка методических рекомендаций для студентов по изучению архитектуры процессора i386 на машинно-ориентированном языке низкого уровня.

В процессе выполнения бакалаврской работы необходимо решить следующие задачи:

- Изучить правила разработки методических рекомендаций;
- изучить основные регистры процессора;
- изучить аппаратные и программные прерывания процессора;
- изучить различные режимы работы процессора;
- изучить организацию памяти и привилегии процессора при работе в защищённом режиме;
- рассмотреть процесс создания программ в ассемблере NASM под операционной системой AltLinux [1];

- разработать методические указания, позволяющие студентам изучить работу процессора в защищённом режиме.

В первой главе описаны правила разработки методических рекомендаций. Вторая глава даёт необходимый теоретический материал для изучения архитектуры процессора в защищённом режиме работы. В третьей главе необходимый материал для выполнения лабораторных работ.

Выпускная квалификационная работа состоит из 3 глав и 4 приложений:

1 Правила разработки методических рекомендаций;

2 Архитектура процессора x86;

2.1 Регистры процессора;

2.2 Прерывания процессора;

2.3 Режимы работы процессора;

2.4 Организация памяти в защищённом режиме;

3 Программирование в защищённом режиме работы процессора;

3.1 Синтаксис командной строки ассемблера NASM;

3.2 Методические указания для выполнения лабораторных работ;

ПРИЛОЖЕНИЕ А. Таблица кодировки символов ASCII;

ПРИЛОЖЕНИЕ Б. Использование простейших арифметических операций;

ПРИЛОЖЕНИЕ В. Методические рекомендации вызова подпрограмм;

ПРИЛОЖЕНИЕ Г. Работа с часами реального времени и CMOS.

**Структура лабораторных работ**

Каждая лабораторная работа выделена в отдельное приложение в бакалаврской работе и состоит из цели, примера задания, самостоятельных заданий и контрольных вопросов. Порядок лабораторных работ выбран таким образом, чтобы каждая следующая была сложнее предыдущей.

Цель работы показывает, какие знания получит обучающийся и что научиться делать на практике. Задание с комментариями показывает пример выполнения самостоятельных заданий, применения команд требуемых для изучения и использования теоретического материала на практике. Контрольные вопросы позволяют закрепить полученные знания и проверить понимание изученного материала.

**Программирование в защищённом режиме работы процессора**

**Методические указания для выполнения лабораторных работ**

Для выполнения лабораторных работ в качестве основной операционной системы использовалась операционная система “AltLinux 7.0.5” [15] и ассемблер NASM. Компиляция должна проводится с указанием формата объектного файла «elf». А сборку приложения необходимо проводить с указанием ключа «elf_i386», который указывает что тип получаемого исполняемого файла должен быть 32-битный.

Пример создания приложения и запуска приложения показан на рисунке 1. Предварительно необходимо перейти в папку с созданным файлом. В данном случае файл с исходным кодом test1.asm.

```
[linkor@host-15 ~]$ cd nasm
[linkor@host-15 nasm]$ nasm -f elf test1.asm
[linkor@host-15 nasm]$ ld test1.o -o test1 -m elf_i386
[linkor@host-15 nasm]$ ./test1
First
[linkor@host-15 nasm]$
```

Рисунок 1 – компиляция и запуск приложения.

При выполнении заданий для вывода данных на экран может потребоваться таблица символов ASCII, которой кодируются символы. Данная таблица приведена в приложении А.

В первой лабораторной работе, которая находится в приложении Б используются команды для выполнения арифметических операций с целыми числами. Она позволяет обучающимся освоить выполнение основных действий с регистрами общего назначения, изучить синтаксис расширенного ассемблера NASM и команды компилирования и сборки приложения. Изучаются команды сложения, вычитания, умножения и деления.

Операции сложения и вычитания над целыми числами производятся соответственно командами add и sub. Обе команды имеют по два операнда причём первый из них задаёт и одно из чисел участвующих в операции и место куда следует записать результат, а второй операнд задаёт второе число для операции: второе слагаемое или вычитаемое.

Первый операнд обязан быть регистровым либо типа память. Второй операнд у обеих команд может быть любого типа. Как и для команды mov, для команд add и sub нельзя использовать два операнда типа память одновременно.

В зависимости от полученного результата команды add и sub выставляют значения флагов OF, CF, ZF и SF.

Флаг ZF устанавливается в единицу, если в результате последней операции получился ноль, в противном случае флаг сбрасывается.

Флаг SF устанавливается в единицу, если получено отрицательное число. В другом случае он сбрасывается в ноль. Процессор производит установку этого флага, попросту копируя в него старший бит результата. То есть для знаковых чисел этот бит действительно означает знак числа, но для беззнаковых значение флага SF не имеет никакого смысла [17].

Флаг OF устанавливается, если произошло переполнение. Это означает, что в результате сложения двух положительных, получилось отрицательное. Так же возможен другой вариант, если в результате сложения двух

отрицательных чисел получилось положительное. Для беззнаковых чисел этот флаг не имеет смысла.

Флаг CF устанавливается, если для беззнаковых чисел произошел перенос из старшего разряда, либо произошел заём из несуществующего разряда. Для знаковых чисел этот флаг не имеет смысла.

Наличие флага переноса позволяет организовать сложение и вычитание чисел не помещающихся в регистры способом напоминающим школьное сложение и вычитание в столбик. Для этого в процессоре предусмотрены команды adc и sbb. По своей работе и свойствам они полностью аналогичны командам add и sub, но отличаются от них тем что учитывают значение флага переноса CF на момент начала выполнения операции. Команда adc добавляет к своему итоговому результату значение флага переноса, а команда sbb напротив вычитает значение флага переноса из своего результата. После того как результат сформирован, обе команды заново выставляют все флаги включая и CF уже в соответствии с новым результатом.

Так же существуют команды inc и dec, которые имеют всего один операнд и производят увеличение и уменьшение на единицу соответственно. Обе команды затрагивают флаги ZF, 0F и SF, но не затрагивают флаг CF. При использовании операнда типа память, указание размера операнда является обязательным [18].

Команда neg имеет так же только один операнд и обозначает операцию «унарный минус». Применяется обычно к знаковым числам, но в том числе устанавливает все четыре флага ZF, 0F, SF и CF.

Команда cmp производит точно такое же вычитание, как и команда sub, за исключением того, что результат никуда не записывается. Команда используется для установки флагов, после неё обычно используют команду условного перехода.

Все команды целочисленного умножения и деления имеют только один операнд, задающий второй множитель в командах умножения и делитель в

командах деления, причём этот операнд может быть регистровым или типа «память».

В качестве первого множителя задаётся неявный операнд из таблицы 1.

Таблица 1 – Неявные операнды при умножении и делении в NASM.

| Разрядность (бит) | умножение | | деление | | |
|---|---|---|---|---|---|
| | неявный множитель | результат умножения | делимое | частное | остаток |
| 8 | AL | AX | AX | AL | AH |
| 16 | AX | DX:AX | DX:AX | AX | DX |
| 32 | EAX | EDX:EAX | EDX:EAX | EAX | EDX |

Для умножения беззнаковых чисел применяют команду mul, для умножения знаковых imul. Эти флаги устанавливают флаги CF и 0F в ноль, если старшая половина результата равна нулю, в противном случае устанавливаются в единицу. Значения остальных флагов неопределены.

Для деления и нахождения остатка от деления целых числе применяют команду div (для беззнаковых) и команду idiv (для знаковых). Единственный операнд команды задаёт делитель. Делимое берётся из регистра или регистров в соответствии с таблицей 1. Частное всегда округляется в сторону нуля. Знак остатка, вычисляемого командой imul всегда совпадает со знаком делимого, а абсолютная величина остатка всегда строго меньше модуля делителя.

Во второй лабораторной работе рассматривается создание подпрограмм и вызов их из основной программы. Так же лабораторная работа позволяет изучить работу с циклами и стеком. Для выполнения работы, необходимо ознакомиться с понятием стека. Это структура данных, построенная по принципу «последний вошёл – первый вышел». Под стеком понимается непрерывная область памяти, для которой в специальном регистре хранится адрес вершины стека. При добавлении в стек некоторого значения уменьшается адрес вершины, сдвигая тем самым вершину вверх и в новую вершину

записывается добавляемое значение. Операция извлечения значения из стека считывает значение с вершины стека и сдвигает вершину вниз, увеличивая её адрес. Стек можно использовать для временного хранения значения любого регистра. Но самое главное предназначение, это хранение адресов возврата для передачи фактических параметров в подпрограммы и для хранения локальных переменных. Использование стека так же позволяет применять механизм рекурсии, при котором подпрограмма может прямо или косвенно вызывать сама себя.

Подпрограммой называется некоторая обособленная часть программного кода, которая может быть вызвана из главной программы или другой подпрограммы. Под вызовом необходимо понимать временную передачу управления подпрограмме с тем, чтобы, когда подпрограмма сделает свою работу, она вернула управление в точку, откуда её вызвали [19].

В современных вычислительных системах перед вызовом подпрограммы в стек помещаются значения параметров вызова, затем производится собственно вызов, то есть передача управления, которая совмещена с сохранением в том же стеке адреса возврата. Наконец, когда подпрограмма получает управление, она резервирует в стеке определённое количество памяти для хранения локальных переменных, обычно просто сдвигая адрес вершины вниз на соответствующее количество ячеек.

Третья лабораторная работа предполагает чтение значений из портов ввода-вывода. Операционная система AltLinux требует предварительно запрашивать доступ к нужным портам ввода-вывода. Для этого необходимо выполнить системный вызов к ядру ioperm.

Для того, чтобы открыть доступ к портам для чтения и записи, необходимо в регистр eax записать номер системного вызова, в регистры ebx и edx начальный номер порта, а в регистр ecx диапазон требуемых портов. После этого вызывается системный вызов int 80h и доступ к портам будет открыт.

Чтение или запись в порты ввода-вывода осуществляются командами in, out соответственно.

В команде in первый аргумент указывает в какой регистр сохранить прочитанное значение из регистра, а второй номер регистра. В команде out первый аргумент указывает номер регистра в который требуется записать данные, а второй аргумент требуемое значения для записи.

Все три лабораторные работы находятся в Приложении Б, Приложении В и Приложении Г.

# ЗАКЛЮЧЕНИЕ

В процессе выполнения бакалаврской работы был прочитан и систематизирован теоретический материал, а так же выполнены следующие поставленные во введении задачи:

- Изучены правила разработки методических рекомендаций;
- изучена архитектура процессора, различные режимы работы, организация памяти и привилегии процессора при работе в защищённом режиме;
- рассмотрен процесс создания программ в ассемблере NASM под операционной системой AltLinux;
- разработаны методические указания и лабораторные работы под расширенный ассемблер NASM.

Выполнение созданных лабораторных работ в расширенном ассемблере NASM позволит студентам закрепить знания в области создания программ в расширенном ассемблере NASM и изучить внутреннюю программную архитектуру процессора. Важной особенностью является то, что для их выполнения нет необходимости устанавливать какое-либо дополнительное ПО, эмулирующее работу операционной системы DOS и реальный режим работы процессора.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1 ALT Linux - Альт Линукс Kdesktop [Электронный ресурс] : URL: http://www.altlinux.ru/products/7th-platform/kdesktop/ (дата обращения: 22.04.2016). Загл. с экрана. Яз. рус.

2 Как подготовить и защитить методическую разработку / Серова Н. Л., Федина Н. П.: ББК 74. 202 К 14, 2004, 16 с.

3 Программирование на языке ассемблера NASM для ОС Unix / А.В. Столяров : ООО «МАКС Пресс», 2011, 188 с.

4 Ассемблер для DOS, Windows и UNIX / Зубков С. В : М.:ДМК, 2006.

5 Основы языка Ассемблера: учеб. курс / К. Г. Финогенов. - М. : Радио и связь, 2000. 288 с.

6 Ассемблер для процессоров Intel Pentium / Магда Ю. С.: СПб.: Питер, 2006. 410 с.

7 Микропроцессоры Intel: 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium Pro Processor, Pentium II, Pentium III, Pentium 4. Архитектура, программирование и интерфейсы. / Брей Б. : СПб.: БХВ-Петербург, 2005. 1328 с.

8 Ассемблер на примерах. Базовый курс / Рудольф Марек: СПб: Наука и техника, 2005. 240 с.

9 Использование языка ассемблера и языка сценариев. Учебное пособие / Мезенцев К. Н. : М.: МАДИ, 2015. – 176 с.

10 Ассемблер – это просто. Учимся программировать / Калашников О. А.: СПб.: БХВ-Петербург, 2011, 336 с.

11 ASSEMBLER. Учебный курс. / Пирогов В. Ю. : М.: Издатель Молгачева С. В., Издательство Нолидж, 2001, 848 с.

12 Программирование на ассемблере на платформе x86-64 / Аблязов Р. З. : М.: ДМК Пресс, 2011, 304 с.

13 Архитектура компьютера / Э. Танненбаум : СПб.: Питер, 2003, 848 с.

14 NASM [Электронный ресурс] : URL: http://www.nasm.us (дата обращения: 15.04.2016). Загл. с экрана. Яз. англ.

15 ALT Linux снаружи. ALT Linux изнутри : М.: ALT Linux; Издательский дом ДМК-пресс, 2006, 416 с.

16 Коды символов ASCII [Электронный ресурс] : URL: http://book.itep.ru/10/ascii.htm (дата обращения: 1.04.2016). Загл. с экрана. Яз. англ.

17 Assembler. Специальный справочник. / Юров В. : СПб.: Питер, 2001, 496 с.

18 Assembler. Практикум. / Юров В. : СПб.: Питер, 2001, 399 с.

19 Assembler – Учебный курс. / Юров В., Хорошенко В. : СПб.: Питер, 2000, 672 с.

20 Операционная система Linux: Курс лекций. Учебное пособие / Курячий Г. В., Маслинский К. А. : М.: ALT Linux; Издательство ДМК Пресс, 2010, 348 с.