**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

# ДОКЛАД

## на тему «Ассемблер NASM. Основные команды, типы данных, структура программы»

*дисциплина: Архитектура компьютера*

Студент: Матюшкин Денис Владимирович

Группа: НПИбд-02-21

№ ст. билета: 1032212279

**МОСКВА**

2021 г.

# Содержание:

# Введение.

Язык ассемблера (англ. assembly language) — машинно-ориентированный язык программирования низкого уровня. Представляет собой систему обозначений, используемую для представления в удобно читаемой форме программ, записанных в машинном коде. Его команды прямо соответствуют отдельным командам машины или их последовательностям. Является существенно платформо-зависимым: языки ассемблера для различных аппаратных платформ несовместимы, хотя могут быть в целом подобны.

Следует отметить, что процессор понимает не команды ассемблера, а последовательности из нулей и единиц — машинные коды. До появления языков ассемблера программистам приходилось писать программы, используя только лишь машинные коды, которые были крайне сложны для запоминания, так как представляли собой числа, записанные в двоичной или шестнадцатеричной системе счисления. Преобразование или трансляция команд с языка ассемблера в исполняемый машинный код осуществляется специальной программой транслятором — Ассемблер. Программы, написанные на языке ассемблера, не уступают в качестве и скорости программам, написанным на машинном языке, так как транслятор просто переводит мнемонические обозначения команд в последовательности бит (нулей и единиц).

Расширенный ассемблер NASM - это 80x86 ассемблер, разработанный исходя из принципов переносимости и модульности. Он поддерживает широкий диапазон форматов объектных файлов, включая форматы Linux a.out и ELF, NetBSD/FreeBSD, COFF, Microsoft 16-bit OBJ и Win32. Он способен также создавать простые бинарные файлы. Синтакс NASM максимально упрощен для понимания и похож на синакс Intel, но слегка посложнее. Он поддерживает инструкции Pentium, P6 и MMX, а также имеет макро-расширения.

# 1. Типы данных.

В языке ассемблера используются данные следующих типов:

1. Непосредственные данные, представляющие собой числовые или символьные значения, являющиеся частью команды.Непосредственные данные формируются программистом в процессе написания программы для конкретной команды ассемблера.

2. Данные простого типа, описываемые с помощью ограниченного набора директив резервирования памяти, позволяющих выполнить самые элементарные операции по размещению и инициализации числовой и символьной информации. При обработке этих директив ассемблер сохраняет в своей таблице символов информацию о местоположении данных (значения сегментной составляющей адреса и смещения) и типе данных, то есть единицах памяти, выделяемых для размещения данных в соответствии с директивой резервирования и инициализации данных.

Эти два типа данных являются элементарными, или базовыми; работа с ними поддерживается на уровне системы команд микропроцессора. Используя данные этих типов, можно формализовать и запрограммировать практически любую задачу.

3. Данные сложного типа, которые были введены в язык ассемблера с целью облегчения разработки программ. Сложные типы данных строятся на основе базовых типов, которые являются как бы кирпичиками для их построения. Введение сложных типов данных позволяет несколько сгладить различия между языками высокого уровня и ассемблером. У программиста появляется возможность сочетания преимуществ языка ассемблера и языков высокого уровня (в направлении абстракции данных), что в конечном итоге повышает эффективность конечной программы.

| Тип | Директива | Количество байт |
|---|---|---|
| Байт | DB | 1 |
| Слово | DW | 2 |
| Двойное слово | DD | 4 |
| 8 байт | DQ | 8 |
| 10 байт | DT | 10 |

# 2. Структура программы.

Программирование на уровне машинных команд — это тот минимальный уровень, на котором возможно составление программ. Система машинных команд должна быть достаточной для того, чтобы реализовать требуемые действия, выдавая указания аппаратуре вычислительной машины.

Каждая машинная команда состоит из двух частей:

- операционной — определяющей, «что делать»;
- операндой — определяющей объекты обработки, «с чем делать».

Типичный формат записи команд NASM имеет вид:

[метка:] **команда/директива** [операнд {, операнд}] [; комментарий]

Строка текста на языке ассемблера NASM состоит (в общем случае) из четырёх полей:

• Метки

• Имени команды

• Операндов

• Комментария

При-чём метка, имя команды и комментарий являются полями необязательными, что касается операндов, то требования к ним налагаются командой; если имя команды отсутствует, то отсутствуют и операнды. Могут отсутствовать и все четыре поля, тогда строка оказывается пустой. Ассемблер пустые строки игнорирует, но мы можем использовать их, чтобы визуально разделять между собой части программы.

Если команду или директиву необходимо продолжить на следующей строке, то используется символ обратный слеш: \. По умолчанию язык ассемблера не различает заглавные и строчные буквы в написании команд или директив.

**Метка** в языке ассемблера может содержать следующие символы:

- все буквы латинского алфавита;
- цифры от 0 до 9;
- спецсимволы: _, @, $, ?.

В качестве первого символа метки может использоваться точка, но некоторые компиляторы не рекомендуют применять этот знак. В качестве меток нельзя использовать зарезервированные имена Ассемблера (директивы, операторы, имена команд).

Первым символом в метке должна быть буква или спецсимвол (но не цифра). Максимальная длина метки – 31 символ. Все метки, которые записываются в строке, не содержащей директиву ассемблера, должны заканчиваться двоеточием :.

**Операнд** – объект, над которым выполняется машинная команда или оператор языка программирования.

Команда может иметь один или два операнда, или вообще не иметь операндов. Число операндов неявно задается кодом команды.

Примеры:

- Нет операндов ret ;Вернуться
- Один операнд inc ecx ;Увеличить ecx
- Два операнда add eax,12 ;Прибавить 12 к eax

Метка, команда (директива) и операнд не обязательно должны начинаться с какой-либо определенной позиции в строке. Однако рекомендуется записывать их в колонку для большего удобства чтения программы.

В качестве операндов могут выступать

- идентификаторы;
- цепочки символов, заключенных в одинарные или двойные кавычки;
- целые числа в двоичной, восьмеричной, десятичной или шестнадцатеричной системе счисления.

**Комментарии** отделяются от исполняемой строки символом «;» . При этом все, что записано после символа точка с запятой и до конца строки, является комментарием. Использование комментариев в программе улучшает ее ясность, особенно там, где назначение набора команд непонятно. Комментарий может содержать любые печатные символы, включая пробел. Комментарий может занимать всю строку или следовать за командой на той же строке.

**Идентификаторы** – последовательности допустимых символов, использующиеся для обозначения таких объектов программы, как коды операций, имена переменных и названия меток. Правила записи идентификаторов:

- Идентификатор может состоять из одного или нескольких символов.
- В качестве символов можно использовать буквы латинского алфавита, цифры и некоторые специальные знаки: _, ?, $, @.
- Идентификатор не может начинаться символом цифры.
- Длина идентификатора может быть до 255 символов.
- Транслятор воспринимает первые 32 символа идентификатора, а остальные игнорирует.

Примеры строк кода:

```
Count    db     1        ;Имя, директива, один операнд
mov     eax,0     ;Команда, два операнда
cbw              ; Команда
```

## 3. Компиляция и компоновка.

NASM компилирует программы под различные операционные системы в пределах x86-совместимых процессоров. Находясь в одной операционной системе, можно беспрепятственно откомпилировать исполняемый файл для другой.

Компиляция программ в NASM состоит из двух этапов. Первый — ассемблирование, второй — компоновка. На этапе ассемблирования создаётся объектный код. В нём содержится машинный код программы и данные, в соответствии с исходным кодом, но идентификаторы (переменные, символы) пока

не привязаны к адресам памяти. На этапе компоновки из одного или нескольких объектных модулей создаётся исполняемый файл (программа). Операция компоновки связывает идентификаторы, определённые в основной программе, с идентификаторами, определёнными в остальных модулях, после чего всем идентификаторам даются окончательные адреса памяти или обеспечивается их динамическое выделение.

Для компоновки объектных файлов в исполняемые в Windows можно использовать свободный бесплатно распространяемый компоновщик alink[4](для 64-битных программ компоновщик GoLink), а в Linux — компоновщик ld, который есть в любой версии этой операционной системы.

Для ассемблирования файла нужно ввести следующую команду:

```
nasm -f format filename -o output
```

## 4. Основные команды.

Команда указывает транслятору, какое действие должен выполнить микропроцессор. В сегменте данных команда (или директива) определяет поле, рабочую область или константу.

Типичными командами языка ассемблера являются (большинство примеров даны для Intel-синтаксиса архитектуры x86):

- Команды пересылки данных (mov и др.)
- Арифметические команды (add, sub, imul и др.)
- Логические и побитовые операции (or, and, xor, shr и др.)
- Команды управления ходом выполнения программы (jmp, loop, ret и др.)
- Команды вызова прерываний (иногда относят к командам управления): int
- Команды ввода-вывода в порты (in, out)

## 5. Достоинства и недостатки языка ассемблера.

К преимуществам Ассемблера можно отнести:

• Данный язык программирования позволяет создавать приложения, которые будут более эффективны, чем аналогичные приложения, написанные на языке высокого уровня, т.е. приложения будут более короткими и при этом более быстро выполнимыми.

• Язык Ассемблера позволяет программисту выполнять действия, которые либо вообще нельзя реализовать на других языках и в частности на языках высокого уровня, либо выполнение которых займет слишком много машинного времени в случае привлечения дорогих средств языка высокого уровня.

К недостаткам языка следует отнести:

• По мере увеличения своего размера программа на Ассемблере теряет наглядность. Это связано с тем, что в ассемблерных программах следует уделять много внимания деталям. Язык требует от вас планирования каждого шага ЭВМ. Конечно, в случае небольших программ это позволяет сделать их оптимальными с точки зрения эффективности использования аппаратных средств. В случае же больших программ бесконечное число деталей может помешать вам добиться оптимальности программы в целом, несмотря на то, что отдельные фрагменты программы будут написаны очень хорошо.

• Для программирования на данном языке необходимо очень хорошо знать структуру компьютера и работу аппаратных устройств, так как Ассемблер работает непосредственно с устройствами.

# Заключение.

Язык ассемблера — тип языка программирования низкого уровня, представляющий собой формат записи машинных команд, удобный для восприятия человеком.

На языке ассемблера пишут:

- программы, требующие максимальной скорости выполнения: основные компоненты компьютерных игр, ядра операционных систем реального времени и просто критичные по времени куски программ;
- программы, взаимодействующие с внешними устройствами: драйверы, программы, работающие напрямую с портами, звуковыми и видеокартами;
- программы, использующие полностью возможности процессора: ядра многозадачных операционных систем, серверы;
- программы, полностью использующие возможности операционной системы: вирусы, антивирусы, защита от несанкционированного доступа, программы обхода защиты от несанкционированного доступа.

# Литература.


1. https://ru.m.wikipedia.org/wiki/Язык_ассемблера

2.https://esystem.rudn.ru/pluginfile.php/1185223/mod_resource/content/0/Лабораторная%20работа%202.pdf

3.https://esystem.rudn.ru/pluginfile.php/1227266/mod_resource/content/0/Лабораторная%20работа%203.pdf

4. А. В. Столяров Программирование на языке ассемблера NASM для ОС Unix

5. https://ru.wikipedia.org/w/index.php?title=NASM&stable=1

6. https://prog-cpp.ru/asm/