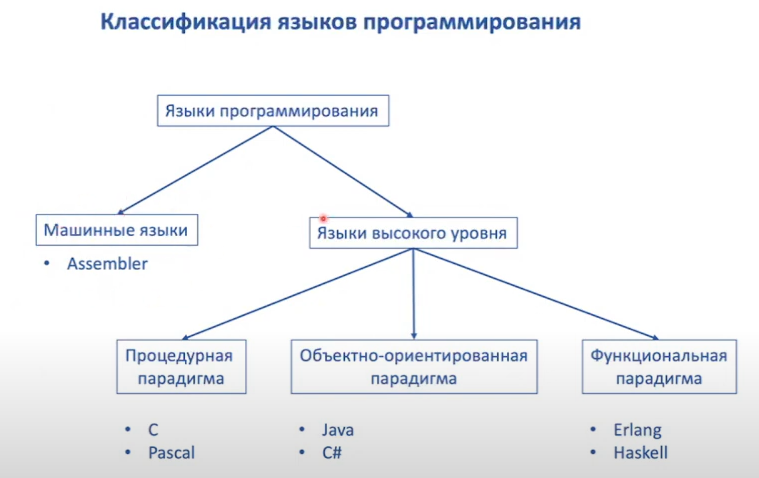
Лекция 1



Assembler

+ Напрямую обращаемся к процессору на его языке

+ Самый близкий к машинному

- Большой код и не структурировано

- Сложен для изучения

- индивидуален для каждой архитектуры

Процедурная парадигма – парадигма в которой программа выглядит в виде линейного кода. Алгоритм описывается в виде последовательности шагов, преобразования входной информации в выходную. Программа состоит из двух частей, часть в виде данных где описываются все переменные и часть где машинные инструкции языка, которые преобразовывают эти данные.

Си

+ Самый быстрый язык

+ Является лучшей версией Assembler

- Чистый язык мало что умеет без библиотек

Объектно-ориентированная парадигма — 3 основных кита: наследование, инкопсуляция и полиморфизм. Данные с кодом находятся в одном классе. Программа состоит из различных классов, а в классе функции с помощью которых мы и взаимодействуем.

Функциональная парадигма - весь код реализуется в виде функций, в отличие от других парадигм если в функцию передается параметр то он не изменен и функция всегда должна что то вернуть, так же нет циклов они реализуются в виде рекурсии то есть вызывает сама себя.

- Сложно перестроится с других парадигм и начать писать на функциональной

+ Простота чтения кода так как название функции обычно объясняет что делает данная функция

Трансляторы — программа которая переводит из любого языка программирования в машинный код.

Компилятор — это транчлятор который сразу проводит программу сразу через три этапа лексический, синтаксический и семантический анализ, на выходе получается бинарная программа которую можно в последствии запустить и посмотреть результат.

+ Если есть какие то ошибки типа лексические. Синтаксические или семантические то компилятор вам укажет на них и не бкдет создавать бинарную программу.

- Если же ошибка была логической то придется найти ошибку, переписать исходный код и снова прогнать всю программу через 5 этапов компиляции

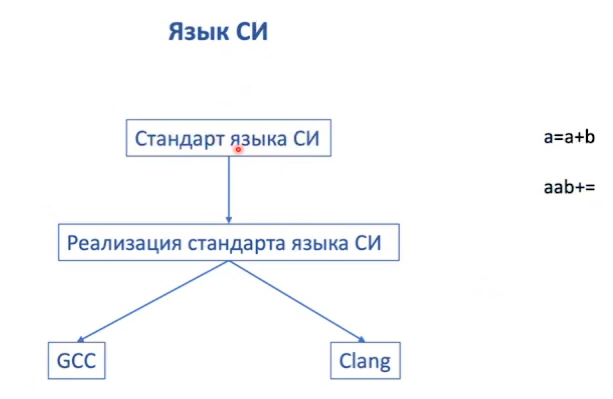
- Бинарный код можно запускать только на той системе которая бинарно совместима то есть та же архитектура, операционная система и тот же машинный язык

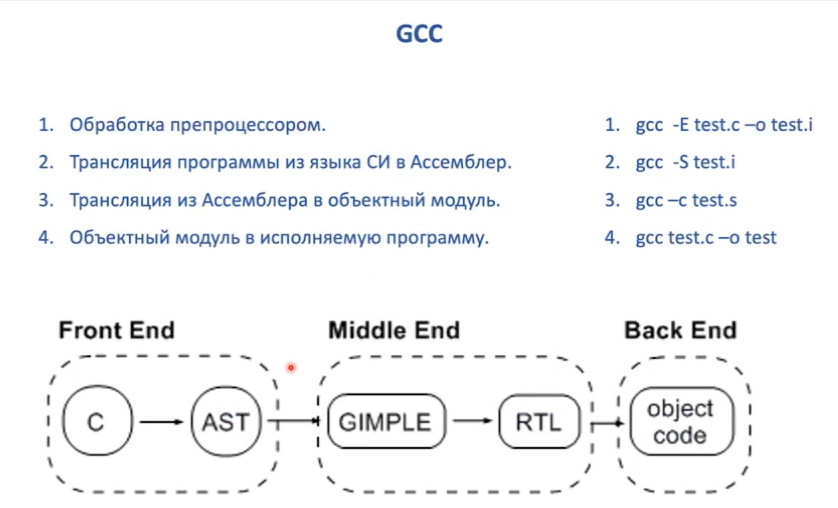
Интерпретатор — Выполняет код по строчно то есть сразу выдает результат по строке.

+ Выдает ошибку сразу как её нашел

+ Архитектурно независимый

- Медленно работает в отличии от компилятора

GCC – ((GNU C Compile) GNU Compiler Collection).

Clang – более современный и быстрый предназначен для x86 архитектуры.

Этапы компиляции

1. Обработка препроцессора — это вспомогательные программы для упрощения программирования так что сначала приводим к чистому си.

+ Посмотреть какие именно версии библиотек использовались

2. Трансляция из языка программирования в Assembler (мнимоники).

+ можно посмотреть код на Assembler и посмотреть что компилятор мог удалил из программы

3. Из Assembler в машинный код. Выполнение не возможно так как данную программу нужно разместить в оперативной памяти по определенному адресу и записать данный адрес в определенный регистр процессора.

+ Для раздельной компиляции

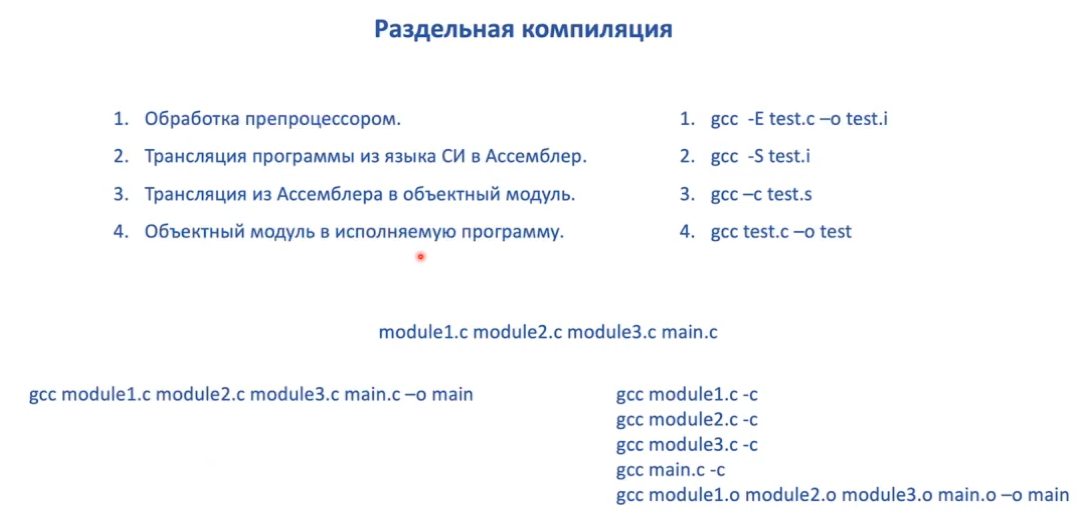
4. Размещается в нужном месте и дописываются некоторые инструкции для того что бы программа могла выполнятся в операционной системе.

Front End – часть в которой программу переводим во внутреннее представление.

Внутреннее представление — (AST) Абстрактное синтаксическое дерево предназначено для перевода кода в более удобный вариант для машины.

Middle End – преобразования оптимизационные (GINPLE) и переводится практически в машинный язык (RTL). То есть приведение над кодом всех возможных оптимизаций.

Back End – трансляция в машинный язык

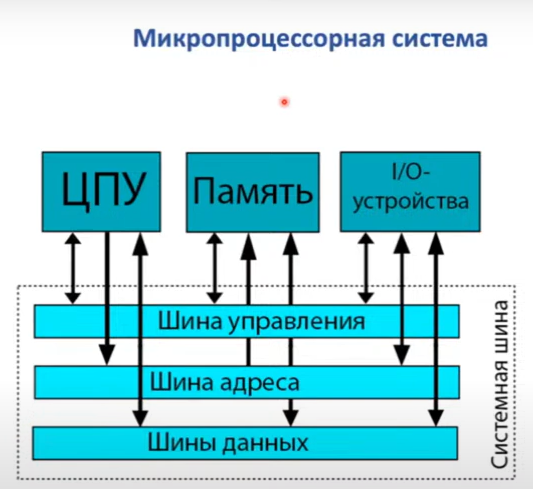
Раздельная компиляция — позволяет разбить программу на нескоько отдельных модулей и их уже объединить в бинарник.

Лучший вариант для такой компиляции это каждый модуль довести до объектного файла что бы прошли первые 3 этапа компиляции и уже объектные файлы объединить в бинарник.

+ в большой программе дает возможность пере компилировать не всю программу, а только тот модуль который был изменен и в последствии объединим в бинарник

<https://gcc.gnu.org/onlinedocs/gccint/> - Про GCC

Лекция 2

 Микропроцессорная система — система в которой есть центральный процессор который делает основные вычисления, имеется память и устройства ввода/вывода. И все это соединяется с системной шиной.

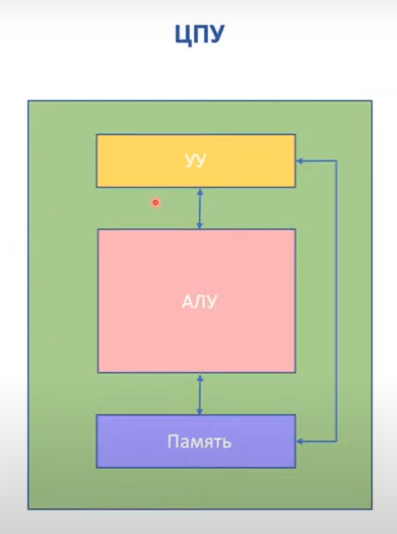
Системная шина — набор проводников по которой распространяется сигнал, разрядность системы зависит от количества этих проводников.

Шина управления — по ней распространяются сигналы которые говорят о том что хотим сделать с данными например записать или прочитать.

Шина адреса и шина данных объединены в одну в реальной системе.

Центральный процессор отправляет сигнал контроллеру памяти что он хочет прочитать некоторую ячейку памяти и выставляет на шину адреса адрес ячейки из которой он хочет прочитать. Контроллер памяти получив сигнал забирает с шины адреса адрес, забирает из ячейки по этому адресу и забирает данные, и выставляет эти данные на шину данных. Центральный процессор в следующей итерации забирает эти данные с шины данных.

Пример копирования данных в буфер с мыши . Центральный процессор отправляет сигнал на чтение данных с устройства ввода данные попадают на шину данных, затем центральный процессор отправляет управляющий сигнал контроллеру памяти о том что сейчас данные необходимо прочитать и записать по определенному адресу. Контроллер памяти забирает адрес по нему данные и записывает их в другую ячейку памяти.

Структурная схема Джона фон Неймана

Устройство управления — занимается тем что извлекает данные из оперативной памяти через системную шину и затем декодирует команду (считывает код операции) и передает сигнал АЛУ.

Арифметико логическое устройство — аппаратное реализация всех машинных инструкций процессора, то есть машинный язык.

Память — регистр, каждый регистр нужен для своей операции.

Изъяли машинную инструкцию → декодировали её(УУ) → Выполнили(АЛУ) → Сохранили результат(Память)

+ Программы стали храниться в оперативной памяти

- Выполняется только одну операцию одновременно

Конвейер — разбиение основной задачи на под задачи независимо. Независимо для того что бы могло все работать параллельно.

Этапы выполнения машинной инструкции

1 — извлечение машинной инструкции из памяти

2 — декодирование

3 — вычисление инструкции

4 — получить доступ к памяти что бы забрать операнд для того что бы делать вычисления

5 — сохранить результат

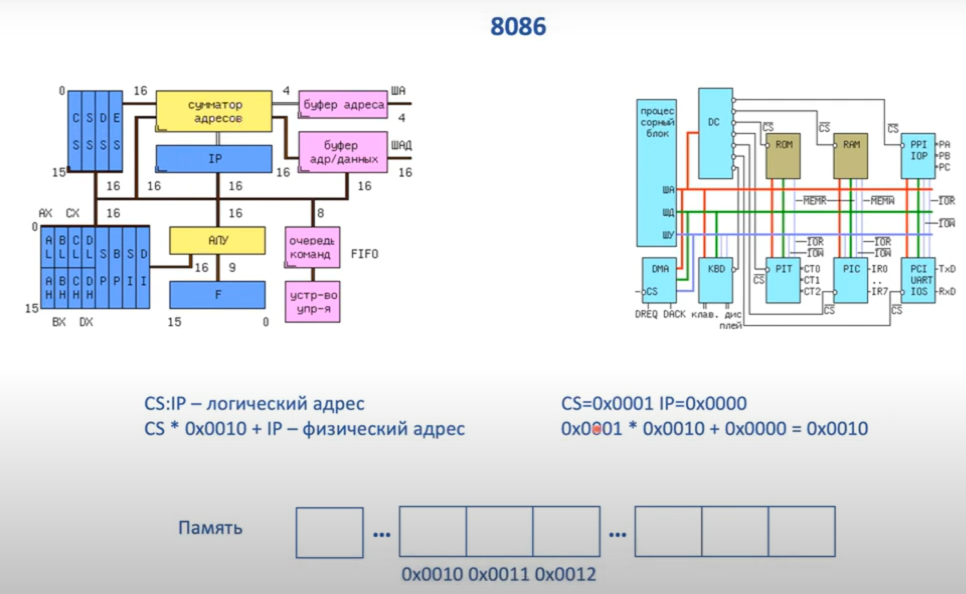
Конвейер позволяет начать выполнять новую инструкцию когда предыдущая еще не выполнена благодаря чему увеличивать производительность.

Программное обеспечение: Прикладное — все пользовательские программы, Системное — ОС, драйвера, прошивки и SDK

Архитектура компьютера — набор показателей и свойств системы которые призваны удовлетворить потребность пользователя.

Микроархитектура — отвечает за аппаратную реализацию набора инструкций.

Лекция 3

Структурная схема 8086 процессора

1

2

Данный процессор 16 разрядный то есть шина данных и адреса 16 разрядные.

Очередь команд — небольшой участок памяти в которой может поместиться какое то количество команд, команды поступают из оперативной памяти.

На схеме синим цветом выделены регистры: Ron(общего назначения) и специального назначения.

1. Ron – ячейки памяти определенной разрядности в нашем случае 16 разрядные у которых есть имена. Имена нужны для обращения к ним то есть не через адрес а через имя. Имена это номера машинных конструкций.

AX, CX, BX, DX – это регистры в которых хранятся 2 байта есть возможность обращаться к старшему или младшему байту, для этого надо в место X указать H – старший или L – младший. В данных регистрах хранятся операнды.

SP, BP – регистры для работ со стеком. Стек — структура данных в виде очереди.

SI, DI – индексные регистры

F – регистр флагов нужен для того что бы проверить корректно завершилась предыдущая машинная инструкция.

2. Специального назначения хранится начало программы. Позволяют структурировать программу в памяти корректно. Разделяет программу на отдельные сегменты: кода, данных и стека.

CS - сегмент где хранятся только машинные инструкции которые нужно выполнить,

DS - сегменте данных расположены переменные,

SS - сегмент стека хранит информацию необходимую для выполнения функции.

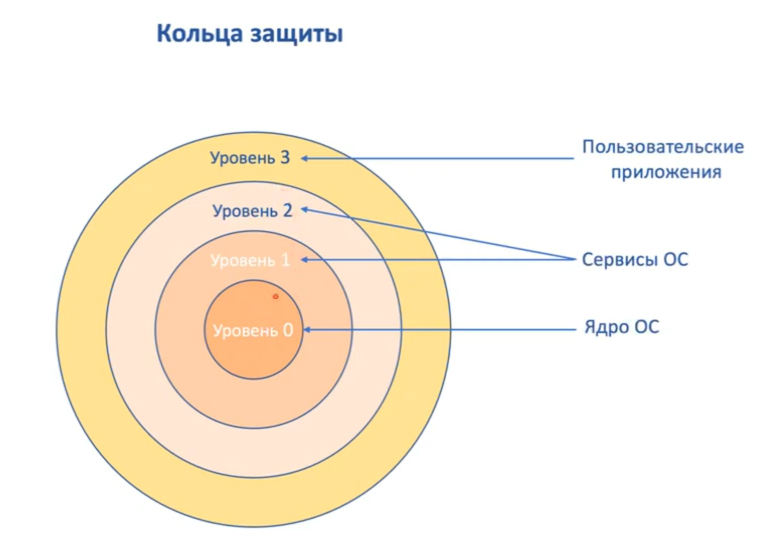
ES – дополнительный сегмент

IP(instruction pointer) – хранит смещение

Физический адрес — адрес в оперативной памяти (линейный) передается контролеру адреса

Логический адрес - это комбинация начала сегмента и смещения

Сегменты нужны для того что бы процессор мог выполнить программу и что бы ему было проще идентифицировать данные переменные и машинные инструкции. И выполнять только машинные инструкции.

Кольца это просто абстракция. Номер кольца хранится в виде числа.

При создании сегмента программы мы присваиваем ему число

0 — Самая привилегированный уровень. Код на данном уровне может делать все что хочет и использовать любые ячейки памяти

1 — Не может обращаться на 0 уровень но может на 2 и 3 чаще всего используется для драйверов

2 — может обращаться на 2 и 3 уровень

3 — Программы не могут обращаться к другим уровням и даже к другим программам на данном уровне

Лекция 5

1 байт = 8 бит → 2^8 = 256 но 255 так как отсчет начинается с 0

Дополнительный код

1. переводим число в двоичную систему счисления

2. инвертируем полученный результат то есть 0 превращаем в 1, а 1 в 0

3. прибавляем единицу

4. в старший разряд записываем единицу в не зависимости что там уже стоит

Пример: - 7

1. Переводим 7 в двоичную систему счисления

7 2 записываем в обратном порядке и до 8 разрядов 00000111

6 3 2

1 2 1

1

2. инвертируем 00000111 = 11111000

3. прибавляем единицу 11111001

4. В старший разряд записываем единицу 11111001

Ответ: -710 = 111110012

Из 10 в 16-ричную систему счисления

1000 16 Записываем остатки в обратном порядке

96 62 16

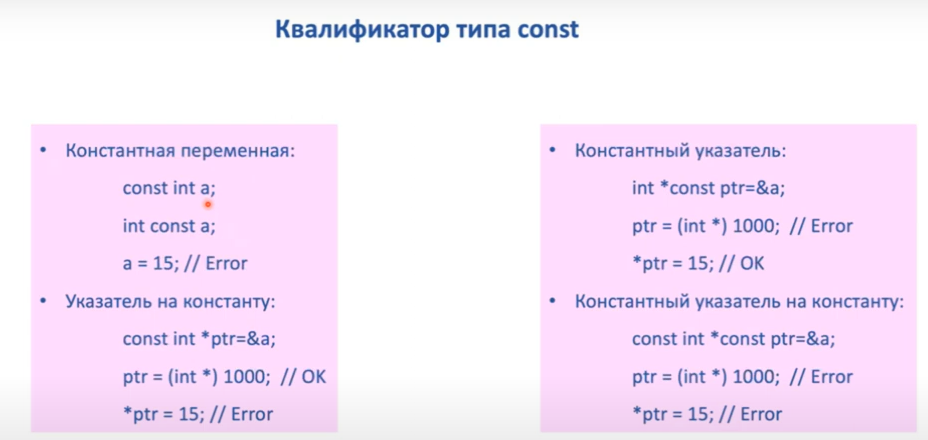
40 48 3 16 3E(14)8

32 14 0 0

8 3

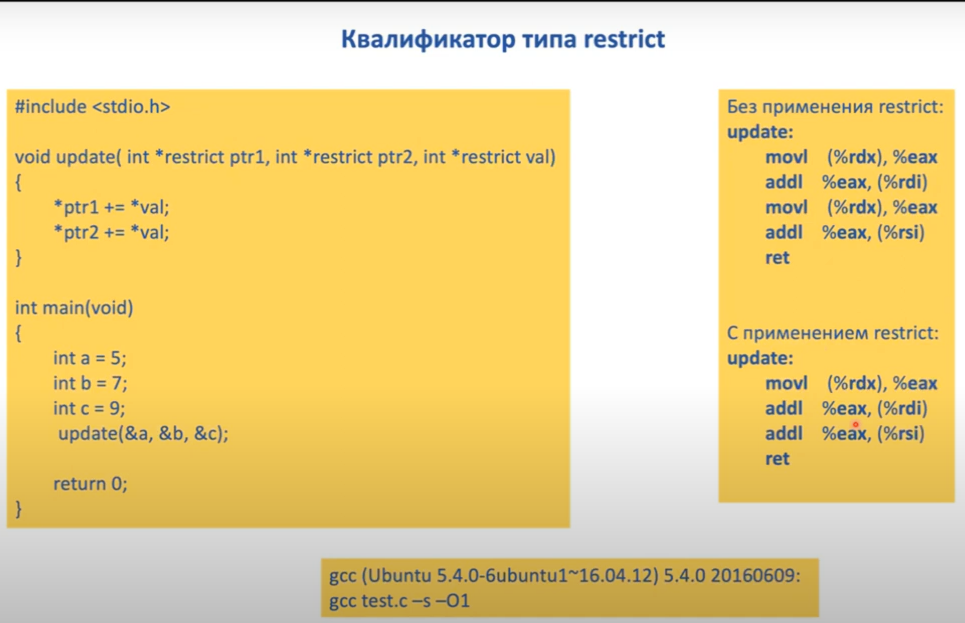
A – 10; B – 11; C – 12; D – 13; E – 14; F – 15.

Ответ: 100010 = 3E816

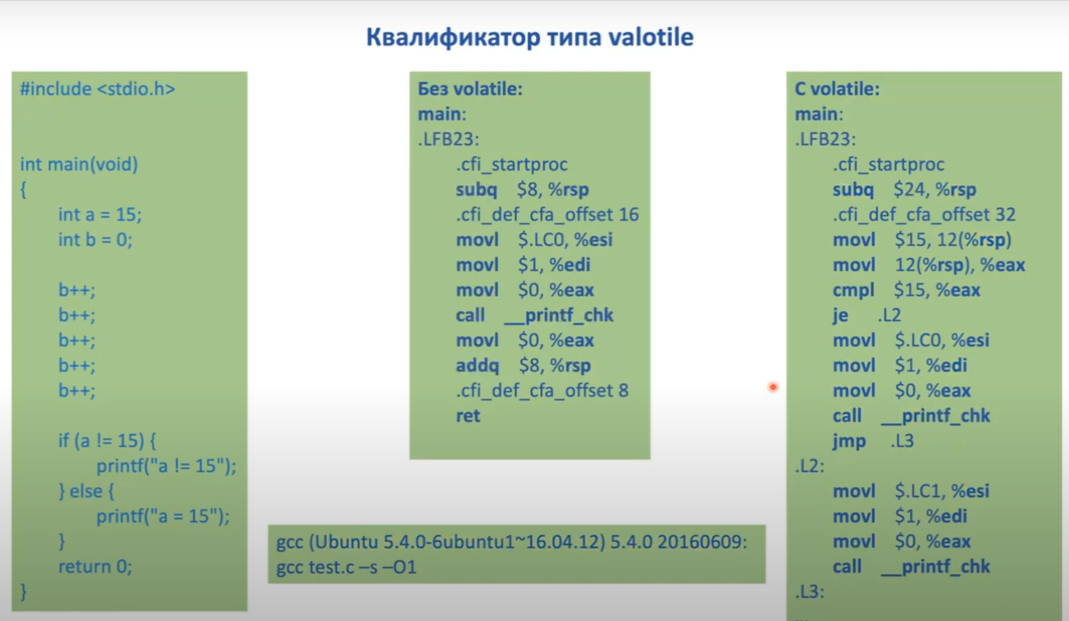
В си необходимо сразу после объявления переменной сразу необходимо её инициализировать.

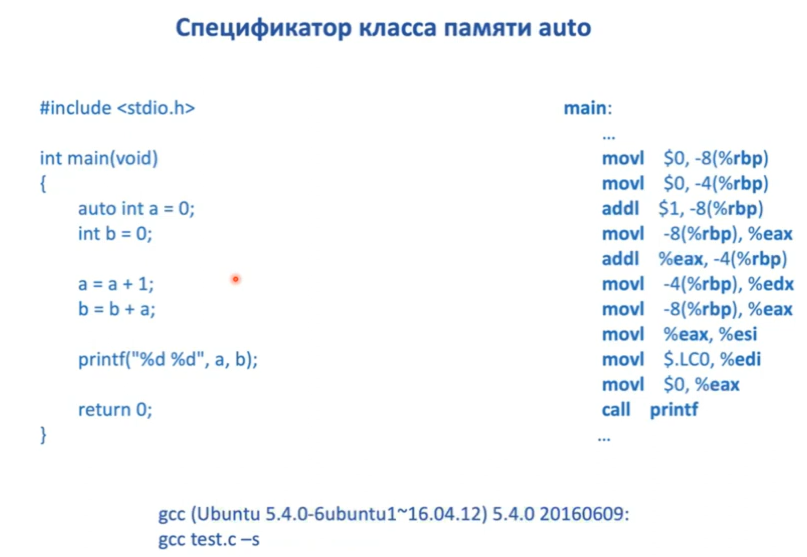
В константную переменную нельзя записывать новые значения.

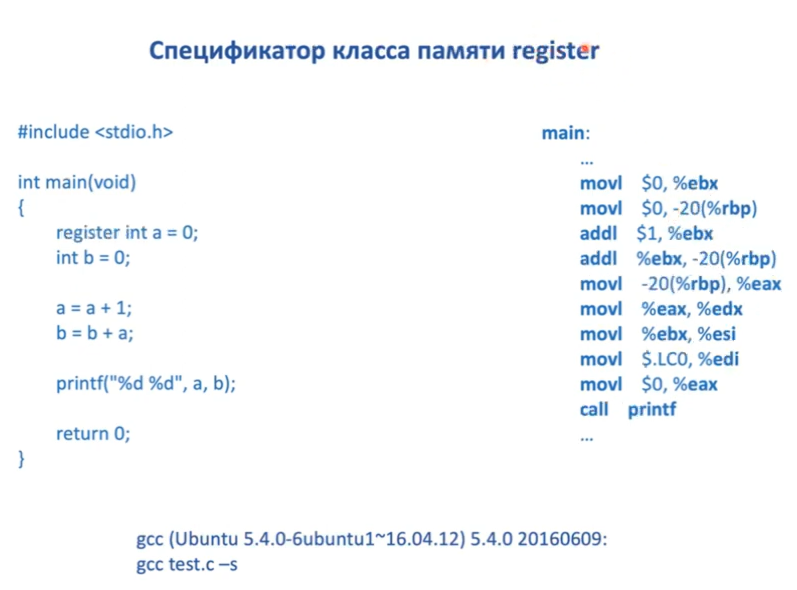
В указатели на константу мы можем изменить адрес на переменную, но не можем изменить значение самой переменной.

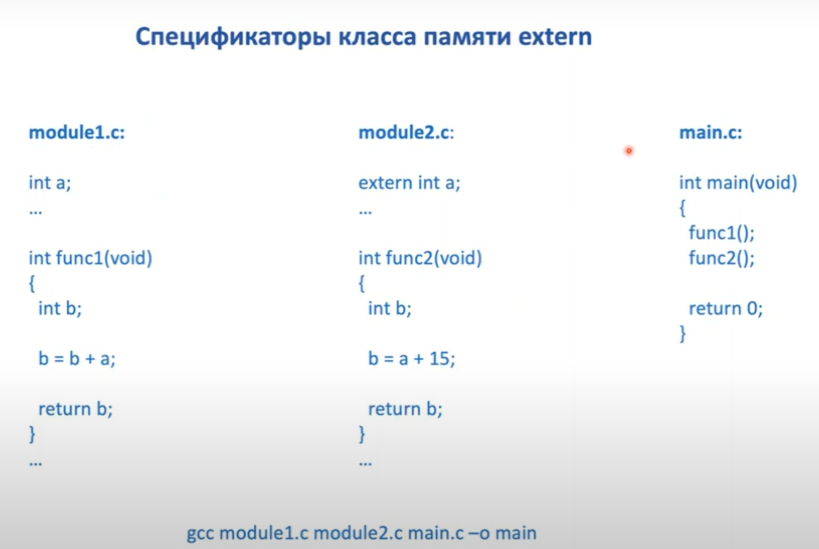
В константном указатели мы можем изменять значения переменной, но не можем менять адрес который указывает на эту переменную.

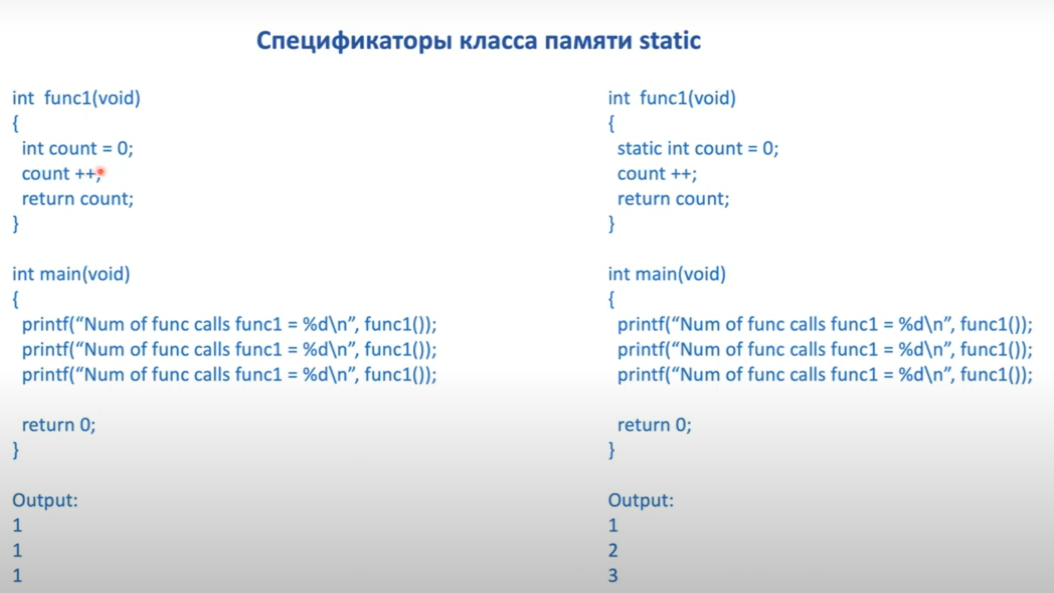
Restrict – употребляется к указателям что бы сообщить компилятору, что данный указатель указывает на уникальную область и эта область с другими областями памяти не пересекается. Это рекомендация компилятору от нас, сам компилятор этого проверять не будет. Restrict величает скорость выполнения программы.

Volatile - указываем что какие то части кода могут быть изменены не очевидным образом для компилятора и что бы он не удалял эту часть кода из машинных инструкций. (volatile int a = 15).

auto - указывает компилятору что переменную он может расположить куда он хочет либо в оперативную память, либо в регистре процессора. Хранит переменную в регистре имеет преимущество в виде большей скорости к обращению к ней так как необходимо меньше тактов чем если бы нам необходимо было бы сначала вытащить из памяти. Она больше рекомендация компилятору, в случае если регистре нет места то он разместит в оперативной.

Register – указывает компилятору что необходимо разместить переменную в регистер. Так же является рекомендованной.

Extern – указывает что эта переменная где то уже объявленная поэтому снова этого не делать.

Static – глобализация переменная которая является глобальной только для определенной функции.