Информатика и программирование

Романов Евгений Леонидович, к.т.н., доц. каф. ВТ НГТУ,

romanov@corp.nstu.ru, WhatApp 9139449081, не забываем представляться (Фамилия, группа)

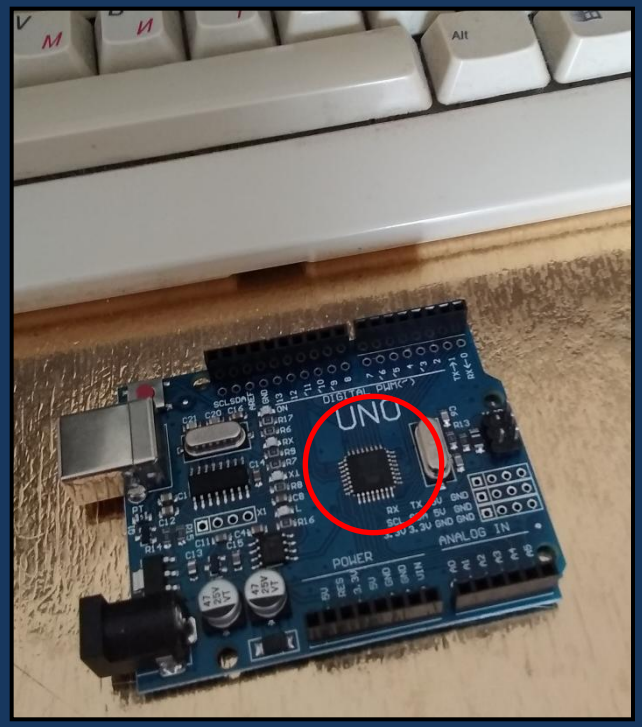
https://dispace.edu.nstu.ru/didesk/course/show/13239 - курс в Dispace

Все материалы: **vk.com/cprog_cs -** полезные ссылки:

- Метод. Материалы и задания – Информатика
- Презентации – Информатика и ЯП
- Романов Е.Л. Си++. От дилетанта до профессионала.rar (Cprog в pdf)
- Cprog.rar - web-версия (Си++. От дилетанта до профессионала) Файлы - исходники фрагментов, упомянутых в cprog, содержатся в папке programs архива cprog.rar с именами, которые находятся в заголовках фрагментов.

Содержание лекции: Компьютерная архитектура, структура программного кода, языки программирования **(ЯП)**. Парадигмы программирования. Императивное программирование и ООП.

«Я – преподаватель со стажем, стоял у истоков вычислительной техники и до сих пор там стою»



1979: Электроника-100И (PDP-8)

- 32К памяти программ и данных (ОП, сохраняемая) – 12 разрядных слов
- Тактовая частота - 1 Мгц
- Среда разработки – язык Ассемблера
- ОС – «самоделка»
- Заводской №29

2019: Arduino UNO:

- 32 Кб памяти программ (Flash)
- 2 Кб памяти переменных (RAM)
- Тактовая частота - 16 Мгц
- Среда разработки на хост-компьютере - классический Си
- Встроенный загрузчик программ

Реальность Arduino – классический Си

```
int PIN_TRIG=12;
int PIN_ECHO=11;
int play=3;
int ledPin = 13; // к пину 13 подключён встроенный светодиод
long duration, cm;

void setup() {

  // Инициализируем взаимодействие по последовате

  Serial.begin (9600);
  //Определяем вводы и выводы
  pinMode(PIN_TRIG, OUTPUT);
  pinMode(PIN_ECHO, INPUT);
  pinMode(ledPin, OUTPUT); // задаём вывод 13 в к
  pinMode(play,OUTPUT);
  digitalWrite(play,LOW);
}
```

```
void setup() {

  // Инициализируем взаимодействие по последов

  Serial.begin (9600);
  //Определяем вводы и выводы
  pinMode(PIN_TRIG, OUTPUT);
  pinMode(PIN_ECHO, INPUT);
  pinMode(ledPin, OUTPUT); // задаём вывод 13
  pinMode(play,OUTPUT);
  digitalWrite(play,LOW);
}
```

```
void loop() {

  // Сначала генерируем короткий импульс длительностью

  digitalWrite(PIN_TRIG, LOW);
  delayMicroseconds(5);
  digitalWrite(PIN_TRIG, HIGH);

  // Выставив высокий уровень сигнала, ждем около 10 м
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG, LOW);

  //  Время задержки акустического сигнала на эхолокат
  duration = pulseIn(PIN_ECHO, HIGH);

  // Теперь осталось преобразовать время в расстояние
  cm = (duration / 2) / 29.1;

  Serial.print("Расстояние до объекта: ");
  Serial.print(cm);
  Serial.println(" см.");
  if (cm < 40){
    digitalWrite(play,HIGH);
    delay(3000);
    digitalWrite(play,LOW);
    }
  // Задержка между измерениями для корректной работы
  delay(250);
}
```

Компьютерная и программная архитектура

Архитектура фон Неймана (1940-е годы):

- прямо адресуемая память
- принцип хранимой программы: программа с точки зрения представления в памяти = данные
- **поток управления (поток команд) –** последовательность команд с адресами операндов

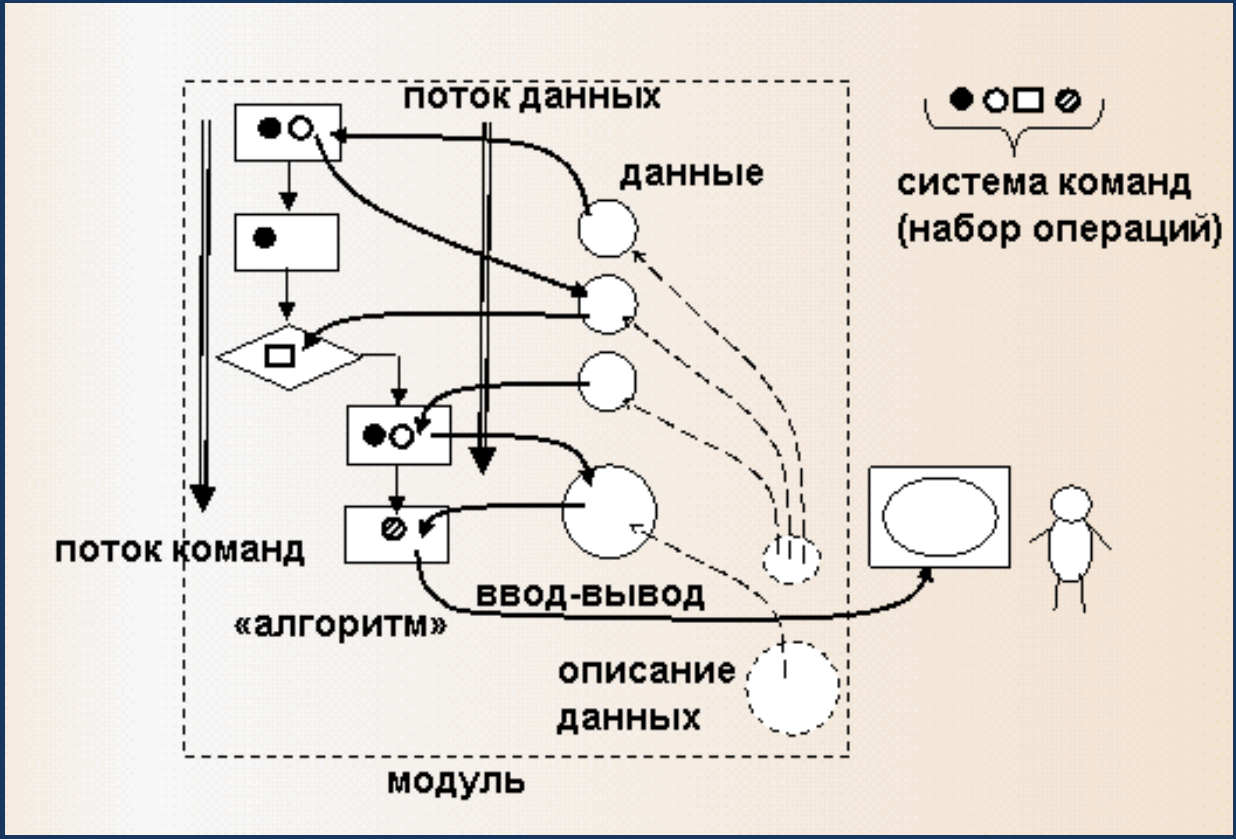
«**Медицинские факты»:**

- архитектура «железа» - фон Неймановская
- все технологии представления программ (ООП) компилируются во внутреннюю архитектуру
- архитектура может быть реальной и виртуальной (Java, байт-код – двоичный код для виртуальной архитектуры, JVM – процессор, написанный на Си)

Структура программы и ЯП

Алгоритм (в широком смысле) – однозначное описание порядка (последовательности) выполнения действий из заданного набора над системой объектов предметной области, позволяющее получить требуемый результат за конечное число шагов

Программа – в общем случае **алгоритм (в широком смысле)**



Структура программы и ЯП

Особенности программы (взгляд не user-а, а программиста**):**

- предметная область – **данные,** программа работает не с пользователем, а с данными, пользователь – внешний поток данных
- **тип данных (ТД) –** форма представления данных (формат, диапазон, операции)
- **переменная –** именованная или адресуемая (по ссылке) область памяти с данными определенного типа (статическая привязка ТД)
- **набор операций (системе команд),** которые можно выполнять над данными: арифметические операции, присваивание (сохранение результата в переменной), ввод-вывод, проверка значения переменной…
- **алгоритм «в узком смысле» -** описание порядка, последовательности выполняемых действий: 2 уровня – выражения и операторы.
- **выражения** - описание линейной последовательности выполнения простейших действий из набора операций (арифметические и логические операции, присваивание, условные выражения)
- **операторы -** собственно описание произвольной последовательности действий

Структура программы и ЯП

Компоненты языка программирования:

- **средства описания данных** определение типов данных (форма представления) и переменных
- **набор операций** над основными типами данных (включая ввод-вывод), а также средства записи выражений
- **набор операторов**, определяющих различные варианты порядка выполнения выражений в программе (последовательность, условие, повторение, блок, переключатель);
- средства поддержки **модульности** (функции, процедуры, классы, пакеты, библиотеки
  - -логически завершенные фрагменты кода с программным интерфейсом – функции, процедуры, методы, классы
  - физические компоненты кода – файлы, пакеты, библиотеки
- средства **обработки ошибок -** исключения

Программа = алгоритм + данные

Программа = алгоритм + данные (Н.Вирт, ЯП Паскаль, Оберон)

Различные формальные системы, имеющие отношение к формализму (программа) [cprog, 3.8]:

- машина Тьюринга – формальная модель для доказательства **алгоритмической разрешимости или неразрешимости** проблемы
- **конечный автомат** – «программа без данных», «чистая» блок-схема, использующая только данные внешнего объекта управления + текущее состояние => автоматные модели программ, использующих переменные состояния, описание сетевых протоколов

Конечный автомат – инстинктивное поведение

Программа – рефлекторное поведение.

Великая операция присваивания – программа может запоминать не только данные, но и свое состояние, чтобы использовать это в дальнейшем

Структура системы исполнения кода

- **Транслятор –** программная компонента, обеспечивающая исполнение текста программы, написанной на языке программирования (**исходник, source code**)
- **компиляция –** перевод текста программы с исходного формального языка на другой, более простой язык (в т.ч. двоичный код системы команд)
- **интерпретация –** непосредственное выполнение инструкций (команд, операторов) формального языка над образами языковых объектов программы (обозначенных в программе именами)
- **программный (машинный) код -** язык программирования компьютерной архитектуры. Данные в этом языке представлены **машинными словами,** хранящимися в памяти и в регистрах процессора, а выполняемые действия – **системой команд**
- **процессор + память =** интерпретатор машинного кода
- **язык Ассемблера** - язык программирования, система команд, способы адресации и машинные слова и их адреса обозначаются **символическими именами**

«**Медицинский факт»:** любая система исполнения программного кода = компилятор + интерпретатор

Структура системы исполнения кода

«Целевой» компилятор – компилирует исходный код в машинный код целевой машины (обычно «родной», иногда «чужой» - кросс-компилятор) – Си, Паскаль

«Чистый» программный код - «самодостаточный», не содержит неявных обращений к сторонним библиотекам, неявных проверок и действий.

Компиляция на промежуточный код (виртуальной машины)

- **Java –** байт-код + JVM (Java Virtual Machine) – интерпретатор байт-кода
- **платформа .Net (Microsoft) –** компиляция на виртуальный Ассемблер с языков программирования VisualBasic, Java#, C#

Среда исполнения программного кода - библиотека окружения + системные вызовы ОС, «среда», в которой программный код может «существовать».

Средства сборки (компоновки) программного кода – независимая компиляция файлов исходных текстов кода на промежуточный уровень (объектный модуль, библиотека) с последующей сборкой (**компоновщик, linker**)

Средства управления проектом – объединение файлов исходного текста, библиотек, настроек компилятора и компоновщика

Особенности Си/Си++

Машинно – независимый ассемблер - одновременно язык высокого и низкого уровня (не всегда справедливо для Си++):

- «самодостаточный» код, не использующий вызовы среды исполнения – встроенные системы для «голой» машины, ядро ОС.
- как правило компиляция «операция - команда», совпадение базовых типов данных с форматами представления данных в процессоре (машинные слова)
- управление памятью на низком уровне (адресная арифметика)

Следствия:

- максимальная производительность кода
- компактность кода
- отсутствие привязки к операционному окружению (ОС)

Си – это не то, на чем пишется всё, а то, на чем пишется то, на чем пишется и работает всё остальное

История ЯП и технологий программирования

50 – годы – машинный (двоичный) код, первые компиляторы Алгол – Фортран, блок-схема

60 – годы – Бейсик на основе Фортрана, «чистый» интерпретатор

70 – годы – Паскаль, Си (классический), структурное программирование

90 – годы – Си++, Java – платформенно – независимая, объектно-ориентированное программирование (ООП)

00 – годы – программная инженерия, C#, Scala,

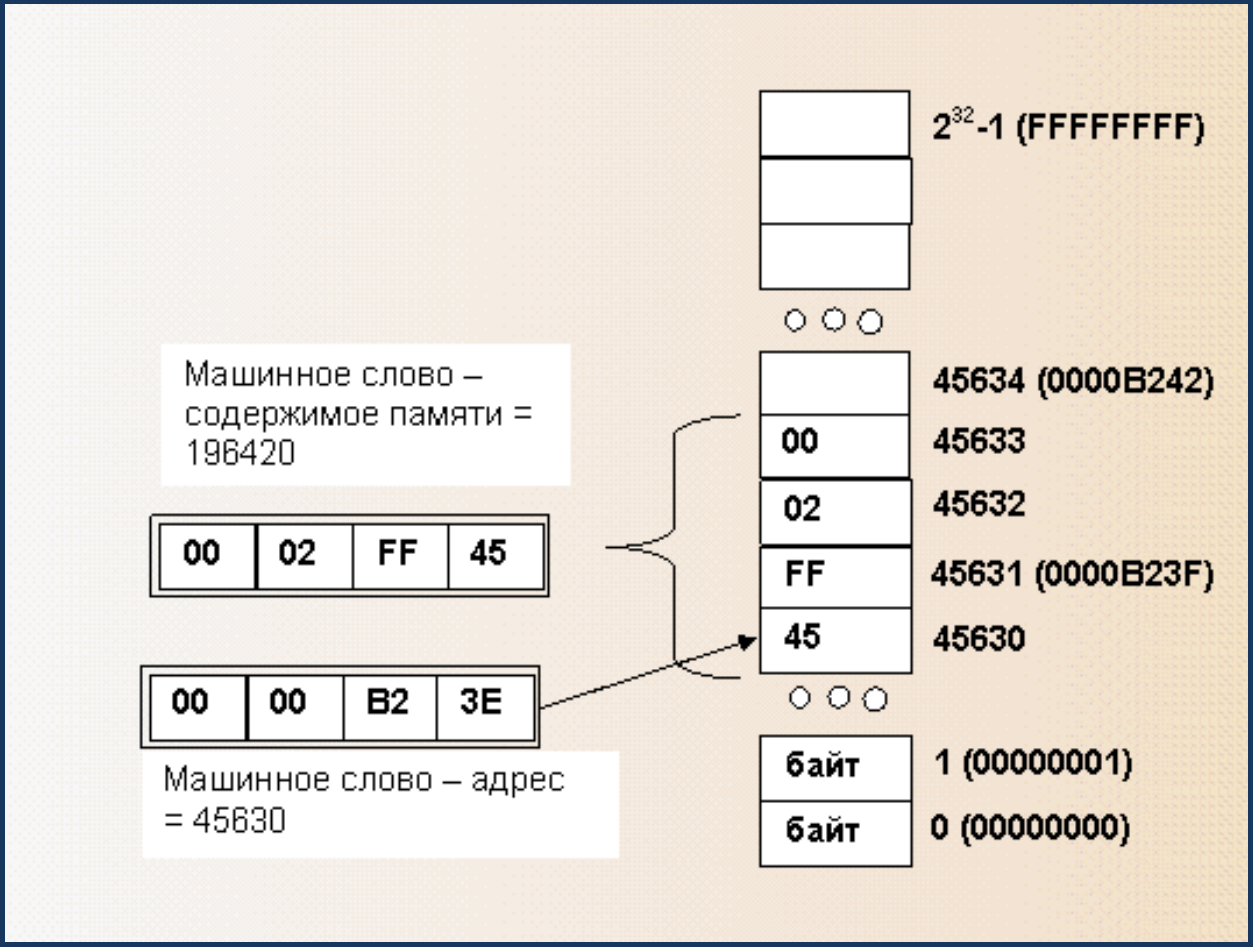
Семейства ЯП

- Фортран – Бейсик – VBA
- Паскаль – Delphi – Oberon
- Си – C++ - C# - Java - Scala
- Функциональное и логическое программирование – Prolog, Lisp, Scala

Компьютерная архитектура

Прямо – адресуемая память

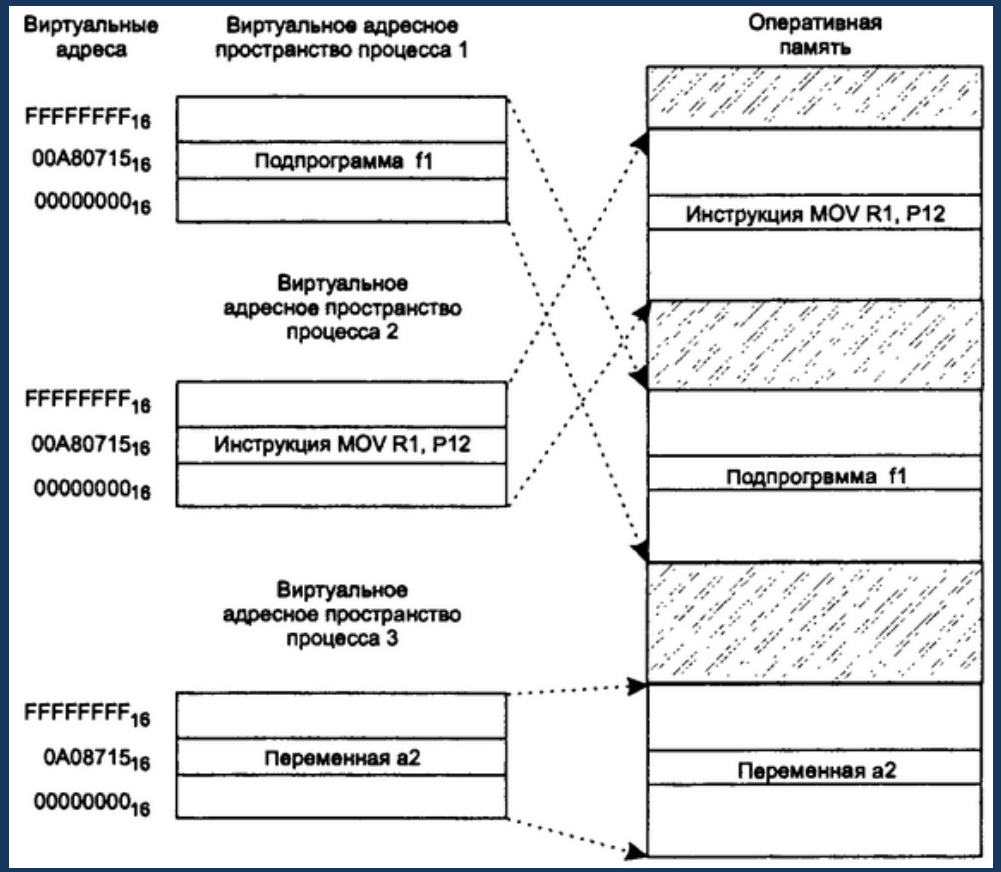
- **адрес –** двоичный (16-ричный) номер байта
- адресное пространство (АП) – непрерывный диапазон адресов



Компьютерная архитектура

Прямо – адресуемая память

- **машинное слово (МС)** – байт (byte), короткое МС (short), стандартное МС = разрядность процессора, длинное (long)
- **регистры** – собственные МС процессора
- **физическое АП (ФАП)** – АП реальных адресов на шине памяти
- **виртуальное АП (ВАП)** – АП программы (адресов, содержащихся в командах), которое фрагментарно отображается на ФАП блоками по 4Кб (страницами) конкуренция задач за ФАП



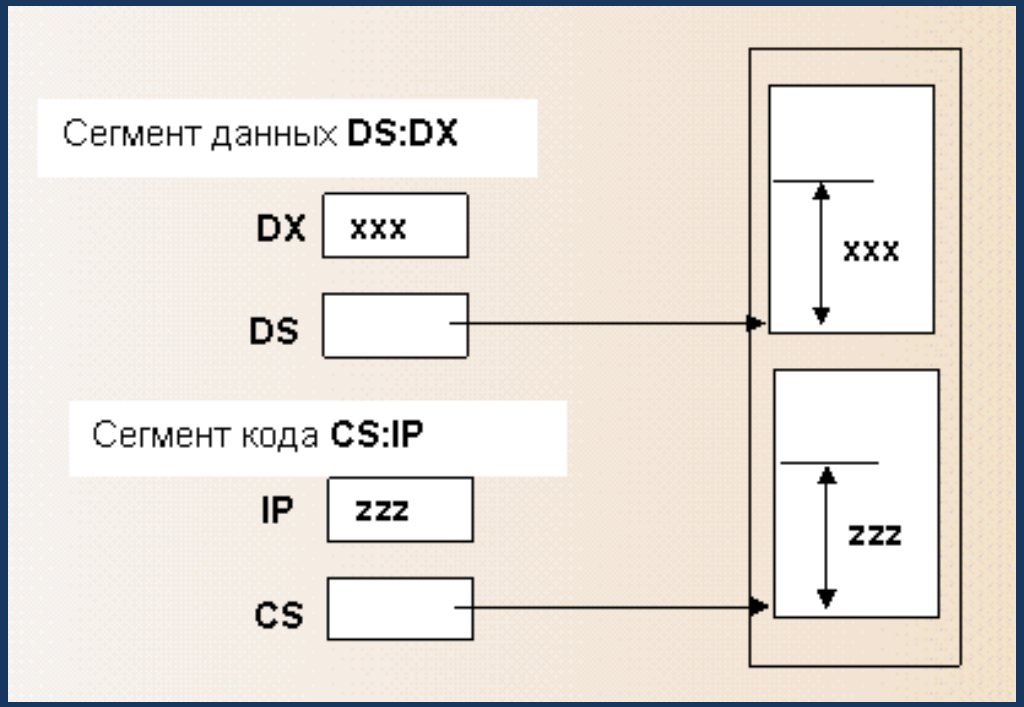
Компьютерная архитектура

Сегментация памяти

Сегмент - непрерывная область памяти, хранящая данные одного вида (назначения) и имеющая собственную систему относительной адресации и ограничения доступа

База – начальный адрес сегмента в АП (базовый регистр)

Смещение – относительный адрес от начала сегмента



Компьютерная архитектура

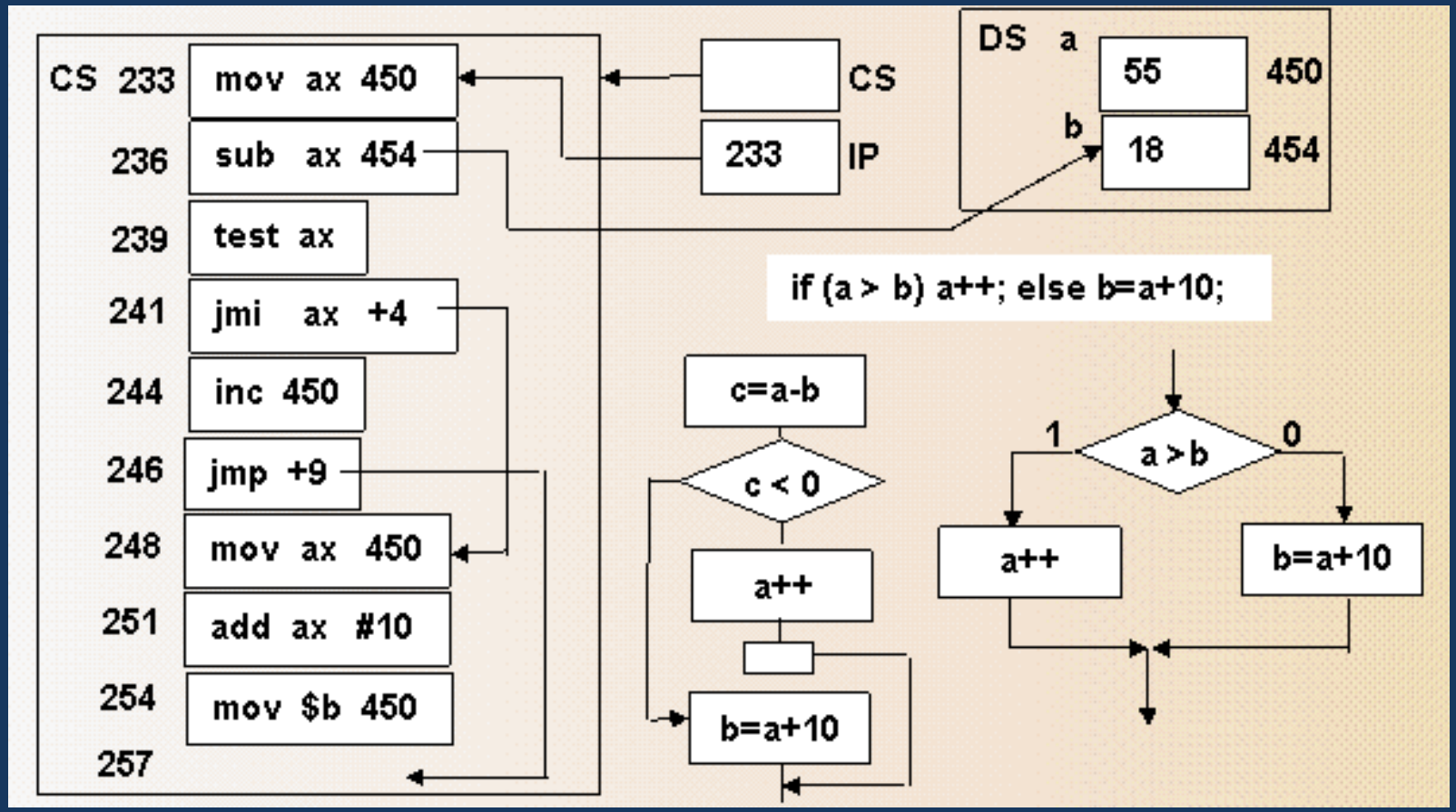
Сегментация памяти

Сегмент	Регистры	Что содержит	Когда создается
Сегмент команд	CS- сегментный, IP- адрес команды	Программный код (операции, операторы)	Трансляция
Сегмент данных	DS – сегментный	Глобальные (статические) данные	Трансляция
Сегмент стека	SS – сегментный, SP- указатель стека	Локальные данные функций, «история» работы программы	При загрузке
Динамическая память	DS – сегментный	Динамические переменные, создаваемые при работе программы	При загрузке, выполнении
Динамически связываемые библиотеки (DLL)	CS- сегментный, IP- адрес команды	Программный код разделяемых библиотек	При загрузке

Компьютерная архитектура

Внутреннее представление программы

CS – базовый регистр сегмента кода, IP – адрес следующей команды



Компьютерная архитектура

Система команд процессора = блок – схема

1. команды **перемещения и обработки** (**mov,add,sub**) - прямоугольник
2. команды **проверки состояний (условий** (**test**) - ромб
3. команды **условного и безусловного переходов -** стрелка

Структура команды: код операции + адреса операндов (данные для их вычисления)

Способы адресации операндов и команд - способ преобразования «коротких» данных адресной части в адреса операндов (регистры ЦП, память (длинные))

- неявный (по умолчанию, например, стек)
- регистровый (номер регистра CPU)
- стековый (операнд в стеке)
- абсолютный – в команде адрес (смещение)
- относительный – смещение от текущей (следующей) команды (IP)
- косвенный регистровый - в регистре адрес операнда в памяти
- базовый – относительно начала сегмента

Структура средств разработки

- **транслятор Си** (gcc, g++) , программа, вызывающая Си-компилятор, ассемблер, компоновщик, командный процессор (make) + библиотеки – компоненты **режима командной строки**
- **интегрированная среда разработки (IDE) –** система графической разработки (редактирования, отладки, дизайна, конфигурирования) проектов на определенном ЯП
- **фреймворк (каркас) –** система заготовок и поддержки разработки приложений определенного вида (обычно в рамках IDE и языка) – например, web, БД, мобильные…

Средства сборки программ в ЯП

Основные идеи:

- независимая компиляция **файла исходного текста** в промежуточный **объектный модуль**
- **библиотека** – архив **объектных модулей**
- **компоновка** – сборка объектных модулей исходного текста и библиотек в **исполняемый файл**

Термины:

- **объектный модуль** - «полуфабрикат», содержащий оттранслированную часть программы во внутреннем представлении, а также информацию о внешних связях модуля программы в исходном (символьном) виде (точки входа и внешние ссылки)
- **исполняемый файл –** двоичный файл, «образ памяти», содержит внутренне представление программы в машинном коде

Средства сборки программ в ЯП

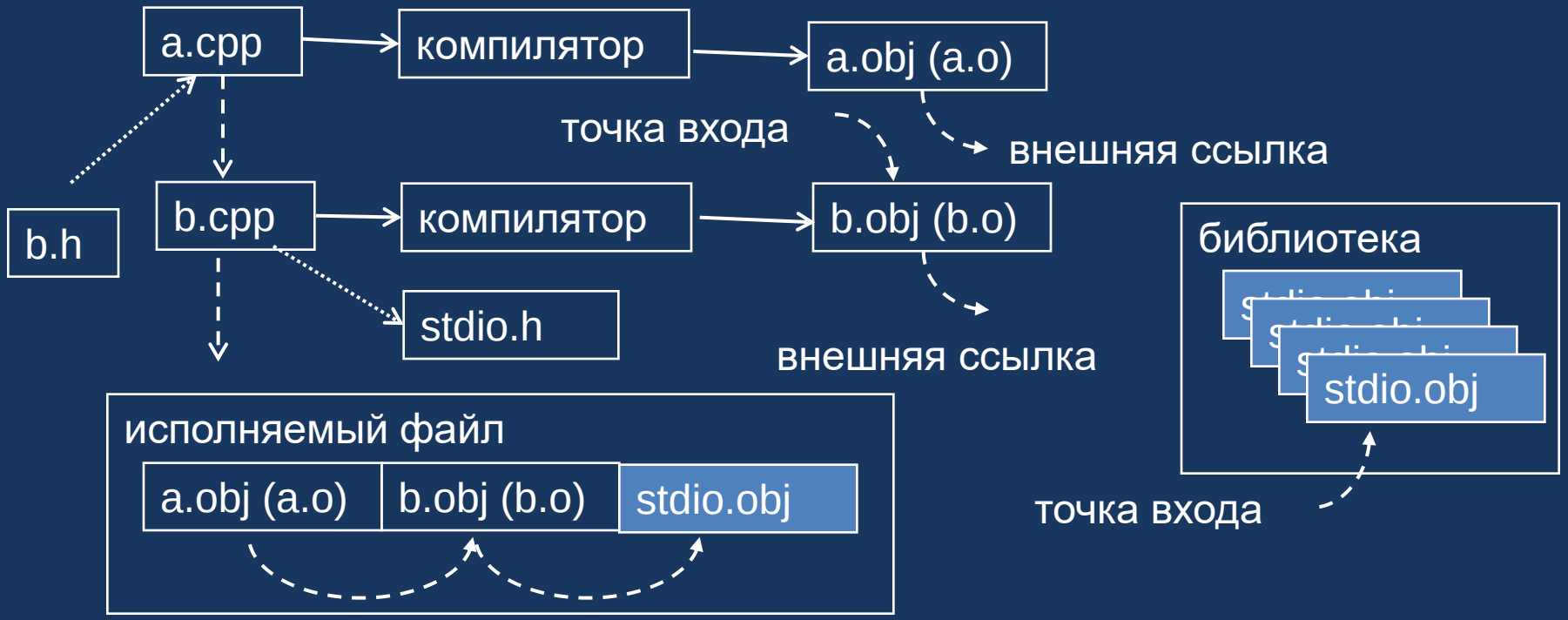
Реализация:

- логический модуль (функция, метод), содержит **интерфейсную часть – заголовок и тело**
- тело компилируется во внутреннее представление
- интерфейс определяет правила передачи данных между телом модуля и точной его вызова в специальной области параметров (стек)

Функции linker-а:

- **компоновка** – размещение тел логических модулей в едином АП исполняемого файла
- **редактирование связей -** замена символьных внешних ссылок на адреса точек входа в исполняемом файле
- **cpp-файл –** файл исходного текста программы
- **h-файл (заголовочный)** – интерфейсы логических модулей (заголовки функций) + определение типов данных
- **#include <xxx.h>** - включение в процесс трансляции определения интерфейсов для внешних ссылок

Средства сборки программы на Си

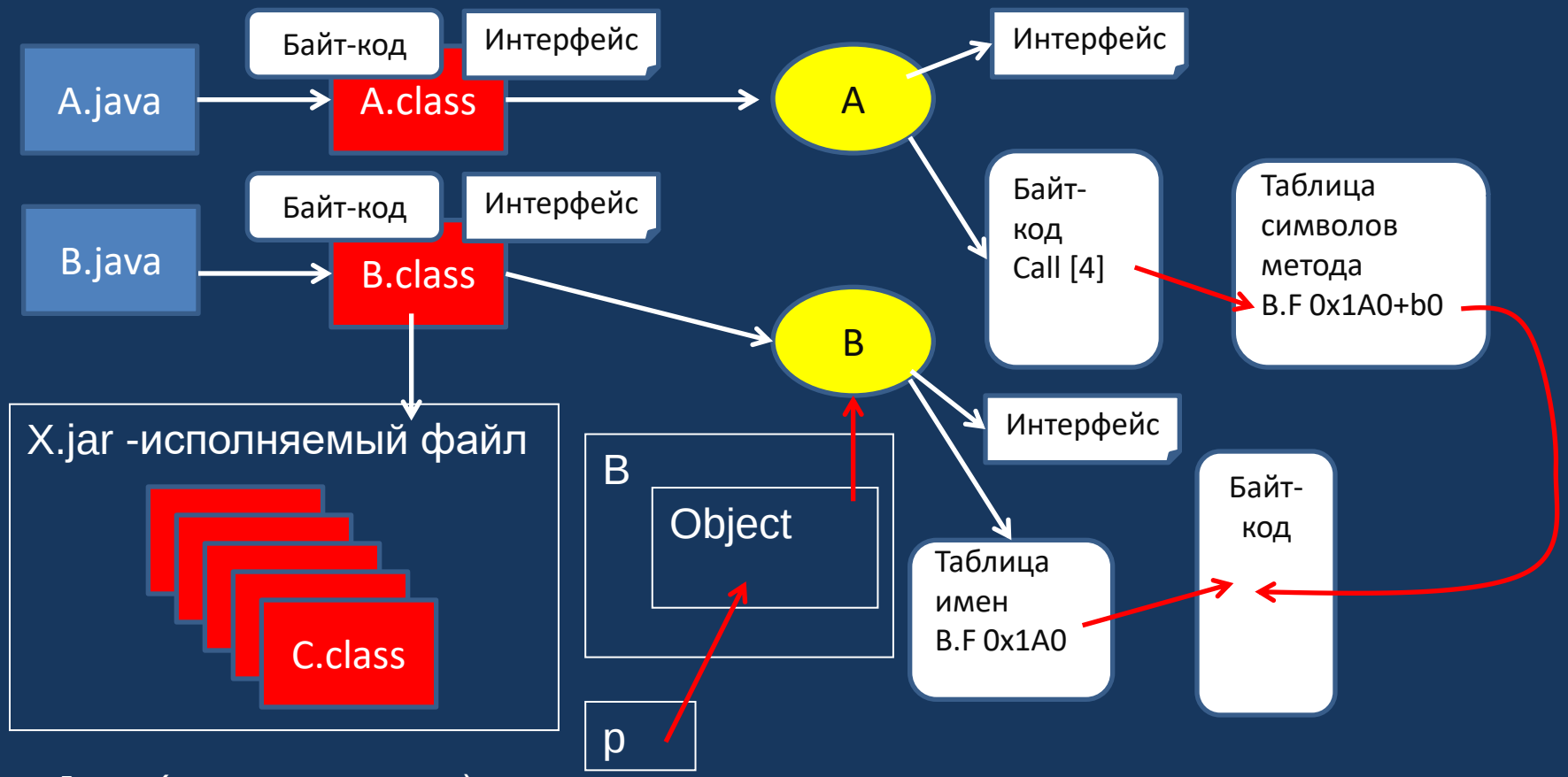


Команды IDE:

- **compile** – компилировать выбранный файл (модуль) в объектник
- **make** – компилировать измененные и зависимые от них файлы, собрать исполняемый файл
- **build** – компилировать все файлы и собрать исполняемый файл

Статическая компиляция и сборка программного кода и кода библиотек в единый исполняемый файл «кирпич», статическое связывание модулей.

Средства сборки программы на Java



Java (для сравнения):

- независимая компиляция «исходников» (исходные тексты классов) java-class (байт-код – двоичный код виртуальной машины JVM + описание программного интерфейса класса)
- runTime загрузка классов в процессе работы JVM перед обращением («ленивая», lazy), runTime связывание при обращении к объектам

Статические и динамические свойства в ЯП

Статическое свойство - устанавливается при компиляции и не меняется при исполнении

Динамическое свойство - меняется при исполнении (runTime)

Связывание - процесс установления соответствия между объектами и их свойствами в программе на формальном языке (операции, операторы, данные) и элементами архитектуры компьютера (команды, адреса).

Время связывания - фаза подготовки программы к выполнению, на которой происходит связывание

1. при определении языка
2. при реализации компилятора
3. во время трансляции
4. при компоновке (связывании)
5. во время загрузки программы
6. *во время выполнения программы, в том числе (RunTime):*

6.1. при входе в модуль (процедуру, функцию)

6.2. в произвольной точке выполнения программы, перед использованием

RunTime – свойство, действие, имеющее место **при выполнении программы**

Парадигмы программирования

Парадигма программирования —идеи и понятия, определяющие стиль написания и исполнения компьютерных программ. «Краеугольный камень».

Парадигмы, реализованные в языках:

Императивное программирование – последовательность команд с адресами операндов, программирование «от функции к функции».

Структурное программирование – технология для императивного подхода, Pascal (основан на идеях), большинство языков программирования.

Объектно-ориентированное программирование (ООП) – модуль – класс = данные(свойства) + функции(методы). Наследование, полиморфизм. Программа как система объектов. Языки программирования и технология разработки ПО в программной инженерии. *Cu++,Object C, Java, Scala, Kotlin*

Функциональное программирование – функция как объект данных (функция получает функцию как параметр, лямбда-выражения), «чистые» функции-методы, не изменяющие скрытых данных, конвейеры данных. *Scala*

Декларативное программирование – описание структуры результата, *SQL* – язык запросов к БД, сервер БД генерирует программу, которая создает выборку данных в соответствии с декларацией (фильтрация, упорядочение, группирование, связывание).

Парадигмы программирования

Парадигмы, реализованные в библиотеках , фреймворках :

Событийное программирование, реактивное программирование – потоки событий и описание реакции на них

Парадигмы, связанные с технологией разработки:

Структурное программирование – нисходящая пошаговая, «без goto» разработка алгоритма и связанных с ним данных.

Ad hoc (Как получится), «историческое» программирование – [cprog 3.2]

«Грязное» программирование – [cprog 3.5]

TDD (Test Driven Development) – разработка «тестами вперед»

Паттерны (шаблоны) программирования (кодирования, разработки) – общеизвестные приемы и варианты решения типовых задач, возникающих при проектировании программ: *взаимодействие модулей, универсальность (повторная применимость) кода, взаимодействие и управление в параллельных программах, структуры данных…*

Уровни освоения ремесла

«Учитель сказал: — Если хороший человек учил людей семь лет, Их можно посылать в сражение». Из Конфуция

1. Функция – 50-100 строк кода, язык программирования – синтаксис, алгоритмизация, структурное программирование, императив.
2. 3-7 сущностей (классов) – 200-1000 строк кода, ООП как инструмент.
3. 10-15 сущностей (классов), абстракции, интерфейсы, структуры данных, паттерны проектирования, ООП как технология разработки, фреймворки, 5000-10000 строк кода
4. >100 сущностей (классов) рефлексия, мета-данные, контроль трудоемкости, производительности и использования памяти, архитектурные паттерны, фреймворки, мат.аппарат и программные средства предметной области, >50000 строк кода.

P.S. Деление и критерии весьма условны

Контрольные вопросы

1. Почему представление программы в виде блок-схемы является наиболее «близкой по духу» к системе команд процессора ?
2. Что может обозначать фраза: размерность массива в Си является статическим свойством?
3. Как «чистота» компилируемого кода связана с возможностью его использования во встроенных системах, не имеющих собственной развитой ОС ?