

## Постановка задачи.

Конструируя новый компьютер, акцентируя внимание на слове «новый», разумно обратить внимание на то, что мы обычно называем компьютером. И, первая, самая сложная часть предмета обсуждения, это безусловно, процессор. За его особые заслуги его вначале даже называли «арифметико-логическим устройством». Выполнение арифметических операций очевидная задача, и она была решена достаточно быстро. Имеется еще такая штука как память. Она наличествует и у человека, и даже как-то связана с интеллектом. С арифметикой интеллект связывают гораздо реже. Но, вопрос как пользоваться памятью актуальный и не только для человека.

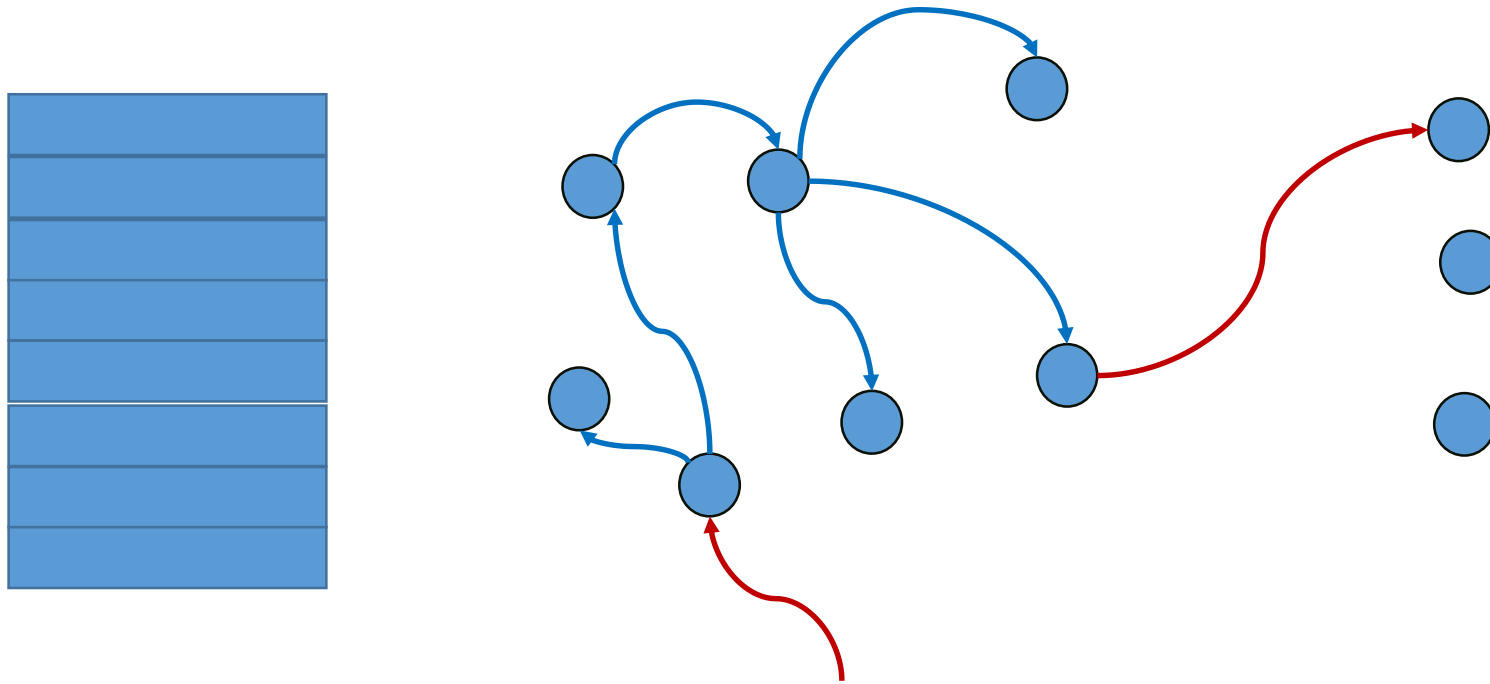
Память компьютера представляется массивом бит разделенных для удобства адресации на ячейки фиксированного размера (8-16-32 бита) и работа процессора заключается в адресации ячеек памяти и выполнения каких-то манипуляций с ними. Одни ячейки содержат команды, другие содержат данные с которыми работают команды. В общем ничего понятного если наблюдать на эту кашу бит со стороны. Тот, кто занимался отладкой ПО без инструментов должен меня понять. Потому идея структурировать данные в памяти в то что б они имели семантический смысл и структуру кажется не просто разумной, но и более высоким уровнем представления данных, чем массив двоичных данных. Итак, попробуем вместе сочинить простые и **единые** правила представления всех данных с минимальным количеством базовых понятий. Ибо это и есть синтаксис которым оперирует программист формулируя смысл понятиями, которые определяются правилами работы компьютера с данными. Такой подход избавляет от необходимости создавать пособия по программированию, а создавать понятия и систему программно. Одновременно сообщая их структуру и компьютеру и программисту. Опытный программист не обнаружит ничего нового. Мало ли какие правила можно сочинить. Их и так немало существует. Есть и хелпы и классы и объекты. Есть и события и подписки. В чем же фишка? Зачем сочинять все это по новому? А дело том, что вся текстовая красота после трансляции превращается в объектный код, затем этот код оптимизируется и компонуется и только после этого превращается в загрузочный в котором мало что остается от имен, хелпов и очень затруднительно узнать в этом загрузочном коде визуальную красоту заложенную программистом. Как минимум хелпы и имена исчезают. Порядок свойств и операторов совсем другой да и вообще что-то добавлено в код, что-то исчезло. Это приятно для оптимизации и выполнения машиной, но очень не приятно для вопросов отладки, верификации, протоколов обмена и визуализации. Я предлагаю оставить все как создал Бог в облике программиста! С хелпами, именами и порядком который задал программист в тексте программы и, само собой с новыми возможностями и приемами программирования. Понятно для реализации такой идеи понадобится умная адресация. Но все уже придумано. Так как структура одинаковая, то и относительная адресация будет работать всегда. Итак в бой!

Напомним что структура данных единая (**Концепт**). С необходимостью базовых понятия три:

1. **Concept**. –создание новой структуры данных (Концепта).
2. **Token** - единица размера памяти 4бита.
3. **Group**-объединение структур данных. В синтаксисе выражается скобками.

Из базовых понятий и единых правил строим строго формально все что образует систему. Правило

## Память. Структура и работа машины.



Слева, это представление данных в классическом компьютере. Справа граф с вершинами концептами, и ветвями-переходами подписками. Напоминает нейро сеть. Ну, и так же выполняется. Можно рассматривать вершины как перцептроны, а ветви-адресации как синапсы со значениями – параметрами в качестве значений.

Структура концепта. Компоненты концепта.

**Def.** Байт. Ниже значение 1 каждого бита признак наличия соответствующей не обязательной компоненты.

**Class.** Байт. Обязательный компонент.

Бит Def 0. **Name.** Name. Не обязательный компонент.

Бит Def 1. **Help.** Текст. Не обязательный компонент.

Бит Def 2. **Subscribe.** Подписки. Группа концептов. Не обязательный компонент.

Бит Def 3. **Организация.** Метод группирования и данные. Не обязательный компонент.

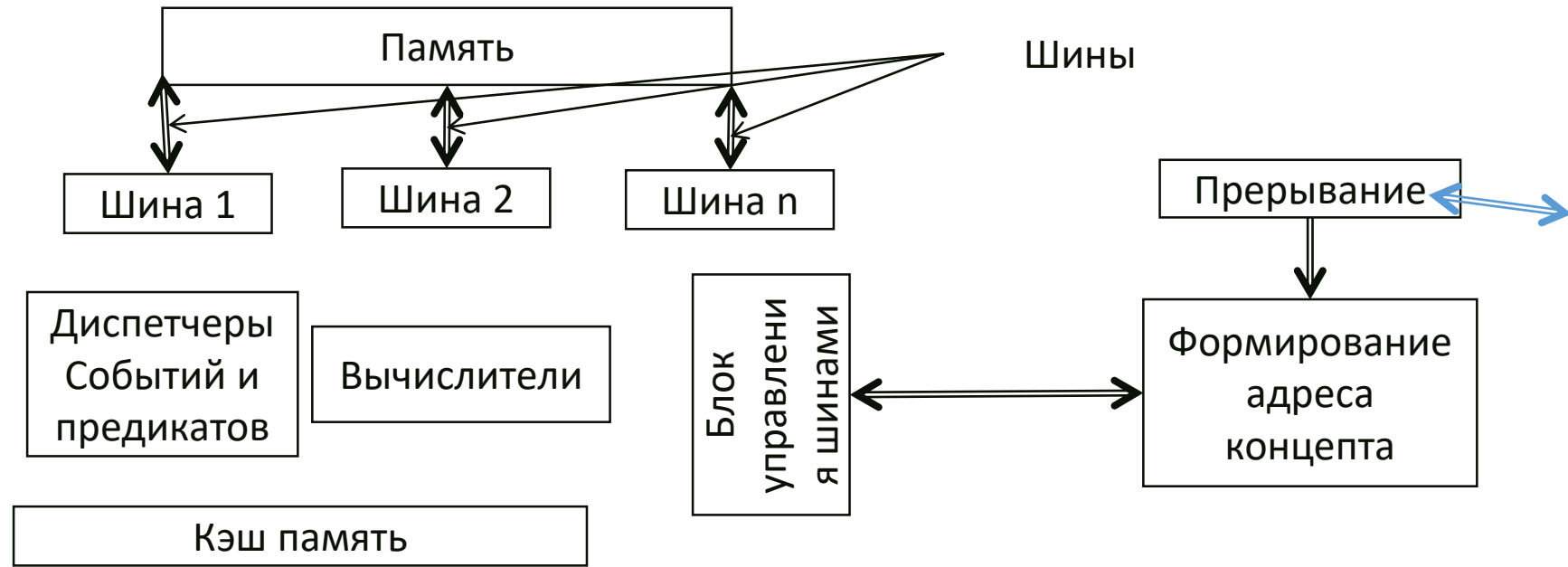
Бит Def 4. **Attributes/Property.** Свойства. Группа концептов. Не обязательный компонент.

Бит Def 5. **Event. События.** Группа концептов. Не обязательный компонент.

Бит Def 6. **Method. Методы.** Группа концептов запускаемых по событию инициализации.

Бит Def 7. **Child. Вложения.** Группа концептов. Не обязательный компонент.

# Архитектура компьютера. Адресация. Начало работы. Инициализация.



1. Адресация существует 4-х видов. Для чтения, записи, выполнения и инициализации. Каждый вид адресации порождает свою группу событий и выполняется по разному. Общий момент- адресация.
2. На блок управления шинами поступает адрес.
3. Выбирается свободная шина и к ней привязывается стек и параметр, если он есть.
4. По адресу производится чтение класса концепта (экспресс адрес) и байт описания концепта (**Def**).
5. Если в байте описания взведен бит наличия подписок, то запускается диспетчер обработки событий и понятий. Если при проверке подписанных событий обнаруживается истинное, то запускается переход по подписке. И выполняется подписка.
6. При отсутствии подписок при адресации для чтения выполняется чтение, при записи-запись, при выполнении запускается вычислитель для выполнения, при инициализации создается новый концепт. Более подробно выполнение каждого концепта описана ниже. Особое внимание уделено выполнению групп.

# Базовые понятия. Синтаксис. Структура

Напомним что структура данных единая (**Концепт**). С необходимостью базовых понятия три:

1. **Concept**. –создание новой структуры данных (Концепта).

2. **Token** - единица размера памяти 4бита.

3. **Group**-объединение структур данных. В синтаксисе выражается скобками.

Из базовых понятий и единых правил строим строго формально все что образует систему. Правило первое –структура и состав концепта.

Структура концепта. Компоненты концепта.

**Def.** Байт. 1 каждого бита признак наличия соответствующей не обязательной компоненты.

**Class.** Байт. Обязательный компонент.

Бит Def 0. **Name**. Name. Не обязательный компонент.

Бит Def 1. **Help**. Текст. Не обязательный компонент.

Бит Def 2. **Subscribe**. Подписки. Группа концептов. Не обязательный компонент.

Бит Def 3. **Организация**. Метод группирования и данные. Не обязательный компонент.

Бит Def 4. **Attributes/Property**. Свойства. Группа концептов. Не обязательный компонент.

Бит Def 5. **Event. События**. Группа концептов. Не обязательный компонент.

Бит Def 6. **Method. Методы**. Группа концептов запускаемых по событию инициализации.

Бит Def 7. **Child. Вложения**. Группа концептов. Не обязательный компонент.

Синтаксис: *Имя класса Имя концепта Атрибуты класса Подписки на события класса*

**Byte** | 0 *#Счетчик цикла.»*

**Concept** *Type #Определение концепта Type.»*

*:: Token Size #Определение динамического свойства размер.»*

**Type** *Token Size 0 #Определение типа Byte размером 2 токена.»*

*:: Token "" #Атрибут без имени для хранения значения данного типа.*

**Type** *Byte Size 1 #Определение типа Byte размером 2 токена.»*

*:: Token {}<sup>Size</sup> "" #Атрибут без имени для хранения значения данного типа.»*

**Graphics** *Point :: Int16 (x, y) #Определение концепта с двумя атрибутами.»*

**Point** *Position 20;40 #Агрегатное присвоение значений атрибутам.»*