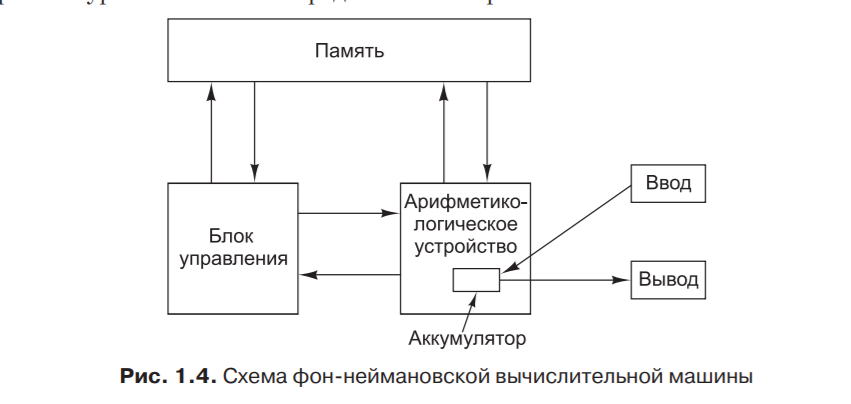
# Билет 2.1 – Архитектура Фон-Неймана и её альтернативы

Под архитектурой фон Неймана могут пониматься разные вещи. Ныне в большинстве случаев имеется в виду такое устройство компьютера, что программы и данные хранятся в одной и той же памяти и внешне неотличимы. Изначально термин предполагал (и Скаков ожидает от вас именно такую интерпретацию) архитектуру, удовлетворяющую следующим принципам (принципы приведены в порядке и трактовке Скакова).

Кстати говоря, вычислители, соответствующие принципам фон-Неймана называются ВНЕЗАПНО машиной фон-Неймана.



## Принципы Фон-Неймана

### Двоичная система счисления

И данные, и программы хранятся в виде последовательностей из 0 и 1, то есть в виде чисел в двоичной системе счисления.

### Адресность памяти

Основная память состоит из последовательно занумерованных уникальным номером ячеек, что позволяет в конкретно взятый момент времени получить доступ к любой ячейке памяти.

### Однородность памяти

Код программ и обрабатываемые данные хранятся в одной памяти, то есть в разные моменты времени в одном и том же месте памяти могут храниться данные, программа или адрес. Единственное, что их будет отличать друг от друга – способ их интерпретации вычислителем.

### Программное управление

Всё, что делает ваш кудахтер должно определяться программой, а не железом.

### Последовательное выполнение

Команды должны выполняться последовательно, друг за другом.

## Альтернативы

### Система счисления

Троичная, десятичная, какая угодно. На практике использовались троичная и десятичная система счисления. Но, троичная и десятичная система требуют более сложного аппаратного устройства (Вследствие того, что прои двоичной системе аппаратуре нужно различать всего два состояния, которые кодируются напряжением, а в случае систем счисления более высокого порядка, таких состояний больше), даже несмотря на то, что теоретически, 3 – более выгодное основание системы счисления для вычислений. (Если быть точнее, то самое выгодное – e).

Почему же тогда сейчас распространена двоичная система, а не троичная?

Во-первых, как уже было сказано, двоичная логика проще в реализации. Во-вторых, просто исторически так сложилось, на использование двоичной системы завязано много всякого и просто перейти с двоичных вычислений на троичные не получится.

### Адресность памяти

Принцип устройства другого абстрактного вычислителя – машины Тьюринга. Она представляет из себя черный ящик и бесконечную ленту, по которой может переходить этот черный ящик. При этом абсолютной адресации на ленте нет, только относительная, отсчитывая от текущей активной ячейки.

Плюсы принципа адресности очевидны – у нас в любой момент времени есть доступ к любой ячейке памяти, что, как минимум, повышает быстродействие.

### Однородность памяти

В противовес фон-Нейману существует Гарвардская архитектура, которая отличается тем, что данные и код хранятся в разных физических устройствах, причем доступ к этим устройствам осуществляется по физически разным шинам. Такая архитектура позволяет значительно увеличить производительность системы, вследствие того, что одновременно могут идти запросы и к хранилищу кода, на получение следующих команд, и к хранилищу данных. (Более подробно -- пункт “Гарвардская архитектура”)

### Программное управление

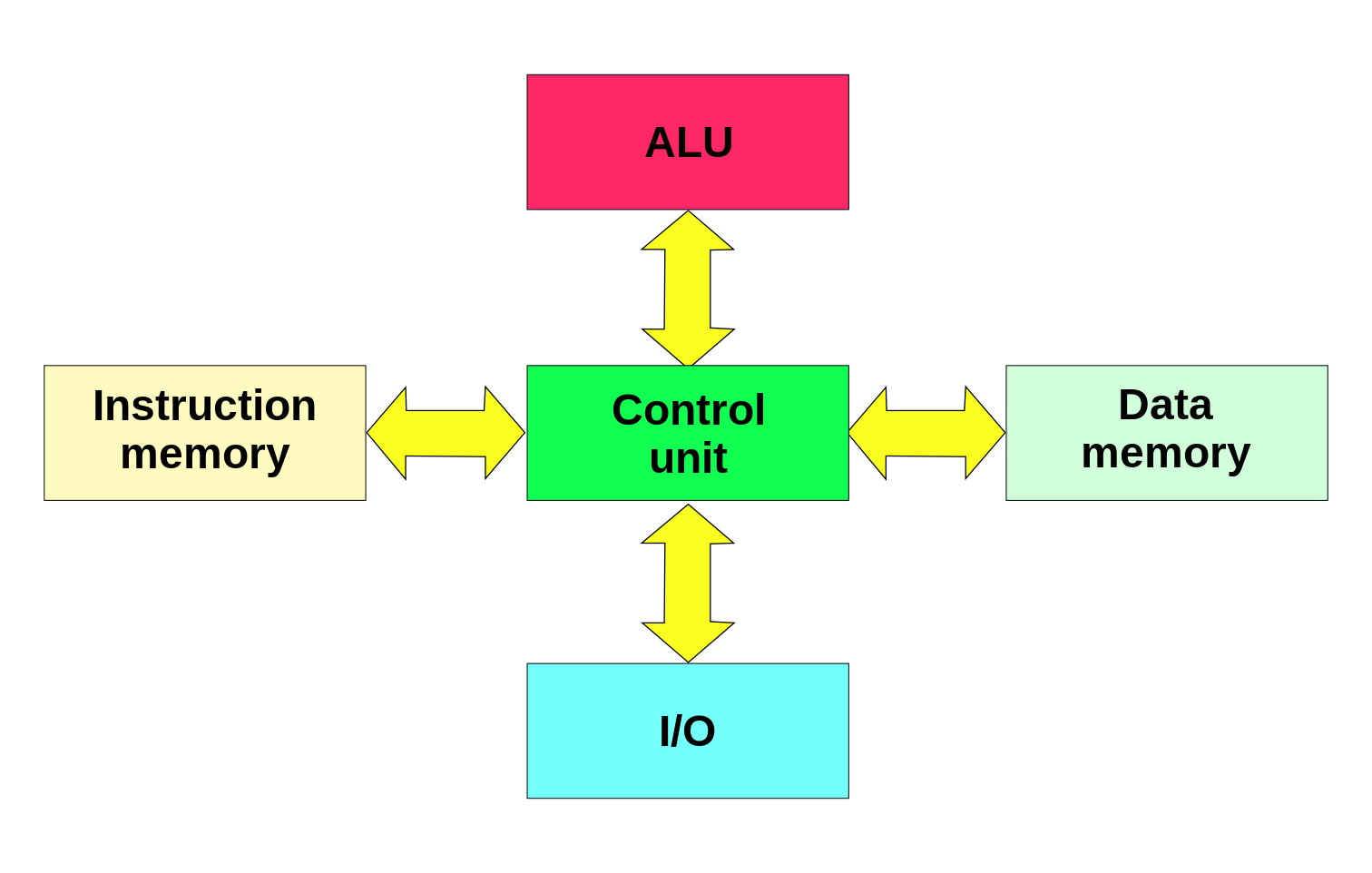
Альтернатива – аппаратное управление. Но у него есть очевидные минусы, которые перекрывают очевидные плюсы, а именно, если ты один раз построил кудактер, то он только и будет уметь считать то, что считал изначально. Чтобы поменять программу нужно будет менять железо. Плюсы же такого подхода – он проще в реализации, не нужно всяких там ваших кэшей и прочего. Однако, иногда применяется аппаратное управление для экономии энергии. Например, аппаратные H.264 энкодеры в мобильных процессорах.

### Последовательное выполнение

Другие варианты – параллельное выполнение. Но такие системы сильно сложнее. Альтернатива в реальном мире – суперскаляр.

Плюсы последовательных систем -- оди дешевле и проще.

## Гарвардская архитектура



Принципы:

* Двоичная система счисления
* Адресность памяти
* Неоднородность памяти
* Программное управление
* Последовательное выполнение

Таким образом, отличается только пункт про память. То есть единственное отличие Гарвардской архитектуры от фон Неймановской в том, что память разделена на память команд и память данных. Это позволяет увеличить быстродействие за счет раздельных шин команд и данных, улучшить безопасность (программы не могут создавать исполняемый код “на лету”). Однако, такая архитектура сложнее в реализации, в том числе потому, что необходимо рассчитывать, сколько памяти отдать под команды, а сколько под данные.

### Сравнение Гарвардской и фон Неймана

Итак, Гарвардская архитектура:

* Быстрее, за счет раздельных шин данных и кода.
* Код программ защищен от вмешательства другими программами, а именно, когда память не разделена, то есть вероятность, что некоторая программа перезапишет область, хранящую другой исполняемый код и таким образом будет исполнен нежелательный код.
* Необходимо рассчитывать, сколько памяти выделить под код, а сколько под данные.
* Программы не могут генерировать исполняемый без каких-либо дополнительных телодвижений код. Следовательно, не может существовать этих ваших JVM, pypy, .NET и прочих вкусностей.
* Дороже, ибо необходимы разные входы для шины команд и шины данных на процессоре.

Фон-Нейман:

* Возможность динамически менять исполняемый код
* Проще в реализации
* Потенциально возможны ситуации, когда программа неаккуратно меняет значения в памяти, и, тем самым, выполняется нежелательный код.

### Модифицированная гарвардская архитектура

Схема раздельной памяти в гарвардской архитектуре имеет очевидный недостаток в виде высокой стоимости. При разделении каналов передачи команд и данных на кристалле процессора последний должен иметь почти вдвое больше выводов, так как шина адреса и шина данных составляют основную часть выводов микропроцессора. Способом решения этой проблемы стала идея использовать общие шину данных и шину адреса для всех внешних данных, а внутри процессора использовать шину данных, шину команд и две шины адреса. Такую концепцию стали называть модифицированной гарвардской архитектурой.

## FAQ

### Почему машина фон Неймана является абстрактной?

Потому что описание машины фон Неймана слишком размыто и содержит множество неточностей. Это было сделано намеренно, чтобы не ограничивать инженеров в будущем.

### Какое узкое место в архитектуре фон Неймана?

Совместное хранение команд и данных (принцип однородности памяти) сильно ограничивает быстродействие системы, потому что процессор вынужден ждать два раза для того, чтобы исполнить команду -- пока придёт сама команда и пока придут данные. Этот недостаток устранен в Гарвардской архитектуре.

### Придерживаются ли современные системы принципов фон Неймана?

В целом -- да, но есть нюансы. Например, вне процессора имеет место быть принцип однородности памяти, однако, внутри, он не всегда выполняется, например, кэши первого уровня зачастую разделены на кэш данных и кэш команд.

Всякие ухищрения в виде суперскаляров не придерживаются принципа последовательного выполнения (но это не точно).

Иногда используется аппаратное управление (пример -- выше по тексту)