1 Принципы Фон Неймана

1. Двоичная система счисления
   Альтернатива: система счисления по основанию 3 (на самом деле даже лучше, т.к. ближе к "идеальной"СС по основанию e). Не прижилась т.к. слишком много всего уже работает на двоичной.

2. Адресность памяти
   Доступ к любой ячейке памяти осуществляется за одинаковое время.
   Альтернатива: стековая архитектура, машина Тьюринга.

3. Однородность памяти
   Одна память для команд и для данных.
   Альтернатива: отдельная память для команд, отдельная для данных (Гарвардская архитектура).

4. Программное управление
   Альтернатива: аппаратное управление

5. Последовательное выполнение
   Команды должны выполняться последовательно.
   Альтернатива: многопроцессорные архитектуры и (внезапно) конвеер.

2 Сравнение гарвардской и фон Неймана

2.1 Гарвардская

2.1.1 Преимущества

- Быстрее. Раздельные шины под данные и инструкции позволяют читать инструкции одновременно с запросами к памяти.
- Код программ защищен от вмешательства других программ.
- Память под данные и под инструкции может иметь разные характеристики (разный рамер шины, частоту и т.д.)

2.1.2 Недостатки

- Нужен более сложный и дорогой контроллер памяти
- В два раза больше шин тоже дорого
- Не можем использовать "лишнюю"память данных под инструкции и наоборот
- С точки зрения программиста: не можем модифицировать код. Память команд read-only.

2.2 фон Нейман

2.2.1 Преимущества

- Программист может использовать всю доступную память, разделяя инструкции и данные как удобно.
- Одна шина вместо двух - проще и дешевле
- Контроллер памяти тоже проще
- Обращаемся к инструкциям и к данным одинакого

2.2.2 Недостатки

- Одна шина это всё-таки грустно. Структурные хазарды у конвеера и все такое :(
- Одна программа может перезаписать другую

3 Модифицированная гарвардская архитектура

Существует несколько модификаций гарвардской архитектуры.

3.1 Раздельный кэш

Используем раздельный кэш первого уровня L1i под инструкции, L1d под данные.

3.2 Доступ к памяти команд как к данным*

В этой модификации существуют команды, которые позволяют считывать константы из инструкций в регистры. Пример такой архитектуры - AVR8. Но все равно это головная боль для программиста, почти как труъ-гарвардская

3.3 Чтение инструкций из памяти данных*

В общем, такое можно провернуть, но доступ к константам всё равно грустный. Тоже почти труъ-гарвард.

4 Почему двоичная система счисления?

Потому что так исторически сложилось. Изначально пытались сделать десятичные компьютеры (аналитическая машина Чарльза Бэббиджа, например). Но это точно неудобно. Оптимальной системой счисления является e. Тройка по идее ближе к e, но уже слишком много софта написано для двоичной, а железка без софта никому не нужна. Компьютеры на троичной системе счисления, кстати, вполне существовали. Пример: советский компьютер Сетунь.

5 Источники информации

1. Википедия

2. Какой-то сайтик с ЗАЙКОЙ