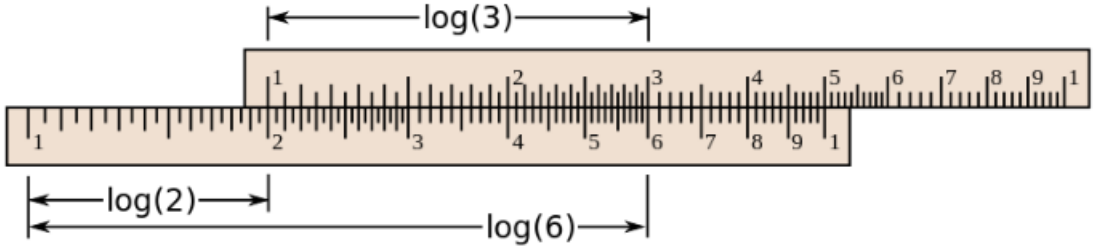
Краткая история развития компьютеров: XVII–XVIII вв.

Логарифмическая линейка – механический аналоговый компьютер. Первые линейки могли выполнять операции * и /. Позднее их «научили» вычислять тригонометрические функции, возводить в степень, извлекать корни.



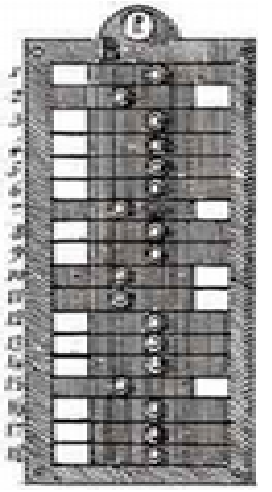
$\log_x(3*2) = \log_x(3) + \log_x(2)$

$\log_x(6/3) = \log_x(6) - \log_x(3)$

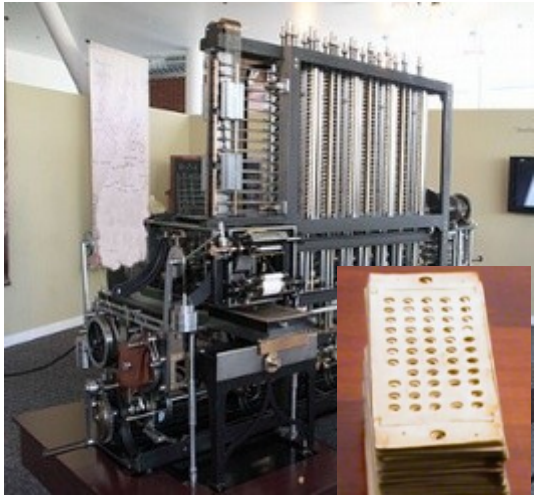
«Считающие часы» Шиккарда и арифометр Лейбница выполняли операции **+, – , *, /**

Краткая история развития компьютеров: XIX век

- Жаккар придумал программировать рисунок для ткацкого станка с помощью **перфокарт**.
- «Интеллектуальные машины» Семёна Корса́кова решали задачи поиска, классификации и сравнения в перфокарточных базах данных.

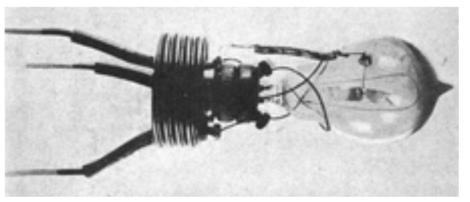


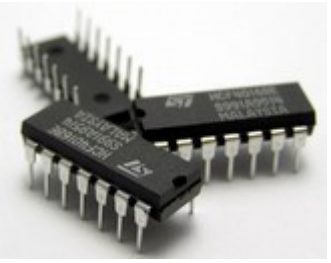
- «Разностная машина» Бэббиджа выполняла аппроксимацию и табуляцию функций с помощью многочленов:
  $\sin(x) \approx a*x^3 + b*x^2 + c*x + d$ (как подобрать a, b, c и d?).
- Концепция «Аналитической машины» Бэббиджа стала прообразом современных компьютеров («мельница» выполняла команды, записанные на перфокартах, используя «склад» для хранения результатов).



Краткая история развития компьютеров: XX век

- Вплоть до 1980-х годов продолжали развиваться аналоговые компьютеры: интегрирование, решение дифф. уравнений ($y'_x = x^2 + 5*y$) и многое другое.
- Параллельно возникали следующие поколения цифровых компьютеров:
  1. **Первое** (1940-50): электронные лампы
  2. **Второе** (1950-60): транзисторы
  3. **Третье** (1960-70): интегральные схемы
  4. **Четвёртое** (1970-н.в): микропроцессоры (МП)
  5. Пятое (будущее):
     - ? взаимодействие неограниченного числа МП ?
     - ? взаимодействие с человеком напрямую ?
     - ? искусственный интеллект, нанотехнологии ?
     - ? квантовые и молекулярные компьютеры ?





Принципы Фон-неймановской архитектуры

Бёркс, Голдстайн и Фон Нейман в 1946 г. в книге «*Предварительное рассмотрение логического конструирования электронного вычислительного устройства*» описали принципы, на которых основаны большинство современных компьютеров.

- Принцип двоичного кодирования
- Принцип однородности памяти
- Принцип адресуемости памяти
- Принцип жесткости архитектуры
- Принцип программного управления

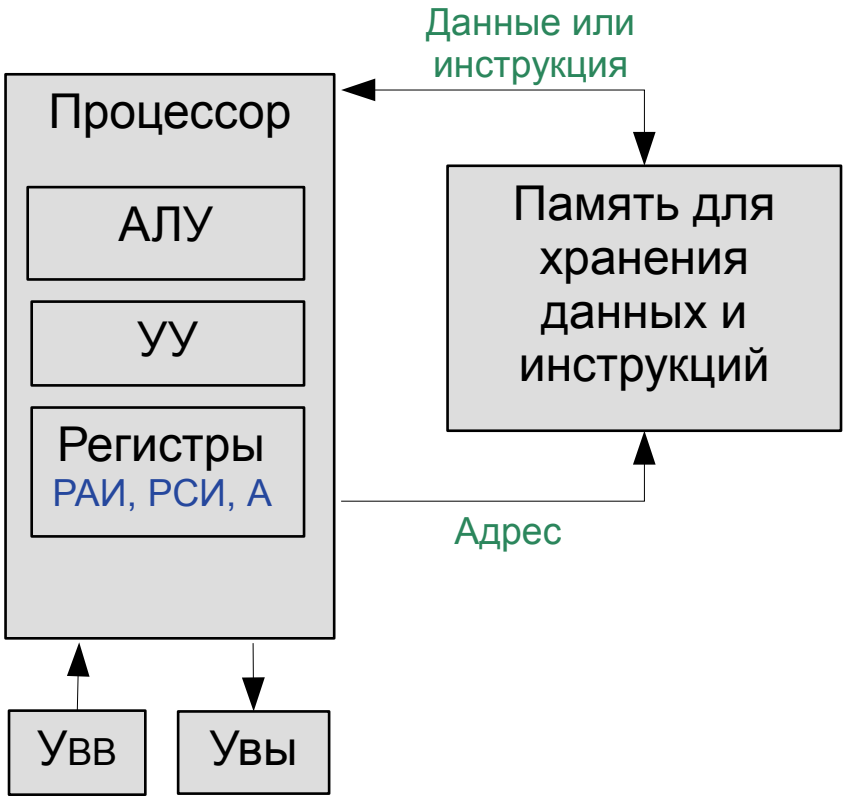
Джон фон Нейман (1903-1957)

Принцип программного управления

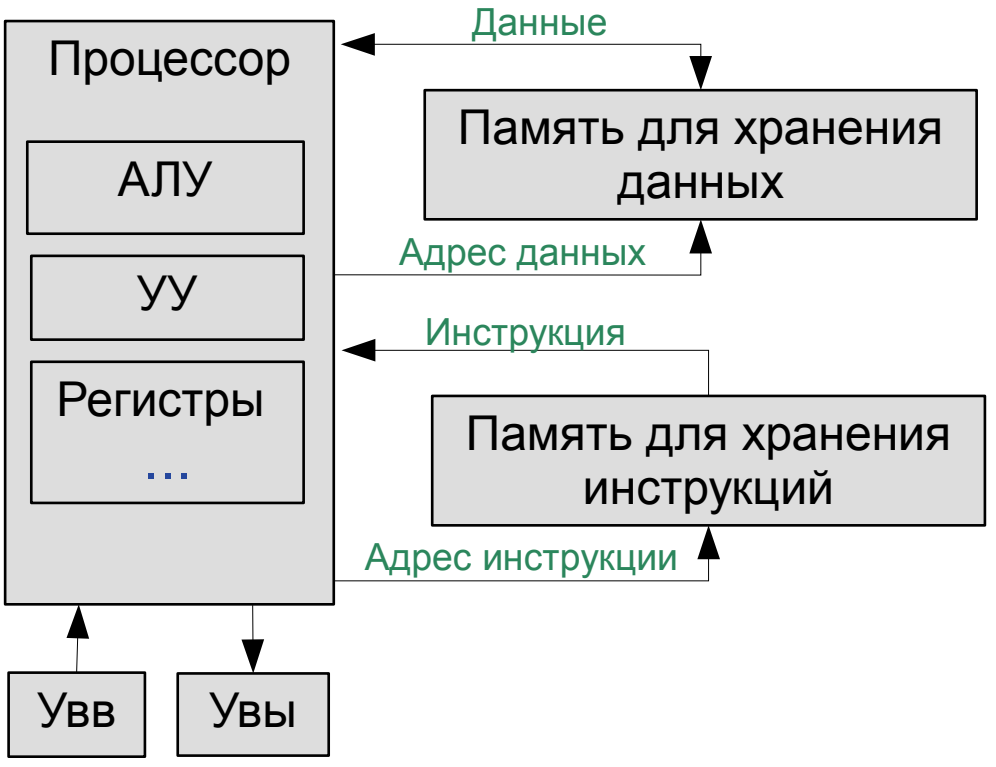
1. При запуске программы процессору сообщается адрес первой инструкции программы. Этот адрес заносится в регистр адреса инструкции (РАИ).
2. Используя адрес в РАИ, процессор копирует из памяти нужную инструкцию в специальный регистр содержимого инструкции (РСИ).
3. Процессор выполняет инструкцию, содержащуюся в РСИ.
4. Процессор вычисляет адрес следующей инструкции программы одним из двух способов (для последовательного и непоследовательного выполнения):
   - увеличивает РАИ на длину только что выполненной инструкции или
   - копирует в РАИ адрес, явно указанный в инструкции перехода.
5. Перейти к шагу 2. Далее программа управляет «сама собой», продвигаясь по заданным программистом инструкциям в нужной последовательности.

Принцип однородности памяти: принстонская архитектура

Фон-неймановская архитектура



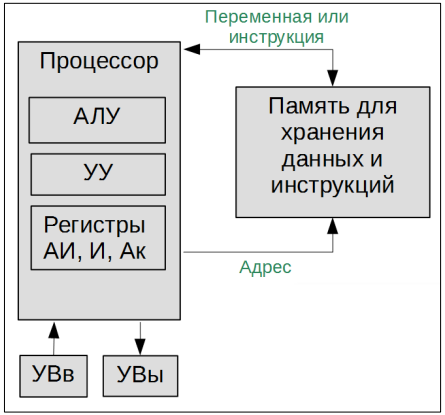
Гарвардская архитектура



Сравнение гарвардской и принстонской архитектур

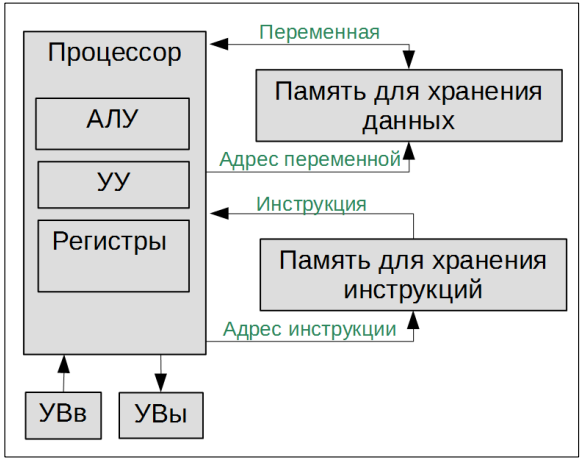
Преимущества принстонской архитектуры (ПА)

1. ПА схемотехнически проще и дешевле.
2. Система команд ПА проще и меньше.
3. ПА позволяет выполнять самомодификацию программы и генерировать программу на ходу (JIT-компиляция).
4. ПА позволяет хранить программы на внешних устройствах.



Преимущества гарвардской архитектуры (ГА):

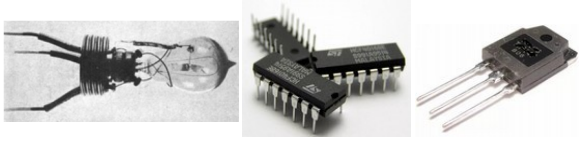
1. У ГА более высокая производительность (можно одновременно читать данные для текущей команды и считывать следующую команду).
2. ГА устраняет главное узкое место ПА: связку ЦП-ОП.
3. В ГА можно асимметрично управлять обменом инстр. и д-ми.
4. Программу без самомодификаций намного проще документировать, отлаживать и защищать от НСД.



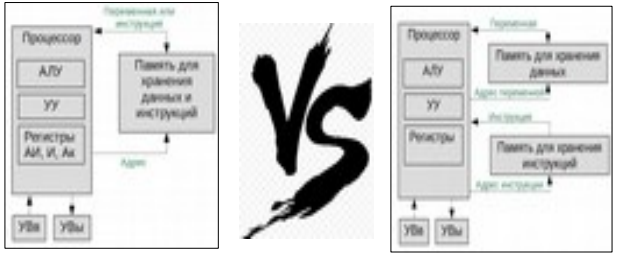
Понятие архитектуры компьютера

Архитектура компьютера – набор правил, которые описывают особенности работы компьютера. Архитектуру обычно рассматривают на нескольких независимых уровнях:

1. Физический уровень
   (ламповые, транзисторные, ...)



2. Микроархитектурный уровень
   (составные элементы компьютера и связь между ними).



3. Уровень системы команд процессора
   (количество и назначение команд процессора, их операнды).

4. Уровень операционной системы
   (многозадачность, вирт. память, файловая систем, сеть).

5. Уровень языка программирования (ЯП)
   (компиляторы; низкоуровневые и высокоуровневые ЯП)

Архитектура компьютера: уровень микроархитектуры

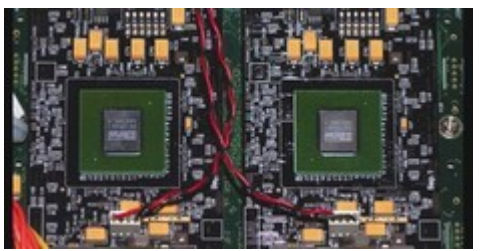
Разрядность интерфейсов (шин), регистров и машинных слов:

8-, 16-, 32-, 64-, 128-разрядные

Количество вычислителей:

однопроцессорные, многопроцессорные,

одноядерные, многоядерные.

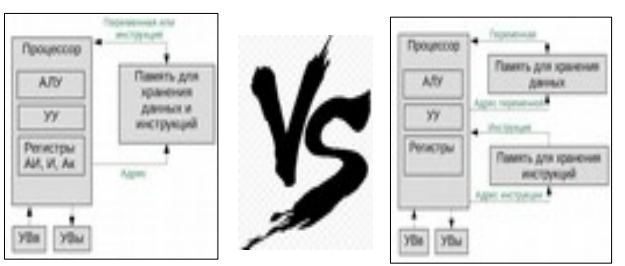


Однородность памяти команд и данных

принстонская архитектура,

гарвардская архитектура,

гибридная.



Архитектура компьютера: уровень системы команд

Определение: система команд процессора содержит инструкции, которые реализованы на физическом уровне в виде транзисторов.

- Архитектура системы команд
  - Аккумуляторная архитектура 1950-е
    - 
  - Регистровая архитектура 1960-е
    - Сложный набор команд **CISC** (197x – н.вр.)
    - Сокращенный набор команд **RISC** (198x – н.вр.)
    - Команды с явным параллелизмом **VLIW, EPIC** (198x – н.вр.)
  - Стековая архитектура 1960-е
    - Минимальный набор команд **MISC** (200x – н.вр.)

Примечание: существуют и другие виды микроархитектур, не указанные на схеме!

Сравнение архитектур CISC, RISC, EPIC(VLIW)

Пусть в С-программе написано выражение «*a = b + c + d + e;*». С помощью каких команд процессора компьютер рассчитает значение *a*?

RISC

```
Обнулить   r2
Прочитать r1, b
Сложить    r2, r1
Прочитать r1, c
Сложить    r2, r1
Прочитать r1, d
Сложить    r2, r1
Прочитать r1, e
Сложить    r2, r1
Записать   r2, a
```

MISC

```
Push b
Push c
Push d
Push e
Add
Add
Add
Pop a
```

CISC

```
Сложить   b, c, a
Инкремент  a, d
Инкремент  a, e
```

EPIC (VLIW)

```
Сложить b, c, r1, d, e, r2
Сложить r1, r2, a, nop, nop, nop
```

Отличительные признаки архитектур команд

- число, разновидности и сложность команд
- число и разновидности операндов
- совмещение выполняемой операции с обращением в память (или явные операции чтения/записи в память)
- длина команды (постоянная, плавающая)
- количество доступных регистров (это косвенный признак!)

Почему RISC победил CISC

Большинство современных процессоров – это RISC, либо “CISC-поверх-RISC”.

1. Реализация системы команд RISC-процессора требует меньше транзисторов

- можно снизить энергопотребление
- можно повысить тактовую частоту
- можно увеличить размер кэш-памяти

2. На практике оказалось, что при использовании CISC-компьютеров доля сложных интеллектуальных команд при выполнении программы не превышает 10-20%, а остальные команды вполне сопоставимы с RISC-аналогами.

3. RISC-программы лучше приспособлены для упреждающего выполнения, конвейерной обработки и других видов оптимизации.

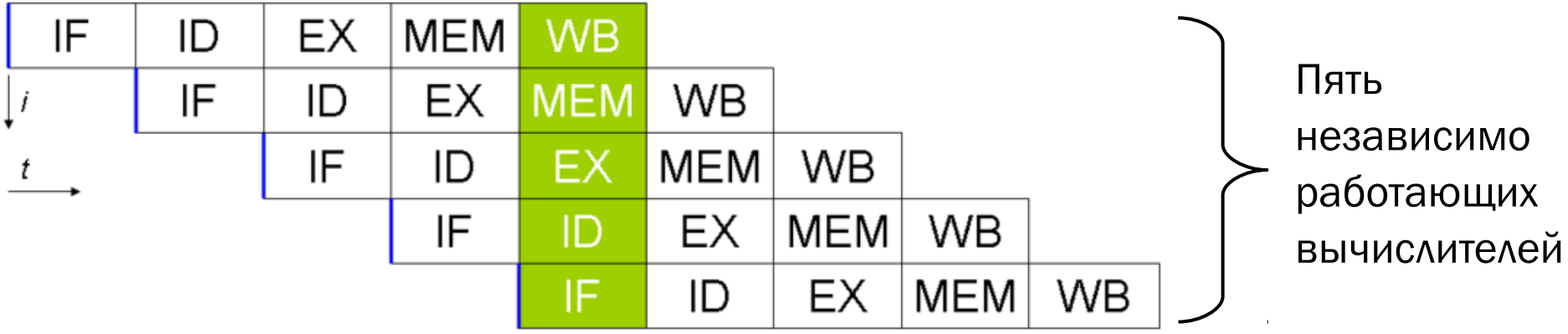
Этапы выполнения команд процессором

Полный цикл выполнения типичной команды процессора обычно включает в себя несколько этапов. Пример: команда инкремента «inc b»

1. Вычисление адреса команды.
2. Выборка команды.
3. Декодирование команды.
4. Вычисление адреса операнда.
5. Выборка операнда.
6. Выполнение заданной операции.
7. Запись результата операции.

Конвейерная обработка команд

Одним из наиболее распространённых способов увеличения скорости обработки команд является конвейерная обработка команд.
Пример 5-уровневого конвейера в реальном RISC-процессоре:



Обозначения этапов конвейера (см. предыдущий слайд):
IF = 1+2, ID = 3+4+5, EX = 6, MEM+WB = 7

Память в компьютере: характеристики систем памяти

1. **Месторасположение**
   - процессорная память (на общем кристалле с ЦП: регистры, кэш-память 1-го уровня)
   - внутренняя память (на материнской плате: ОП, кэш-память 2-го и послед. уровней)
   - внешняя память (жёсткий диск, флеш-накопитель, CD).
2. **Ёмкость** – число байт, которое можно хранить на ЗУ.
3. **Единица пересылки** – количество бит, передаваемых по линиям шины данных параллельно и одновременно (ширина шины данных)
4. **Физический тип**
   - полупроводниковая память (ОП, кэш-память, SSD)
   - магнитная память (жёсткий диск и лента в стриммере)
   - оптическая память (CD, DVD, Blu-ray).
5. **Особенности энергопотребления:** энергозависимость, потребляемая мощность и др.
6. **Стоимость** – стоимость хранения одного бита информации.

Характеристики систем памяти (продолжение 1)

7. **Метод доступа к данным** (данные хранятся в виде последовательности блоков некоторого размера; каждый блок имеет адрес)

- **Последовательный доступ.** Для доступа к нужному блоку нужно прочитать все промежуточные блоки (стримеры).

- **Прямой доступ**. Блоки сгруппированы в кластеры. Доступ к началу кластера происходит относительно быстро, а внутри блока осуществляется медленный последовательный доступ (DVD, HD).

- **Произвольный доступ**. Время доступа к блоку не зависит от его адреса и одинаково для всех блоков (в ОП блоки называются ячейками и имеют адресуемый размер 1 байт).

- **Ассоциативный доступ**. Поиск ячеек происходит не по адресу, а по содержимому. Может найтись несколько блоков, удовлетворяющих поисковому критерию. С поисковым шаблоном сравниваются все блоки одновременно, независимо от ёмкости памяти (кэш).

Характеристики систем памяти (продолжение 2)

8. Быстродействие

- **Время доступа $T_д$ (мс)** – время, прошедшее от момента появления запроса на чтение/запись минимального блока памяти до момента его фактического получения/размещения.
- **Длительность цикла памяти $T_ц$ (мс)** – минимальное время между двумя последовательными обращениями к памяти. Время $T_ц$ включает $T_д$, а также некоторую дополнительную «служебную» задержку.
- **Скорость чтения/записи S (бит/с)** – средняя скорость, с которой данные передаются в память или из неё при чтении/записи нескольких блоков. S зависит от разрядности шины, а также значений $T_д$ и $T_ц$.

9. Эксплуатационные характеристики: надёжность, температурный режим и т.д.

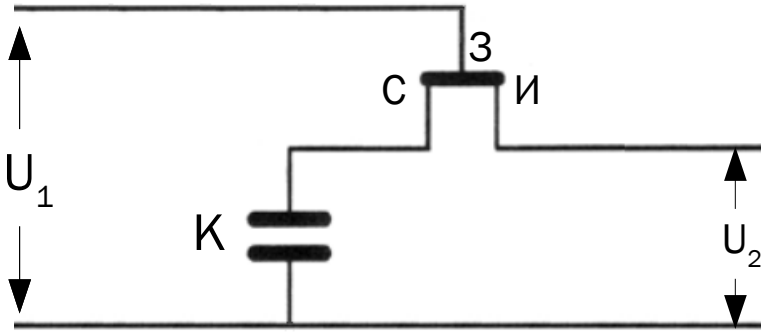
Физическое устройство памяти

- Память состоит из адресуемых ячеек (типичный размер: 1–128 бит).
- Ячейки состоят из физических запоминающих элементов, способных хранить 1 бит.
- Запоминающий элемент может находиться в одном из двух устойчивых состояний:
  - конденсатор заряжен/разряжен;
  - транзистор в проводящем/непроводящем состоянии;
  - полупроводниковый материал имеет высокое/низкое сопротивление и т.п.
- Одно из таких физических состояний интерпретируется как 0, другое – как 1.

Триггер – пример запоминающего элемента, состояние которого можно изменить, подав нужное напряжение на один из его **входов** (т.е. входящих в триггер проводов). Текущее состояние триггера можно «прочитать», измерив уровень напряжения на его **выходе** (т. е. выходящем из триггера проводе).

Физическое устройство оперативной памяти

1 бит оперативной памяти



З – затвор транзистора
И – исток транзистора
С – сток транзистора
К – конденсатор
U_1 – сигнал начала чтения/записи (напряжение)
U_2 – считываемый/записываемый сигнал (0 или 1)

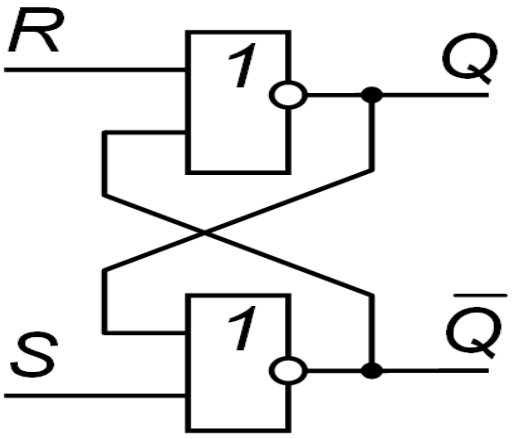
Итого: один конденсатор, один транзистор и резиторы

Особенности:

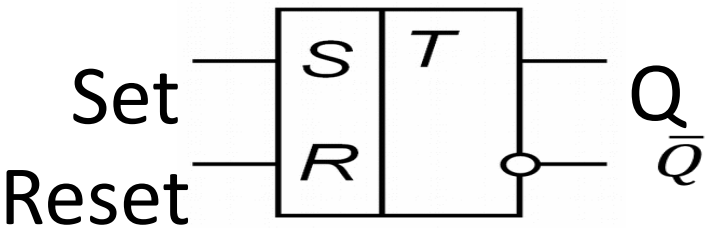
+ дешево, занимает мало места
– конденсатор необходимо периодически подзаряжать
– зарядка/разрядка конденсатора – это медленный процесс

1 бит кэш-памяти

RS-триггер: схема

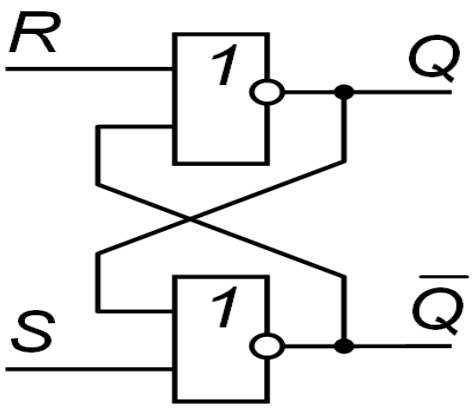


RS-триггер: обозначение



Итого: четыре транзистора и резисторы (2 элемента ИЛИ-НЕ)

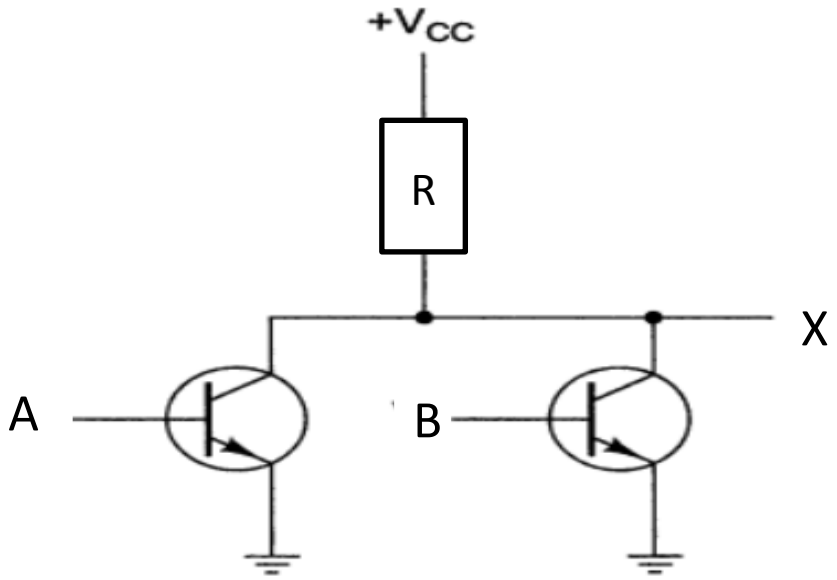
Физическое устройство кэш-памяти (продолжение 1)

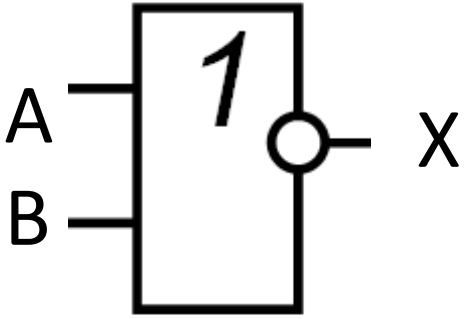


R	S	Q
0	0	0 или 1
0	импульс 0→1→0	1
импульс 0→1→0	0	0
0	1	0
1	0	1
1	1	0 или 1

Физическое устройство кэш-памяти (продолжение 2)

Электрическая схема элемента ИЛИ-НЕ





A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Итого: 2 транзистора, сопротивление

Принцип локальности памяти

- С высокой вероятностью адрес следующей команды программы следует непосредственно за адресом текущей команды, либо расположен вблизи него. Это называется **пространственной локальностью программы**.
- Обрабатываемые программой данные обычно структурированы, и такие структуры обычно хранятся в последовательных ячейках памяти. Это называется **пространственной локальностью данных**.
- Программы обычно содержат множество циклов и подпрограмм. Значит, некоторые команды многократно повторяются в течение работы программы. Это называется **временно́й локальностью.**
- Все три вида локальности называют **локальностью по обращению**.

```
0040763E |. PUSH EDX
0040763F |. MOV EAX,DWORD...
00407642 |. SUB EAX,20
00407645 |. PUSH EAX
00407646 |. CALL 0040CC10
0040764B |. ADD ESP,0C
```

```
Dat: array [1..7] of integer;
begin
        Dat[1] := 7;
        Dat[2] ;= 9;
        Dat[3] := 10;
```

```
string = "Hello, World"
for x in string:
    print x
```

Принцип локальности иногда представляют в виде принципа Парето (или прицила «20/80»): **80% времени работы программы связано с доступом к 20% адресного пространства этой программы.**

Устранение узкого места принстонской архитектуры

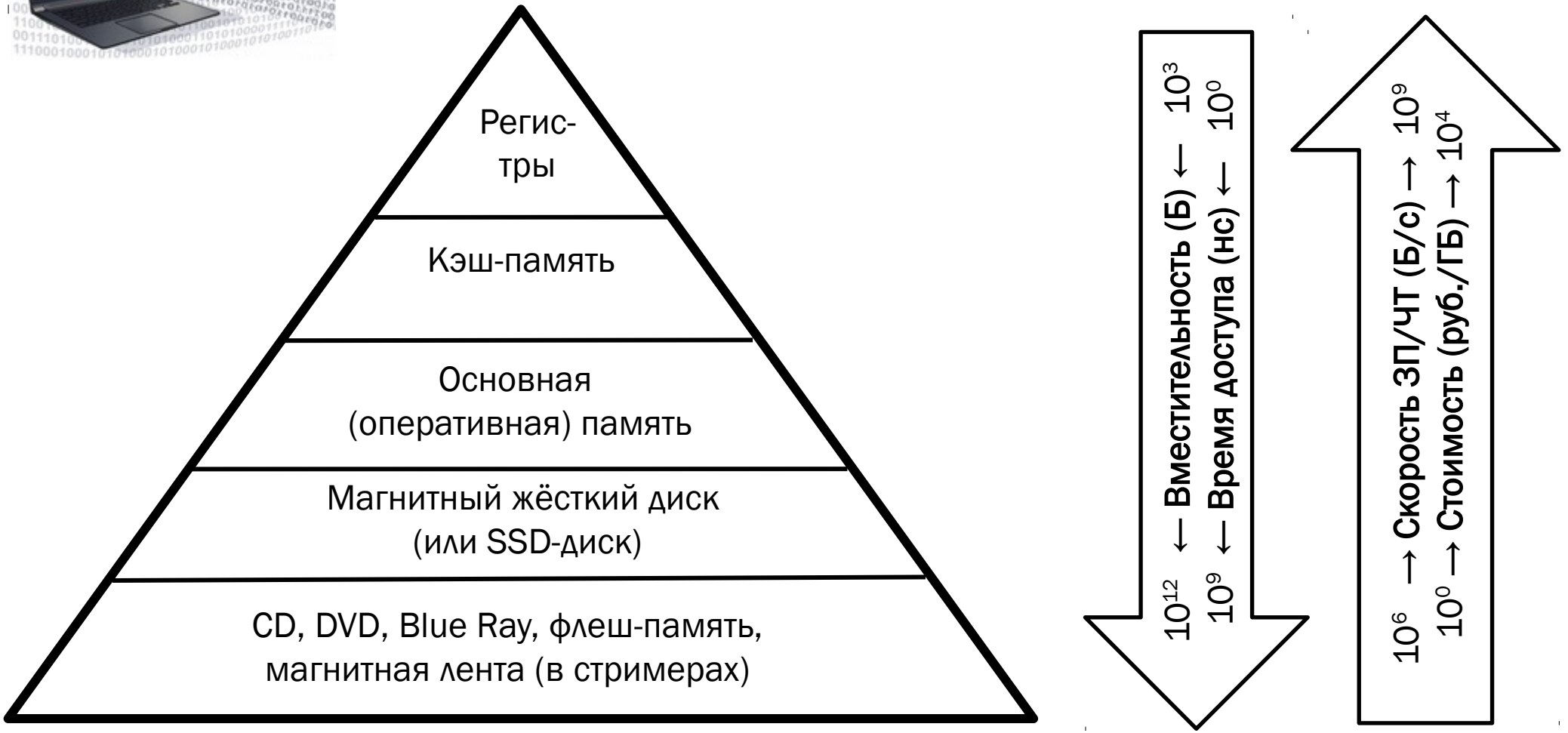
Эффект кэширования

Задача: как сильно увеличится производительность компьютера, если всю оперативную память заменить на быструю кэш-память?

- Пусть при работе программы 80% обращений происходит к кэш-памяти (время доступа – 1 нс), а оставшиеся 20% обращений приходятся на ОП (время доступа – 10 нс).
- Среднее время, затраченное на N обращений к памяти равно:
  T_1 = 80% * N * 1нс + 20% * N * 10нс = 2,8*N нс
- Если всю ОП заменить на кэш-память, то среднее время N обращений сократится до T_2 = N нс.

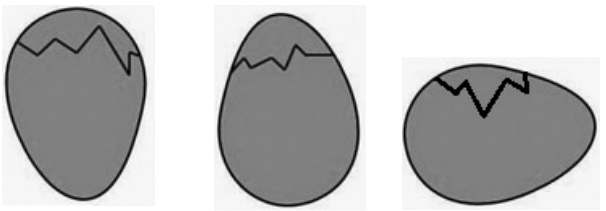
Вывод: ускорение памяти в 10 раз (переплата за такое ускорение может оказаться 1000-кратной) приведёт к ускорению работы программы с памятью всего в T_1/T_2 = 2,8 $\approx$ 3 раза!

Иерархия видов памяти компьютера



Порядок хранения байт в памяти

Проблема: как записать четырехбайтное число $1A2B3C4D_{16}$ в однобайтные ячейки памяти?



Порядок увеличения адреса ячейки →

Big-endian	0x1A	0x2B	0x3C	0x4D
Little-endian	0x4D	0x3C	0x2B	0x1A
Middle-endian	0x2B	0x1A	0x4D	0x3C
Адрес ячейки:	0	1	2	3