

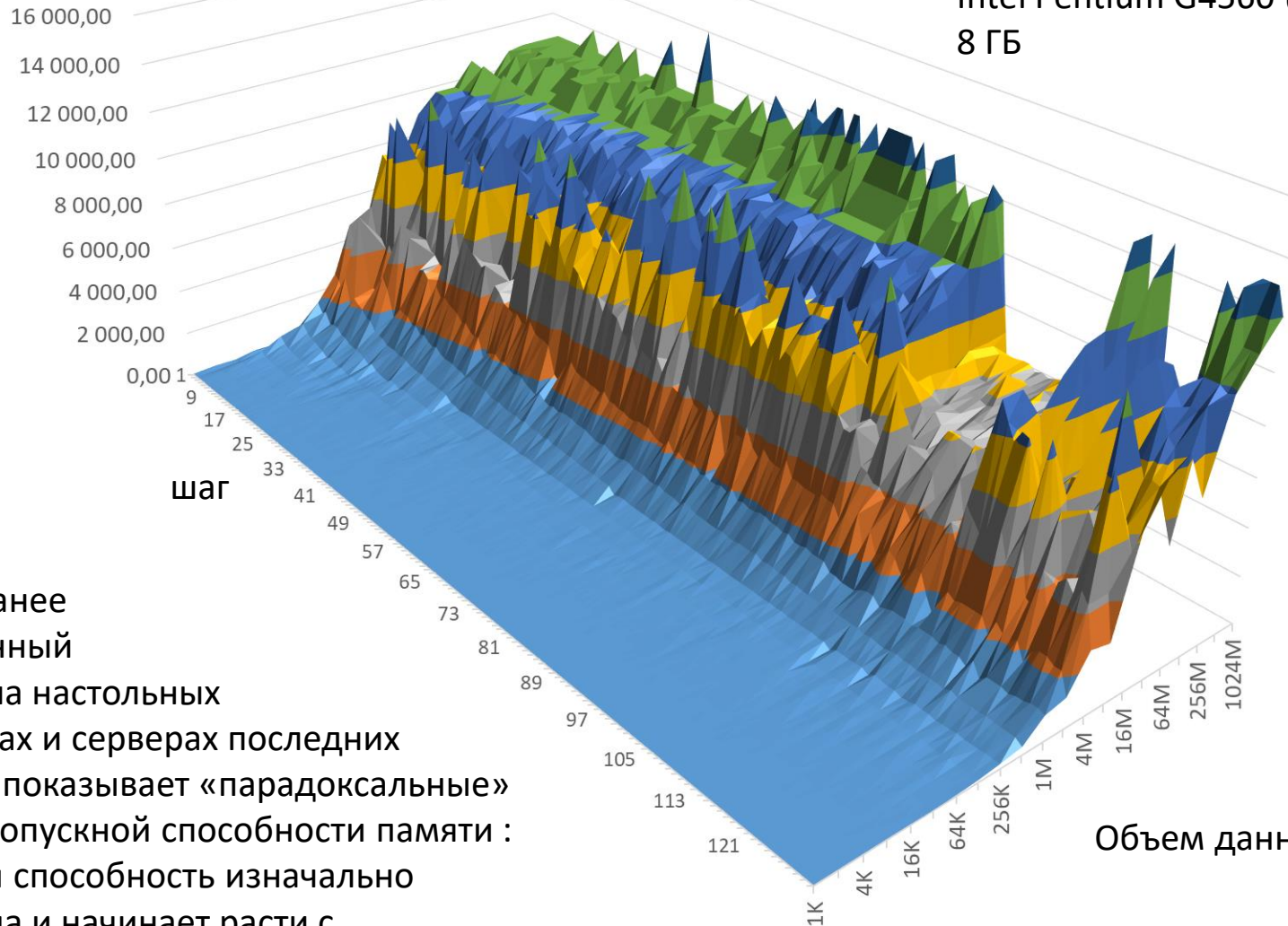
Лекция 0x16

22 апреля

Don't Try This at Home

Пропускная способность,
чтение
МБ/с

Intel Pentium G4560 @ 3.50 ГГц
8 ГБ



Простой, ранее рассмотренный бенчмарк на настольных компьютерах и серверах последних поколений показывает «парадоксальные» графики пропускной способности памяти : пропускная способность изначально крайне мала и начинает расти с увеличением объема данных.

Неучтенные факторы

- Пропуск тактов процессора (дресселирование тактов, CPU throttling) – механизм снижения тактовой частоты процессора
 - Энергопотребление процессора $P \approx C \times V^2 \times f$, где C – электр. емкость, V – питающее напряжение, f – частота
 - Снижение частоты снижает энергопотребление, а следовательно, и энерговыделение
 - Защита от перегрева
 - Экономия энергии, когда компьютер не занят выполнением программ, требующих всех вычислительных возможностей
- Современные процессоры способны динамически менять свою тактовую частоту
 - Снижение и восстановление частоты происходит не мгновенно, оно занимает сотни миллисекунд
 - Быстро отработавший бенчмарк не успевает дожидаться восстановления штатной частоты процессора

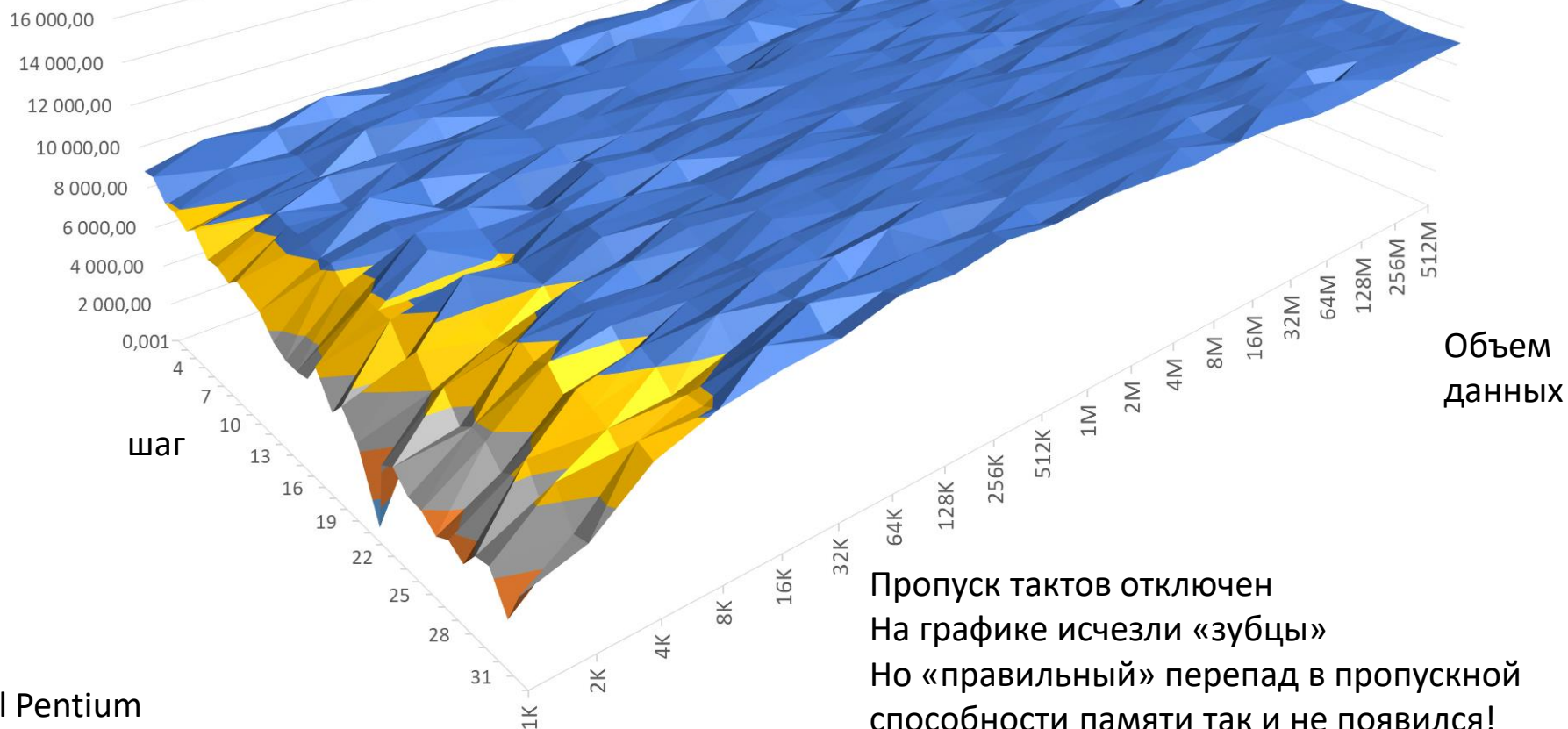
```
#!/bin/sh
```

Как отключить в Linux пропуск тактов на всех ядрах процессора?

```
for CPUFREQ in /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor; do  
  [ -f $CPUFREQ ] || continue; echo -n performance > $CPUFREQ;  
done
```

Попытка №2, что я делаю не так?

Пропускная способность,
чтение, МБ/с



Intel Pentium
G4560 @ 3.50 ГГц
8 ГБ

Пропуск тактов отключен
На графике исчезли «зубцы»
Но «правильный» перепад в пропускной
способности памяти так и не появился!
Почему?

Остались и другие неучтенные факторы

Микроархитектура процессора

- Архитектура набора команд – интерфейс, который определяет, **что** могут делать программы
- Микроархитектура определяет **как** это будет выполняться
- Разработчики аппаратуры стремятся сохранить совместимость ISA и развить возможности микроархитектуры
 - (Обратная) совместимость – возможность выполнять на новых моделях процессора ранее написанные программы без каких-либо изменений
 - ISA можно дополнить новыми командами
 - Развитие микроархитектуры улучшает важные для пользователя свойства процессора
 - Производительность
 - Энергопотребление
 - Безопасность
 - ...

Программное
обеспечение

Прикладные программы

Системные программы

Архитектура набора команд
(ISA)

Микроархитектура

Кэши

Многоядерность

Предсказание
переходов

Упреждающая
выборка

Конвейер
команд

Управление энерго-
потреблением

Цифровой логический
уровень

Физические устройства

Аппаратное
обеспечение

Упреждающая выборка данных

- Используемый бенчмарк обращается за данными строго последовательно, хоть и с различным шагом
- Упреждающая выборка – микроархитектурный механизм для повышения производительности за счет извлечения инструкций и/или данных из памяти в кэш до того, как они фактически потребуются
 - В той модели процессора, на которой запускался бенчмарк, аппаратно реализовано 4 различных алгоритма упреждающей выборки

Пропускная способность vs. Латентность

- Попробуем измерить другую характеристику системы памяти, которую труднее «спрятать» микроархитектурными решениями
- Пропускная способность памяти
 - Объем данных, который может быть передан из памяти за единицу времени. Если построить аналогию с трубопроводом – это ширина трубы
 - Микроархитектурные решения стремятся приблизить эту характеристику к пиковой величине
 - Совокупность решений: расслоение памяти, многоканальность, многоуровневое кэширование, упреждающая выборка, ...
 - Хорошо работают при «предсказуемых» обращениях в память, таких как последовательный проход
 - Ограничениями становятся пропускные способности шин и модулей памяти
- Латентность (задержка)
 - Время, которое затрачивается на чтение из памяти минимальной порции данных (одного слова). В примере с трубой – это длина трубопровода
 - Для оценки требуется обращаться в память по случайным адресам
- Пропускная способность и латентность – относительно независимые друг от друга характеристики

Синтетический микробенчмарк

Оценка латентности памяти

- Измеряется время, затрачиваемое на проход по однонаправленному списку
 - Элементы списка размещаются в памяти последовательно
 - Список «перемешивается», каждый элемент ссылается на другой в произвольном порядке
 - Для снижения погрешностей измерения времени по небольшим спискам проходят несколько раз
- Попытаемся измерить время доступа для списков разных размеров
 - Проход по «перемешанному» списку
 - Последовательный проход по списку
 - Последовательный проход с отключенной упреждающей выборкой

```
typedef struct Node Node;  
  
struct Node {  
    uint64_t payload;  
    Node* next;  
};
```

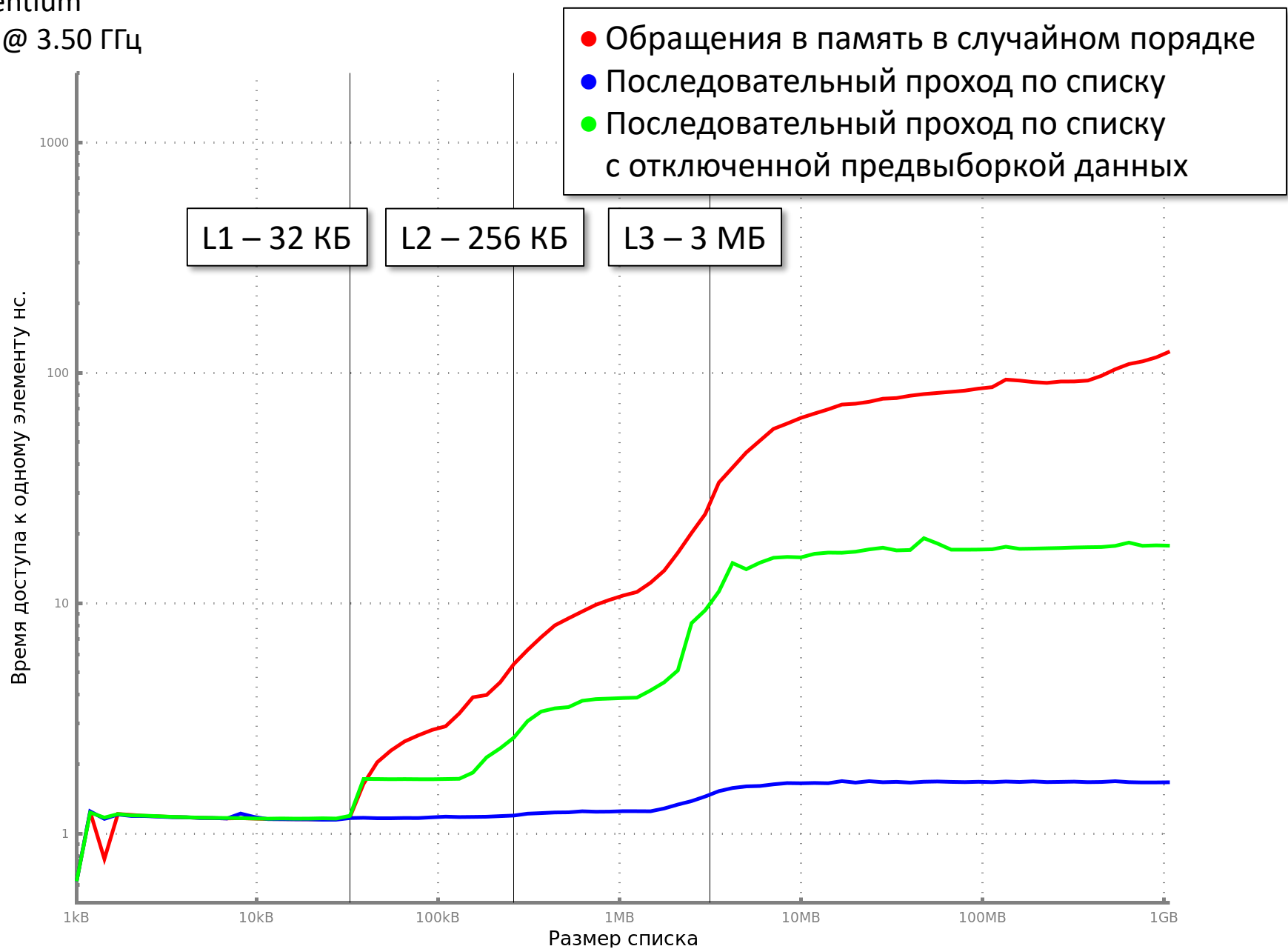
```
for (Int it=0; it<iters; ++it) {  
    Node* node = start_node;  
    while (node) {  
        node = node->next;  
    }  
}
```


Как отключить упреждающую выборку?

- Каждая модель процессора может иметь свои микроархитектурные особенности
 - Управлять ими через специализированные команды – крайне неудобно для разработчиков процессора, придется постоянно дорабатывать ISA
 - Проще организовать управление через специальный набор управляющих и статусных регистров (Model specific register, MSR)
- Моделезависимые регистры (MSR)
 - Обращение к регистру по его номеру
 - Чтение/запись – команды RDMSR и WRMSR, могут выполняться только операционной системой
 - Разработчики процессоров имеют право свободно менять состав и поведение регистров от модели к модели
 - На некоторые MSR доступна документация
 - В Linux обратиться к MSR можно через специальные утилиты, названия которых повторяют имена соответствующих машинных команд
- У используемой автором слайдов модели процессора упреждающая выборка управляется MSR 0x1a4

```
# Устанавливаем пакет msr-tools
sudo apt install msr-tools
# Загружаем ядерный модуль msr
modprobe msr
# Для каждого процессорного ядра выполняем
# запись 4 единиц в младшие разряды MSR 0x1a4
wrmsr -p0 0x1a4 15
wrmsr -p1 0x1a4 15
...
```

Intel Pentium
G4560 @ 3.50 ГГц
8 ГБ



Выводы

- Измерение даже простых характеристик компьютера требует учета многих факторов, без гарантий, что про **все** эти факторы найдется информация в открытых источниках (**зеленая линия** на графике)
 - Отключены только пропуск тактов и упреждающая выборка. Последовательные обращения памяти замедлились на порядок, но все еще на порядок быстрее, чем обращения к произвольным адресам
- Регулярные, последовательные, обращения к памяти могут сохранять высокую пропускную способность и латентность даже при потере локальности, благодаря комплексу дополняющих друг друга микроархитектурных решений (**синяя линия** на графике)
- Латентность памяти скрыть гораздо тяжелей по ряду причин
 - В частности, потому что память физически удалена от процессора
 - Пример: 1ГГц, длительность 1 такта – 1 нс. За это время световой сигнал в вакууме успеет распространиться примерно на 30 см.
 - Скорость распространения электрического сигнала в меди – 60-90% скорости света
 - Необходимо передать в память команду на считывание, строб строки, строб столбца, получить обратно данные, ...
- Если данные не умещаются в кэш, то длительность произвольных обращений к ним микроархитектурные решения улучшить не в состоянии (**красная линия** на графике)

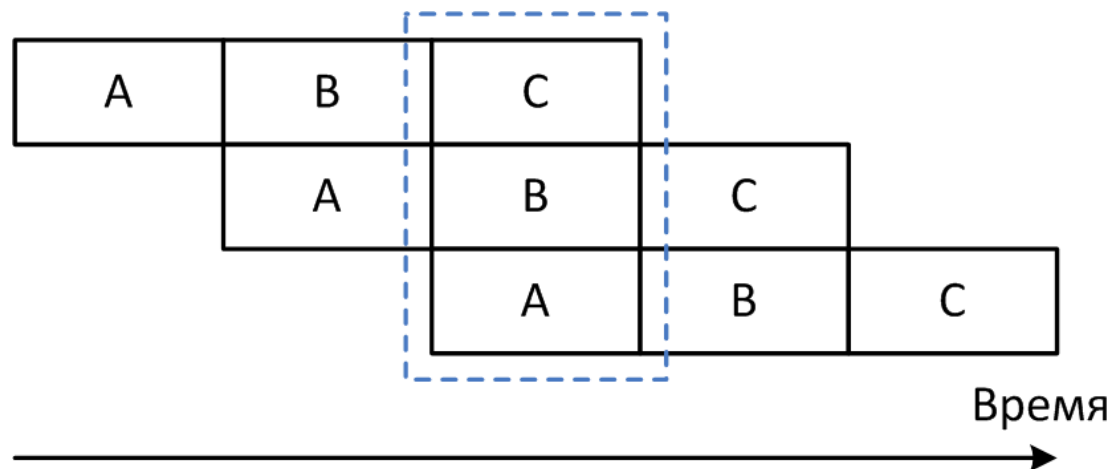
Микроархитектурные решения

- Конвейер команд
 - Суперскалярная архитектура
 - Несколько параллельно работающих конвейеров
- Многоуровневое кэширование
 - Отдельные кэши для кода и данных
- Многоядерность
- Предсказание передачи управления (переходов)
- Упреждающая выборка кода и данных

- Микрокод
- Внеочередное выполнение команд

Конвейер – совмещение разных действий в один момент времени

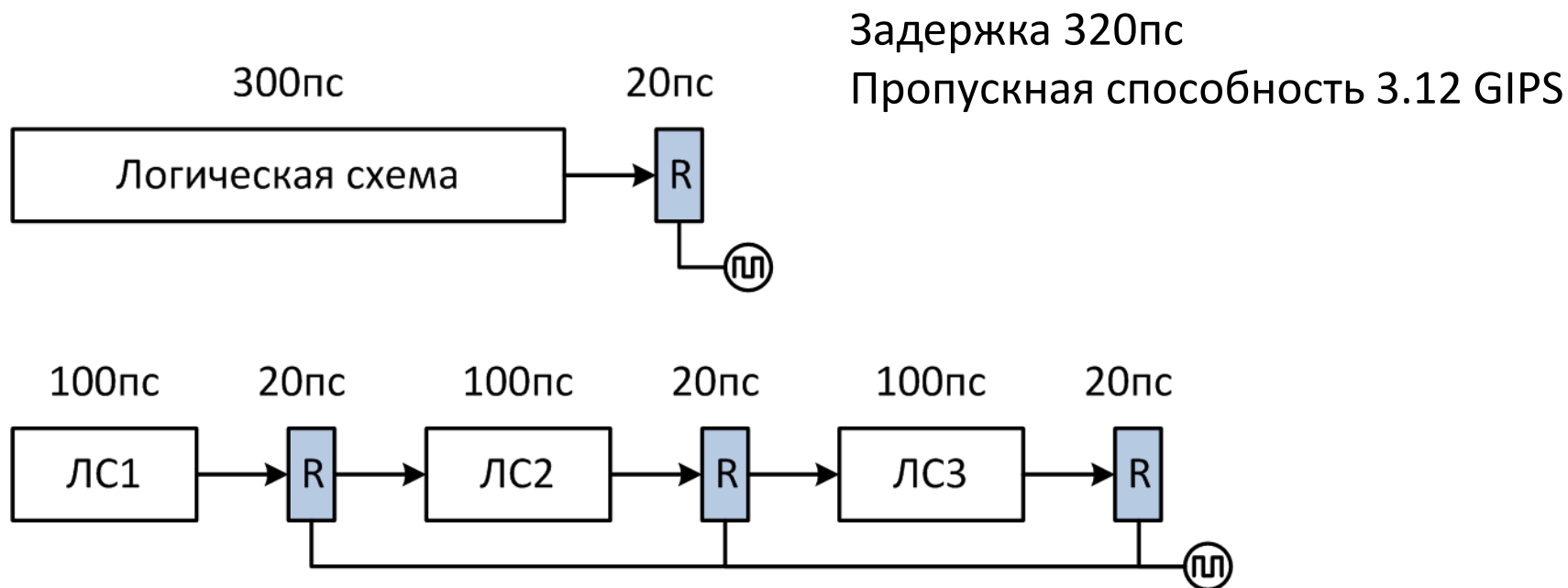
- Общая для различных предметных областей методика
- Длительность обработки неизменна или несколько увеличивается
- Увеличение пропускной способности



Упрощенная схема работы конвейера x86

1. (Fet) Извлечение инструкции из памяти
 2. (Dec) Декодирование, обновление EIP
 3. (D-F) Извлечение данных, подготовка операндов к выполнению операции
 4. (Eхe) Непосредственное выполнение операции
 5. (Wrt) Запись результата
- Для IA-32 такое модельное деление конвейера на этапы (стадии) обусловлено тем, что почти *каждая* команда способна работать с операндами различных типов: регистрами, константами и *даже* ячейками памяти.
 - Определение исполнительного адреса операнда, размещенного в памяти, в общем случае требует получения значений двух регистров и двух констант, одного умножения и двух сложений.

Организация конвейерных вычислений



Задержка 320пс

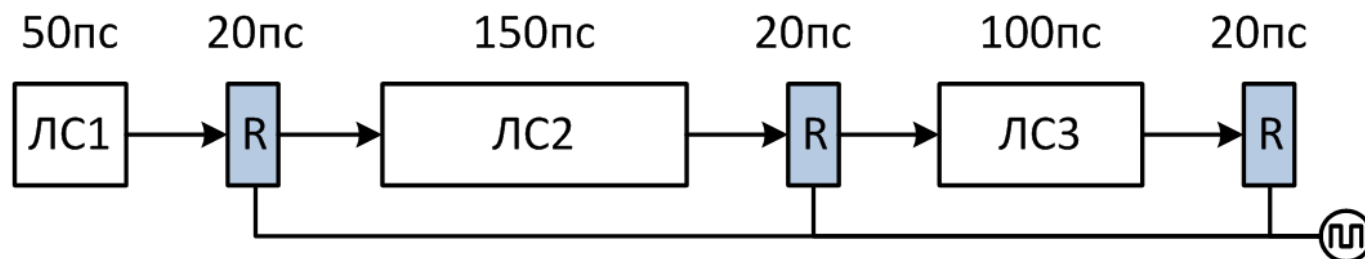
Пропускная способность 3.12 GIPS

Задержка 360пс

Пропускная способность 8.33 GIPS

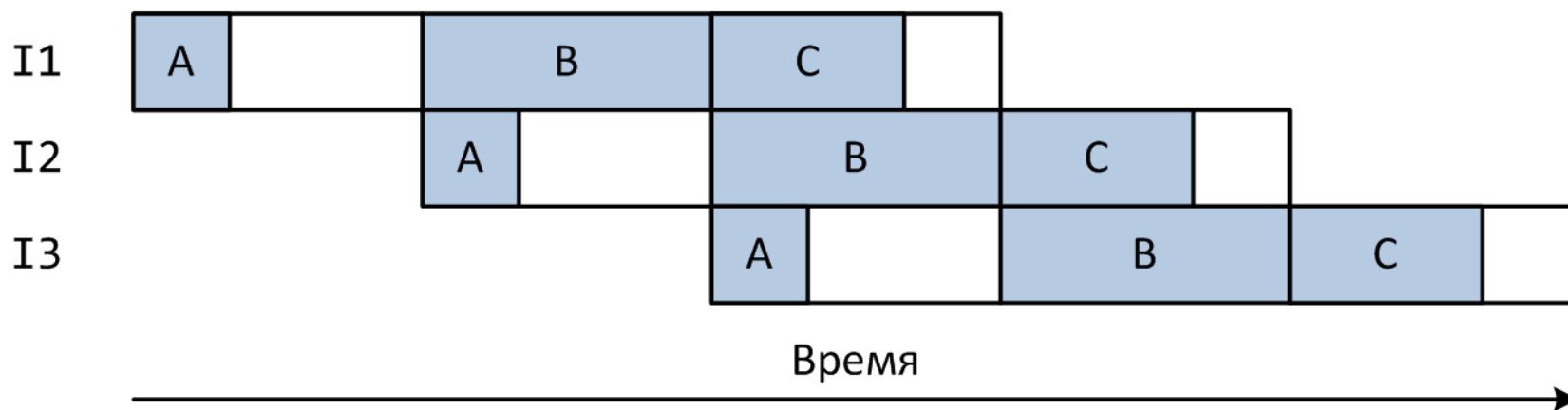
Для временного хранения результатов работы каждой логической схемы требуются служебные регистры

Неоднородность ступеней конвейера



Задержка 510пс

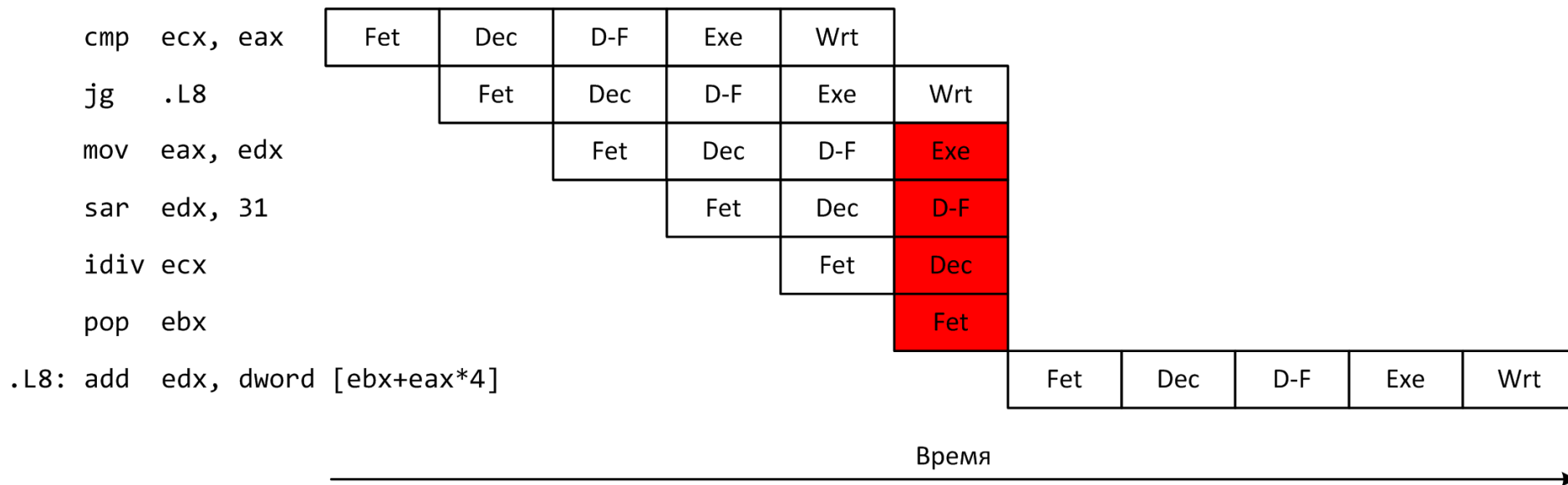
Пропускная способность 5.88 GIPS



Проблемы конвейерной организации

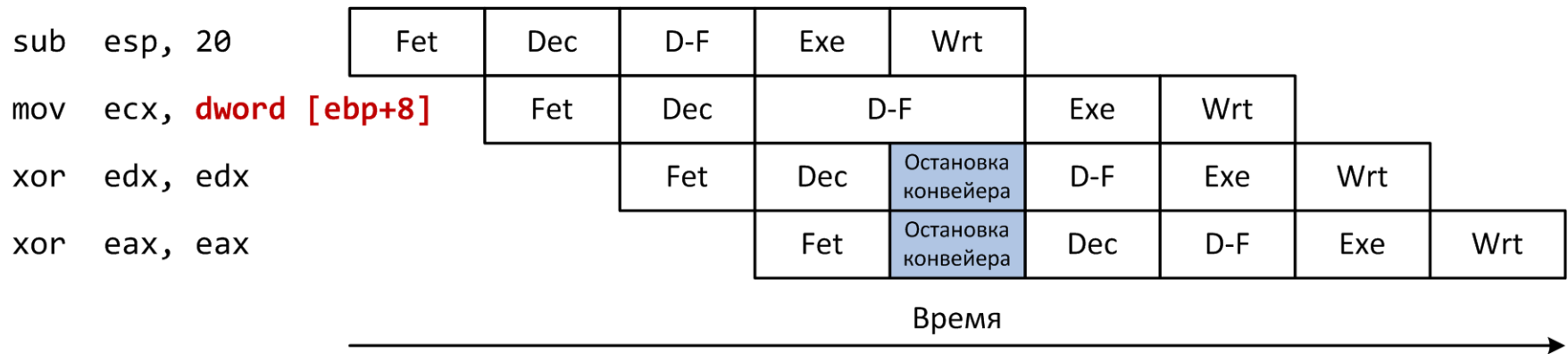
- Опустошение конвейера: сброс промежуточных результатов уже выполняющихся в конвейере команд
 - Современные процессоры могут содержать конвейеры длины 15 и более (P4 Prescott – 31 стадия конвейера)
 - Изменение EIP сбрасывает промежуточные результаты выполнения следующих инструкций
 - Возникновение исключительной ситуации при обращении к памяти или при выполнении операции над данными
- Зависимости по данным между «близко» расположенными командами
- Остановки из-за обращения к памяти

Опустошение конвейера при передаче управления

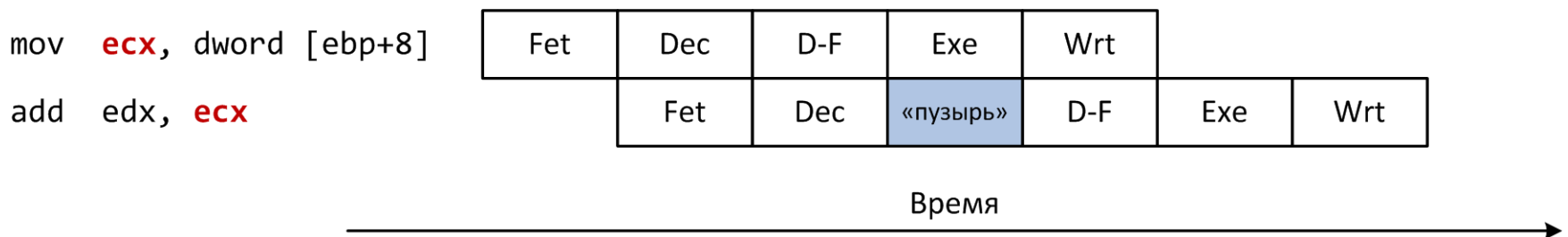


Приостановка конвейера

Некоторые этапы не удастся выполнить за один такт.

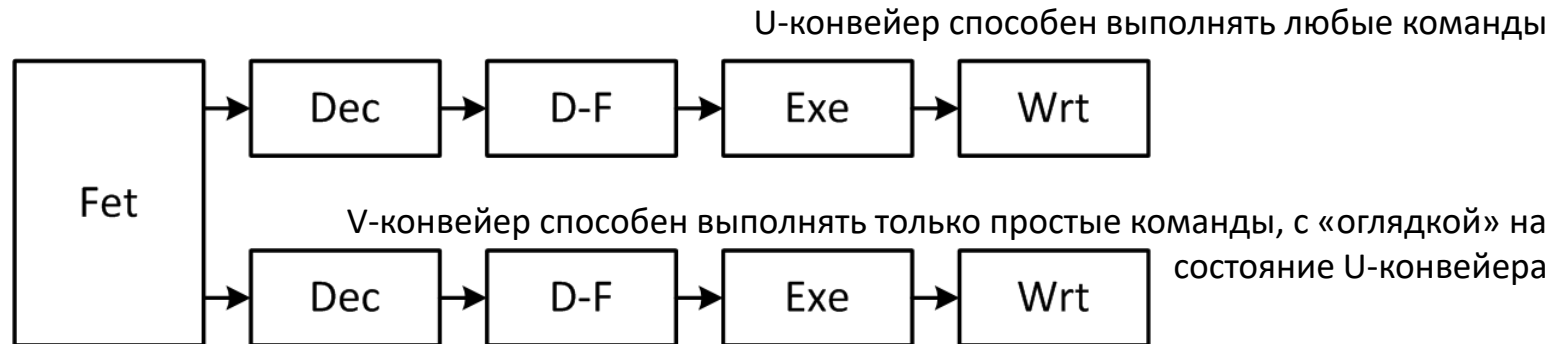


Одновременное выполнение нескольких команд требует учета зависимостей по данным между командами: нельзя извлекать из регистра значение, если предыдущая команда его туда еще не записала.



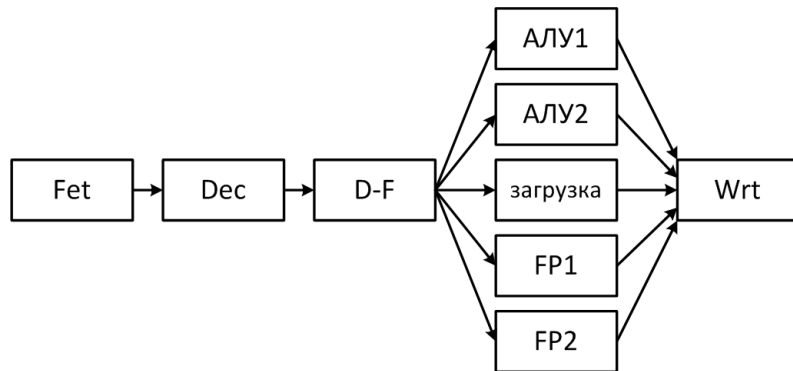
Двойной конвейер

- Двойной конвейер с общим этапом выборки команд
 - (Fet) Извлечение инструкции из памяти
 - (Dec) Декодирование, обновление EIP
 - (D-F) Извлечение данных
 - (Exe) Непосредственное выполнение операции
 - (Wrt) Запись результата



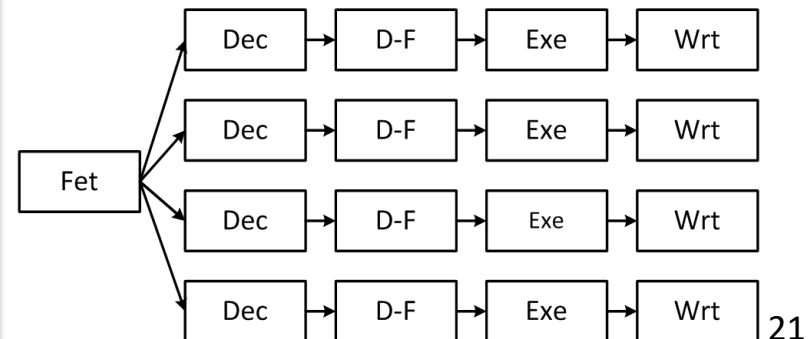
- Впервые появился на Pentium: введены u- и v-конвейеры
- Возможности конвейеров неравноценны

Суперскалярная и VLIW архитектуры



- Многие операции над данными не могут быть реализованы за один такт.
- В суперскалярной архитектуре выполняющиеся команды на стадии операции расходятся по нескольким функциональным устройствам (ФУ).
- За распределение (планирование) потока команд по ФУ отвечает аппаратура процессора.

- Во VLIW-архитектуре (Very Long Instruction Word) каждая команда состоит из нескольких микрокоманд, независимо друг от друга, выполняющих операции над данными.
- За то, что бы команды действительно не конфликтовали между собой и могли выполняться параллельно, отвечает не процессор, а компилятор.



RISC vs CISC

(Reduced vs Complex Instruction Set Computer)

- Исторически CISC предшествовал RISC
 - PDP-11 → VAX / CISC, 1977 г.
 - MIPS, SPARC / RISC, конец 80-х
- Простые операции
 - Ограниченный набор простых команд (например, нет деления)
 - Команда выполняется за один такт
 - Фиксированная длина команды (простота декодирования)
- Конвейер
 - Каждая операция разбивается на однотипные простые этапы, которые выполняются параллельно
 - Каждый этап занимает 1 такт, в т.ч. декодирование
- Регистры
 - Много однотипных взаимозаменяемых регистров (могут использоваться и для данных, и для адресации)

RISC vs CISC

(Reduced vs Complex Instruction Set Computer)

- Модель работы с памятью
 - Отдельные команды для загрузки/сохранения в память
Альтернативное название RISC-архитектур – Load/Store-машина
 - Команды обработки данных работают только с регистрами
- Результат
 - Устройство ядра процессора становится проще
 - Проще наращивать частоту процессора
 - Меньше энергопотребление
 - По сравнению с CISC объем кода (число команд) при реализации того же алгоритма на RISC-машине будет больше
- Сложность оптимизаций перенесена из процессора в компилятор
 - Производительность сильно зависит от компилятора



- Процессорная архитектура с открытым исходным кодом
 - Представлена в 2010 году, разработчик – Университет Беркли, при непосредственном участии Дэвида Паттерсона (Лекция 1, дополнительная литература)
 - Получила признание не только у ученых, но и в промышленности: Western Digital, Huawei, Alibaba Cloud, Hitachi, NVIDIA, Google, NXP, Samsung, ...
 - Спецификация ISA публикуется RISC-V Foundation
<https://riscv.org/specifications/isa-spec-pdf/>
 - Исходные коды процессорных ядер опубликовали десятки разработчиков: <https://github.com/riscv/riscv-cores-list>
Языки разработки: Verilog, SystemVerilog, Chisel, VHDL и др.
- В 2019г. в России порядка 10 центров разработки вели проекты с использованием RISC-V
 - Промышленный выпуск систем-на-чипе в состав которых включен процессор RISC-V
- Учебные и исследовательские проекты в МИЭТ, МФТИ, ИСП РАН, МГУ, ИТМО

RISC-V: архитектура набора команд

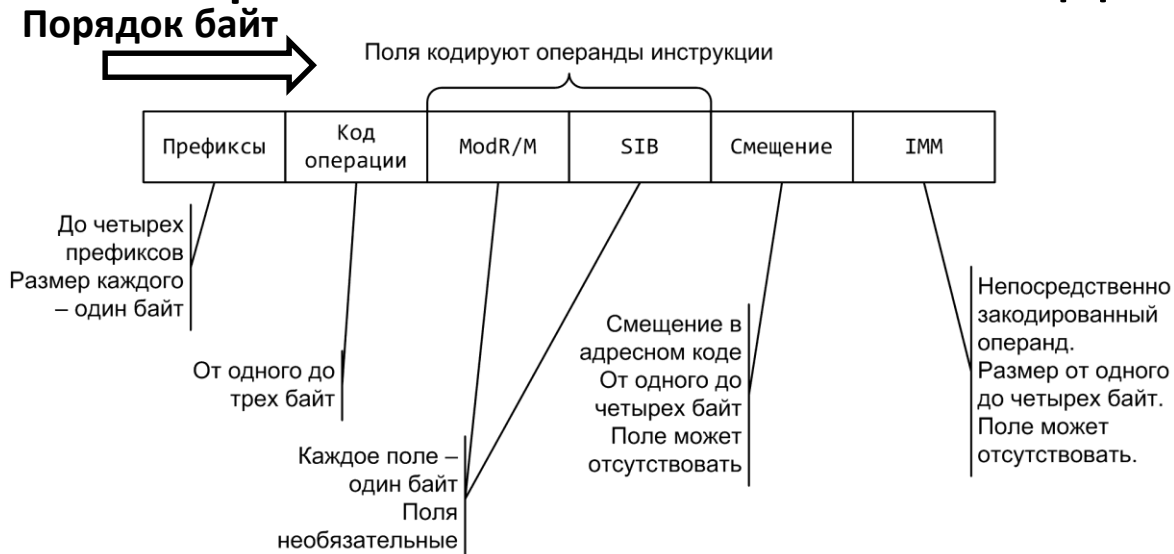
- Две версии процессора: 32 и 64 разряда
- Характеристики RV 32 ISA
 - Машинное слово – 32 разряда. Такой размер имеют: адрес, регистры и **размер команды**.
 - 32 регистра общего назначения x0 – x31. Счетчик команд **pc**. x0 «запаян» всегда хранить 0, на записи не реагирует.
 - Базовый набор команд над целыми числами RV32I – 47 команд. Только простые операции: сложение, вычитание, логика, сдвиги, сравнение, передача управления.
 - Большинство команд трехадресные.
 - Умножение, деление и взятие остатка вынесены в расширение набора команд RV32M, который необязателен для реализации, как и другие расширения.
 - Флагов состояний нет, аппаратной поддержки стека нет, как и многих других излишеств, например, команды MOV.

ADDI x28, x29, 0 ; пересылка из x29 в x28

В чем RISC-V совсем не похож на x86

Формат машинной команды

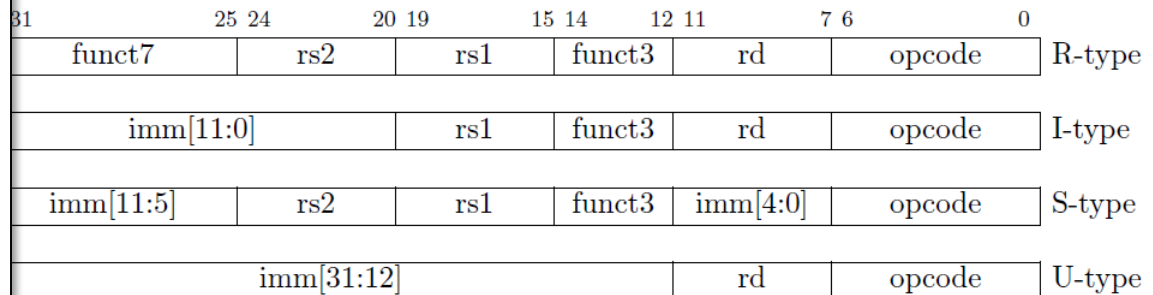
- x86



Переменная длина, от 1 до 15 байт.

- RISC-V – всегда 32 разряда, 4 простых формата

- Как сложить регистр с 32-х разрядной константой?
- Как обратиться к произвольному адресу памяти?!?



В чем RISC-V совсем не похож на x86

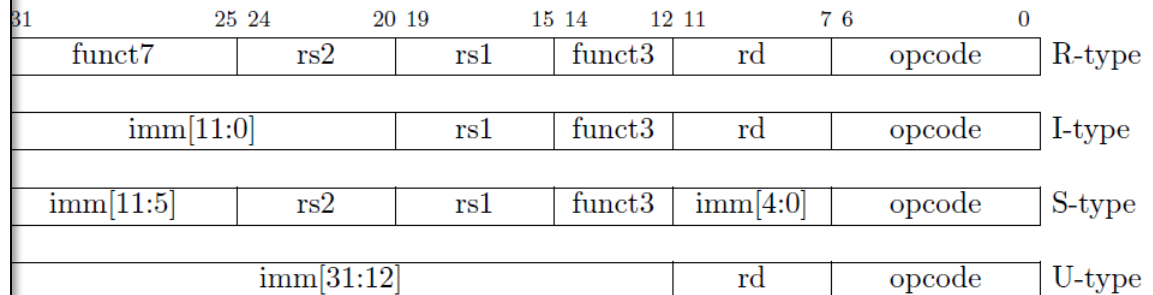
Формат машинной команды

LUI x5, 0x12345 ; загружаем старшие 20 разрядов x5
 ADDI x5, x5, 0x678 ; заполняем младшие 12 разрядов x5

LW x5, 0xc(x6) ; загружаем в x5 слово из памяти
 ; по адресу x6+0xc
 ; константа кодируется 12 бит
 ; как получить произвольный базовый
 ; адрес – см. пример выше

- RISC-V – всегда 32 разряда, 4 простых формата

- Как сложить регистр с 32-х разрядной константой?
- Как обратиться к произвольному адресу памяти?!?



В чем RISC-V не похож на x86

Сравнения и передача управления

- Безусловный переход

```

; jump and link [register]
JAL  x1, 0x12345 ; x1 ← pc + 4
; pc ← pc + ОП2 (20 разрядов)
JALR x1, x28, 0x20 ; x1 ← pc+4
; pc ← ОП1 + ОП2 (12 разрядов)

```

- Сравнения

```

SLT[U] x28, x29, x30 ; if (ОП2 <[u] ОП3) {
;     ОП1 ← 1 } else {
;     ОП1 ← 0
; }

```

- Условные переходы

```

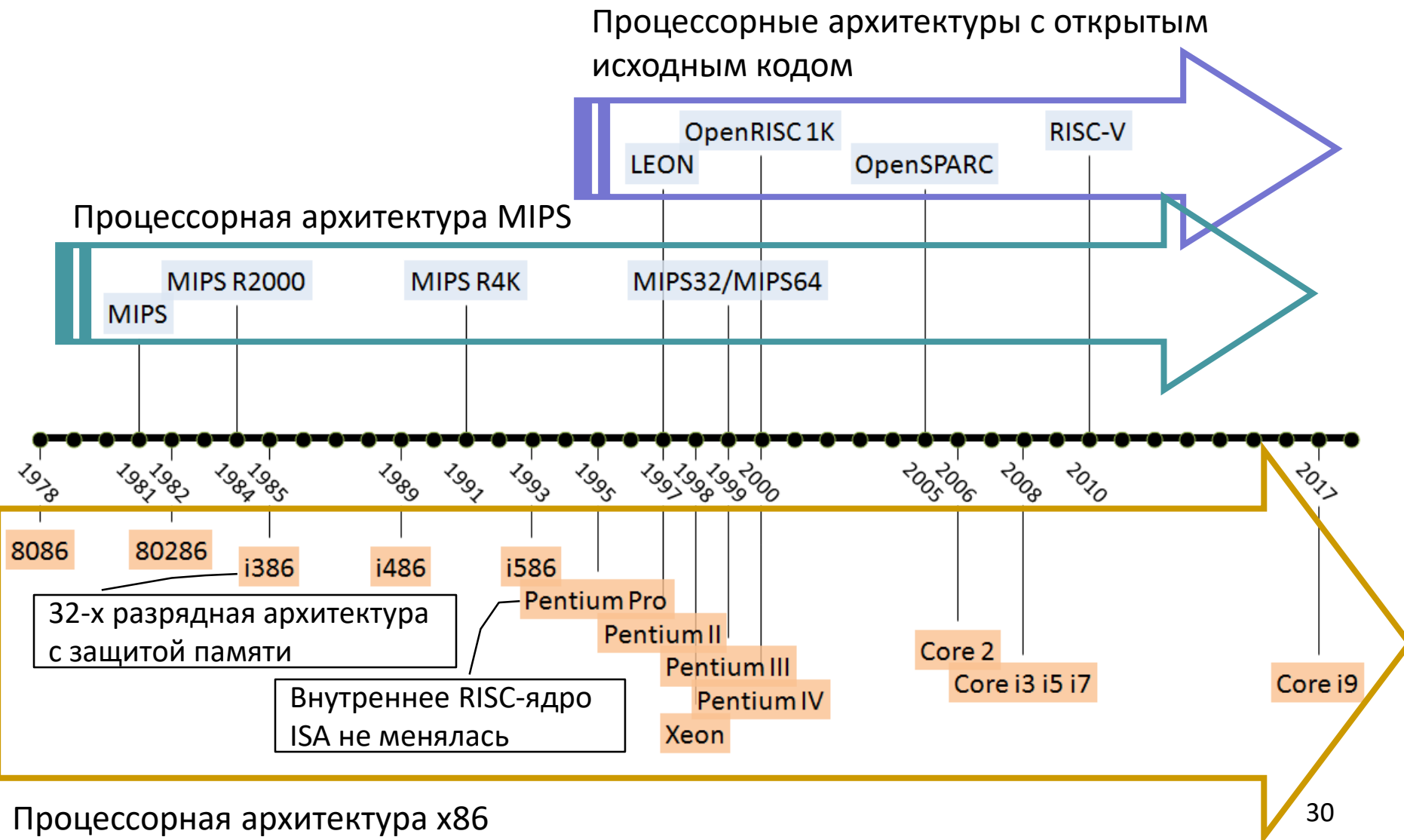
BEQ  x28, x29, 0x10 ; if (ОП1 == ОП2) {
;     pc ← pc + (2 * ОП3)
; } // ОП3 - 12 разрядов

```

Конвейер RISC-V

- В каждой команде присутствует операция над данными из регистров
 - Если происходит обращение в память, исполнительный адрес получается сложением базового регистра и непосредственно закодированного значения
- Этапы конвейера RISC-V
 1. Извлечение команды из памяти, увеличение pc на 4
 2. Декодирование и извлечение значений операндов – регистров и непосредственно закодированной константы
 3. Выполнение операции
 4. Обращение к памяти, если только команда с памятью работает
 5. Запись результата в регистр

Развитие x86 и некоторых RISC процессоров



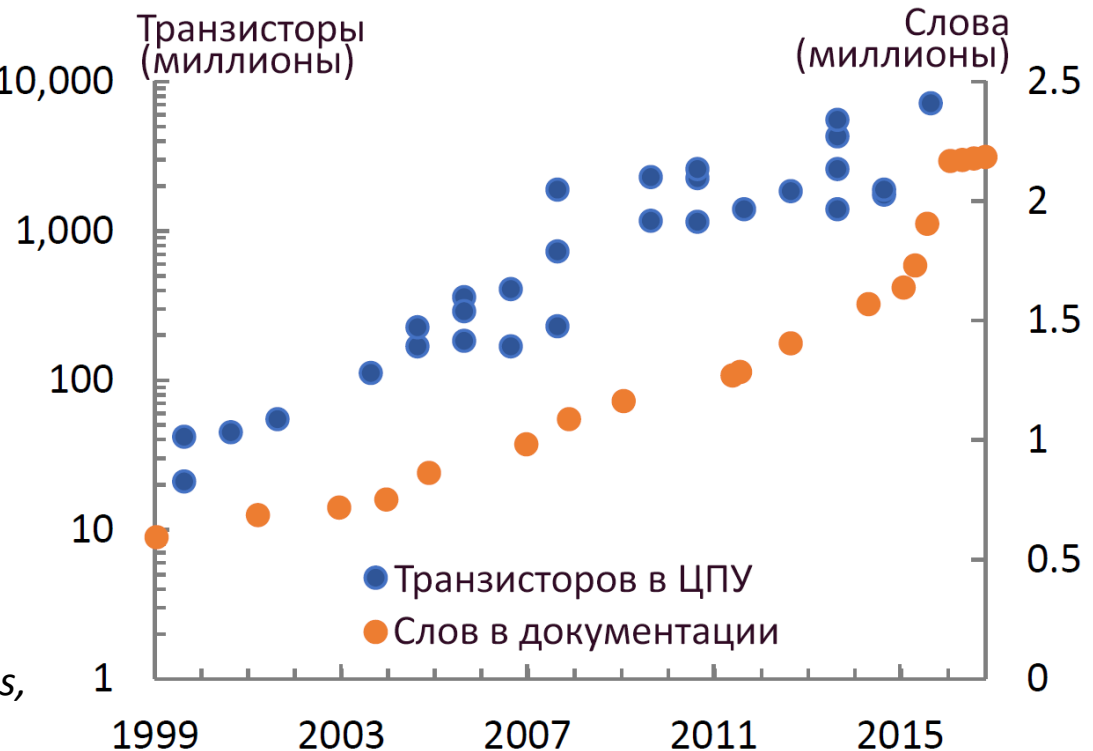
История развития x86

- 4004 – ноябрь 1971. 4-битный микропроцессор. Первый в мире коммерчески доступный однокристалльный микропроцессор.
- 8008 – апрель 1972. 8080 – апрель 1974. 8-битные процессоры.
- 8086 – **1978**. Размер слова – 16 бит, ширина адресной шины – 20 бит. Адреса вычисляются с использованием сегментных регистров.
- 80186 – 1982. Добавлено несколько новых инструкций.
- 80286 – 1982. Ширина адресной шины – 24 бита, добавлено устройство контроля памяти – MMU. Процессор мог переключаться между двумя режимами – реальным и защищенным.
- 80386 – октябрь **1985**. Снят с производства в **2007**. Размер слова – 32-разряда. Процессор мог переключаться между тремя режимами – реальным, защищенным, виртуальным. Адресуемая память – 4 ГБ.
- ...
- Intel Xeon E7-8870 – **2011**, 10 ядер, до 8 ЦПУ в системе. 2.4 (2.8) ГГц; 2.2×10^9 транзисторов; системная шина QPI: 6.4 ГТ/с; обращение к памяти одного процессора: 32 ГБ/с (4x8ГБ/с), адресуемая память – 4 ТБ ; Кэш L1: 640КБ L2: 2.5МБ, L3: 30МБ.
- ...
- Intel Xeon 8180M – **2017**, 28 ядер, до 8 ЦПУ в системе. 2.5 (3.8) ГГц; транзисторов в аналоге от AMD 19×10^9 ; системная шина UPI: 10 ГТ/с; обращение к памяти одного процессора: 120 ГБ/с (6x20ГБ/с); адресуемая память – 1.5ТБ. Кэш L1: 896+896КБ, L2: 28МБ, L3: 38.5 МБ.
- ...

Развитие системы команд в архитектуре x86

- 23 расширения команд в период 2011-2016
- Значительная часть – системные команды, отвечающие за безопасность и изоляцию программ и данных

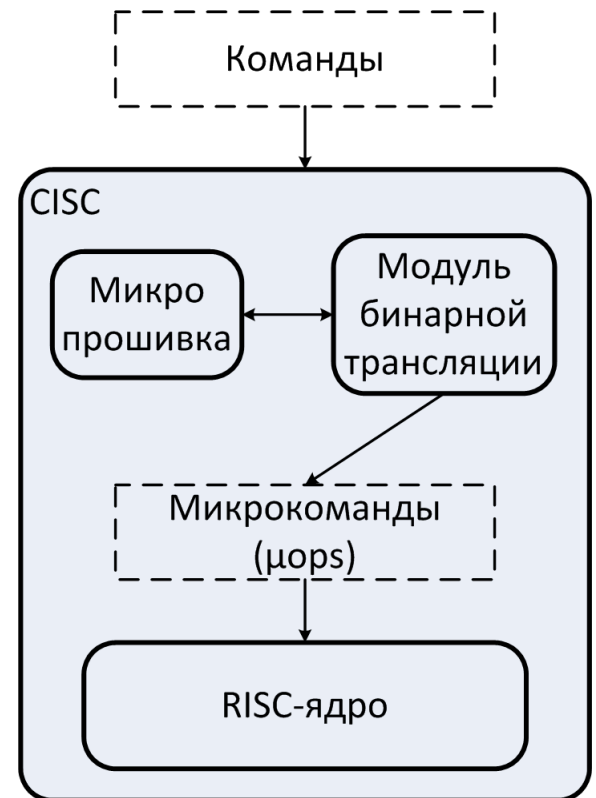
- VT-x, VT-d / AMD-V
- SGX
- MPX / MPK
- CET



Andrew Baumann (Microsoft Research).
 Hardware is the new software. // 16th
 Workshop on Hot Topics in Operating Systems,
 May 2017

Микрокод

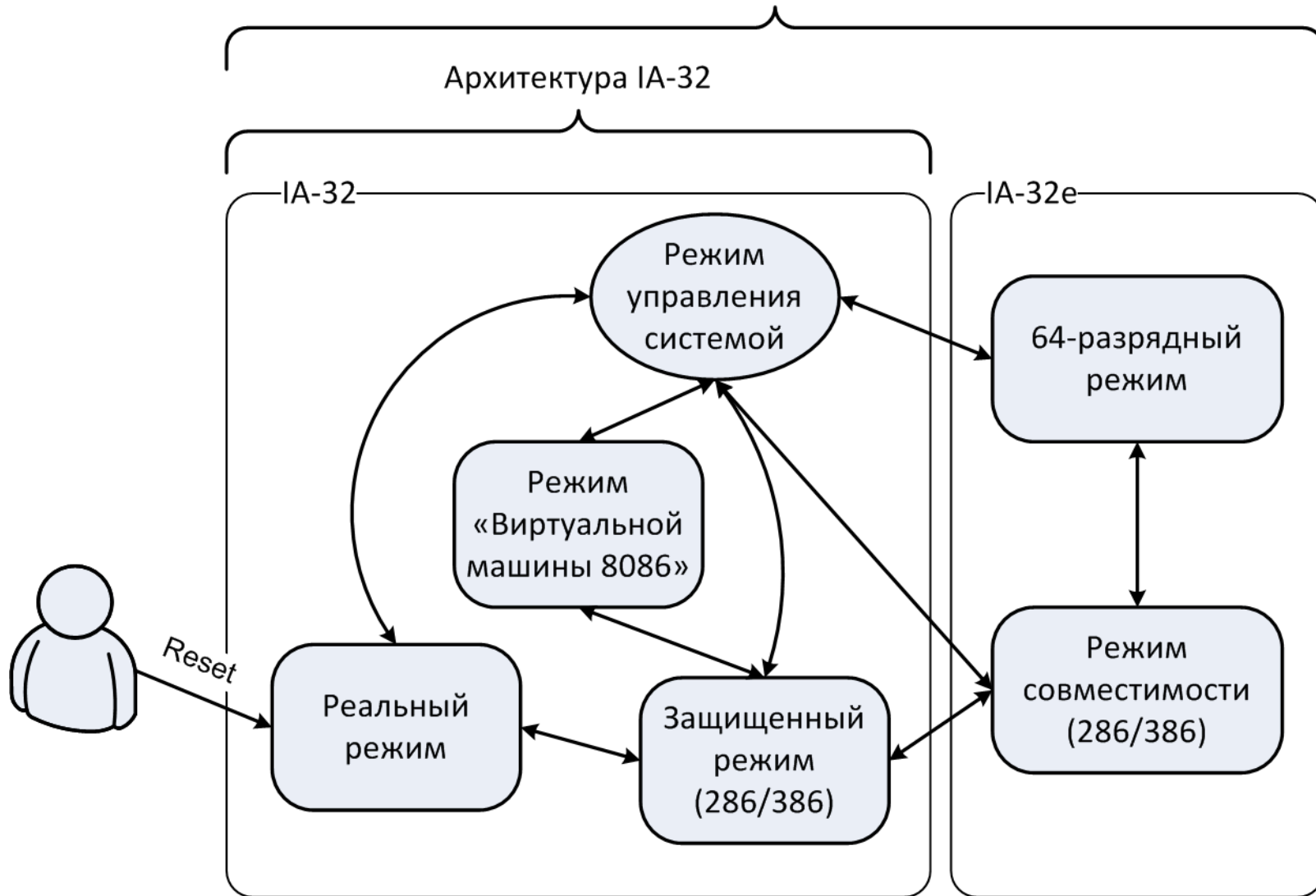
- По мнению исследователя Andrew Baumann на протяжении нескольких лет аппаратное устройство процессоров Intel принципиально не менялось, для поддержки новых команд обновлялась его «прошивка» – микрокод
- Микрокод — программа, реализующая набор инструкций процессора
- Современные процессоры x86 динамически, на лету, транслируют команды x86 в микрокоманды (μops), которые выполняются внутренним RISC-ядром
 - Подход применяется не только в x84 / Intel
 - Удобно исправлять ошибки в поведении сложных команд и распространять эти исправления, т.е. новую версию микрокода (пакет linux-firmware)
 - Удобно расширять набор команд
- Идея микрокода была предложена еще на заре эпохи компьютеров
 - M. V. Wilkes, J. B. Stringer. Microprogramming and the Design of the Control Circuits in an Electronic Digital Computer. // Proc. Cambridge Phil. Soc., pt. 2, vol. 49, April 1953, pp. 230-238.



Что дальше?

- Режимы работы процессора x86
- Загрузка процессора, BIOS, UEFI и ACPI
- Что должно быть в процессоре, чтоб он смог одновременно обслуживать выполнение нескольких программ
- Модели памяти в x86 и механизмы защиты
- Страничная организация памяти, кэширование результатов трансляции адресов
- Прерывания и системные вызовы

Архитектура Intel 64

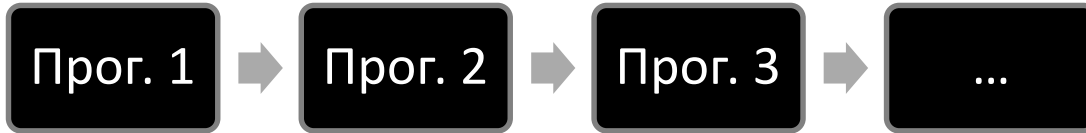


Начальная загрузка компьютера – последовательность (цепочка) выполнения все более сложных загрузчиков.
 Конечная цель – загрузка операционной системы

Схема загрузки IA-32

- При включении питания выполнение команд начинается с адреса `0xFFFFFFF0`
 - Содержимое ROM отображается на адреса памяти
 - ROM содержит BIOS – набор микропрограмм, предоставленных производителем аппаратуры. Их цель – дать возможность провести конфигурацию аппаратуры и программно взаимодействовать с ней.
- BIOS
 - Копирование кода из ROM в RAM
 - Самопроверка кода
 - Начальная конфигурация аппаратуры
 - Размещение в оперативной памяти ACPI таблиц
 - Последовательно пытаемся копировать MBR (сектор №0, 512 байт) каждого загрузочного устройства в память по адресу `0x7c00`
 - Если сектор скопировать удалось, BIOS передает управление на адрес `0x7c00`
- MBR-загрузчик копирует в память загрузчик операционной системы и передает ему управление
- Загрузчик операционной системы копирует в память ядро операционной системы и передает ему управление

Реальный режим / 8086



- В каждый момент времени работает **только одна** программа
 - Эта программа управляет всеми ресурсами (ЦПУ, память, ввод/вывод)
- Машинное слово **16** разрядов, адрес памяти **20** разрядов
 - Сегментные регистры
 - Исполнительный адрес = $(\text{seg_reg}) \ll 4 + \text{смещение}$
- Доступна вся **адресуемая** память
- Периферийные устройства управляются через порты ввода/вывода
 - Команды in и out

