Лекция 0x18

11 мая

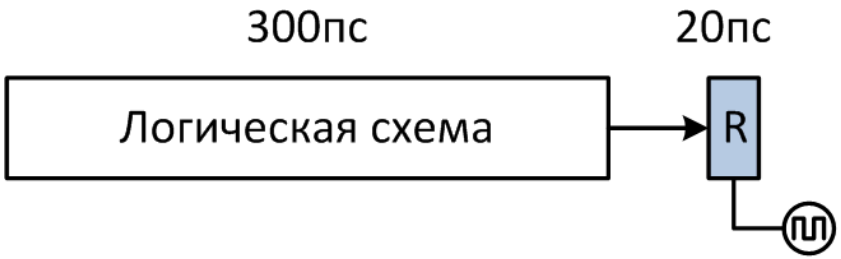
Упрощенная схема работы конвейера x86

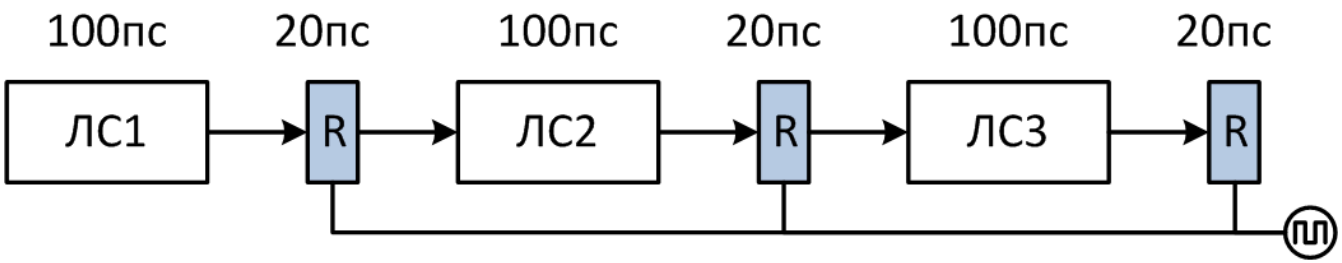
1. `(Fet)` Извлечение инструкции из памяти
2. `(Dec)` Декодирование, обновление EIP
3. `(D-F)` Извлечение данных, подготовка операндов к выполнению операции
4. `(Exe)` Непосредственное выполнение операции
5. `(Wrt)` Запись результата

- 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

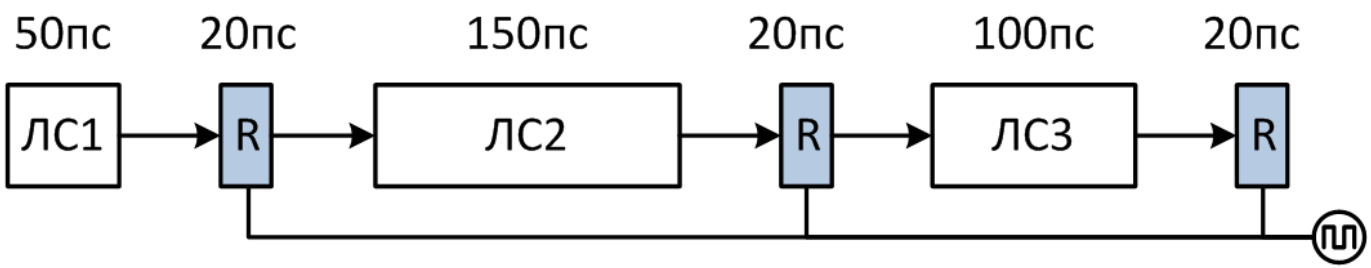
Задержка 320пс
Пропускная способность 3.12 GIPS





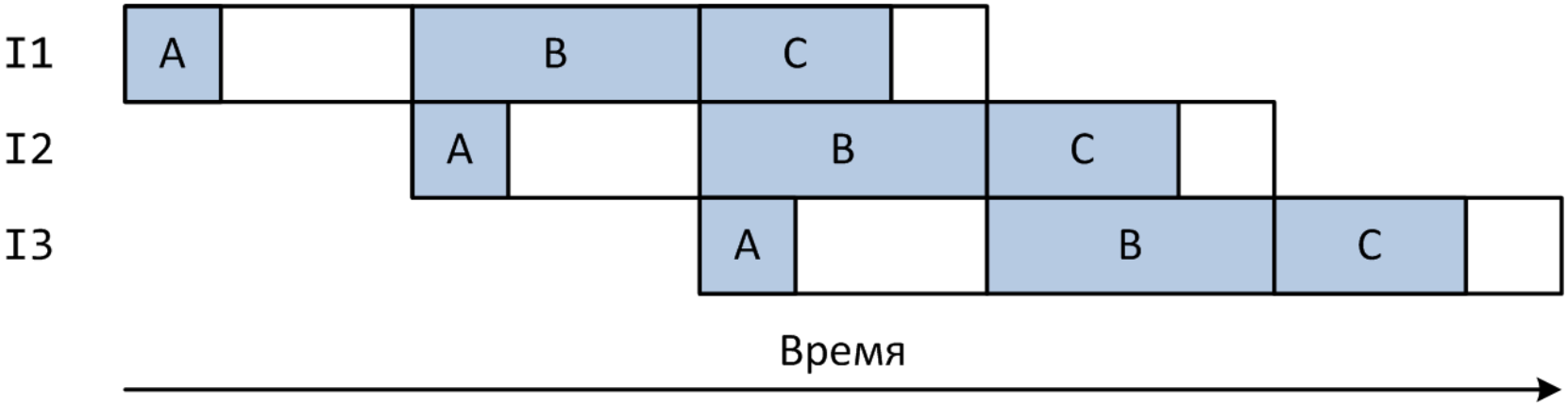
Задержка 360пс
Пропускная способность 8.33 GIPS

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

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

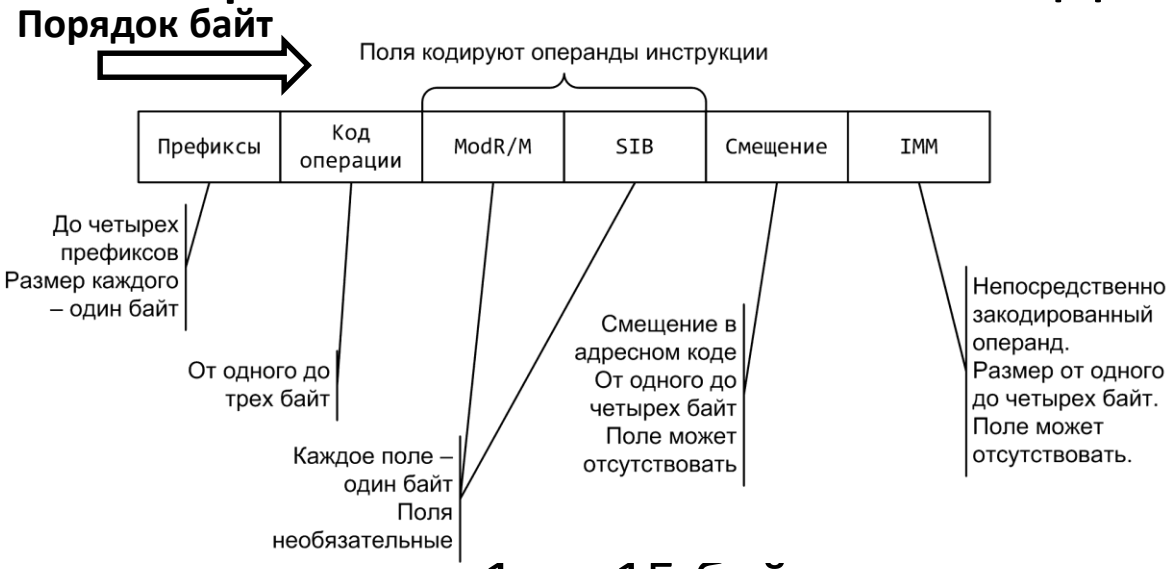
RISC-V

- 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
```

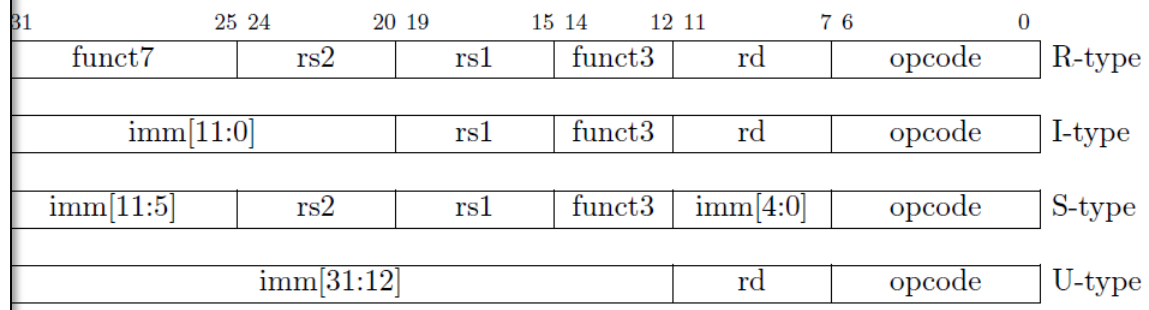
В чем RISC-V совсем не похож на x86
Формат машинной команды

- 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?!?**

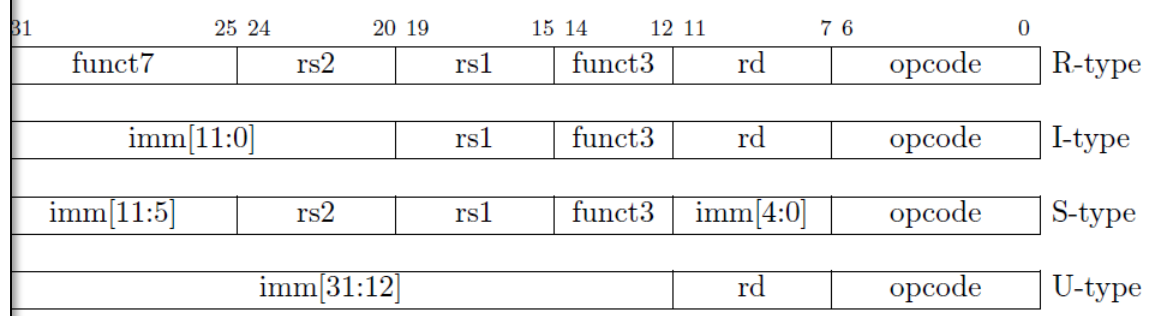
В чем RISC-V не похож на x86
Формат машинной команды

```
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

- **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
                      ; }
```

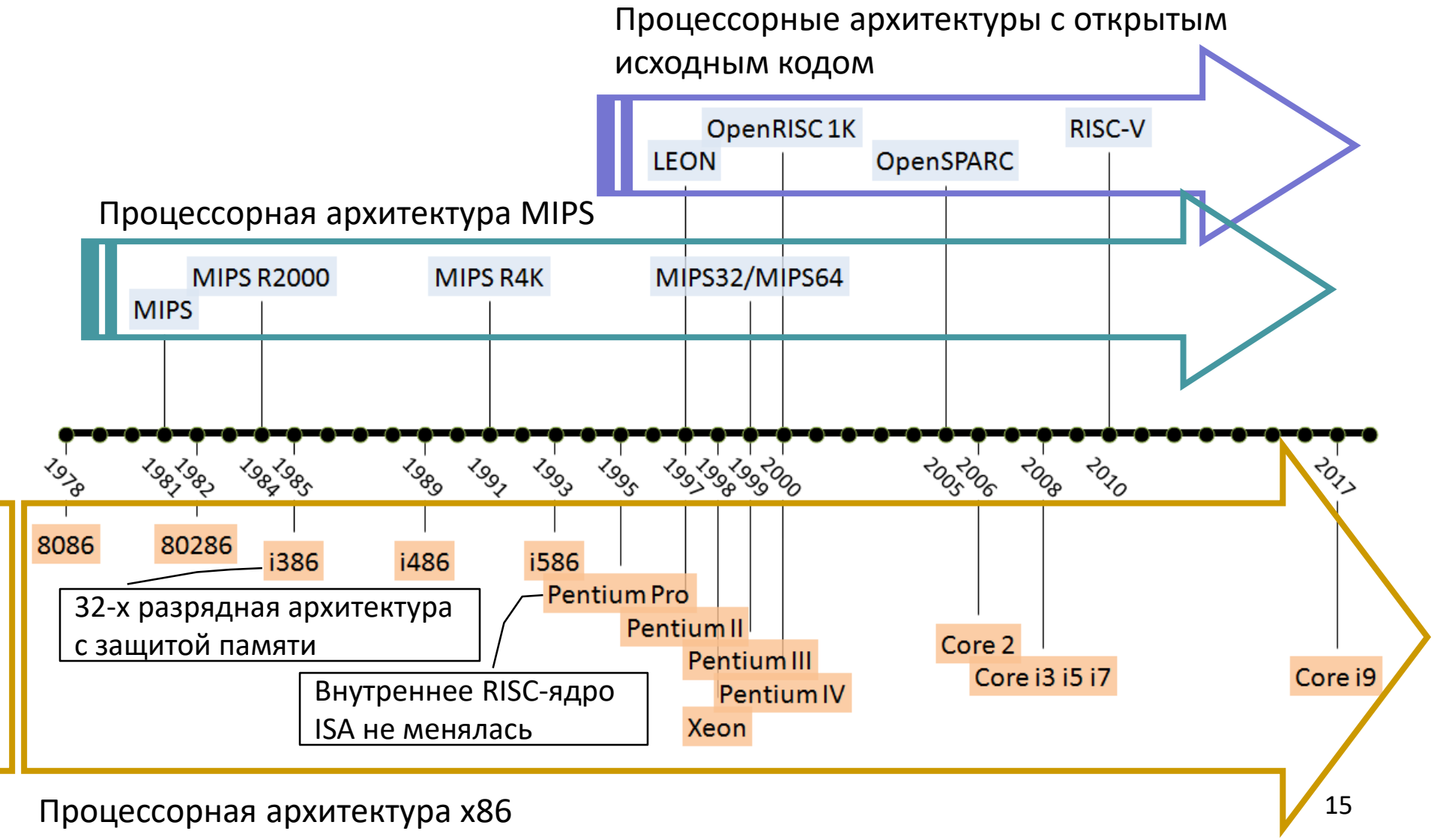
- Условные переходы

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

Конвейер RISC-V

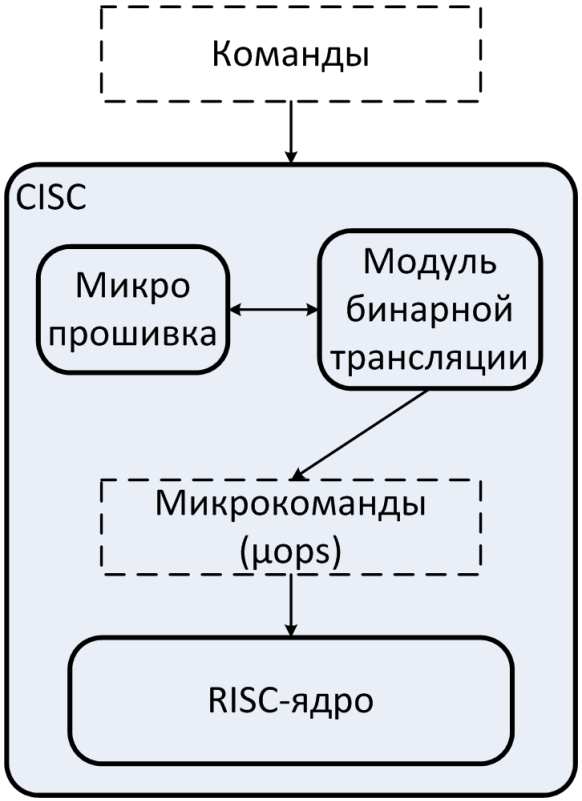
- В каждой команде присутствует операция над данными из регистров
  - Если происходит обращение в память, исполнительный адрес получается сложением базового регистра и непосредственно закодированного значения
- Этапы конвейера RISC-V
  1. Извлечение команды из памяти, увеличение pc на 4
  2. Декодирование 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



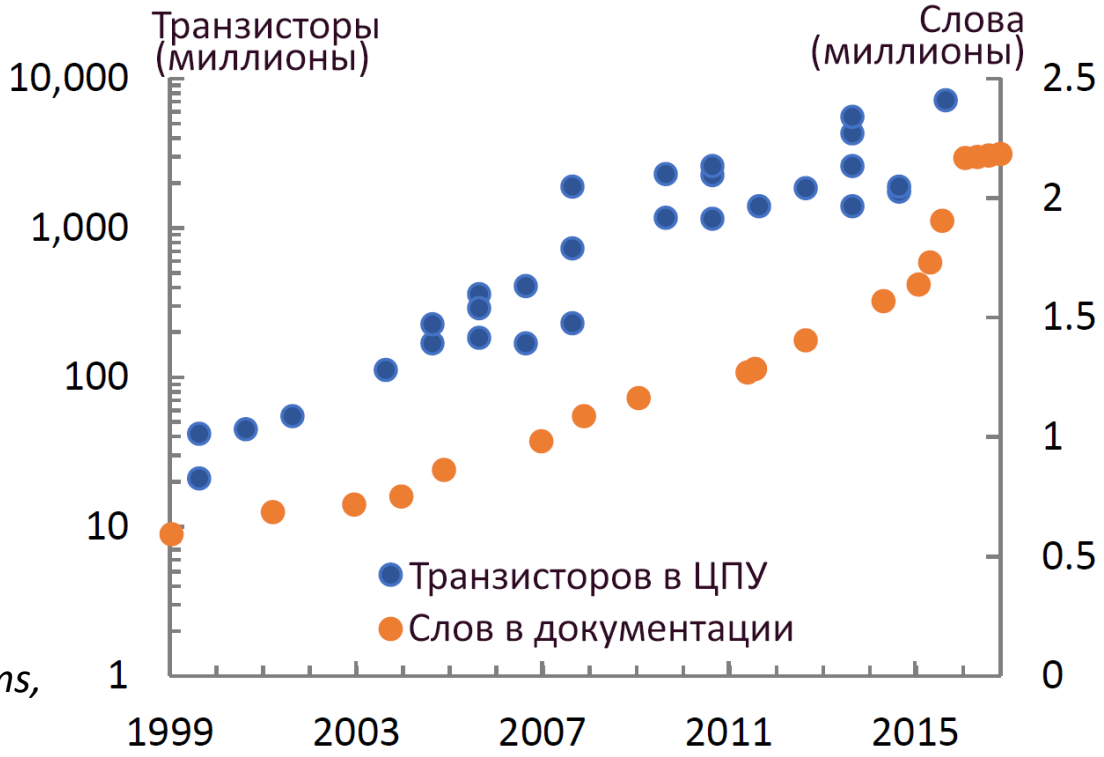
Производительность: особенности современной архитектуры Intel64 и не только

- Многоядерность
- Многоуровневый кэш, раздельный кэш кода и данных
- Суперскалярная архитектура
  - Несколько ***функциональных устройств***, одновременно обслуживающих этап выполнения
- Векторные команды MMX, 3DNow!, SSE, ...
- Внеочередное выполнение команд
- Предсказание переходов
- ...

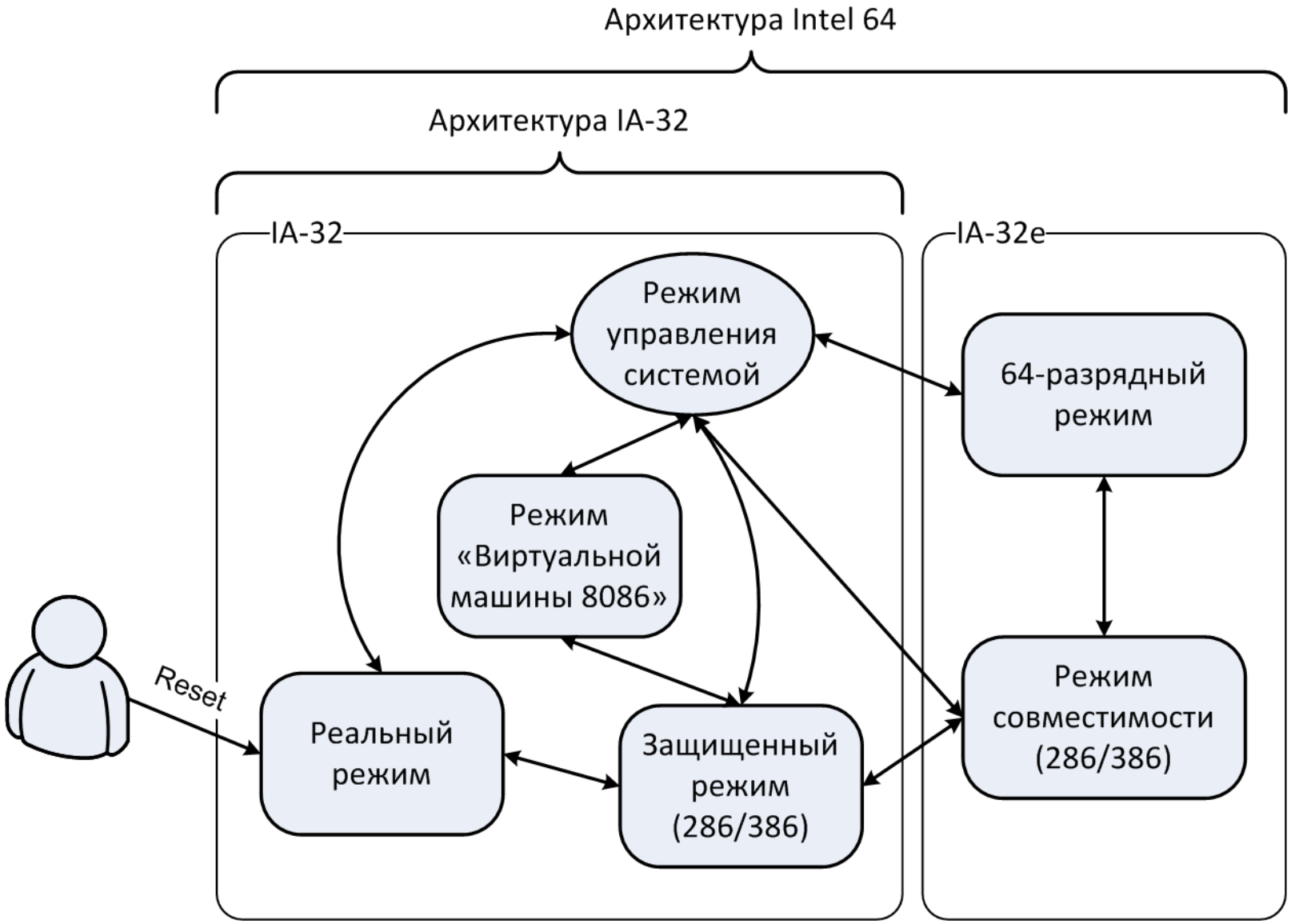


Развитие системы команд в архитектуре x86

- 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

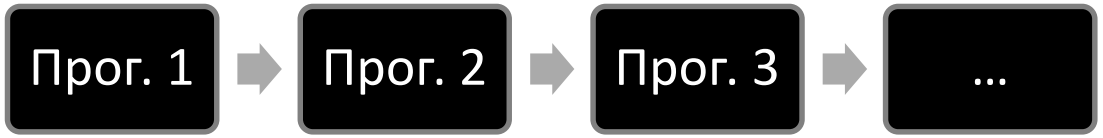


Начальная загрузка компьютера – последовательность (цепочка) выполнения все более сложных загрузчиков.
Конечная цель – загрузка операционной системы

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 = (сег_рег) << 4 + смещение
- Доступна вся ***адресуемая*** память
- Периферийные устройства управляются через порты ввода/вывода
  - Команды `in` и `out`

Обратная совместимость

- Обратная совместимость – свойство семейства процессоров. На более новом компьютере могут выполняться программы, рассчитанные на более ранние модели.

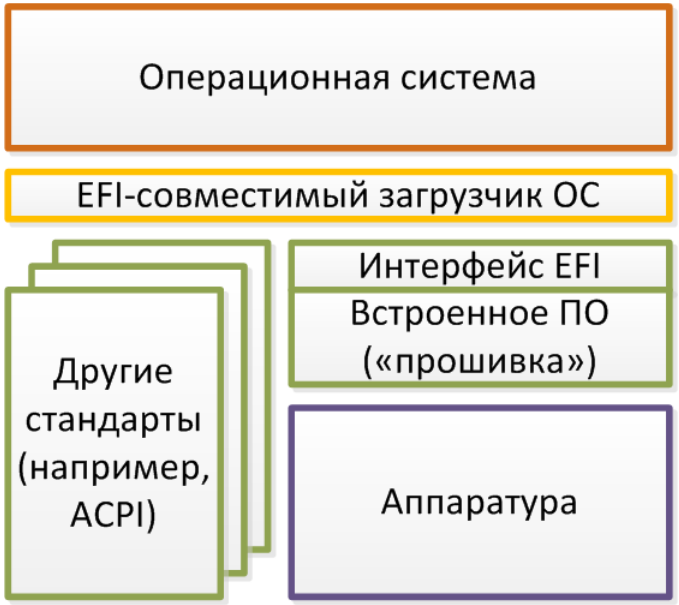
- Линия A20: 8086 → 80286
  - Первое поколение PC адресовало 1 МБ, 80286 – 16 МБ
  - Необходимо воспроизводить прежнее поведение
    - Современная аппаратура предоставляет три различных способа
  - Наиболее экзотический способ: через контроллер клавиатуры Intel 8042, более удобный – через Fast A20 Gate

```
; открываем адресную линию A20
in   al, 92h
or   al, 2
out  92h, al
```


https://www.win.tue.nl/~aeb/linux/kbd/A20.html

BIOS ⇨ UEFI

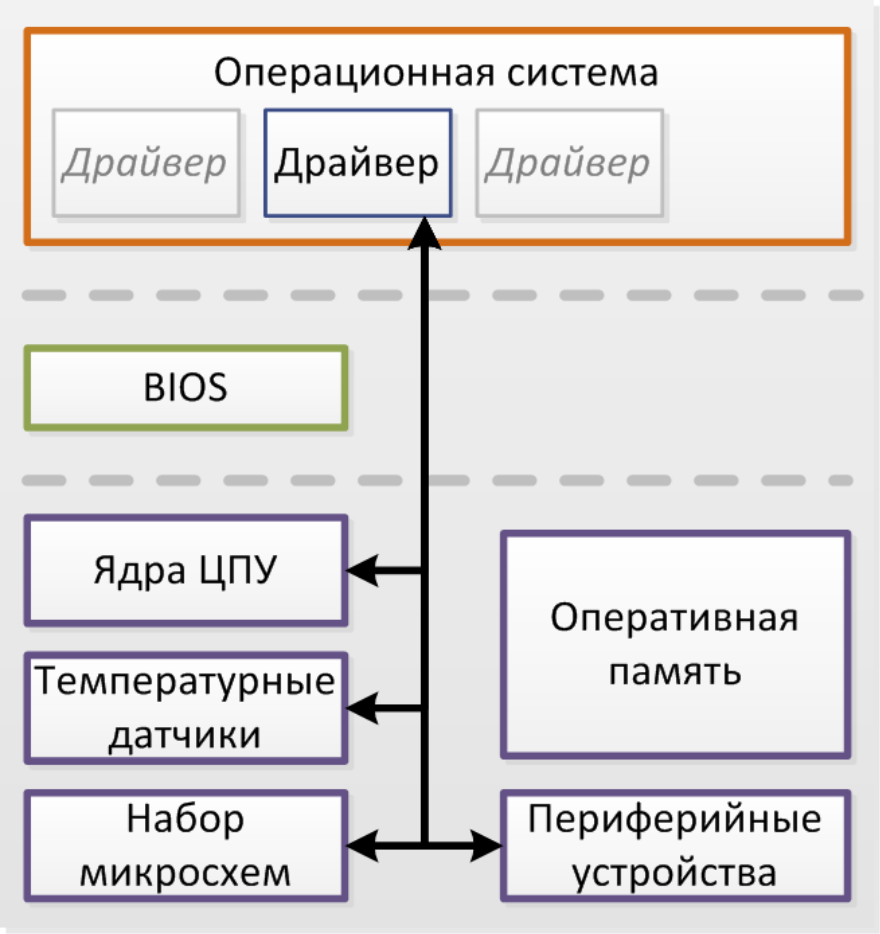
- UEFI – (Unified) Extensible Firmware Interface
- Функции BIOS не могут быть использованы, после перехода в защищенный режим
- UEFI – стандарт интерфейса встроенного ПО, которым может воспользоваться ОС
  - Модульная структура, компоненты можно добавлять или удалять
- Стандартизированные абстракции для работы с периферийными устройствами и шинами
- Спецификация состояния аппаратуры перед загрузкой ОС

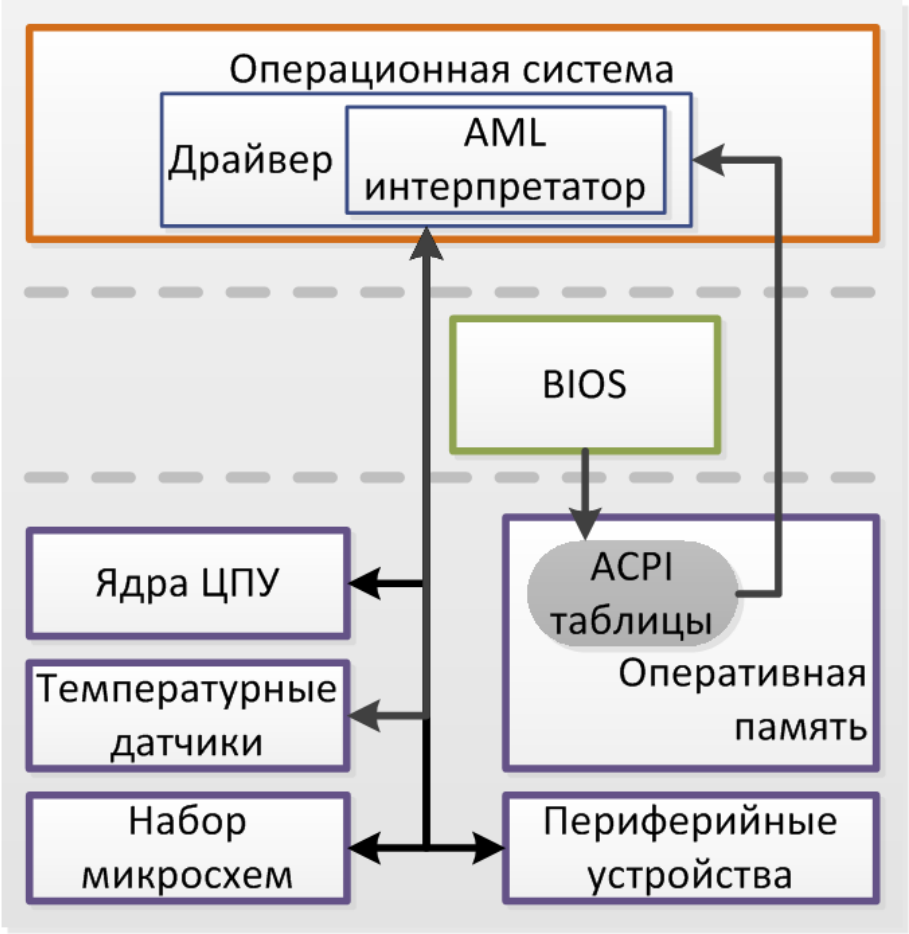


Стандарт ACPI

- ACPI – Advanced Configuration and Power Interface
- Проблема: управление энергопотреблением периферийных устройств и набора микросхем
  - ОС должна содержать драйвер для каждой модели материнской платы
- Решение: организовать драйвер в виде двух раздельных компонент: описания аппаратуры и интерпретатора этого описания
- BIOS размещает в оперативной памяти две группы ACPI таблиц
  - Перечисление имеющейся в компьютере аппаратуры и некоторых ее свойств
  - Набор микропрограмм, описывающих работу с устройствами
    - Архитектурно независимый язык AML
- В состав ОС входит интерпретатор языка AML

ACPI: упрощенная схема взаимодействия BIOS, ОС и аппаратуры





Многозадачная работа компьютера: требования к аппаратуре

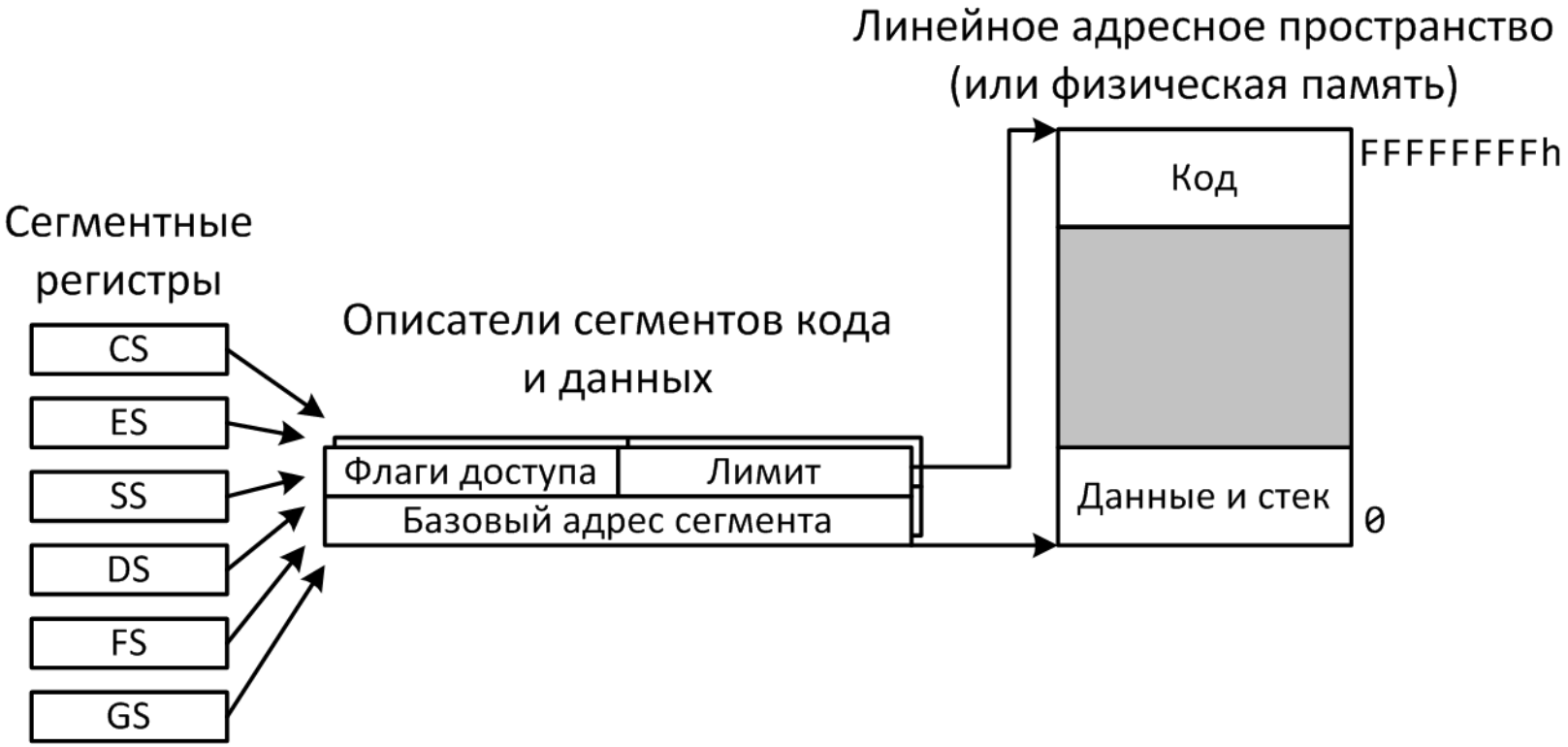
1. Привилегированный режим
   - Разделение машинных команд на две категории: пользовательские и привилегированные
2. Механизм защиты памяти
   - Изоляция кода/данных различных программ (и операционной системы) друг от друга
3. Таймер
   - Принудительное вытеснение программы с процессора
   - Многозадачность: невытесняющая и вытесняющая
4. Механизм прерываний
   - Возможность процессора отреагировать на возникновение некоторого события – передать управление на определенный код и изменить состояние регистров и памяти
     - (не)штатный ход выполнения очередной команды
     - сигналы от других устройств, в том числе – от таймера

Модели памяти в IA-32

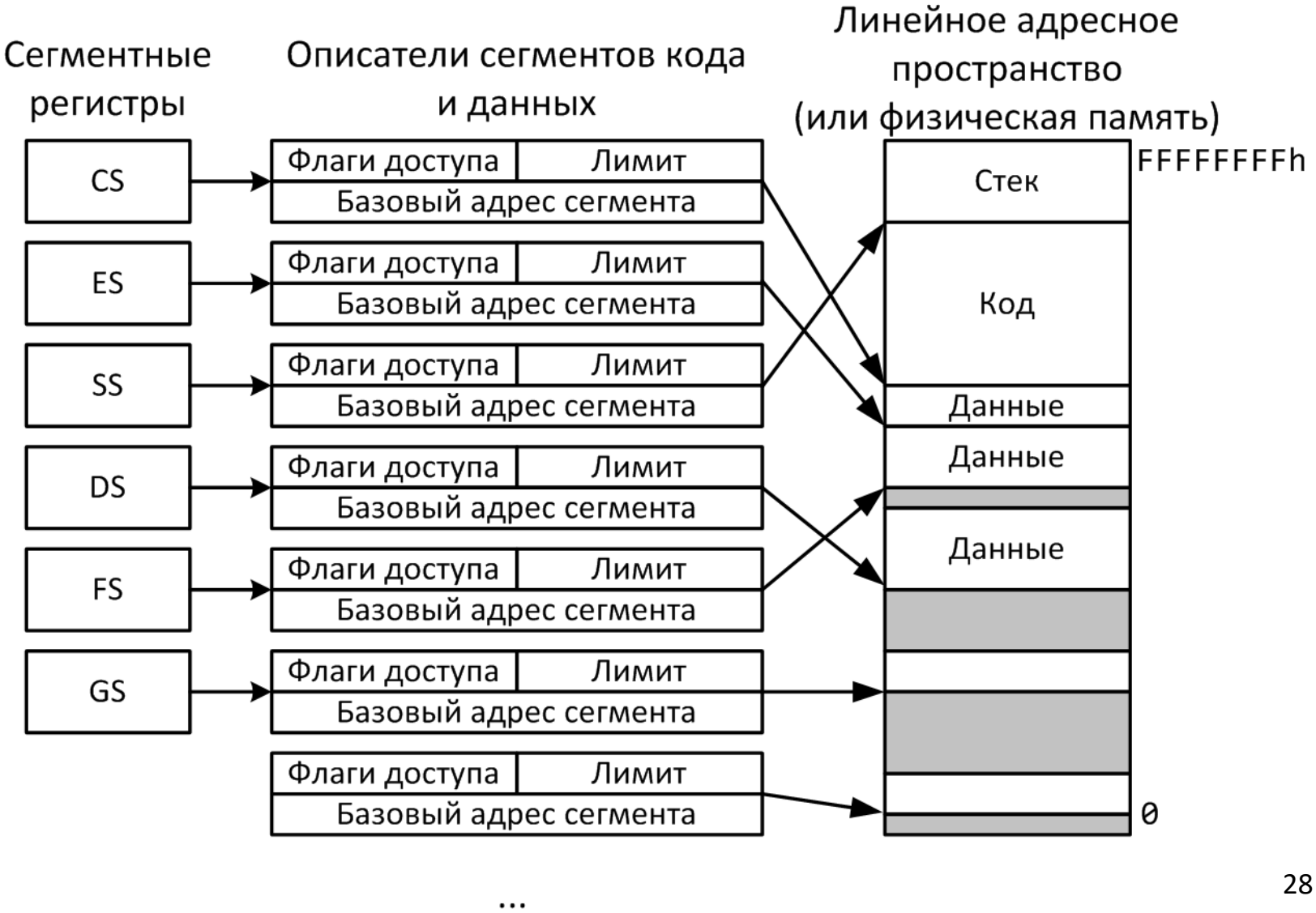
Исполнительный адрес ≡ Линейный адрес

- Реальный режим
  - Модель памяти с реальной адресацией
    линейный адрес = сегментный регистр << 4 + смещение
    Адрес физической памяти = линейный адрес
- Режимы защищенной памяти
  Дополнительный уровень косвенности: начальный (базовый) адрес сегмента определяется по таблице описателей сегментов, размещенной в памяти
  - Базовая плоская модель
    - Защищенная плоская модель
  - Мультисегментная модель
- *Независимо* от используемой модели памяти может быть включена страничная трансляция адресов

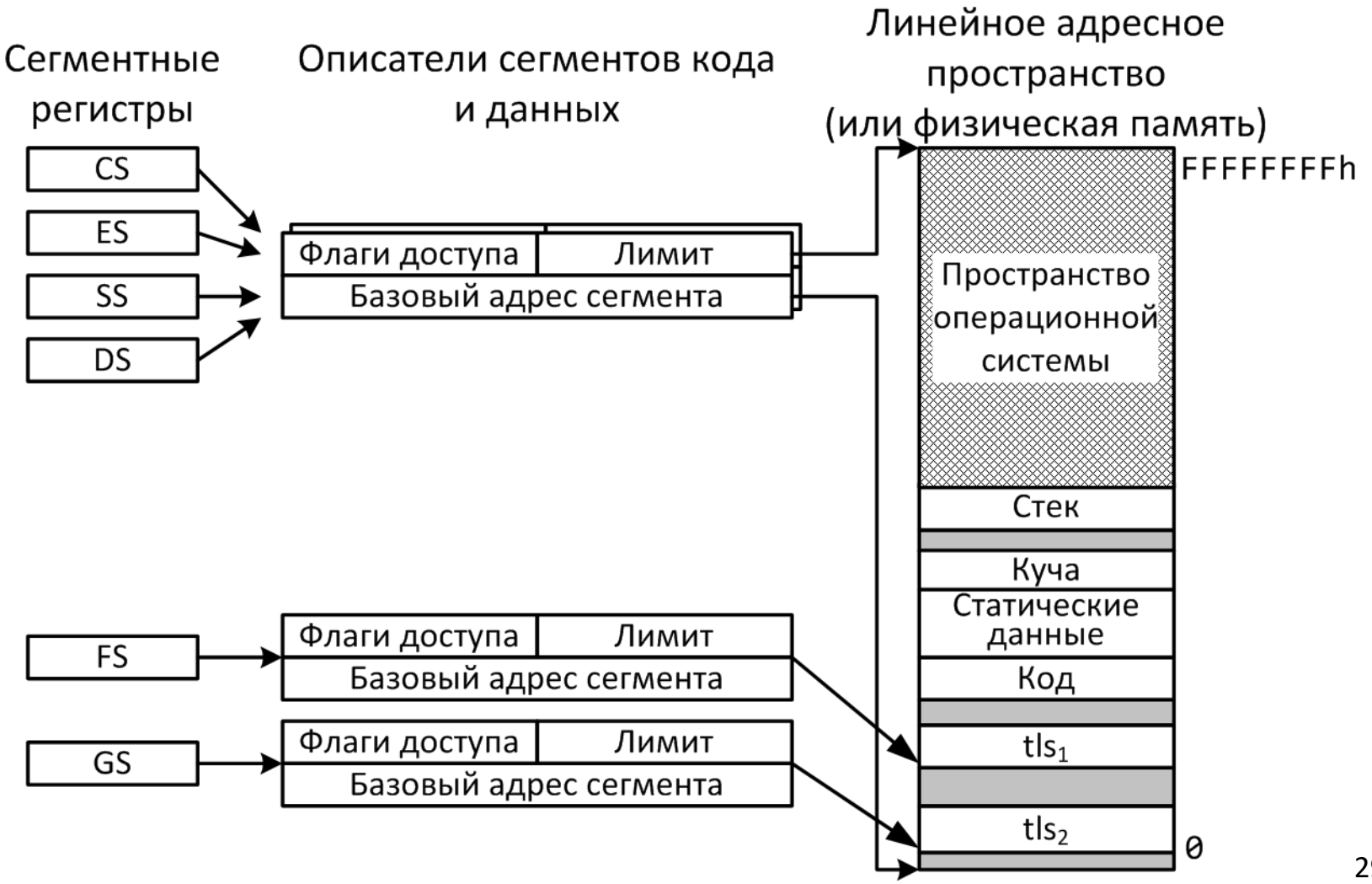
Плоская модель



Мультисегментная модель



А что в реальной жизни?

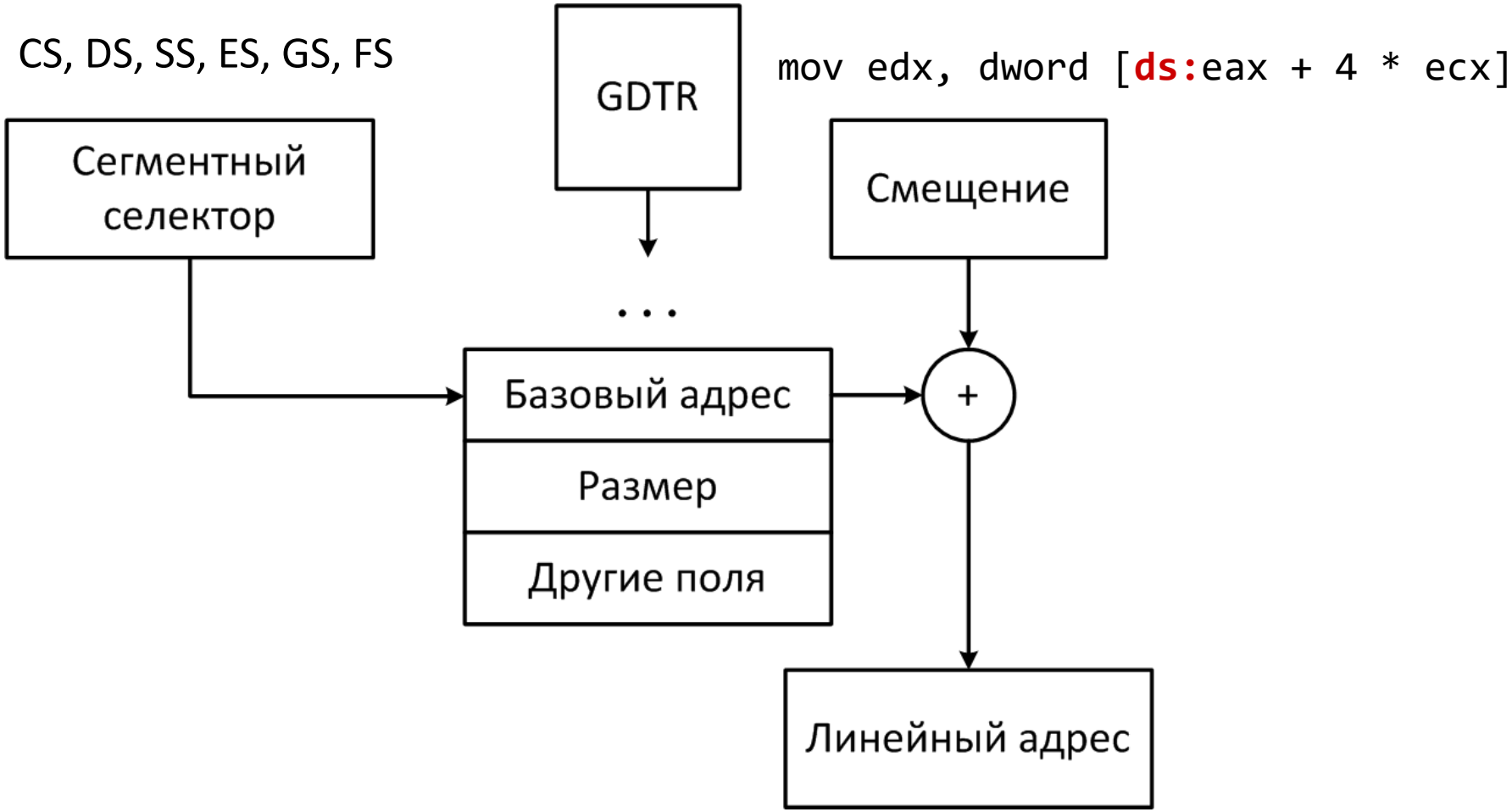


Защищенный режим / 80386

- Машинное слово и адрес 32 бита
- Два дополнительных сегментных регистра gs и fs
- Регистры управления CR***n***
  - ***CR3*** используется для управления доступом к памяти
- Аппаратная защита памяти
  - Базы таблиц дескрипторов GDTR, LDTR, IDTR
  - Мультисегментная модель памяти
    - Фактически не используется
  - Многоуровневая трансляция адреса памяти
- Аппаратное переключение задач
  - фактически не используется

Сегментные селектор и дескриптор

Адресация в защищенном режиме



Страничная организация памяти

- Нехватка физической памяти
- Изоляция одновременно работающих программ
- Линейные адреса автоматически преобразуются в физические адреса.
  - Память разделена на фрагменты-страницы одинакового размера.
  - Старшие биты адреса меняются на биты, взятые из соответствующей записи таблицы страниц
- Отображение адресов выполняет блок управления памятью (MMU)

Страничная организация памяти

Программа А

Таблица страниц

0	0
1	2
2	1
3	н/д
4	н/д
5	7

Виртуальная память

0	H	E	L	L
1	O		W	O
2	R	L	D	!
3				
4				
5	;	-	)	

Программа Б

Таблица страниц

0	3
1	5
2	6
3	4
4	н/д
5	7

Виртуальная память

0	H	A	V	E
1		L	O	T
2	S		O	F
3		F	U	N
4				
5	;	-	)	

Физическая память

0	H	E	L	L
1	R	L	D	!
2	O		W	O
3	H	A	V	E
4		F	U	N
5		L	O	T
6	S		O	F
7	;	-	)	

Номер страницы — Содержимое памяти

Номер страницы виртуальной памяти → Номер страницы физической памяти

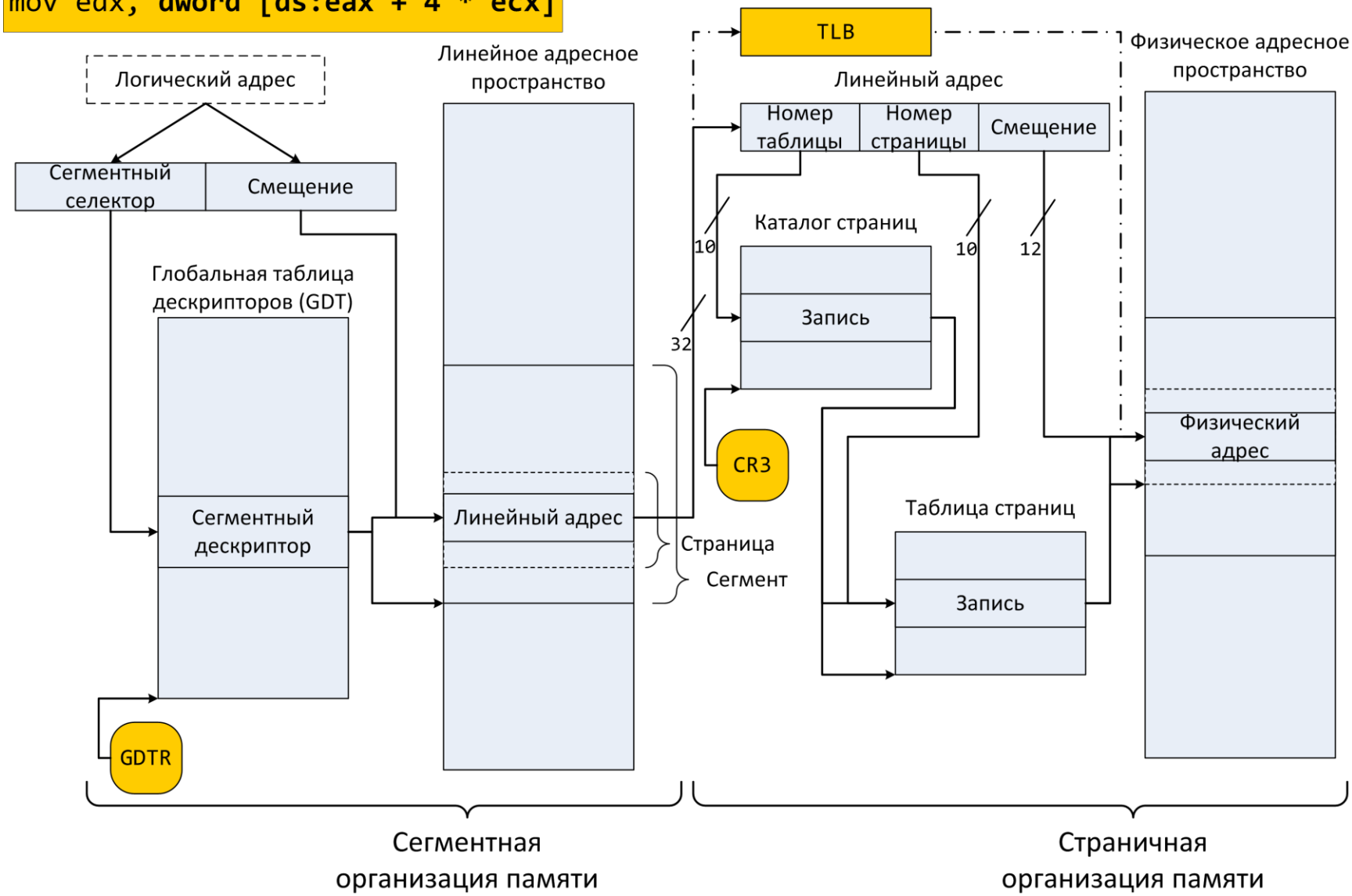
Пример заимствован https://wiki.osdev.org/Paging

Запись в таблице страниц (на примере 4 КБ страниц)

31 … 12	11 … 9	8	7	6	5	4	3	2	1	0
Адрес физической страницы		G	PAT	D	A	PCD	PWT	U/S	R/W	P

- P – присутствие страницы (1 – страница доступна)
- R/W – разрешение записи
- U/S – разрешение доступа к странице пользовательскому коду
- PCD – отключение кэширования
- A – признак того, что к странице обращались
- D – признак того, что в страницу записывали данные

```
mov edx, dword [ds:eax + 4 * ecx]
```



TLB – буфер ассоциативной трансляции

- TLB – translation look-aside buffer
- TLB кэширует записи таблицы страниц, т.е. является служебным (специализированным) кэшем
  - В отличие от кэша данных хранятся не блоки, а номера страниц
- Каждая запись в TLB содержит
  - Поле tag, сформированное из фрагмента номера страницы виртуальной памяти
  - Данные – запись таблицы страниц, в том числе номер страницы физической памяти
- TLB значительно меньше по размеру кэшей данных
  - Организация: от прямого отображения до полностью ассоциативного
  - Разделение на TLB инструкций и TLB данных, раздельные таблицы для пользовательский программ и операционной системы
  - Несколько уровней (размер/скорость)

Упрощенная схема извлечения данных из виртуальной памяти

Логический адрес

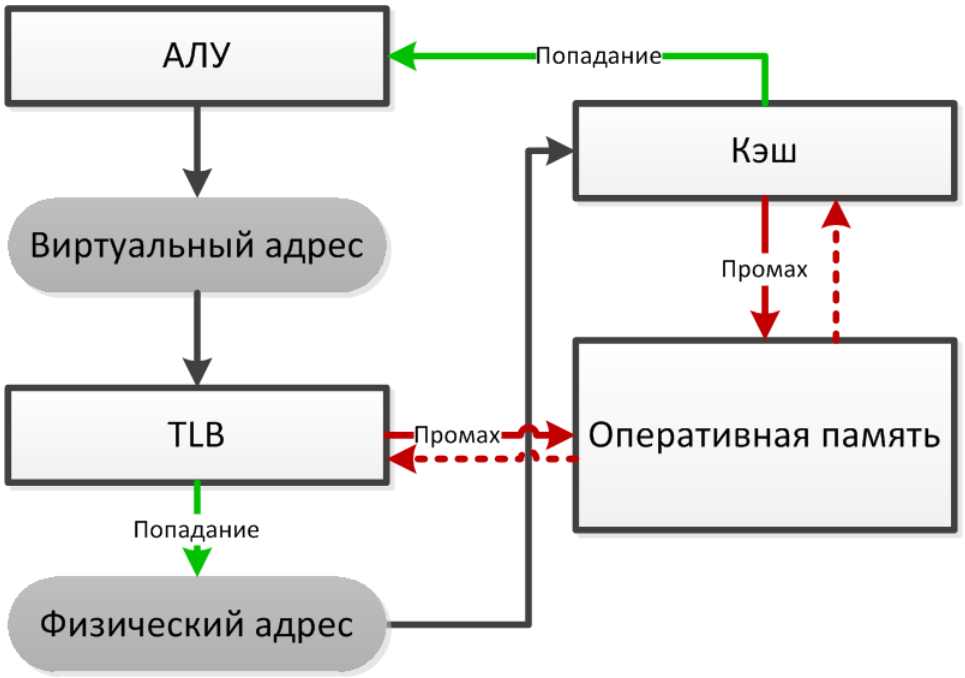
- Вычисляем смещение / адресный код в машинной команде
- Вычисляем линейный адрес / сегменты

Линейный адрес

- Пересчитываем линейный адрес в физический
  - Если есть запись в TLB берем готовый адрес
  - Если в TLB ничего нет, вычисляем физический адрес, проходя по таблицам трансляции и запоминаем его в TLB

Физический адрес

- Обращаемся в память за данными
  - Берем из кэша, если данные в нем присутствуют



Процент промахов в TLB: 0.1 – 1.0%

Задача: память модельного компьютера

- 256 адресуемых ячеек размером 1 байт
- Размер страницы – 16 байт
  - p – бит, показывающий наличие страницы
- TLB – 2-канальный множественно-ассоциативный кэш
  - v – бит, показывающий актуальность записи
- Кэш данных
  - Кэш прямого отображения, 4 байта в строке, 8 наборов
- Считываем ячейку памяти по виртуальному линейному адресу `0xcd`

Первые четыре записи в таблице страниц

VPN	PPN	p
0	-	0
1	f	1
2	2	1
3	-	0

Состояние TLB

Набор	tag	v	PPN	p	tag	v	PPN	p
0	2	1	-	0	6	1	0	1
1	0	1	f	1	3	1	-	0

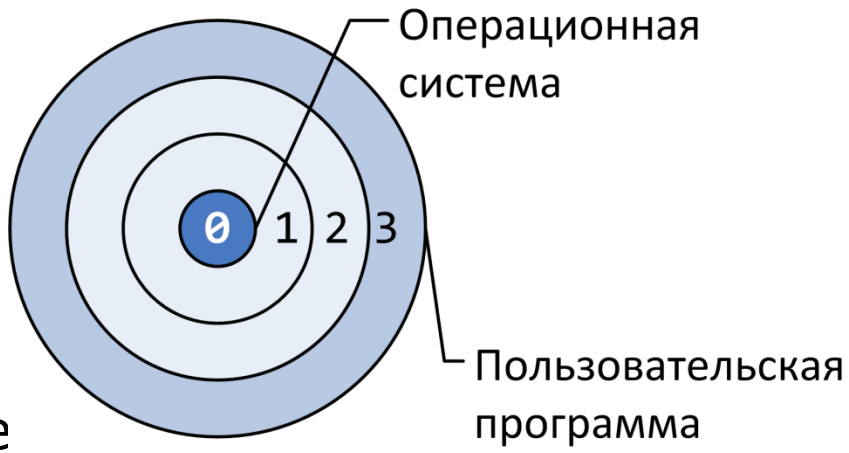
Кэш данных

Набор	tag	v
0	7	1
1	7	0
2	1	0
3	0	1
4	2	1
5	3	1
6	0	0
7	0	0

VPN – Номер страницы виртуальной памяти, PPN – Номер страницы физической памяти

Уровни (кольца) защиты

- Каждый уровень привилегий обладает своим контекстом: состоянием системных регистров и регистров общего назначения, стеком вызовов функций, …
  - Сменили уровень защиты – изменился ESP
- Уровни защиты 1 и 2, как правило, не используются
- Смена уровня привилегий – затратное по времени действие
- Переход между уровнями контролируется полями PL в дескрипторах сегментов
- Возможность выполнения привилегированных команд определяется текущим уровнем привилегий



Типы и классы прерываний

- Два типа прерываний – один механизм обработки
  - Асинхронные прерывания – события, происходящие в периферийных устройствах (Ввод/Вывод).
  - Синхронные прерывания (исключения) – возникновение определенных ситуаций при выполнении команд процессором

Класс	Причина	Синх./ Асинх.	Возврат управления
Прерывания	Сигнал от периферийного устройства (ввод/вывод)	Асинх.	Всегда возвращаемся на следующую команду
Ловушки (Trap)	Преднамеренный выброс исключения	Синхр.	Всегда возвращаемся на следующую команду
Сбои (Fault)	Потенциально восстанавливаемая ошибка	Синхр.	Возможно возвращаемся на текущую команду
АвОст (Abort)	Неустранимая ошибка	Синхр.	Никогда не возвращаем управление

Исключения

В других процессорных архитектурах терминология может незначительной отличаться

Примеры некоторых прерываний

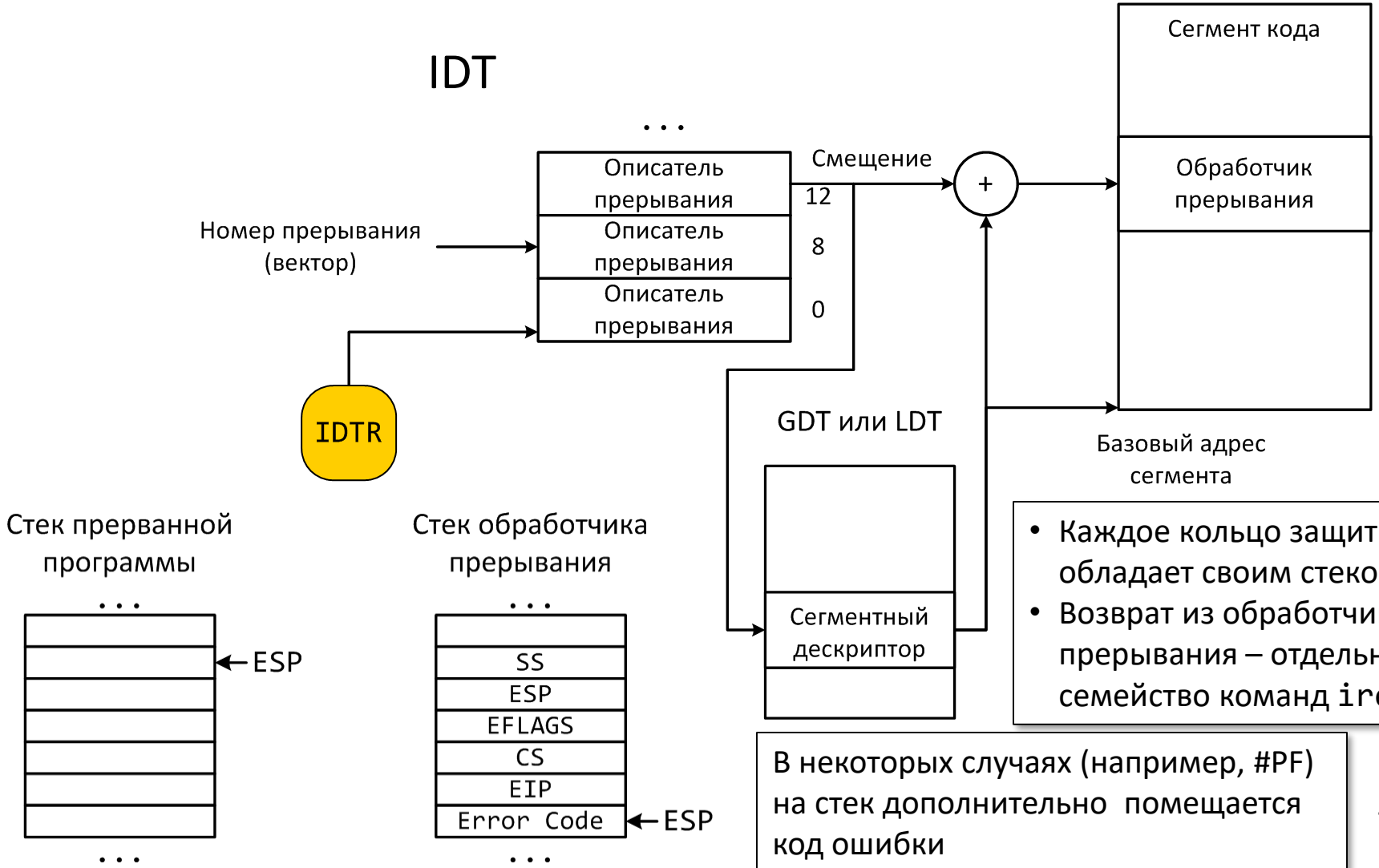
Вектор (№)	Мнемоника	Описание	Источник
0	#DE	Ошибка при делении	Команды DIV и IDIV
3	#BP	Точка останова	Команда INT 3
13	#GP	Сбой защиты	Любые ситуации, связанные с нарушением защиты памяти
14	#PF	Отсутствие страницы	Обращение к отсутствующей странице
16	#MF	Сбой в x87 FPU	Команды x87
18	#MC	Непоправимый сбой в работе аппаратуры	Механизм самопроверки аппаратуры
32-255	–	Определяется пользователем	Внешние прерывания или команда INT *n*

Какие классы у перечисленных прерываний?

Реакция на прерывание

- При возникновении прерывания управление передается обработчику, на заданный адрес. После обработки управление может быть возвращено в прерванную программу.
  - ***В этот момент можно повысить уровень привилегий!***
  - Для определения адреса обработчика используется ***номер прерывания*** (***вектор***) и ***таблица описателей прерываний*** (***IDT***)
  - Будет возвращено управление или нет – зависит от события, приведшего к возникновению прерывания
- Некоторые прерывания можно маскировать
- «Доставку» прерываний в процессор выполняет ***программируемый контроллер прерываний*** (***PIC***)
  - Local APIC, I/O APIC
  - Контроллер прерываний содержит таймер:
    через заданное пользователем число тактов будет выбрасываться прерывание от таймера

Передача управления в обработчик прерывания в защищенном режиме



- Каждое кольцо защиты обладает своим стеком
- Возврат из обработчика прерывания – отдельное семейство команд `iret`

В некоторых случаях (например, #PF) на стек дополнительно помещается код ошибки

Системные вызовы

- Аппаратура, память операционной системы и других программ не доступны
  - Привилегированные команды
  - Страничная трансляция адресов
- Системный вызов – основной способ «обратиться» к операционной системе
  - `syscall/sysret`
  - `sysenter/sysexit`
  - **`int/iret`**
- В ОС Linux системный вызов реализован как прерывание
  - `int 0x80`
  - Передача параметров – через регистры:
    номер функции – `eax`
    параметры – `ebx`, `ecx`, `edx`, `esi`, `edi`

Примеры некоторых системных вызовов ОС Linux

EAX	Название	EBX	ECX	EDX
1	sys_exit	int	–	–
3	sys_read	unsigned int	char *	size_t
4	sys_write	unsigned int	const char *	size_t
5	sys_open	const char *	int	int
6	sys_close	unsigned int	–	–
116	sys_sysinfo	struct sysinfo *	–	–

Источник: http://docs.cs.up.ac.za/programming/asm/derick_tut/syscalls.html (Eng)

Дальнейшие и дополнительные материалы

- Основной курс 3 семестра «Операционные системы»
- Коллекция статей об устройстве ОС http://wiki.osdev.org/Expanded_Main_Page (Eng)

```
snoop@earth:~/samples$ nasm -f elf32 -o syscall.o syscall.asm
snoop@earth:~/samples$ nm syscall.o
00000000 T _start
00000000 d msg
0000000e a msg_len
snoop@earth:~/samples$ ld -o syscall syscall.o
snoop@earth:~/samples$ nm syscall
080490b2 A __bss_start
080490b2 A _edata
080490b4 A _end
08048080 T _start
080490a4 d msg
0000000e a msg_len
snoop@earth:~/samples$ ./syscall
Hello, world!
snoop@earth:~/samples$
```

```
section .data
  msg db `Hello, world!\n`
  msg_len equ $-msg

section .text
global _start
_start:
   mov eax, 4
   mov ebx, 1
   mov ecx, msg
   mov edx, msg_len
   int 80h

   mov eax, 1
   mov ebx, 0
   int 80h
```

```
#include <unistd.h>
#include <stdlib.h>

void main() {
   write(1, "Hello, world!\n", 14);
   exit(0);
}
```