Архитектура x86
Дополнительные главы

М. А. Соловьев
ИСП РАН
Осенний семестр 2017 г.

Раздел 1

ВСПОМНИТЬ ВСЁ

Логический адрес

- *Логический адрес* в x86 состоит из двух частей:
  - сегмент — CS, DS, ES, FS, GS или SS;
  - смещение — результат вычисления адресного выражения.
- Сегмент может быть не указан явно, тогда он выбирается по умолчанию:
  - при выборке команд — CS (в паре с RIP);
  - при использовании базы RSP или RBP — SS;
  - при использовании части строковых команд — ES;
  - в остальных случаях — DS.
- Пример:
  - `MOV RAX, QWORD [RBP + 16]` `=> SS`
  - `MOV RAX, QWORD [RCX + 16]` `=> DS`

Сегментная трансляция

- Логический адрес подлежит *сегментной трансляции:*
  - выполняется всегда;
  - выполняется до страничной трансляции, если она включена;
  - результат — линейный адрес — виртуальный или физический.
- Сегментная трансляция:
  - проверка корректности адреса:
    - смещение не должно быть больше, чем *лимит*;
    - тип доступа должен соответствовать *атрибутам* и DPL;
  - линейный адрес — сумма *базы* и смещения.

Реальный режим

- Реальный режим повторяет поведение 16-разрядной машины 8086.
- База любого сегмента вычисляется как значение *селектора*, умноженное на 16.
- Лимит любого сегмента — константа 65535.
- Таким образом может адресоваться 1 Мбайт памяти, шина адреса имеет ширину 20 бит. 20-й бит, если он окажется выставленным после сложения, отбрасывается. Линия A20.
- Пример:
  - ```
    MOV AX, 0xD800
    MOV DS, AX
    MOV AX, WORD [0x0100]          => D8100
    ```
  - `0xFFFF:0xFFFF` => `0FFEF`
    ```

Защищенный режим

- Защищенный режим 80386 использует *глобальную* (GDT) и *локальные* (LDT) *таблицы дескрипторов* для конфигурации сегментов.
- Таблицы хранятся в оперативной памяти, работа через команды LGDT, SGDT, LLDT, SLDT.
- Каждая запись, соответствующая сегменту, компактно задаёт его базу, лимит и атрибуты (в т.ч. DPL) в 8-байтовом формате.
- При выборе сегмента в качестве селектора используется смещение от начала таблицы до нужного сегмента:
  - младшие два бита хранят признак RPL;
  - следующий бит хранит индикатор таблицы TI — GDT или LDT.
- При загрузке селектора база, лимит и атрибуты кешируются в теневых регистрах процессора.
- Механизм использования таблиц включается битом CR0.PE.

Плоская адресация

- В современных операционных системах сегментная трансляция не применяется.
- Тем не менее, аппаратура требует настройки таблицы сегментов.
- Сегменты настраиваются в соответствии с моделью *плоской адресации*.
- Общие характеристики сегментов при плоской адресации:
  - база — 0;
  - лимит — (4 Гбайт – 1 байт).
- Сегменты кода помечены как исполняемые, сегменты данных — как записываемые.
- В простом случае достаточно четырёх сегментов: код ядра, данные ядра, код пользователя, данные пользователя.

Раздел 2

НАЧАЛО РАБОТЫ ПРОЦЕССОРА ПОСЛЕ СБРОСА

Сигнал начала работы

- Аппаратный способ (RESET):
  - RESET# — «нога» процессора с логическим сигналом;
  - передний фронт: сброс состояния процессора;
  - задний фронт: начало выборки и выполнения команд;
  - автоматически подаётся при подаче питания.
- Программный способ (INIT):
  - последовательность межпроцессорных прерываний (IPI);
  - подробнее будет рассмотрен далее.

Встроенная самопроверка

- Встроенная самопроверка (BIST) выполняется после RESET:
  - проверка работы кешей команд и данных;
  - проверка работы кеша трансляций TLB;
  - проверка содержимого встроенной постоянной памяти процессора:
    - микрокод;
    - таблицы констант для математических операций;
  - механизм предсказания ветвлений.
- BIST требует порядка нескольких миллионов циклов.
- Результат — код ошибки или ноль как признак успеха.

Многоядерный процессор

- При переходе нескольких ядер многоядерного процессора в состояние RESET происходит распределение ролей.
- Одно ядро назначается *загрузочным* (BSP) и после выхода из состояния RESET переходит к выборке и выполнению команд.
- Остальные ядра назначаются *прикладными* (AP) и не начинают выборку и выполнение команд до запуска программным способом (INIT).

Начальное состояние: RESET

Аспект	Начальное состояние
RAX	Код результата BIST
RDX	Семейство, модель, версия процессора
Остальные РОН	0000_0000_0000_0000
Регистры x87	0.0
Регистры XMM	0
Кеши команд и данных	сброшены
Кеш трансляций TLB	сброшен
MTRR	механизм выключен
PAT	0007_0406_0007_0406
IORR	механизм выключен
TOP_MEM / TOLUD	0000_0000_0400_0000
TOP_MEM2 / TOUUD	механизм выключен
SMBASE	0000_0000_0003_0000

Начальное состояние: RESET

Аспект	Начальное состояние
CR0	`0000_0000_6000_0010`
CR2, CR3, CR4, CR8, EFER	`0000_0000_0000_0000`
RFLAGS	`0000_0000_0000_0002`
RIP	`0000_0000_0000_FFF0`

CS	`Селектор:  F000`
	`База/лимит: 0000_0000_FFFF_0000 + FFFF`
	`Атрибуты:  16-битный код, DPL 0`

DS, ES, FS, GS, SS	`Селектор:  0000`
	`База/лимит: 0000_0000_0000_0000 + FFFF`
	`Атрибуты:  16-битные данные, DPL 0`

GDTR, IDTR	`База/лимит: 0000_0000_0000_0000 + FFFF`

LDTR, TR	`Селектор:  0000`
	`База/лимит: 0000_0000_0000_0000 + FFFF`
	`Атрибуты:  TSS «занята»`

Раздел 3

КОНТРОЛЛЕР ПРЕРЫВАНИЙ

Контроллер прерываний

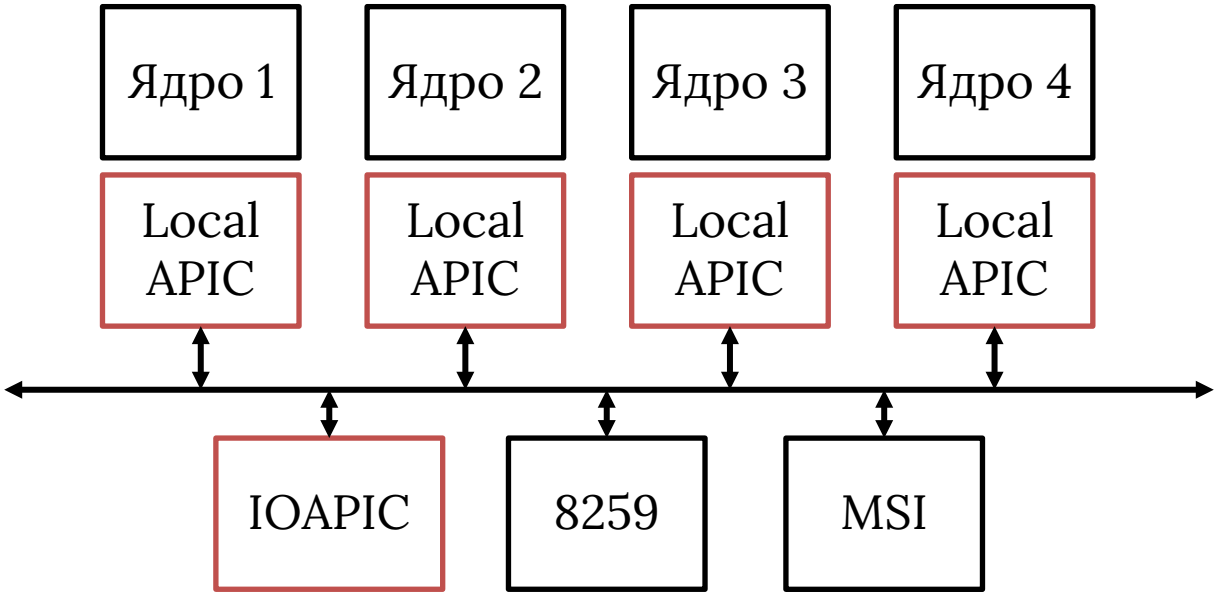
- *Контроллер прерываний* — устройство, которое обрабатывает входящие прерывания, назначает им приоритеты и передаёт на обработку процессору.
- Каждому прерыванию соответствует своя физическая *линия*.
- Варианты сигнализации:
  - по фронту (edge);
  - по высокому уровню (active high);
  - по низкому уровню (active low).
- Устаревший контроллер 8259:
  - обычно представлен в виде каскада двух чипов;
  - передаёт прерывания на загрузочное ядро;
  - работает либо с фиксированными приоритетами, либо с «карусельной» схемой.

APIC

- *Advanced Programmable Interrupt Controller* — современная реализация контроллера прерываний с двумя типами устройств.
- Каждое процессорное ядро имеет свой *локальный* APIC.
- Регистры локального APIC отображены на физическую память в окне размером 4 Кбайт:
  - адрес окна может быть настроен системным кодом через MSR 0000_001B;
  - адрес по умолчанию — 0000_0000_FEE0_0000.
- Помимо локальных APIC для каждого ядра, в системе есть одно или более устройство IO APIC.

IOAPIC

- IOAPIC имеет подключение к внешним линиям прерываний и транслирует их в сообщения системной шины, адресатами которых являются локальные APIC.



Источники прерываний

- Локальный APIC имеет следующие источники прерываний:
  - прерывания ввода-вывода со стороны IOAPIC (IRQ);
  - устаревшие прерывания со стороны 8259 (в т.ч. NMI);
  - MSI (Message-Signaled Interrupts);
  - межпроцессорные прерывания со стороны другого (или того же самого) локального APIC;
  - локальные прерывания:
    - прерывание по таймеру;
    - прерывание по счётчикам производительности;
    - прерывание по сенсорам температуры;
    - другие локальные прерывания.

Типы сообщений шины

- При получении прерывания в зависимости от настроек системы выбранный локальный APIC получает сообщение системной шины одного из следующих типов:
  - *fixed* — всем ядрам из набора *адресов* с заданным вектором;
  - *lowest priority* — одному ядру из набора адресов с заданным вектором, выбирается наименее занятое;
  - *SMI* — System Management Interrupt, отдельный тип прерывания, будет рассмотрен далее;
  - *NMI* — Non-Maskable Interrupt, отдельный тип прерывания для «непоправимых ситуаций»;
  - *INIT* — сброс состояния всех ядер из набора адресов;
  - *STARTUP* — начало выборки и выполнения команд.

Сброс состояния: INIT

Аспект	Начальное состояние
RAX	0000_0000_0000_0000
RDX	Семейство, модель, версия процессора
Остальные РОН	0000_0000_0000_0000
Регистры x87	без изменений
Регистры XMM	без изменений
Кеши команд и данных	без изменений
Кеш трансляций TLB	без изменений
MTRR	без изменений
PAT	без изменений
IORR	без изменений
TOP_MEM / TOLUD	без изменений
TOP_MEM2 / TOUUD	без изменений
SMBASE	без изменений

Сброс состояния: INIT

Аспект	Начальное состояние
CR0	0000_0000_?000_0010 (сохранены CD, NW)
CR2, CR3, CR4, EFER	0000_0000_0000_0000
CR8	без изменений
RFLAGS	0000_0000_0000_0002
RIP	0000_0000_0000_FFF0
CS	Селектор: F000 База/лимит: 0000_0000_FFFF_0000 + FFFF Атрибуты: 16-битный код, DPL 0
DS, ES, FS, GS, SS	Селектор: 0000 База/лимит: 0000_0000_0000_0000 + FFFF Атрибуты: 16-битные данные, DPL 0
GDTR, IDTR	База/лимит: 0000_0000_0000_0000 + FFFF
LDTR, TR	Селектор: 0000 База/лимит: 0000_0000_0000_0000 + FFFF Атрибуты: TSS «занята»

Старт прикладных ядер

- Запуск прикладных ядер в работу осуществляет загрузочное ядро.
- Используется последовательность межпроцессорных прерываний:
  - *INIT*, чтобы перевести прикладное ядро в начальное состояние;
  - *STARTUP*, чтобы начать выборку и выполнение команд;
  - некоторые старые процессоры требуют двух прерываний *STARTUP*;
  - требуется определённая задержка между прерываниями.
- В STARTUP номер вектора интерпретируется иначе: вектор *vv* соответствует начальному значению селектора CS *vv*00.

Старт прикладных ядер

- Последовательность действий загрузочного ядра для старта прикладных ядер:
  - инициализация локального APIC;
  - подготовка начального кода для прикладных ядер;
  - рассылка широковещательного сообщения *INIT*;
  - рассылка широковещательного сообщения *STARTUP*;
  - каждое работоспособное прикладное ядро начинает выполнять начальный код:
    - инициализация процессора и локального APIC;
    - внесение сведений о процессоре в таблицы в памяти (под семафором);
    - приостановка работы (CLI, MONITOR, MWAIT).

Раздел 4

ОТОБРАЖЕНИЕ УСТРОЙСТВ НА ПАМЯТЬ

Порты ввода-вывода

- Компьютер состоит из большого количества различных взаимодействующих устройств, объединённых системой шин.
- В современном исполнении даже процессор представляет собой множество устройств, в т.ч. и PCI-устройства.
- Изначально в 8086 для взаимодействия с устройствами был реализован механизм *портов ввода-вывода* — отдельное 16-разрядное пространство адресов и команды для работы с ним: IN, OUT, INS, OUTS.
- Ввод-вывод через порты всегда явный — инициатива по взаимодействию исходит со стороны процессора. Такой ввод-вывод называется *программным* (PIO).

Недостатки PIO

- Схема PIO логически простая и достаточно удобная, но не масштабируется:
  - «пространство» — количество портов ограничено архитектурно, а одному устройству может быть нужно несколько портов;
  - «время» — каждое взаимодействие с устройством требует выполнения команды:
    - как быстро можно прочитать сетевой пакет?
    - как быстро можно обновить картинку на экране?
- Параллельно с PIO применяется также схема MMIO, когда устройство отображается на физическую память:
  - классический пример — видеоадаптер VGA;
  - сегменты A000, B000, B800.

MMIO и DMA

- Близкое понятие — *прямой доступ к памяти* (DMA).
- Без DMA устройство имеет только некоторые назначенные диапазоны адресов фиксированного размера, доступ к которым расценивается как доступ к регистрам устройства.
- С DMA устройство имеет возможность читать и писать произвольные байты физической памяти:
  - сетевые карты;
  - звуковые карты;
  - контроллеры накопителей.
- Наличие механизма DMA вызывает вопросы, связанные с безопасностью.

Контроллер памяти

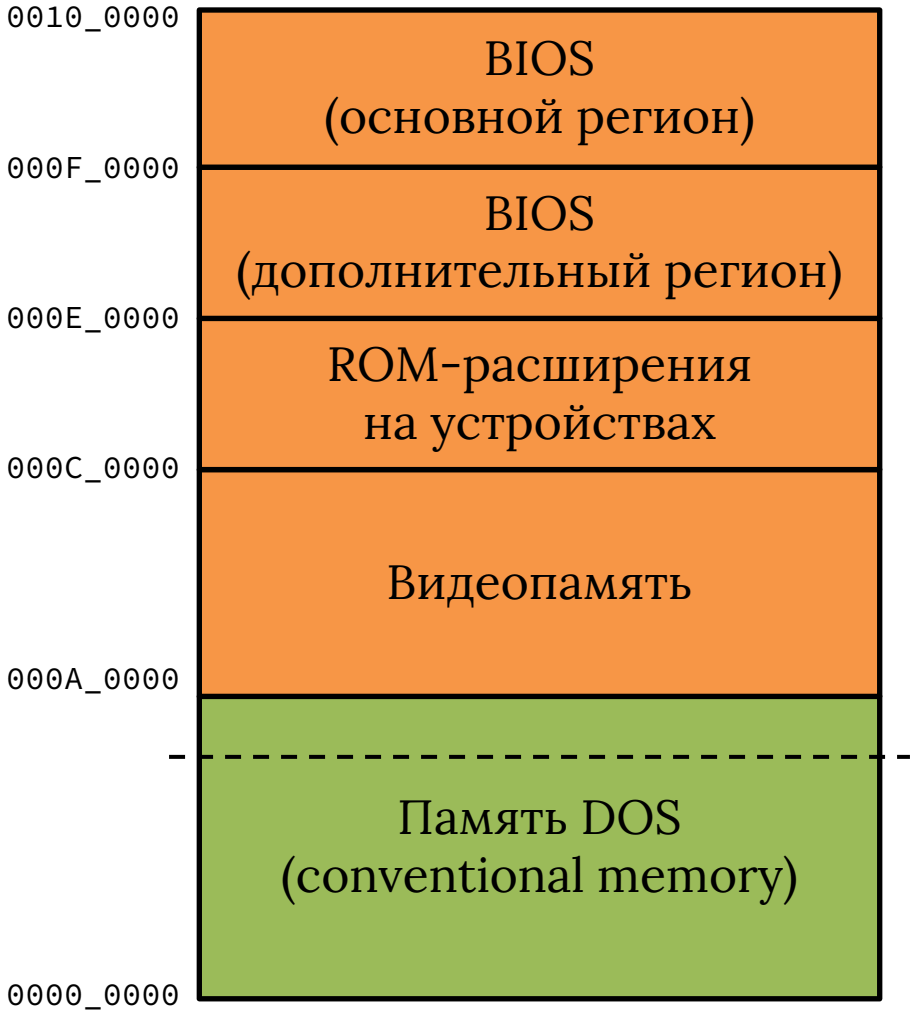
- *Контроллер памяти* — устройство, отвечающее за обработку и доставку сообщений системной шины, связанных с доступом к физической памяти:
  - чтение блока памяти;
  - запись блока памяти;
  - цикл read-modify-write над блоком памяти;
  - другие сообщения.
- Контроллер памяти может быть частью самого процессора, а может быть размещён в чипсете («северный мост»).
- Любое обращение к памяти в конечном счёте превращается в транзакцию системной шины, которую обрабатывает контроллер памяти. Есть исключение — какое?
- Контроллер должен знать, куда переслать сообщение.

Карта физической памяти

- Далее мы будем по частям рассматривать карту физической памяти современного компьютера:
  - регион с адресами до 1 Мбайт;
  - регион с адресами от 1 Мбайт до 4 Гбайт;
  - регион с адресами от 4 Гбайт.
- Для каждого диапазона должно определяться, отправляются ли соответствующие транзакции на DRAM или в сторону остальных устройств.

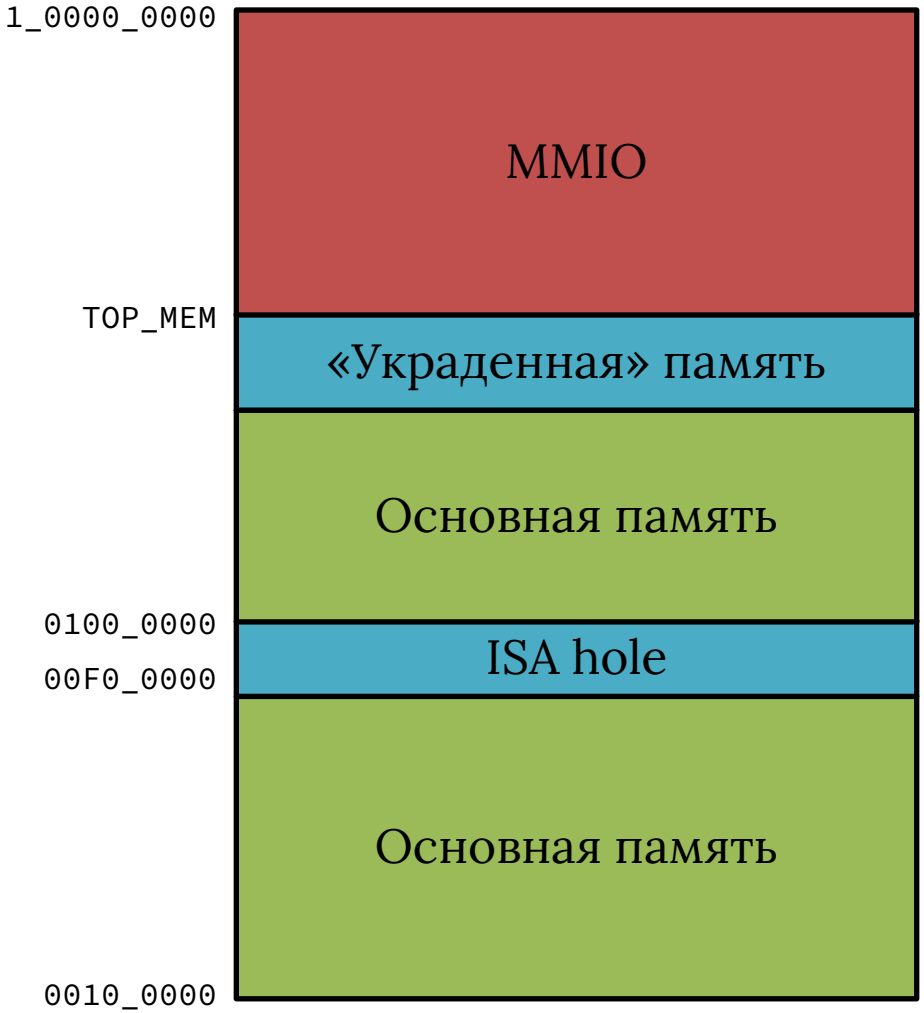
Нижний регион памяти

- Прекрасные времена DOS, когда всё было очень просто.
- Память DOS:
  - всегда DRAM.
- Видеопамять:
  - либо видеоадаптер (CGA, EGA, VGA);
  - либо DRAM.
- ROM-расширения:
  - либо ПЗУ устройств;
  - либо DRAM.
- BIOS:
  - либо ПЗУ BIOS;
  - либо DRAM.



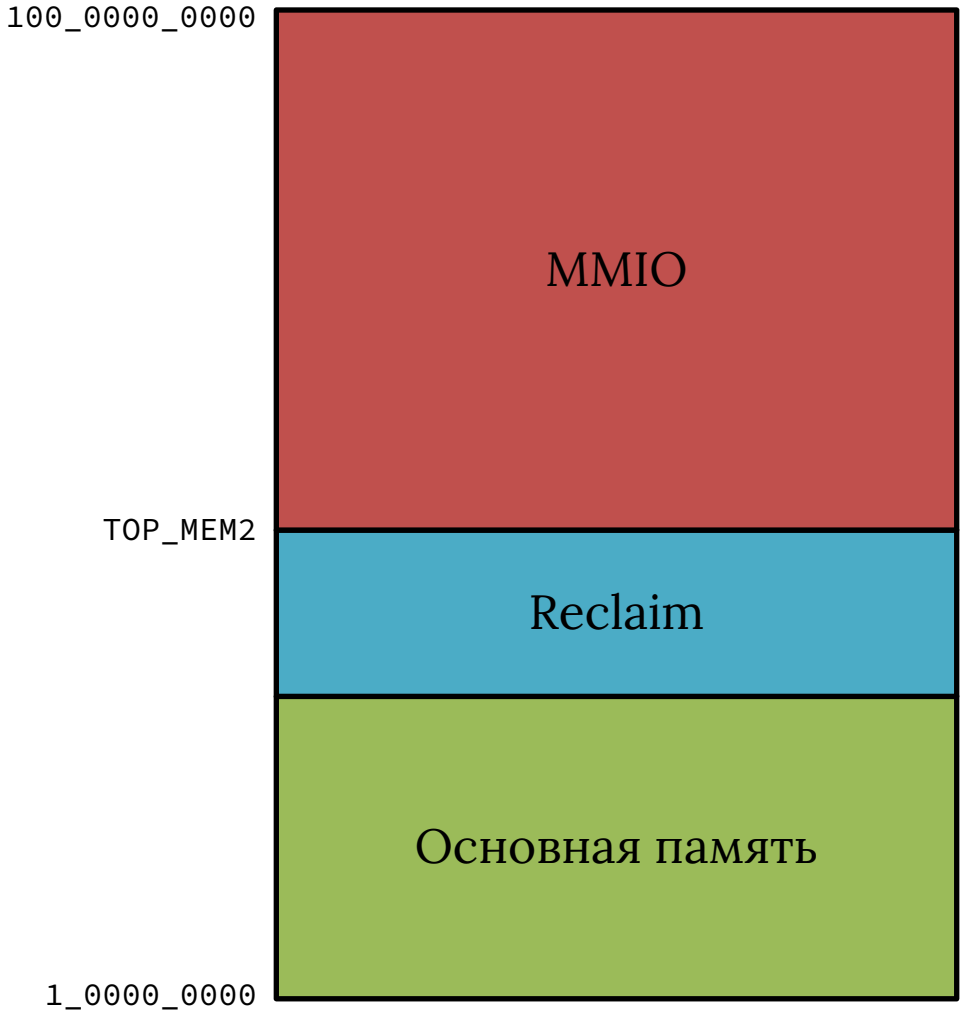
Средний регион памяти

- Картина до 64 разрядов.
- Основная память:
  - всегда DRAM;
  - часть может быть «украдена» на нужды встроенного GPU, ME, SMM.
- ISA hole — теневая память для ISA-устройств, давно не актуальна.
- MMIO:
  - диапазон для работы с различными устройствами.

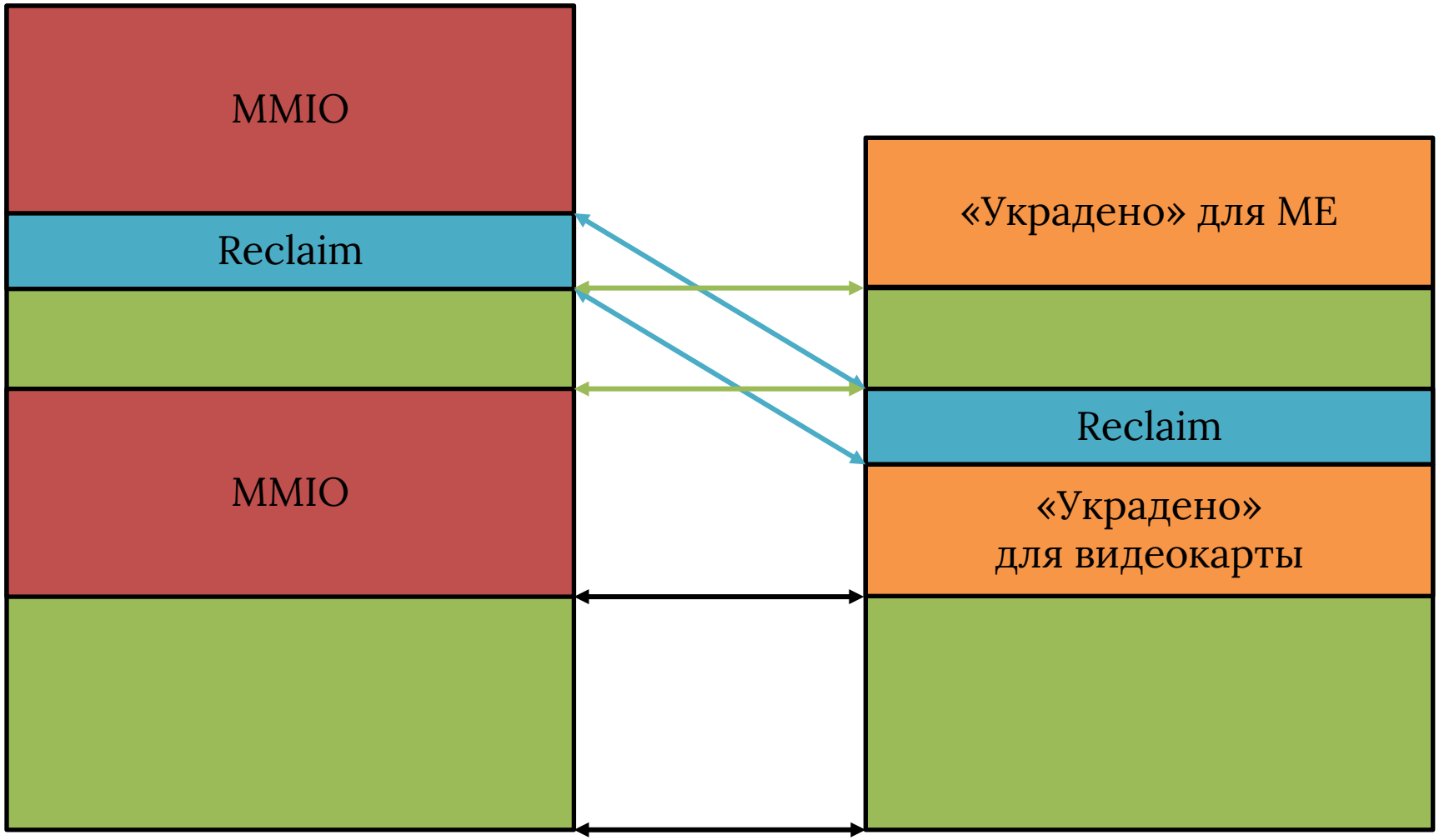


Верхний регион памяти

- Картина после 64 разрядов.
- Основная память:
  - всегда DRAM;
  - появляется возможность компенсировать часть потерянной памяти до 4 Гбайт.
- MMIO:
  - диапазон для работы с PCI-устройствами.



Физическая память и DRAM



Механизмы настройки

- Основные механизмы настройки конфигурации памяти:
  - регистры TOP_MEM (TOLUD) и TOP_MEM2 (TOUUD);
  - регистры IORR:
    - два диапазона физических адресов;
    - признаки — куда направить чтение, куда запись;
  - регистры MTRR:
    - *фиксированные* позволяют задавать направление чтения и записи аналогично IORR:
      - 0_0000…7_FFFF гранулярность по 1_0000;
      - 8_0000…B_FFFF гранулярность по 0_4000;
      - C_0000…F_FFFF гранулярность по 0_1000;
    - *переменные* не позволяют управлять типом памяти.

Корректность доступа

- Система в общем случае будет работать некорректно, если:
  - разные процессорные ядра имеют разные представления о назначении диапазона памяти;
  - осуществляется кеширование диапазонов памяти MMIO.
- Управление кешем:
  - грубое — можно не включать кеширование;
  - тонкое:
    - на уровне MTRR — и фиксированные, и переменные MTRR позволяют указать тип кеширования для диапазона;
    - на уровне таблиц страниц — механизм PAT позволяет указать тип кеширования для страницы.

Типичная карта MMIO

Назначение	Диапазон адресов
PCI Express Configuration Space	`E000_0000...F000_0000`
IOAPIC	`FEC0_0000...FEC0_0FFF`
HPET	`FED0_0000...FED0_03FF`
TPM	`FED4_0000...FED4_4FFF`
VT-d	`FED9_0000...FED9_3FFF`
Локальный APIC	`FEE0_0000...FEEF_FFFF`
Флеш-память BIOS / UEFI	`FF00_0000...FFFF_FFFF`
Настраиваемые: PCI BARs	

Если эта карта «типичная», то как операционная система понимает, какие устройства существуют и по каким адресам?

- Для шины PCI есть возможность перебрать все устройства на шине и настроить их BAR.
- Для остальных устройств такой возможности нет.

Раздел 5

BIOS / UEFI

Встроенное ПО: BIOS

- Встроенное ПО на x86 (и некоторых вариантах ARM и RISC-V) используется для первоначальной настройки аппаратуры и загрузки операционной системы.
- *BIOS (Basic Input-Output System)* — более старый вариант встроенного ПО:
  - имеет меньший размер и более простую организацию;
  - хуже специфицирован, чем UEFI;
  - разные поставщики имеют разные расширения;
  - предоставляет сервисы операционной системе через механизм программных прерываний;
  - большая часть сервисов может вызываться только в реальном режиме.

Встроенное ПО: UEFI

- *UEFI* (*Unified Extensible Firmware Interface*) — попытка решить проблемы BIOS:
  - реализация должна соответствовать набору спецификаций и тестов;
  - подходит не только для x86;
  - реальный режим не используется, в зависимости от типа процессора — либо 32-, либо 64-разрядный защищённый режим;
  - по сути представляет собой небольшую операционную систему.
- Обратная совместимость — *CSM* (*Compatibility Support Module*).

Задачи встроенного ПО

- Начальная настройка аппаратуры:
  - конфигурирование памяти:
    - инициализация DRAM;
    - установка TOP_MEM, TOP_MEM2, IORR, MTRR, PAT;
  - настройка основных устройств системы:
    - контроллеры прерываний;
    - контроллеры памяти;
    - PCI-контроллеры;
    - другие;
  - инициализация и проверка прикладных процессорных ядер.
- Встроенное ПО может это сделать, т.к. его конкретная версия делается под конкретную материнскую плату.

Таблицы ACPI

- Для того, чтобы передать операционной системе знания о составе платформы, адресах отображённых на память устройств и т.п., встроенное ПО формирует в памяти набор таблиц ACPI (*Advanced Configuration and Power Interface*).
- Некоторые таблицы:
  - MADT — перечень локальных APIC (т.е. ядер), IOAPIC, информация о прерываниях для встроенных устройств;
  - HPET — конфигурация таймера высокого разрешения;
  - DSDT — байт-код для получения списка встроенных устройств, их ресурсов и управления их питанием;
  - FADT — таблица поддержки глобального управления питанием, в т.ч. сон и перевод системы на дежурное питание (выключение).

Задачи встроенного ПО

- Загрузка операционной системы:
  - выбор загрузочного устройства;
  - выбор файла или набора секторов на загрузочном устройстве, содержащих загрузчик ОС;
  - копирование образа загрузчика в память;
  - передача управления на первую команду загрузчика;
  - предоставление сервисов для загрузчика и ОС.
- Управление аппаратурой (энергопотребление, эмуляция) во время работы ОС:
  - требуется такой режим работы процессора, который позволил бы коду встроенного ПО продолжать работать «параллельно» с ОС.

Загрузка ОС: BIOS

- В настройках BIOS пользователь может выбрать порядок просмотра устройств при загрузке.
- Поддерживается, как правило, загрузка с дискеты, CD-ROM, жёсткого диска.
- Загрузка с USB-накопителя поддерживается в режиме эмуляции дискеты, CD-ROM или жёсткого диска.
- Загрузка с CD-ROM — отдельный стандарт El Torito.
- Загрузка с дискеты или жёсткого диска:
  - BIOS загружает в память по адресу 0000_7C00 первый сектор диска — 512 байт;
  - BIOS проверяет сигнатуру — байты 55, AA в конце сектора;
  - если сигнатура корректна, то подготавливается окружение и управление передаётся на 0000_7C00.

Загрузка ОС: UEFI

- В настройках UEFI пользователь может выбрать порядок просмотра устройств при загрузке.
- UEFI умеет загружать файлы с FAT-разделов. Файл для загрузки выбирается по определённому пути.
- Некоторые поставщики (Apple) имеют свой способ выбора файла и свой формат файловой системы.
- Загружаемый файл представляет собой PE (Portable Executable) — по сути это Windows DLL, только без возможности подключать внешние библиотеки.
- В качестве параметра функции *main* передаётся указатель на *системную таблицу*, через которую можно вызывать сервисы UEFI — *boot services* и *runtime services*.

Старт встроенного ПО

- Выше мы видели, что:
  - после RESET загрузочное ядро начнёт выбирать команды в реальном режиме начиная с адреса FFFF_FFF0;
  - это самый «потолок» 32-разрядного адресного пространства, в соответствии с первоначальным значением TOP_MEM (0400_0000) это MMIO-диапазон;
  - в типичной карте MMIO этот диапазон соответствует флеш-памяти;
  - флеш-память вне режима перепрошивки — по сути ПЗУ.
- Таким образом, код начинает выбираться из памяти только для чтения, отображённой с флеш-чипа.
- Осложнение: DRAM ещё не сконфигурирована. Коду встроенного ПО некуда писать.

Cache-as-RAM

- Решение проблемы: включить и определённым образом настроить кеш.
- При записи в диапазоны памяти, которые в будущем станут отображаться на DRAM, будет происходить внесение данных в кеш. При чтении — попадание в кеш. Тем самым временно можно обойтись без DRAM, но только в небольшой области памяти.
- Поэтому типичная последовательность действий следующая:
  - сконфигурировать режим cache-as-RAM;
  - обнаружить и настроить планки DRAM (протокол SPD);
  - выключить режим cache-as-RAM;
  - скопировать образ встроенного ПО в RAM;
  - переключить диапазон с флеш-памяти на RAM (или нет).

Раздел 6

РЕЖИМ СИСТЕМНОГО МЕНЕДЖМЕНТА

SMM

- *Режим системного менеджмента* (SMM; *System Management Mode*) — особый режим работы процессорного ядра.
- Режим похож на реальный, но имеет отличия — лимиты сегментов установлены в 4 Гбайт вместо 64 Кбайт. Похожий режим можно сконфигурировать явно, его принято называть *нереальным* (*unreal*).
- Режим системного менеджмента работает «параллельно» с нормальным функционированием машины. Параллельность здесь понимается так же, как в многозадачной ОС на одном процессорном ядре.
- Режим системного менеджмента впервые был реализован в процессоре 80386SL.
- Основные задачи — эмуляция устаревших устройств, поддержка управления энергопотреблением.

Переключение в/из SMM

- Переход в режим системного менеджмента управляется внешними событиями:
  - прерыванием SMI — сконфигурированным программно или пришедшим со стороны чипсета и других устройств;
  - доступом к определённым портам ввода-вывода;
  - событиями таймера, температурных сенсоров и т.п.;
  - другими событиями.
- При переходе в режим SMM сохраняется полное состояние процессорного ядра. Процессорное ядро переводится в предопределённое состояние (похоже на RESET и INIT).
- Для выхода из режима SMM используется особая команда RSM, которая восстанавливает состояние процессорного ядра перед переключением.

Состояние при входе в SMM

Аспект	Начальное состояние
CR0	сброшены биты PE, EM, TS, PG
CR4	0000_0000_0000_0000
EFER	0000_0000_0000_0000
RFLAGS	0000_0000_0000_0002
RIP	0000_0000_0000_8000
CS	Селектор: SMBASE >> 4 База/лимит: SMBASE + FFFF_FFFF Атрибуты: 16-битный код, DPL 0
DS, ES, FS, GS, SS	Селектор: 0000 База/лимит: 0000_0000_0000_0000 + FFFF_FFFF Атрибуты: 16-битные данные, DPL 0

Области памяти SMM

- Специальный скрытый регистр процессора SMBASE отвечает за базовый адрес области обработчика SMM.
- При переключении в SMM управление передаётся по определённому смещению относительно SMBASE.
- Состояние процессорного ядра при переходе в SMM сохраняется в конце области, на которую указывает SMBASE.
- Единственный способ изменить значение SMBASE — изнутри обработчика режима SMM, т.к. значение SMBASE входит в область сохранённого состояния.
- Всё это сделано для того, чтобы никто (в т.ч. операционная система) не мог поменять поведение обработчика SMM после того, как его сформирует встроенное ПО.
- Однако что мешает просто перезаписать область SMBASE своим кодом?

ASEG и TSEG

- Существует два диапазона физической памяти, которые предназначены специально для использования обработчиком SMM.
- ASEG — фиксированный диапазон A_0000...B_FFFF:
  - для не-SMM этот диапазон соответствует видеопамяти;
  - когда процессорное ядро работает в режиме SMM, контроллер памяти воспринимает обращения в ASEG как обращения к DRAM;
  - таким образом, ASEG «прячется» за видеопамятью.
- TSEG — перемещаемый диапазон:
  - может быть размещён где угодно и иметь переменный размер;
  - обращение к нему возможно только из SMM.

SMM и ACPI

- SMM играет важную роль в реализации ACPI на современных платформах.
- Часть особенно сложных взаимодействий с платформой, которые нельзя описать в таблице DSDT в виде байт-кода, вместо этого сводятся к отправке команд в определённый порт ввода-вывода.
- Этот порт ввода-вывода перехватывается таким образом, чтобы запись в него приводила к переходу в SMM.
- Обработчик SMM выясняет, что от него требовалось, выполняет команду и возвращает управление. Для операционной системы это абсолютно прозрачно, она «не ощущает» никаких команд после записи в порт.

SMM и безопасность

- Таким образом, в архитектуре x86 сделано всё для того, чтобы во время выполнения нельзя было никаким образом изменить код SMM или даже его прочитать.
- Это значит, что аудит этого кода во время выполнения невозможен.
- Код SMM имеет доступ ко всему пространству физической памяти, а значит и ко всем устройствам.
- Насколько печально будет, если в обработчике SMM найдётся уязвимость, позволяющая внедрить туда вредоносный код?
- Бывало ли такое? Да, много раз. Из свежего (сентябрь 2017 г.) — CVE-2017-3753 — внедрение кода в SMM на Lenovo (AMI UEFI).

Внедрение кода в SMM

- Сам код обработчика SMM имеет сравнительно небольшой размер.
- Поверхность атаки маленькая — при правильной настройке памяти до кода сложно добраться, обработчик сам выбирает события, по которым он срабатывает.
- Во многих случаях проще скомпрометировать встроенное ПО ещё до того, как оно окончательно настроит обработчик SMM.
- UEFI имеет несколько последовательно загружаемых компонентов, каждый из которых вынужден доверять предыдущему.
- Необходима организация цепочки *доверенной загрузки.*
- Сначала стоит рассказать про *модуль TPM.*

Раздел 7

TRUSTED PLATFORM MODULE

TPM

- *TPM (Trusted Platform Module)* — специализированный криптографический сопроцессор, пассивное устройство.
- Две несовместимые версии спецификации — 1.2 и 2.0:
  - концептуально одинаковы;
  - отличаются в первую очередь набором поддерживаемых криптографических алгоритмов, а также структурой хранимых данных;
  - версия 2.0 существенно сложнее, но охватывает больше платформ для внедрения;
  - далее говорим про версию 1.2.
- Реализация:
  - «дискретный» чип (обычно на шине LPC);
  - реализация в виде встроенного ПО более низкого уровня, чем BIOS/UEFI — подробности последуют.

Компоненты TPM

- Безопасный ввод-вывод.
- Криптографический сопроцессор:
  - генератор случайных чисел;
  - генератор ключей схемы RSA;
  - генератор криптографических хешей SHA-1;
  - подсистема шифрования, дешифрования, цифровой подписи.
- Энергонезависимая память:
  - корневые ключи (endorsement и storage).
- Оперативная память:
  - PCR;
  - производные ключи.

Применение TPM

- Целостность платформы: организация «корня доверия» (root of trust):
  - подробности в следующем разделе.
- Полнодисковое шифрование:
  - хранение ключей шифрования.
- Парольная защита:
  - проверка паролей;
  - встроенная в аппаратуру защита от словарных атак.
- Защита «цифровых прав» (DRM; Digital Rights Management), anti-cheat, проверка лицензий.

PCR

- PCR (*Platform Configuration Register*) — регистр модуля TPM с особым поведением.
- TPM 1.2 требует наличия как минимум 16 PCR в модуле.
- Существует три основных операции при работе с PCR:
  - *сброс* — значение PCR сбрасывается на начальное (обычно нулевое);
  - *чтение* — значение PCR может быть прочитано с целью сравнения с известным «правильным» значением;
  - *расширение* — $PCR'_n = SHA\text{-}1(PCR_n \,||\, data)$.
- Условия, при которых возможен сброс, достаточно сложные.
- Главное: прямой записи в PCR нет.
- Таким образом, итоговое значение PCR учитывает все предыдущие значения.

Binding & Sealing

- Две более сложные операции, которые умеет выполнять TPM.
- *Binding (привязывание)* — шифрование ключей при помощи RSA-ключа, производного от корневого ключа storage. Такие данные может расшифровать только данный модуль TPM.
- *Sealing (запечатывание)* — то же самое, что binding, но с дополнительным требованием:
  - значения определённого подмножества PCR при запечатывании и распечатывании должны совпасть;
  - таким образом достигается проверка одинакового состояния системы при запечатывании и распечатывании.

Раздел 8

АППАРАТНАЯ ВИРТУАЛИЗАЦИЯ

Виртуализация

- Основная задача — обеспечить одновременную работу нескольких экземпляров ОС (одинаковой или разных) на одной физической машине.
- Если код ОС написан специальным образом, так что ОС «знает», что выполняется в определённом виртуальном окружении, то это — *паравиртуализация*.
- В каждый момент времени на отдельно взятом ядре выполняется:
  - либо код одной из *гостевых систем*;
  - либо код *монитора виртуальных машин (гипервизора)*.
- Гипервизор решает задачи, похожие на задачи ядра ОС:
  - распределение ресурсов (процессорное время, память);
  - виртуализация устройств.

Требования Попека-Голдберга

- Сформулированы Дж. Попеком и Р. Голдбергом в 1974 г.
- Свойства виртуального окружения:
  - *эквивалентность* — неразличимость выполнения программы в виртуальной машине и на реальной аппаратуре;
  - *управление* ресурсами — монитор должен обладать полным контролем над виртуализируемыми ресурсами;
  - *эффективность* — статистически подавляющая доля выполняемых (реальным) процессором машинных команд должна выполняться без вмешательства монитора.
- Классификация машинных команд:
  - привилегированные;
  - чувствительные по управлению;
  - чувствительные по поведению.

Теоремы Попека-Голдберга

Теорема 1. Эффективный монитор может быть построен только если множество чувствительных команд является подмножеством привилегированных.

- Виртуализация *trap-and-emulate*, она же *классическая*.
- Не годится для x86 без аппаратной поддержки:
  - SGDT, SLDT, SIDT, SMSW, PUSHF, POPF;
  - LAR, LSL, VERR, VERW, POP, PUSH, CALLF, JMPF, INT, RETF, STR, MOV SR.
- Как работала виртуализация до появления расширений?

Аппаратная виртуализация: x86

- Intel VT-x: 2005 г., AMD SVM: 2006 г. — первые реализации в кремнии.
- Реализации не совместимы, хотя и предлагают примерно равные возможности.
- Важный момент — добавление *nested paging*:
  - возможность иметь дополнительный уровень страничной трансляции адресов из гостевых в системные;
  - до добавления аппаратной поддержки использовались теневые таблицы, это намного медленнее.
- Дальнейшее обсуждение — по реализации AMD.

Общая схема работы SVM

- Для каждой гостевой системы поддерживается область памяти размером 4 Кбайт, в которой содержится состояние виртуальной машины — VMCB (*virtual machine control block*):
  - различные управляющие флаги;
  - список событий гостя, которые будут перехватываться;
  - состояние гостевого процессора.
- Передача управления от монитора гостю — команда VMRUN.
- Передача управления от гостя монитору — по наступлению состояния #VMEXIT.
- TLB — тегирование с учётом ASID (*address space identifier*).
- Флаг GIF (*global interrupt flag*): отключить прерывания вообще, даже INIT, NMI, SMI. Команды CLGI, STGI.

Вопросы безопасности

- HVM и SVM:
  - существует возможность создать отдельную виртуальную машину, в которой будет выполняться код обработчика SMM;
  - теоретически это позволяет защититься от недоверенного кода в обработчике и в какой-то степени от атак на него.
- «Синяя таблетка»:
  - рекурсивная виртуализация;
  - гипервизор может выполняться под управлением другого гипервизора — это хорошо или плохо?
- Доверие гипервизору:
  - гипервизор имеет доступ ко всей физической памяти;
  - может быть, в нём есть закладка или уязвимость, и данные, которые обрабатываются в госте, могут утечь.

Intel SGX / AMD SEV

- Механизмы, нужный для защиты критического кода и данных от ОС и гипервизора.
- Идея:
  - ОС, гипервизор, обработчик SMM вынужденно имеют доступ ко всей памяти, изменить это нельзя;
  - необходимо сделать так, чтобы даже в таком случае данные были защищены;
  - решение — аппаратное шифрование.
- Реализация (AMD SEV):
  - набор AES-ключей (один для гипервизора, несколько для гостей);
  - поддержка шифрования в контроллере памяти;
  - ключевой момент — гипервизор «не видит» ключ гостя на всём протяжении существования гостя;
  - подходит ли для этого TPM?

Раздел 9

ДОВЕРЕННАЯ ЗАГРУЗКА

Доверенная загрузка

- Цель — убедиться, что каждый следующий загружаемый и выполняемый компонент не скомпрометирован.
- Решение — организация цепочки доверия (chain of trust).
- *Статический корень цепочки доверия* — самый первый блок кода, который выполняется системой, ему приходится доверять безусловно.
- Выполняющийся компонент при загрузке следующего проверяет его цифровую подпись или криптографический хеш. Если проверка не проходит — загрузка не продолжается.
- Если требуется цифровая подпись каждого следующего компонента (включая загрузчик ОС), то это режим *Verified Boot*.

Режим Measured Boot

- Идея — не требовать наличие цифровой подписи у всех загружаемых компонентов (абсолютное свойство), а вместо этого сравнивать текущий процесс загрузки с каким-то ранее произошедшим и считающимся доверенным (относительное свойство).
- Реализация — через регистры PCR и запечатанные приватные ключи. Пример: невозможность расшифровать жёсткий диск, если загрузка «странная».
- Возможна также *удалённая аттестация* — подписанный уникальным для TPM ключом журнал загрузки (в т.ч. значения PCR) отправляется на удалённый сервер для проверки. Только после проверки сервер выполняет чувствительную операцию.

Динамический корень доверия

- Некоторые процессоры Intel и AMD поддерживают команды SENTER и SKINIT, соответственно.
- Эти команды используются для создания *динамического корня доверия*, т.е. для попытки перевести систему из произвольного состояния в доверенное.
- Примеры реализации:
  - tBoot;
  - OSLO.
- Далее рассматривается команда SKINIT как более простая.

SKINIT

1. При выполнении команды SKINIT в EAX — физический адрес загрузчика (SL; Secure Loader).
2. Ядро сбрасывает состояние как будто получило прерывание INIT, после чего переходит в плоский защищённый режим без страничной трансляции, загружает сегментные регистры CS и SS (в том числе теневые части).
3. Устанавливается защита региона SL (64 Кбайт) от DMA-доступа.
4. Содержимое образа SL отправляется в TPM.
5. TPM осуществляет хеширование и проверку подписи SL.
6. Очищается GIF.
7. Управление передаётся на SL.

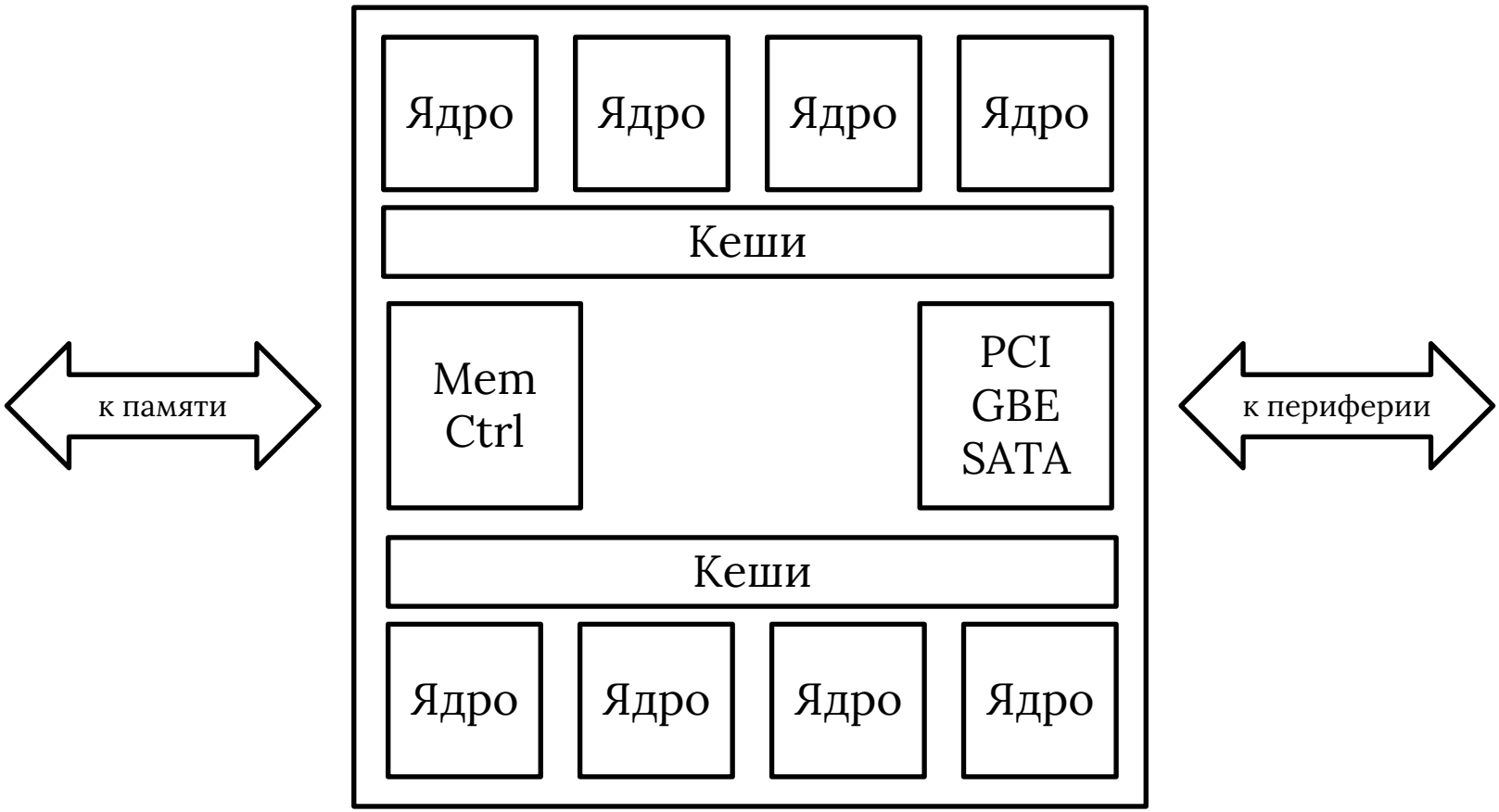
Вопросы безопасности

- Динамический корень доверия с полной степенью уверенности организовать нельзя, потому что есть аппаратная поддержка виртуализации.
- Остаётся только статический корень доверия.
- Проблема «курицы и яйца»:
  - кто проверит, что самый первый блок кода, часть ПЗУ встроенного ПО — благонадёжный?

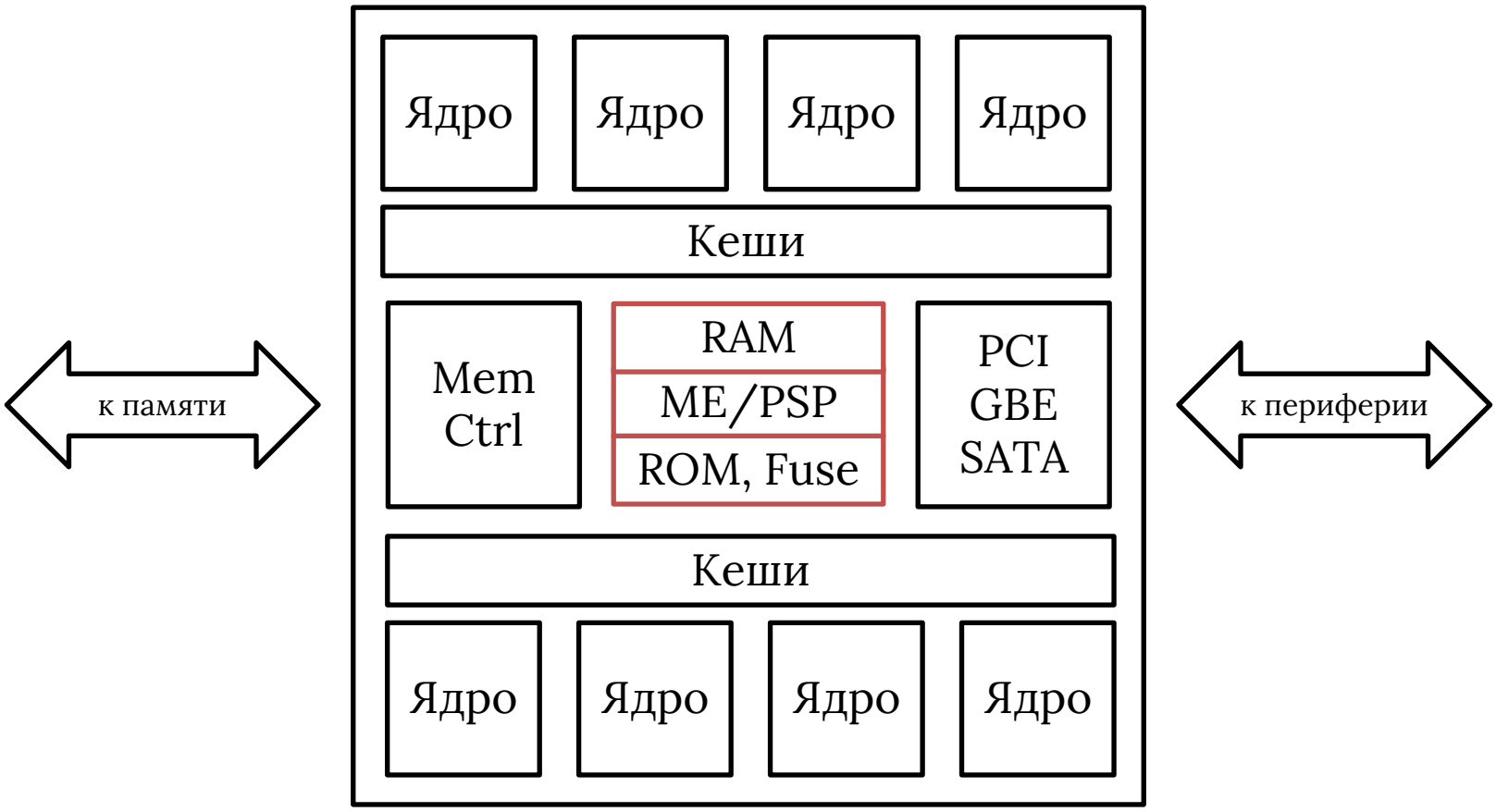
Раздел 10

INTEL MANAGEMENT ENGINE AMD SECURE PROCESSOR

Особое процессорное ядро



Особое процессорное ядро



Некоторые задачи ME/PSP

- Работа с ключами при шифровании памяти.
- Выполнение самого раннего этапа доверенной загрузки:
  - в fuse-памяти ME/PSP прошиты ключи, позволяющие проверить подпись начальной части встроенного ПО;
  - это позволяет организовать статическую цепочку доверенной загрузки;
  - ключи прошиваются однократно и не могут быть изменены.
- Firmware TPM:
  - реализация TPM не в виде дискретного чипа, а как одну из программ, выполняющихся в ME/PSP.
- Удалённое управление серверами:
  - сбор информации о состоянии машин;
  - запуск, перезагрузка и т.п.

Вопросы безопасности

- Некоторые вводные:
  - исходный код прошивки ME/PSP недоступен;
  - про ME известно, что это x86-ядро, в рамках которого работает упрощённая версия ОС Minix;
  - про PSP известно, что это ARM-ядро;
  - бинарный код прочитать нельзя, он зашифрован.
- Таким образом, аудит выполняющегося кода невозможен.
- Некоторые причины для беспокойства:
  - ME/PSP имеют прямой доступ ко всей памяти;
  - ME/PSP имеют прямой доступ ко всем устройствам;
  - ME/PSP работают от дежурного питания (в т.ч. RAM).
- Sleep tight, ignorance is bliss.

Приложение

ЛИТЕРАТУРА

Аппаратура

1. AMD64 Architecture Programmer's Manual. Volume 2: System Programming.
2. AMD BIOS and Kernel Developer's Guide.
3. Intel 64 and IA-32 Architectures Software Developer's Manual. Volume 3: System Programming Guide.
4. 8259A Programmable Interrupt Controller (8259A/8259A-2).
5. 82093AA I/O Advanced Programmable Interrupt Controller (IOAPIC).
6. HyperTransport I/O Link Specification.
7. PCI Local Bus Specification.
8. PCI Express Base Specification.
9. Intel Software Guard Extensions Programming Reference.
10. AMD Secure Encrypted Virtualization API.

Встроенное ПО

1. BIOS Boot Specification.
2. “El Torito” Bootable CD-ROM Format Specification.
3. Unified Extensible Firmware Interface Specification.
4. Advanced Configuration and Power Interface Specification.

Беллетристика

1. A Tour beyond BIOS Memory Map Design in UEFI BIOS.
2. OSDev Wiki.