Низкоуровневое программирование

Лекция 8

Прерывания и исключения

Режимы работы процессора

Процесс загрузки

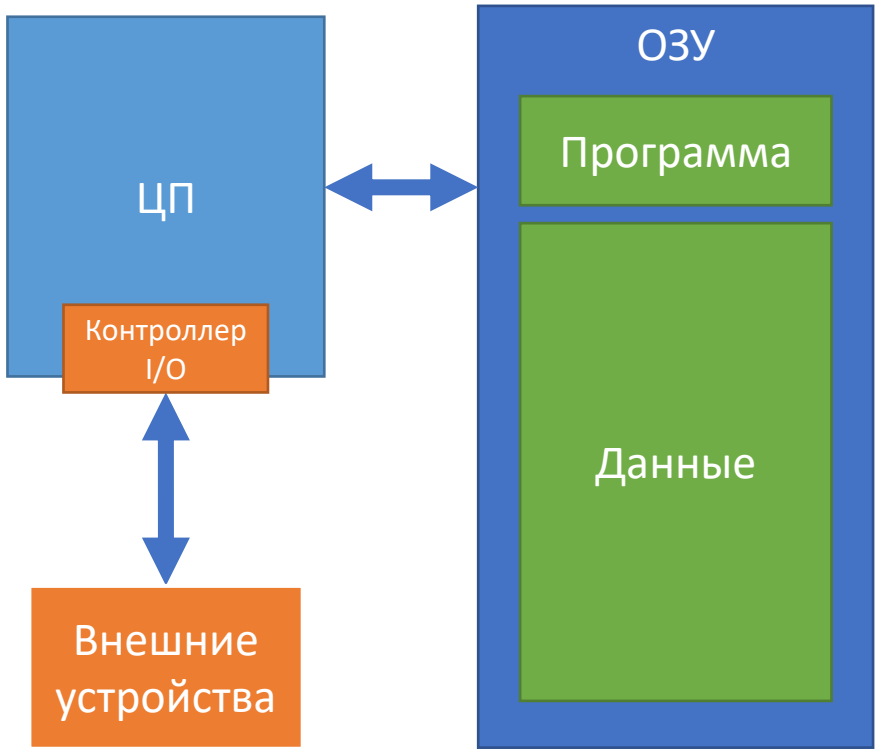
Внешние устройства

Для функционирования компьютера формально необходимы только ЦП, ОЗУ и связующее их звено. Тем не менее, очевидно, что для исполнения простейшей программы необходимо загрузить ее в ОЗУ из третьего устройства. Кроме того, желательно также вывести результат работы программы из ОЗУ. С точки зрения ЦП, всем этим заняты **внешние устройства**.

К числу внешних устройств относятся видеокарты, HDD и SSD, дисководы, аудиокарты и пр.

Для того, чтобы общаться с данными устройствами, процессору необходим специальный **механизм ввода-вывода**.

Замечание: описанные далее механизмы недоступны в обычных пользовательских программах



Порты ввода-вывода

Общение с внешними устройствами осуществляется через **порты ввода-вывода** (I/O-порты).

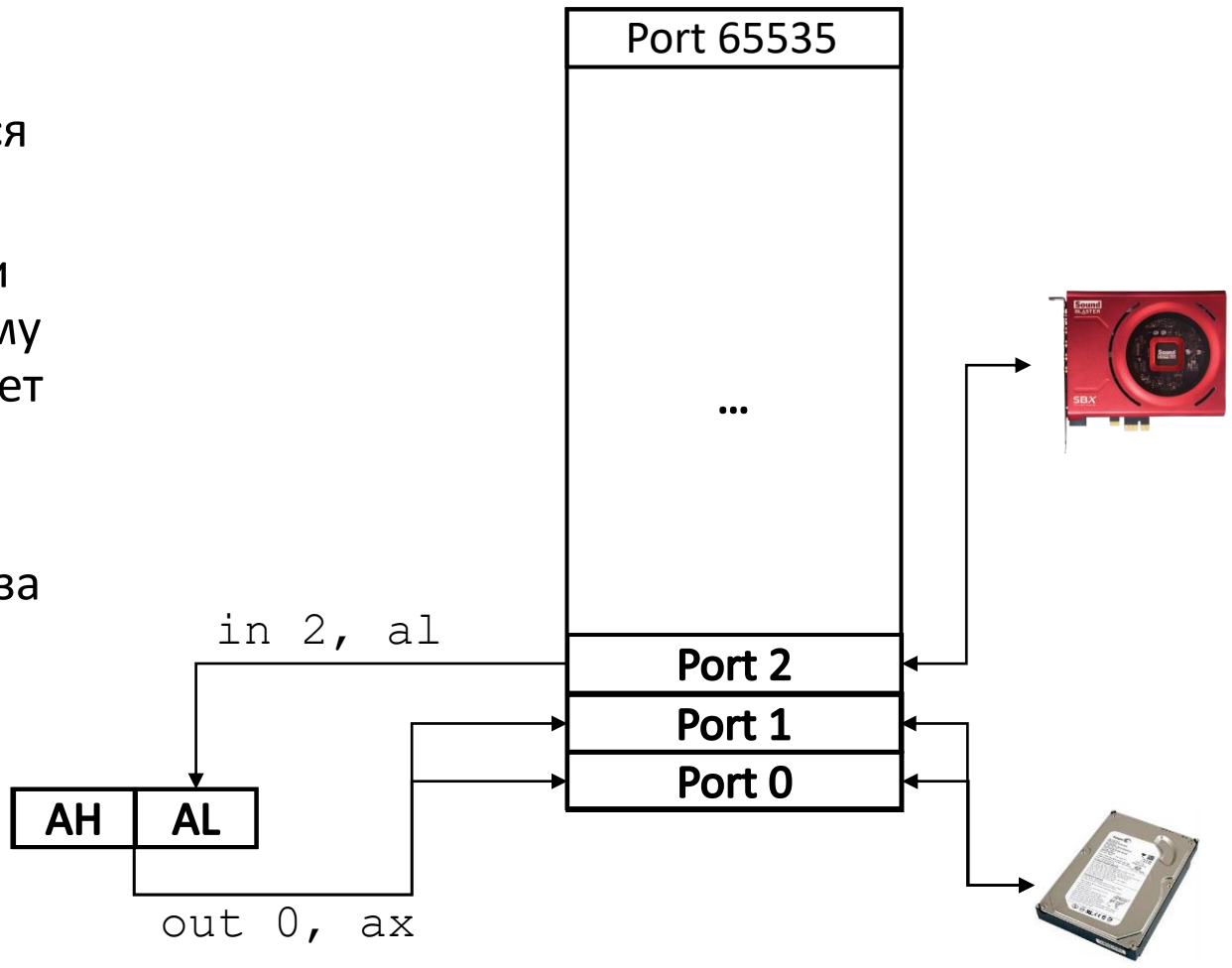
Пространство портов эквивалентно области памяти размером 64 Кбайт. Один байт соответствует одному порту => в системе максимум 65536 устройств может быть подключено к портам.

Поскольку пространство портов эквивалентно памяти, допустимо читать/записывать 1/2/4 байта за 1 раз (затрагивая 1/2/4 порта).

Чтение I/O-порта осуществляется инструкцией `in`. Результат считывается в AL/AX/EAX.

Запись в порт производится инструкцией `out`. Данные берутся из AL/AX/EAX.

Номера портов являются константами либо читаются из DX.

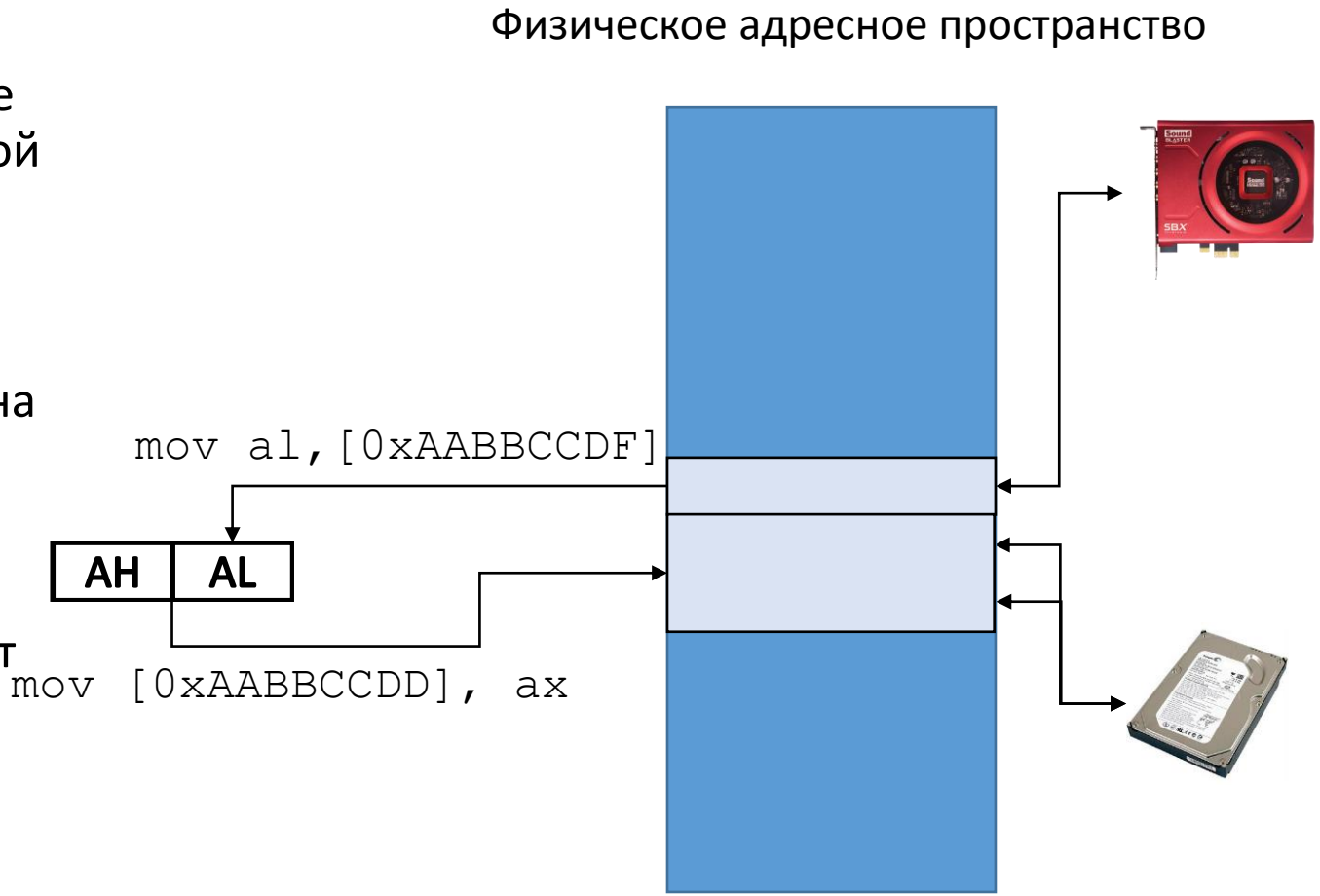


Отображение в память

Механизм отображения в память позволяет представить устройство в виде некоторой области памяти. Чтение из этой области памяти эквивалентно чтению с устройства (аналогично с записью).

Отображение устройств в память настраивается операционной системой на этапе загрузки.

Устройства отображаются в физическое адресное пространство (отличия физических и виртуальных адресов будут рассмотрены далее).



Прерывания

Прерывание (interrupt) – сигнал о наступлении события, требующего скорейшей обработки.

Прерывания могут инициироваться:

- внешними устройствами – посредством сигнальных линий между устройством и процессором;
- программой – инструкцией `int`;
- самим процессором – при ошибках.

С каждым типом прерывания ассоциированы **номер** и **обработчик прерывания**

В момент получения запроса на прерывание процессор останавливает выполнение текущей программы и переключается на выполнение обработчика прерывания.

После выполнения обработчика прерывания ЦП переключается обратно, программа продолжает выполнение.

С точки зрения программы прерывания не происходило вовсе.

Прерывания от внешних устройств

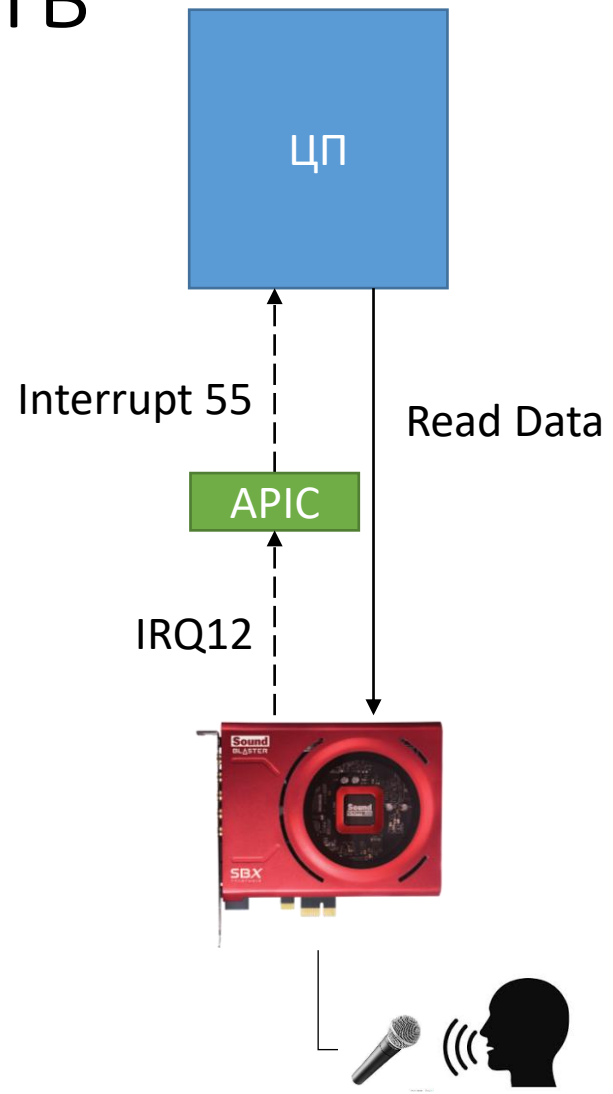
За получение запросов на прерывание системе отвечает **APIC** (advanced programmable interrupt controller). APIC конфигурируется при загрузке ПК.

При наступлении определенного события устройство посылает запрос на прерывание (**IRQ**, interrupt request).

С точки зрения APIC, каждый запрос на прерывание имеет порядковый номер, указывающий на устройство-инициатор (например, IRQ12).

При получении запроса на прерывание APIC выбирает одно из ядер ЦП и посылает ему сигнал наличия прерывания и номер прерывания. Соответствие между номером IRQ и номером прерывания задается при загрузке ОС. Одному номеру прерывания может соответствовать несколько IRQ (например, если в системе есть несколько однотипных устройств).

После получения сигнала прерывания ядро считывает номер прерывания и переключается на обработчик.



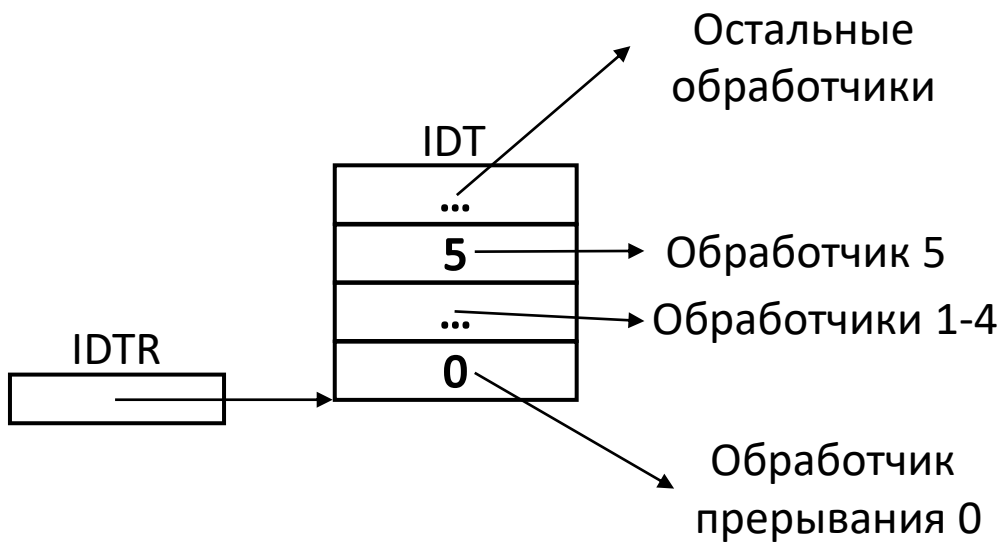
Регистр IDTR

Адреса обработчиков прерываний находятся в **таблице дескрипторов прерываний*** (Interrupt Descriptor Table, IDT).

Адрес IDT находится в специальном регистре **IDTR** (IDT Register)

Номер прерывания соответствует индексу адреса обработчика в данной таблице

Заполнение таблицы и загрузка ее адреса в IDTR инструкцией `lidt` производятся операционной системой.



*изначально – таблица векторов прерываний (IVT)

Флаг IF.

Обработчик прерывания является критичным участком кода, который производит обновление состояния внешнего устройства.

Если в момент обработки прерывания возникнет другое прерывание, на обработку которого переключится ЦП, исходное устройство может оказаться в нестабильном состоянии. Как следствие, необходим способ отложить прием прерываний до завершения обработчика.

Флаг `RFLAGS.IF` (бит 10) отвечает за блокирование прерываний (0 – заблокированы, 1 - разблокированы).

Если `RFLAGS.IF=0`, то прием прерывания откладывается до снятия блокировки.

Обычные программы не могут менять флаг `RFLAGS.IF.` , если это не разрешено ОС.

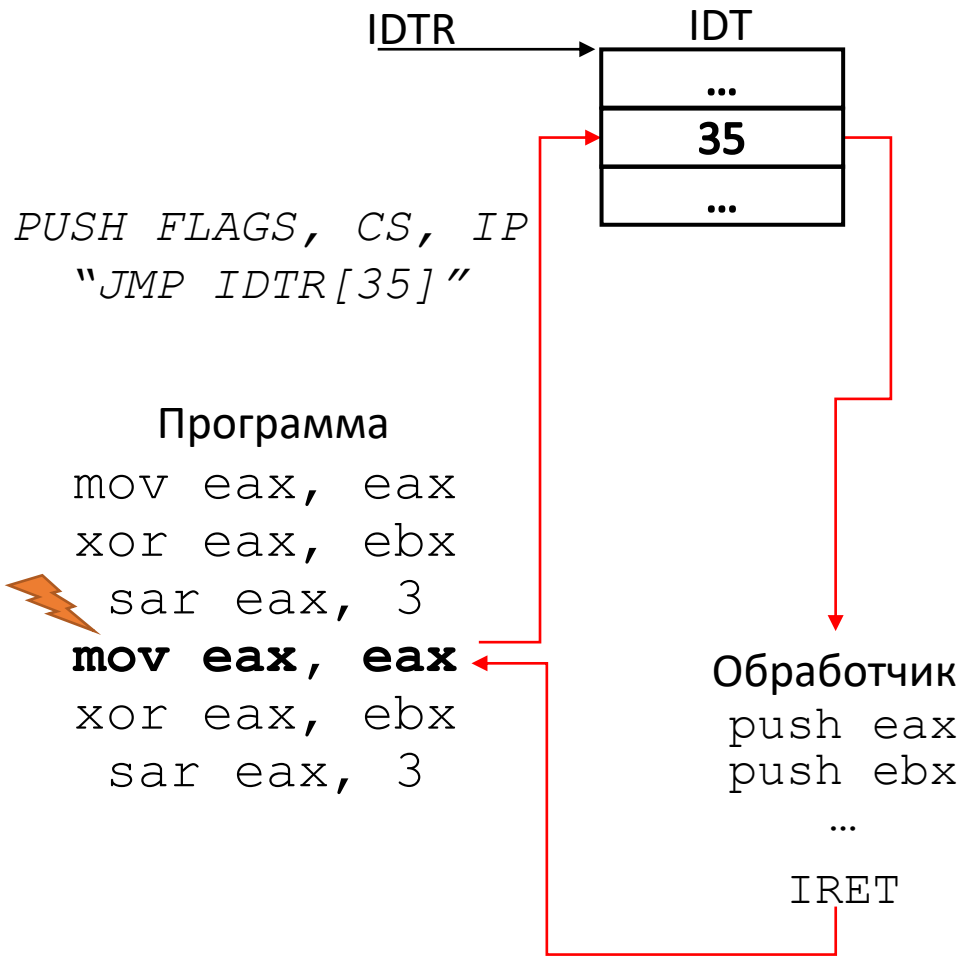
Флаг IF не влияет на прием т.н. немаскируемых прерываний (NMI), которые сигнализируют о серьезных ошибках в аппаратном обеспечении.

Обработка прерывания

В момент получения прерывания процессор:

1. сохраняет на стеке* текущие значения FLAGS,CS, IP (и код ошибки - для исключений).
2. автоматически устанавливает флаг `RFLAGS.IF=0`.
3. Прыгает на адрес [IDTR+K*N], где N – номер прерывания, K – размер элемента таблицы (4 в реальном режиме, 8 – в защищенном, 16-в длинном).
4. Выполняет обработчик.
5. Возвращается обратно в программу инструкцией `IRET`. Сохраненные значения из шага 2 восстанавливаются автоматически.

** в длинном режиме для обработки прерываний используется отдельный стек (см.далее)*

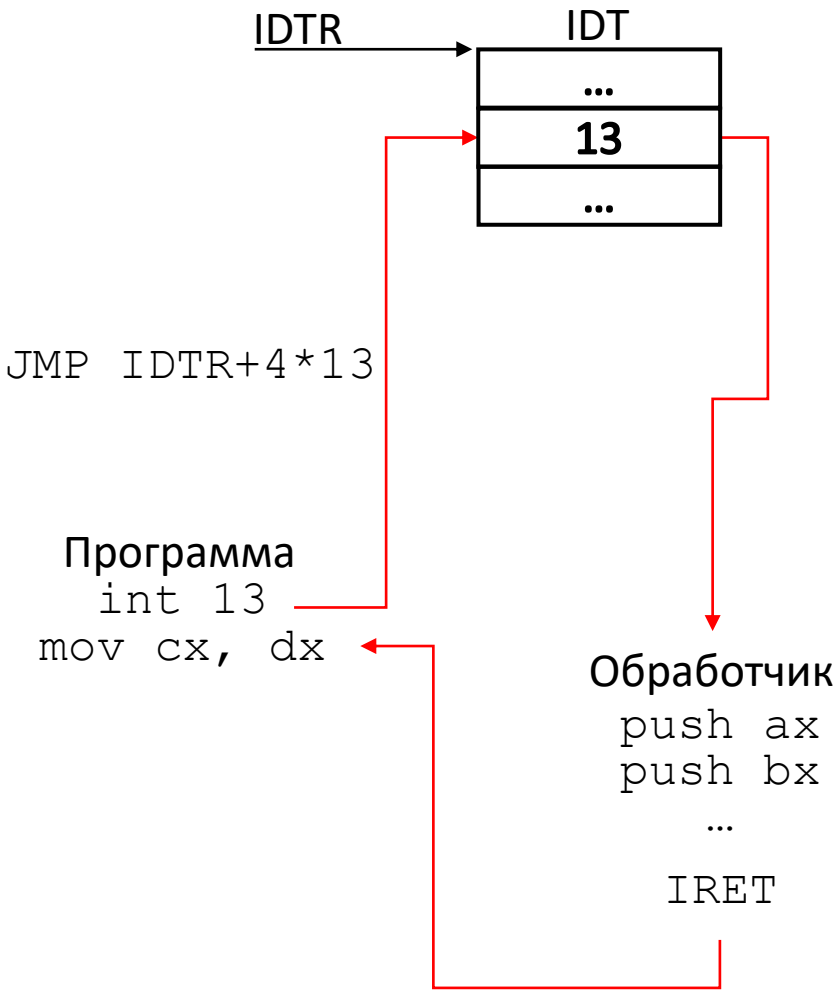


Программные прерывания

Программа может самостоятельно вызвать прерывание инструкцией `int`, единственным параметром которой является число – номер прерывания.

Ранее программные прерывания служили для обращения к ОС. Например, в MS-DOS вызов `int 16` позволял прочитать введенный с клавиатуры символ, вызов `int 13` отвечал за чтение с диска и т.д. Параметры в этом случае передавались в регистрах AX/DX.

В UNIX-системах механизм прерываний используется для осуществления системных вызовов из 32-битных программ и по сей день.



Аппаратные исключения

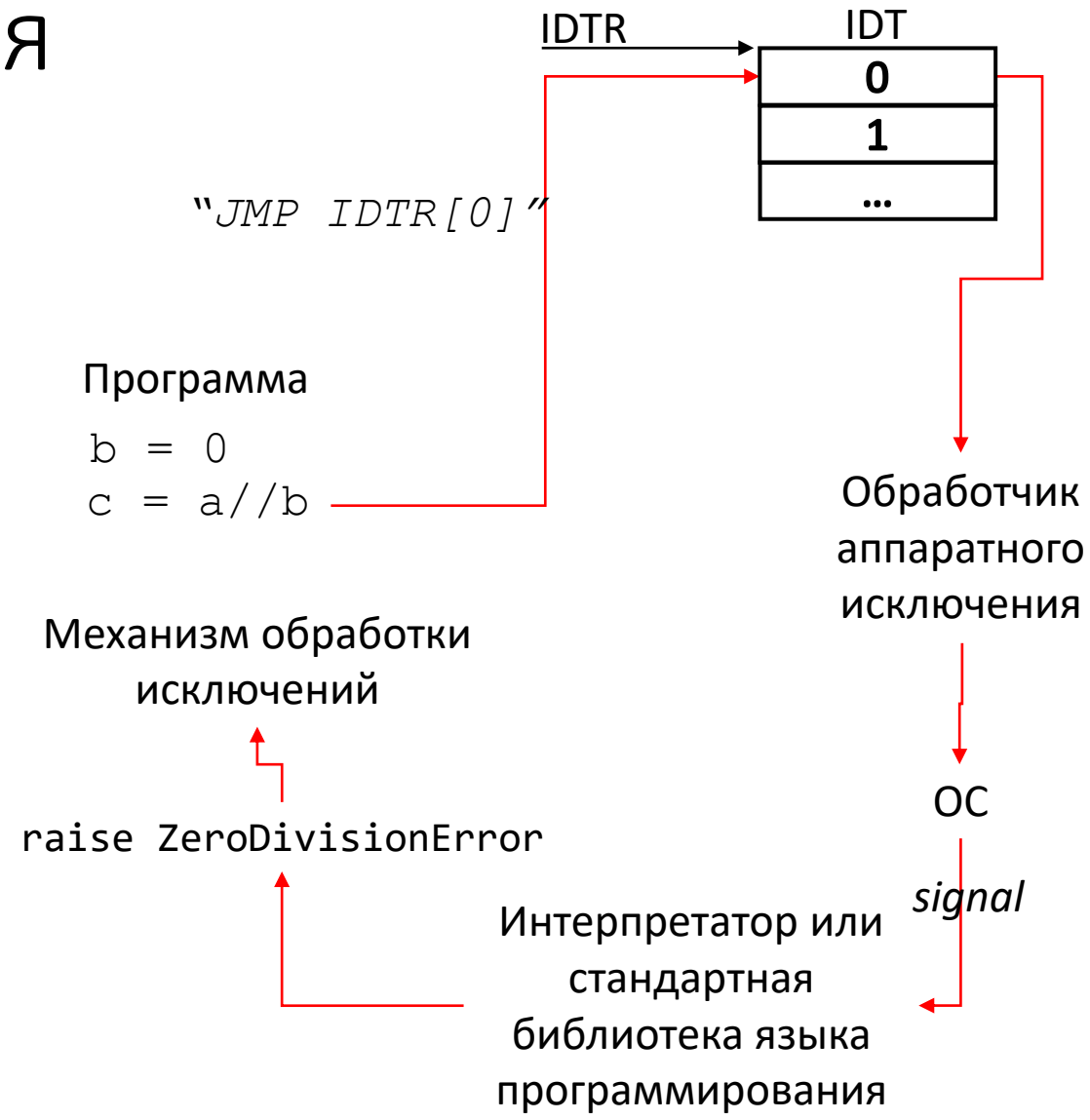
Одним из подвидов прерываний являются **исключения**.

Исключения генерируются самим процессором при определенных (как правило, неблагоприятных) условиях. При этом на стек дополнительно кладется код ошибки, более конкретно определяющий ситуацию.

Первые 32 номера прерываний зарезервированы за исключениями.

Обычно подобные исключения ОС преобразует в сигналы, которые затем превращаются стандартной библиотекой в программные исключения.

Флаг RFLAGS.IF не влияет на получение аппаратных исключений и не устанавливается автоматически при их обработке.



Некоторые аппаратные исключения

Номер	Исключение
0	Деление на 0
1	Отладка (см. флаг TF)
3	Точка останова (инструкция `int3`)
4	Арифметическое переполнение (инструкция `into`)
6	Неизвестный опкод
12	Ошибка стека
13	Общая защита
14	Отсутствие страницы
16	Исключение x87
19	Исключение SSE
30	Нарушение безопасности

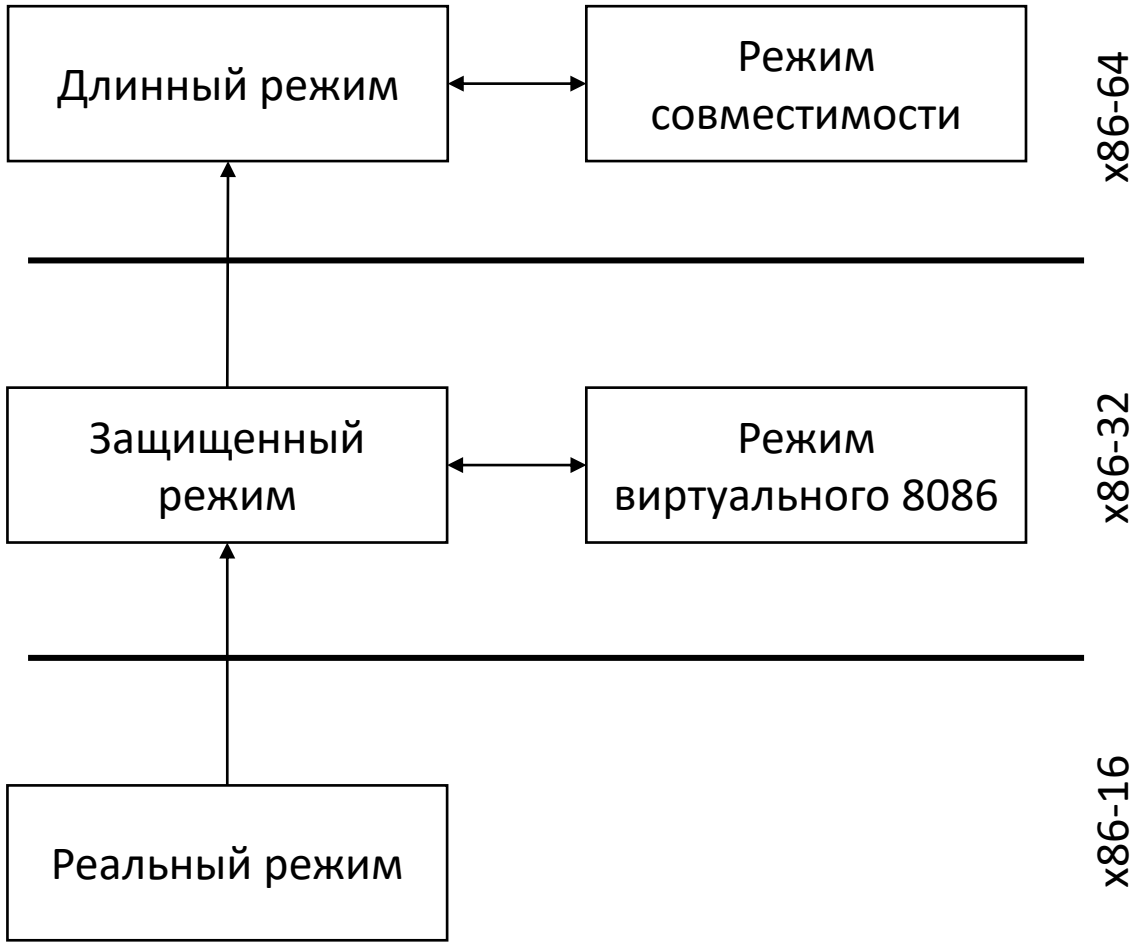
Режимы работы

С приходом новой версии архитектуры процессоры могут приобретать новые возможности (и иногда терять старые).

В x86 идея совместимости является одной из центральных - процессор с ISA x86-64 обязан уметь запускать программы, написанные под x86-32 и x86-16. С этой целью вводятся разные **режимы работы ЦП**.

Режимы работы практически полностью повторяют эволюцию ISA.

Каждый из режимов работы соответствует набору активных функций.



Управляющие регистры

За включение или отключение функций процессора отвечают 3 группы регистров:

- Регистр `FLAGS`;
- Управляющие регистры `CR0-8`;
- Регистры, зависящие от модели ЦП(**MSR,** Model Specific Register) – регистры, структура которых явно не фиксирована в ISA, но (иногда) доступна в документации на конкретный ЦП. Доступ к таким производится инструкциями `RDMSR/WRMSR`.

Примером MSR-регистра является регистр `EFER` (код 0xC0000080)

CR0	Флаги управления процессором
CR1	Недоступен
CR2	Адрес ошибки страницы
CR3	Адрес таблицы трансляции
CR4	Флаги управления процессором
CR5-7	Недоступны
CR8	Приоритет задачи

EFER	Флаги длинного режима

Реальный режим

В **реальном режиме** процессор эквивалентен процессору 8086:

- 20-битное адресное пространство;
- 16-битные регистры (старшие части регистров недоступны);
- прямая адресация физической памяти;
- полный доступ ко всем ресурсам компьютера;
- таблица IDT содержит прямые адреса обработчиков.

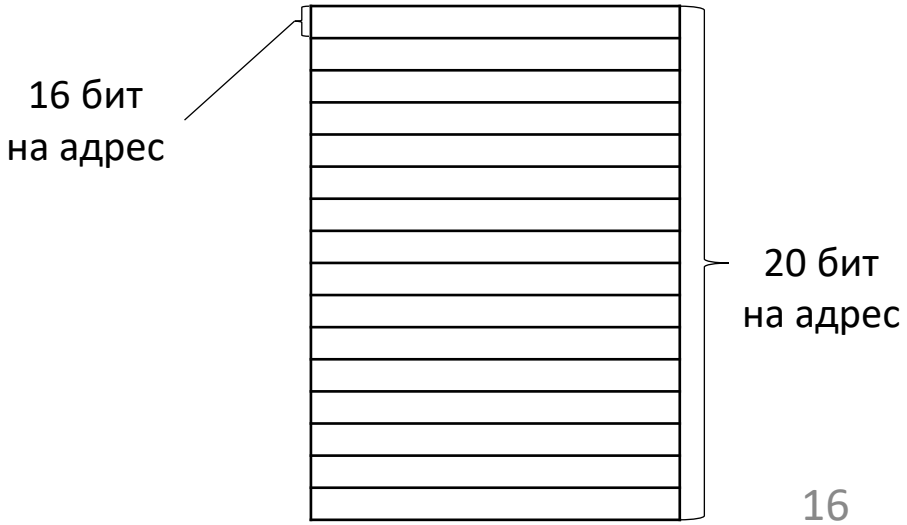
Процессор в момент включения запускается в реальном режиме.

Адресация в реальном режиме

В процессоре 8086 программа могла адресовать все адресное пространство без ограничений.

Поскольку шина адреса была 20-битной, а регистры – 16-битными, для адресации использовался вспомогательный механизм с использованием сегментных регистров:

- CS – сегмент кода.
- DS – сегмент данных.
- SS – сегмент стека .
- ES – дополнительный (extra) сегмент.



Адресация в реальном режиме

При использовании сегментной адресации значение сегментного регистра интерпретируется как начальный адрес сегмента, к которому прибавляется линейное смещение.

Адрес начала сегмента перед сложением сдвигается на 4 влево. Таким образом из 2-х 16-битных значений формируется 20-битный адрес.

При чтении кода неявно используется регистр CS (`jmp/call [AX]` = `jmp/call [CS:AX]`), при обращении к стеку – SS (`mov [sp],2` = `mov [ss:sp],2`), в остальных случаях – сегмент DS.

Регистр ES мог устанавливаться ОС/программистом для адресации дополнительных данных.

```
                mov ax,[es:si]

ES: 1010 1010 1010 1010 0000
SI: 0000 0001 0111 0110 1101
=
    1010 1100 0010 0000 1101
```

16 бит – 64 КБ

↓

20 бит – 1 МБ

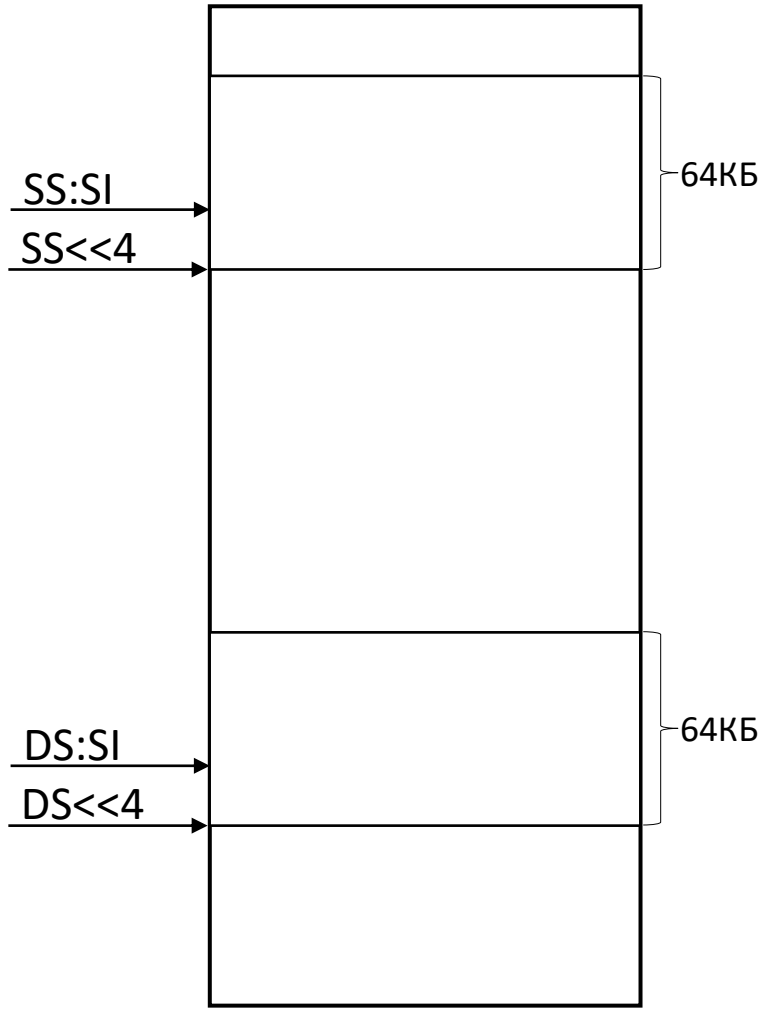
Адресация в реальном режиме

```
      mov ax,[ss:si]

SS: 1010 1010 1010 1010 0000
SI: 0000 0001 0111 0110 1101
=
    1010 1100 0010 0000 1101
```

```
      mov ax,[ds:si]

SS: 0101 0101 0101 0101 0000
SI: 0000 0001 0111 0110 1101
=
    0101 0110 1100 1011 1101
```



Защищенный режим

Из реального режима процессор может быть переключен в защищенный режим.

В защищенном режиме:

- 32-битное адресное пространство;
- 32-битные регистры ;
- сегментная защита памяти, регистры LDTR, GDTR;
- [При `CR3.PG=1`] страничная адресация (3 уровня таблиц, 4 уровня при `CR4.PAE=1`*);
- концепция колец защиты;
- таблица IDT содержит дескрипторы вместо прямых указателей.
- аппаратная поддержка многозадачности**, регистр TR.

*PG=PaGing; PAE=Physical Address Extension

** которую в итоге никто не использует ☺

Кольца защиты

В защищенном режиме явно вводится разделение инструкций на привилегированные и непривилегированные. Данное разделение выражено в виде т.н. **колец защиты**.

Кольцо 3 имеет наименьший уровень привилегий. Обычные программы работают в кольце 3.

Кольцо 0 имеет наивысший уровень привилегий. В кольце 0 работает ядро операционной системы.

Кольца 1 и 2 в настоящее время не используются.

Код, работающий в кольце 0, может выполнять любые действия.

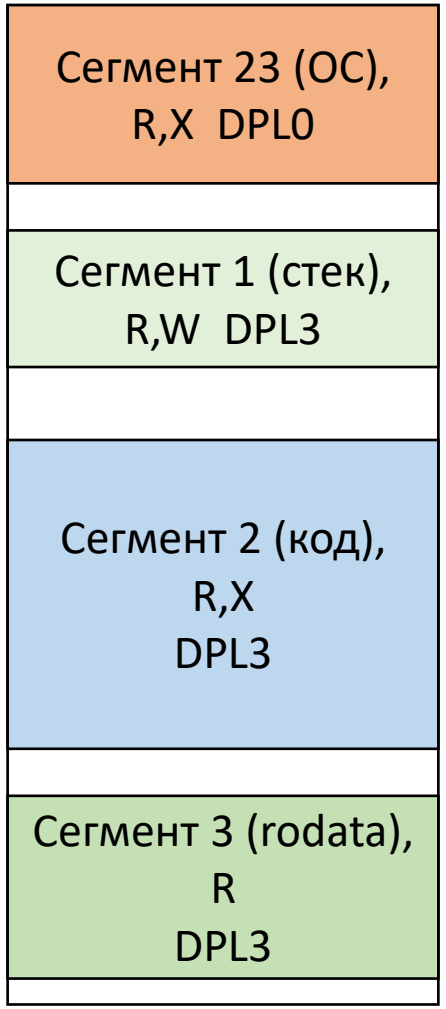
Код, работающий в кольце 3, может выполнять только разрешенные действия (например, доступ к I/O-портам возможен только при определенной конфигурации).

Сегментная адресация (x86-32)

С приходом 32-битной архитектуры изначальный сегментный механизм адресации перестал быть нужным и был видоизменен:

Сегменты стали самостоятельными сущностями. Каждый сегмент получил начальный адрес, собственную длину, разрешения и уровень требуемых привилегий, описываемые **дескриптором сегмента**.

Вместо линейного адреса начала сегмента сегментные регистры стали хранить **селектор сегмента** – индекс сегмента в таблице дескрипторов сегментов.



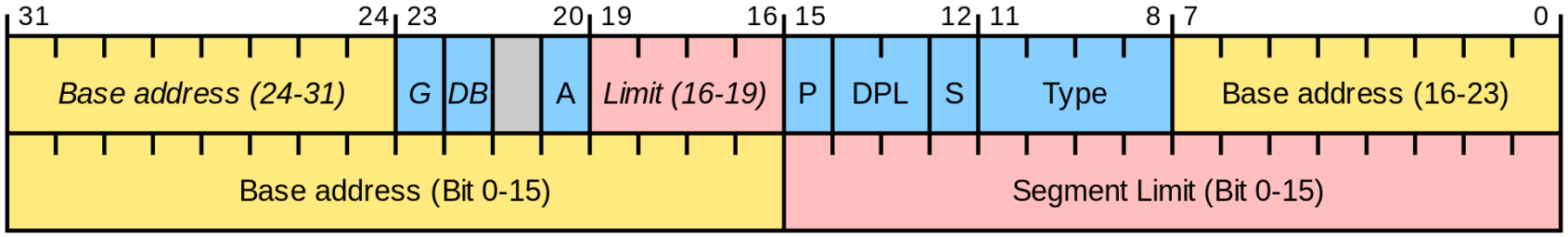
Дескриптор сегмента (x86-32)

Дескриптор сегмента описывает сегмент адресного пространства.

Подобно регистру FLAGS, дескриптор является набором битовых полей.

Среди полей – адрес начала сегмента (*base address*), тип дескриптора, размер сегмента (*segment limit*), разрешения на чтение/запись/выполнение, уровень привилегий (*DPL*).

Замечание: дескриптором может описываться не только сегмент памяти.



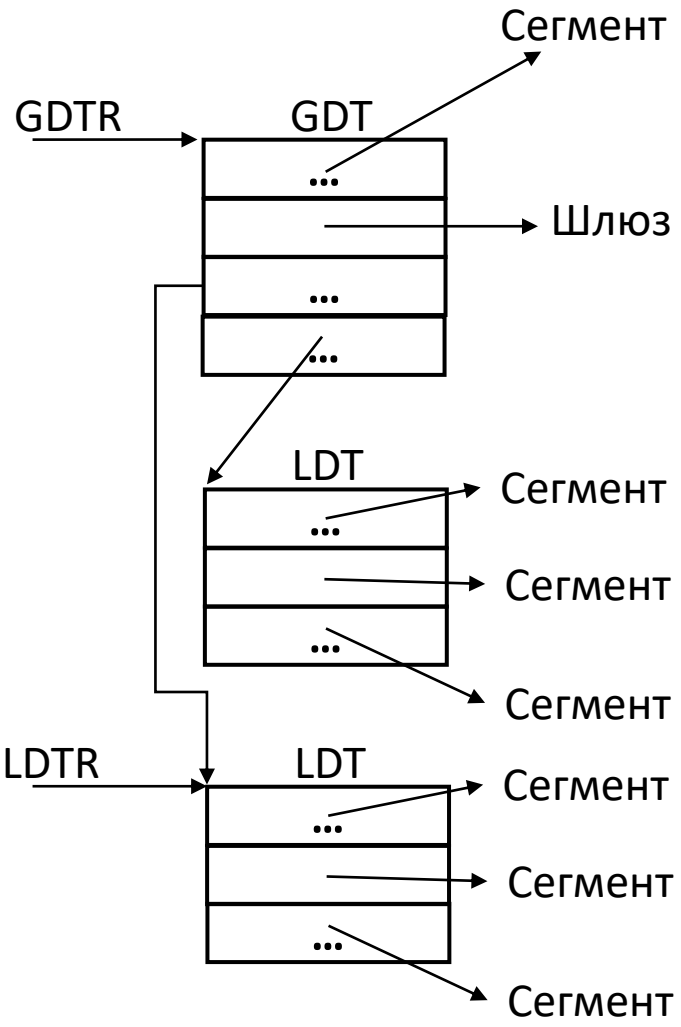
Таблицы дескрипторов(x86-32)

Дескрипторы, подобно адресам обработчиков прерываний, хранятся в специальных таблицах: **локальной таблице дескрипторов** (LDT) и **глобальной таблице дескрипторов** (GDT). Каждой LDT соответствует дескриптор в GDT.

Адреса таблиц хранятся в регистрах **LDTR** и **GDTR**.

Глобальная таблица дескрипторов хранит дескрипторы общесистемных сегментов, дескрипторы задач (tasks), дескрипторы шлюзов (точек перехода между уровнями привилегий) и др.

Локальная таблица дескрипторов хранит адреса сегментов, специфичных для текущей программы. Разным программам могут соответствовать разные LDT.



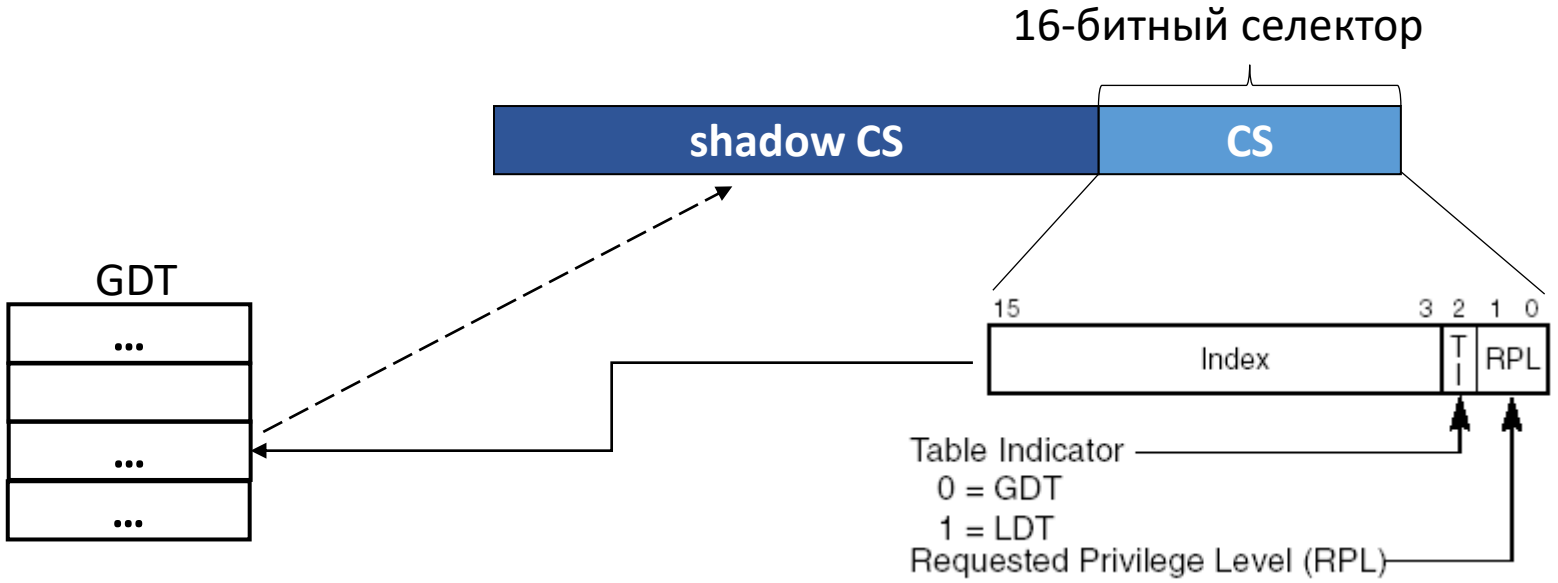
Селектор сегмента(x86-32)

Поскольку таблицы дескрипторов не видны обычным программам, а необходимость в переключении сегментов иногда возникает, для переключения между сегментами используются селекторы.

Селектор сегмента является видимым значением сегментного регистра.

Селектор состоит из индекса дескриптора в таблице, флага таблицы (1 – локальная таблица, 0 – глобальная таблица) и поля запрашиваемого уровня привилегий (RPL, requested privilege level).

Дескриптор сегмента, соответствующего селектору, хранится в теневой части сегментного регистра.



Установка сегментных регистров(x86-32)

Селекторы в регистрах SS/ES/DS/FS/GS могут быть установлены инструкцией `pop ss [ds/fs/gs]` (селектор должен быть сформирован на стеке заранее) .

Селектор в регистре CS может быть изменен только инструкциями `call far/jmp far/ret far`.

Данные инструкции используются для вызова функции из другого сегмента и используют 48-битный адрес: 16 бит – селектор сегмента, 32 бита – адрес вызова.

Инструкция `call far` при этом кладет на стек 48-битный адрес возврата (предыдущий селектор + адрес), и потому требует инструкции `retf` при возврате.

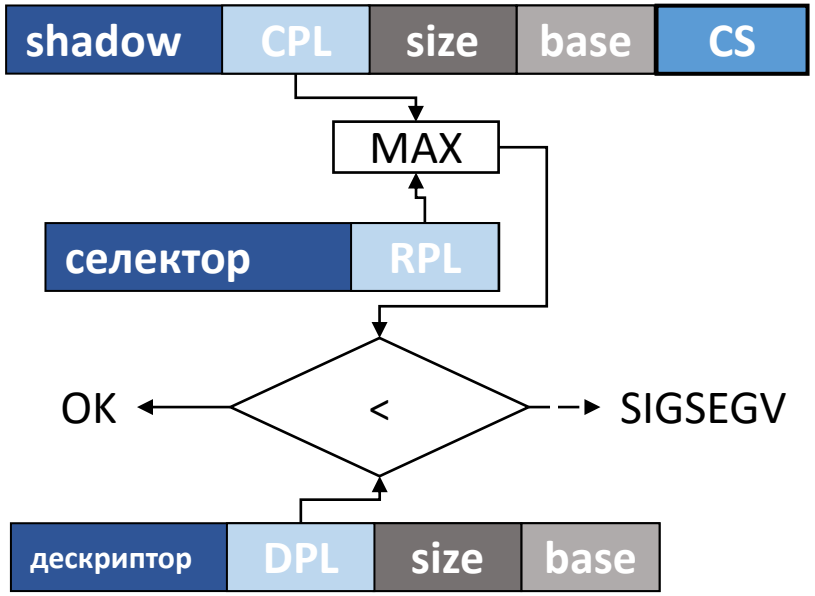
```
push dword 0xAABBCCDD
push word  0x13;селектор
call far [esp]
```

Проверка уровня привилегий(x86-32)

При установке сегментного регистра происходит проверка привилегий. Проверяются 3 значения:

- Текущий уровень привилегий (*CPL*, current privilege level) совпадает с уровнем привилегий дескриптора в теневой части CS (уровень привилегий текущего сегмента кода).
- Запрашиваемый уровень привилегий (*RPL*) берется из загружаемого селектора.
- Уровень привилегий дескриптора (*DPL*, descriptor privilege level) берется из битового поля дескриптора.

Если *DPL<max(CPL, RPL)*, то возникает исключение.

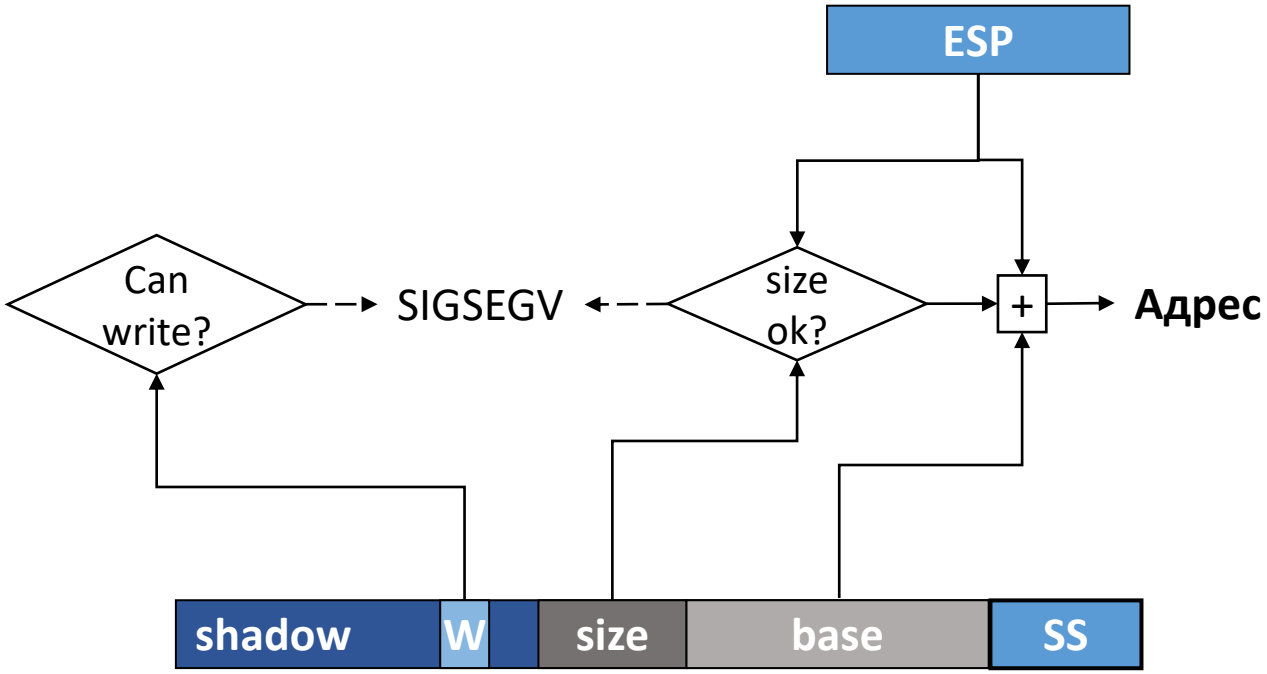


Сегментная адресация(x86-32)

```
mov [ss:esp],eax
```

Поскольку вся адресация (явно или неявно) происходит с участием сегментов, итоговый адрес формируется, как сумма 32-битного линейного адреса и 32-битного адреса начала сегмента.

При этом, если итоговый адрес выходит за границу сегмента, или нарушаются права доступа (например, попытка записи в read-only сегмент) - возникает ошибка сегментации.



Переход между кольцами

Номер текущего кольца защиты соответствует DPL дескриптора в теневой части сегментного регистра CS. Как следствие, переход между кольцами защиты соответствует переключению селектора в CS.

Однако попытка загрузить дескриптор с DPL < CPL вызовет ошибку. Для того, чтобы переходить между кольцами защиты используются специальные механизмы.

Данные механизмы нужны, в частности, для осуществления **системных вызовов** – передаче контроля ядру ОС для выполнения определенных действий в кольце 0 с возвратом результата обратно в кольцо 3.

В роли механизма переключения между кольцами могут использоваться:

- дескрипторы шлюзов и задач (не используются в современных ОС);
- механизм прерываний;
- специальные инструкции syscall/sysenter.

Механизм шлюзов не используется в современных ОС и рассматриваться не будет.

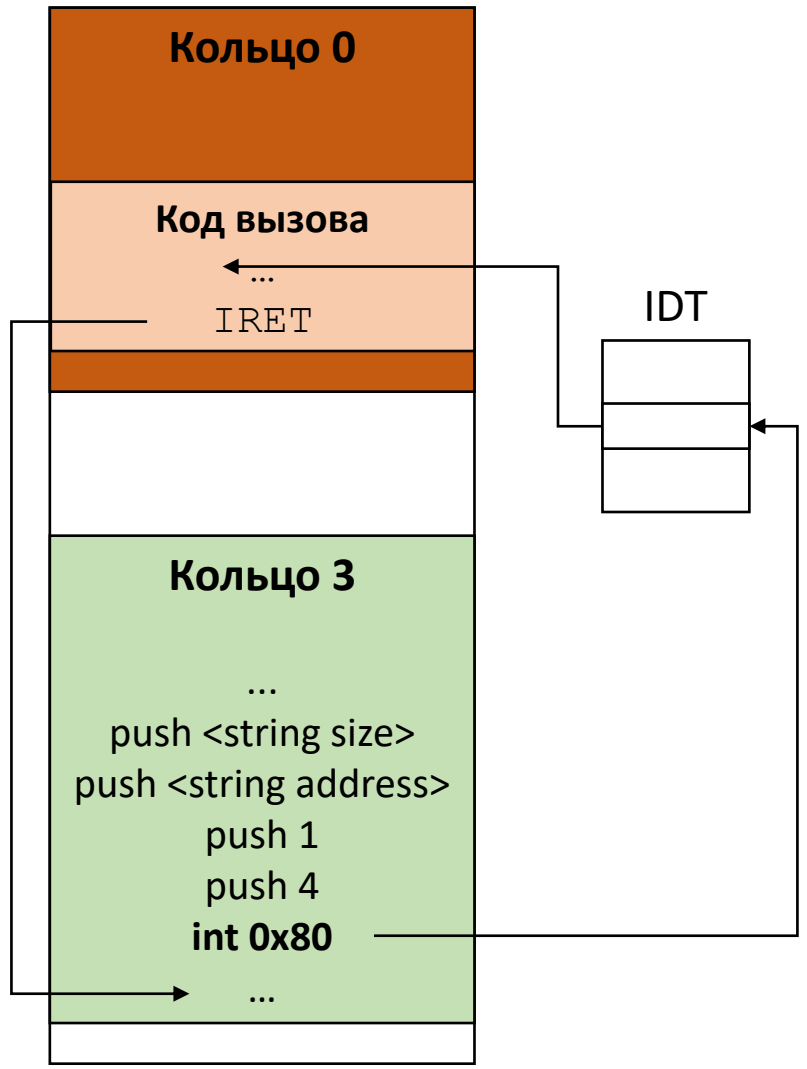
Переход через прерывание

В защищенном режиме в IDT находятся дескрипторы обработчиков прерываний (interrupt gate) и т.н. trap gate.

Дескриптор прерывания может быть сконфигурирован так, что его можно будет вызывать напрямую из программы инструкцией *int N*.

Разработчики ОС могут зарезервировать некоторые номера прерываний для обеспечения взаимодействия ОС <-> программа. Номера таких прерываний записываются в документацию по ОС.

Поскольку адрес точки перехода фиксирован, атакующий не может выполнить произвольный код.



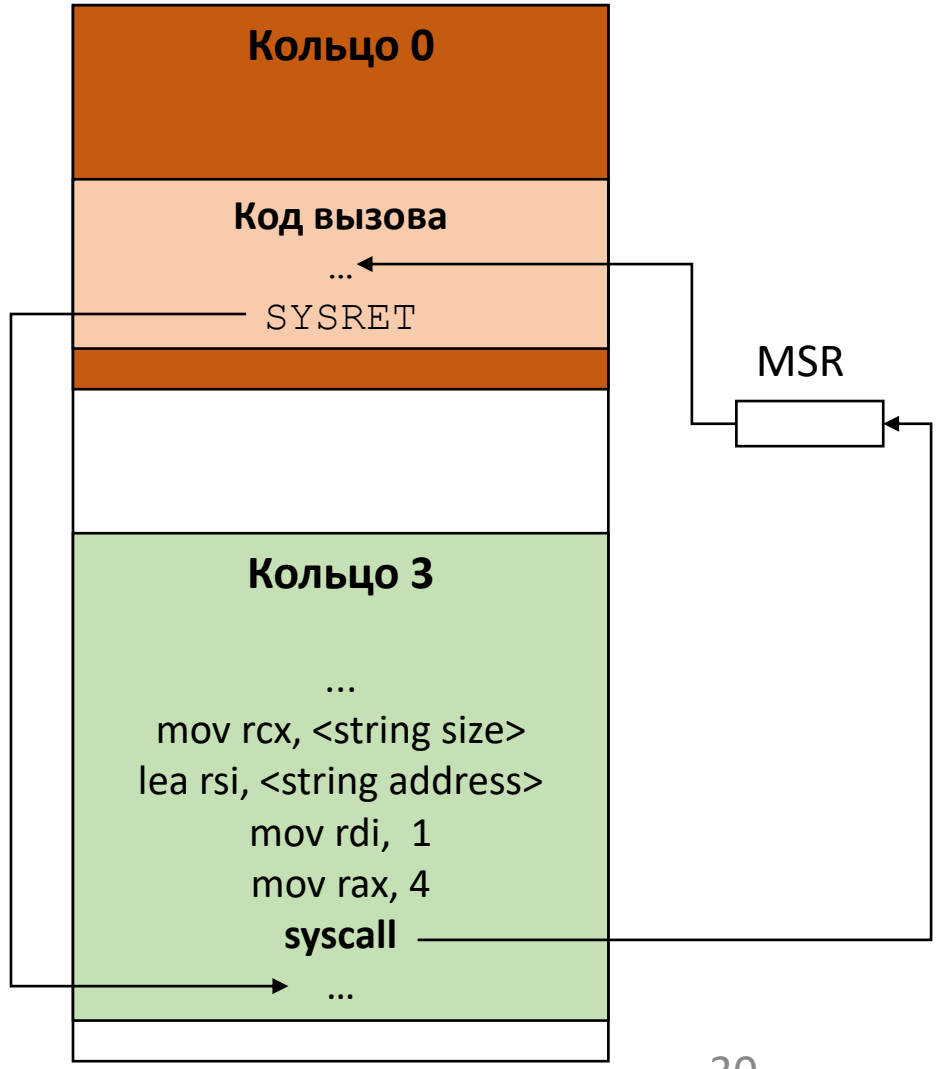
Переход специальной инструкцией

Со временем появились специализированные инструкции перехода из кольца 3 в кольцо 0 – SYSENTER и SYSCALL. *SYSENTER не поддерживается в длинном режиме.*

Оба типа инструкций используют выделенный MSR-регистр, который хранит адрес кода системных вызовов.

Поскольку адрес точки перехода фиксирован, атакующий не может выполнить произвольный код.

.



Task State Segment

Еще одним нововведением является поддержка аппаратной многозадачности.

Изначально предполагалось, что каждой запущенной программе будет соответствовать структура **TSS** (Task State Segment).

Для текущей программы адрес TSS хранится в регистре **TR** (task register).

Предполагалось, что при переключении на другую программу / на обработчик прерываний в TSS будут сохраняться регистры общего назначения. При обратном переключении регистры восстанавливались бы обратно.

SSP			+68h
I/O-Permission Bitmap Base Address	Reserved, IGN	T	+64h
Reserved, IGN	LDT Selector		+60h
Reserved, IGN	GS		+5Ch
Reserved, IGN	FS		+58h
Reserved, IGN	DS		+54h
Reserved, IGN	SS		+50h
Reserved, IGN	CS		+4Ch
Reserved, IGN	ES		+48h
EDI			+44h
ESI			+40h
EBP			+3Ch
ESP			+38h
EBX			+34h
EDX			+30h
ECX			+2Ch
EAX			+28h
EFLAGS			+24h
EIP			+20h
CR3			+1Ch
Reserved, IGN	SS2		+18h
ESP2			+14h
Reserved, IGN	SS1		+10h
ESP1			+0Ch
Reserved, IGN	SS0		+08h
ESP0			+04h
Reserved, IGN	Link (Prior TSS Selector)		+00h

Task State Segment

Помимо места для сохранения регистров, TSS содержит 6 полей **SSn/ESPn**, которые хранят селектор и указатель стека для каждого кольца защиты. При переходе в другое кольцо защиты процессор автоматически переключается на соответствующий стек. Это повышает безопасность и уменьшает количество потенциальных ошибок.

Кроме того, TSS содержит поле IOPB Address (IO Permissons Base), которое указывает на битовую карту доступных портов ввода/вывода. Если бит N в этой карте равен 0, то доступ к порту N закрыт.

SSP			+68h
I/O-Permission Bitmap Base Address	Reserved, IGN	T	+64h
Reserved, IGN	LDT Selector		+60h
Reserved, IGN	GS		+5Ch
Reserved, IGN	FS		+58h
Reserved, IGN	DS		+54h
Reserved, IGN	SS		+50h
Reserved, IGN	CS		+4Ch
Reserved, IGN	ES		+48h
EDI			+44h
ESI			+40h
EBP			+3Ch
ESP			+38h
EBX			+34h
EDX			+30h
ECX			+2Ch
EAX			+28h
EFLAGS			+24h
EIP			+20h
CR3			+1Ch
Reserved, IGN	SS2		+18h
ESP2			+14h
Reserved, IGN	SS1		+10h
ESP1			+0Ch
Reserved, IGN	SS0		+08h
ESP0			+04h
Reserved, IGN	Link (Prior TSS Selector)		+00h

Task State Segment

В современных ОС используется полностью программный способ управления задачами.

Обычно ОС создает по одному TSS для каждого ядра процессора. Далее TSS используется только для автоматического переключения между стеками разных уровней привилегий и управления доступов к портам ввода/вывода.

Для сохранения состояния программы ОС использует свои внутренние структуры данных, работа с которыми происходит на программном уровне.

SSP			+68h
I/O-Permission Bitmap Base Address	Reserved, IGN	T	+64h
Reserved, IGN	LDT Selector		+60h
Reserved, IGN	GS		+5Ch
Reserved, IGN	FS		+58h
Reserved, IGN	DS		+54h
Reserved, IGN	SS		+50h
Reserved, IGN	CS		+4Ch
Reserved, IGN	ES		+48h
EDI			+44h
ESI			+40h
EBP			+3Ch
ESP			+38h
EBX			+34h
EDX			+30h
ECX			+2Ch
EAX			+28h
EFLAGS			+24h
EIP			+20h
CR3			+1Ch
Reserved, IGN	SS2		+18h
ESP2			+14h
Reserved, IGN	SS1		+10h
ESP1			+0Ch
Reserved, IGN	SS0		+08h
ESP0			+04h
Reserved, IGN	Link (Prior TSS Selector)		+00h

Виртуальная память

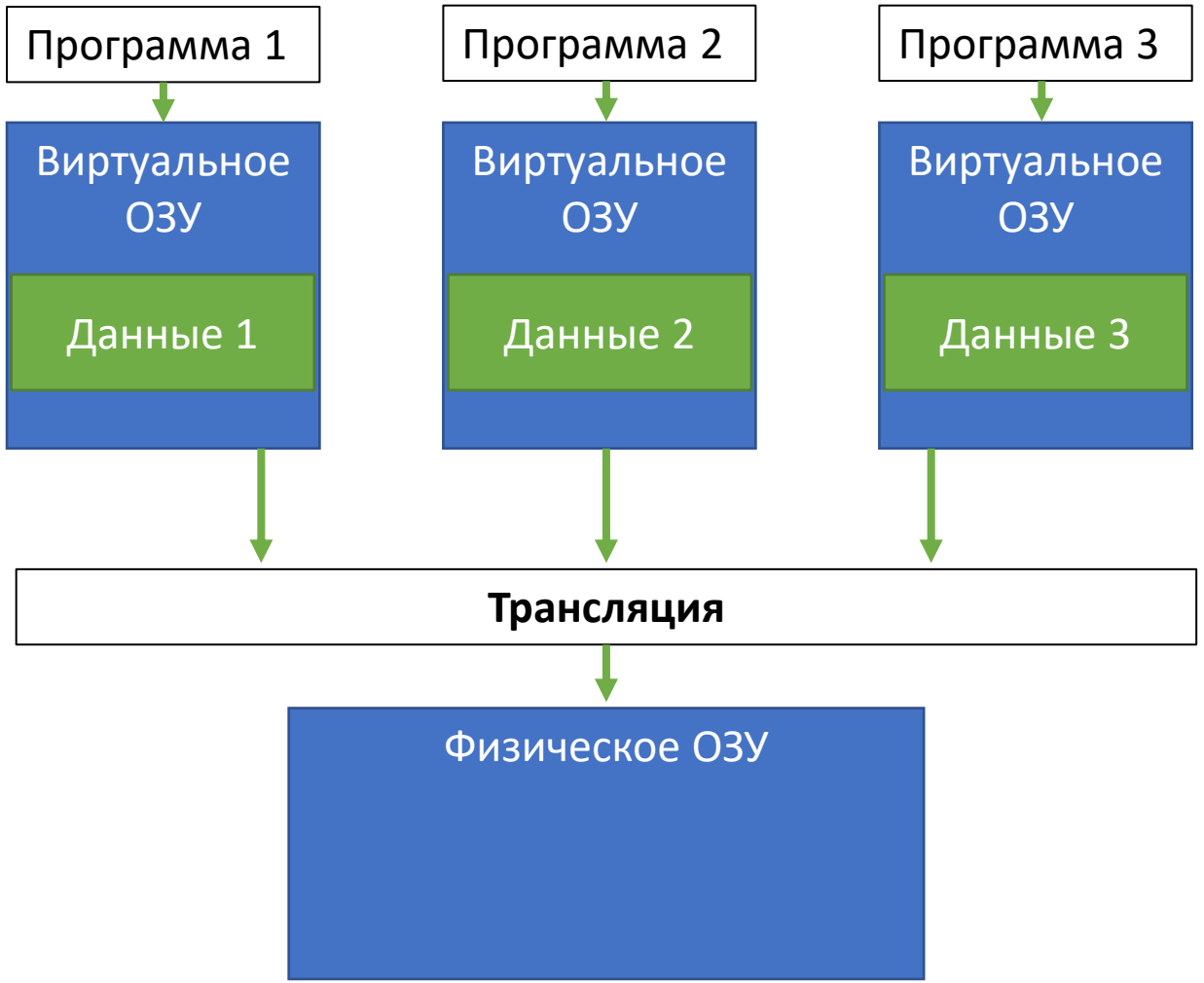
При использовании виртуальной памяти каждая программа имеет **собственное адресное пространство** и может получить доступ к данным другой программы.

Распределением физической памяти между программами управляет операционная система.

Процесс сопоставления виртуального и физического адреса называется **трансляцией**.

Формально, для решения задачи трансляции можно назначить каждой программе собственный набор сегментов.

Однако программ может быть много, а максимальное число записей в GDT/LDT ограничено. Кроме того, сегменты являются слишком «крупными» и загрузка/выгрузка сегмента из памяти затратны.



Страничная адресация

При страничной адресации виртуальная память делится на области равного размера – **страницы**. Типовой размеры страницы - 4КБ.

Поскольку объем физической памяти ограничен, в нее загружаются только те страницы, которые необходимы исполняющимся программам в данный момент.

При этом страницы могут загружаться куда угодно – нет никакого соответствия между физическим адресом страницы и ее виртуальным адресом.

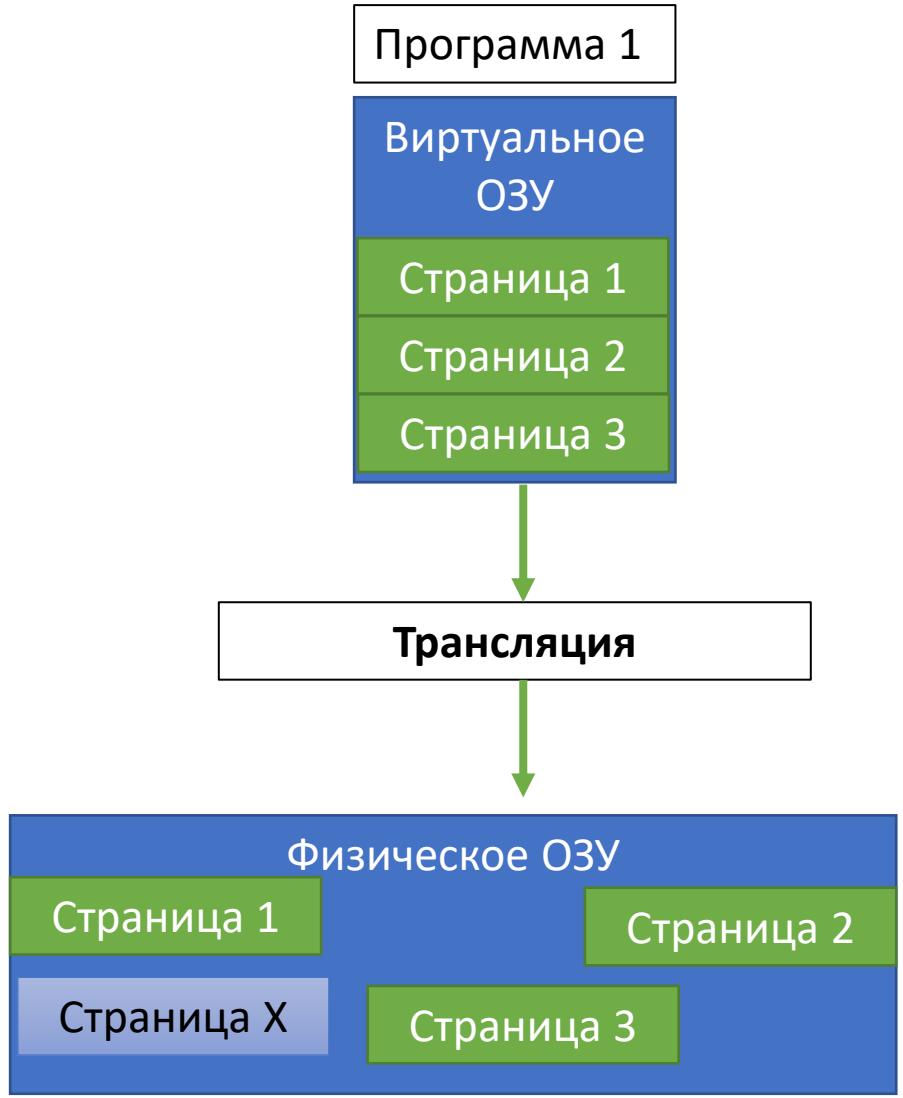


Таблица страниц

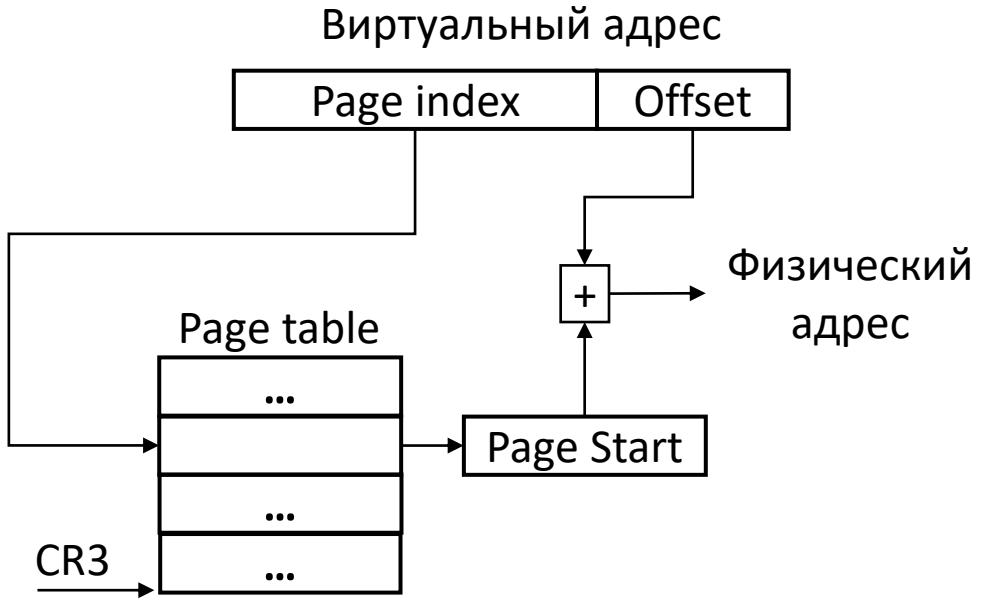
Для отображение виртуального адреса на физический используются таблицы страниц.

Виртуальный адрес делится на 2 части:

- Индекс в таблице страниц;
- Смещение в странице.

Каждая программа имеет свою таблицу страниц, адрес которой находится в специальном регистре **CR3**.

В x86-32 используется 3 уровня таблиц. В x86-64 – 4 уровня.



Файл подкачки

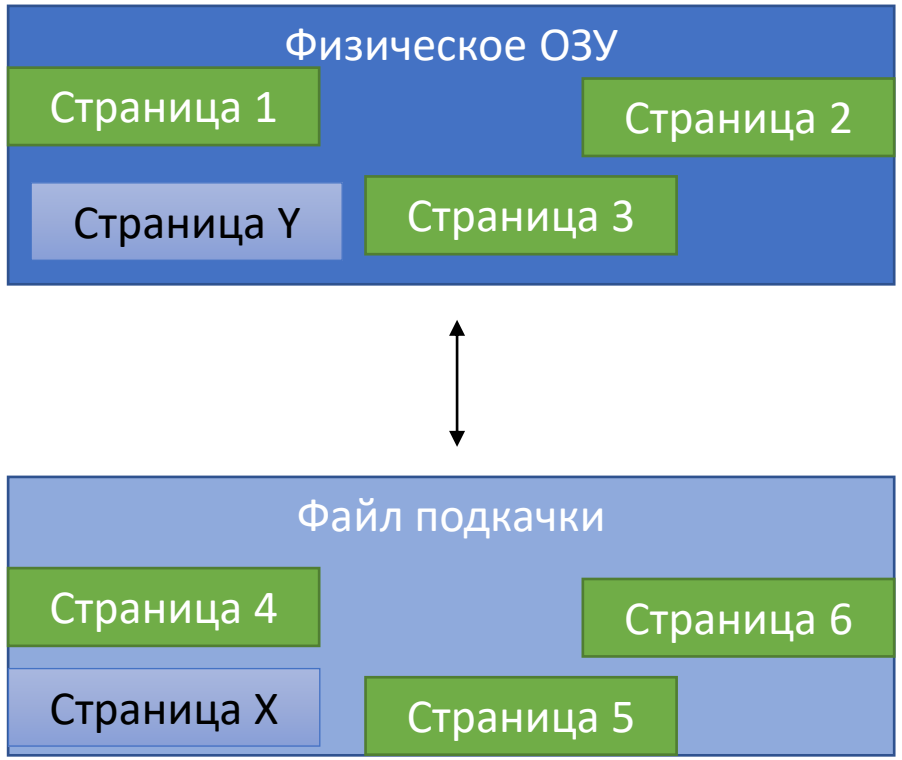
Если памяти хватает, то все страницы памяти всех процессов находятся в физической памяти.

Если памяти не хватает, то ОС начнет сбрасывать неактивные страницы в т.н. **файл подкачки**, для того, чтобы освободить место для новых страниц.

Если программа обратится к странице, которая не загружена в память, то при обращении возникнет аппаратное исключение 14 – ошибка отсутствия страницы (**page fault**). *При этом адрес, который вызвал ошибку, будет занесен в регистр CR2.*

Обработчик этого исключения:

-определит запрошеную страницу (X) по значению в CR2;
-выберет наименее используемую страницу(Y);
-сбросит в файл подкачки Y;
-заменит в ОЗУ Y на X.

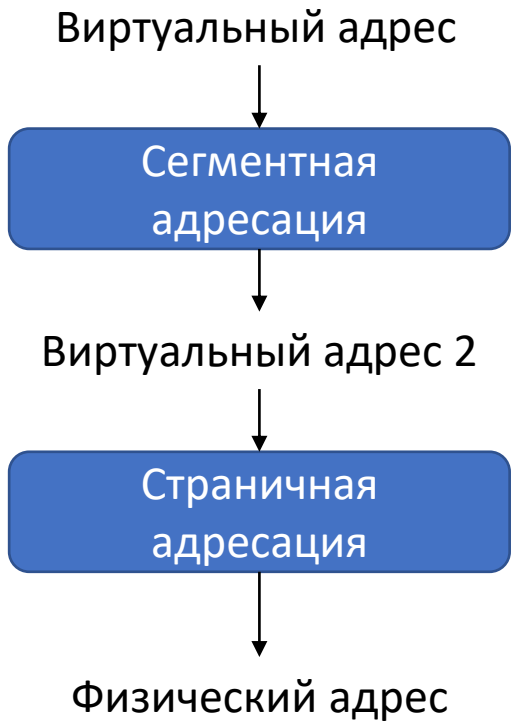


Страничная адресация и сегменты

Страничная адресация работает на более низком уровне, чем сегментная адресация.

Если сегментная адресация используется, то:

- Адреса начала сегментов являются виртуальными.
- Результатом сегментной адресации также являются виртуальный адрес.
- Полученный виртуальный адрес транслируется в физический механизмами страничной адресации.



Выделение памяти и страничная адресация

При выделении памяти с помощью malloc()/new()/mmap() операционная система добавляет записи в таблицу страниц программы.

При первом обращении к новой странице возникает ошибка отсутствия страницы (т.н. soft page fault). Обработчик находит свободную область в ОЗУ и назначает физический адрес странице.

ОС может предоставлять системный вызов для выделения общей для нескольких программ памяти. В этом случае таблицы страниц каждой из программ будут иметь записи, ссылающиеся на одни и те же физические страницы (при этом виртуальные адреса страниц могут отличатся).

Длинный режим

Из защищенного режима процессор может быть переключен в длинный режим.

В длинном режиме:

- 64-битное адресное пространство;
- 64-битные регистры ;
- страничная адресация с постраничной защитой доступа;
- *сегментная защита доступа к памяти отключена*;
- дескрипторы в GDT используются только для перехода между кольцами защиты и входа в режим совместимости.

Сегментная адресация в x86-64

В x86-64 разбиение памяти на сегменты более не используется – все адресное пространство является одним сегментом.

Таблицы LDT и регистр LDTR более не используются.

Формат дескрипторов в GDT не изменяется, но поля размера сегмента и начала сегмента игнорируются. Фактически, сегменты используются только для работы с уровнями привилегий. Защита по уровню привилегий при установке сегментных регистров производится без изменений.

Task State Segment в длинном режиме

TSS в длинном режиме больше не предназначен для хранения состояния процессора.

В TSS по прежнему хранятся указатели стека для каждого кольца защиты и указатель на карту доступа к портам.

Освободившееся пространство используется хранения адресов стеков для обработки прерываний (8-шт, номер стека указывается в дескрипторе обработчика прерывания).

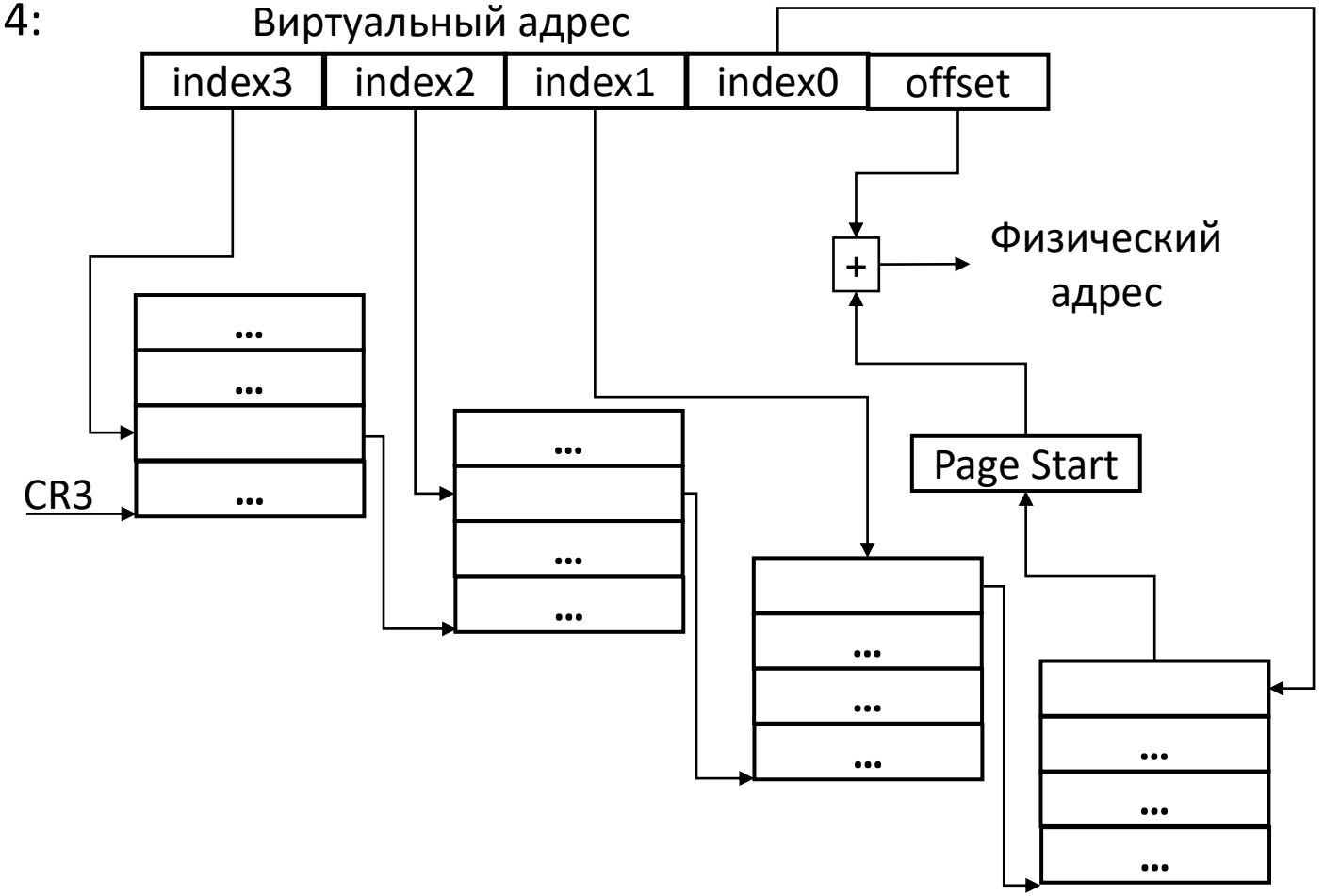
I/O Map Base Address	Reserved, IGN	+64h
Reserved, IGN		+60h +5Ch
IST7[63:32]		+58h
IST7[31:0]		+54h
IST6[63:32]		+50h
IST6[31:0]		+4Ch
IST5[63:32]		+48h
IST5[31:0]		+44h
IST4[63:32]		+40h
IST4[31:0]		+3Ch
IST3[63:32]		+38h
IST3[31:0]		+34h
IST2[63:32]		+30h
IST2[31:0]		+2Ch
IST1[63:32]		+28h
IST1[31:0]		+24h
Reserved, IGN		+20h +1Ch
RSP2[63:32]		+18h
RSP2[31:0]		+14h
RSP1[63:32]		+10h
RSP1[31:0]		+0Ch
RSP0[63:32]		+08h
RSP0[31:0]		+04h
Reserved, IGN		+00h

4 уровня трансляции в x64

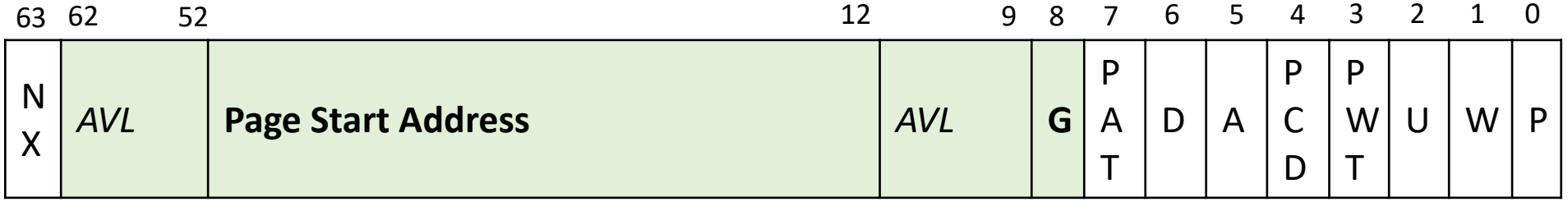
В x86-64 вместо 1 таблицы используются 4:

- Таблица карт страниц
- Таблица указателей каталогов страниц
- Таблица каталога страниц
- Таблица страниц.

Регистр CR3 указывает только на таблицу верхнего уровня.



Структура записи в таблице страниц(x86-64)

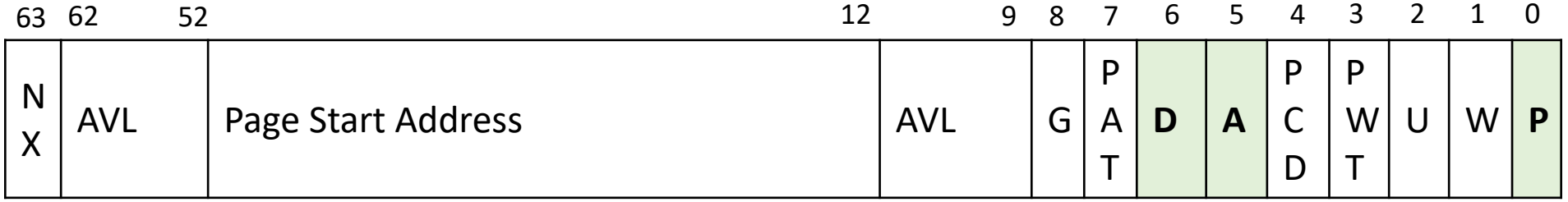


Поле **Page Start Address** содержит адрес начала страницы. Так как страница не может быть меньше 4 КБ, а объем физической оперативной памяти не превышает 2^{52} байт, достаточно хранить только 40 бит адреса (младшие 12 бит равны 0).

Бит **G** (Global) указывает, что страница является глобальной (используется всеми процессами). Обычно в таких страницах располагается код и данные ядра операционной системы.

Некоторые биты не используются процессором при трансляции адреса. Данные биты могут использоваться ядром ОС для хранения некоторых дополнительных данных о странице.

Структура записи в таблице страниц(x86-64)



Бит **P** (Present) равен 1, если страница физически находится в памяти, и 0 – если страница отсутствует.

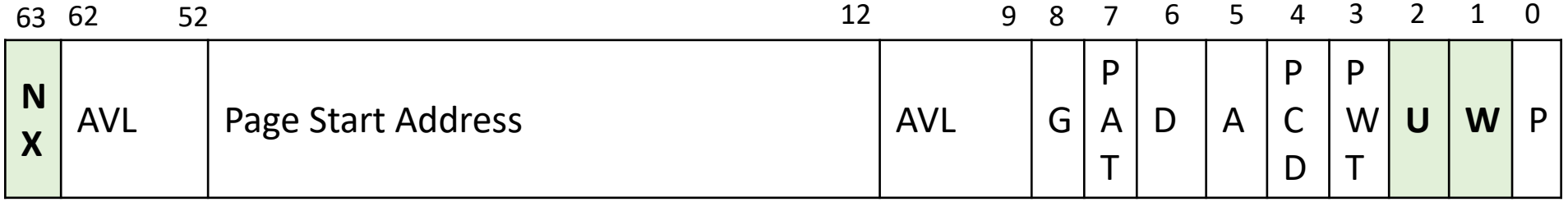
Если программа пытается обратиться к странице, которой нет в памяти, возникает исключение 14 (Page Fault), обработчик которого загружает страницу в память и выставляет бит P=1.

Бит **A** (Accessed) указывает, что к странице с момента загрузки в память обращались хотя бы 1 раз.

Бит **D** (Dirty) указывает, что в страницу с момента загрузки была произведена запись.

Эти биты также используются обработчиком исключения отсутствия страницы.

Структура записи в таблице страниц(x86-64)



Биты **NX** (No Execute), **W** (Write) и **U** (User) указывают права доступа.

Если бит W=0, то запись в страницу запрещена.

Если бит NX=0, то исполнение кода в странице запрещено.

Если бит U=0, то доступ к странице из кольца 3 запрещен (т.е. доступ имеет только код ядра).

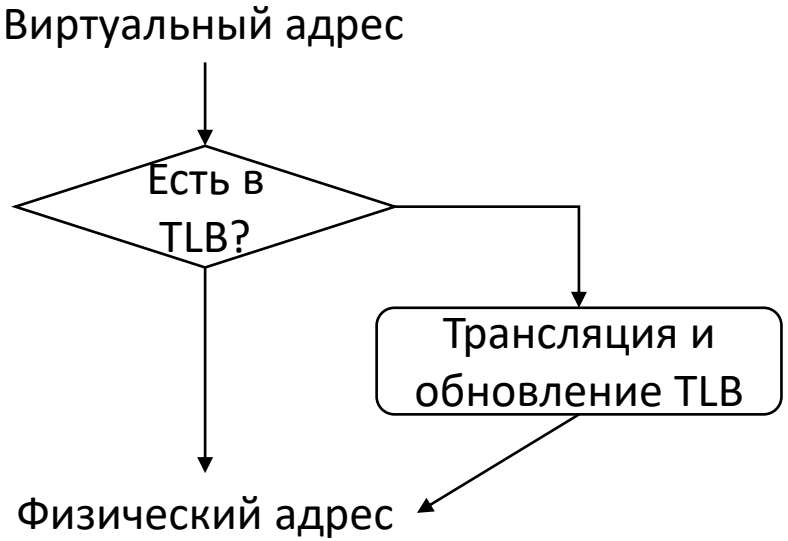
Данные биты позволяют отказаться от сегментной защиты памяти.

Translation Lookaside Buffer

Поскольку проводить трансляцию адреса накладно, результаты трансляции кэшируются в буфер трансляции адресов (Translation Lookaside Buffer).

Во избежание утечки данных, TLB частично либо полностью очищается при переключении между процессами.

Записи, соответствующие глобальным таблицам, не очищаются, если не запрошена полная очистка TLB.



Режим совместимости

Для поддержки работоспособности программ, скомпилированных для x86-32 и x86-16, длинный режим поддерживает переключение в режим совместимости.

Данный режим эмулирует особенности x86-32 и/или x86-16 только для программ в кольце 3. Программы в кольце 0 продолжают работать в длинном режиме со всеми возможностями – таблицы LDT, GDT, IDT уровня ядра останутся 64-битными.

Так как в режиме совместимости используется сегментная защита, в GDT и LDT для режима совместимости должны быть сформированы соответствующие записи о сегментах.

Переход в режим совместимости происходит автоматически при загрузке в CS дескриптора с флагом L=0. Обратный переход в длинный режим происходит при загрузке в CS дескриптора с флагом L=1.

Ввод/вывод в защищенном и длинном режимах

Рядовые программы не должны иметь доступ к аппаратному обеспечению и инструкциям ввода/вывода. Однако в некоторых случаях это необходимо.

Поле `RFLAGS.IOPL` (биты 13,14) содержит уровень привилегий, требуемый для выполнения инструкций ввода-вывода и изменения флага `RFLAGS.IF`. Для выполнения инструкций необходимо выполнение условия `CPL<=RFLAGS.IOPL`.

Для обычных программ `RFLAGS.IOPL=0`. Изменить само значение `RFLAGS.IOPL` можно только из кольца 0 (т.е. сделать это может только ОС). Для этого ОС останавливает программу, изменяет `RFLAGS.IOPL` и вновь запускает программу.

Кроме того, могут быть установлены ограничения на доступ к отдельным портам в карте доступа к портам (см. регистр TSS).

Если устройство поддерживает отображение в память, то ОС может добавить в таблицу страниц запись с физическим адресом, соответствующим устройству – в этом случае, программа сможет работать с устройством без изменения CPL/IOPL.

BIOS и UEFI

На предыдущем этапе развития ПК первой программой, загружавшейся процессором являлась **BIOS** (Basic Input Output System).

BIOS – минимальная ОС, предназначенная для первичной инициализации ЦП и оборудования и передаче управления загрузчику основной ОС. Код BIOS хранится в отдельной микросхеме на материнской плате.

В настоящее время вместо BIOS в качестве первичного ПО используется **UEFI** (Unified Extensible Firmware Interface).

UEFI расширяет возможности BIOS, в частности, UEFI позволяет загружаться с устройств объемом более 2,2 ТБ. Кроме того, UEFI может иметь графический интерфейс.

Начало загрузки

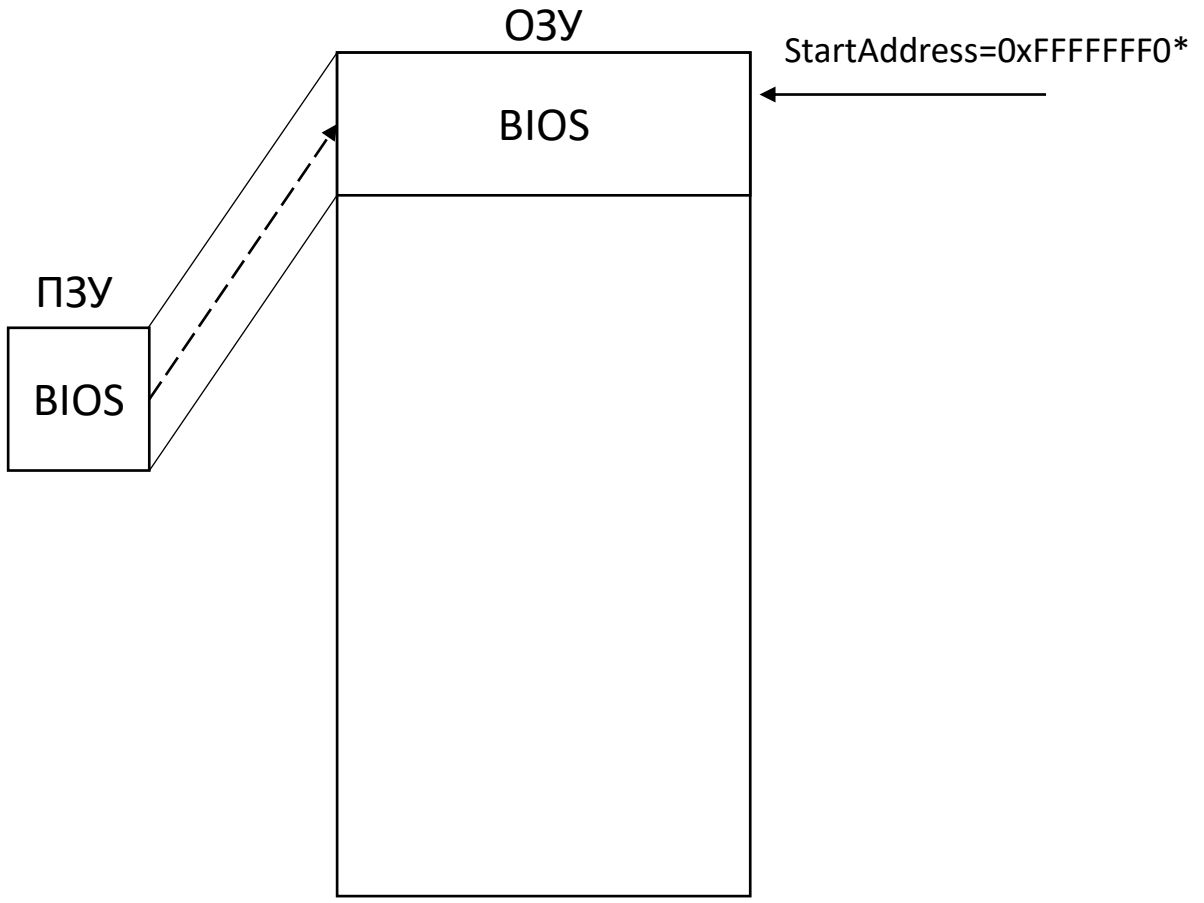
Процессор загружается в реальном режиме.

При старте процессора содержимое ПЗУ-микросхемы, содержащей BIOS/UEFI, отображается в память по фиксированному адресу.

Этот адрес указывается в документации на процессор. Сразу после включения по этому адресу считывает первую инструкцию и начинает выполнение.

Обычно одним из первых действий BIOS/UEFI является копирование себя же в оперативную память (с возможным разархивированием).

После загрузки, BIOS/UEFI выполняет POST (Power On Self Test) – базовую проверку функциональности аппаратуры. При наличии проблемы ее код сообщается звуковым сигналом или с помощью доп. индикаторов на материнской плате.



* В момент старта процессора с точки зрения программы реального режима CS:IP=0xFFFF0 .

Загрузка

- После выполнения POST, BIOS/UEFI считывает код загрузки из загрузочного носителя и передает ему управление.
- Системное ПО* формирует в памяти 32-битные таблицы GDT и IDT, устанавливает значения регистров `GDTR` и `IDTR` и устанавливает флаг `CR0.PE=1`. Установка флага переводит процессор в защищенный режим.
- Системное ПО* формирует в памяти 64-битные таблицы GDT и IDT, устанавливает значения регистров `GDTR`, `IDTR` и активирует (но не включает) длинный режим установкой флага `EFER.LME=1**`.
- Системное ПО* формирует в памяти таблицы страниц 4-х уровней, устанавливает значение регистра CR3 и включает страничную адресацию установкой флагов `CR4.PAE=1`, `CR3.PG=1`. Включение страничной адресации переводит процессор в длинный режим.

*или UEFI **LME=Long Mode Enabled