

# Адресное пространство, физическая и виртуальная память

Луцив Дмитрий Вадимович

Кафедра системного программирования СПбГУ



- 1 Виды адресации (кроме страничной)
  - Прямая адресация
  - Банковые расширения
  - Совместимые расширения и эволюция ЦП
  - Сегментная адресация
- 2 Виртуальная память и страничная адресация
  - Реализация
  - Вытеснение и замещение
  - Пример: Intel i80386
  - TODO: ARM и RISC-V

- **Физический адрес** — номер байта в оперативной памяти, начиная с нулевого
- **Виртуальный (Логический) адрес** — адрес, с которым работает прикладное ПО, например, значение указателя
- **Адресное пространство** — множество логических адресов
- **Адресное преобразование** — вычисление физического адреса по виртуальному

## Виды адресации (кроме страничной)

- Прямая адресация
- Банковые расширения
- Совместимые расширения и эволюция ЦП
- Сегментная адресация

- **Прямая адресация** — способ адресации, при которой физический адрес равен логическому

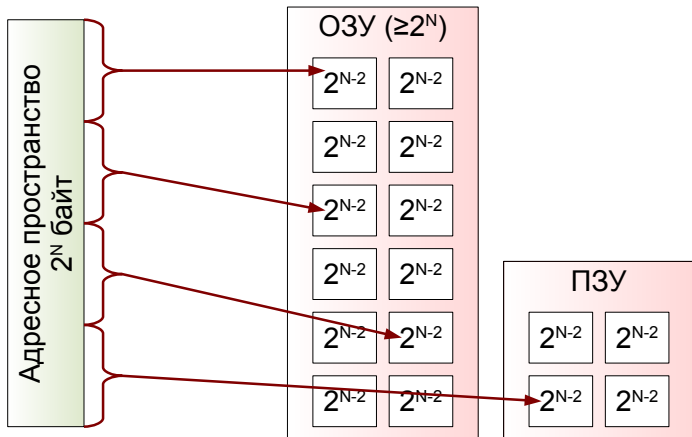
Т.е. адресное преобразование тривиально, фактически «указатель» сразу выдаётся на шину адреса

- Типично для восьмибитной шины данных — 16-битные внутренняя арифметика и шина адреса
- Изначально даже возможные 64KiB не использовались. Примеры:
  - Sinclair ZX 80 — ЦП Z80 (64K), но только 1K ОЗУ; Sinclair ZX
  - Spectrum — тоже Z80, но от 16 К ОЗУ

- Типично для восьмибитной шины данных — 16-битные внутренняя арифметика и шина адреса
- Изначально даже возможные 64KiB не использовались. Примеры:
  - Sinclair ZX 80 — ЦП Z80 (64K), но только 1K ОЗУ; Sinclair ZX
  - Spectrum — тоже Z80, но от 16 K ОЗУ

Для офисных приложений и игр нужны графика и больше памяти

# Сущность банковых расширений



**Банковая адресация** — способ адресации, при которой адресное пространство разделяется на *банки адресного пространства*, которым в соответствие ставятся *банки оперативной памяти*

Соответствие обычно не произвольное, варианты ограничены



## Хорошо

- Процессор тот же, вообще вмешательство в архитектуру минимально
- Дешево стоит
- Память можно расширять произвольно, доступно радиолюбителям
- Сравнительно удобно работать с данными

## Плохо

- Работает не быстро
- Программируется через порты; тяжело менять банки кода, особенно для:
  - Вызова процедур
  - Прерываний
  - Переключения задач
- Разные расширения часто несовместимы

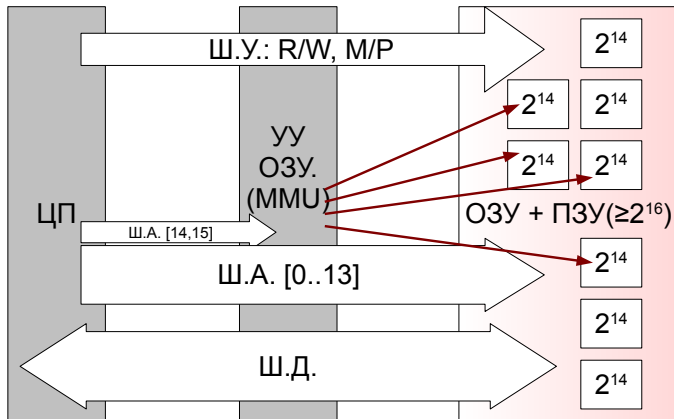
# Пример 1: ZX Spectrum 128K

## ZX Spectrum 128K [↗](#)

```
0xFFFF --+-----+-----+-----+-----+-----+-----+-----+
| Bank 0 | Bank 1 | Bank 2 | Bank 3 | Bank 4 | Bank 5 | Bank 6 | Bank 7 |
|         |         | (also at|         | (also at|         |         |         |
|         |         | 0x8000)|         | 0x4000)|         |         |         |
|         |         |         |         | screen |         | screen |
+ 0xC000 +-----+-----+-----+-----+-----+-----+-----+
| Bank 2 |         Any one of these pages may be switched in.
|         |
|         |
|         |
+ 0x8000 +---+
| Bank 5 |
|         |
|         |
| screen |
+ 0x4000 +-----+
| ROM 0 | ROM 1 | Either ROM may be switched in.
|         |         |
|         |         |
|         |         |
+ 0x0000 +-----+
```

## Пример 1: Как это реализовано

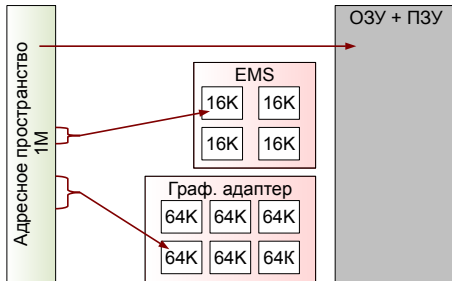
Адресное преобразование вычисляет номер банка ОЗУ по номеру банка (старшим 2 битам) АП. Отображение хранится в регистрах схемы MMU



Адресное преобразование:  $A_{ph} = MMU[A_{log;14...15}] + A_{log;0...13}$

## Пример 2: Окна в адресном пространстве IBM PC

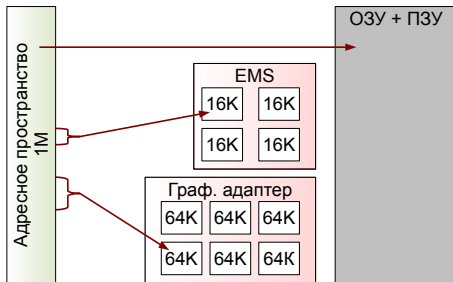
Можно рассматривать, как вариант банковской системы



- Некоторые внешние устройства, например графические адаптеры, поддерживают до сих пор
- Использовалась в EMS — сначала плата расширения, затем встроенная поддержка на материнской плате, затем эмуляция при помощи страничной адресации

## Пример 2: Окна в адресном пространстве IBM PC

Можно рассматривать, как вариант банковской системы



- Некоторые внешние устройства, например графические адаптеры, поддерживают до сих пор
- Использовалась в EMS — сначала плата расширения, затем встроенная поддержка на материнской плате, затем эмуляция при помощи страничной адресации
- Использовалась в XMS — сразу чисто программное решение (эмуляция), Intel 80286+

Адресное преобразование: устройства перехватывают обращение к ОЗУ на системной шине

## Расширение адресных регистров

- Старый машинный код не знает, что у адресного регистра есть старшая половинка
- Все переходы и обращения к данным в старом коде «короткие»
- Т.к. старые приложения не используют всю память, появляется внутренняя фрагментация

## Расширение адресных регистров

- Старый машинный код не знает, что у адресного регистра есть старшая половинка
- Все переходы и обращения к данным в старом коде «короткие»
- Т.к. старые приложения не используют всю память, появляется внутренняя фрагментация

### Пример:

- i8086 – i80286 – 16-битный регистр BP
- i80386 – i80586 – 32-битный регистр EBP (но младшие 16 бит доступны, как BP)
- x86\_64 – 64-битный регистр RBP (но доступны EBP и BP)

- Компилятор (точнее, компоновщик) генерирует программу, в бинарном файле которой обозначены данные разных видов — машинный код, статические данные (проинициализированные глобальные переменные) и т.д.
- Операционная система может размещать эти данные в физической памяти по своему усмотрению
- Получается, что нужен уровень косвенности, как при вызове API ядра через прерывания или спец. инструкции, только уже внутри программы.



# Реализация сегментной адресации (1): Жёсткие сегменты x86

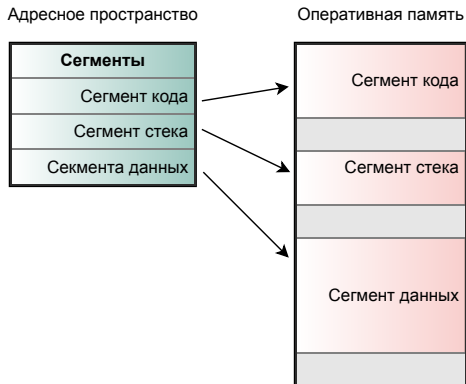
В 16-битном режиме x86 полный указатель состоит из *смещения* (адресный регистр) и *номера сегмента* (хранится в сегментном регистре). Пример для x86 (i8086):

- Адрес исполняемой инструкции CS : IP — Code Segment, Instruction Pointer
- Адрес вершины стека SS : SP — Stack Segment, Stack Pointer
- Адреса данных:
  - DS : SI, ES : SI, DS : BX и т.д., много сочетаний

В ранних x86 защиты не было, и программа могла работать с разными сегментами \*  
Обычно ОС (чаще всего DOS) выдавала ей по одному сегменту кода, стека и данных \*  
Если 64К для чего-то из этого было мало, можно было использовать другие модели памяти, работая с разными сегментами. [Подробнее здесь ↗](#)

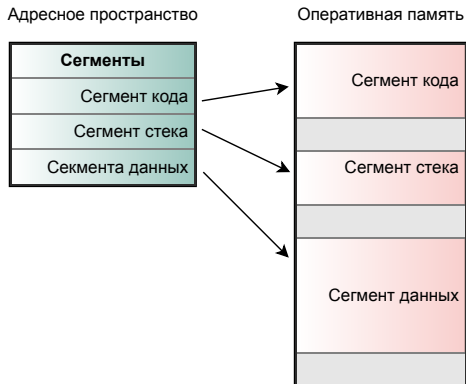
Адресное преобразование:  $A_{ph} = S \cdot 16 + O$

# Реализация: Управляемые сегменты



- Произвольного размера
- Требуют специальных таблиц размещения (таблицы дескрипторов)
- Могут ассоциироваться с правами на хранение данных и кода и выполнение кода

# Реализация: Управляемые сегменты



- Произвольного размера
- Требуют специальных таблиц размещения (таблицы дескрипторов)
- Могут ассоциироваться с правами на хранение данных и кода и выполнение кода

Если включена защита, процесс не может обращаться к чужим сегментам (как минимум должна «попросить разрешения» у ОС)

Адресное преобразование:  $A_{ph} = T_{desc}[S] + O$

## Виртуальная память и страничная адресация

- Реализация
- Вытеснение и замещение
- Пример: Intel i80386
- TODO: ARM и RISC-V

# Что такое виртуальная память

**Виртуальная память** — механизм работы с адресным пространством, превосходящим по размеру оперативную память или тот её фрагмент, который выделен данному процессу

Виртуальная память может по размеру превосходить физическую, и состоять из различных фрагментов, находящихся ОЗУ (по разным физическим адресам) или даже жесткого диска (за счёт чего и можно наращивать её размер). При этом адресное преобразование скрывает всю эту неоднородность от программы

## Задачи

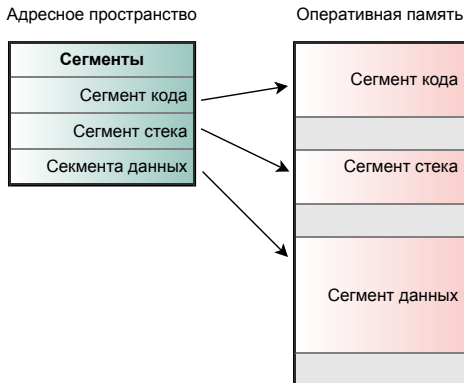
- Ранние ЭВМ — научные расчёты, обычно сравнительно небольшой код и объёмные данные. Данные можно обрабатывать порциями. ПО относительно недорого в масштабах системы
- Позже — большее количество программ, больший объём кода. Большая относительная стоимость ПО

## Решения

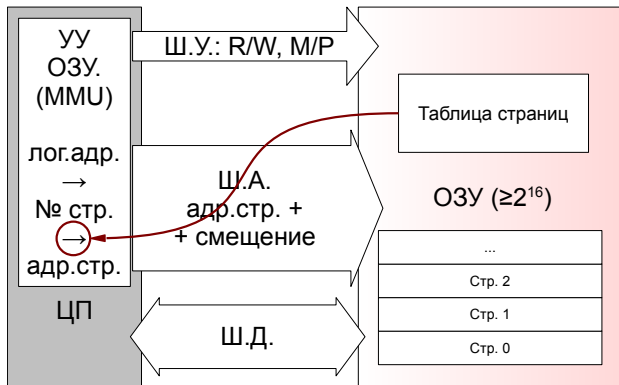
- Типичный способ работать с большим машинным кодом — загружать его кусками — *оверлеями* (overlay). Иногда использовался и для DOS на PC.
- В некоторых архитектурах часть адресного пространства проецировалась на диск (или барабан), чтобы облегчить программисту работы
- ...
- Сделать прозрачную для программиста систему

J. Fotheringham. Dynamic Storage Allocation in the Atlas Computer Including an Automatic Use of a Backing Store», Commun. of the ACM, vol. 4, pp. 435–436, 1961.

Идея: разделить адресное пространство и ОЗУ на одинаковые по размеру страницы адресного пространства и страницы ОЗУ.



# Страничная адресация: реализация



Адресное преобразование считывает таблицу страниц из ОЗУ и по номеру страницы АП находит номер страницы ОЗУ

Адресное преобразование (пример 80386):  $A_{ph} = T_{pages}[A_{log;12...31}] + A_{log;0...11}$



- Страницы небольшие и не пересекаются
- В отличие от жестких сегментов, часто могут «гулять» по физическим адресам под управлением ОС.
- Для управляемых страниц нужны таблицы размещения в основной памяти, к которым обращается ЦП  $\Rightarrow$  нужен хороший (и специально оптимизированный кэш)
- Удобны для реализации виртуальной памяти — их можно выгружать и загружать без ведома пользовательского процесса

- Изначально страницы процесса не проинициализированы
- По мере фактического обращения к памяти, страницам адресного пространства сопоставляются страницы памяти
- При нехватке памяти часть страниц процесса выгружаются на диск и помечаются, как выгруженные, освобождая физическую память для текущего и других процессов. Это называется *вытеснением*
- На освободившееся место проецируются данные новой виртуальной страницы или страницы, подгруженной из подкачки. Это называется *замещением*

Термины *вытеснение* и *замещение* обычно отождествляют, т.к. первое нужно для второго, и второе за первым обычно следует

- Изначально страницы процесса не проинициализированы
- По мере фактического обращения к памяти, страницам адресного пространства сопоставляются страницы памяти
- При нехватке памяти часть страниц процесса выгружаются на диск и помечаются, как выгруженные, освобождая физическую память для текущего и других процессов. Это называется *вытеснением*
- На освободившееся место проецируются данные новой виртуальной страницы или страницы, подгруженной из подкачки. Это называется *замещением*

Термины *вытеснение* и *замещение* обычно отождествляют, т.к. первое нужно для второго, и второе за первым обычно следует

А что происходит, когда процесс обращается к не проинициализированной или выгруженной странице?

- Изначально страницы процесса не проинициализированы
- По мере фактического обращения к памяти, страницам адресного пространства сопоставляются страницы памяти
- При нехватке памяти часть страниц процесса выгружаются на диск и помечаются, как выгруженные, освобождая физическую память для текущего и других процессов. Это называется *вытеснением*
- На освободившееся место проецируются данные новой виртуальной страницы или страницы, подгруженной из подкачки. Это называется *замещением*

Термины *вытеснение* и *замещение* обычно отождествляют, т.к. первое нужно для второго, и второе за первым обычно следует

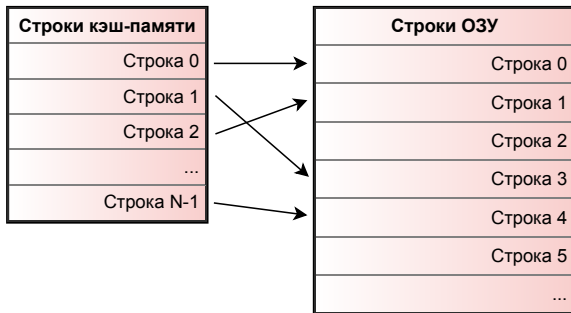
А что происходит, когда процесс обращается к не проинициализированной или выгруженной странице?


Процессор сам генерирует прерывание, обрабатывает которое драйвер виртуальной памяти

- Принцип Белادي (László Bélády) — вытеснить страницу, которая дольше всего не понадобится

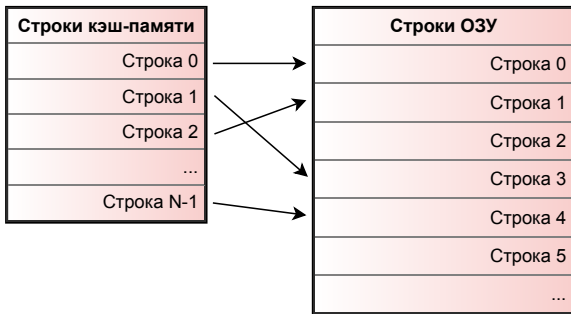
- Принцип Белادي (László Bélády) — вытеснить страницу, которая дольше всего не понадобится
- Поскольку предсказать этого нельзя, используются различные стратегии. Самая популярная — LRU (Least Recently Used) — вытеснить страницу, к которой дольше всего не обращались

## А что ещё вытесняется и замещается




**Кэш-память** — специальный вид памяти между процессором и оперативной памятью, в которой хранятся копии используемых в данный момент фрагментов оперативной памяти. Кэш-память делится на строки размером в несколько десятков или сотен байт. При обращении по адресу в памяти кэш-память проверяет наличие данных у себя и загружает их из ОЗУ, если они ещё не загружены. Отображение не произвольное, его гибкость задаётся **степенью ассоциативности кэша** 

## А что ещё вытесняется и замещается



**Кэш-память** — специальный вид памяти между процессором и оперативной памятью, в которой хранятся копии используемых в данный момент фрагментов оперативной памяти. Кэш-память делится на строки размером в несколько десятков или сотен байт. При обращении по адресу в памяти кэш-память проверяет наличие данных у себя и загружает их из ОЗУ, если они ещё не загружены.

Отображение не произвольное, его гибкость задаётся **степенью ассоциативности кэша** 

Физическую память можно рассматривать, как кэш виртуальной памяти.



- Выделить массив объёмом существенно больше, чем кэш процессора и «бегать» по нему случайно — вызывает кэш-промахи
- Выделить массив объёмом больше, чем ОЗУ и тоже бегать по нему случайно — вызывает «толкотню» виртуальной памяти

- Также, как и с сегментами, залезть в «чужие» страницы процесс не может. Он просто не может адресовать их!
- Если для страниц поддерживаются разрешения и привилегии, сегментами можно не пользоваться
  - Многие современные архитектуры поддерживают только страничную адресацию
  - От сегментной адресации хотели отказаться в 64-битном режиме x86\_64, но обратная совместимость потребовала её оставить

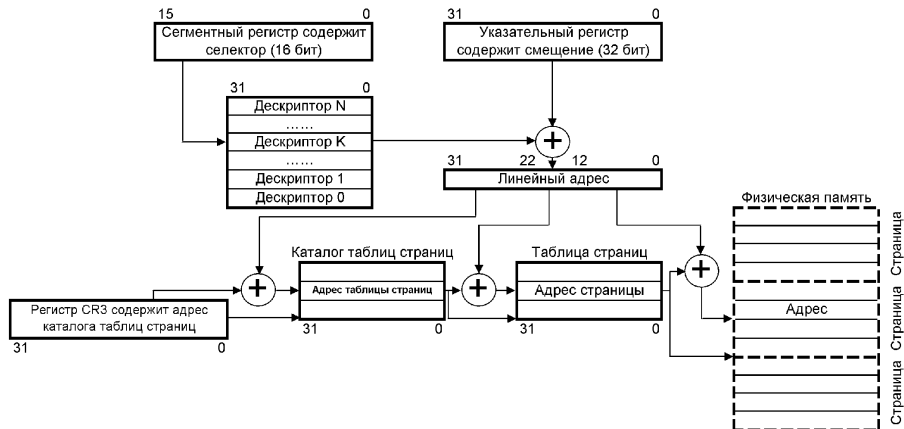
- Процесс получает сегмент (реже несколько) кода и сегмент (реже несколько) данных
- Процесс получает сегмент стека
- По прерыванию таймера происходит переход в другой сегмент кода (адрес полный обработчика в другом сегменте), при этом автоматом:
  - регистры сохраняются в стек (на то оно и прерывание)
  - меняются текущие сегменты стека и данных
- Сегмент логически до 2 Гиб — адресные регистры расширены, для работы со всем регистром новые инструкции

Адресные регистры по 32 бита вместо 16 (у 286 и ранее). В режиме DOS старшие разряды игнорируются

- Память разбита на страницы по 4 КиБ, в отдельных таблицах указано их размещение в основной памяти. Независимо от сегментов. Это дает возможности:
  - выделять физическую память по мере использования, даже если процесс сразу запросил много
  - не перемещать данные физически, если надо перенастроить сегменты процесса
- Маленькие одинаковые страницы позволяют эффективно реализовать виртуальную память больше физической — с подкачкой. при отсутствии нужной страницы в физической памяти она загружается с диска. Тоже через прерывание.

- Тормоза — сначала надо читать таблицы дескрипторов сегментов, потом страниц  
⇒ нужен умный кэш
- Обратная совместимость — процесс может пользоваться младшими 16 битами адресного регистра, как и раньше
- Старым процессам, рассчитанным на жесткие сегменты, выделяются сегменты, пересекающиеся, как раньше (реальный режим)
- Про страницы пользовательский процесс не знает вообще ничего
- По прерываниям таймера переход идет на диспетчер процессов, а он уже дает переход на следующий процесс, выбирая его исходя из приоритета
- По прерываниям отсутствующих страниц переход идет на диспетчер виртуальной памяти

- Виртуальная память позволяет эмулировать устройства, перехватывая отсутствие страниц, адрес которых задан в несуществующей физической памяти
  - Например, банковые расширения памяти — EMS, например — на месте окна EMS — страницы, физически расположенные в несуществующей памяти
  - Плоские адресные буферы внешних устройств, например, **графических адаптеров, в действительности не поддерживающих их** ↗
- 386 адресует до 4 Гиб физической и до 64 Тиб виртуальной, при этом виртуальная м.б. тоже реализована, как физическая, хоть и не адресуется на прямую



Адресное преобразование 80386

На иллюстрации опечатка, дескрипторы сегментов по 8 байт, а не по 4

- Устройства располагают свою память в 4-м ГиБ, так что, если установлено 4 ГиБ, часть памяти перекроется (будет недоступно).
- 4 ГиБ может быть тоже мало.

Проблема решается при помощи Physical Address Extension — доп. расширения, появившегося в Pentium Pro (на тот момент — для серверов). Расширение позволяет страницам памяти располагаться не в контексте 4 ГиБ ОЗУ, а в контексте 64 ТиБ ОЗУ, по объему совпадающим с виртуальной памятью. При этом изменяется формат дескрипторов сегментов и страниц.

Пользовательские процессы обычно не работают с памятью такого объема, для них всё выглядит по-старому. Картина меняется для ядра. Программы, которым объем памяти критичен, можно пересобрать со специальными библиотеками.



# Вопросы и упражнения

## Вопросы

- Программа выделила большой объём памяти, но не обращалась к ней. Сколько физической памяти было выделено?
- Опишите механизм сегментного преобразования
- Что такое адресное пространство, виртуальный и физический адреса?
- Что такое виртуальная память?
- Опишите механизм страничного преобразования
- Что такое замещение?
- Каким образом сегментная и страничная адресация позволяют изолировать программы в многозадачном режиме?
- Что такое Physical Address Extension?

## Упражнения

- Напишите программу, выделяющую много (больше размера Вашего ОЗУ) памяти, но не обращающуюся к ней; наблюдайте над расходом физической памяти
- Допишите программу выше, чтобы она интенсивно работала с памятью; опишите наблюдаемый эффект
- Повторите подобный эксперимент в меньших масштабах, измерьте, во сколько раз эффективная работа кэш-памяти ускоряет обращение к ОЗУ

# Вопросы



[EDU.DLUCIV.NAME](#) ↗