Безпека операційних систем і комп'ютерних мереж

Лекція 11

Апаратні засоби

Завдання апаратного забезпечення засобів захисту

- За усталеною термінологією склад АЗЗЗ визначається не за ознакою апаратної реалізації, а за колом завдань, що вирішуються:
  - підтримка керування пам’яттю;
  - підтримка керування процесами (завданнями);
  - підтримка взаємодії між процесами.
- АЗЗЗ не забезпечують виконання завдання повністю, а лише надають необхідну підтримку
- Як правило, АЗЗЗ виконують такі функції, як доступ до об’єктів, перевірки, переключення, а реалізація складних алгоритмів, які здійснюють планування таких функцій, виноситься за межі апаратних засобів

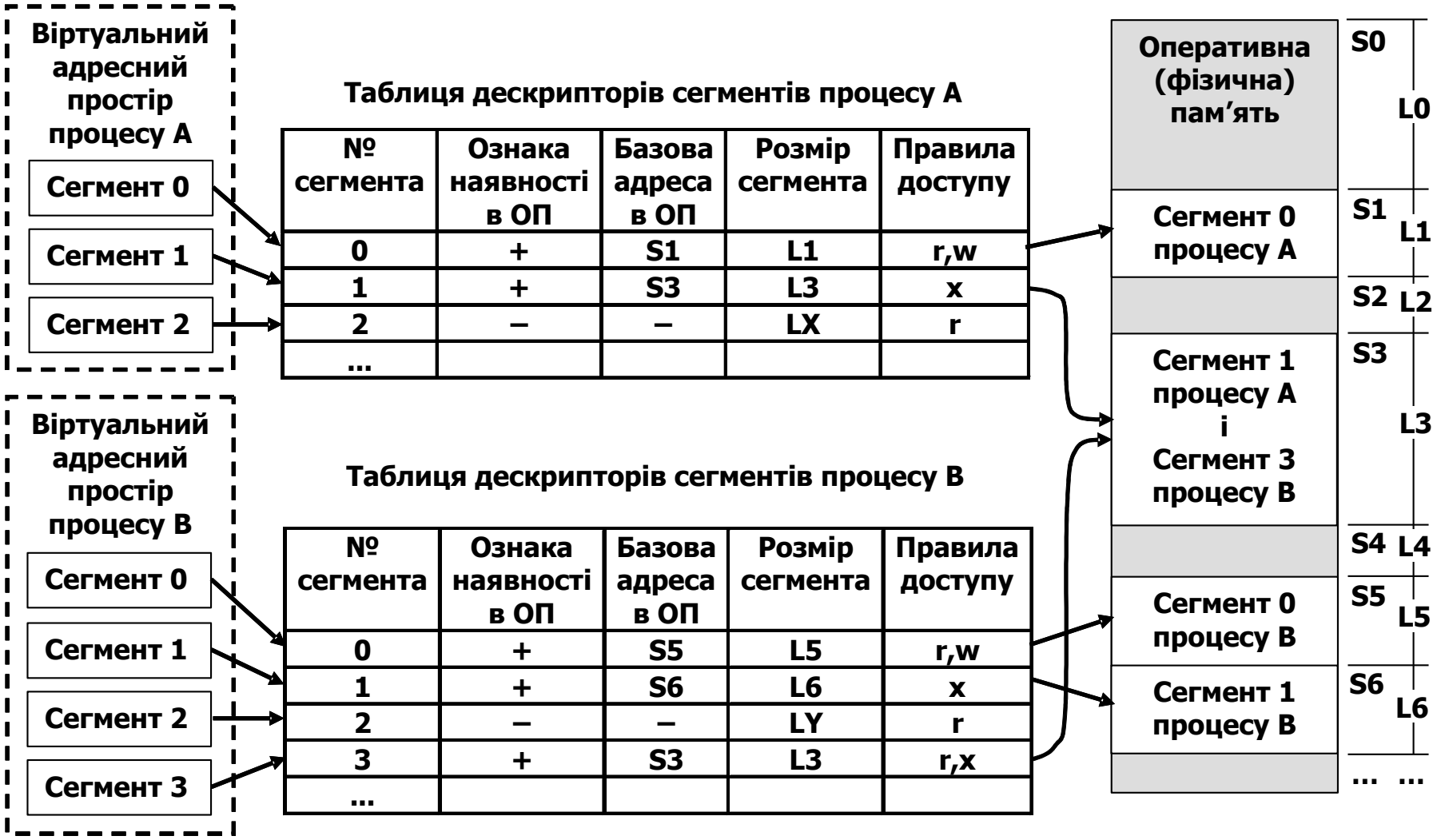
Підтримка керування пам’яттю

- Основними завданнями розподілу пам’яті є:
  - відстеження вільної і зайнятої пам’яті, виділення пам’яті процесам під час їх запуску і в процесі роботи, звільнення пам’яті, дефрагментація пам’яті;
  - трансляція адрес, що використовуються в програмах;
  - організація віртуальної пам’яті;
  - розмежування доступу процесів до окремих областей пам’яті як з метою ізоляції адресних просторів процесів від доступу інших процесів, так і з метою організації контрольованого спільного доступу процесів до виділених областей пам’яті.
- Лише четверте з зазначених завдань безпосередньо пов'язано з захистом інформації

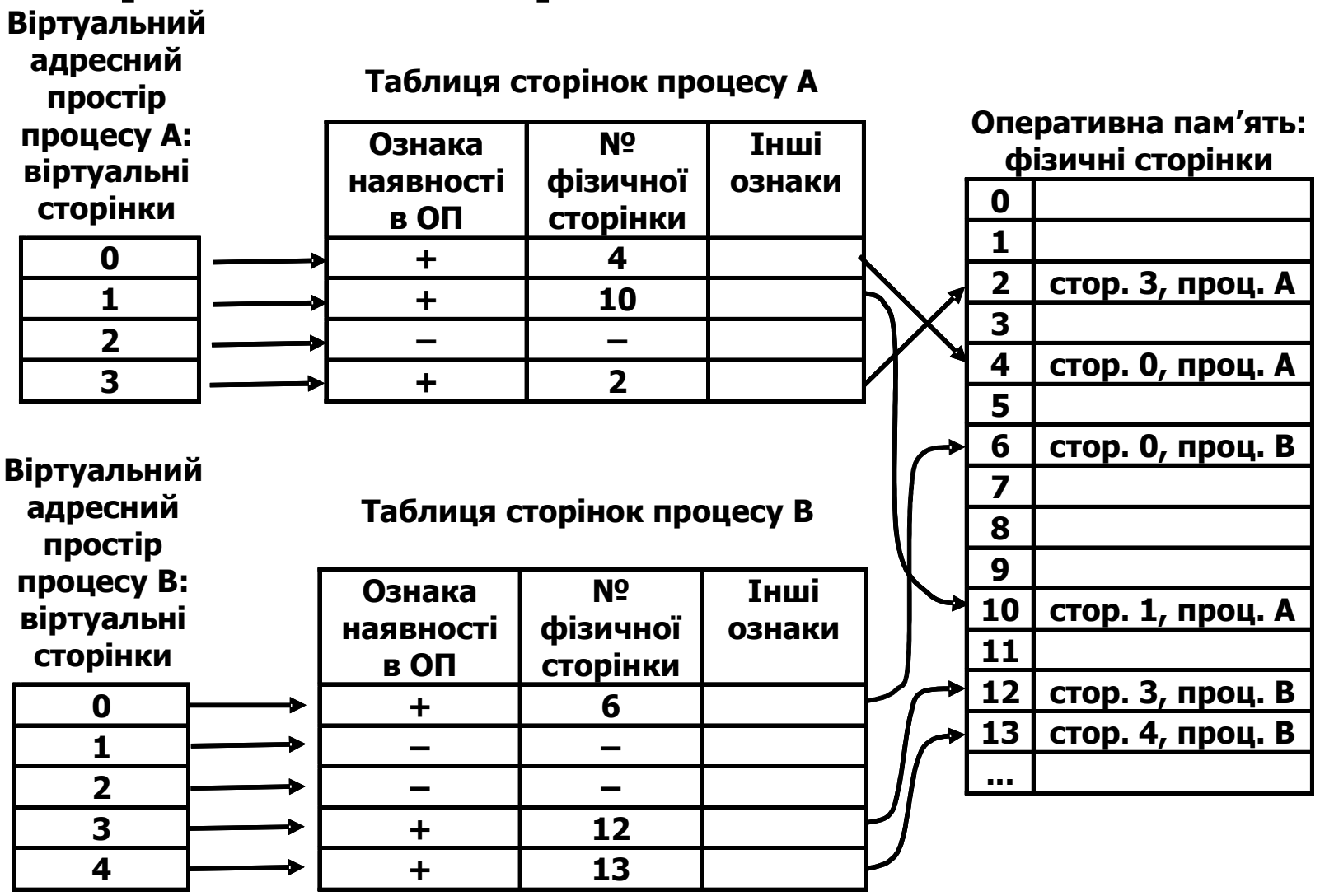
Віртуальні адреси

- Віртуальні адреси дозволяють забезпечити коректну адресацію незалежно від розташування програми в оперативній пам’яті комп’ютера
  - Для цього використовуються відносні адреси, тобто *зміщення* від деякої *базової адреси*
- Моделі пам’яті:
  - *Пласка* модель – кожному процесу виділяється єдина неперервна послідовність віртуальних адрес
    - Зміщення дозволяє однозначно вказати на положення даних або команди в адресному просторі даного процесу
  - *Сегментна* модель – адресний простір процесу поділяється на окремі частини – *сегменти* (*секції*, *області*)
    - Віртуальна адреса задається парою чисел (*n*, *m*), де *n* визначає сегмент, а *m* – зміщення в даному сегменті

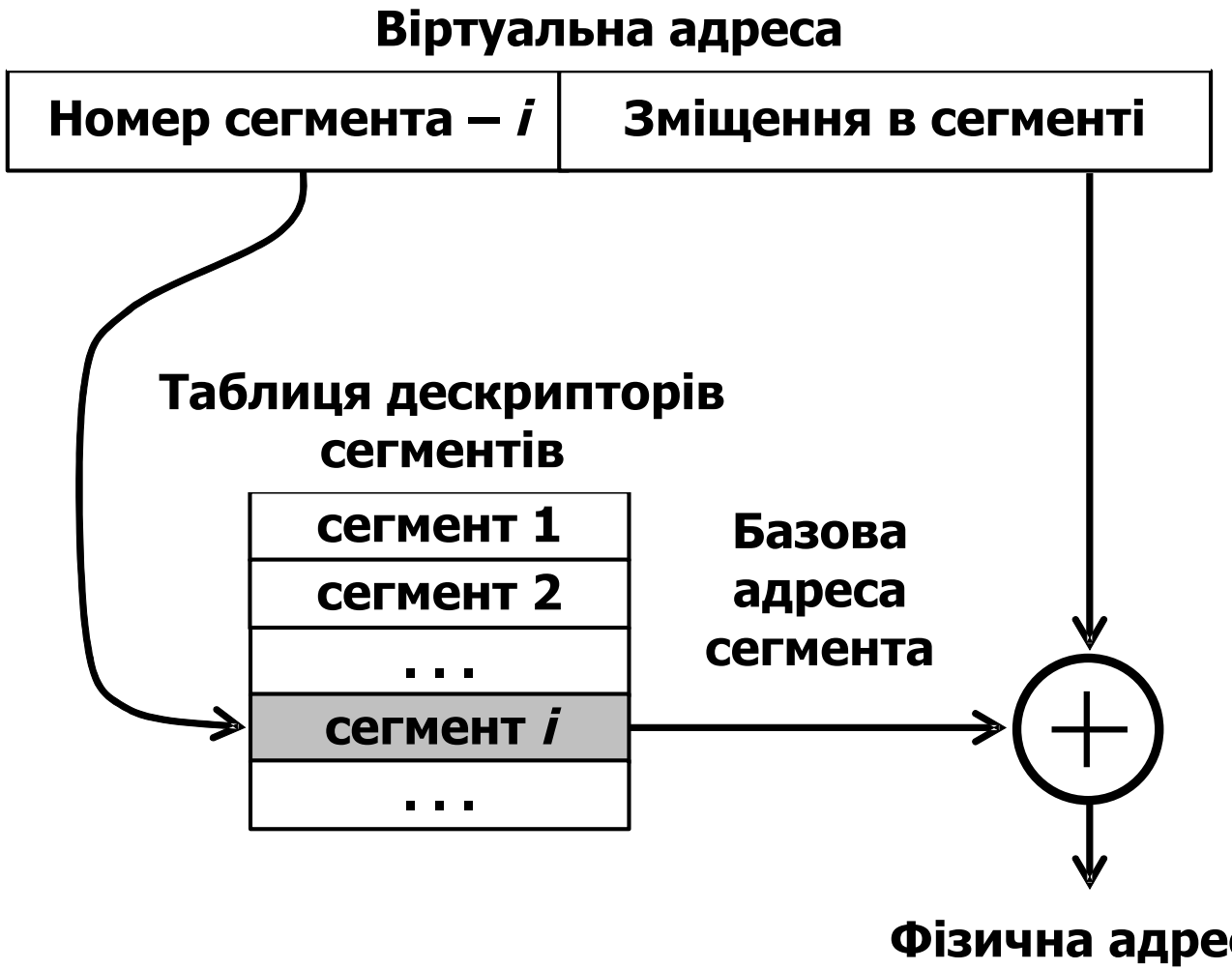
Сегментний розподіл пам’яті



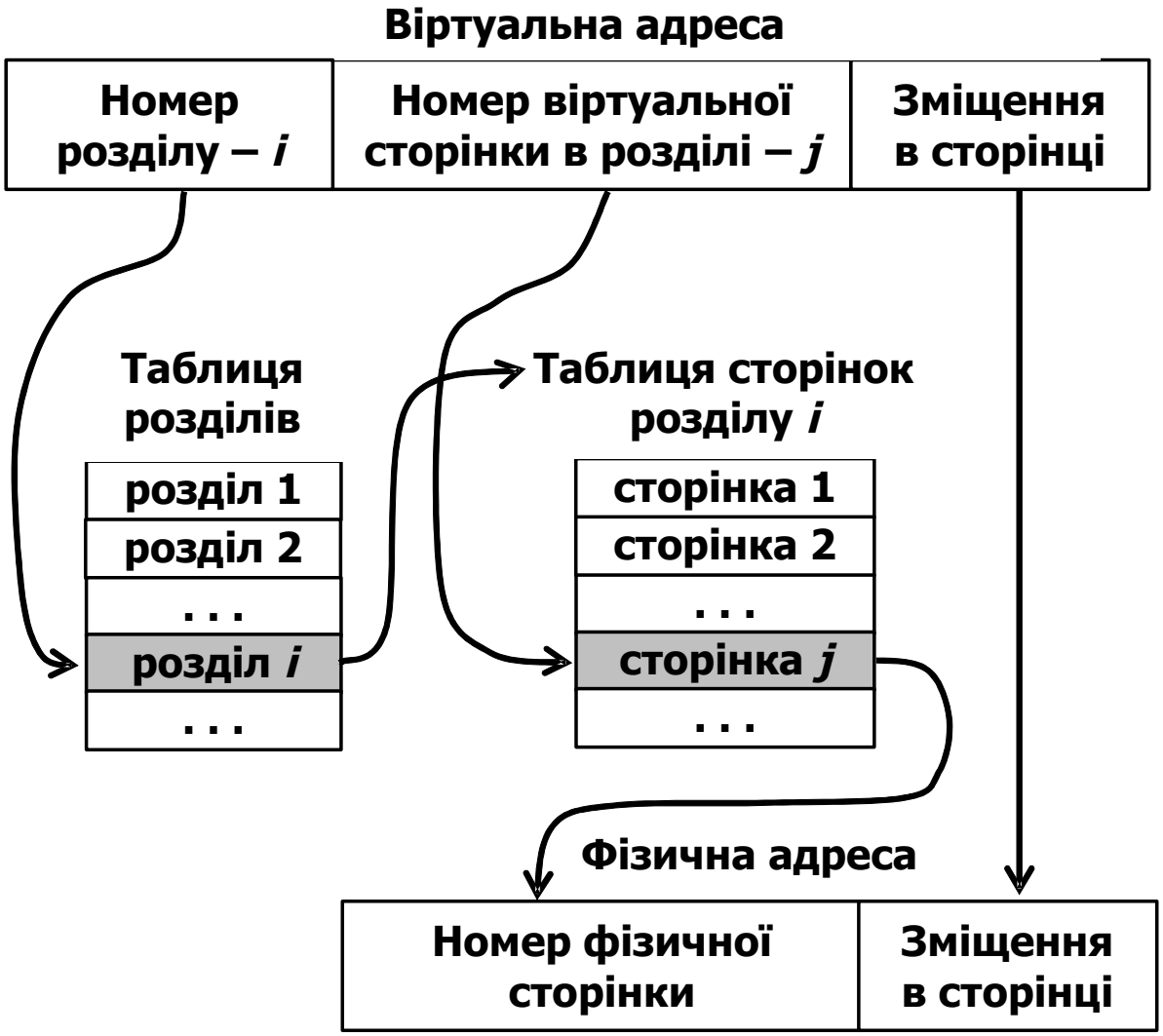
Сторінковий розподіл пам’яті



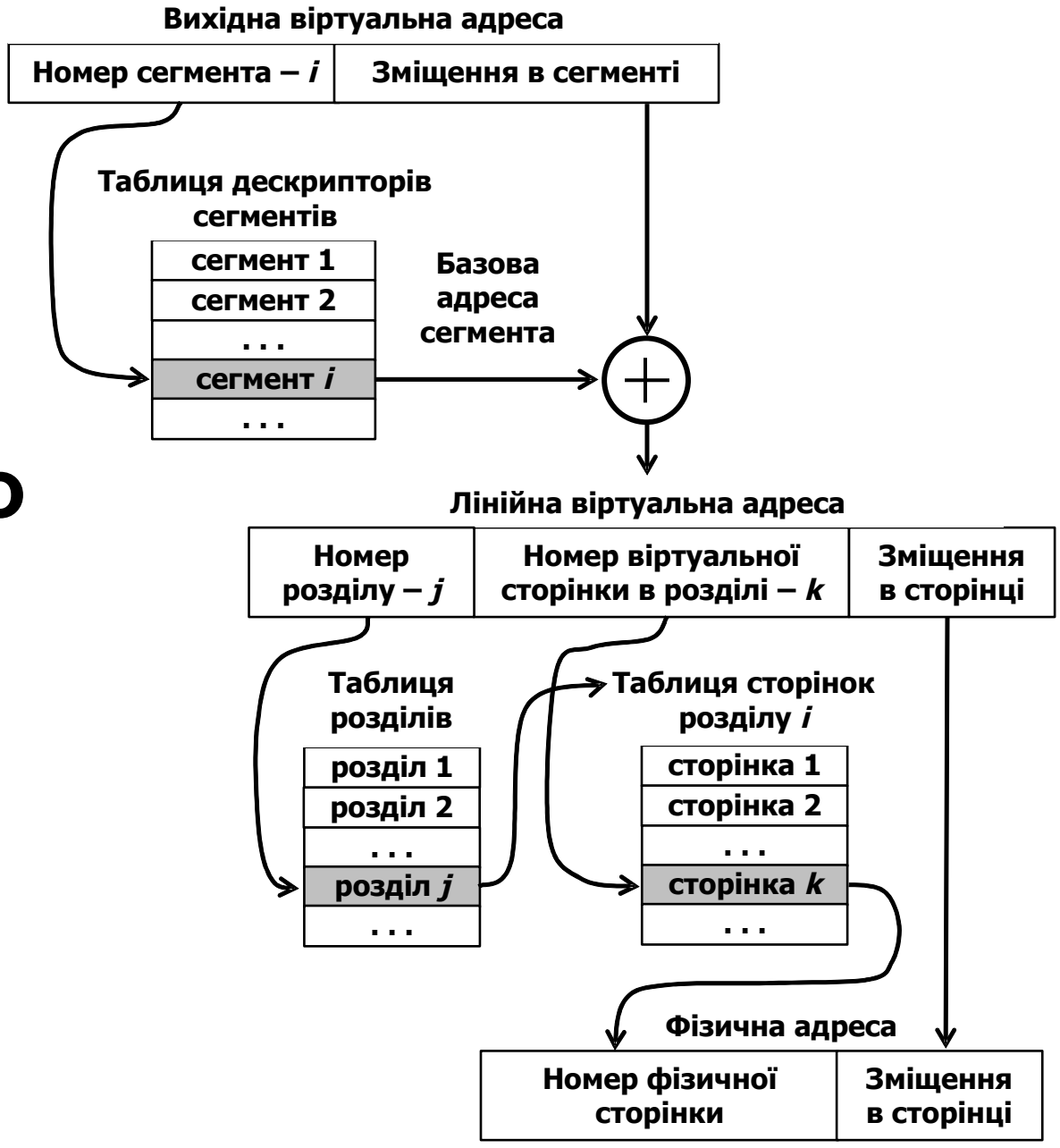
Трансляція віртуальної адреси за сегментної організації пам’яті



Трансляція віртуальної адреси за дворівневої сторінкової організації пам'яті



Трансляція віртуальної адреси за сегментно-сторінкового розподілу пам’яті



Регістри сегментів процесорів x86

Позначення	Назва	Особливості використання
cs	сегмент коду (*Code Segment register*)	**адресує сегмент коду, який виконується процесором** Для переходу в інший сегмент коду необхідно завантажити нове значення в регістр cs, причому явних команд для цього не існує: це здійснюється автоматично при виконанні процесором команд jmp або call.
ss	сегмент стека (*Stack Segment register*)	**адресує сегмент стека, з яким в даний момент працює процесор** Для організації іншого стека необхідно завантажити в ss нове значення, при цьому необхідно зберегти поточне значення ss, щоби мати змогу повернутись назад.
ds	сегмент даних (*Data Segment register*)	**адресує дані в оперативній пам'яті, з якими працює процесор** Сегмент ds є основним, і адресується неявно.
es, gs, fs	додаткові сегменти даних (*Extension Data Segment register*)	**адресують дані в оперативній пам'яті, з якими працює процесор** Ці сегменти даних вимагають явної адресації за допомогою спеціальних префіксів в командах.

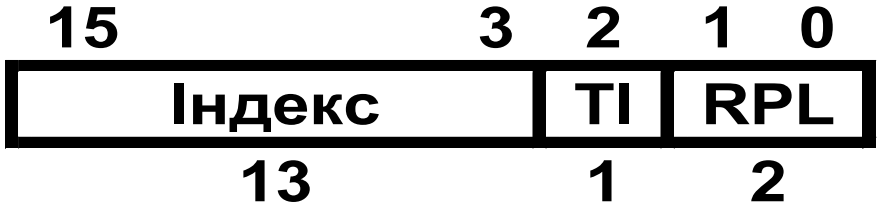
Регістри системних адрес процесорів x86

Позначення	Назва	Особливості використання
gdtr	регістр глобальної таблиці дескрипторів (*Global Descriptor Table Register*)	має 48 розрядів * 32 старших розряди представляють лінійну базову адресу глобальної таблиці дескрипторів у пам'яті * 16 молодших розрядів задають розмір таблиці
ldtr	регістр локальної таблиці дескрипторів (*Local Descriptor Table Register*)	16-розрядний регістр, що містить селектор, який вказує на дескриптор в глобальній таблиці, який описує спеціальний сегмент – локальну таблицю дескрипторів процесу, який виконується в даний момент
idtr	регістр таблиці дескрипторів переривань (*Interrupt Descriptor Table Register*)	48-розрядний регістр, за будовою аналогічний регістру gdtr
tr	регістр задачі (*Task Register*)	16-розрядний регістр, що містить селектор, який вказує на дескриптор в глобальній таблиці, який описує спеціальний сегмент – сегмент стану завдання (TSS), який містить контекст поточного процесу

Регістри керування процесорів x86

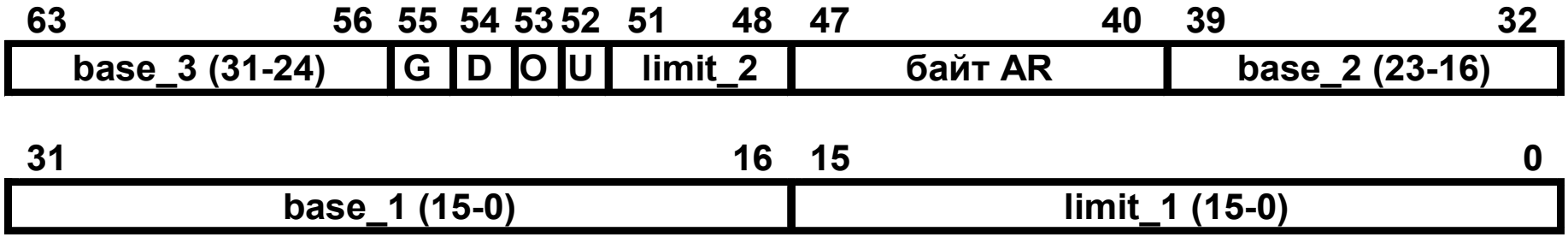
cr0	**містить прапорці, які суттєво впливають на роботу процесора і відображають глобальні (незалежні від конкретної задачі) ознаки його функціонування** Деякі важливі системні прапорці з цього регістру: * **pe** (Protect Enable), біт 0 – вмикає захищений режим роботи процесора; * **cd** (Cache Disable), біт 30 – вмикає використання внутрішньої кеш-пам'яті (кеш першого рівня); * **pg** (Paging), біт 31 – вмикає сторінкову трансляцію адрес.
cr2	**призначений для роботи сторінкового механізму віртуальної пам'яті** Містить лінійну віртуальну адресу команди, яка викликала виняткову ситуацію 14 – відсутність сторінки у пам'яті Обробник цієї виняткової ситуації після завантаження необхідної сторінки у пам'ять має змогу відновити нормальну роботу програми, передавши керування на адресу з cr2
cr3	**призначений для роботи сторінкового механізму віртуальної пам'яті** Містить фізичну базову адресу каталогу сторінок
cr4	**містить прапорці-ознаки підтримки різних архітектурних елементів, що впроваджувались у різних моделях процесорів** Наприклад, підтримка 36-розрядної адресації, підтримка 4-мегабайтних сторінок тощо

Селектор сегмента



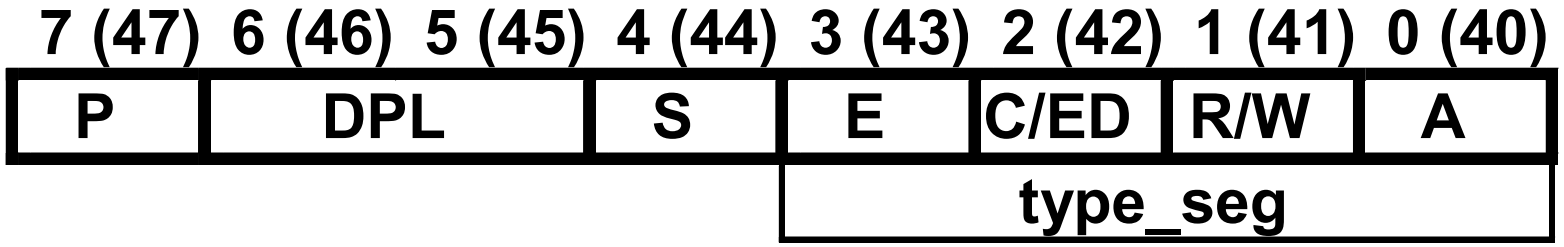
- Селектор – це 16-розрядна структура, яка завантажується в сегментні регістри
- Селектор адресує не сам сегмент, а його дескриптор.
- В кожний момент часу процесору доступні дві таблиці дескрипторів:
  - глобальна (GDT), яка є спільною для всіх процесів
  - локальна (LDT), яка є власною таблицею для поточного процесу
- 13 старших розрядів селектора є індексом в таблиці дескрипторів
  - Таким чином, кожна таблиця може містити 213=8192 дескрипторів
- Один розряд селектора (біт 2), який позначається як прапорець TI, вказує в якій з таблиць знаходиться дескриптор:
  - TI=0 → GDT
  - TI=1 → LDT
- Останні 2 розряди селектора (біти 1,0) відведені для задавання рівня привілеїв (*Requested Privilege Level,* RPL)

Дескриптор сегмента



- Дескриптор сегмента є 8-байтовою структурою
- Головні поля дескриптора:
  - 32-розрядна базова адресу (*base*)
  - 20-розрядна межа сегмента (*limit*)
  - Байт захисту (*AR*)

Байт захисту

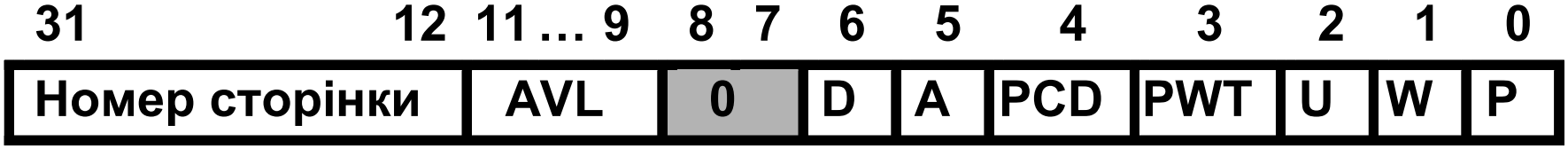


- Чотири молодших розряди байта захисту прийнято називати полем типу сегмента (*type_seg*)
  - насправді тип сегмента задається розрядами не 0...3, а 1...4
  - Розряд 0 встановлюється при доступі до сегмента і може використовуватись операційною системою при реалізації алгоритмів керування віртуальною пам'яттю
  - Розряд 4 визначає, чи є об'єкт, який описує цей дескриптор, сегментом у пам'яті чи спеціальним системним об'єктом
  - Розряди 1...3 визначають, власне, тип сегмента і права доступу до нього
- Два розряди байта захисту дескриптора – розряди 5 і 6 – задають рівень привілеїв дескриптора (*DPL – Descriptor Privilege Level*)
  - Разом з рівнем привілеїв селектора RPL і поточним рівнем привілеїв процесу, що виконується (*CPL – Current Privilege Level*), ці рівні дозволяють організувати розмежування доступу до сегментів за мандатним принципом – кільця захисту

Значення поля типу сегмента

Біт S	Комбінація бітів у полі type_seg	Тип сегмента
0	0100	Таблиця локальних дескрипторів (LDT)
0	0001 1000 1101 1101	Сегмент стану завдання (TSS)
1	000x	Сегмент даних, тільки для зчитування
1	001x	Сегмент даних, дозволені зчитування і записування
1	010x	Не визначено
1	011x	Сегмент стека, дозволені зчитування і записування
1	100x	Сегмент коду, дозволено тільки виконання
1	101x	Сегмент коду, дозволені зчитування і виконання
1	110x	Підпорядкований сегмент коду, дозволено тільки виконання
1	111x	Підпорядкований сегмент коду, дозволені зчитування і виконання

Дескриптор сторінки



- Віртуальна лінійна адреса є 32-розрядною
  - старші 20 розрядів інтерпретуються як номер віртуальної сторінки (індекс дескриптора сторінки)
    - З дескриптора визначається номер сторінки у фізичній пам’яті
  - молодші 12 розрядів інтерпретуються як зміщення в сторінці
    - Зміщення у віртуальній і фізичній сторінках співпадають

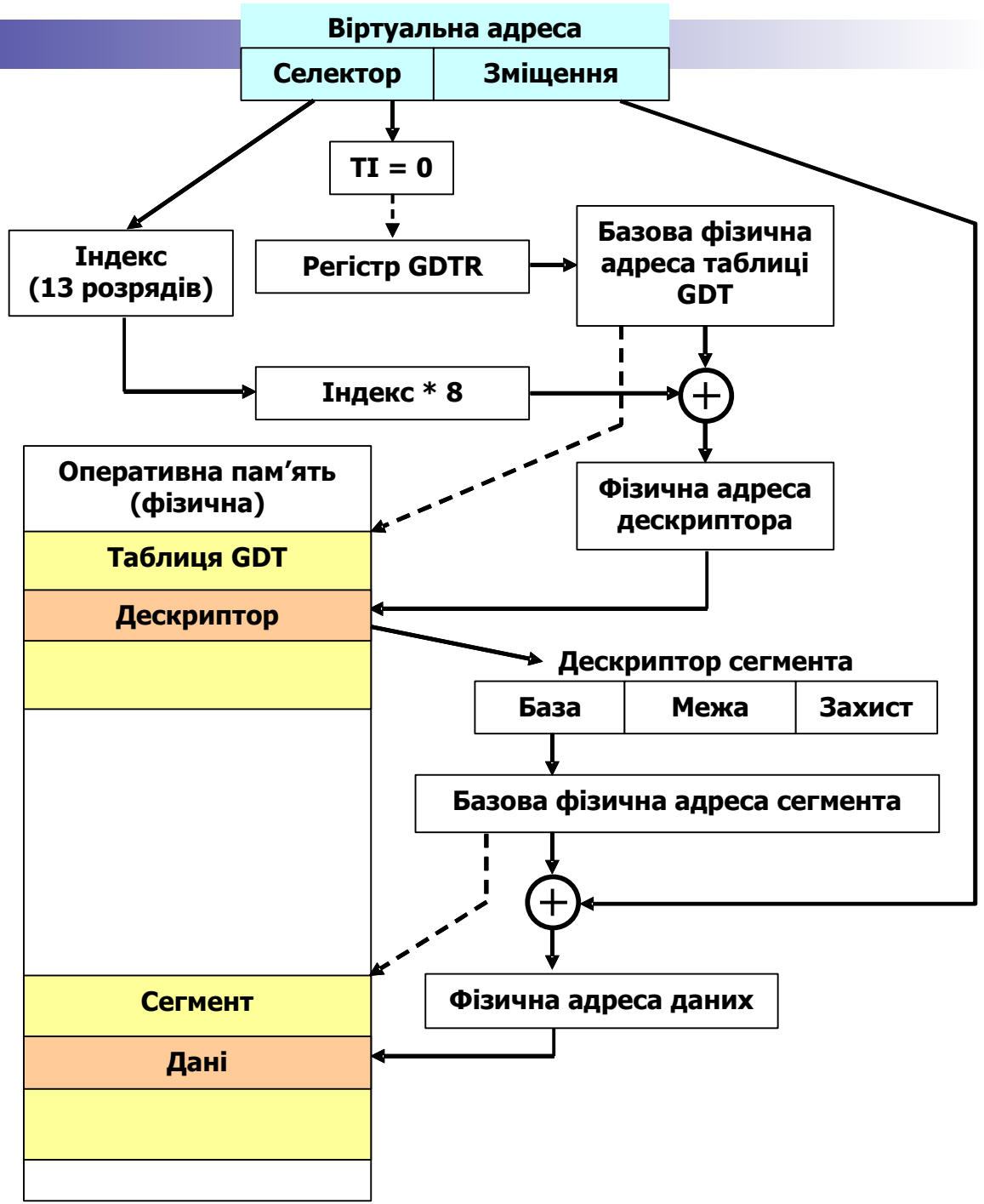
Значення полів дескриптора сторінки

Номер біта	Символічне позначення	Призначення і вміст
0	P (Present)	Прапорець присутності сторінки у фізичній пам’яті.
1	W (Writable)	Прапорець дозволу записування у сторінку
2	U (User mode)	Прапорець користувач/супервізор
3	PWT	Керують механізмом кешування сторінок (введені, починаючи з процесора i486)
4	PCD	
5	A (Accessed)	Ознака, що мав місце доступ до сторінки
6	D	Ознака модифікації вмісту сторінки
7…8	0	Зарезервовані
9…11	AVL (Available)	Зарезервовані для потреб операційної системи
12...31		Номер сторінки у пам’яті

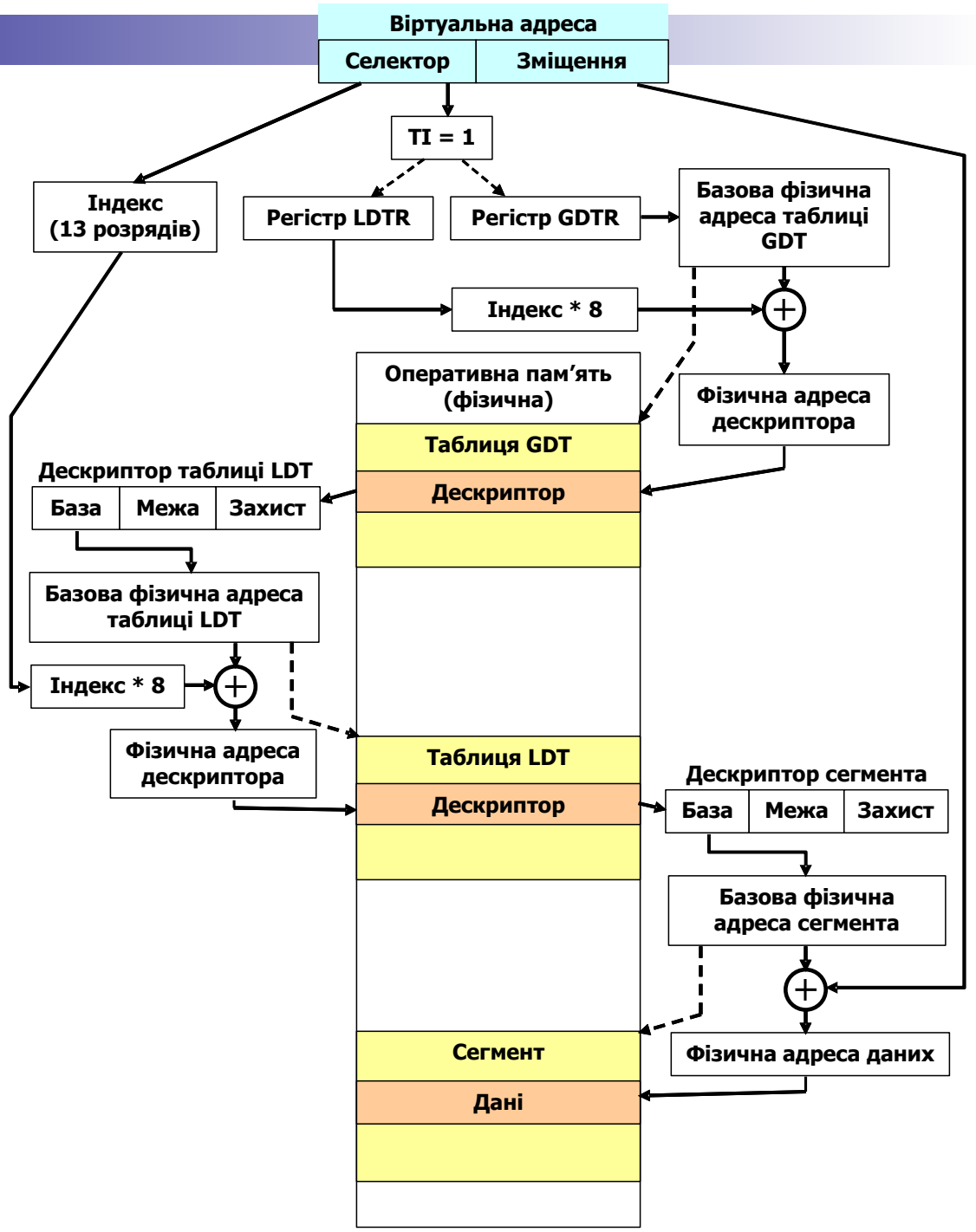
Завантаження селектора в сегментний регістр – пошук дескриптора

- Якщо у селекторі прапорець TI = 0, то дескриптор міститься у GDT
  - Базова адреса і межа (розмір) таблиці GDT визначаються з регістру **gdtr**
  - Для вибору потрібного дескриптора використовується індекс, що міститься в селекторі
  - Перевіряється, чи не виходить дескриптор за встановлену межу таблиці GDT
  - Якщо ні – відбувається звернення до дескриптора
  - В разі виходу за межу, генерується внутрішнє переривання процесора – виняткова ситуація 11
- Якщо у селекторі прапорець TI = 1, то дескриптор міститься в таблиці LDT
  - Першим кроком є пошук таблиці LDT, для чого в якості селектора використовується вміст регістру **ldtr**
  - З таблиці GDT вибирається дескриптор, що описує таблицю LDT
  - Перевіряється
    - чи відповідає тип дескриптора таблиці дескрипторів
    - чи присутня таблиця у фізичній пам’яті (біт P байта AR дескриптора дорівнює 1
  - З дескриптора таблиці LDT визначаються базова адреса таблиці та її межа
  - Здійснюється операція вибору з таблиці LDT необхідного дескриптора, яка аналогічна операції вибору дескриптора з таблиці GDT, з тими ж перевірками

Механізм перетворення віртуальної адресу у фізичну при роботі процесора x86 в сегментному режимі, дескриптор сегмента знаходиться у таблиці GDT



Механізм перетворення віртуальної адресу у фізичну при роботі процесора x86 в сегментному режимі, дескриптор сегмента знаходиться у таблиці LDT



Завантаження селектора в сегментний регістр – перевірка сумісності типів

- За звернення до дескриптора здійснюється перевірка сумісності типів селектора і дескриптора
- Дескриптор повинен описувати сегмент у пам'яті (біт S = 1)
- Якщо селектор завантажується в регістр, що вказує на сегмент даних, то несумісним буде дескриптор, що описує сегмент коду із захистом від читання (біт E = 1, біт R = 0)
- Якщо селектор завантажується в регістр cs, то сумісним типом буде лише сегмент коду
- Якщо селектор завантажується в регістр ss, то сумісним типом буде лише сегмент стека
  - Очевидно, що більш жорсткі умови сумісності для сегментів стека і коду спрямовані на запобігання "підсовування" процесору зловмисного коду через сегменти даних
- Якщо перевірка дає негативний результат, фіксується несумісність типів – генерується виняткова ситуація 13 (загальна помилка захисту)

Дозволені комбінації бітів байту захисту дескриптора при виконанні операції завантаження селектора в сегментний регістр

Регістр	P	DPL	S	E	C/ED	R/W	A	Примітка
cs	x	x x	1	1	x	x	x	сегмент коду
ss	x	x x	1	0	1	1	x	сегмент стека
ds, es, fs, gs	x	x x	1	1	x	1	x	сегмент коду
				0	x	x	x	сегмент даних або стека

Завантаження селектора в сегментний регістр – перевірка привілеїв доступу

- Після перевірки сумісності типів здійснюється перевірка привілеїв доступу до сегмента
  - Для цього порівнюються значення
    - RPL — рівня привілеїв селектора, який ми завантажуємо в сегментний регістр процесора,
    - DPL — рівня привілеїв дескриптора, на який посилається селектор, що ми завантажуємо, і
    - CPL (Current Privilege Level) — рівня привілеїв процесу (потоку), що виконується, який дорівнює RPL селектора, що знаходиться в регістрі cs
  - Для сегмента стека в разі спроби завантаження селектора в регістр ss повинна виконуватись рівність DPL = RPL = CPL
  - Для всіх інших сегментів рівень DPL повинен бути не вищим за рівень RPL і CPL, тобто DPL ≥ RPL і DPL ≥ CPL
    - найвищому рівню привілеїв відповідає значення 0, а найнижчому – 3
  - Якщо потрібне співвідношення не виконується, фіксується недостатній рівень привілеїв (виняткова ситуація 13 – загальна помилка захисту).

Приклад 1. Завантаження селектора в сегментний регістр (1/2)

Виконується команда

mov ds, ax

Регістр **ax** містить значення **Ox37 = 0000000000110111**

Процес виконується у нульовому кільці захисту, тобто **CPL = 0**

- Аналізується біт TI селектора
  TI = 1 (тобто, дескриптор знаходиться в LDT)
- Обчислюється фізична адреса дескриптора **dt** локальної таблиці дескрипторів LDT, що відповідає поточному процесу (дескриптор **dt** знаходиться в таблиці GDT)
- З дескриптора **dt** добувається межа LDT **ldt_limit**
- Якщо в **dt** G=1, то **ldt_limit *= 4096** (межа у 4кБ сторінках)
- Із селектора добувається індекс дескриптора
  index = 6

Приклад 1. Завантаження селектора в сегментний регістр (2/2)

- Перевіряється, чи не вказує індекс за межі таблиці LDT
  (index + 1) * 8 – 1 = 55
  Якщо **ldt_limit < 55**, фіксується некоректність селектора (помилка 11)
- З таблиці LDT добувається дескриптор d сегмента
  offset = index * 8 = 48;
  d_addr = ldt_base + offset
- Якщо в d біт S = 0 або одночасно E = 1 і R = 0, фіксується несумісність типів селектора і регістра (помилка 13)
- Із селектора добувається RPL
  RPL = 3
- Порівнюються CPL, RPL і DPL
  CPL = 0, RPL = 3
  Якщо DPL < 3, фіксується недостатній рівень привілей (помилка 13)
- Селектор з регістру **ax** завантажується в регістр **ds**, одночасно дескриптор **d** завантажується в дескрипторний регістр, що відповідає регістру **ds**

Звернення до пам’яті

- В процесі звернення до пам’яті за сегментного розподілу перевіряється дозвіл на операцію і коректність доступу
  - Для команди зчитування з пам’яті обмеження можуть бути встановлені лише для сегментів коду (біт E = 1): якщо біт R = 0, то зчитування заборонено
  - Для команди записування в пам’ять сегмент коду взагалі є несумісним типом, а всі інші типи сегментів можуть бути захищені бітом W: якщо W = 0, записування заборонено
  - В усіх зазначених випадках генерується виняткова ситуація 13 – загальна помилка захисту
- Перевірка коректності доступу полягає у перевірці відсутності виходу за межу сегмента
  - Перевірка здійснюється з урахуванням розміру даних і напряму зростання сегмента
  - Якщо сегмент є сегментом стека (біт E = 0 та біт ED = 1), то він зростає в бік молодших адрес, і тоді обчислена адреса повинна бути не меншою за межу сегмента
  - Якщо ж сегмент є сегментом коду (біт E = 1) або даних (біт E = 0 та біт ED = 0), то він зростає в бік старших адрес, тому до обчисленої адреси додається розмір даних (для команди mov – в залежності від того, який з регістрів процесора бере участь у передаванні/прийманні даних), і отримана адреса повинна бути не більшою за межу сегмента
  - В разі помилки генерується виняткова ситуація 13 – загальна помилка захисту

Дозволені комбінації бітів байту захисту при виконанні операцій звернення до пам’яті

Операція	P	DPL	S	E	C/ED	R/W	A	Примітка
Зчитування з пам’яті	1	x x	1	0	x	x	x	сегмент даних або стека
	1	x x	1	1	x	1	x	сегмент коду
Записування я у пам’ять	1	x x	1	0	x	1	x	сегмент даних або стека

Приклад 2
Звернення до пам’яті (1/2)

Виконується команда

mov es:[ebx+4],eax

1. Перевіряється біт E дескриптора **d**, що завантажений в дескрипторний регістр, який відповідає сегментному регістру **es**
   Якщо E = 1, фіксується спроба запису у сегмент коду (помилка 13)
2. Перевіряється біт W дескриптора **d**
   Якщо W = 1, фіксується спроба запису у сегмент, захищений від запису (помилка 13)
3. Із сегмента **d** добувається межа сегмента **seg_limit**.
4. Якщо в **d** G=1, то
   seg_limit *= 4096 (межа у 4кБ сторінках)
5. Обчислюється зміщення:
   offset = ebx + 4

Приклад 2
Звернення до пам’яті (2/2)

6. Перевіряється відсутність виходу за межу сегмента:
   - **Якщо ED = 0** (сегмент даних, росте в бік старших адрес), то
     - порівнюється **offset + data_size – 1 = offset + 3** і **seg_limit** (зміщення останнього байта даних, розмір даних – 4 байта)
       - Якщо **offset + 3 > seg_limit**, то фіксується некоректне звернення (помилка 13)
   - **інакше якщо ED = 1** (сегмент стека, росте в бік молодших адрес), то
     - порівнюється **offset** і **seg_limit**,
       - Якщо **offset < seg_limit**, то фіксується некоректне звернення (помилка 13)
7. З дескриптора **d** добувається **seg_baseї**
8. Обчислюється адреса **seg_base + offset**, і в пам’ять за цією адресою заносяться дані з регістру **eax**