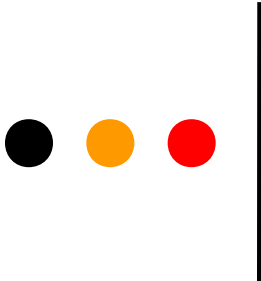




Операційні системи

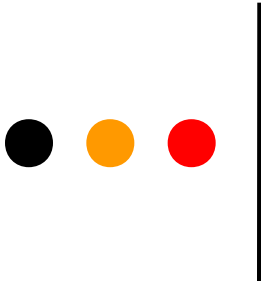
Додаткова лекція

Довідковий матеріал з курсу
“Архітектура комп’ютерних систем”



План лекції

- Керування пам'яттю у процесорах архітектури x86
 - Системні таблиці і регістри системних адрес
 - Селектор і дескриптор сегмента
 - Захист сегментів
 - Завантаження селектора у сегментний регістр
 - Звернення до пам'яті
 - Сторінковий механізм керування пам'яттю
- Керування завданнями у процесорах Intel x86
 - Виклик процедур
 - Виклик завдань
 - Привілейовані та чутливі команди



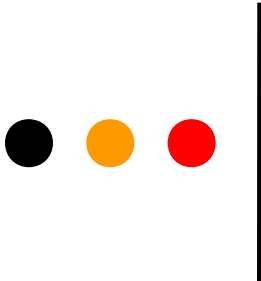
Керування пам'яттю у процесорах архітектури x86

- У захищеному режимі (32-розрядному) підтримується **сегментний** і **сегментно-сторінковий** розподіл пам'яті
 - Тобто, сегментація підтримується постійно, а сторінкове перетворення може бути відключено
 - Сторінкове перетворення включається або відключається окремим прапорцем у регістрі керування процесором **cr0**
- Максимальний розмір сегмента – 4 ГБ
- У 16-розрядні сегментні регістри процесора завантажуються **селектори**, які вказують на **дескриптори** сегментів
 - **cs** – сегмент коду
 - **ss** – сегмент стека
 - **ds, es, fs, gs** – сегменти даних
- Дескриптори містяться у спеціальних системних таблицях, на які вказують **регістри системних адрес**
- У дескрипторах містяться:
 - базова адреса сегмента
 - межа (розмір) сегмента
 - правила доступу до сегмента



Регістри сегментів процесорів x86

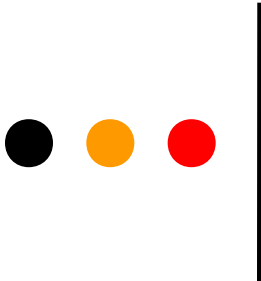
Позна-чення	Назва	Особливості використання
CS	сегмент коду (Code Segment register)	адресує сегмент коду, який виконується процесором <ul style="list-style-type: none">Для переходу в інший сегмент коду необхідно завантажити нове значення в реєстр CSЦе здійснюється автоматично при виконанні процесором команд jmp або call
SS	сегмент стека (Stack Segment register)	адресує сегмент стека, з яким в даний момент працює процесор <ul style="list-style-type: none">Для організації іншого стека необхідно завантажити в SS нове значенняПри цьому необхідно зберегти поточне значення SS, щоби мати змогу повернутись назад
ds es, gs, fs	сегменти даних (Data Segment register)	адресують дані в оперативній пам'яті, з якими працює процесор <ul style="list-style-type: none">Сегмент ds є основним, і адресується неявноІнші сегменти даних вимагають явної адресації за допомогою спеціальних префіксів в командах



Системні таблиці і реєстри системних адрес

- **Реєстри системних адрес** містять покажчики на системні таблиці, призначені для керування пам'яттю та диспетчеризації процесів
- Реєстри, що вказують на глобальні системні об'єкти (**gdt** та **ldtr**) містять базові лінійні адреси цих об'єктів, а також задають розмір об'єктів
- Реєстри, що вказують на локальні для кожного процесу об'єкти (**ldtr** та **tr**) містять лише селектори, які адресують відповідні дескриптори у глобальній таблиці
 - Таким чином, таблиця **GDT** містить дескриптори, що описують сегменти стану задач і локальні таблиці дескрипторів усіх процесів, що виконуються в системі

Реєстр	Таблиця (об'єкт)		Формат реєстру	
gdt	GDT	Global Descriptor Table	48	Містить базову адресу і межу таблиці
ldtr	LDT	Local Descriptor Table	16	Селектор у таблиці GDT
idtr	IDT	Interrupt Descriptor Table	48	Містить базову адресу і межу таблиці
tr	TSS	Task Status Segment	16	Селектор у таблиці GDT



Системні таблиці і реєстри системних адрес

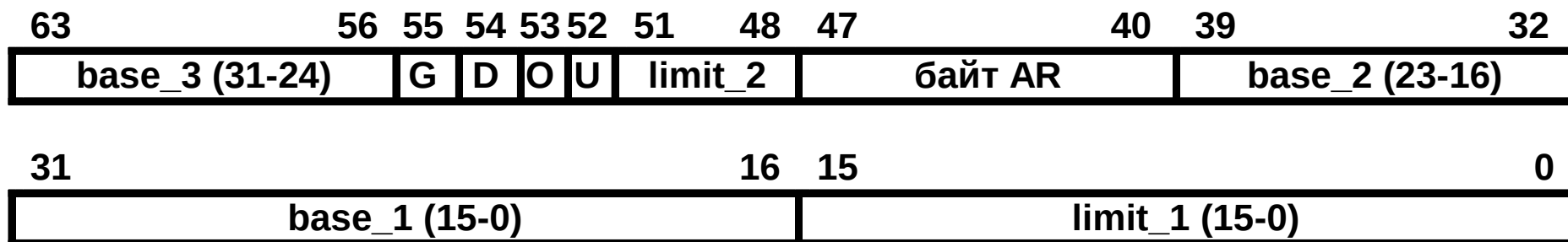
- Доступ до сегментів у пам'яті здійснюється через **дескриптори**
- Дескриптори сегментів пам'яті містяться у двох таблицях, доступних процесу
 - **Глобальна таблиця дескрипторів** (*Global Descriptor Table, GDT*)
 - Містить дескриптори, що описують програмний код і дані, спільні для усіх процесів (наприклад, бібліотеки, драйвери пристроїв, тощо), а також численні системні об'єкти
 - На цю таблицю вказує реєстр **gdtr**
 - **Локальна таблиця дескрипторів** (*Local Descriptor Table, LDT*)
 - Доступна лише тому процесу, який виконується в даний момент
 - Кожний процес має свою власну локальну таблицю
 - На цю таблицю вказує реєстр **ldtr**
- Обробники переривань доступні через **таблицю дескрипторів переривань** (*Interrupt Descriptor Table, IDT*)
 - На неї вказує реєстр **idtr**
- **Контекст процесу** знаходиться у спеціальному системному об'єкті, що називається **сегментом стану задачі** (*Task Status Segment, TSS*)
 - Цей об'єкт описується дескриптором, на який вказує реєстр **ts**

Селектор сегмента



- **Селектор** – це 16-розрядна структура, яка завантажується в сегментні реєстри
- Селектор адресує не сам сегмент, а його **дескриптор**
- 13 старших розрядів селектора є індексом в таблиці дескрипторів
 - Таким чином, кожна таблиця може містити до $2^{13}=8192$ 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**), який використовується механізмом захисту

Дескриптор сегмента



- Дескриптор сегмента є 8-байтовою структурою (64 розряди)
- Головні поля дескриптора:
 - 32-розрядна **базова адреса** (*base*)
 - 20-розрядна **межа сегмента** (*limit*) – визначає розмір сегмента в залежності від **прапорця гранулярності** *G*:
 - **G==0** → розмір у байтах (максимальний розмір сегмента 1 МБ)
 - **G==1** → розмір у 4-кБ сторінках (максимальний розмір сегмента 4 ГБ)
 - **Байт захисту** (*AR*)
 - **Прапорець розрядності** (*D*):
 - **D==0** → 16-розрядні операнди і режими 16-розрядної адресації
 - **D==1** → 32-розрядні операнди і режими 32-розрядної адресації



Байт захисту

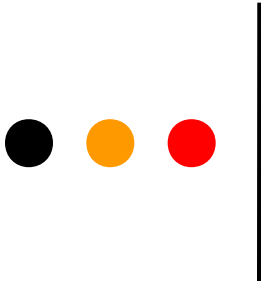
7 (47)	6 (46)	5 (45)	4 (44)	3 (43)	2 (42)	1 (41)	0 (40)
P	DPL	S	E	C/ED	R/W	A	
				type_seg			

- Розряд 7 (**P** – *Present*) показує наявність сегмента у пам'яті
- Розряди 5 і 6 (**DPL** – *Descriptor Privilege Level*) визначають рівень привілеїв дескриптора
- Розряд 4 (**S** – *System / Segment*) визначає, чи є об'єкт, який описує цей дескриптор, сегментом у пам'яті чи спеціальним системним об'єктом
- Розряди 1 – 3 визначають тип сегмента і права доступу до нього
 - Розряд 3 (**E** – *Executable*)
 - Розряд 2
 - Для сегментів коду – біт підпорядкованості (**C** – *Conforming*)
 - Для сегментів даних – біт розширення вниз (**ED** – *Expand Down*)
 - Розряд 1
 - Для сегментів коду – дозвіл на зчитування (**R** – *Readable*)
 - Для сегментів даних – дозвіл на записування (**W** – *Writable*)
- Розряд 0 (**A** – *Accessed*) встановлюється при доступі до сегмента



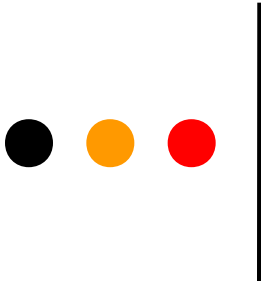
Значення поля типу сегмента

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



Завантаження селектора у сегментний регістр

- Якщо у селекторі **TI == 0**, то дескриптор міститься в **GDT**
 - З регістру **gdtr** визначають базову адресу і розмір таблиці **GDT**
 - За індексом вибирають з **GDT** потрібний дескриптор
 - Перевіряють, чи не виходить дескриптор (з урахуванням його розміру – 8 байтів) за встановлену межу таблиці **GDT**. В разі виходу за межу – **виняткова ситуація 11**
 - Перевіряють сумісність типу дескриптора і достатність привілеїв для доступу до нього. Якщо щось не так – **виняткова ситуація 13**
- Якщо у селекторі **TI == 1**, то дескриптор міститься в **LDT**
 - З регістру **gdtr** визначають базову адресу і розмір таблиці **GDT**
 - З таблиці **GDT** вибирають дескриптор, що описує таблицю **LDT**, для чого в якості селектора використовують вміст регістру **ldtr**
 - Перевіряють, чи відповідає тип дескриптора таблиці дескрипторів і чи присутня таблиця у фізичній пам'яті
 - З дескриптора таблиці **LDT** визначають базову адресу таблиці та її межу
 - Далі здійснюють операцію вибору з таблиці необхідного дескриптора, яка аналогічна операції вибору дескриптора з таблиці **GDT**, з тими ж перевітками.



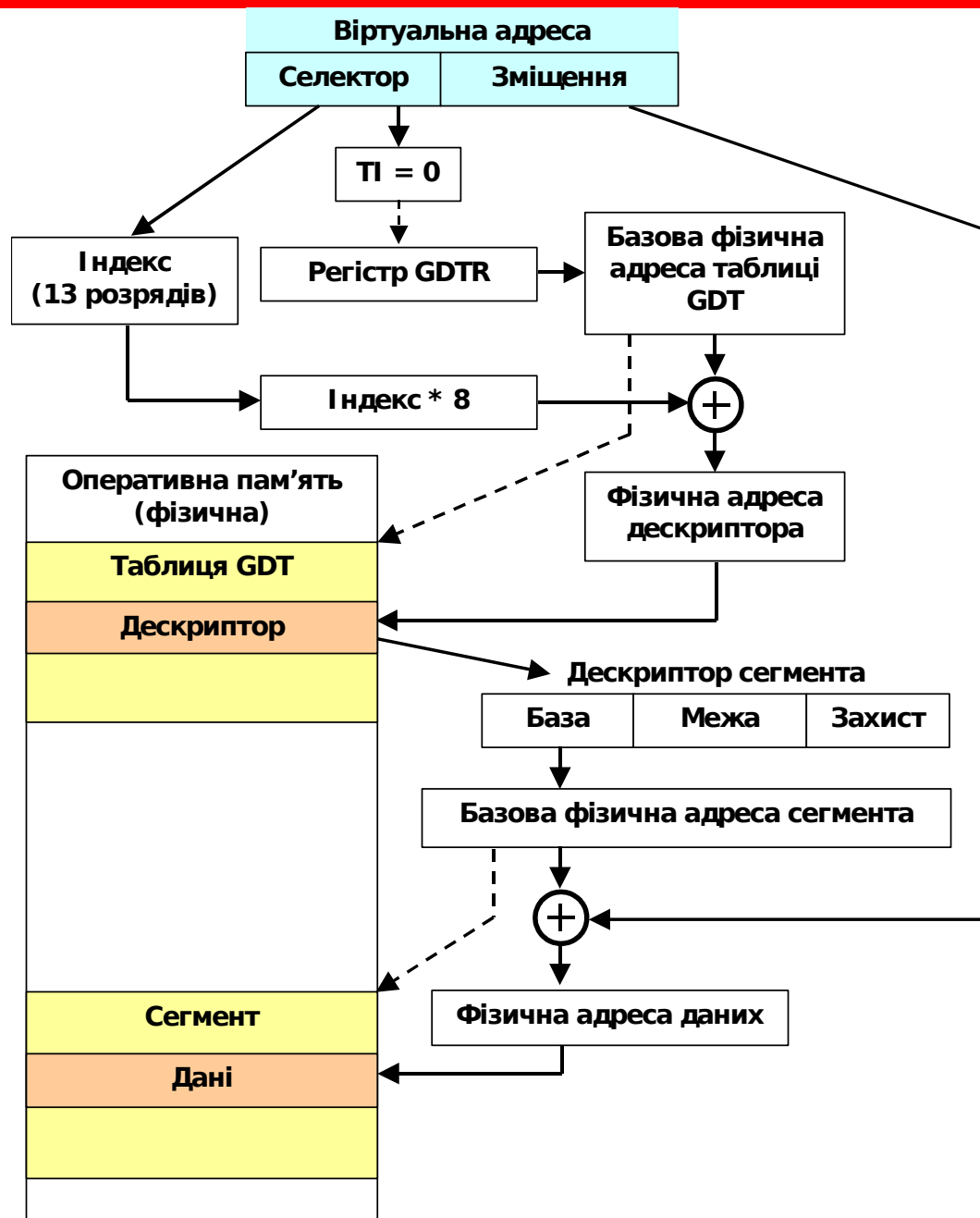
Дозволені комбінації бітів байту захисту дескриптора при завантаженні селектора у сегментний регістр

Регістр	DPL	S	E	C/ED	R/W	Примітка
CS	$\geq RPL$ $\geq CPL$	1	1	x	x	сегмент коду
SS	$= RPL$ $= CPL$	1	0	1	1	сегмент стека
ds, es, fs, gs	$\geq RPL$ $\geq CPL$	1	1	x	1	сегмент коду
			0	x	x	сегмент даних або стека



Перетворення адрес за сегментного розподілу пам'яті

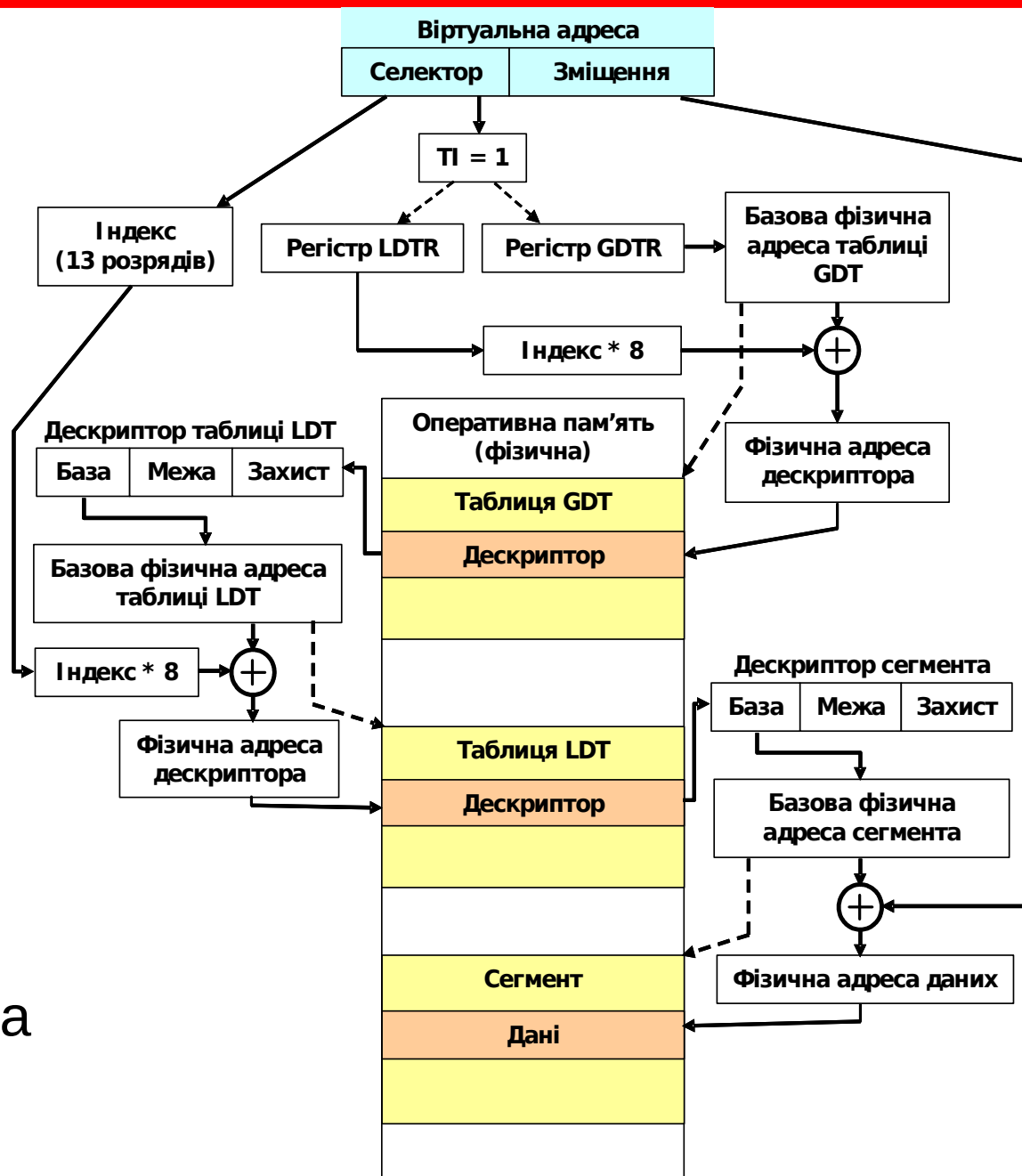
дескриптор сегмента
знаходиться у
таблиці GDT





Перетворення адрес за сегментного розподілу пам'яті

дескриптор сегмента
знаходиться у
таблиці LDT





Звернення до пам'яті

- Перевіряють дозвіл на операцію
- Перевіряють коректність доступу (відсутність виходу за межу сегмента)
 - Сегмент стека зростає в бік молодших адрес, і обчислена адреса повинна бути не меншою за межу сегмента
 - Сегменти коду і даних зростають у бік старших адрес, тому до обчисленої адреси додається розмір даних, і отримана адреса повинна бути не більшою за межу сегмента
- Дозволені комбінації бітів байту захисту:

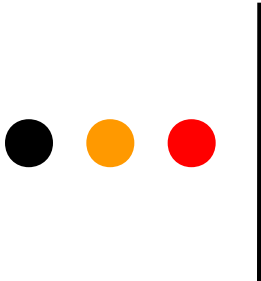
Операція	S	E	C/ED	R/W	Примітка
Зчитування з пам'яті	1	0	x	x	сегмент даних або стека
	1	1	x	1	сегмент коду
Записування у пам'ять	1	0	x	1	сегмент даних або стека

Сторінковий механізм керування пам'яттю

- Лінійна адреса є 32-розрядною, старші 20 розрядів інтерпретуються як номер сторінки, а молодші 12 – як зміщення в сторінці
- Номер сторінки інтерпретується як індекс дескриптора сторінки, а з дескриптора визначається номер сторінки у фізичній пам'яті

31	12	11	...	9	8	7	6	5	4	3	2	1	0
Номер сторінки		AVL		0	D	A	PCD	PWT	U	W	P		

P (Present)	Прапорець присутності сторінки у фізичній пам'яті
W (Writable)	Прапорець дозволу записування у сторінку
U (User mode)	Прапорець користувач/супервізор
PWT, PCD	Керують механізмом кешування сторінок (введені, починаючи з процесора 80486)
A (Accessed)	Ознака, що мав місце доступ до сторінки
D	Ознака модифікації вмісту сторінки
0	Зарезервовані
AVL (Available)	Зарезервовані для потреб операційної системи



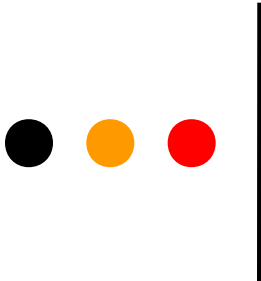
Регістри, що забезпечують сторінковий механізм

cr0	<p>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:</p> <p>pe (<i>Protect Enable</i>), біт 0 – вмикає захищений режим роботи процесора;</p> <p>cd (<i>Cache Disable</i>), біт 30 – вмикає використання внутрішньої кеш-пам'яті (кеш першого рівня);</p> <p>pg (<i>Paging</i>), біт 31 – вмикає сторінкову трансляцію адрес.</p>
cr2	<p>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</p>
cr3	<p>Містить фізичну базову адресу каталогу сторінок</p>



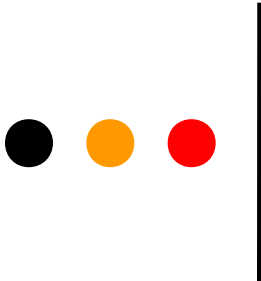
Особливості використання пам'яті в x86-64 (amd64)

- У сучасних реалізаціях x86-64 насправді може використовуватись лише невелика частина 64-розрядних адрес
 - Для перетворення адреси (пошуку по таблиці сторінок) використовуються лише **найменші значимі 48 біт** адреси віртуальної пам'яті
 - Біти від 48 до 63 для будь-якої віртуальної адреси повинні бути копією біта 47 — інакше процесор згенерує виняткову ситуацію
 - Формат адрес що відповідають цьому правилу називають "канонічною формою"
- Реально використовуються 2 діапазони адрес:
 - від **0** до **00007FFFFFFFFFFFFF**,
 - від **FFFF800000000000** до **FFFFFFFFFFFFFFFF**
 - Кожний з цих діапазонів — у 128 TiB, разом — 256 TiB віртуального адресного простору
 - Це приблизно в 64000 разів більше від віртуального адресного простору в 32-розрядній архітектурі, але все ж є мізерною частинкою повного 64-розрядного адресного простору



Керування завданнями у процесорах Intel x86

- Процесори x86 надають засоби для підтримки **виклику процедур** і **виклику завдань**
- **Виклик процедури** – це виклик коду в межах одного процесу (потoku)
 - При цьому відбувається переключення на інший сегмент коду, але зберігаються структури, які керують адресним простором процесу
- **Виклик завдання** – це створення нового процесу з усіма відповідними структурами (що є здебільшого роботою ОС) і переключення на цей процес
 - Виклик завдання також відбувається у багатозадачній ОС при переключенні між процесами
 - У цьому випадку необхідні структури вже створені, але необхідно здійснити переключення на них, забезпечивши збереження відповідних структур поточного процесу



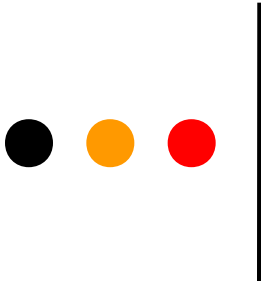
Виклик завдань

- Для переключення між завданнями (процесами) в команді **CALL** повинен бути заданий селектор, що вказує на дескриптор системного сегмента **TSS** – *сегмента стану завдання*
 - Такі дескриптори можуть знаходитись лише у таблиці **GDT**
 - Формат дескриптора аналогічний формату дескриптора сегмента даних, але значення біту **S = 0**
- Сегмент **TSS** зберігає *контекст процесу*, тобто всю інформацію, необхідну для поновлення виконання процесу після переривання
- Переключення контекстів здійснюється апаратно



Структура сегмента TSS

Бітова карта введення/виведення (БКВВ)		8 кБ
Додаткова інформація ОС		
Відносна адреса БКВВ	0 . . . 0	64
0 . . . 0	Селектор LDT	60
0 . . . 0	gs	5C
0 . . . 0	fs	58
0 . . . 0	ds	54
0 . . . 0	ss	50
0 . . . 0	cs	4C
0 . . . 0	es	48
edi		44
esi		40
ebp		3C
esp		38
ebx		34
edx		30
ecx		2C
eax		28
eflags		24
eip		20
cr3		1C
0 . . . 0	ss рівня 2	18
esp2		14
0 . . . 0	ss рівня 1	10
esp1		0C
0 . . . 0	ss рівня 0	08
esp0		04
0 . . . 0	селектор TSS повернення	00



Послідовність дій під час переключення контекстів

1. Виконується команда **CALL**, в якій задано селектор, що указує на дескриптор сегмента типу **TSS**
2. Здійснюється перевірка прав доступу
 - Якщо **CPL > DPL**, доступ заборонено
3. Селектор поточного сегмента **TSS** береться з регістру **tr**
До **TSS** заносяться значення регістрів процесора
4. В регістр **tr** завантажується селектор **TSS** завдання, на яке переключається процесор
5. З нового **TSS** в регістр **ldtr** завантажується селектор **LDT**
6. З відповідних полів нового **TSS** оновлюються всі регістри процесора
7. Селектор сегмента **TSS** попереднього завдання заноситься в поле селектора повернення нового сегмента **TSS**



Вплив рівнів привілеїв процесів на виклик завдань

- Для виклику нового завдання поточний процес повинен мати рівень привілеїв, **не нижчий за нове завдання**
 - Таким чином, більш привілейований код має змогу викликати менш привілейовані завдання
 - Наприклад, ОС має змогу запускати і переключати програми користувача
- Передбачена також можливість виклику і більш привілейованого коду
 - Для цього використовуються **шлюзи**
 - Для виклику завдання шлюз повинен указувати на дескриптор TSS
 - При цьому шлюзи можуть знаходитись як у таблиці GDT, так і в LDT



Привілейовані команди

- Процесори x86 надають права виконання деяких команд лише програмам, що мають визначений рівень привілеїв **CPL**
 - Такі команди називаються **привілейованими**
 - Цей механізм дозволяє реалізувати захист структур ОС від дій процесів користувача
- До числа команд, які можуть виконуватись лише при **CPL = 0**, тобто з найвищими привілеями, відносяться:
 - команди роботи з регістрами керування **cr0...cr4**;
 - команди завантаження регістрів системних адрес **gdtr, ldtr, idtr, tr**;
 - команда зупинки процесора **HALT**.
- Існують також команди, які стосуються операцій введення/виведення і вимагають, аби рівень привілеїв поточного процесу **CPL** був **вищий, ніж рівень привілеїв введення/виведення IOPL** (тобто, **CPL ≤ IOPL**)
 - Такі команди іноді називають **чутливими**
 - До них належать:
 - команди заборони/дозволу маскованих переривань **CLI/SLI**;
 - команди введення/виведення **IN, INS, OUT, OUTS**.
- Рівень привілеїв введення/виведення **IOPL** зберігається у спеціальному полі регістру **eflags**