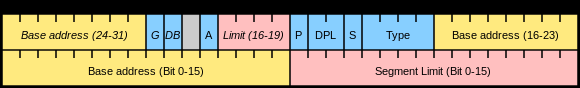
**Схеми перетворення логічної адреси в лінійну у захищеному режимі (продовження). Дескрипторні таблиці GDT, LDT.**

**Мал. 1.**  Структура дескриптора фрагмента ОП



(https://lawbooks.news/telekommunikatsionnyie-sistemyi-kompyuternyie/struktura-deskriptora-segmenta-60013.html)

**Дескрипторні таблиці** - це масиви пам'яті змінної довжини, які складаються з 8-байтних елементів, дескрипторів. Дескрипторна таблиця може мати довжину від 8 байт до 64 Кбайт і в кожній таблиці може бути до 8192 дескрипторів. Існують дві обов'язкові дескрипторні таблиці - глобальна дескрипторна таблиця (Global Descriptor Table - GDT) і дескрипторна таблиця переривання (Interrupt Descriptor Table - IDT), а також множина (аж до 8191) необов'язкових локальних дескрипторних таблиць (Local Descriptor Table - LDT), з яких в кожен момент часу процесору доступна лише одна. Розташування дескрипторних таблиць визначається регістрами процесора GDTR, IDTR, LDTR.

GDT містить дескриптори, доступні всім задачам в системі. GDT може містити дескриптори будь-яких типів: і дескриптори фрагментів, і системні дескриптори (окрім шлюзів переривань і пасток). Перший елемент GDT (з нульовим індексом) не використовується. Йому відповідає нуль-селектор, що позначає "порожній" покажчик.

LDT забезпечують спосіб ізоляції сегментів програми і даних задачі, що виконується, від інших задач. LDT пов'язана з конкретною задачею і може містити тільки дескриптори фрагментів, шлюзи викликів і шлюзи задач.

Фрагмент не може бути доступний задачі, якщо його дескриптор не існує ні в поточній таблиці LDT, ні в таблиці GDT. Використання двох дескрипторних таблиць дозволяє, з одного боку, ізолювати і захищати сегменти задач, що виконуються, а з іншої - дозволяє розділяти глобальні дані і код між різними задачами.

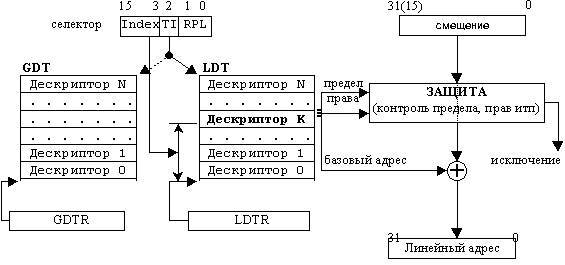
IDT містить тільки шлюзи завдань, шлюзи переривань або шлюзи пасток.

Для обчислення лінійної адреси МП виконує наступні дії (рис. 2):

1. МП використовує селектор фрагменту для знаходження дескриптора сегменту. Селектор містить індекс дескриптора в дескрипторній таблиці (Index), біт TI, що визначає, до якої дескрипторної таблиці проводиться звернення (LDT або GDT), а також права доступу до сегменту (RPL). Якщо селектор зберігається в сегментному регістрі, то звернення до дескрипторних таблиць відбувається тільки при завантаженні селектора в сегментний регістр, оскільки кожен сегментний регістр зберігає відповідний дескриптор в программно-недоступному ("тіньовому") регістрі-кеші.
2. МП аналізує дескриптор сегменту (фрагменту), контролюючи права доступу (фрагмент доступний з поточного *рівня привілеїв*) і межу сегменту (зсув не перевищує межу);
3. МП додає зсув до базової адреси сегменту і отримує лінійну адресу.

Якщо сторінкова трансляція відключена, то сформована лінійна адреса вважається фізичною і виставляється на шину процесора для виконання циклу читання або запису пам'яті.

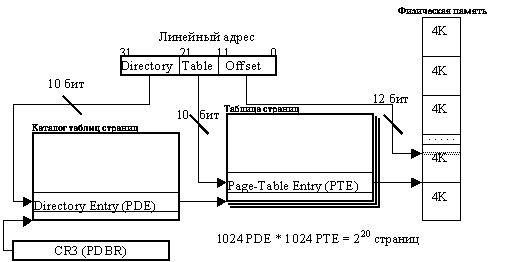
**Мал. 2.**  Схема перетворення логічної адреси в лінійну.



Механізм *сегментації у Protected Mode (фрагментації)* забезпечує захист, але він не дуже зручний для реалізації віртуальної пам'яті (підкачки). У дескрипторі фрагменту є біт присутності, по ньому процесор визначає, чи знаходиться даний фрагмент у фізичній пам'яті або на зовнішньому пристрої, де він запам'ятовується (на вінчестері). У останньому випадку генерується виключення #11, обробник якого може завантажити фрагмент в пам'ять.

Незручність полягає в тому, що різні фрагменти можуть мати різну довжину. Цього можна уникнути, якщо механізм підкачки реалізовувати на основі сторінкового перетворення. Особливістю цього перетворення є те, що процесор в цьому випадку оперує з блоками фізичної пам'яті рівної довжини (4 Кбайт) - сторінками. Сторінки не мають безпосереднього відношення до логічної структури програми. Крім того, в МП підрежиму P6 сторінкова трансляція забезпечує 36-бітову фізичну адресацію пам'яті (64 Гбайт). Сторінкове перетворення діє тільки в захищеному режимі і включається установкою в 1 біта PG в регістрі CR0.

У *сторінковому перетворенні* беруть участь два типи структур: каталоги таблиць (Page Directory) і таблиці сторінок (Page Table). Ці структури складаються з 1024 32-бітових елементів. Елементи містять старші 20 біт фізичної адреси об'єктів, що адресуються. Елементи таблиці сторінок (Page Table Entry - PTE) адресують сторінки, а елементи каталога таблиць (Page Directory Entry - PDE) адресують таблиці сторінок. Старші 20 біт фізичної адреси каталога таблиць зберігаються в регістрі CR3 (Page Directory Base Register - PDBR) (це єдиний регістр процесора, який містить фізичну адресу пам'яті). Всі структури вирівняні на границі сторінки.

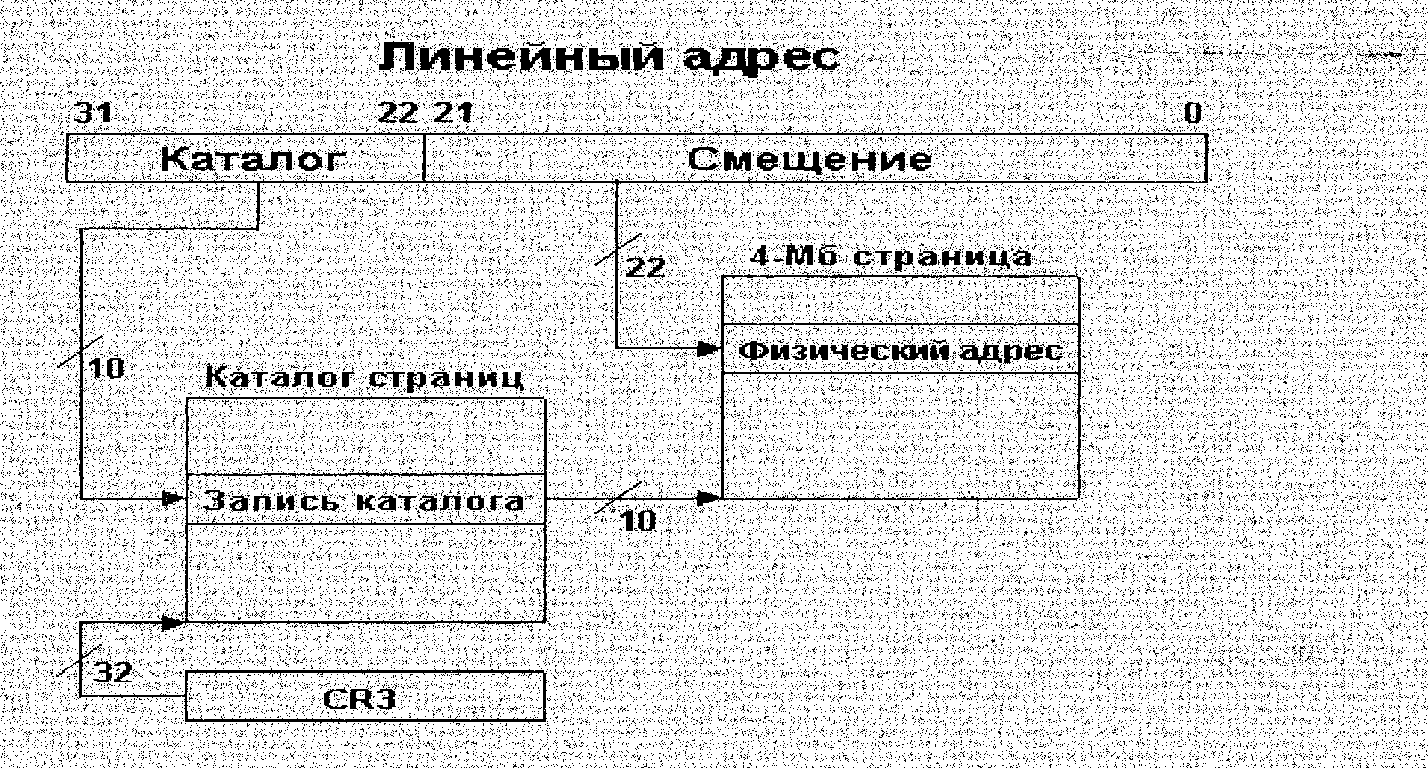


**Мал. 3.**  Стандартна 3-рівнева схема сторінкової трансляції

В процесі сторінкової трансляції адрес отримана лінійна адреса розбивається на три частини. Старші десять біт (Directory) лінійної адреси є індексом елементу з каталога таблиць. По цьому елементу визначається фізична адреса таблиці сторінок. Біти 21-12 (Table) лінійної адреси вибирають елемент з цієї таблиці сторінок. Вибраний елемент визначає фізичну адресу сторінки. Молодші 12 біт (Offset) лінійної адреси визначають зсув від початку сторінки.

Сторінки починаються на границі 4 Кбайт областей пам'яті, тому молодші 12 біт адреси сторінки завжди рівні нулю. У каталозі таблиць елементи зберігають фізичні адреси таблиць сторінок. У таблиці сторінок елементи зберігають фізичні адреси самих сторінок.

У МП реалізувано також іншу можливість - розширення розміру сторінок (Page Size Extension - PSE). PSE дозволяє використовувати сторінки розміром 4 Мбайт і однорівневий механізм сторінкової трансляції. У підрежимі P6 шина адреси була розширена до 36 біт. Відповідно, процесори Pentium Pro, Pentium II, Pentium III і пізніші (сучасні) здатні адресувати до 64 Гбайт фізичної пам'яті. Подальше розширення ширини адресної шини дозволило розширити адресний простір ще більше. Ця можливість називається розширенням фізичної адреси (Physical Address Extension - PAE) і доступна тільки при використанні сторінкової трансляції. Таким чином, застосування PSE і PAE впливає на *сторінкове перетворення.*

**

**Мал. 4.**  2-рівнева схема сторінкової трансляції

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Таблиця 1. Особливості сторінкового перетворення в PSE і PAE** | | | | | |
| **CR0.PG** | **CR4.PAE** | **CR4.PSE** | **PDE.PS** | **Розмір сторінки** | **Розрядність фіз. Адреси** |
| 0 | X | X | x | - | 32 бит |
| 1 | 0 | 0 | x | 4 Кб | 32 бит |
| 1 | 0 | 1 | 0 | 4 Кб | 32 бит |
| 1 | 0 | 1 | 1 | 4 Мб | 32 бит |
| 1 | 1 | X | 0 | 4 Кб | 32 бит |
| 1 | 1 | X | 1 | 2 Мб | 32 бит |

Вбудовані засоби перемикання завдань забезпечують реалізацію *багатозадачності* в захищеному режимі. **Задача** - це "одиниця вимірювання" завдань для процесора, яке процесор може виконувати, припиняти і здійснювати над ним диспетчеризацію. Як задача може виконуватися прикладна програма, сервіс операційної системи, ядро операційної системи, обробник переривання або виключення і т.п. У захищеному режимі архітектура надається механізм для збереження стану задачі та перемикання з однієї задачі на іншу.

Середовище задачі складається з вмісту регістрів МП і всього коду з даними в просторі пам'яті. МП здатний швидко перемикатися з одного середовища виконання в інше, імітуючи паралельне виконання декількох задач. Для деяких задач може емулюватися управління пам'яттю, як у МП 8086. Такий режим виконання задачі називається режимом віртуального 8086 (Virtual 8086 Mode). Про перебування задачі в даному стані сигналізує біт VM в регістрі флагів. При цьому задачі віртуального МП 8086 ізольовані і захищені, як одна від іншої, так і від звичайних задач захищеного режиму.

Задачу складають два компоненти: адресний простір задачі та сегмент (фрагмент) стану завдання (Task State Segment - TSS). У адресний простір задачі входять доступні для неї фрагменти коду, даних і стека. Сегмент стану завдання TSS зберігає стан регістрів (контекст) процесора:

* стан сегментних регістрів (селектори сегментів, що формують адресний простір задачі);
* стан регістрів загального призначення;
* стан регістра флагів (EFLAGS);
* значення регістра-лічильника для визначення наступної команди (EIP);
* значення регістра CR3 (PDBR);
* значення регістра LDTR.

У багатозадачних системах TSS надає механізм для вкладення задач. Кожна задача ідентифікується селектором відповідного їй TSS. Цей селектор завантажується в регістр завдання (Task Register - TR) при перемиканні на задачу. У тіньову частину регістра завантажуються базова адреса, границя і атрибути TSS. Операційна система може передбачати для кожної задачі свій лінійний адресний простір (свій набір сторінок для *сторінкового перетворення*), тоді при перемиканні завдання завантажується і регістр CR3 (PDBR), що зберігає адресу каталогу таблиць для сторінкового перетворення.

**Глобальна таблиця дескрипторів (GDT):**

КОЖНА ОС ПОВИННА МАТИ ОДНУ ТАБЛИЦЮ GDT. Цією таблицею можуть користуватися ВСІ програми і задачі системи: зберігати в ній СВІЙ дескриптор.

Таблиця GDT - сама по собі НЕ ФРАГМЕНТ, а СЕГМЕНТ! Це структура даних в лінійному адресному просторі (у пам'яті). ПОЧАТОК таблиці GDT зберігатися В РЕГІСТРІ GDTR. Регістр GDTR - це звичайний регістр, як ЕАХ, EIP, ES, тільки ось його функція полягає не в зберіганні проміжних даних, а в зберіганні фіксованої величини - ПОЧАТКА ТАБЛИЦІ GDT. Адреса початку таблиці GDT у пам'яті обов’вязково має кратна 8.

**Регістр GDTR:**

32-бітовий лінійний базовий адрес|16-битный ліміт таблиці

Регістр GDTR має довжину 48 бит. Містить не тільки адресу початку таблиці GDT в пам'яті, а ще і її ліміт. Ліміт таблиці - 16-бітове значення, показує величину таблиці в байтах -1 (як і у випадку з лімітом сегменту: якщо ліміт таблиці в GDTR рівний 0, то насправді це означає що реально (у пам'яті) є фрагмент рівний одному байту).

Сегментний дескриптор ЗАВЖДИ займає 8 байт (2 подвійних слова). Отже, ліміт таблиці дескрипторів - величина, рівна 8N-1 байт.

Перший дескриптор в GDT не використовується: він називається «нульовий дескриптор» (null descriptor). При зверненні до пам'яті через цей дескриптор виникає знайома ситуація #GP.

Завантажити/прочитати значення регістра GDTR можна командами LGDT/SGDT. За замовчанням (після натискання на кнопку Reset або включення комп'ютера) база GDT дорівнює нулю, а ліміт - FFFFh, тобто фактично за умовчанням виділено максимум місця, під FFFFh/8 = 8191 дескрипторів (мінус один, враховуючи null descriptor).

**Локальна таблиця дескрипторів (LDT):**

На відміну від GDT абсолютно не ЗОБОВ'ЯЗАНА бути присутньою взагалі. І в той же самий час, їх можна завести довільну кількість (GDT ПОВИННА БУТИ лише одна!). Кожна задача може мати свою власну LDT, і в той же час декілька задач можуть використовувати одну LDT на всіх.

LDT - це ФРАГМЕНТ (GDT – 64 КБ структура даних). Оскільки LDT - це фрагмент, то це означає у неї є теж свій дескриптор у таблиці GDT. Так, у таблиці дескрипторів є свій дескриптор.

Так само, як і у GDT у LDT теж є свій регістр - LDTR. На відміну від GDTR цей регістр, крім інформації про базу і ліміт LDT, містить ще одне поле - сегментний селектор.

**Регістр LDTR:**

Сегментний селектор (16 біт)

32 - бітова лінійна базова адреса

16 - бітний ліміт фрагменту

Інструкції LLDT і SLDT дозволяють писати/читати регістр LDTR. Так само, при reset-e значення бази в LDTR стає нульовим, а ліміт - в FFFFh.

*Приклад:* Descr\_code2 db 0DEh,0BCh,01h,20h,10h,XXh,0X001010b,00h

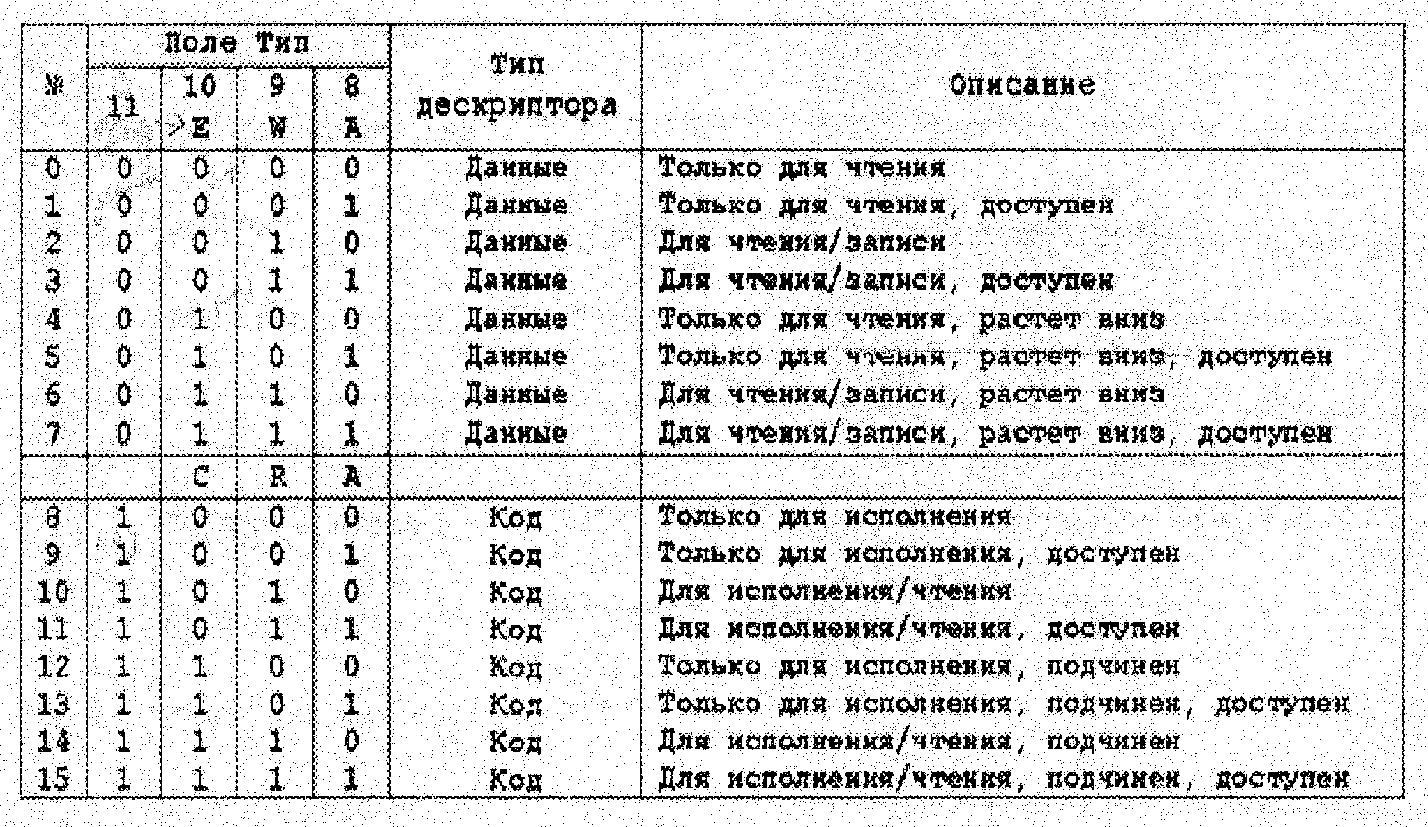
Розглянемо докладніше поля дескриптора, відмічені хрестиками **(Ixxx)** при S=1 (12-й біт) в другому слові, тобто якщо дескриптор описує або дані, або код.

11 біт в другому подвійному слові **І** (він же четвертий біт поля «Тип») показує, що фрагмент є фрагментом коду або даних (0 - дані, 1 - код). Якщо це фрагмент даних, то молодші три біта поля «Тип» (10-8) інтерпретуються як біт Е (expansion-direction) - зростання напряму, біт W (write-enable) - запис дозволений і біт A (accessed) - доступний.

Перший приємний сюрприз: виявляється, немає такого спеціального типу сегменту, як сегмент стека. Сегмент стека - це ні що інше, як фрагмент даних, що росте вниз. Фрагмент стека ОБОВ'ЯЗКОВО повинен бути доступний для читання/запису! Це повинно бути врахувати це при описі фрагменту стека.

ДЛЯ фрагменту коду молодші три біта поля «Тип» інтерпретуються, як З (conforming) -підлеглий, R (read enable) - читання дозволено і A (accessed) - доступний. Фрагмент коду може бути доступний як лише для виконання, так і для виконання/читання. Для виконання/читання він може бути доступний у тому випадку, коли зберігаються деякі константи або інші статичні дані в пам'яті, безпосередньо в сегменті коду. А ГОЛОВНЕ: У ЗАХИЩЕНОМУ РЕЖИМІ В СЕГМЕНТ КОДУ ПИСАТИ НЕ МОЖНА!

Усі можливі варіанти наведено у таблиці.



ТЕПЕР, ЯКЩО S=0 (фрагмент є системним фрагментом). Процесор розрізняє 6 типів системних дескрипторів:

• Дескриптор сегменту «таблиця LDT»

• Дескриптор сегменту «стан завдання» (TSS, про нього пізніше)

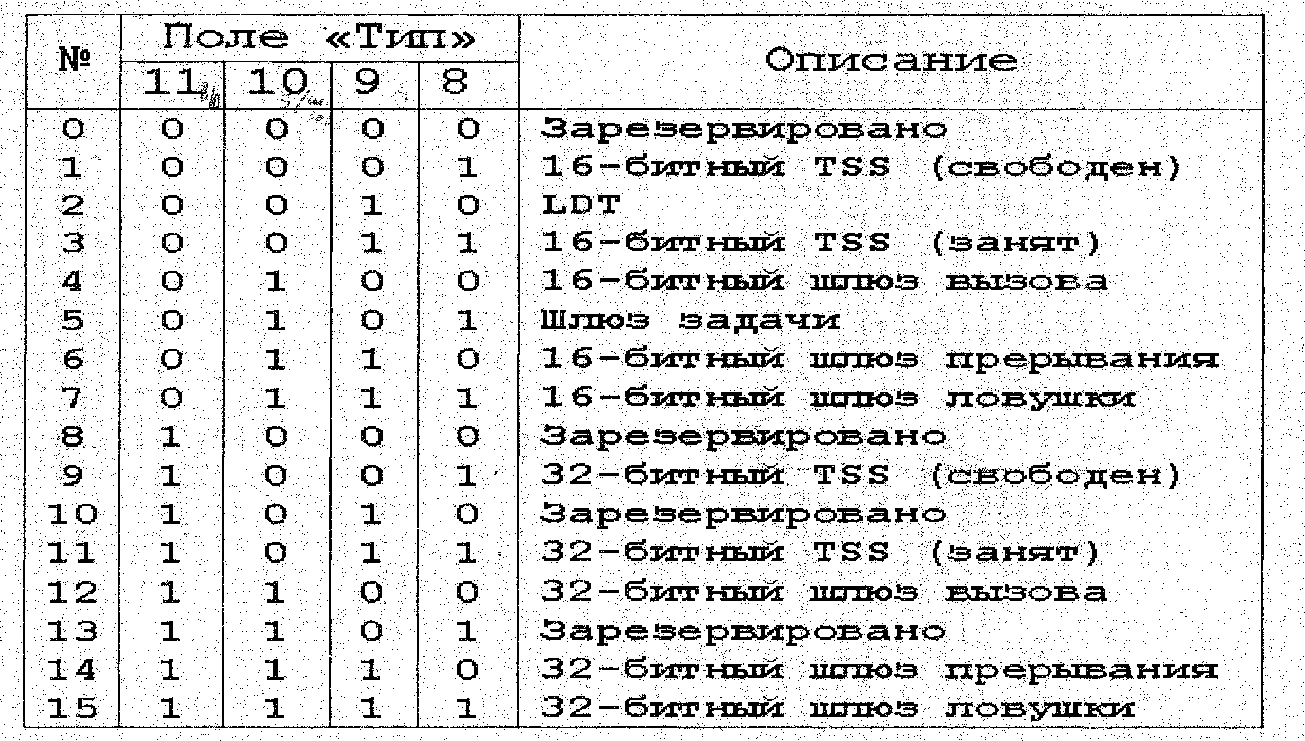
• Дескриптор шлюзу виклику (Call-Gate) - як зубна паста :)

• Дескриптор шлюзу переривання (Interrupt-Gate)

• Дескриптор шлюзу пастки (Trap-Gate)

• Дескриптор шлюзу завдання (Task-Gate)

У свою чергу, ці 6 типів дескрипторів діляться на дві групи: дескриптори системних сегментів (перші два) і дескриптори шлюзів (всі інші). Найперший (дескриптор сегменту «таблиця LDT») вже відомий, а у таблиці нижче наведено інші варіанти системних дескрипторів.



**Селектор**

Рухаємось далі вниз по сходинках: фрагмент в пам'яті <­-- дескриптор <-- таблиця дескрипторів. Розглянемо наступне поняття

СЕЛЕКТОР: селектор - це 16-бітова структура даних, яка є ідентифікатором фрагменту.

У кожного фрагменту є свій дескриптор, нам відомо де дескриптори розташовані. Селектор указує не на фрагмент в пам'яті, а на його дескриптор, у таблиці дескрипторів. СЕЛЕКТОР ЖИВЕ В СЕГМЕНТНОМУ РЕГІСТРІ (CS, DS...).

Поле ІНДЕКС (биті 3-15) указує на один з 8192 дескрипторів в таблиці дескрипторів (GDT або LDT).

Дескриптор займає 8 байт. Якщо індекс рівний двійці, то це, означає, що селектор указує на другій дескриптор у таблиці. ПРОЦЕСОР МНОЖИТЬ значення поля ІНДЕКС НА 8 і додає до отриманого значення АДРЕСУ БАЗИ ТАБЛИЦІ. Тобто процесор множить «двійку» на 8, а потім додає значення регістра таблиці – як результат, отримуємо ПОЧАТОК ДЕСКРИПТОРА.

З ЯКОЇ САМЕ ТАБЛИЦІ дескриптор? Для цього потрібно встановити флаг TI (table indicator) (другий біт селектора). Якщо він = 0, то розглядається значення регістра GDTR (тобто дескриптор з таблиці GDT), якщо ж встановлений (=1) - LDTR.

Запрошуваний рівень привілеїв (Requested Privilege Level) розглянемо пізніше.

*Приклад:* припустимо, що заносимо в DS число 0000000000110 0 00. Відразу розбиваємо DS на шматочки (15-3 биті - індекс, 2 - TI, 1-0 – RPL). Індекс рівний 6. Значить, шостий по рахунку дескриптор. З GDT, бо Т1=0.

Занесемо в ES число 0000000001000 1 00Ь. Восьмий дескриптор, який знаходиться у LDT.