

Тема:

# Організація пам'яті

## Лекція 3

*Викладач: Порєв Віктор Миколайович*

## Фізична пам'ять

Пам'ять, яку адресує процесор по шині адреси, зветься **фізичною пам'яттю**. Фізична пам'ять організована як послідовність байтів. Кожний байт фізичної пам'яті має унікальну адресу, яка зветься **фізичною адресою**.

У захищеному режимі процесори архітектури IA-32 штатно забезпечують фізичний адресний простір до 4 Гбайт ( $2^{32}$  байт). Адресний простір визначається розрядністю шини адреси. Цей адресний простір є пласким та суцільним (несегментованим) – для нього адреса може бути у діапазоні від 0 до FFFFFFFFh.

Починаючи з процесора Pentium Pro, архітектура IA-32 також підтримує розширення фізичного адресного простору до  $2^{36}$  байтів (64 Гбайтів) із максимальною адресою FFFFFFFFh. На тепер у архітектурі IA-32 є вже два механізми розширення фізичних адрес:

- physical address extension (PAE)
- 36-bit page size extension (PSE-36), запропонований у процесорах Pentium III.

## Фізична пам'ять

Архітектура 64-бітна дозволяє використовувати значно більше фізичної пам'яті – завдяки більшій розрядності шини адреси, ніж для 32-бітної архітектури. Проте, це не означає, що сучасні 64-бітні процесори підтримують  $2^{64}$  байт фізичної пам'яті.

Скільки фізичної пам'яті може підтримувати 64-бітний процесор архітектури Intel 64 (або сумісний по системі команд конкуруючий процесор AMD64)? Це залежить від версії реалізації процесора. Довідку можна отримати, якщо виконати команду CPUID із параметром (значенням регістру EAX) 80000008h.

**mov EAX, 80000008h**

**cuid**

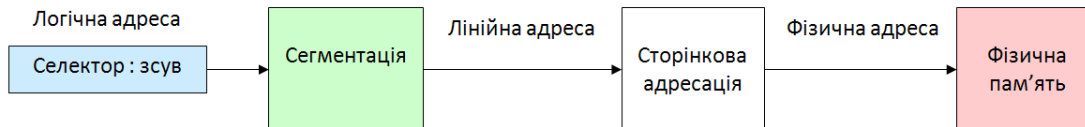
Команда CPUID запише у регістр EAX деяке значення – його біти 0-7 означатимуть кількість бітів адреси фізичної пам'яті. Наприклад, якщо результат EAX = 00003028h, то молодший байт 28h означає 40 розрядів фізичної адреси даного процесора. Розрядність адрес 40 біт означає для відповідного процесора можливість адресувати до  $2^{40} = 1\text{Tбайт}$  фізичної пам'яті.

## Сегментація пам'яті та сторінкова адресація

Будь-яка операційна система або програма, розроблені для процесорів IA-32 або Intel 64, для доступу до пам'яті використовують сервіс процесорної системи керування пам'яттю. Можливості системи керування пам'яттю розділюються на дві частини: сегментація та сторінкова адресація.

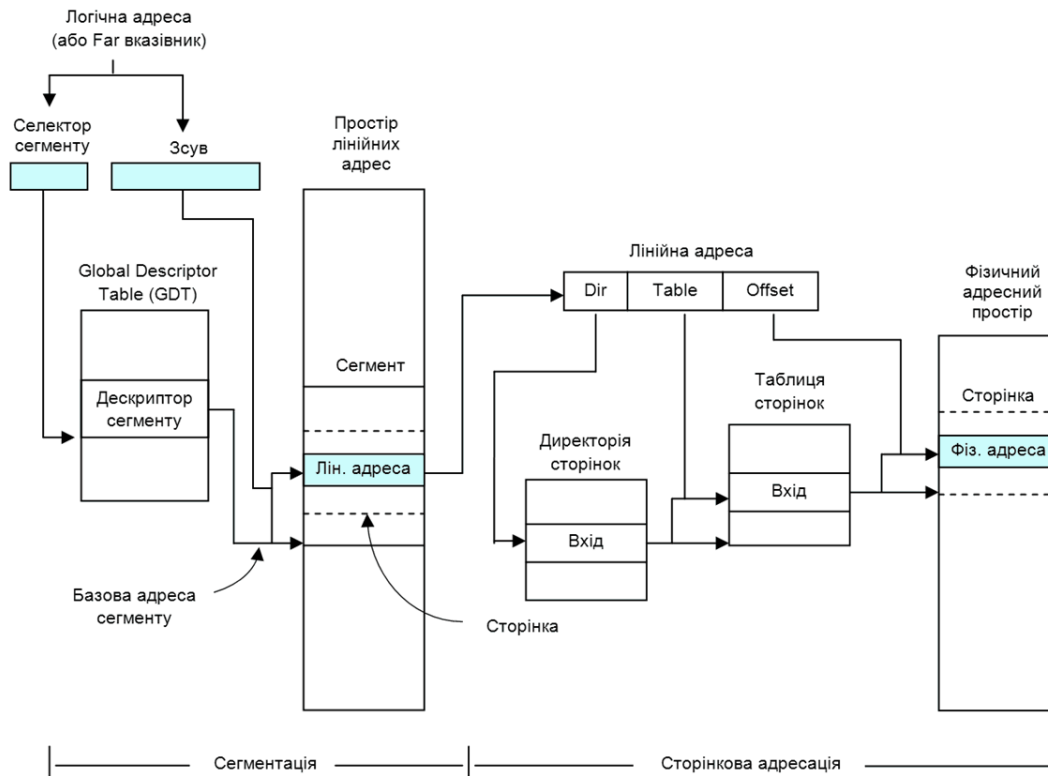
**Сегментація** (segmentation) надає механізм **ізолювання блоків пам'яті** - індивідуальних модулів коду, даних та стеку. У захищеному режимі сегментація не дозволяє програмі втручатися у пам'ять, виділену для інших програм.

**Сторінкова адресація** (paging) використовується для створення послідовного віртуального простору для чого логічні блоки пам'яті відображаються на фізичну пам'ять (**віртуальна пам'ять**). Сторінкова адресація також може використовуватися для ізоляції пам'яті різних програм.



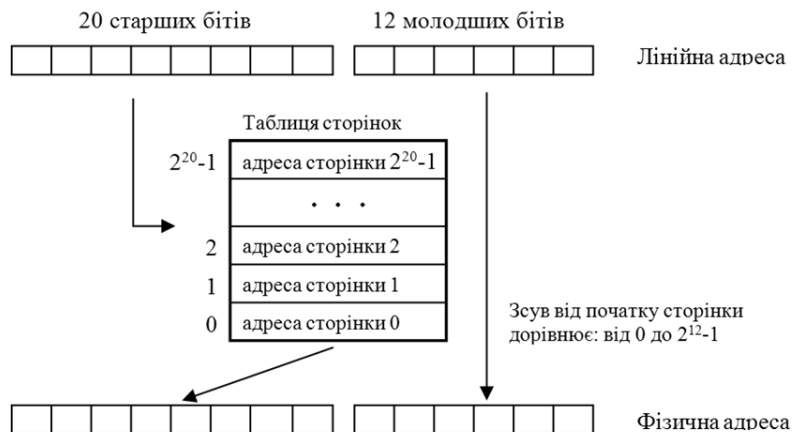
Процесор обов'язково використовує якусь форму сегментації. На відміну цього сторінкова адресація є необов'язковою, опціональною.

## Сегментація пам'яті та сторінкова адресація



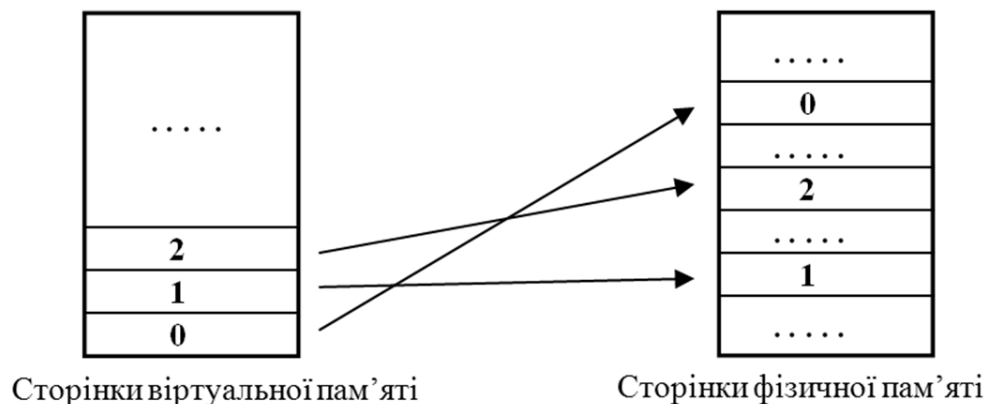
## Сторінкова адресація та віртуальна пам'ять

Сторінкова адресація виконується так. Весь адресний простір лінійних адрес в 4 Гбайти розділяється на окремі сторінки розміром 4096 ( $2^{12}$ ) байтів кожна. Лінійна адреса розщеплюється на дві частини. Одна частина складає 12 молодших розрядів і є зсувом в межах одної сторінки. Старші 20 розрядів використовуються як індекс в таблицях сторінок. В таблицях сторінок зберігаються фізичні адреси сторінок в ОЗП. Спрощено (насправді в процесорах використовується декілька таблиць і вибір фізичної адреси є багатоступінним) сторінкову адресацію можна уявити так:



## Сторінкова адресація та віртуальна пам'ять

Таким чином досягається відображення розташування сторінок віртуальній пам'яті на фактичне розташування в фізичній пам'яті. Сторінкова адресація використовується для логічного переміщення блоків пам'яті. Завдяки чому можна ефективно робити дефрагментацію пам'яті без фактичного переміщення блоків інформації в фізичній пам'яті.



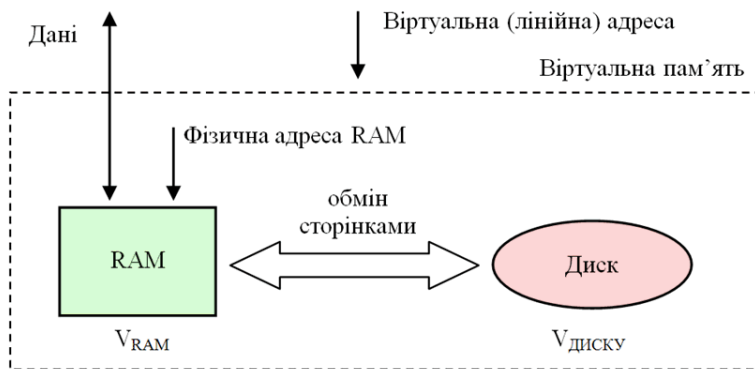
## Сторінкова адресація та віртуальна пам'ять

Сторінкова адресація використовується для побудови віртуальної пам'яті, об'єм котрої може бути значно більшим, ніж об'єм фізичної пам'яті (RAM). Для цього використовується дискова пам'ять. Робота пам'яті організовується так, щоб в ОЗП знаходилися тільки ті сторінки, котрі зараз використовуються. Тимчасово не потрібні сторінки записуються на диск. Як тільки будь-яка програма потребує дані, що записані в сторінках на диску, ці сторінки завантажуються з диска в ОЗП. В таблицях сторінок є поле, яке указує – в RAM чи на диску розташована кожна сторінка. Такий обмін з диском для організації віртуальної пам'яті виконує процесор під керуванням операційної системи.

Прикладна програма зазвичай не знає, де саме розташовані зараз її дані – в RAM чи на диску.

Програма використовує дані так, нібито вони усі розташовані в одному суцільному RAM великого об'єму

$$V_{\text{ВІРТ}} = V_{\text{RAM}} + V_{\text{Диску}}$$



Примітка. Чим більша доля дискової пам'яті, тим повільніше працює віртуальна пам'ять загалом, оскільки доступ до диска значно повільніший, аніж до RAM. Звідти й рекомендації – щоб прискорити роботу комп'ютера, потрібно збільшувати обсяг RAM.




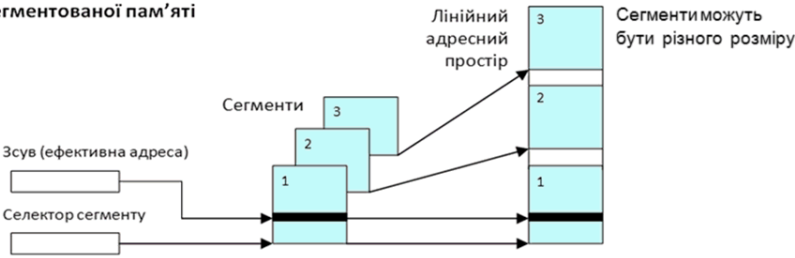
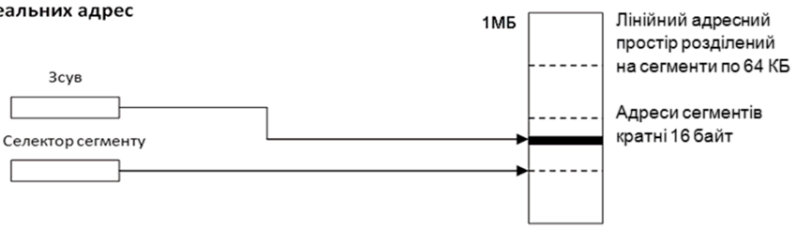
## Моделі пам'яті

Програма, яка використовує сервіс процесора щодо системи керування пам'яттю, не може безпосередньо адресувати фізичну пам'ять.

Для доступу до пам'яті використовується одна з трьох моделей пам'яті:

- модель **flat**
- модель **сегментованої пам'яті**
- модель **реальної адресації**

## Моделі пам'яті

<p><b>Flat - модель</b></p> 	
<p><b>Модель сегментованої пам'яті</b></p> 	<p>У цих двох моделях може також використовуватися сторінкова переадресація (paging), яка використовується для організації віртуальної пам'яті</p> <p>Якщо сторінкова переадресація вимкнена, то лінійна адреса ідентична фізичній адресі</p>
<p><b>Модель реальних адрес</b></p> 	<p>У цій моделі лінійна адреса ідентична фізичній адресі</p>

## Модель реальної адресації

Модель реальної адресації (мовою оригіналу *real-address mode memory model*) – історично це перша модель пам'яті, яка використовувалася у мікропроцесорах Intel сімейства x86. Ця модель пам'яті була прийнята для процесора Intel 8086. Наступні процесори підтримують цю модель заради сумісності програмного забезпечення.

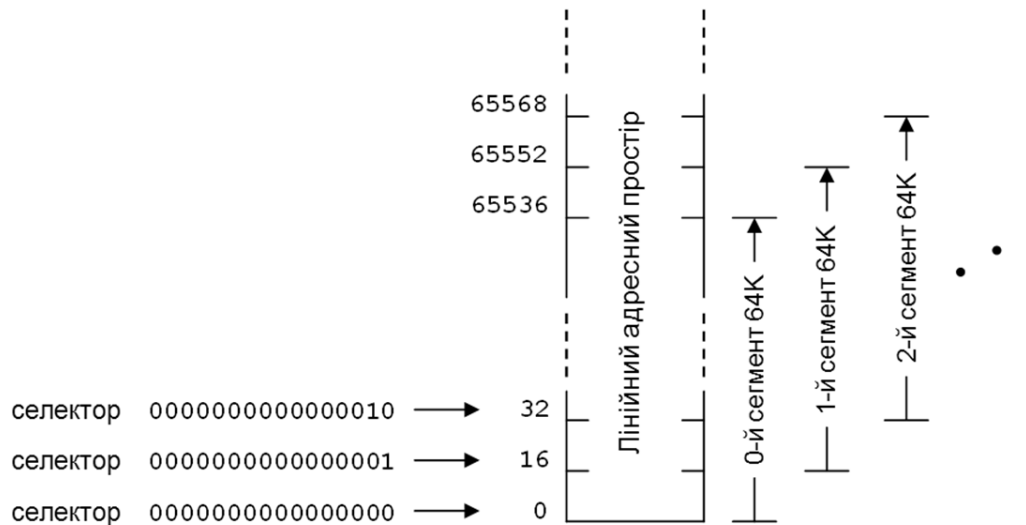
Від початку для архітектури x86 був прийнятий обсяг фізичної пам'яті 1МБ. А для цього шина адреси повинна бути 20-бітовою.

**Основна ідея моделі реальної адресації:** забезпечення можливості формування 20-бітової фізичної адреси 16-бітовими процесорами (у яких регістри, відповідно, є 16-бітовими).

Для доступу до пам'яті у програмах вказується логічна адреса у вигляді пари 16-бітних значень **селектор : зсув**. Селектором є лінійна адреса сегменту розділена на 16.

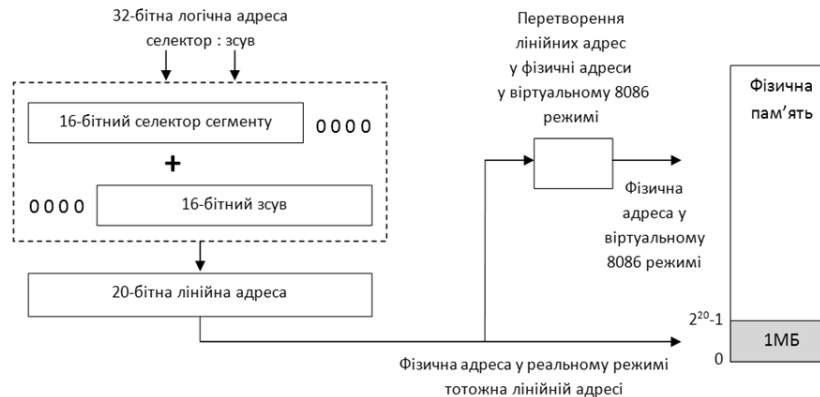
## Модель реальної адресації

Оскільки кожний селектор є 16-бітним, то загалом є  $2^{16}$  сегментів, початкові адреси сегментів відрізняються на 16 байт – від 0 до  $16 \cdot (2^{16}-1)$ . Оскільки зсув є 16-бітним, то розмір кожного сегменту 64K байтів. Таким чином, сегменти із сусідніми селекторами перекриваються.



## Модель реальної адресації

Процесор перетворює запис **селектор : зсув** у лінійну адресу дуже просто. 16-бітний селектор зсувається вліво на чотири розряди (що означає множення на 16) і додає зсув. Так обчислюється 20-бітна лінійна адреса – кожна програма може адресувати до 1Мбайт пам'яті.



У **реальному режимі** роботи процесора лінійна адреса є фізичною адресою, яка видається на адресну шину пам'яті.

У **віртуальному 8086 режимі** забезпечується можливість одночасного функціонування декількох програм, розроблених для реального режиму, тому фізична пам'ять для них розділюється.

## Модель сегментованої пам'яті

Модель сегментованої пам'яті (*segmented memory model*). Програмі надається пам'ять у вигляді декількох незалежних адресних просторів, які зветься сегментами. Код, дані та стек зазвичай розподіляються по окремим сегментам. Для доступу до байтів сегментованої пам'яті програми вказують **логічні адреси** у вигляді двох компонент – селектора та зсуву. Селектор вказує сегмент, а зсув означає відстань потрібного байту від початку сегменту.

Для процесорів архітектури IA-32 у захищеному режимі надається можливість створення до  $2^{14} = 16384$  сегментів розміром до  $2^{32}$  байтів кожний. Таким чином, **теоретично** можна адресувати до  $2^{14} \times 2^{32} = 2^{46} = 64$  Тбайт пам'яті.

Усі сегменти, які визначені у системі, відображаються на лінійний адресний простір. При доступі до пам'яті процесор перетворює логічну адресу у лінійну адресу – таке перетворення адрес відбувається прозоро та непомітно для програм.

Основний сенс використання сегментованої моделі пам'яті полягає у тому, щоб запобігти пошкодженню коду, даних або стеку, особливо при функціонуванні декількох програм.

Див. також вище: Сегментація пам'яті та сторінкова адресація

## Модель flat

Flat-модель пам'яті (*flat memory model*) – пласка, суцільна. Пам'ять надається програмі як єдиний безперервний адресний простір. Цей простір зветься **лінійним адресним простором**. Код, дані та стек разом усі містяться у цьому просторі. Діапазон адрес у лінійному адресному просторі від 0 до  $2^{32}-1$  (якщо це не 64-бітний режим). Адреса кожного байту у лінійному адресному просторі зветься **лінійною адресою**.

У моделі flat для адресації пам'яті програміст використовує тільки одне число – зсув. Оскільки у архітектурі x86 сегментні регістри так чи інакше використовуються завжди, то не можна казати, що сегментів тут немає. Це так здається програмісту, а реалізоване так: базові адреси сегментів даних, коду та стеку однакові та ці сегменти мають максимальний розмір. Це забезпечує операційна система, створюючи відповідні таблиці дескрипторів.

Таким чином, для 32-бітної flat-моделі пам'яті у програмах можна адресувати до  $2^{32}$  байтів = 4 Гбайт пам'яті.

У 64-бітному режимі модель flat є основною моделлю адресації пам'яті. Забезпечується 64-бітовий лінійний адресний простір.

Популярність такої моделі пам'яті значною мірою обумовлюється більшою **швидкістю доступу до пам'яті**, порівняно із сегментованою моделлю.

### 3.3.4 Modes of Operation vs. Memory Model

When writing code for an IA-32 or Intel 64 processor, a programmer needs to know the operating mode the processor is going to be in when executing the code and the memory model being used. The relationship between operating modes and memory models is as follows:

- **Protected mode** — When in protected mode, the processor can use any of the memory models described in this section. (The real-addressing mode memory model is ordinarily used only when the processor is in the virtual-8086 mode.) The memory model used depends on the design of the operating system or executive. When multitasking is implemented, individual tasks can use different memory models.
- **Real-address mode** — When in real-address mode, the processor only supports the real-address mode memory model.
- **System management mode** — When in SMM, the processor switches to a separate address space, called the system management RAM (SMRAM). The memory model used to address bytes in this address space is similar to the real-address mode model. See Chapter 34, “System Management Mode,” in the *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 3C*, for more information on the memory model used in SMM.
- **Compatibility mode** — Software that needs to run in compatibility mode should observe the same memory model as those targeted to run in 32-bit protected mode. The effect of segmentation is the same as it is in 32-bit protected mode semantics.
- **64-bit mode** — Segmentation is generally (but not completely) disabled, creating a flat 64-bit linear-address space. Specifically, the processor treats the segment base of CS, DS, ES, and SS as zero in 64-bit mode (this makes a linear address equal an effective address). Segmented and real address modes are not available in 64-bit mode.

(з документації Intel)



# Регістри процесора

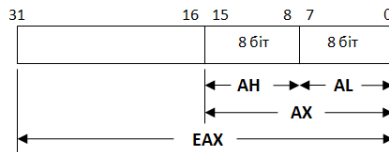
## Регістри загального призначення в архітектурі IA -32

До них відносяться 32-бітні регістри EAX, EBX, ECX, EDX, а також ESI, EDI, EBP, ESP.

Молодші 16-бітні половинки цих регістрів отримали власні назви (AX, BX, CX, DX, SI, DI, BP, SP) і можуть вказуватися та використовуватися індивідуально.

У 16-бітних регістрів AX, BX, CX, DX можна окремо використовувати також і 8-бітні половинки: старші – AH, BH, CH, DH, та молодші – відповідно AL, BL, CL, DL.

31	16	15	8	7	0	32 - бітні	16 - бітні	8 - бітні
		AH		AL		EAX	AX	AH, AL
		BH		BL		EBX	BX	BH, BL
		CH		CL		ECX	CX	CH, CL
		DH		DL		EDX	DX	DH, DL
		SI				ESI	SI	
		DI				EDI	DI	
		BP				EBP	BP	
		SP				ESP	SP	



Індивідуальні назви частин регістра EAX

## Регістри загального призначення в архітектурі IA -32

Не зважаючи на те, що вказані вище регістри названо регістрами загального призначення, не усі з них можна вільно використовувати без обмежень.

Регістри EAX, EBX, ECX, EDX можна вважати найбільш універсальними – програміст може широко використовувати їх для різноманітних цілей. Проте, потрібно враховувати, що деякі команди процесора зроблені так, що їм можна передавати вхідні дані тільки у регістри з певними іменами. Деякі команди записують результати тільки у відповідні регістри. Наприклад, результат роботи команди MUL завжди записується у регістри EAX та EDX. Для таких команд, як MOVS кількість слів завжди вказується у регістрі ECX. Таких прикладів можна навести ще багато.

Решта регістрів - ESI (індекс, адреса джерела), EDI (індекс, адреса призначення), EBP (вказівник бази стеку), ESP (вказівник стеку) – мають ще більш спеціальне призначення. Так, наприклад, регістри ESI и EDI необхідні у операціях з рядками даних, EBP и ESP - при роботі зі стеком.

Необхідно відмітити, що невеличка кількість регістрів загального призначення, наявних у архітектурі IA-32, є проблемою для програмістів на Асемблері та компіляторів мов високого рівня.

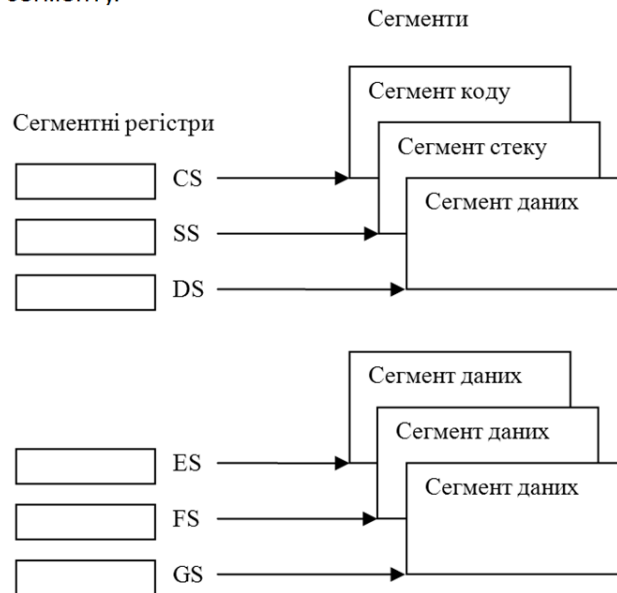
## Сегментні регістри

Ці регістри призначені для зберігання селекторів сегментів пам'яті.

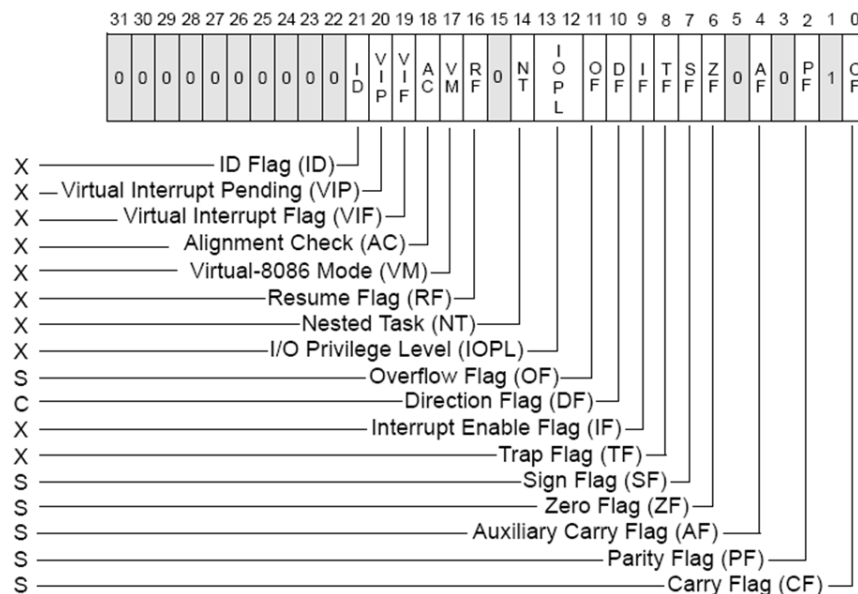
У сегментованих моделях адреса представляється у вигляді двох чисел – базової адреси сегменту та зсуву від початку сегменту.

Для вказування адрес сегментів призначені 16-бітні регістри CS, DS, ES, FS, GS, SS.

Ці регістри зберігають не самі адреси сегментів а, так звані, селектори. Значення селекторів процесор сам перетворює у базові адреси сегментів. Як перетворює – залежить від режиму роботи процесора відповідно моделі адресації пам'яті.



## Регістр EFLAGS



Кожний біт цього регістру позначає стан чогось у вигляді “так” (1) чи “ні” (0). Такі біти називаються “прапорцями”.

- S Indicates a Status Flag
- C Indicates a Control Flag
- X Indicates a System Flag

Reserved bit positions. DO NOT USE.  
Always set to values previously read.

(з документації фірми Intel)

## Регістр EFLAGS. Прапорці статусу

Прапорці статусу (біти 0, 2, 4, 6, 7 та 11) відображають результати виконання арифметичних команд – таких як ADD, SUB, MUL, DIV. Означають наступне:

**CF** (біт 0) – прапорець переносу (Carry Flag). Встановлюється у 1, якщо виникає перенос або потрібне позичання (наприклад, при відніманні) для старшого розряду результату арифметичної операції; інакше біт обнулюється. Цей біт вказує на переповнення розрядної сітки арифметики. Також використовується у арифметиці підвищеної розрядності (точності).

**PF** (біт 2) – прапорець парності (Parity Flag). Встановлюється у 1, якщо молодший байт результату має парну кількість одиничних бітів; інакше біт обнулюється.

**AF** (біт 4) – прапорець корекції (Adjust Flag). Встановлюється у 1, якщо виникає перенос або потрібне позичання у 4-му біті результату; інакше біт обнулюється. Це використовується для двійково-десятькової (BCD) арифметики.

**ZF** (біт 6) – прапорець нуля (Zero Flag). Встановлюється у 1, якщо результат є нуль; інакше біт обнулюється.

**SF** (біт 7) – прапорець знаку (Sign Flag). Повторює старший біт результату, який є знаковим для цілого числа результату. 0 означає позитивне, 1 – від'ємне значення.

**OF** (біт 11) – прапорець переповнення (Overflow Flag). Встановлюється у 1, якщо результат є занадто великим позитивним числом або замалим від'ємним числом; інакше біт обнулюється. Цей біт показує переповнення розрядності для арифметики цілих зі знаком у додатковому коді.

Серед бітів статусу тільки біт CF може бути прямо змінений – командами STC (CF=1), CLC (CF=0) та CMC (інверсія CF). Також команди BT, BTS, BTR, BTC копіюють визначений біт у CF.

## Регістр EFLAGS. Прапорець DF

Прапорець DF (**Direction flag** – біт 10 регістру EFLAGS) вказує напрямок обробки рядків командами MOVS, CMPS, SCAS, LODS, STOS.

Якщо DF=0, то обробка йде із збільшенням адрес, якщо DF=1, то обробка рядків йде навпаки - від більших до менших адрес.

Для вказування значення прапорця DF передбачено команди:

- STD (встановлюється DF = 1)
- CLD (DF = 0)

## Регістр EFLAGS. Системні прапорці

Деякі біти регістру EFLAGS керують операціями привілейованих (системних) програм і не можуть бути змінені звичайними прикладними програмами. До них відносяться:

**TF** (біт 8) – прапорець налагодження (Trap Flag).

**IF** (біт 9) – прапорець дозволу переривання (Interrupt Enable Flag).

**IOPL** (біти 12, 13) – поле рівня привілеїв (I/O Privilege Level Field).

**NT** (біт 14) – прапорець вкладеності виконання завдань (Nested Task Flag).

**RF** (біт 16) – прапорець (Resume Flag).

**VM** (біт 17) – прапорець (Virtual-86 Mode Flag).

**AC** (біт 18) – прапорець (Alignment Check Flag).

**VIF** (біт 19) – прапорець (Virtual Interrupt Flag).

**VIP** (біт 20) – прапорець (Virtual Interrupt Pending Flag).

**ID** (біт 21) – прапорець (Identification Flag).



## Регістр EIP

Регістр EIP зберігає вказівник команд (*Instruction Pointer*). Цей вказівник означає адресу наступної команди (точніше кажучи, її зсув відносно початку поточного сегменту коду). Запис значень у цей регістр робиться автоматично процесором при виконання переходів, викликів процедур та повернення з процедур.

Програміст не може безпосередньо отримати доступ до регістру EIP, наприклад, намагатися прямо записати якесь значення:

```
mov eip, 200      ;помилка, так не можна
```

Так само не можна прямо прочитати вміст регістру EIP, наприклад:

```
mov eax, eip      ;помилка, так не можна
```

При компіляції таких рядків буде повідомлення про помилку – компілятор забороняє будь-яке вказування EIP у якості операнду команд.