**1. Адресація операндів**

**опера́нд** — аргумент [операції](https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)); дані, які обробляються командою; граматична конструкція, яка позначає вираз, що задає значення аргументу операції; іноді операндом називають місце або позицію в тексті, де має стояти аргумент операції.

Наступний вираз є прикладом операції та операндів.

3+6=9У цьому прикладі "+" це символ операції яка називається [додавання](https://uk.wikipedia.org/wiki/%D0%94%D0%BE%D0%B4%D0%B0%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F" \o "Додавання).

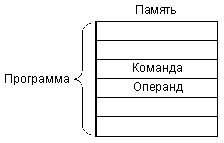
Операнд  3 є одним із вхідних величин з наступним додаванням [оператора](https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), а операнд  6  це інша, необхідна для роботи, вхідна величина.

Результат операції є  9. (Число "9" також називається сумою доданків "3" та "6".)

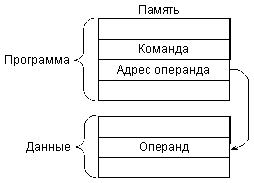
**3.1.1. Методи адресації**

Кількість методів адресації в різних процесорах може бути від 4 до 16. Розглянемо кілька типових методів адресації операндів, використовуваних зараз у більшості мікропроцесорів.

**Безпосередня адресація** (Рис. 3.1) припускає, що операнд (вхідний) знаходиться в пам'яті безпосередньо за кодом команди. Операнд звичайно являє собою константу, яку треба кудись переслати, до чого додати і т.д. Наприклад, команда може полягати в тому, щоб додати число 6 до вмісту якогось внутрішнього регістра процесора. Це число 6 буде розташовуватися в пам'яті, усередині програми в адресі, що слідує за кодом даної команди додавання.

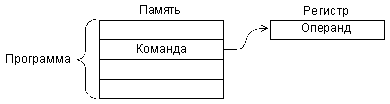
  
**Рис. 3.1.** Безпосередня адресація.

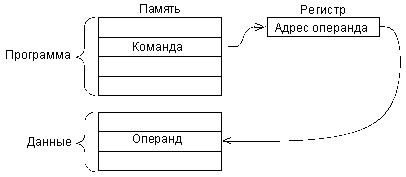
**Пряма (вона ж абсолютна) адресація** (мал. 3.2) припускає, що операнд (вхідний чи вихідний) знаходиться в пам'яті за адресою, код якого знаходиться усередині програми відразу ж за кодом команди. Наприклад, команда може полягати в тому, щоб очистити (зробити нульовим) уміст комірки пам'яті з адресою 1000000. Код цієї адреси 1000000 буде розташовуватися в пам'яті, усередині програми в наступній адресі за кодом даної команди очищення.

  
**Рис. 3.2.** Пряма адресація.

**Регістрова адресація** (Рис. 3.3) припускає, що операнд (вхідний чи вихідний) знаходиться у внутрішньому регістрі процесора. Наприклад, команда може полягати в тому, щоб переслати число з нульового регістра в перший. Номера обидвох регістрів (0 і 1) будуть визначатися кодом команди пересилання.

**Непрямо-реєстрова (вона ж непряма)** адресація припускає, що у внутрішньому регістрі процесора знаходиться не сам операнд, а його адреса в пам'яті (Рис. 3.4). Наприклад, команда може полягати в тому, щоб очистити комірку пам'яті з адресою, що знаходиться в нульовому регістрі. Номер цього регістра (0) буде визначатися кодом команди очищення.

  
**Рис. 3.3.** Регістрова адресація.

  
**Рис. 3.4.** Косвенная адресация.

Рідше зустрічаються ще два методи адресації.

**Автоінкрементна адресація** дуже близька до непрямої адресації, але відрізняється від неї тим, що *після* виконання команди уміст використовуваного регістра збільшується на одиницю чи на два. Цей метод адресації дуже зручний, наприклад, при послідовній обробці кодів з масиву даних, що знаходиться в пам'яті. Після обробки якогось коду адреса в регістрі буде вказувати вже на наступний код з масиву. При використанні непрямої адресації в даному випадку довелося б збільшувати вміст цього регістра окремою командою.

**Автодекрементна адресація** працює подібно до автоінкрементної, але тільки вміст обраного регістра зменшується на одиницю чи на два *перед* виконанням команди. Ця адресація також зручна при обробці масивів даних. Спільне використання автоінкрементної та автодекрементної адресацій дозволяє організувати пам'ять стекового типу (див. розділ 2.4.2).

З інших розповсюджених методів адресації можна згадати про індексні методи, що припускають для обчислення адреси операнда додавання до вмісту регістра заданої константи (індексу). Код цієї константи розташовується в пам'яті безпосередньо за кодом команди.

Відзначимо, що вибір того чи іншого методу адресації в значній мірі визначає час виконання команди. Найшвидша адресація - це регістрова, тому що вона не вимагає додаткових циклів обміну магістраллю. Якщо ж адресація вимагає звертання до пам'яті, то час виконання команди буде збільшуватися за рахунок тривалості необхідних циклів звертання до пам'яті. Зрозуміло, що чим більше внутрішніх регістрів у процесора, тим частіше і вільніше можна застосовувати регістрову адресацію, і тем швидше буде працювати система в цілому.

#### Сегментування пам'яті.

Говорячи про адресацію, не можна обійти питання про **сегментування** пам'яті, яке застосовується в деяких процесорах, наприклад у процесорах IBM PC-сумісних персональних комп'ютерів.

У процесорі Intel 8086 сегментування пам'яті організоване в такий спосіб:

Уся пам'ять системи представляється не у виді безупинного простору, а у виді декількох кусків - сегментів заданого розміру (по 64 Кбайти), положення яких у просторі пам'яті можна змінювати програмним шляхом.

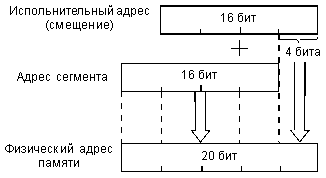
Для збереження кодів адрес пам'яті використовуються не окремі регістри, а пари регістрів:

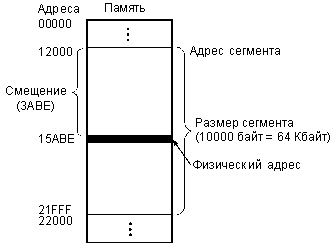
* **сегментний регістр** визначає адресу початку сегменту (тобто місце сегмента в пам'яті);
* **регістр показника** (регістр зсуву) визначає положення робочої адреси усередині сегмента.

При цьому фізична 20-розрядна адреса пам'яті, яка виставляється на зовнішню шину адреси, утвориться так, як показано на Рис. 3.5, тобто шляхом додавання зсуву й адреси сегмента зі зсувом на 4 біти. Положення цієї адреси в пам'яті показане на Рис. 3.6.

Сегмент може починатися тільки на 16-байтній межі пам'яті (тому що адреса початку сегменту, по суті, має чотири молодших нульових розряди, як видно з Рис. 3.5), тобто з адреси, яка кратна 16. Ці припустимі межі сегментів називаються границями параграфів.

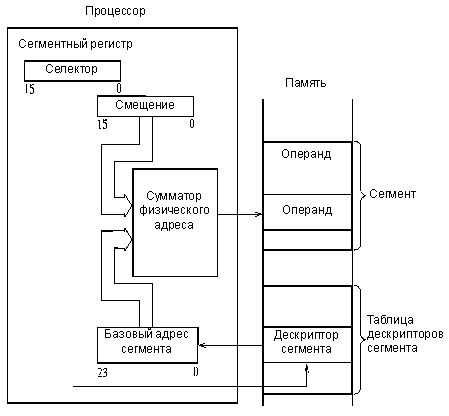
Відзначимо, що ввід сегментування, насамперед, зв'язане з тим, що внутрішні регістри процесора 16-розрядні, а фізична адреса пам'яті 20-розрядний (16-розрядна адреса дозволяє використовувати пам'ять тільки в 64 Кбайт, що явно недостатньо). У процесорі, який з'явився в той самий час, MC68000 фірми Motorola внутрішні регістри 32-розрядні, тому там проблеми сегментування пам'яті не виникає.

  
**Рис. 3.5.** Формування фізичної адреси пам'яті з адреси сегмента і зсуву.

  
**Рис. 3.6.** Фізична адреса в сегменті (усі коди - шіснадцяткові).

Застосовуються і більш складні методи сегментування пам'яті. Наприклад, у процесорі Intel 80286 у так званому захищеному режимі адреса пам'яті обчислюється відповідно до Рис. 3.7.

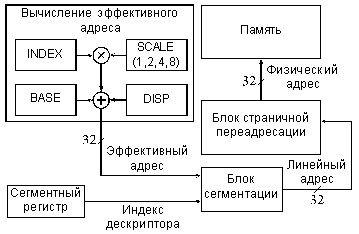
У сегментному регістрі в даному випадку зберігається не базова (початкова) адреса сегментів, а коди селекторів, що визначають адреси в пам'яті, по яких зберігаються дескриптори (тобто описи) сегментів. Область пам'яті з дескрипторами називається таблицею дескрипторів. Кожен дескриптор сегмента містить базову адресу сегмента, розмір сегмента (від 1 до 64 Кбайт) і його атрибути. Базова адреса сегмента має розрядність 24 біт, що забезпечує адресацію 16 Мбайт фізичної пам'яті.

  
**Рис. 3.7.** Адресація пам'яті в захищеному режимі процесора Intel 80286.

Таким чином, на суматор, що обчислює фізичну адресу пам'яті, подається не вміст сегментного регістра, як у попередньому випадку, а базова адреса сегмента з таблиці дескрипторів.

Ще складніший метод адресації пам'яті із сегментуванням використаний у процесорі Intel 80386 і в більш пізніх моделях процесорів фірми Intel. Цей метод ілюструється Рис. 3.8.

Адреса пам'яті (фізична адреса) обчислюється в три етапи. Спочатку обчислюється так звана ефективна адреса (32-розрядна) шляхом підсумовування трьох компонентів: бази, індексу і зсуву (Base, Index, Displacement), причому можливе множення індексу на масштаб (Scale). Ці компоненти мають наступний зміст:

  
**Рис. 3.8.** Формування фізичної адреси пам'яті процесора 80386 у захищеному режимі.

* *зсув* - це 8-, 16- чи 32-розрядне число, включене у команду.
* *база* - це вміст базового регістра процесора. Звичайно воно використовується для вказівки на початок деякого масиву.
* *індекс* - це вміст індексного регістра процесора. Переважно воно використовується для вибору одного з елементів масиву.
* *масштаб* - це множник (він може бути рівним 1, 2, 4 чи 8), зазначений у коді команди, на який перед підсумовуванням з іншими компонентами збільшується індекс. Він використовується для вказівки розміру елемента масиву.

Потім спеціальний блок сегментації обчислює 32-розрядну *лінійну адресу*, що представляє собою суму базової адреси сегмента із сегментного регістра з ефективною адресою. Нарешті, фізична 32-бітна адреса пам'яті утвориться шляхом перетворення лінійної адреси блоком сторінкової переадресації, що здійснює переклад лінійної адреси у фізичний сторінками по 4 Кбайти.

У будь-якому випадку сегментування дозволяє виділити в пам'яті один чи кілька сегментів для даних і один чи кілька сегментів для програм. Перехід від одного сегмента до іншого зводиться усього лише до зміни вмісту сегментного регістра. Іноді це буває дуже зручно. Але для програміста працювати із сегментованою пам'яттю звичайно складніше, ніж з безупинною, несегментованою пам'яттю, тому що приходиться стежити за межами сегментів, за їх описом, перемиканням і т.д.