

## ОСНОВИ АЛГОРИТМІЗАЦІЇ ТА ПРОГРАМУВАННЯ

### Будова комп'ютерних систем

Перші комп'ютери (ЕОМ) були переважно аналоговими, тобто, опрацьовували неперервні сигнали. Вони могли виконувати лише *обмежений набір завдань*. Перейти до виконання інших завдань можна було лише *змінивши будову таких комп'ютерів*.

Це було незручно і не вигідно. Джон фон Нейман здійснив революцію в галузі комп'ютерної техніки: він першим запропонував, що комп'ютери мають бути *універсальними* для того, щоб мати можливість *виконувати різні завдання*. Тобто, комп'ютери виконують програму (набір команд), яка описує алгоритм виконання завдання. Для того, щоб виконати інше завдання, не потрібно міняти будову комп'ютера – достатньо завантажити іншу програму.

### Принципи Джона фон Неймана

У 1946 році Джон фон Нейман опублікував звіт «Попереднє обговорення логічного конструювання електронного обчислювального пристрою», в якому сформулював чотири принципи архітектури комп'ютерів. *Всі сучасні комп'ютери побудовані за цими принципами:*

#### 1) використання двійкової системи числення для кодування інформації;

Комп'ютер – електронний пристрій, побудований на простих принципах «є сигнал (1) – немає сигналу (0)»; тому обрано двійкову систему числення для подання інформації. Один двійковий розряд може містити значення 0 або 1. Такий розряд називається *біт* (binary digit – англ. «двійковий розряд»).

В одному біті можна записати  $2^1 = 2$  значення (0 та 1).

В двох бітах можна записати  $2^2 = 4$  значення (00, 01, 10, 11).

В трьох бітах можна записати  $2^3 = 8$  значень (000, 001, 010, 011, 100, 101, 110, 111).

В вісьмох бітах можна записати  $2^8 = 256$  значень.

Сукупність 8 бітів називається *байт* (byte – англ. «шматочок»). В різних комп'ютерних системах в різні часи байт складався з різної кількості бітів – і 7, і 9. В наш час найбільш поширеними є комп'ютерні системи, в яких байт складається саме із 8 бітів.

1 кб = 1 кілобайт = 1024 байти =  $2^{10}$  байт

1 Мб = 1 мегабайт = 1024 кілобайти =  $2^{20}$  байт

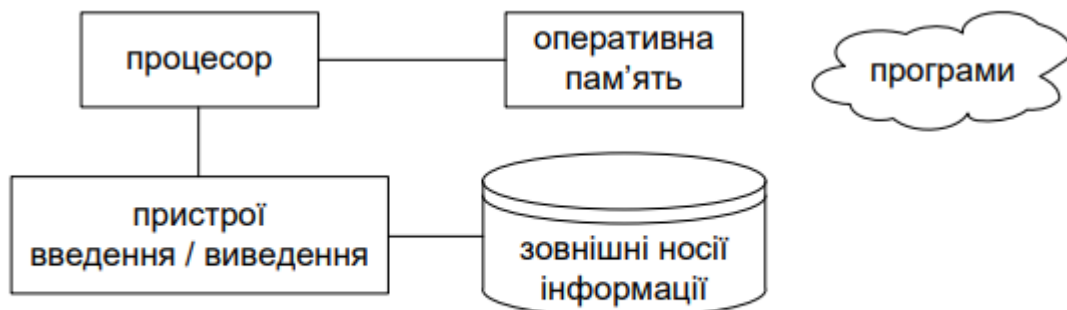
1 Гб = 1 гігабайт = 1024 мегабайти =  $2^{30}$  байт

1 Тб = 1 терабайт = 1024 гігабайти =  $2^{40}$  байт

1 Пб = 1 петабайт = 1024 терабайти =  $2^{50}$  байт

## 2) програмне керування роботою комп'ютера;

Комп'ютер – сукупність технічних та програмних засобів для автоматизованого опрацювання дискретних даних відповідно до заданого алгоритму. Має *фізичну* (скупність певних з'єднаних між собою пристроїв та деталей) та *програмну* складові, а ще і набір відповідних програм.



*Складові частини комп'ютера*

### *Комп'ютер опрацьовує дискретні дані*

Перші комп'ютери опрацьовували інформацію, яка подавалася саме *неперервними* (тобто, аналоговими) сигналами. Оскільки використовується двійкова система числення та обмежений обсяг пам'яті для представлення даних, то цифрові комп'ютери можуть опрацьовувати лише *дискретні* дані. Для того щоб опрацьовувати неперервні дані, їх потрібно *дискретизувати* – тобто, замінити дискретними даними, які (приблизно) співпадають в точках сітки значень з початковими неперервними даними.

### *Комп'ютер працює відповідно до заданого алгоритму.*

Програми, які виконує комп'ютер – це машинне представлення запису конкретного алгоритму. Комп'ютер не може виконати нічого, що не входить до відповідного алгоритму.

## 3) зберігання програм у пам'яті комп'ютера;

Модель пам'яті

- *Принстонська* (університету м. Принстон): пам'ять однорідна – команди та дані зберігаються в одних і тих самих областях пам'яті. Це повільніше, але дешевше.

- Гарвардська ( університету м. Гарвард): команди і дані використовують конструктивно різні види пам'яті. Це швидше, але дорожче

#### 4) адресація пам'яті

Кожний байт має свою адресу. Це дозволяє звертатися до окремих байтів пам'яті для того, щоб записати / прочитати певне значення. Найменша комірka пам'яті, яка може мати свою адресу – байт. Інші (похідні) одиниці вимірювання розміру пам'яті:

$$1 \text{ Кб} = 1 \text{ кілобайт} = 1024 \text{ байти} = 2^{10} \text{ байт}$$

$$1 \text{ Мб} = 1 \text{ мегабайт} = 1024 \text{ кілобайти} = 2^{20} \text{ байт}$$

$$1 \text{ Гб} = 1 \text{ гігабайт} = 1024 \text{ мегабайти} = 2^{30} \text{ байт}$$

$$1 \text{ Тб} = 1 \text{ терабайт} = 1024 \text{ гігабайти} = 2^{40} \text{ байт}$$

$$1 \text{ Пб} = 1 \text{ петабайт} = 1024 \text{ терабайти} = 2^{50} \text{ байт}$$

#### Архітектура комп'ютерів

Вперше означення терміну “архітектура комп'ютера” було зроблене в 1964 році розробниками комп'ютера IBM 360 Г. Амдалем та його колегами. З їхньої точки зору, *архітектура комп'ютера* – це його структура і поведінка як їх бачить програміст на мові асемблер. Вона включає: *формати даних і команд, методи адресації, систему команд, а також загальну організацію процесора, основної пам'яті і пристроїв введення-виведення.*

Пізніше Пейджез запропонував: *архітектура комп'ютера – інтерфейс між його апаратним та програмним забезпеченням*

Команда в комп'ютері зберігається у двійковій формі. Вона вказує *тип операції*, яка має бути виконаною, *адреси операндів*, над якими виконується операція, та *адреси розміщення результатів* виконання операції. Відповідно до цього команда складається з двох частин: *коду операції та адресної частини.*

В комп'ютері використовується *двійкове подання команд*. При написанні програми, крім двійкової, можуть використовуватись і інші форми подання команд: вісімкова, шістнадцяткова, символна (мнемонічна). Використання *вісімкового і шістнадцяткового кодування дає змогу скоротити записи і спростити роботу програміста.*

*Мнемонічне* кодування спрощує процес написання, читання і відлагодження програми. Основний принцип такого кодування – кожен команду подають 3-х або 4-х буквеним символом, який показує назву команди. Операнди також подають символно. Враховуючи, що операція виконується над вмістом, а не над адресою комірки пам'яті та регістра, то з'являється можливість написання машинних програм

в символній формі.

Повний набір символних назв і правила їх використання утворюють мову програмування *асемблер*. Запис певної програми на мові асемблер є символним записом цієї ж програми, написаної на внутрішній мові комп'ютера, тобто у формі послідовності команд, поданих у двійкових кодах. Тому і з'явилося визначення архітектури комп'ютера як інтерфейсу між його апаратним та програмним забезпеченням.

Крім коду операції до складу команди входить *адресна частина*. Цією частиною визначається місце знаходження даних, над якими виконується операція, задана кодом операції.

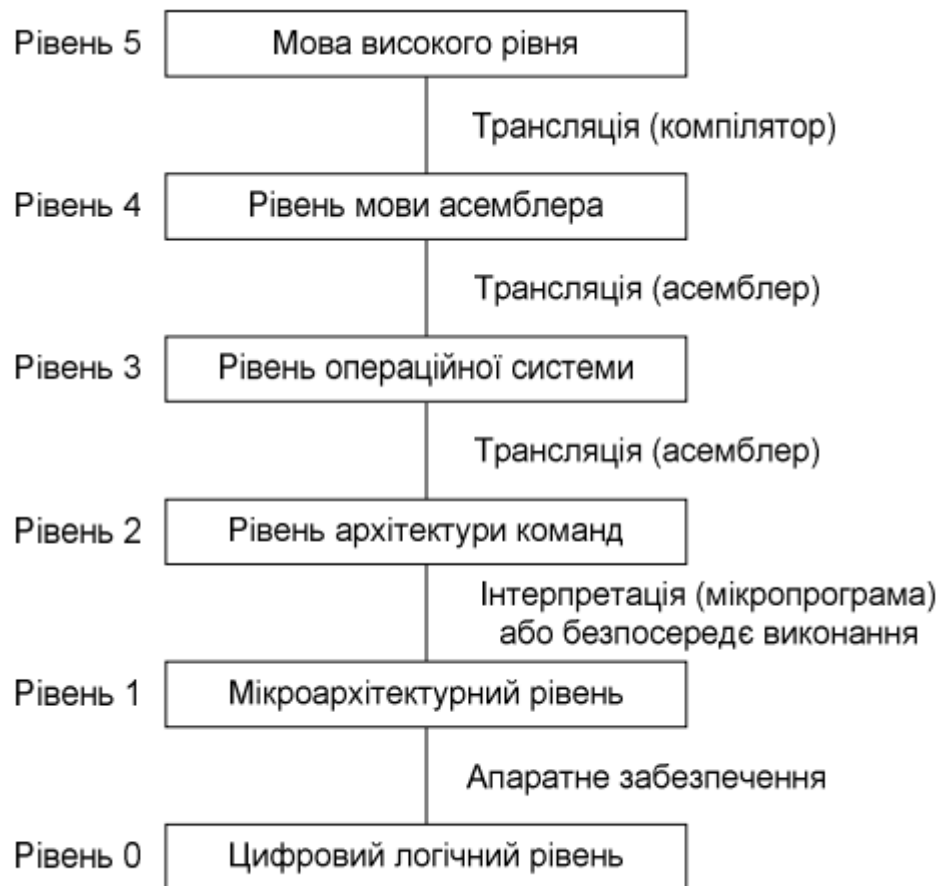
Даних може бути декілька, і, крім того, вони можуть знаходитися в основній пам'яті, в регістрах процесора чи в запам'ятовуючих елементах інших вузлів комп'ютера. Тому і формати команд в цих випадках будуть різними. Можна здійснити класифікацію архітектури комп'ютера за типом адресованої пам'яті.

Залежно від того, який тип пам'яті адресується, розрізняють наступні типи архітектур комп'ютера: *стекова, акумуляторна, на основі регістрів загального користування*.

При розробці комп'ютера необхідно враховувати багато особливостей вибору системи команд. З одного боку, система команд повинна бути функціонально повною, тобто комп'ютер повинен забезпечувати виконання всіх заданих функцій. З іншого боку, система команд має бути ортогональною, тобто не повинна бути надлишковою. Для нового комп'ютера, який є розширенням відповідної серії, першочерговою є *сумісність* – програми, які виконуються на одному комп'ютері, повинні виконуватись і на іншому комп'ютері.

*Архітектура комп'ютера має визначальний вплив на його споживчі характеристики: коло вирішуваних задач, продуктивність, ємність основної пам'яті, ємність зовнішньої пам'яті, вартість, організація технічного обслуговування, надійність тощо.*

*Рівні процесу опрацювання даних в сучасному комп'ютері за Е. Таненбаумом*



### *Рівні процесу опрацювання даних в сучасному комп'ютері*

Нижче нульового рівня є **рівень фізичних пристроїв** – на цьому рівні знаходяться *транзистори*.

На **цифровому логічному рівні** розташовані *вентилі*. У кожного вентиля є одно або декілька цифрових входних даних (сигналів, що подають 0 або 1). Вентиль обчислює прості функції цих сигналів, такі як І чи АБО.

На **мікроархітектурному рівні** є сукупності 8 або 32 регістрів, які формують локальну пам'ять та схему, яка називається **арифметично-логічним пристроєм** (АЛП). АЛП виконує прості арифметичні операції.

**Рівень архітектури системи команд** – описується індивідуально виробниками конкретних комп'ютерів. Описують команди, які виконуються мікропрограмою-інтерпретатором або апаратним забезпеченням.

**Рівень операційної системи**, як правило, гібридний. Більшість команд в його мові є також і на рівні архітектури системи команд (команди, що знаходяться на одному із рівнів, цілком можуть знаходитись на інших рівнях). У цього рівня є деякі додаткові особливості: набір нових команд, інша організація пам'яті, здатність виконувати дві і більше програм одночасно і деякі інші. Одна частина команд третього рівня інтерпретується операційною системою, а інша – мікропрограмою.

**Рівень мови асемблера** є символічною формою однієї з мов більш низького

рівня. На цьому рівні можна писати програми в зручній для людини формі. Ці програми спочатку транслюються на мову рівня 1, 2 або 3, а потім інтерпретуються відповідною віртуальною або фактично існуючою машиною. Програма, яка виконує трансляцію, називається асемблером.

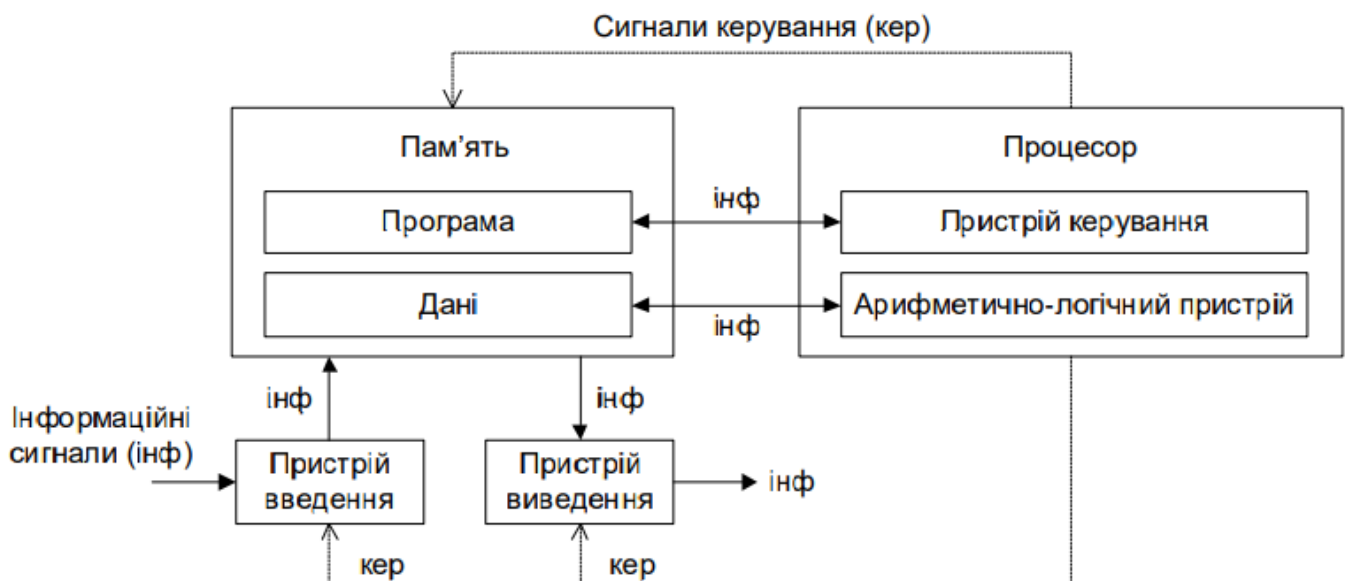
**П'ятий рівень**, як правило, складається із мов програмування, розроблених для прикладних програмістів – *мови високого рівня* ( C, C++, Java тощо).

### Архітектура фон Неймана

*Архітектура фон Неймана* – архітектура електронних обчислювальних машин (ЕОМ), головною характеристикою якої є спільне зберігання даних та машинних команд в комірках однієї й тієї ж пам'яті, що робить неможливим розрізнити дані та команди за способом представлення або кодування. Цей вид архітектури названий на честь відомого математика та теоретика обчислювальної техніки Джона фон Неймана. Найбільш поширена схема організації ЕОМ загального призначення.

*Обчислювальна машина є машиною з архітектурою фон Неймана, якщо:*

1. Програма та дані зберігаються в одній загальній пам'яті. Це дає можливість виконувати над командами ті ж дії, що і над даними.
2. Кожна комірка пам'яті машини ідентифікується унікальним номером, який називається адресою.
3. Різні слова інформації (команди та дані) розрізняються за способом використання, але не за способом кодування чи представленням в пам'яті.
4. Кожна програма виконується послідовно, починаючи з першої команди, якщо немає спеціальних вказівок. Для зміни цієї послідовності використовуються команди переходу.



*Архітектура комп'ютера згідно принципів Дж. фон Неймана*



## **Структура машини фон Неймана**

**1. Арифметико-логічний пристрій** – виконує команди з визначеного набору (системи команд) над порціями інформації, що зберігаються у відокремленій від операційного пристрою пам'яті. Сучасні архітектури мають в складі 14 операційного пристрою додаткову пам'ять (зазвичай банк регістрів), в якій операнди зберігаються порівняно короткий час безпосередньо в процесі проведення обчислень.

**2. Пристрій керування процесором** – організує послідовне виконання алгоритмів, розшифрування команд, які надходять із пристрою пам'яті (див. далі), реагує на аварійні ситуації та виконує загальні функції керування всіма вузлами обчислювальної машини. Зазвичай АЛП та пристрій керування об'єднуються в структуру, яка називається центральним процесором. Слід звернути увагу, що вимога саме послідовного, в порядку надходження з пам'яті (в порядку зміни адрес в лічильнику команд) виконання команд є принциповою. Архітектури, які не додержуються такого принципу, взагалі не вважаються фон-нейманівськими.

**3. Комп'ютерна пам'ять** – масив комірок з унікальними ідентифікаторами (адресами), в яких зберігаються команди та дані.

**4. Пристрої вводу/виводу**, які забезпечують зв'язок ЕОМ з зовнішнім світом, передають інформацію на переробку в ЕОМ та приймають результати.

### *Недоліки архітектури Дж. фон Неймана*

#### *Розділення операційного пристрою та пам'яті*

Швидкість обробки інформації в процесорі зазвичай є набагато більшою, ніж швидкість роботи пристрою пам'яті, який не встигає забезпечувати процесор новими порціями інформації, що призводить до простоїв. Проблема вирішується за рахунок побудови складнішої ієрархії пам'яті, зокрема введенням кеш-пам'яті (швидшої, але й дорожчої за основну). У кеш-пам'яті зберігаються дані, які часто використовуються в обчисленнях, що зменшує кількість звертань до повільнішої основної пам'яті.

#### *Послідовний принцип виконання*

Архітектура фон Неймана є принципово послідовною. І це є суттєвим обмежуючим фактором в підвищенні швидкодії машин з такою організацією, унеможлиблює введення явного паралелізму в систему. Майже всі ЕОМ загального призначення є фон-нейманівськими, вони суттєво використовують механізми розпаралелювання обчислень, хоча це відбувається й неявно, на рівні внутрішньої організації процесора, який непомітно для програміста виявляє схований паралелізм в послідовних програмах для фон-нейманівських машин.

## **Машинні та високорівневі мови програмування**

Послідовність команд, що описують вирішення певної задачі, називається **програмою**.

Кожна команда програми містить вказівки відповідно до певної виконуваної операції, місця розташування (адреси) операндів та низки службових ознак.

**Операнди** – це величини (змінні, константи), значення яких використовуються в операціях опрацювання даних. Список усіх змінних (початкових даних, проміжних значень та результатів обчислень) є ще одним необхідним елементом будь-якої програми.

**Мова програмування** – це формалізована мова, призначена для опису алгоритмів розв'язування задач на обчислювальних машинах. Мова програмування дає змогу записати алгоритм у термінах, зрозумілих компілятору, який перекладає його на зрозумілу для процесора машинну мову.

Будь-яка мова програмування складається з **алфавіту** (вживаних символів), **слів** (елементарних конструкцій, які називають лексемами), **словосполучень** (виразів) та **речень** (операторів). Лексеми утворюються з символів, вирази – з лексем та символів, а оператори – з символів, виразів та лексем.

Електронні схеми кожного комп'ютера можуть розпізнавати та виконувати обмежений набір простих команд.

У сукупності ці команди складають **машинну мову** – мову, якою люди спілкуються з комп'ютером. При створенні нового комп'ютера розробники вирішують, які саме команди включати в його машинну мову. Набір команд залежить від призначення комп'ютера і задач, які він повинен виконувати.

Існує велика різниця між тим, що зручно для людей, і тим, що зручно для комп'ютерів. Через це виникають різні проблеми, які вирішують за допомогою **компіляторів** та **інтерпретаторів**.

Термінами “компілятор” та “інтерпретатор” позначаються *способи виконання програми*. Теоретично програми на будь-якій мові програмування можуть бути або відкомпільовані, або інтерпретовані, але на практиці деякі мови дають змогу робити з програмою тільки щось одне. Варто зауважити, що спосіб виконання програми не визначається мовою, на якій вона написана.

**Інтерпретатор** зчитує програму по одному рядку і одразу ж виконує машинні інструкції, що містяться у цьому рядку, а **компілятор** зчитує програму повністю і лише після цього перетворює її в об'єктний модуль, тобто у вигляд, який безпосередньо може виконати комп'ютер.



Об'єктний модуль ще називають **двійковим** (бінарним) або **машинним кодом**.

Повний набір символічних назв і правил їх використання утворюють *мову програмування*. Символьні імена називаються **мнемонічними**, а правила їх використання для створення команд і програм – **синтаксисом мови**.

*Програма, яка переводить код із мнемонічного вигляду у машинний, називається асемблером*. Команди асемблера вказують як інтерпретувати назви, де розмістити програму в пам'яті і яка кількість комірок пам'яті необхідна для зберігання даних.

За рівнем абстракції розрізняють мови програмування **низького** та **високого рівнів**.

Мови програмування *низького рівня (машинні мови)* орієнтовані на певний тип процесора і враховують його особливості. Вони забезпечують компактність, швидкодію і можливість прямого доступу до апаратних ресурсів. Їх переважно використовують для написання невеликих системних додатків, драйверів пристроїв, модулів сполучення з певними пристроями.

На відміну від мов програмування низького рівня, *мови програмування високого рівня* є набагато простішими у розумінні та зручнішими у написанні коду. Програма, написана такою мовою, взаємодіє з апаратними ресурсами через засоби операційної системи, тому вона потребує подальшого перекладу у машинний код. Для цього використовуються програми-транслятори.

Використання мов високого рівня порівняно з використанням асемблера має ряд таких переваг:

- програміст пише програми на мові, близькій до мови його спілкування; більше того, мови високого рівня орієнтуються на класи вирішуваних задач;
- скорочується час написання програм;
- мови високого рівня не залежать від типу та архітектури комп'ютера, що дає змогу використовувати написані цими мовами програми на всіх комп'ютерах, а програміста звільнити від знання їх структури та організації роботи.

### ***Розвиток мов програмування***

Основними парадигмами програмування є *декларативне та імперативне*.

*Декларативне програмування* описує, який результат необхідно отримати, замість опису послідовності дій для отримання цього результату.

*Імперативне програмування* описує процес отримання результатів як послідовність інструкцій зміни стану програми.

До декларативної парадигми належать функційне та логічне програмування.

**Функційне програмування** – парадигма програмування, яка розглядає програму як обчислення математичних функцій, та уникає станів і зміни даних.

Функція є базовим елементом функційного програмування. Функції використовуються для всього, навіть для простих обчислень. Змінні також замінюються функціями. Функційні програми можуть зберігати стан, але вони використовують для цього не змінні, а функції. За рахунок такого підходу функційне програмування дуже добре вирішує питання паралелізму і розподілених обчислень.

До основних понять функційного програмування належать функції першого класу, функції вищого порядку, чисті функції, замикання та незмінний стан.

У функційних мовах цикл переважно реалізовується у вигляді рекурсії. Оскільки при використанні рекурсії може знадобитися великий стек, тому у функційному програмуванні використовують хвостову рекурсію – випадок рекурсії, коли останній рекурсивний виклик функції повертає результат безпосередньо за адресою повернення, записаною для початкової функції.

Найвідомішими мовами функційного програмування є XQuery/XSLT, Haskell, LISP, Erlang, JavaScript, Python, Scala та Clojure.

**До імперативної парадигми належать структурне, процедурне, модульне (об'єктне), об'єктно-орієнтоване програмування**

Згідно із **структурною методологією** програмування будь-яка програма може бути створена за допомогою трьох конструкцій – послідовного виконання, розгалуження та циклу. У зв'язку із зростанням вимог до програмного забезпечення структурне програмування розвинулось у процедурне.

**Процедурне програмування** відображає програму у вигляді набору алгоритмів, для подачі яких можуть бути використані іменовані програмні блоки, тобто процедури та функції.

Основним механізмом процедурного програмування є *функція*. Процес розв'язання задачі в рамках процедурної парадигми називається функціональною абстракцією. Він дає змогу розробляти окремі функції, реалізуючи зв'язки між ними за допомогою механізму передачі параметрів і повертання результатів.

*Основна ідея модульності* полягає у забезпеченні доступу до даних і оперування ними незалежно від способу їхнього конкретного кодування у пам'яті комп'ютера. Дані разом із функціями їхнього опрацювання вбудовують (інкапсулюють) в окрему одиницю програми – **модуль**.

Модулі мають дві головні риси. По-перше, вони об'єднують структури даних з алгоритмами їхнього опрацювання. По-друге, у них відокремлено специфікацію від

реалізації інкапсульованих у модулі конструкцій, і це перетворює модуль на абстрактний тип даних

**Об'єктно-орієнтоване програмування** (ООП) – це технологія програмування, яка ґрунтується на понятті класів, об'єктів та успадкуванні елементів базових класів похідними класами. Клас визначає абстрактний тип даних, за допомогою якого описується певна сутність, а об'єкт є екземпляром класу.

Об'єктно-орієнтована методологія орієнтована на розроблення великих програмних систем групою програмістів і складається з об'єктно-орієнтованого аналізу, проектування та програмування. Принципи об'єктно-орієнтованого проектування – *інкапсуляція* (приховування від зовнішнього користувача деталей реалізації даних та методів їх опрацювання), *успадкування*, *поліморфізм* (використання однакових інтерфейсів для роботи з різними за функціональністю об'єктами)

### **Системи числення та бітові (розрядні) операції**

**Система числення** – це спосіб подання довільного числа за допомогою алфавіту символів, які називають цифрами.

Слід зауважити, що вся інформація, що зберігається та опрацьовується засобами обчислювальної техніки, незалежно від її типу (числа, текст, графіка, звук, відео), подана у двійковому коді.

Розрізняють *позиційні* та *непозиційні* системи числення.

Якщо в послідовності цифр, які зображають число, враховується позиція цифри, то систему числення називають **позиційною**. Такі системи числення характеризуються наочністю відображення чисел та простим виконанням арифметичних операцій

У позиційній системі числення значення цифри залежить від позиції, яку вона займає в числі. Розряди (позиції, в яких записуються цифри) нумеруються від крайнього правого (наймолодший розряд) до крайнього лівого (найстарший розряд). Нумерація починається від нуля.

Кожний розряд має значення множника  $\{\text{основа системи числення}\}^{\text{[номер розряду]}}$  для відповідної цифри

Наприклад, у числі 5839 остання цифра 9, що перебуває на нульовій позиції, відповідає кількості одиниць, 3 на першій позиції вказує на кількість десятків, 8 на другій позиції – це кількість сотень і цифра 5 на третій позиції – це кількість тисяч.

## 10-ва система числення

Це – найбільш відома система. Десять цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Розглянемо запис чисел в цій системі на прикладі числа 137

<b>Цифри</b>	<b>1</b>	<b>3</b>	<b>7</b>
<b>№№ розрядів</b>	<b>2</b>	<b>1</b>	<b>0</b>
<b>Множники</b>	$10^2$	$10^1$	$10^0$
<b>Значення множників</b>	100	10	1

$$1 \cdot 10^2 + 3 \cdot 10^1 + 7 \cdot 10^0 = 1 \cdot 100 + 3 \cdot 10 + 7 \cdot 1 = 137$$

## 2-ва система числення

У двійковій системі числення лише 2 цифри: 0 та 1.

Розглянемо запис чисел на прикладі двійкового числа 0101 0110. Його значення:

<b>Цифри</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>
<b>№№ розрядів</b>	7	6	5	4	3	2	1	0
<b>Множники</b>	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
<b>Значення множників</b>	128	64	32	16	8	4	2	1

$$\begin{aligned}
 &0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = \\
 &0 \cdot 128 + 1 \cdot 64 + 0 \cdot 32 + 1 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 = \\
 &64 + 16 + 4 + 2 = 86
 \end{aligned}$$

## 16-ва система числення

Цифри 16-кової системи числення:

16-ва цифра	значення (10-ве зображення)	2-ве зображення	16-ва цифра	значення (10-ве зображення)	2-ве зображення
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	A	10	1010
3	3	0011	B	11	1011
4	4	0100	C	12	1100
5	5	0101	D	13	1101
6	6	0110	E	14	1110
7	7	0111	F	15	1111

Для того, щоб вказати, що число подане у 16-вій системі числення, в мові C/C++ воно записується з префіксом 0x Розглянемо, наприклад, шістнадцяткове число 0x2C. Його значення:


<b>Цифри</b>	<b>2</b>	<b>С</b>
№№ розрядів	1	0
Множники	$16^1$	$16^0$
Значення множників	16	1

$$2 \cdot 16^1 + C \cdot 16^0 = 2 \cdot 16 + 12 \cdot 1 = 32 + 12 = 44$$

### Переведення чисел із 10-вої системи числення до 2-вої

Алгоритм переведення числа із однієї системи числення до іншої полягає в послідовному застосуванні операцій ділення націло та обчислення остачі від ділення націло.

№ кроку	Число x	Результат ділення числа x націло на 2 $x / 2$	Остача від ділення числа x націло на 2 $x \% 2$
0	137	68	1
1	68	34	0
2	34	17	0
3	17	8	1
4	8	4	0
5	4	2	0
6	2	1	0
7	1	0	1


1000 1001

Записуємо остаточний результат: номери кроків – це номери розрядів отриманого двійкового зображення числа  $x=137$ , а цифри в цих розрядах – результати операцій обчислення остачі від ділення на 2:

Розряд	7	6	5	4	3	2	1	0
Цифра	1	0	0	0	1	0	0	1

## Переведення чисел із 2-вої системи числення до 10-вої

Скористаємося правилом тлумачення двійкових чисел, поданим в пункті «2-ва система числення» параграфу «Позиційні системи числення». Розглянемо приклад такого переведення для числа 1001 0001


Цифра	1	0	0	1	0	0	0	1
Розряд	7	6	5	4	3	2	1	0
Множник	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
Значення множника	128	64	32	16	8	4	2	1

$$1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 128 + 16 + 1 = 145$$

## Переведення чисел із 10-вої системи числення до 16-вої

Розглянемо алгоритм переведення чисел із десяткової у шістнадцяткову систему числення на прикладі числа 317. Це – дуже подібно до переведення числа із десяткової у двійкову систему числення: як і раніше, потрібно послідовно застосувати операції ділення націло та обчислення остачі від ділення націло. Проте тепер ділимо та обчислюємо остачу від ділення не на 2, а на 16.

№ кроку	Число x	Результат ділення числа x націло на 16 $x / 16$	Остача від ділення числа x націло на 16 $x \% 16$
0	317	19	13 → D
1	19	1	3 → 3
2	1	0	1 → 1


0x13D

Записуємо остаточний результат: номери кроків – це номери розрядів отриманого шістнадцяткового зображення числа  $x=317$ , а цифри в цих розрядах – результати операцій обчислення остачі від ділення на 16

Розряд	2	1	0
Цифра	1	3	D

Остаточню отримали: 0x13D

## Переведення чисел із 16-вої системи числення до 10-вої

Розглянемо приклад переведення для числа 0x13D:



Цифра	<b>1</b>	<b>3</b>	<b>D</b>
Розряд	2	1	0
Множник	$16^2$	$16^1$	$16^0$
Значення множника	256	16	1

$$1 \cdot 16^2 + 3 \cdot 16^1 + D \cdot 16^0 = 1 \cdot 256 + 3 \cdot 16 + 13 \cdot 1 = 256 + 48 + 13 = 317$$

### Переведення чисел із 2-вої системи числення до 16-вої

Для того, щоб перевести число із двійкової системи числення до шістнадцяткової, запишемо це число, згрупувавши по 4 двійкових розряди (починаючи з наймолодшого).

Наприклад, число у двійковій системі 10010001. Після групування отримаємо 1001'0001. Тепер скористаємося таблицею шістнадцяткових цифр та їх двійкового та десяткового представлення та переведемо кожну групу із чотирьох двійкових цифр у одну шістнадцяткову:

$$1001 \rightarrow 9_{(16)}$$

$$0001 \rightarrow 1_{(16)}$$

$$1001'0001 \rightarrow 91_{(16)} \rightarrow 0x91.$$

### Переведення чисел із 16-вої системи числення до 2-вої

Розглянемо приклад переведення числа 0x89:  $0x89 \rightarrow 89_{(16)} \rightarrow 1000'1001$

### Алгоритм та способи його подання

**Алгоритм** – це система формальних правил, які чітко та однозначновизначають послідовність дій обчислювального процесу від початкових даних до шуканого результату. Алгоритм визначає певні правила перетворення інформації, тобто визначає певну послідовність операцій опрацювання даних для отримання розв'язку задачі.

*Алгоритм* – це система правил, яка сформульована мовою, зрозумілою виконавцеві алгоритму, визначає процес переходу від допустимих початкових даних до певного результату і володіє властивостями масовості, скінченності, визначеності, детермінованості.

Таким чином, *алгоритм* – це скінченна послідовність команд, які слід виконати над вхідними даними, щоб отримати результат.

Процес побудови алгоритмів називають **алгоритмізацією**.

Алгоритм може перебувати на вищому рівні абстракції, бути вільним від деталей

реалізації, пов'язаних з особливостями мови програмування та конкретної обчислювальної системи.

Засоби, прийняті для зображення алгоритмів, переважно називають **алгоритмічною мовою**.

Для подання алгоритма використовують такі *способи*:




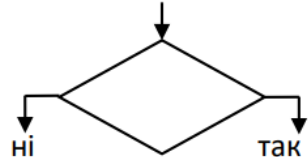

- *словесний опис* послідовності дій, який потребує подальшої формалізації;
- *аналітичний опис* у вигляді таблиць, формул, схем, малюнків тощо дає змогу описати алгоритм за допомогою системи умовних позначень і є найбільш формалізованим;
- *графічне подання* у вигляді схеми алгоритму (блок-схеми) використовується для наочності подання алгоритму за допомогою набору спеціальних блоків;
- *запис алгоритмічною мовою програмування*; алгоритм записується зрозумілою для комп'ютера мовою, тобто мовою програмування.

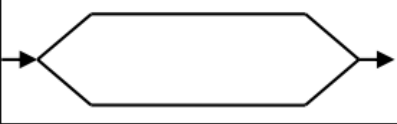
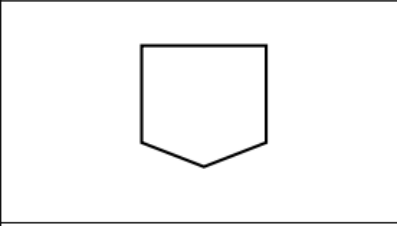
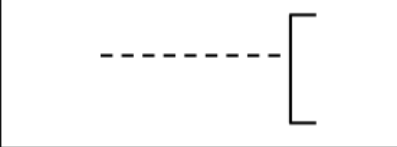
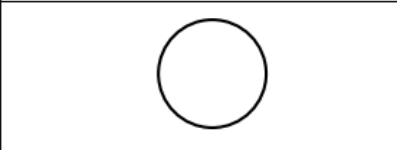
**Схема алгоритму (блок-схема)** – це графічне зображення його структури, в якому кожний етап процесу опрацювання даних подається у вигляді певних геометричних фігур (блоків).

Форма блока визначає спосіб дій, а записи всередині блока – деталі (змінні, параметри) відповідного етапу

Блоки з'єднуються між собою лініями потоків, які для кожного етапу визначають напрям дій.

### Функціональні блоки

	Блок початку/завершення алгоритму
	Блок введення/виведення даних
	Арифметичний блок, модифікування (виконання операції або групи операцій, в результаті яких змінюються значення, форма подання або розміщення даних)
	Умовний блок, розгалуження (вибір напрямку галуження алгоритму залежно від результату аналізу умови)
	Блок виклику підпрограми (функції, процедури)

	Блок циклу з параметром
	Міжсторінковий з'єднувач (вказівка зв'язку між роз'єднаними частинами схем алгоритмів та програм, розташованих на різних аркушах)
	Коментар (зв'язок між елементами схеми і поясненнями)
	З'єднувач (вказівка зв'язку між перерваними лініями потоку в межах однієї сторінки)

*Розробити алгоритм* – це розбити задачу на етапи (більш прості задачі), які послідовно виконуються

Якщо обсяг тексту, розміщеного всередині блока, перевищує його розміри, то використовується *коментар*.

У схемах слід уникати перетину ліній потоків. Лінії, що перетинаються, не мають логічного зв'язку між собою, тому зміна напрямку у точках перетину не допускається.

Розрізняють три базових алгоритмічних структури – *лінійна* (послідовність), *розгалужена* та *циклічна* (повторювання). Алгоритмічні структури будь-якої складності утворюються комбінацією послідовності базових структур.

– **Лінійним алгоритмом** називається такий алгоритм, в якому не порушується природний порядок обчислень, тобто, лінійні команди виконуються послідовно одна за одною (подання цієї конструкції у блок-схемах здійснюється послідовністю блоків модифікування);

– **Розгалуженим алгоритмом** називається алгоритм, який містить хоча б одну умову, в результаті аналізу якої здійснюється перехід до одного з можливих подальших кроків. Процес розгалуження організовується за допомогою логічного блоку;

– **Циклічний алгоритм** містить повторення певну кількість разів з новими початковими даними певної послідовності команд, яка утворює тіло циклу. У циклі повинна існувати змінна (параметр циклу), яка при кожному наступному виконанні тіла циклу змінює своє значення і визначає кількість повторень. У простому циклі змінюється один параметр від заданого початкового до кінцевого значення з постійним кроком (залежно від вимог задачі параметр циклу може збільшуватися або

зменшуватися). При кожному значенні параметра виконуються дії, які знаходяться всередині циклу, тобто тіло циклу. Вихід з циклу здійснюється після досягнення параметром циклу свого кінцевого значення

Запис алгоритму алгоритмічною мовою програмування має бути зрозумілим не лише людині, а й комп'ютеру і тому потрібно точно дотримуватися правил цієї мови