# Частина 1

# Організація програм

# Розділ 1

# Основні поняття та означення

- Основні принципи побудови обчислювальних систем
- Архітектура системи команд і форми зображення даних
- Системи числення
- Етапи створення програм — від аналізу поставленої задачі до налагодження
- Поняття алгоритму й елементарні алгоритмічні структури

## 1.1. Поняття архітектури комп'ютера

Поняття архітектури обчислювальних систем є одним з основних в інформатиці. Уперше термін «архітектура комп'ютера» був введений фірмою IBM при розробці обчислювальних систем серії IBM 360 і застосований до тих засобів, які може використовувати програміст під час написання програм на рівні машинних команд.

Під цим терміном розуміли структуру комп'ютера з погляду програміста, так звану логічну архітектуру, що є поняттям архітектури у вузькому розумінні. Воно охоплює формати команд, форми зображення даних, методи адресації й управління операціями введення-виведення. При цьому з розгляду випадають такі аспекти, як фізична організація пристроїв комп'ютера, ієрархія пам'яті, методи паралельної обробки інформації, а також багато-багато інших, які часто узагальнюються терміном «організація комп'ютера» чи структурна організація комп'ютера.

Далі ми використовуватимемо термін «архітектура», розуміючи під цим поняттям логічну структуру у сукупності з фізичною структурною організацією.

В основу архітектури більшості сучасних комп'ютерів покладено принципи, які ще 1946 року були сформульовані у звіті «Попереднє обговорення логічного конструювання електронного обчислювального пристрою» Джоном фон Нейманом і його колегами Г. Голдстайном та А. Берксом. Усі комп'ютери, побудовані згідно з цими принципами, тепер відомі як комп'ютери з фоннеймановською архітектурою.

Основні принципи архітектури фон Неймана такі:

- використання двійкової системи числення для кодування інформації у комп'ютері;
- програмне керування роботою комп'ютера;
- збереження програм у пам'яті комп'ютера;
- адресація пам'яті.

### 1.1.1. Принцип використання двійкової системи числення

Інформація (команди і дані) у комп'ютері кодуються у *двійковій системі числення*. *Система числення* — це система позначення чисел. У повсякденній практиці використовується десяткова система, в якій числа записуються за допомогою десяти арабських цифр 0, 1, 2, ..., 9. У двійковій системі числення є тільки дві цифри: 0 та 1. Зручність використання двійкової системи в обчислювальній техніці обумовлена тим, що електронні перемикачі можуть перебувати тільки в одному із двох станів: увімкненому чи вимкненому. Ці стани можна кодувати двома цифрами: одиницею або нулем. Так само в двох станах у кожен окремий момент часу може перебувати канал передачі даних: «рівень напруги високий» або «рівень напруги низький». Крім того, логіка висловлень є двійковою логікою: будь-яке висловлення в кожен момент є істинним або хибним. Якщо висловлення є істинним, йому відповідає значення 1, якщо хибне — значення 0.

Одна двійкова цифра називається *бітом* (від англ. *bi*nary digi*t* — двійкова цифра). За допомогою одного біта можна закодувати два інформаційних повідомлення, що умовно позначаються символами «0» та «1». Із двох бітів можна утворити коди чотирьох інформаційних повідомлень: «00», «01», «10» та «11»; із трьох бітів — восьми. У загальному випадку за допомогою $n$ бітів можна закодувати $2^n$ інформаційних повідомлень.

Послідовність, що складається з восьми бітів, у сучасній обчислювальній техніці прийнято називати *байтом*. За допомогою одного байта можна закодувати $2^8 = 256$ різних повідомлень і зобразити, наприклад, значення цілих чисел від 0 до 255. Ці самі повідомлення можна розглядати і як числа від –128 до 127 (їх кількість теж дорівнює 256), символи, логічні значення тощо.

Байт є одиницею обсягу пам'яті. Для позначення тисяч та мільйонів байтів використовуються такі одиниці, як кілобайт (1 Кбайт = $2^{10}$ = 1 024 байт), мегабайт (1 Мбайт = $2^{20}$ = 1 048 576 байт) і гігабайт (1 Гбайт = $2^{30}$ = 1 073 741 824 байт).

Послідовностями двійкових цифр можна кодувати не лише числа, а й довільні інформаційні повідомлення. Зокрема, загальноприйнята система кодування ASCII ставить коди у відповідність 256 символам. Наприклад, латинській літері «a» відповідає код $97_{10} = 1100001_2$, а символу «@» — код $64_{10} = 1000000_2$. Складніші системи кодування дають можливість закодувати зображення, звук тощо.

### 1.1.2. Принцип програмного керування роботою комп'ютера

Сучасний комп'ютер — це сукупність технічних і програмних засобів, які призначені для автоматизованої обробки дискретних даних відповідно до заданого алгоритму. Алгоритм — це основне поняття математики та обчислювальної техніки і згідно зі стандартом ISO 2382/1-84 він визначається як «скінченний набір інструкцій, які описують процес розв'язування задачі за допомогою скінченної кількості операцій».

Усі обчислення, що виконує комп'ютер, мають бути зображені у пам'яті у вигляді програми, складеної з інструкцій, які прийнято називати командами. Кожна

команда вказує на певну операцію, яку має виконати комп'ютер. Сукупність команд, що їх використовує конкретний комп'ютер, називається системою команд.

Програма складається з послідовності команд, які процесор виконує автоматично в певному порядку. Змінювати порядок виконання команд залежно від проміжних результатів обчислень дозволяють команди умовного та безумовного переходів. Завдяки цьому можна багаторазово виконувати одну й ту саму послідовність команд програми, а отже, набагато скоротити її розміри.

### 1.1.3. Принцип збереження програм у пам'яті комп'ютера

Цей один із найважливіших принципів, який називають також принципом однорідності пам'яті, полягає в тому, що команди та дані зберігаються в одних і тих самих областях пам'яті і нерідко кодуються тими самими станами комірок пам'яті. Комп'ютер обробляє і команди, і дані однаково, тобто над командою можна виконати ті самі операції, що й над будь-якими даними. Це розкриває нові можливості для перетворення програми безпосередньо при її виконанні. Наприклад, команди однієї частини програми можна отримати як результат виконання команд іншої. Ця можливість покладена в основу трансляторів, які перетворюють текст програми на мові високого рівня у послідовність команд конкретного комп'ютера.

Принцип однорідності пам'яті, запропонований у статті фон Неймана, використовувався в обчислювальних системах, які створювались у Прінстонському університеті, тому архітектура таких обчислювальних систем дістала назву *прінстонської*. Практично водночас у Гарвардському університеті запропонували іншу архітектуру, згідно з якою команди і дані повинні використовувати окрему пам'ять. Прінстонська архітектура була і є домінуючою, однак останнім часом завдяки широкому використанню нових технологій пам'яті все більше поширюються комп'ютери з гарвардською архітектурою.

### 1.1.4. Принцип адресності пам'яті

Для виконання програми необхідно, щоб команди та дані перебували в основній пам'яті. Основна пам'ять — це послідовність комірок, кожна з яких має свій номер — *адресу*. Розмір комірки зазвичай дорівнює восьми двійковим розрядам — байту. Числове значення у пам'яті, як правило, займає декілька сусідніх байтів. Для збереження цілих чисел найчастіше використовують один, два або чотири байти, для нецілих (дійсних) — 4, 6, 8 або 10 байт.

Адресою числа вважається адреса першого байта області пам'яті, відведеної для збереження числа. Так, якщо 32-розрядне двійкове число зберігається в комірках 200, 201, 202 та 203, то його адресою буде 200. Такий метод адресації називається адресацією молодшого байта і використовується в комп'ютерах фірми Intel і міні- комп'ютерах компанії DEC. Існує й інший метод, коли адресою числа є його старший байт (метод адресації старшого байта). Такий метод використовується в комп'ютерах фірм IBM та Motorola.

Процесор у будь-який момент може отримати доступ до будь-якої комірки пам'яті. Така пам'ять називається пам'яттю з довільним доступом.

## 1.2. Архітектура комп'ютерів фон Неймана

У класичній роботі фон Неймана перелічено основні пристрої, з використанням яких може бути побудований комп'ютер. Вважають, що комп'ютери такого типу мають *фоннейманівську архітектуру*. Типова фоннейманівська обчислювальна машина має такі складові: арифметико-логічний пристрій, пристрій керування, пам'ять і пристрої введення-виведення (рис. 1.1).

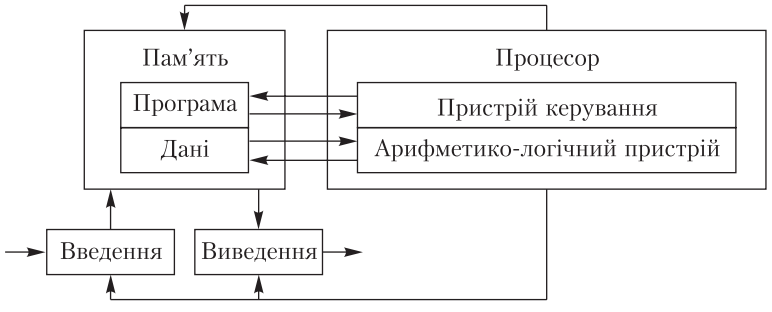


**Рис. 1.1.** Структура фоннейманівської обчислювальної машини

В обчислювальних машинах із класичною фоннейманівською архітектурою функції обробки інформації покладені на *арифметико-логічний пристрій* (АЛП). До функцій АЛП входить у першу чергу виконання арифметичних і логічних команд та команд зсуву. Отже, цей пристрій забезпечує обробку вхідних даних і формування результату. АЛП — це не один, а ціла група операційних пристроїв, кожен з яких реалізує певну підмножину операцій.

*Пристрій керування* (ПК) — найголовніша частина комп'ютера, що координує роботу всіх його пристроїв та забезпечує виконання всіх програм. Основною функцією цього пристрою є формування сигналів, необхідних для вибирання команд із пам'яті в порядку, що задається програмою, і подальше виконання цих команд. Крім того, ПК формує сигнали, необхідні для синхронізації та координації дій зовнішніх і внутрішніх пристроїв комп'ютера. Синхронізуючі сигнали — це сигнали, що визначають, коли має бути виконана певна операція. Реальні синхронізуючі сигнали, які керують пересиланням даних і виконанням команд, генеруються схемами керування.

Пристрій керування можна уявити собі як окремий блок, котрий взаємодіє з іншими блоками комп'ютера. Однак на практиці так буває рідко. Більша частина керуючих схем фізично розподілена між різними компонентами комп'ютера: процесором, чипсетом, контролерами введення-виведення та шин.

АЛП і ПК дуже тісно взаємодіють між собою, і їх часто реалізують єдиним пристроєм, відомим як центральний процесор, чи просто процесор. Процесор — це основа будь-якого комп'ютера, і швидкість роботи комп'ютера цілковито залежить від моделі та параметрів процесора.

Крім результуючих даних, АЛП формує також ознаку, яка визначає коректність отриманого результату, а також інші його характеристики (рівність нулю,

переповнення, парність і т. ін.). Значення ознаки результату може аналізуватися ПК для прийняття рішення стосовно подальшої послідовності виконання програми.

*Пам'ять комп'ютера* призначена для збереження інформації й оперативного обміну нею з іншими компонентами комп'ютера. Пам'ять поділяють на внутрішню та зовнішню. *Внутрішня пам'ять* складається з регістрів процесора, основної пам'яті та кеш-пам'яті.

*Регістри процесора* — це найбільш швидкодіючий, але найменший за обсягом різновид пам'яті комп'ютера. Зазвичай регістрів у процесорі небагато, і тільки в комп'ютерах з неповним набором команд (Reduced Instruction Set Computer, RISC) їх кількість може сягати кількох сотень. Регістри залежно від свого призначення можуть мати обсяг від 1 до 10 байт.

*Основна пам'ять* може включати пам'ять двох типів — *постійну* й *оперативну*.

*Постійна пам'ять* (Read Only Memory, ROM) реалізується у вигляді постійного запам'ятовуючого пристрою, дані в якому не можна модифікувати, їх можна тільки читати. Основне призначення ROM — підтримання процедур початкового завантаження операційної системи та обслуговування переривань, які припиняють виконання програми з метою реалізації спеціальних системних дій.

Процес зчитування інформації з постійної пам'яті майже не відрізняється від процесу зчитування з оперативної пам'яті. З іншого боку процес записування інформації в ROM набагато складніший і потребує великих затрат часу й енергії. Записування інформації в постійну пам'ять називають її програмуванням. Сучасні пристрої постійної пам'яті — це мікросхеми, інформація до яких може записуватись як при виготовленні, так і після нього.

*Оперативна пам'ять* (Ramdom Access Memory, RAM) використовується і для читання, і для запису інформації. Під час роботи комп'ютера в ній зберігаються програми та дані. Оперативна пам'ять комп'ютера організована у вигляді множини байтів, або комірок, у яких зберігаються числові та символьні значення. Байти оперативної пам'яті послідовно пронумеровані починаючи з 0. Ці номери байтів є їх *адресами*. Залежно від типів даних, над якими виконуються дії, байти групуються у слова (два байти) або подвійні слова (чотири байти).

Число, що зберігається в комірці, — це її *значення*, або *вміст*. Якщо в $k$-й комірці міститься, наприклад, число $m$, то прийнято говорити «вміст комірки з адресою $k$ дорівнює $m$». Під час читання даних з оперативної пам'яті вміст комірки не змінюється. Для його зміни потрібно записати в цю комірку нове значення, або, інакше кажучи, *присвоїти* комірці нове значення. Поняттю адреси оперативної пам'яті повною мірою відповідає поняття змінної в алгебрі. Змінну позначають ідентифікатором. Цьому ідентифікатору ставиться у відповідність адреса комірки оперативної пам'яті. Якщо змінній надати певне значення, то воно записується за адресою відповідної комірки пам'яті.

Особливістю RAM є неможливість зберігання інформації в ній після вимикання живлення комп'ютера. Саме цим вона відрізняється від пам'яті ROM, в якій дані зберігаються незалежно від наявності живлення.

Між регістрами та процесором часто розміщується *кеш-пам'ять*, призначена для узгодження швидкостей роботи основної пам'яті та процесора. Сучасні ком-

п'ютери мають декілька рівнів кеш-пам'яті, які позначаються буквою L з певним номером. У більшості персональних комп'ютерів кеш-пам'ять має два рівні — L1 та L2, в останніх розробках все частіше з'являється кеш-пам'ять третього рівня, а в проектах навіть і четвертого. Обсяг кеш-пам'яті вимірюється десятками і сотнями кілобайтів, а оперативної — десятками і сотнями мегабайтів. Оперативна пам'ять сучасних персональних комп'ютерів сягає кількох гігабайтів.

Для довгострокового збереження великого обсягу даних і програм потрібна *зовнішня пам'ять*. Зокрема, у зовнішній пам'яті зберігається все програмне забезпечення комп'ютера (системне та прикладне). До складу зовнішньої пам'яті входять різноманітні пристрої: накопичувачі на жорстких і гнучких магнітних дисках, пристрої на касетній магнітній стрічці (стримери), накопичувачі на оптичних дисках CD-ROM та CD-RW (від англ. Compact Disk Read Only Memory — компакт- диск тільки для читання, Compact Disk Read Write Memory — компакт-диск для запису та читання відповідно) тощо. Накопичувачі на жорстких магнітних дисках відрізняються від накопичувачів на гнучких магнітних дисках за конструкцією, обсягом даних, що зберігаються, а також часом пошуку, запису і зчитування інформації. Необхідно пам'ятати, що інформація, яка зберігається у зовнішній пам'яті, стане доступною для процесора тільки після того, як буде переписана в основну пам'ять.

Накопичувач можна розглядати як пристрій, у якому носій інформації поєднаний з приводом, що є механізмом зчитування-запису інформації, та відповідним керуючим пристроєм. Дисковий привід називається дисководом, а стрічковий — стримером. Комп'ютери зазвичай мають декілька дисководів для роботи з дисками різних типів.

*Пристрої введення-виведення* є важливою складовою будь-якого комп'ютера. Вони забезпечують взаємодію комп'ютера з навколишнім середовищем, користувачами, об'єктами керування та іншими комп'ютерами.

*Зовнішні пристрої* за їх призначенням можна розділити на два основних різновиди — для тривалого зберігання інформації та для взаємодії комп'ютера із зовнішнім середовищем, а саме користувачами, іншими комп'ютерами й об'єктами керування.

Серед них можна виділити пристрої введення (клавіатура, сканер, графічний планшет, маніпулятори типу «миша», джойстик, світлове перо тощо), виведення (принтер, плотер), діалогові засоби користувача (дисплеї, пристрої мовного введення-виведення — мікрофонні акустичні системи, синтезатори звуку тощо), засоби зв'язку та телекомунікацій (мережні плати, модеми).

Більшість зовнішніх пристроїв мають свої процесори (контролери), які є простішими за центральний процесор і виконують інші набори команд. Контролери можуть переносити дані із зовнішніх носіїв до оперативної пам'яті (читання, або введення із «зовнішнього світу») чи навпаки (запис, або виведення даних у «зовнішній світ»). Кожному пристрою введення-виведення виділено окрему ділянку оперативної пам'яті — порт. Із нього пристрій бере дані для зовнішнього носія, записуючи їх, наприклад, на диск або на екран комп'ютера. І саме в порт записуються дані, що надходять з клавіатури або дисковода.

Сигнали синхронізації дій усіх пристроїв передаються по керуючих лініях — шинах. Шини комп'ютера повинні забезпечити передачу інформації між:

- процесором та оперативною пам'яттю (шина процесор–пам'ять);
- процесором і портами введення-виведення зовнішніх пристроїв;
- оперативною пам'яттю і портами введення-виведення зовнішніх пристроїв у режимі прямого доступу до пам'яті.

Шина процесор–пам'ять забезпечує безпосередній зв'язок між процесором комп'ютера та основною пам'яттю. У сучасних комп'ютерах таку шину іноді називають шиною переднього плану і позначають як FSB (від англ. Front-Side Bus). Інтенсивний обмін даними між процесором та пам'яттю вимагає, щоб кількість інформації, яка передається за одиницю часу (секунду) цією шиною (пропускна спроможність шини), була якомога більшою. Від пропускної спроможності шини процесор–пам'ять значною мірою залежить продуктивність комп'ютерів із фоннеймановською архітектурою. Функції різних шин конструктори іноді покладають на єдину системну шину, але з погляду швидкодії краще, коли дані між процесором і пам'яттю передаються окремою шиною.

Зв'язок процесора чи пам'яті із пристроями введення-виведення забезпечують *шини введення-виведення*. На відміну від шини процесор–пам'ять, такі шини містять менше ліній, але фізична довжина цих ліній може бути більшою. Велика кількість і різноманітність зовнішніх пристроїв у різних типах комп'ютерів обумовили необхідність розроблення стандартів для таких шин. Стандартизація шин дозволяє розробникам зовнішніх пристроїв працювати незалежно, а користувачам — самостійно формувати потрібну конфігурацію комп'ютера.

Фізично шини складаються з великої кількості паралельних металевих провідників — ліній, розміщених на системній платі чи кристалі мікросхеми. Серед ліній будь-якої шини можна виділити три функціональні групи: *шина адреси*, *шина даних* і *шина керування*.

Шиною адреси передаються адреси комірок пам'яті, номери регістрів процесора, адреси портів введення-виведенні і т. ін. Кількість ліній, виділених для передачі адреси, становить ширину шини адреси і визначає максимально можливий обсяг пам'яті комп'ютера, який може адресуватися.

Лінії, якими дані (команди чи операнди) передаються між блоками системи, називаються шиною даних. Найважливіші параметри шини даних — ширина та пропускна спроможність. Ширина шини даних вказує на кількість бітів інформації, які можуть бути передані по шині за один її цикл.

Використання окремих шин для адрес і даних дає можливість значно підвищити продуктивність комп'ютера, особливо під час записування інформації в пам'ять — адреса комірки пам'яті та дані, що записуються, передаються паралельно.

Поряд із лініями для передачі адрес та даних обов'язковим атрибутом будь-якої шини є лінії, якими передається керуюча інформація та інформація про стан пристроїв введення-виведення. Сукупність таких ліній прийнято називати шиною керування. Цією шиною передаються сигнали синхронізації, переривання, арбітра шини тощо.

Загалом, функціонування фоннеймановського комп'ютера можна описати так:

- комп'ютер за допомогою пристроїв введення приймає інформацію у вигляді програм та даних і записує її в пам'ять;
- інформація, що зберігається в пам'яті, під керуванням програми шинами пересилається в арифметико-логічний пристрій для подальшої обробки;
- дані, отримані після обробки інформації, спрямовуються (також шинами) на пристрої виведення.

## 1.3. Архітектура системи команд

Щоб виконати операції над даними будь-якого типу, в оперативній пам'яті потрібно розмістити відповідні команди програми. Коди команд, так само як і їх структура, розробляються під час проектування комп'ютера. Повний перелік команд, які здатний виконати даний комп'ютер, називається його *системою команд*. Система команд сучасного комп'ютера може містити сотні і навіть тисячі інструкцій. Типова команда комп'ютера фоннеймановської архітектури у загальному випадку має задавати:

- операцію, яку слід виконати;
- адреси даних (операндів), над якими має бути виконана операція;
- адресу пам'яті, де потрібно зберегти результат виконання операції.

Отже, команда складається з двох частин — командної й адресної. У даному разі термін «адресність» означає кількість операндів, вказаних в адресній частині команди. Триадресна команда може мати таку структуру, як показано на рис. 1.2. КОП — код операції, А1, А2, А3 — адреси комірок оперативної пам'яті, в яких містяться дані та результат виконання операції.


**Рис. 1.2.** Спрощена структура триадресної команди

Як правило, команда розміщується в декількох байтах. Адреса команди визначається як адреса її першого байта. Від формату команди залежить її структура, тобто кількість двійкових розрядів, які відводяться для збереження команди в пам'яті, а також кількість та розташування окремих груп розрядів команди. Код операції та адреси комірок даних — це послідовності двійкових розрядів.

Команди оперують даними, які прийнято називати операндами. Базовими типами операндів є адреси, числа, символи та логічні значення. Крім базових типів комп'ютер здатний обробляти і складніші інформаційні структури — графічні зображення, аудіо-, відеоінформацію, анімацію. Така інформація є похідною від базових типів і зберігається, як правило, на зовнішніх носіях.

Наведена триадресна команда (рис. 1.2) відповідає зображенню звичайних алгебричних операцій вигляду $x = y \times z$, що читається так: «виконати операцію

множення над змінними *y* і *z*, результат присвоїти змінній *x*» або «виконати операцію множення над вмістом комірок *y* і *z*, результат записати до комірки з адресою *x*».

Команди програми виконуються послідовно, починаючи з першої. Це можна реалізувати завдяки тому, що адреси комірок оперативної пам'яті, в яких розміщено команди програми, становлять послідовність. Процесор має *регістр команд*, призначений для їх автоматичного виконання, та *регістр-лічильник команд*, у якому зберігається адреса наступної виконуваної команди. Для виконання наступної команди вміст лічильника команд необхідно збільшити на довжину поточної команди (визначається в байтах), з тим щоб отримати адресу наступної команди. З метою автоматичного виконання команди потрібно вибрати її за вказаною адресою і передати до регістра команд. Щоб зв'язати адреси комірок оперативної пам'яті з ідентифікаторами змінних, які використовуються у процесі розв'язання конкретної задачі, створюється таблиця адрес. Процес її упорядкування називається *розподілом пам'яті*.

**Приклад 1.1**

Нехай потрібно обчислити значення змінної *a*, заданої виразом $a = (b + c) \times d$, за умови, що пам'ять розподіляється, починаючи від комірки з адресою 100. Тоді пам'ять, призначену для зберігання змінних, можна розподілити, скажімо, так: змінній *a* поставити у відповідність комірку з адресою 100, змінній *b* — комірку з адресою 101 тощо (табл. 1.1); змінна *r* призначена для збереження проміжних результатів.

**Таблиця 1.1.** Приклад розподілу пам'яті під змінні

| Ідентифікатор змінної | Адреса комірки оперативної пам'яті |
|---|---|
| *a* | 100 |
| *b* | 101 |
| *c* | 102 |
| *d* | 103 |
| *r* | 106 |

Запишемо машинну програму обчислення значення змінної *a*, заданої виразом $a = (b + c) \times d$. У цій програмі операція введення задається кодом 01, операція виведення — кодом 02, операція додавання — кодом 03, операція множення — кодом 05 (табл. 1.2).

**Таблиця 1.2.** Машинна програма обчислення виразу $a = (b + c) \times d$

| КОП | А1 | А2 | А3 | Примітка |
|---|---|---|---|---|
| 01 | 101 | 000 | 000 | Увести значення змінної *b* до комірки з адресою 101 |
| 01 | 102 | 000 | 000 | Увести значення змінної *c* до комірки з адресою 102 |
| 01 | 103 | 000 | 000 | Увести значення змінної *d* до комірки з адресою 103 |
| 03 | 101 | 102 | 106 | Додати значення змінних *b* і *c*, результат помістити в комірку з адресою 106, яка відповідає змінній *r* |

| КОП | А1 | А2 | А3 | Примітка |
|---|---|---|---|---|
| 05 | 106 | 103 | 100 | Помножити значення змінних *r* і *d,* помістити результат у комірку з адресою 100 (у змінну *a*) |
| 02 | 100 | 000 | 000 | Вивести значення змінної *a* з комірки, що має адресу 100 |
| 00 | 000 | 000 | 000 | Кінець обчислень (адреси операндів та результату не потрібні) |

Нехай програма розміщена в пам'яті так, що адресою її першої команди є комірка 500. Ця адреса запам'ятовується в лічильнику команд (так звана *пускова адреса*), і включається режим автоматичного виконання програми, що передбачає таку послідовність дій.

1. Перша команда програми, адреса якої визначається лічильником команд, передається в регістр команд процесора.
2. Арифметико-логічний пристрій процесора дешифрує код операції.
3. Числа, що зберігаються в комірках, які визначаються адресами команди, передаються в АЛП як операнди.
4. Виконується операція, і результат заноситься в комірку пам'яті з адресою, зазначеною у команді.
5. Вміст лічильника команд модифікується шляхом додавання до нього довжини поточної команди, і для наступної команди повторюються дії, що зазначені в пунктах 1–5.
6. Обчислення завершується, коли поточна команда містить код операції, яка припиняє обчислення.

Інформацію, призначену для обробки комп'ютером, необхідно закодувати у відповідному форматі. У результаті кодування будь-яке число, символ чи команда перетворюються у рядок двійкових цифр. Для зображення кожного типу даних передбачено певний формат.

## 1.4. Інформація в пам'яті комп'ютера

Символьна, графічна, числова та будь-яка інша інформація зображується в пам'яті комп'ютера у вигляді послідовностей одиничних та нульових бітів. Такі послідовності можна розглядати як двійкові, вісімкові або шістнадцяткові числа і виконувати над ними арифметичні операції у відповідних системах числення.

### 1.4.1. Позиційні системи числення

Використання двійкової системи числення набагато спрощує апаратну реалізацію пристроїв комп'ютера. Але двійковий запис числа приблизно втричі довший за його десятковий еквівалент, і тому людині працювати з ним незручно. У програмуванні для скороченого запису двійкових чисел використовують *шістнадцяткову систему числення.* Перші десять цифр шістнадцяткової системи збігаються

з десятьма арабськими цифрами. До них додаються шість латинських літер A, B, C, D, E, F, котрі позначають десяткові числа 10, 11, 12, 13, 14, 15.

Усі згадані системи числення є *позиційними*, оскільки вага кожної цифри залежить від її позиції в записі числа. Наприклад, цифра 3 у числі 31 позначає десятки, тобто має вагу $10^1$, а в числі 13 позначає одиниці, тобто її вага становить $10^0$. Кількість різних символів, що використовуються для запису чисел, називається *основою системи числення*. Позиційна система числення з основою $N$ має $N$ цифр $C_0$, $C_1$, ..., $C_{N-1}$, що позначають натуральні числа від 0 до $N - 1$.

Число $N$ у $N$-ковій системі позначається дворозрядним записом $C_1C_0$. Так, числа 10 у десятковій, 2 у двійковій та 16 у шістнадцятковій системах записуються однаково: 10. Число $N^2$ позначається вже трьома цифрами: $C_1C_0C_0$ тощо. Аналогічно, $m$-розрядні дроби ($m > 0$) у $N$-ковій системі мають вигляд $0,x_{-1}x_{-2}...x_{-m}$, де нулем позначена ціла частина числа, вага цифри $x_{-i}$ дорівнює $N^{-i}$, $i = 1, 2, ..., m$. Так, у десятковому числі 0,1234 цифра 3 має вагу $10^{-3}$.

Остання цифра справа у записі цілого числа (або остання цифра перед комою у дробовому числі) називається *наймолодшою*, перша цифра ліворуч називається *найстаршою*. Основу системи числення вказують праворуч від числа у нижньому індексі, наприклад: $123_{10}$, $1001_2$; якщо це позначення основи опущене, число вважається десятковим. Значення $Y$ числа, що записане у $N$-ковій системі числення, дорівнює значенню такого полінома:

$$Y = \sum_{i=-m}^{n} x_i N^i.$$

Тут $x_i$ — значення цифри в $i$-му розряді числа; $N^i$ — вага $i$-го розряду числа; $i = -1, -2, ..., -m$ — розряди дробової частини числа; $i = 0, 1, ..., n$ — розряди цілої частини числа.

Зауважимо, що скінченний $N$-ковий дріб дорівнюватиме скінченному десятковому дробу лише тоді, коли число $N$ не матиме інших дільників, окрім $5_{10}$ та $2_{10}$. Інакше скінченний $N$-ковий дріб буде зображений у десятковій системі у вигляді нескінченного періодичного дробу.

**Приклад 1.2**

$512_{10} = 5\times10^2 + 1\times10^1 + 2\times10^0$;

$(512,346)_{10} = 5\times10^2 + 1\times10^1 + 2\times10^0 + 3\times10^{-1} + 4\times10^{-2} + 6\times10^{-3}$;

$(10\ 011)_2 = 1\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = 19_{10}$;

$(10,011)_2 = 1\times2^1 + 0\times2^0 + 0\times2^{-1} + 1\times2^{-2} + 1\times2^{-3} = 2,375_{10}$;

$(1BC)_{16} = 1\times16^2 + 11\times16 + 12 = 444_{10}$;

$(1B,C)_{16} = 1\times16^1 + 11\times16^0 + 12\times16^{-1} = 27,75_{10}$;

$(12,2)_3 = 1\times3^1 + 2\times3^0 + 2\times3^{-1} = 5,(6)_{10}$ (число зображується нескінченним періодичним десятковим дробом).

Оскільки в комп'ютері числа обробляються у двійковій системі числення, а для людини звичним є десятковий запис чисел, то постає потреба у переведенні числа з однієї системи числення до іншої.

### Переведення натуральних чисел з однієї системи числення до іншої

Розглянемо задачу переведення числа $P$ з $M$-кової системи числення до $N$-кової.

Нехай число $P$ у $N$-ковій системі числення має запис $P = (x_k x_{k-1}...x_1 x_0)_N$, який містить цифри $x_i$ у невідомій кількості $k + 1$. Записати це число можна так:

$$P = x_k \times N^k + x_{k-1} \times N^{k-1} + ... + x_1 \times N + x_0 = ((x_k \times N + x_{k-1}) \times N + ... + x_1) \times N + x_0.$$

Звідси видно, що значенням наймолодшої цифри $x_0$ є остача від ділення числа $P$ на основу $N$ (усі операції здійснюються над $M$-ковими числами). Значенням другої справа цифри $x_1$ буде остача від ділення частки, яку отримано на попередній ітерації, на основу $N$. Продовжуючи ці міркування, отримаємо циклічну процедуру, кожна ітерація якої полягатиме у знаходженні частки та остачі від ділення деякого числа $Q$ на основу системи числення. При цьому отримана на $i$-й ітерації частка стає самим числом $Q$ на $(i+1)$-й ітерації. Якщо частка менша від $N$, обчислення завершують. Остання частка є старшою цифрою числа, інші цифри записуються в порядку, зворотному до порядку отримання остач, тобто перша остача дає наймолодшу цифру.

**Приклад 1.3**

Нехай потрібно перевести число $25_{10}$ з десяткової системи числення до двійкової. Для розв'язання цієї задачі застосуємо щойно описану схему обчислень. Остачу від ділення будемо записувати в дужках після частки.

25 : 2 = 12(1);
12 : 2 = 6(0);
6 : 2 = 3(0);
3 : 2 = 1(1).

Остання частка менша двох. Вона є старшою цифрою двійкового числа, до якого треба дописати остачі у порядку, зворотному до порядку їх отримання.

Результатом є число $11001_2$.

З математичної точки зору переведення чисел з будь-якої $M$-кової позиційної системи числення до будь-якої $N$-кової здійснюється за тією процедурою, що її було розглянуто вище. Проте в цій процедурі операції знаходження частки та остачі від ділення здійснюються над числами у $M$-ковій системі, а для людини звичними є лише операції над десятковими числами. Тому, як зазначалося вище, десяткове значення $M$-кового числа $P$ зручніше обчислювати, користуючись його зображенням у вигляді поліному $P = x_k \times M^k + x_{k-1} \times M^{k-1} + \ldots + x_1 \times M + x_0$ .

Наприклад:

$5AF_{16} = 5_{16} \times 16^2 + A_{16} \times 16^1 + F_{16} \times 16^0 = 5 \times 16^2 + 10 \times 16^1 + 15 \times 16^0 = 1455$;

$11001_2 = 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 25$.

### Переведення дробових чисел з однієї системи числення до іншої

Нехай задане $M$-кове дробове число $V$, $(0 \le V \le 1)$. Припустимо, що у $N$-ковій системі числення число $V$ має запис $V = (0,x_{-1}x_{-2}...)_N$, тобто

$$V = N^{-1} \times (x_{-1} + N^{-1} \times (x_{-2} + \ldots)).$$

Тут цифри $x_{-i}$ невідомі. Помножимо обидві частини рівності на $N$:

$$V \times N = x_{-1} + N^{-1} \times (x_{-2} + \ldots).$$

Значення цифри $x_{-1}$ отримаємо, взявши цілу частину добутку $V \times N$, тобто $x_{-1} = \lfloor V \times N \rfloor$, а $N^{-1} \times (x_{-2} + \ldots) = \{V \times N\}$, де $\lfloor V \times N \rfloor$ та $\{V \times N\}$ позначають цілу та дробову частини $V \times N$. Помножимо дробову частину $\{V \times N\}$ на $N$ і знову візьмемо цілу частину. В результаті отримаємо значення цифри $x_{-2}$ і т. д. Якщо на деякій ітерації значення $\{V \times N\}$ стане рівним 0, процес можна припинити. Але, можливо, значення $\{V \times N\}$ ніколи нулю не дорівнюватиме. В такому разі запис точного значення $V$ у $N$-ковій системі буде нескінченним, а кожна додатково отримана цифра уточнюватиме наближене значення $V$.

Отже, для переведення дробу з $M$-кової системи числення до $N$-кової потрібно послідовно множити дробову частину на основу системи числення $N$. Цілі частини добутків, отриманих у результаті виконання послідовності операцій множення, є цифрами дробу в $N$-ковій системі числення.

У разі, коли $V$ задане у вигляді скінченного десяткового дробу і число $N$ має серед своїх простих дільників як 2, так і 5, зображення числа $V$ в $N$-ковій системі числення буде скінченним. Інакше в $N$-ковій системі числення десятковий дріб $V$ може зображуватися нескінченним періодичним дробом.

**Приклад 1.4**

Нехай потрібно перевести десятковий дріб 0,75 до двійкової системи. Послідовність операцій множення така: $0{,}75 \times 2 = 1{,}5$; ціла частина $\lfloor 1{,}5 \rfloor = 1$, дробова частина $\{1{,}5\} = 0{,}5$; $0{,}5 \times 2 = 1$, дробова частина $\{1\} = 0$. Усі подальші цифри будуть нулями, тому $0{,}11_2$ є скінченним двійковим зображенням дробу $0{,}75_{10}$.

У процесі переведення десяткового дробу $0{,}26_{10}$ у двійкову систему виконується така послідовність операцій множення: $\lfloor 0{,}26 \times 2 \rfloor = \lfloor 0{,}52 \rfloor = 0$; $\lfloor 0{,}52 \times 2 \rfloor = \lfloor 1{,}04 \rfloor = 1$; $\lfloor 0{,}04 \times 2 \rfloor = \lfloor 0{,}08 \rfloor = 0$; $\lfloor 0{,}08 \times 2 \rfloor = \lfloor 0{,}16 \rfloor = 0$; $\lfloor 0{,}16 \times 2 \rfloor = \lfloor 0{,}32 \rfloor = 0$; $\lfloor 0{,}32 \times 2 \rfloor = \lfloor 0{,}64 \rfloor = 0$, $\lfloor 0{,}64 \times 2 \rfloor = \lfloor 1{,}28 \rfloor = 1$. Точне двійкове зображення десяткового дробу 0,26 є нескінченним періодичним, а $0{,}010\ 000\ 1_2$ — його наближення.

Переведення десяткового дробу 0,8 у шістнадцяткову систему числення здійснюється так: $0{,}8 \times 16 = 12{,}8$ з цілою частиною $\lfloor 12{,}8 \rfloor = 12$ і дробовою частиною $\{12{,}8\} = 0{,}8$. Десяткове число 12 замінюємо на шістнадцяткове значення $C_{16}$. Усі подальші шістнадцяткові цифри будуть також дорівнювати $C$. Отже, маємо періодичний дріб $0{,}(C)_{16}$.

### Зв'язок двійкової та шістнадцяткової систем

Розглянемо простий спосіб переведення шістнадцяткового числа у двійкове та навпаки. Оскільки $16 = 2^4 = 10000_2$, то одна шістнадцяткова цифра використовується для зображення чотирьох бітів:

0 — 0000, 1 — 0001, 2 — 0010, 3 — 0011, 4 — 0100, 5 — 0101, 6 — 0110, 7 — 0111,
8 — 1000, 9 — 1001, A — 1010, B — 1011, C — 1100, D — 1101, E — 1110, F — 1111.

Отже, для переведення двійкового числа у шістнадцяткове достатньо у двійковому записі, починаючи з наймолодшої цифри, замінити кожні чотири біти відповідними шістнадцятковими цифрами (якщо найстарша четвірка розрядів неповна, до неї дописуються незначущі нулі). Навпаки, шістнадцятковий запис переводиться в двійковий заміною цифр відповідними четвірками бітів. Четвірка бітів, що відповідає найстаршій шістнадцятковій цифрі, може мати незначущі нулі у старших розрядах.

При переведенні дробових чисел четвірки розрядів слід виділяти, починаючи від найближчого до коми розряду як у цілій, так і у дробовій частині числа. При цьому до неповних четвірок дописуються нулі (йдеться про наймолодшу четвірку розрядів у дробовій частині та найстаршу четвірку у цілій частині числа).

**Приклад 1.5**

Перевести в шістнадцяткову систему числення двійкове число 1111010. Спочатку число, починаючи від молодшої цифри, розбивають на групи: 1010 та 111. Старша група доповнюється до тетради 0111. Цій тетраді відповідає шістнадцяткова цифра 7, а тетраді 1010 — цифра A. Отже, результатом переведення є 7A.

Тепер зобразимо у двійковому вигляді шістнадцяткове число A7. Шістнадцяткова цифра A зображує двійкове число 1010, а шістнадцяткова цифра 7 — двійкове число 111. Доповнивши двійковий запис числа 7 нулем у старшому розряді, отримаємо $A7_{16} = 10100111_2$.

### Арифметичні операції в різних системах числення

Основними арифметичними операціями у системі команд будь-якого комп'ютера є додавання, віднімання, множення та ділення. Правила виконання цих операцій у десятковій системі числення загальновідомі. Вони застосовуються і при виконанні аналогічних дій в інших позиційних системах числення.

Розглянемо операцію додавання двійкових чисел. У десятковій системі 9 + 1 = = 10, а у двійковій 1 + 1 = 10. Таким чином, при додаванні у стовпчик двох одиничних розрядів отримаємо нульовий розряд суми і одиницю перенесення у наступний, старший розряд. Додамо у стовпчик числа $11_2$ і $110_2$, вказуючи розряди перенесень у дужках.

$$\begin{array}{r} (110) \\ 11 \\ + \quad 110 \\ \hline 1001 \end{array}$$

Додавання у системі числення з довільною основою $P$ виконується аналогічно тому, як це робиться у десятковій та двійковій системах. При додаванні двох однорозрядних $P$-кових чисел $a_1$ і $a_2$ обчислюється сума $S = a_1 + a_2$, і результатом є $(S \text{ div } P)S \bmod P$, де в дужках вказано одиницю перенесення у наступний розряд, $S \text{ div } P$ — ціла частина, а $S \bmod P$ — остача від ділення $S$ на $P$. При додаванні багаторозрядних $P$-кових чисел, починаючи від молодших розрядів, обчислюються суми розрядів і одиниць перенесення від попереднього розряду. Остача від ділення на $P$ залишається у відповідному розряді результату, а частка переноситься в наступний розряд.

**Приклад 1.6**

У цьому прикладі обчислимо суму двох десяткових чисел $548_{10}$ і $54_{10}$, а також їх вісімкових та шістнадцяткових еквівалентів — $1044_8$ і $66_8$, $224_{16}$ і $36_{16}$ (у дужках зверху над значеннями доданків вказуються одиниці перенесення у відповідні розряди).

$$\begin{array}{r} (110) \\ 548_{10} \\ +\ 54_{10} \\ \hline 602_{10} \end{array} \qquad \begin{array}{r} (110) \\ 1044_8 \\ +\ 66_8 \\ \hline 1132_8 \end{array} \qquad \begin{array}{r} \\ 224_{16} \\ +\ 36_{16} \\ \hline 25\text{A}_{16} \end{array}$$

Віднімання в будь-якій системі числення виконується за тими самими правилами, що й у десятковій.

**Приклад 1.7**

Віднімемо одиницю від чисел $100_2$, $100_8$ і $100_{16}$:

$$\begin{array}{r} 100_2 \\ -\ 1_2 \\ \hline 11_2 \end{array} \qquad \begin{array}{r} 100_8 \\ -\ 1_8 \\ \hline 77_8 \end{array} \qquad \begin{array}{r} 100_{16} \\ -\ 1_{16} \\ \hline \text{FF}_{16} \end{array}$$

Операцію множення у довільній системі числення можна виконати за допомогою методу, який подібний до методу множення у стовпчик. Обчислення добутку двох $n$-розрядних чисел без знака зводиться до формування часткових добутків для кожної цифри множника і їх подальшого додавання. Коли обидва операнди є двійковими числами, процедура обчислення часткових добутків значно спрощується: якщо цифра множника дорівнює нулю, то і частковий добуток теж становить нуль, якщо одиниці, частковий добуток дорівнює значенню множеного. Перед додаванням кожний частковий добуток необхідно зсунути на один розряд вліво відносно попереднього добутку. Таким чином, результатом множення двох $n$-розрядних двійкових чисел може бути число, яке має $2n$ розрядів.

**Приклад 1.8**

Обчислимо добуток двох чисел у десятковій, вісімковій і шістнадцятковій системах числення. Співмножниками будуть числа $29_{10} = 35_8 = 1D_{16}$, $19_{10} = 23_8 = 13_{16}$.

$$\begin{array}{r} 29_{10} \\ \times\ 19_{10} \\ \hline + \quad 261_{10} \\ \quad\ 29_{10} \\ \hline 551_{10} \end{array} \qquad \begin{array}{r} 35_8 \\ \times\ 23_8 \\ \hline + \quad 127_8 \\ \quad\ 72_8 \\ \hline 1047_8 \end{array} \qquad \begin{array}{r} 1D_{16} \\ \times\ 13_{16} \\ \hline + \quad 57_{16} \\ \quad\ 1D_{16} \\ \hline 227_{16} \end{array}$$

## 1.4.2. Зображення чисел у комп'ютері

Комп'ютери всіх типів мають велику кількість команд для виконання арифметичних операцій над числовими даними. Для того щоб зрозуміти, як комп'ютер виконує ці команди, необхідно знати, як числа зберігаються в пам'яті і як вони додаються та віднімаються. Зрозуміло, що в розрахунках використовуються як додатні, так і від'ємні числа, і нам потрібно їх певним чином зобразити у пам'яті комп'ютера.

У сучасних комп'ютерах здебільшого застосовуються два формата зображення чисел: числа з фіксованою комою і числа з плаваючою комою (fixed point, floating point). Перша з них дістала назву природної (або натуральної), друга — нормальної (експоненціальної, логарифмічної або так званого наукового запису).

### Числа з фіксованою комою

*Число з фіксованою комою* — це формат зображення числа з незмінним розташуванням коми, що відокремлює цілу частину числа від дробової. Числа у такому форматі записуються у вигляді $X = \pm a_{n-1}...a_1\, a_0\, a_{-1}\, a_{-2}...a_{-r}$ (рис. 1.3), де для запису цілої частини числа відводиться $n$ розрядів, а для дробової частини числа — $r$ розрядів.

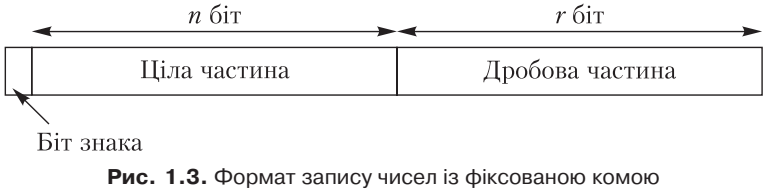


**Рис. 1.3.** Формат запису чисел із фіксованою комою

Розряд коду числа, в якому вказується знак, називається знаковим, а розряди, де знаходяться значущі цифри, називаються цифровими розрядами коду. Знаковий розряд дорівнює 0 для додатних чисел, та 1 — для від'ємних. Положення коми відносно розрядів числа фіксується й у процесі обчислень не змінюється. В самому коді числа кома фізично ніяк не вказується, вона лише «мається на увазі».

У комп'ютері числа з фіксованою комою є одним із базових різновидів числових даних. Припустивши, що число не має дробових розрядів, відразу одержимо множину цілих чисел. Отже, числа з фіксованою комою можна умовно поділити на цілі числа і числа з дробовою частиною. Для роботи з цілими (знаковими і беззнаковими) числами, розмір яких у пам'яті становить 1, 2, 4 і 8 байт, процесор має спеціальні команди. Для процесорів Intel характерним є те, що в них не використовуються числа з фіксованою комою, якщо не вказувати даних, які зображують цілі числа.

Якщо число має цілу та дробову частини (є мішаним), то арифметичні дії над ним виконуються так, нібито це число є цілим (хоч насправді воно не є таким). Для спрощення операцій над такими числами положення коми фіксується або перед старшим цифровим розрядом, або після молодшого. У першому випадку можуть бути зображені тільки правильні дроби (за модулем менші одиниці), у другому — тільки цілі числа. Останній формат найбільш поширений, тому надалі поняття «фіксована кома» зв'язуватимемо з цілими числами, а операції з числами у форматі з фіксованою комою характеризуватимемо як операції над цілими числами (зі знаком і без знака).

Використання чисел у форматі з фіксованою комою значно спрощує апаратну реалізацію арифметико-логічного пристрою комп'ютера і зменшує час виконання машинних команд.

Для зображення додатних та від'ємних цілих чисел у комп'ютері застосовуються прямий, обернений та додатковий коди. Додатні числа у прямому, оберненому та додатковому кодах записуються однаково, а саме двійковим кодом числа з цифрою 0 у знаковому розряді, а від'ємні — по-різному.

*Прямий код* від'ємного числа відрізняється від прямого коду додатного числа тим, що значення знакового розряду (старшого біта числа) дорівнює не 0, а 1. Наприклад, прямим кодом числа 5 є 0101, а числа 127 — 01111111. Відповідно, прямий код числа –5 становить 1101, а код числа –127 записується так: 11111111.

*Обернений код* (чи доповнення до одиниці) від'ємного числа можна отримати шляхом доповнення кожного розряду відповідного додатного значення до одиниці, тобто у всіх розрядах, у тому числі знаковому, нулі потрібно замінити одиницями, а одиниці — нулями. Ця операція еквівалентна відніманню цього числа від $2^n-1$ (наприклад, від 1111 для чотирирозрядних чисел). Таким чином, обернений код числа –5 записується як 1010, а код числа –127 — як 10000000.

*Додатковий код* від'ємного числа можна отримати додаванням одиниці до молодшого розряду оберненого коду або відніманням модуля числа від $2^n$. Наприклад, додатковий код числа –5 записується як 1011, а код числа –127 — як 10000001.

Порівнюючи всі три коди з погляду ефективності виконання арифметичних операцій комп'ютером, слід зазначити, що прямий код найменше підходить для виконання арифметичних операцій, обернений код є більш придатним, а найефективнішим вважається додатковий код.

### Арифметичні операції над цілими числами

У пам'яті комп'ютера операнди, як правило, зберігаються у прямому або додатковому кодах. Зберігати числа в оберненому коді недоцільно, тому що такий код

числа можна дуже легко отримати з прямого коду простою інверсією. Використання прямого коду числа дає переваги під час виконання деяких операцій (наприклад, множення) над операндами, що зображені у форматі з плаваючою комою. Ця обставина і є визначальним фактором при виборі способу збереження інформації у пам'яті комп'ютера.

При додаванні двох $n$-розрядних двійкових чисел (один біт знака та $n-1$ бітів значущих цифр) слід керуватися такими правилами.

1. Для того щоб додати два числа, необхідно додати їх $n$-бітове зображення, не враховуючи біт перенесення, який формується у старшому розряді. Сумою буде значення у додатковому коді, якщо результат належить діапазону від $-2^{n-1}$ до $2^{n-1}-1$.
2. Для віднімання числа $X$ від числа $Y$ необхідно спочатку обчислити додатковий код числа $Y$, а потім додати його до числа $X$, враховуючи перше правило. Результатом буде значення у додатковому коді, за умови, що воно належить діапазону від $-2^{n-1}$ до $2^{n-1}-1$.

При додаванні двох $n$-розрядних двійкових чисел (один біт знака та $n-1$ бітів значущих цифр) із використанням прямого та додаткового кодів можливий результат, кількість значущих цифр у якому дорівнюватиме $n$, тобто результат, що не належатиме діапазону допустимих значень. Таку ситуацію називають *переповненням* розрядної сітки. Ознакою переповнення є перенесення в знаковий розряд результату за відсутності перенесення зі знакового розряду (*додатне переповнення*) або наявність перенесення зі знакового розряду за відсутності перенесення в знаковий розряд (*від'ємне переповнення*). Якщо є перенесення як в знаковий розряд результату, так і зі знакового розряду або якщо ці перенесення відсутні, то переповнення не відбувається. При додатному переповненні результат операції від'ємний, при від'ємному — додатний.

Слід зазначити, що переповнення може виникнути лише при додаванні чисел з однаковими знаками. Додавання чисел із різними знаками не призводить до переповнення.

**Приклад 1.9**

Наведемо приклад додатного переповнення. В області пам'яті обсягом 1 байт зберігається число $+65_{10}$, прямим кодом якого є 0100 0001 (знаковий розряд містить 0). Якщо додати до цього числа таке саме, утвориться від'ємне число 1000 0010, оскільки відбудеться перенесення одиниці в знаковий розряд (знаковий біт результату міститиме 1). Десятковим еквівалентом числа, що утвориться, буде значення $-2_{10}$.

$$\begin{array}{rr} & 0100\ 0001 \\ + & 0100\ 0001 \\ \hline & 1000\ 0010 \end{array}$$

Далі наведемо приклад від'ємного переповнення. В області пам'яті обсягом 1 байт зберігається число $-65_{10}$, додатковий код якого — 1011 1111 (знаковий розряд містить 1). Якщо додати до цього числа таке саме, то утвориться додатне

число 0111 1110, оскільки відбудеться перенесення одиниці зі знакового розряду (знаковий біт результату міститиме 0). Десятковим еквівалентом двійкового числа, що утвориться, буде число $+126_{10}$.

$$\begin{array}{r} 1011\ 1111 \\ +\ 1011\ 1111 \\ \hline 0111\ 1110 \end{array}$$

Виявити переповнення розрядної сітки буде простіше за умови використання *модифікованого додаткового коду*, коли для збереження знака числа відводиться не один, а два розряди. Ці розряди беруть участь в арифметичній операції нарівні з цифровими. За відсутності переповнення обидва знакові розряди містять однакові значення. Розбіжність у значеннях цих розрядів є ознакою того, що виникло переповнення.

Множення, порівняно з додаванням та відніманням, є більш складною операцією з точки зору як програмної, так і апаратної реалізації. У більшості комп'ютерів операція множення виконується як послідовність операцій додавання та зсуву. Для реалізації цієї операції у комп'ютерах застосовуються кілька різних алгоритмів, але кожен із них передбачає наявність в АЛП регістрів множника та множеного, а також спеціального регістру, який називається *накопичувальним суматором*. До початку операції множення цей регістр містить число 0. Під час операції у ньому зберігаються результати проміжних обчислень, а по завершенні — кінцевий результат.

**Приклад 1.10**

Приклад множення двох додатних чисел наведено нижче.

$$\begin{array}{r} \times\ 1101 \\ \underline{\quad 1011} \\ 1101 \\ 1101 \\ \underline{1101\ \ } \\ 10001111 \end{array}$$

Перевіримо:

$1101_2=13$,
$1011_2=11$,
$13\times11=143$,
$10001111_2=2^7+2^3+2^2+2^1+2^0=128+8+4+2+1=143$

Ділення є дещо складнішою, ніж множення, операцією, але її реалізація в комп'ютері ґрунтується на тих самих принципах. Основою є загальновідомий спосіб ділення за допомогою операцій віднімання чи додавання та зсуву (ділення у стовпчик). Ділення виконується як послідовність віднімань дільника спочатку від ді-

леного, а потім і від часткових залишків, які утворюються у процесі ділення. Ділене зазвичай займає $2n$ розрядів, тоді як дільник, частка та залишок є $n$-розрядними.

### Числа з плаваючою комою

У прикладних задачах програмістам досить часто доводиться оперувати дуже великими або дуже маленькими дійсними числами, наприклад такими, як маса Сонця, що складає $2\times10^{30}$ кг, або маса електрона, яка становить $9\times10^{-28}$ г. Записати в пам'ять подібні числа, враховуючи всі значущі цифри, і виконати над ними арифметичні операції, використовуючи арифметику з фіксованою комою, неможливо. У цьому разі для запису чисел використовується формат із плаваючою комою, коли кожне число розбивається на дві групи цифр. Перша група цифр називається мантисою, друга — порядком. Число записується у вигляді добутку.

$$Y = \pm M \times S^{\pm p}.$$

Тут $Y$ — значення дійсного числа; $M$ — мантиса числа; $S$ — основа системи числення; $p$ — порядок числа.

*Мантиса* (дріб зі знаком) і порядок (ціле число зі знаком) зображуються в системі числення з основою $S$. Знак числа збігається зі знаком мантиси. *Порядок p* є додатним або від'ємним цілим числом і визначає положення коми в числі $Y$. Таким чином, у мантисі зберігаються значущі цифри числа, а порядок визначає його величину.

Дійсні числа записуються в 4, 6, 8 або 10 байтах. Для того щоб в комірці оперативної пам'яті не зберігати знак порядку, замість нього використовується характеристика. Вона утворюється додаванням до порядку певного цілого числа. Це число добирається так, щоб характеристика завжди була додатною. Характеристику іноді називають *зсуненим порядком*. При цьому формат зображення дійсних чисел є таким, як показано на рис. 1.4.

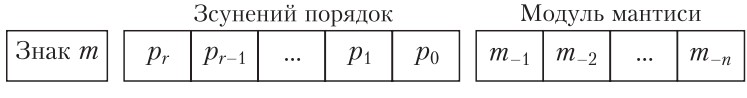


**Рис. 1.4.** Формат чисел із плаваючою комою зі зсуненим порядком

Для збільшення кількості значущих цифр у зображенні дійсного числа і запобігання переповненню при виконанні арифметичних операцій мантису нормалізують. *Нормалізація* означає, що значення мантиси має знаходитися в діапазоні від $S^{-1}$ до 1, де $S$ — основа системи числення. У двійковій системі числення $S = 2$ і мантиса набуває значення від $2^{-1}$ до 1. Це означає, що мантиса будь-якого числа, зображеного у форматі з плаваючою комою, має починатися з одиниці у двійковій системі числення. Наведений метод нормалізації є класичним, при якому результат нормалізації зображується у вигляді правильного дробу, тобто з одиницею після коми і нулем у цілій частині числа (розглядається двійкова система). Є різні алгоритми нормалізації мантиси. В IBM-сумісних комп'ютерах старший біт нормалізованої мантиси розташований зліва від коми. В оперативній пам'яті

цей біт не зберігається, тобто він є прихованим, а його вага дорівнює одиниці, тому мантиса належить інтервалу $1 \le M < 2$. Нормалізована мантиса додатних і від'ємних чисел зображується у прямому коді.

Діапазон чисел з плаваючою комою залежить від кількості розрядів, виділених для зображення порядку і мантиси, а також від основи системи числення. Слід зазначити, що розташування та довжини полів у зображеннях дійсних чисел визначаються типом комп'ютера та мовою програмування.

Визначимо діапазон дійсних чисел, що зображуються чотирма байтами (32 двійковими розрядами). У нормалізованій мантисі перша значуща цифра записана зліва від коми, а справа розташовуються 23 розряди (рис. 1.5). Тому максимальне значення мантиси $M_{max} = 1{,}111..11 = 1+1/2+1/4+1/8+...= 2$, а її мінімальне значення $M_{min} = 1{,}000..00 = 1$ для додатних чисел і $M_{max} = -1$ та $M_{min} = -2$ для від'ємних чисел. Максимальне значення порядку числа визначається як $P_{max} = 2^8 - 2 = 254_{10} = 11111110_2$, а мінімальне значення порядку — як $P_{min} = 00000001 = 1$. Після цього можна визначити діапазон зображення додатних дійсних чисел: від $D_{min} = M_{min} \times 2^{(1-127)} \approx 1{,}17 \times 10^{-38}$ до $D_{max} = M_{max} \times 2^{(254-127)} \approx 3{,}4 \times 10^{38}$.

**Приклад 1.11**

Зобразимо десяткове число $-15{,}375_{10}$ у форматі з плаваючою комою. Спочатку переведемо десяткове число в двійкову систему числення і отримаємо його двійковий еквівалент, тобто $-1111{,}011_2$. Після цього нормалізуємо число і отримаємо значення $-1{,}111011 \times 2^3$, де порядок числа дорівнює трьом. Оскільки число від'ємне, то знаковий біт міститиме одиницю. Обчислюючи характеристику числа, отримаємо значення $2^7 - 1 + 3 = 130_{10}$. Переведення значення характеристики в двійкову систему дає результат $1000\ 0010_2$. Одиниця цілої частини нормалізованого числа відкидається, тому мантиса дійсного числа має вигляд $111011_2$. Тепер можна записати машинне зображення дійсного числа, використавши для цього чотири байти:

11000001 01110110 00000000 00000000.

Точність обчислень при використанні чисел із плаваючою комою визначається кількістю розрядів мантиси, тобто числом достовірних десяткових цифр. Оскільки $2^{23}$ приблизно дорівнює $10^7$, при наявності 23 двійкових розрядів мантиси достовірними є тільки 6–7 значущих десяткових знаків.

Комп'ютери, які серійно випускаються різними фірмами, використовують різні формати зображення чисел. В першу чергу це стосується чисел із плаваючою комою. Щоб спростити використання програм, розроблених для різних платформ, було запропоновано стандарт IEEE 754, який регламентує формат запису чисел із плаваючою комою. Нині цей стандарт широко використовується, і практично всі розробники комп'ютерів дотримуються його вимог. Цей стандарт визначає два базових формати (32- та 64-бітовий), з 8- та 11-розрядними порядками відповідно (рис. 1.5).

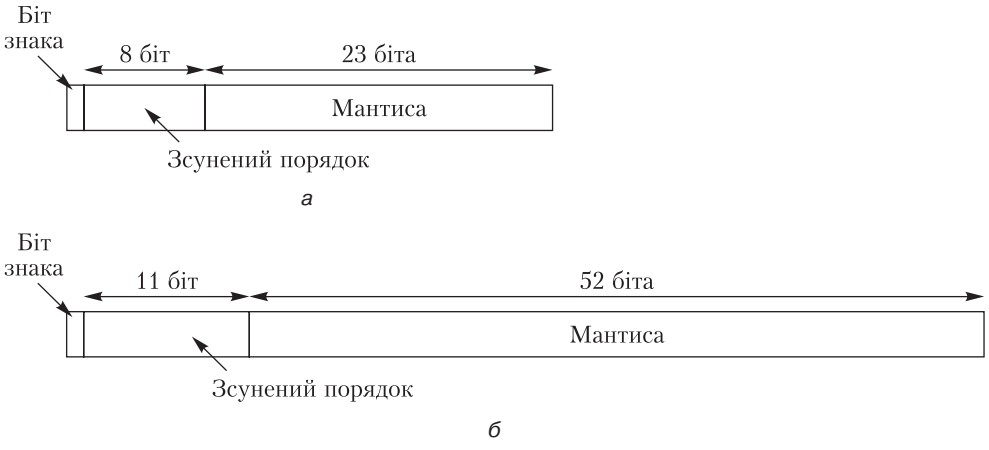


**Рис. 1.5.** Основні формати чисел з плаваючою комою в стандарті IEEE 754: 32-бітовий формат (*а*); 64-бітовий формат (*б*)

Крім двох основних, у стандарті IEEE 754 запропоновано також два розширених формати, одинарний та подвійний, в яких зарезервовано додаткові біти для порядку та мантиси.

## 1.5. Типи комп'ютерів

Різнотиповість комп'ютерів обумовлюється передусім різнотиповістю обчислень і задач, для виконання яких вони призначені. З часом виникають все нові класи задач, що, у свою чергу, приводить до появи комп'ютерів нових типів. Тому наведена нижче класифікація містить лише найбільш поширені типи комп'ютерів:

- суперкомп'ютери;
- мейнфрейми;
- сервери різних типів;
- персональні комп'ютери та робочі станції.

*Суперкомп'ютери* — спеціальний тип комп'ютерів, що створюється для розв'язування надзвичайно складних обчислювальних задач (підготовки прогнозів погоди, моделювання складних процесів та явищ природи, оброблення великих обсягів інформації). Найпотужніший у світі суперкомп'ютер (Earth Simulator), за даними 2003 року, встановлено в Центрі моделювання Землі в Йокогамі (Японія). Він призначений для моделювання основних властивостей складових кліматичної системи Землі: атмосфери, океану, кріосфери, поверхні суші і біосфери, а також зовнішніх і внутрішніх факторів у системі, яка визначає глобальний клімат і його зміни. Суперкомп'ютери — це багатопроцесорні або багатомашинні комплекси продуктивністю понад сотні мільярдів операцій за секунду. Основний принцип роботи всіх суперкомп'ютерів — принцип паралелізму. Суть його полягає в тому, що комп'ютер здатний виконувати одночасно (паралельно) велику кількість операцій.

Однією з провідних компаній світу по виробництву суперкомп'ютерів багато років вважалась компанія Cray Research. Її засновник, людина-легенда Сеймур Крей, уже в середині 70-х років побудував комп'ютер Cray-1, що вражав світ своєю швидкодією: десятки і навіть сотні мільйонів арифметичних операцій за секунду. Сьогодні на ринку суперкомп'ютерів домінують більш сучасні системи компаній NEC, HP, IBM.

*Мейнфрейм* — це синонім поняття «велика універсальна електронно-обчислювальна машина». Мейнфрейми і на сьогодні залишаються найбільш потужними (якщо не враховувати суперкомп'ютери) обчислювальними системами загального призначення, які можуть експлуатуватися у безперервному режимі. Зазвичай мейнфрейми — це багатопроцесорні системи, що містять один або кілька центральних і периферійних процесорів із спільною пам'яттю, зв'язаних між собою високошвидкісними магістралями передачі даних. При цьому основне навантаження щодо обчислень покладено на центральні процесори, а периферійні процесори забезпечують роботу з різноманітними периферійними пристроями.

Основними постачальниками мейнфреймів є відомі комп'ютерні компанії Amdahl, ICL, Siemens Nixdorf і деякі інші, але провідна роль у цій галузі, безумовно, належить компанії IBM. Саме архітектура випущеної в 1964 році системи IBM/360 і її наступних поколінь стала зразком обчислювальних систем цього типу. В СРСР протягом багатьох років випускалися машини серії ЄС ЕОМ (Єдина серія електронно-обчислювальних машин), що були вітчизняним аналогом цієї системи.

Комп'ютер, що працює в локальній або глобальній мережі, може спеціалізуватися на обслуговуванні інших комп'ютерів. Такий комп'ютер називається *сервером* (від англ. serve — обслуговувати, керувати). Є кілька типів серверів, орієнтованих на різні сфери застосування: файловий сервер, сервер бази даних, сервер друку, обчислювальний сервер, сервер програмних застосувань. Тип сервера визначається видом ресурсу, яким він керує (файлова система, база даних, принтери, процесори або пакети прикладних програм).

Існує також класифікація серверів на основі масштабу мережі, в якій вони використовуються: сервер робочої групи, сервер відділу або сервер масштабу підприємства (його ще називають корпоративним сервером). Загалом, ці класифікації досить умовні. Наприклад, розмір групи може змінюватися в діапазоні від кількох людей до декількох сотень, а сервер відділу може обслуговувати від 20 до 150 користувачів. Очевидно, що вимоги до складу устаткування і програмного забезпечення сервера, його надійності і продуктивності дуже варіюються залежно від числа користувачів і характеру розв'язуваних ними задач.

Незважаючи на велике різноманіття типів і моделей обчислювальних систем, у більшості користувачів поняття «комп'ютер» асоціюється в першу чергу з персональним комп'ютером. Перший персональний комп'ютер з'явився у 1975 році. І відразу стало зрозуміло, що невисока ціна і досить потужні обчислювальні можливості комп'ютерів цього класу сприятимуть їхньому швидкому поширенню.

Персональні комп'ютери зробили революцію у професійній діяльності мільйонів людей, вплинули на всі сфери розвитку суспільства. Комп'ютери цього типу стали незамінним інструментом у роботі інженерів і вчених. Особливо значною

їхня роль є при проведенні наукових експериментів, що вимагають складних і тривалих обчислень.

Сучасний світ наповнений не лише звичними персональними комп'ютерами, а й комп'ютерами-невидимками — мікропроцесорами, що є комп'ютерами у мініатюрі. Крім блоку обробки даних такий пристрій містить блок керування і навіть пам'ять. Це означає, що мікропроцесор здатний автономно виконувати всі необхідні дії з інформацією. Масове поширення мікропроцесори одержали у сучасній техніці, якою можна керувати за допомогою обмеженої послідовності команд. Наприклад, керування сучасним двигуном (забезпечення економних витрат палива, обмеження максимальної швидкості руху, контроль за справністю тощо) немислиме без використання мікропроцесорів. Ще однією сферою їхнього використання є побутова техніка — застосування мікропроцесорів додає їй нових споживчих якостей.

## 1.6. Програмне забезпечення

Під програмним забезпеченням розуміють сукупність усіх програм і службових даних, призначених для керування комп'ютером. Деякі програми є вбудованими в апаратні компоненти комп'ютера, однак для забезпечення більшої гнучкості їх зазвичай записують на жорсткий диск, компакт-диск або інші зовнішні носії даних. У цьому разі їх необхідно щораз заново завантажувати в оперативну пам'ять при запуску комп'ютера чи перед виконанням конкретної програми. Програмне забезпечення складається з файлів програм, що керують роботою комп'ютера.

*Файл* — це послідовність літерно-цифрових символів або двійкових даних; до нього можна звертатися за вказаним іменем. У файлах зберігаються програми та дані. Кожний файл має ім'я, розмір, дату створення та певне місце розташування на диску. Залежно від характеру інформації, яка записується у файл, він може мати те чи інше розширення імені, що задається трьома латинськими літерами. Наприклад, текстові файли можуть позначатись розширенням txt, графічні файли — розширенням bmp, програми — розширенням exe тощо. Ім'я, розширення імені, дата створення, розмір файла є його атрибутами. Значення атрибутів файла зберігається у каталогах (папках, директоріях).

*Каталогом* називається група файлів, об'єднаних одним іменем. Слід зазначити, що каталог можна включити до іншого каталогу. Головний каталог диска називається *кореневим каталогом*. Ім'я кореневого каталогу складається з імені диска та символу двокрапки. Місцезнаходження файла на диску визначається маршрутом, який записується у вигляді послідовності імен каталогів, починаючи з кореневого, наприклад: C:\Windows\System. Повне ім'я файла, або його специфікація, записується у вигляді маршруту та імені файла: C:\Windows\System\help.exe.

За своїм призначенням усе програмне забезпечення можна розділити на дві основні категорії:

- системні програми;
- прикладні програми.

*Системні програми*, або *системне програмне забезпечення,* — це набір програм, призначених для виконання таких функцій:

- ✦ отримання й інтерпретація команд користувача;
- ✦ керування процесами збереження файлів на зовнішніх запам'ятовуючих пристроях, а також зчитування інформації із зазначених пристроїв в оперативну пам'ять;
- ✦ запуск і керування процесом виконання прикладних програм, таких як текстові редактори, електронні таблиці або ігри, а також програм, створених користувачем;
- ✦ керування взаємодією апаратних і програмних ресурсів комп'ютера під час виконання прикладних програм.

Отже, системне програмне забезпечення відповідає за координування всіх операцій, що їх пристрої комп'ютерної системи виконують під час реалізації прикладних програм.

*Прикладні програми* призначені для розв'язання задач певних класів, наприклад для математичних обчислень, оброблення рядків тексту або відеоінформації. Для розробки прикладних програм використовуються мови програмування, і зокрема C, C++, Java, Basic, які дозволяють програмістові вказати дії, що їх має виконати програма.

Ключовим компонентом системного програмного забезпечення є *операційна система* (ОС) — комплекс програм, який використовується для керування взаємодією різних пристроїв комп'ютера при виконанні прикладних програм. Компоненти операційної системи відповідають за надання прикладним програмам ресурсів комп'ютера — оперативної пам'яті і пам'яті на магнітних дисках, пристроїв введення-виведення тощо.

Операційна система, з одного боку, виступає як інтерфейс між апаратними засобами комп'ютера і користувачем, а з іншого, забезпечує ефективне використання останнім ресурсів комп'ютера й організацію виконання прикладних програм. Усі компоненти програмного забезпечення комп'ютера працюють під керуванням операційної системи і жоден з них не має безпосереднього доступу до апаратури. Навіть сам користувач взаємодіє зі своїми програмами через інтерфейс операційної системи.

Для того щоб виконати програму, її потрібно спочатку завантажити до оперативної пам'яті із зовнішнього носія, як правило, з диска. Це робиться під час виконання спеціальної програми, що входить до складу операційної системи, — *завантажувача*. Процес переписування програми із зовнішнього носія в оперативну пам'ять називається *завантаженням*. Як тільки це буде зроблено, програма почне виконуватись (рис. 1.6).

Коли обчислення завершаться, прикладна програма знову направить запит операційній системі. В результаті буде запущено відповідну програму операційної системи, що забезпечить пересилання потрібних даних на пристрій введення-виведення. При виконанні прикладної програми керування періодично передається то їй самій, то програмам операційної системи.

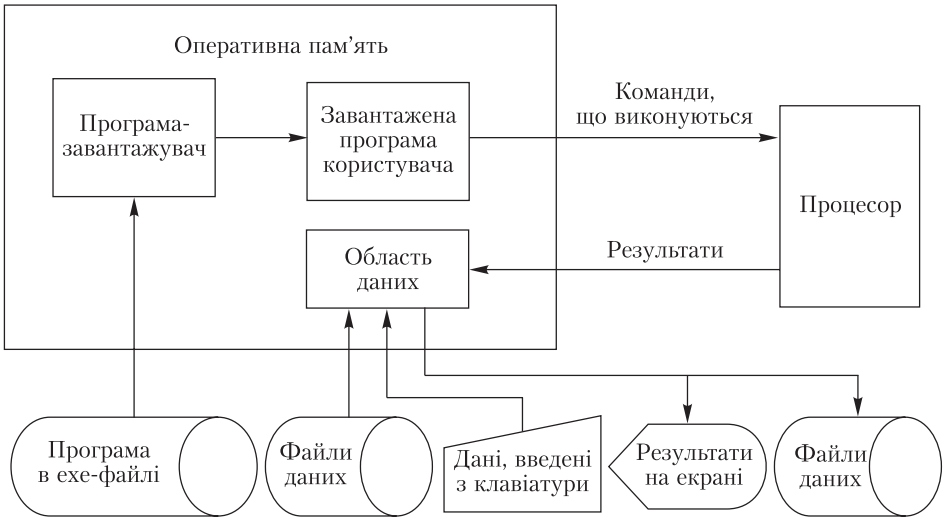


**Рис. 1.6.** Процес завантаження і виконання програми

## 1.7. Засоби створення програм

До засобів створення програм належать насамперед мови і системи програмування. Основна функція всіх мов програмування, крім машинної, полягає у тому, щоб надати програмісту засоби абстрагування від характеристик та особливостей апаратного забезпечення, на якому виконуватимуться програми. Системи програмування містять автоматизовані засоби розробки програм.

### 1.7.1. Класифікація мов програмування

У прикладі 1.1 використовувалися числові коди операцій і числові значення адрес комірок пам'яті. Такий спосіб написання програм називається *програмуванням у числових кодах,* а мова, якою записуються такі програми, — машинною. *Машинна мова* — це «природна мова» певного комп'ютера, яка визначається під час проектування його апаратних засобів. Машинні мови важкі для людського сприйняття. Тому природним виявилося прагнення автоматизувати процес написання програм, для того щоб полегшити працю програміста, частково поклавши його роботу на саму машину. Програмісти почали використовувати більш звичну для людини символьну форму опису обчислень, а перетворення програм у машинні коди здійснювали за допомогою програм трансляції (від англ. translation — переклад), які називаються *асемблерами* (від англ. to assemble — збирати).

Мови програмування, у яких числове кодування команд було замінено їх символьним зображенням, називалися мовами символічного кодування, а системи програмування — системами символічного кодування (ССК). Нині такі мови перетворилися в досить потужні засоби програмування, названі асемблерами.

Під час написання програм мовами асемблерного типу в ролі засобів програмування використовуються такі абстракції, як змінна та символьне зображення операцій, що дає змогу програмісту позбутися проблем, пов'язаних із формою зображення чисел, кодуванням операцій і розподілом пам'яті. Тепер перераховані вище проблеми має вирішувати програма-транслятор на основі інформації, яку їй передає розроблювач програм. Наступний фрагмент програми мовою асемблера є іншою реалізацією програми з прикладу 1.1.

**Приклад 1.12**

Припустимо, що комірки оперативної пам'яті з адресами 100, 101, 102 і 103 позначено відповідно як WORDA, WORDB, WORDC, WORDD. Для виконання операцій необхідні регістри процесора, які позначено як AX і BX. Команди «прочитати» й «записати» позначено як MOV, «додати» і «помножити» — як ADD та MUL. Тоді послідовність команд із прикладу 1.1 матиме такий вигляд (після крапки з комою в кожному рядку наведено коментарі до заданої дії):

```
MOV AX, WORDB  ;прочитати число за адресою WORDB і записати його до регістра AX
MOV BX, WORDC  ;прочитати число за адресою WORDC і записати його до регістра BX
ADD AX, BX     ;додати числа з регістрів AX і BX, суму записати в AX
MUL WORDD      ;помножити вміст регістру AX на число, записане за адресою WORDD
MOV WORDC, AX  ;записати число з регістра AX за адресою WORDC
```

З прикладу видно, що навіть проста програма, написана мовою асемблера, складається з довгої послідовності команд, за структурою близьких до машинних. Написати таку програму нелегко, до того ж потрібно знати дуже багато подробиць щодо устрою комп'ютера (наприклад, для чого призначено ті чи інші регістри, які адреси пам'яті можна використовувати, а які — ні). Тому програмування мовою асемблера — справа окремих програмістів.

Для прискорення процесу програмування були розроблені *мови програмування високого рівня*, які дозволяли писати програми, за формою близькі до людської мови, та використовували загальноприйняту математичну нотацію. Перша з них з'явилася наприкінці 1950-х років і називалася Fortran. Назва була скороченням від слів FORmula TRANslation, що у перекладі означає трансляція формул. Опис мовою високого рівня програми, наведеної у прикладі 1.12, може виглядати так:

```
r=b+c
a=r*d
```

Нагадаємо, що обчислюється вираз за формулою $a = (b + c) \times d$. У тексті програми іменами `a`, `b`, `c`, `d`, `r` позначені комірки пам'яті, в які записуються числа.

Одночасно з мовами високого рівня розроблялися *транслятори* (*компілятори*) — програмні засоби, призначені для перекладу високорівневих програм у машинні. Досвід створення мов високого рівня та їх трансляторів з роками накопичувався. Зокрема, було розроблено математичні основи та технологію реалізації цих програмних засобів. На сьогоднішній день кількість мов програмування й трансляторів вимірюється уже тисячами і продовжує зростати.

## 1.7.2. Технологія створення програми

Розглянемо процес створення програми у найбільш загальних рисах.

Розробка програми починається з постановки задачі, яку пропонує замовник. Іноді аналіз і уточнення задачі дають можливість формалізувати її постановку, в результаті з'являється математично точний і однозначний опис задачі. Після уточнення постановки задачі починається *проектування програми*. Як правило, в задачі можна виділити декілька *підзадач* і описати процес їх розв'язування окремо. Відповідно й алгоритм складається зі зв'язаних та узгоджених між собою частин, які описують процес розв'язання підзадач. У початковому алгоритмі дії подано в абстрактному вигляді, далекому від того, що може виконувати комп'ютер. Алгоритм уточнюють декілька разів і надають йому вигляд, за яким легко написати програму або її частину.

Написання програми або окремих її частин прийнято називати *кодуванням*, або *розробкою*. Найчастіше програму записують однією з мов високого рівня, але іноді деякі її частини записують різними мовами. Далі програма перекладається (транслюється) на машинну мову (зазвичай, частинами). Під час кодування програмісти можуть припускатися помилок. Процес виявлення й виправлення помилок називається *налагодженням* програми. Він дозволяє виявити помилки перелічених нижче типів.

1. Помилки, пов'язані з порушенням правил граматики в тексті програми, написаної мовою високого рівня. Їх можна виявити у процесі трансляції, тому вони називаються *помилками часу трансляції* (compiler error).
2. Помилки, що виявляються під час виконання робочої програми. Вони можуть виникати, наприклад, в результаті переповнення розрядної сітки чи при спробі видобути квадратний корінь із від'ємного числа. Такі помилки називаються *помилками часу виконання* (run time error).
3. Помилки, що не виявляються ні під час трансляції, ні під час виконання програми. Це змістові помилки, пов'язані з некоректністю логічних умов, неправильним використанням розрахункових формул і т. ін. Їх називають *семантичними*.
4. Помилки у вихідних даних.

Налагодження — це процес багаторазового виконання програми з різними варіантами даних, які вона має обробляти. Дані спеціально добираються таким чином, щоб можна було виявити якнайбільше помилок, якщо такі існують. Ця цілеспрямована перевірка працездатності програми називається *тестуванням*. Тестування не гарантує відсутності помилок у програмі, а лише дозволяє виявити деякі з них. Чим ретельніше проводиться тестування, тим більше помилок виявляється та виправляється.

Після налагодження програма проходить *дослідно-виробничу експлуатацію*, для якої необхідно розробити супровідні документи під назвами «Керівництво розробника програми» і «Керівництво користувача», які описують устрій та використання програми. Перший документ дає можливість виправляти помилки під час експлуатації програми та розвивати її надалі, а у другому пояснюється, як використовувати програму.

Проте може з'ясуватися, що під час проектування програми було обрано не найкращий алгоритм, через що програма виконується надто повільно або витрачає забагато ресурсів пам'яті, тобто є неефективною. До програми нерідко вносяться зміни, які вимагають повторного кодування і налагодження. Якщо у замовника з'являються нові ідеї, необхідно заново ставити задачу та проектувати програму.

З метою розробки ефективних з точки зору використаних ресурсів програм і прискорення процесу їх проектування були створені технології, тобто системи методів, які дозволяють «не робити зайвих кроків» на кожному з етапів, від аналізу задачі до налагодження програми. У 70-х роках минулого століття домінувала *технологія структурного програмування* — система правил створення програм, що характеризуються ясністю, простотою тестування та налагодження, легкістю модифікації. Починаючи з 90-х років ключовою технологією є *технологія об'єктно-орієнтованого програмування* — програмування на основі абстрактних типів даних. Застосування різних технологій потребує постійного удосконалення інструментів, які дозволяють створювати програми швидко, якісно й економічно. Такі інструменти називаються *системами програмування* і реалізують дуже важливий *принцип повторного використання коду* завдяки створенню модулів і компонентів. У процедурному програмуванні принцип повторного використання коду реалізується за допомогою процедур і функцій, які розглядаються в розділі 4.

## 1.7.3. Перетворення програми і система програмування

Розглянемо, як із програми, написаної мовою високого рівня, утворюється інша — машинна. Програму (вихідний текст) за допомогою спеціальної програми (вона називається *текстовим редактором*) найчастіше записують на диск у вигляді вихідного файла (рис. 1.7). Програма може складатися з кількох вихідних файлів — у великих програмах їх може нараховуватися десятки.

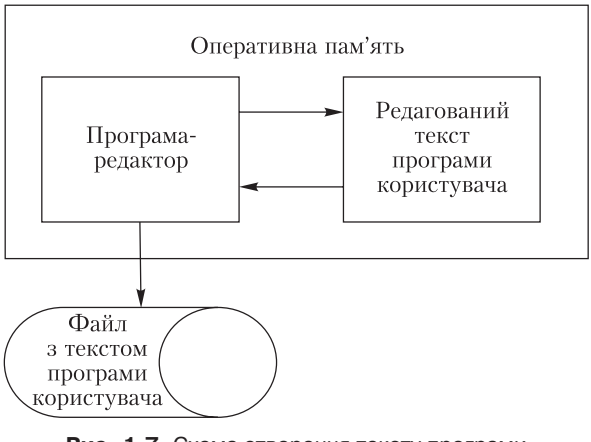


**Рис. 1.7.** Схема створення тексту програми

Під час роботи транслятора прочитується вихідний файл і створюється його машинний еквівалент — *об'єктний код*. Процес виконання програми-транслятора називається *трансляцією*, або *компіляцією* вихідного тексту.

Як правило, об'єктний код програми містить далеко не всі необхідні команди — програма може складатися з частин або включати підпрограми з бібліотек. Об'єктний код обробляється ще однією програмою — *редактором зв'язків*, або *компонувальником*, яка «збирає» (компонує) повний код програми і записує (завантажує) його або в оперативну пам'ять, або на диск у вигляді готового до виконання файла (рис. 1.8), який можна завантажити пізніше.

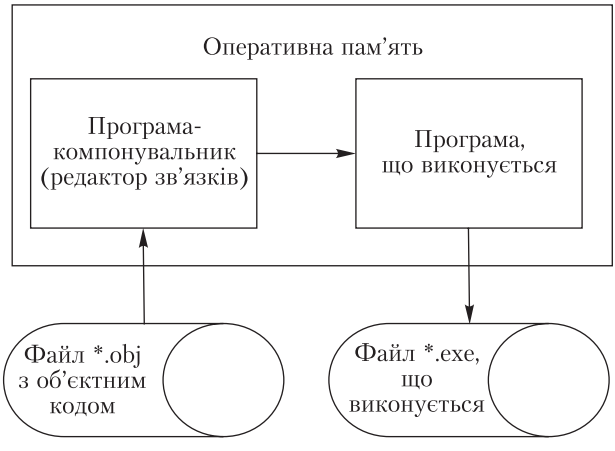


**Рис. 1.8.** Схема створення виконуваної машинної програми

*Інтерпретатор* на відміну від транслятора не створює машинну програму. Вхідні дані для інтерпретатора — це високорівнева програма й дані, що мають зчитуватися під час її виконання (рис. 1.9). *Інтерпретація* програми полягає в тому, що дії, задані програмою, відразу виконуються. Зазвичай інтерпретація вихідної програми відбувається повільніше, ніж виконання відповідної машинної програми.

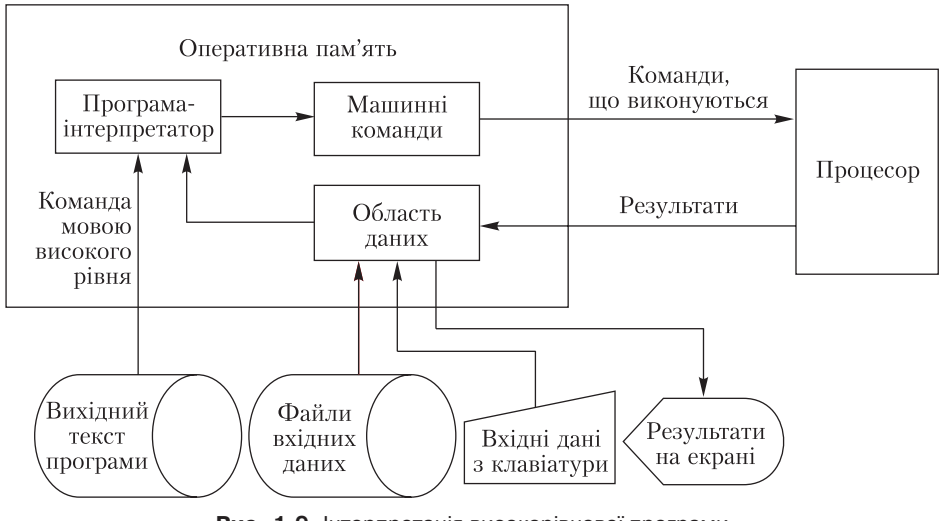


**Рис. 1.9.** Інтерпретація високорівневої програми

Ще один спосіб обробки вихідної програми поєднує в собі трансляцію й інтерпретацію. Програма перекладається (транслюється) не в машинні команди, а в деяке проміжне зображення, що потім інтерпретується. Такий підхід реалізовано, зокрема, в мові Java, яка швидко знайшла собі багатьох прихильників серед програмістів.

Інтерпретація програми здійснюється за допомогою такого інструменту, як *налагоджувач*. Він забезпечує інтерпретацію вихідної програми невеликими порціями (кроками) і дає можливість побачити результати виконання кожного кроку. Це полегшує пошук помилки (її локалізацію) у вихідній програмі.

Описані засоби (текстовий редактор, транслятор та (або) інтерпретатор, компонувальник, завантажувач і налагоджувач) разом утворюють *систему програмування*, або *інтегроване середовище розробки* (Integrated Development Environment, IDE). Крім них до складу IDE входить бібліотека стандартних підпрограм, які можна використовувати під час створення програми.

## 1.7.4. Походження та розвиток мови Pascal

Повернімося до історії. Майже одночасно з мовою Fortran було розроблено і реалізовано мовою Algol 60 (Algorythmic language — алгоритмічна мова). Її конструкції були набагато більше схожі на англійські фрази, ніж конструкції Fortran, тому логіка дій виражалася набагато природніше. На відміну від Fortran, мова Algol мала засоби реалізації рекурсії, за допомогою якої легко записуються численні алгоритми. Завдяки цим і деяким іншим властивостям Algol стала застосовуватися для запису алгоритмів, призначених для вивчення людиною.

У середині 1960-х років на основі мови Fortran було створено мову Basic (Beginner's All-purpose Symbolic Instruction Code — універсальний набір символьних команд для початківців). Basic була простішою за Fortran і дозволяла легко та швидко створювати нескладні програми, але не підтримувала структурне програмування і тому не використовувалася під час створення великомасштабних проектів.

Мову Pascal створив швейцарський учений Ніклаус Вірт спеціально для вивчення структурного програмування. Перший її опис було опубліковано в 1971 році. Вона походила від Algol 60 і двох мов на її основі, розроблених протягом 1960-х років (Algol W та Algol 68). Із середини 1970-х років Pascal стала основною серед мов, які вивчаються першими.

Найбільш поширені версії систем програмування на основі цієї мови для машин типу IBM PC і сумісних із ними почали розроблятися фірмою Borland International у 1983 році і дістали назву Turbo Pascal. Версії з номерами до 4.0 реалізували стандарт мови Pascal з незначними розширеннями. У четвертій версії (1987 рік) було істотно змінено технологію організації модульної структури програм. До складу цієї версії було включено вбудоване інтегроване середовище розробки (Integrated Development Environment, IDE). Починаючи з версії 5.5 (1989 рік) мова концептуально розширена, тобто доповнена засобами об'єктно-орієнтованого програмування. Версія 6.0 постачалася разом із об'єктною бібліотекою Turbo Vision і мовою Object Pascal, вбудованим асемблером (BASM), удосконаленими засобами налагодження в рамках нового IDE й іншими доповненнями. Найбільш

відомою є система програмування Turbo Pascal 7.0, створена фірмою Borland International ще в 1993 році, а також її розширена версія — Borland Pascal 7.0.

З 90-х років на основі мови Object Pascal почала розвиватися Delphi — потужна система програмування, яка використовується для професійної розробки великомасштабних проектів. Вона є однією з найпопулярніших систем програмування, що забезпечують так звану швидку розробку програм (Rapid Application Design, RAD). Ці системи забезпечують візуалізацію процесу створення програм і дозволяють істотно підвищити ефективність роботи програмістів.

Звичайно, більшість реальних програм створюються за допомогою систем програмування, в основу яких покладено не Pascal, а інші мови. Найбільш поширеною серед професіоналів є мова C++, продовжує набирати прихильників мова Java, свої сфери застосування мають такі мови, як Fortran, Basic та COBOL (Common Business Oriented Language — універсальна мова, орієнтована на розв'язання бізнес-задач) у їхніх сучасних версіях. Проте протягом професійної кар'єри програмістам неминуче доводиться освоювати декілька мов і систем програмування, а починати, на думку багатьох спеціалістів, краще все-таки з мови Pascal.

## 1.8. Поняття алгоритму й основні алгоритмічні структури

Одним з базових понять інформатики є поняття алгоритму. Алгоритм вказує, які операції, пов'язані з обробкою даних, і в якій послідовності треба виконати, щоб отримати розв'язок задачі. Алгоритм розрахований на певного виконавця, з погляду котрого вказівки мають бути елементарними, тобто такими, що можуть бути виконані безпосередньо, без подальшого тлумачення. Слово «алгоритм» походить від назви латинського перекладу трактату арабського математика IX століття Аль-Хорезмі («Трактат Аль-Хорезмі про арифметичне мистецтво індусів»).

### 1.8.1. Властивості та способи опису алгоритму

Алгоритм має задовольняти певним вимогам, серед яких потрібно виділити найважливіші.

- *Визначеність* — кожен крок алгоритму має інтерпретуватися виконавцем однозначно.
- *Результативність* — за скінченну кількість кроків алгоритм має приводити до розв'язання задачі або зупинятися через неможливість її розв'язати.
- *Дискретність* — кроки обчислювального процесу мають бути відокремлені один від одного.
- *Ефективність* — під час розв'язання задачі може використовуватися лише обмежений обсяг комп'ютерних ресурсів.
- *Масовість* — алгоритм розробляється у загальному вигляді, тобто його можна застосувати не лише до окремої задачі, але і до деякого класу задач, що розрізняються лише вхідними даними. При цьому вхідні дані мають належати деякій області, яка називається *областю застосовності алгоритму*.

Є декілька способів опису алгоритму: словесний опис послідовності дій, алгоритмічна мова, аналітичний опис у вигляді набору формул, графічний — у вигляді блок-схеми тощо. Формалізована система правил текстуального опису алгоритмів є одним із різновидів мов програмування. А мови, призначені для запису алгоритмічних структур, називаються *мовами структурного програмування*.

**Приклад 1.13**

Нехай квадратне рівняння $ax^2 + bx + c = 0$ задане дійсними коефіцієнтами $a$, $b$, $c$ за умови, що $a \neq 0$. Розглянемо задачу обчислення дійсних коренів цього рівняння. Послідовність операцій, які необхідно виконати, є такою.

1. Прочитати коефіцієнти $a$, $b$, $c$.
2. Обчислити дискримінант $d = b^2 - 4ac$.
3. Якщо $d > 0$, обчислити $x_1 = (-b - \sqrt{d})/2a, x_2 = (-b + \sqrt{d})/2a$ і написати ці числа; якщо $d = 0$, обчислити $x = -b/2a$ і написати це число; інакше написати «дійсних коренів немає».

Отже, у нас є задача — обчислити дійсні корені квадратного рівняння, заданого коефіцієнтами, і є опис процесу розв'язування задачі, або алгоритм. За цим описом ми виконуємо певну послідовність дій, тобто розв'язуємо задачу. Наведений приклад ілюструє словесний спосіб опису алгоритму.

---

Однією з наочних форм зображення алгоритму є *блок-схема*. Вона містить блоки, позначені геометричними фігурами. Усередині блоків записують елементарні дії. Блоки з'єднуються стрілками — так задається послідовність дій. Стрілки не є обов'язковими, якщо їхній напрямок відповідає просуванню «униз» і «праворуч». Кожній геометричній фігурі відповідає певний клас алгоритмічних інструкцій.

Зокрема, прямокутниками позначаються *операторні блоки*. Операторний блок може мати декілька входів і тільки один вихід. Це забезпечує однозначність у визначенні послідовності виконуваних дій. Дії, що позначаються такими блоками змінюють значення, форму подання чи розташування даних.

Ромбами позначається перевірка умови, залежно від результату якої визначається напрямок подальших обчислень. Тому блоки, що позначаються ромбами, називаються *умовними*. Оскільки результатом перевірки умови, записаної в умовному блоці, може бути значення «так» або «ні», тобто «істина» чи «хибність», блок має два виходи.

Алгоритм виконується у зовнішньому середовищі, де перебувають користувачі алгоритму. Від користувачів надходить інформація, яка в той чи інший спосіб оброблятиметься алгоритмом. Користувачі також повинні мати можливість отримати результати роботи алгоритму. Тому виникає потреба у блоках введення і виведення даних. Такі блоки позначаються паралелограмами.

Заокругленими прямокутниками позначається початок та кінець алгоритму. Початковий блок не має входів, а кінцевий блок — виходу. Обмежень на геометричні розміри блоків не існує.

Є три елементарні алгоритмічні структури: послідовності, розгалуження та повторення. Всі інші алгоритмічні структури утворюються з елементарних шляхом

заміни операторних блоків елементарними структурами. *Алгоритмічна структура послідовності* — це послідовність двох операторних блоків. Така структура дає вказівку виконувати одну інструкцію після іншої. Алгоритмічні структури розгалуження та повторення детально розглядатимуться в двох наступних розділах.

## 1.8.2. Алгоритмічна структура розгалуження

Алгоритмічна структура, що дозволяє виконавцеві алгоритму вибрати сценарій подальших дій залежно від істинності певного умовного твердження, називається *розгалуженням*. На блок-схемі (рис. 1.10) структури розгалуження позначаються ромбами. Дві стрілки, які відгалужуються від ромба, позначені словами «Так» і «Ні». Якщо записане всередині ромба умовне твердження є істинним, виконуються дії, на які вказує стрілка, позначена словом «Так». Якщо це твердження є хибним, виконуються дії, на які вказує стрілка, позначена словом «Ні».

Є декілька різновидів структури розгалуження. Структура, використана в алгоритмі обчислення коренів квадратного рівняння, є *альтернативним розгалуженням*. Альтернативне розгалуження припускає вибір виконавцем одного з двох можливих сценаріїв подальших дій залежно від істинності деякого умовного твердження. Крім альтернативного розгалуження є ще розгалуження у формі *множинного вибору альтернатив*. За множинного вибору може існувати більше двох сценаріїв дій виконавця. Вибір сценарію обумовлюється значенням деякого виразу.

## 1.8.3. Алгоритмічна структура повторення

Розглянемо задачу знаходження найбільшого спільного дільника двох натуральних чисел, застосувавши для її розв'язання алгоритм Евкліда. Позначимо найбільший спільний дільник чисел $a$ і $b$ через НСД($a$, $b$), а остачу від ділення $a$ на $b$ — через $a \bmod b$. Алгоритм Евкліда ґрунтується на тому факті, що НСД($a$, $b$) = = НСД($b$, $a \bmod b$), якщо $b \neq 0$, і НСД($a$, $b$) = $a$, якщо $b = 0$. Наприклад:

НСД(12, 5) = НСД(5, 12 mod 5) = НСД(5, 2)=НСД(2, 5 mod 2) = НСД(2, 1) = = НСД(1, 2 mod 1) = НСД(1, 0) = 1.

Алгоритм Евкліда вкрай незручно записувати за допомогою вказівок присвоєння та розгалуження, оскільки кількість таких вказівок залежить від значень $a$ і $b$, тобто не є відомою наперед. Ефективний спосіб розв'язання цієї задачі полягає у застосуванні структури *повторення* (яка називається ще *циклічною структурою*). Алгоритмічна структура повторення дає виконавцеві алгоритму вказівку повторювати деякі дії, поки певне умовне твердження істинне.

Твердження, істинність якого перевіряється під час виконання циклічної структури, на блок-схемі записується всередині ромба (як і у випадку структури розгалуження). Особливістю зображення циклічної структури на блок-схемі є те, що одна зі стрілок повинна «повертатися назад», тобто має утворюватися замкнений «цикл» із блоків та стрілок. Такий цикл має містити умовний блок, в якому записана умова продовження повторення. Одна зі стрілок, що відгалужуються від цього блоку, повинна брати участь у циклі, а інша — вказувати на блок поза циклом.

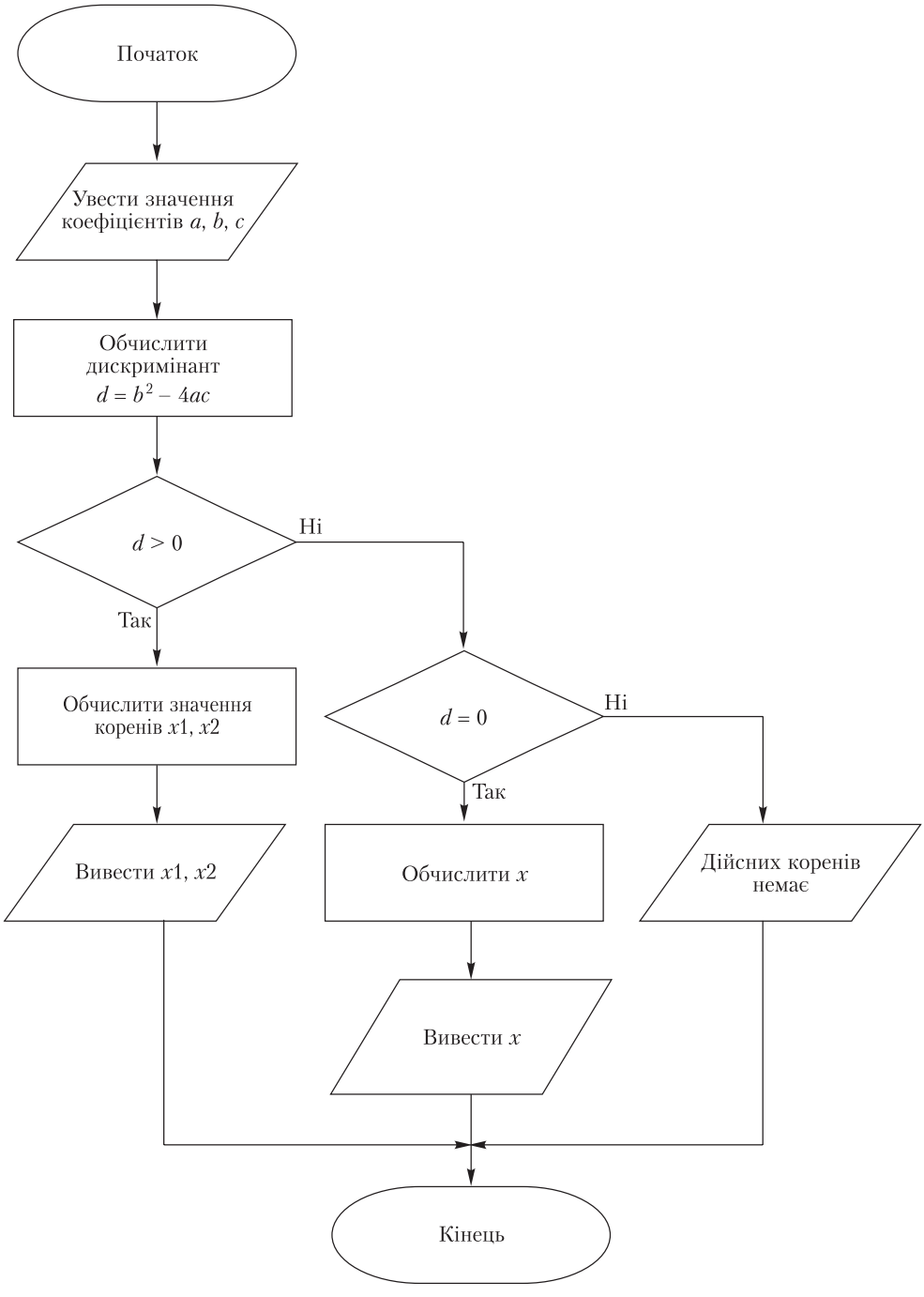


**Рис. 1.10.** Блок-схема алгоритму обчислення коренів квадратного рівняння

**Приклад 1.14**

Використовуючи структуру повторення, опишемо алгоритм Евкліда для знаходження найбільшого спільного дільника двох натуральних чисел.

1. Прочитати значення $a$ та $b$.
2. Поки $b \neq 0$, виконувати дії, описані у пунктах 3–5.
3. Обчислити величину $c = a \bmod b$.
4. Значення $a$ замінити значенням $b$.
5. Значення $b$ замінити значенням $c$.
6. Написати значення $a$.

Структура повторення реалізується кроками 2–5. Виконання описаних на кроках 3–5 дій повторюватиметься доти, доки істинним є твердження $b \neq 0$. Істинність цього твердження перевіряється на кроці 2. Така перевірка здійснюється кожного разу перед тим, як виконуються кроки 3–5. Коли твердження $b \neq 0$ стане хибним, кроки 3–5 будуть пропущені і після кроку 2 буде виконано одразу крок 6.

Блок-схема алгоритму Евкліда зображена на рис. 1.11.

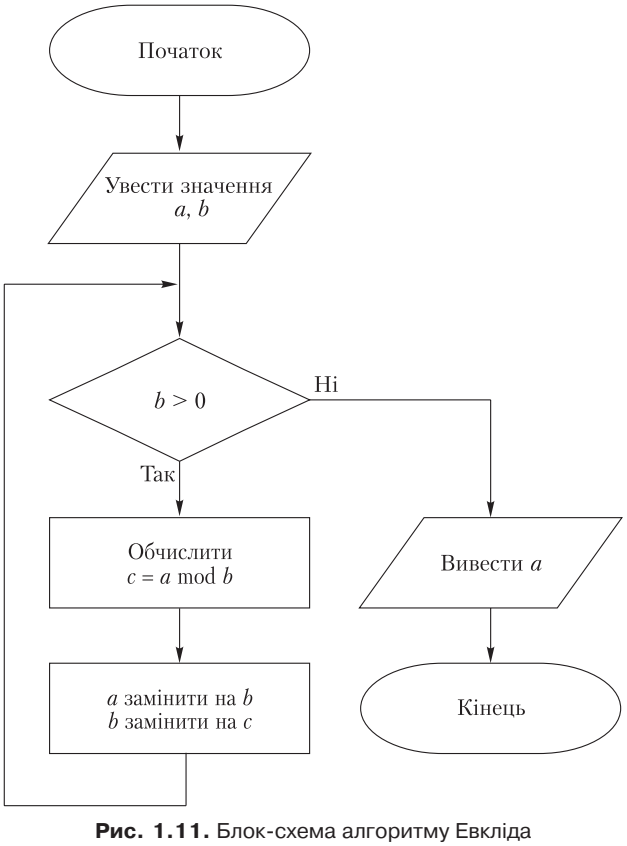


**Рис. 1.11.** Блок-схема алгоритму Евкліда

## Висновки

- Принципи програмного керування фон Неймана визначають ідеологію архітектури електронних обчислювальних машин.
- У комп'ютері числа зображуються у двійковій системі числення цифрами 0 та 1, які кодують два різних стійких стани елемента пам'яті, що називається бітом.
- Вісім послідовних бітів утворюють байт. Байт є базовою одиницею виміру довжини інформаційного повідомлення й обсягу пам'яті.
- Оперативна пам'ять може розглядатись як послідовність байтів. Кожен байт має свій номер — адресу.
- З погляду процесора програма — це розташована в оперативній пам'яті послідовність команд. Команди виконуються арифметико-логічним пристроєм процесора.
- Мови програмування високого рівня дозволяють записувати програми в зрозумілих для людини термінах.
- Переклад програми, що записана мовою високого рівня, на машинну мову (в об'єктний код) називається трансляцією і здійснюється програмою-транслятором.
- Програма-інтерпретатор виконує визначені програмою дії, не транслюючи програми у машинний код.
- Редактор зв'язків, або компонувальник, створює повний код програми шляхом поєднання об'єктного коду її окремих модулів.
- До складу інтегрованого середовища програмування входять текстовий редактор, транслятор та (або) інтерпретатор, компонувальник і налагоджувач.
- Розробка програмного забезпечення — це складний ітеративний процес, що, як правило, складається з таких етапів: постановка задачі, аналіз задачі та побудова її моделі, вибір або розробка алгоритму розв'язання задачі, кодування, налагодження та тестування, дослідно-виробнича експлуатація і супровід програмного забезпечення.
- Алгоритм — це набір елементарних вказівок, розташованих у певній послідовності. Алгоритм розрахований на певного потенційного виконавця, з погляду якого вказівки повинні бути елементарними, тобто такими, що можуть бути виконані безпосередньо, без подальшого тлумачення.
- Розгалуження — алгоритмічна структура, яка дозволяє виконавцеві алгоритму вибрати сценарій подальших дій залежно від виконання певних умов.
- Різновидами розгалуження є альтернативне розгалуження, що припускає вибір виконавцем одного з двох можливих сценаріїв подальших дій, та множинний вибір альтернатив, за якого таких сценаріїв може бути більше двох.
- Алгоритмічна структура повторення дає виконавцеві алгоритму вказівку повторювати деякі дії доти, доки певне умовне твердження є істинним.

## Контрольні запитання та завдання

1. Дайте означення поняття «архітектура комп'ютера».
2. Сформулюйте основні принципи фон Неймана.
3. Які функціональні блоки входять до складу комп'ютера?
4. У чому полягає призначення процесора та його регістрів?
5. Чим відрізняється кеш-пам'ять від інших типів оперативної пам'яті?
6. Які компоненти входять до складу зовнішньої пам'яті?
7. Що таке машинна команда?
8. Що таке позиційна система числення і як знайти значення числа за його записом у певній позиційній системі?
9. Чому інформація в комп'ютері записується у двійковій системі числення?
10. Для чого використовується шістнадцяткова система числення?
11. Як перевести десяткове число до будь-якої іншої системи числення?
12. Вкажіть основні форми зображення чисел у комп'ютері.
13. Що таке біт та байт?
14. Що таке програма?
15. Чим мова програмування високого рівня відрізняється від машинної мови?
16. Опишіть процес створення програми.
17. Що таке транслятор?
18. Чим відрізняється компіляція від інтерпретації?
19. Чим мова Pascal відрізняється від мов Fortran і Basic?
20. Перелічіть складові інтегрованого середовища розробки програм.

## Вправи

1. Побудувати блок-схему для визначення типу трикутника (рівносторонній, рівнобедрений, різносторонній) за довжинами його сторін. Якщо трикутник не можна побудувати, слід вивести повідомлення.
2. Побудувати блок-схему алгоритму піднесення цілого числа до цілого степеня.
3. Побудувати блок-схему алгоритму переведення цілого числа з десяткової системи числення до будь-якої іншої.
4. Побудувати блок-схему алгоритму переведення числа з будь-якої системи числення у десяткову.
5. Побудувати блок-схему алгоритму обчислення факторіала натурального числа.
6. Побудувати блок-схему алгоритму розв'язання такої старовинної задачі. Три лицарі та їх зброєносці мають переправитися на інший берег річки. Човен може вмістити двох осіб. Як їм переправитися за умови, що без свого лицаря жоден зброєносець не може перебувати у товаристві інших лицарів.

7. Записати алгоритм обміну значеннями між двома однотипними змінними за умови, що:
   а) можна використовувати третю змінну;
   б) третю змінну використовувати не можна (змінні числові).
8. Записати алгоритм «циклічного» обміну значеннями між трьома змінними $a$, $b$, $c$ ($a \leftarrow b$, $b \leftarrow c$, $c \leftarrow a$).
9. Нехай $z$ — числова змінна. Використовуючи лише операції множення та вказівки присвоєння, записати алгоритм обчислення таких значень: $z^{10}$, $z^{12}$, $z^{15}$, $z^{31}$. Операцій множення має бути якомога менше. Наприклад, обчислення $z^6$ можна задати так: $z^2 := z \times z$; $z^4 := z^2 \times z^2$; $z^6 := z^4 \times z^2$. Як бачимо, тут лише три операції множення замість п'яти, що виконуються при тривіальному способі обчислення: $z^6 = z \times z \times z \times z \times z \times z$.
10. Перевести десяткові числа 100, 255, 256, 640, 1024, 32767 до двійкової та шістнадцяткової систем числення.
11. Перевести двійкові числа 1000, 1111, 110 0100, 1111 1111, 1 0000 0000 до десяткової та шістнадцяткової систем числення.
12. Подати шістнадцяткові числа F1, FF, 4AB та FFFE у десятковій та двійковій системах числення.
13. Подати 36-кові числа ZY та 100 у десятковому записі (36-кові цифри A, B, ..., Y, Z позначають десяткові числа 10, 11, …, 34, 35 відповідно).
14. За основою $P$ та $P$-ковим записом дробу вказати його десяткове зображення:
   а) $P = 2$; 0,0001; 0,1111; 0,11111111;
   б) $P = 3$; 0,001; 0,22; 0,11;
   в) $P = 16$; 0,1; 0,FF; 0,8; 0,(7).
15. Записати $P$-кове зображення десяткового дробу $d$, де:
   а) $d = 0{,}5$, $P = 2, 3, 5, 8, 16, 20$;
   б) $d = 0{,}1$, $P = 2, 3, 5, 8, 16, 20$.
16. Указати двобайтовий додатковий код чисел –1, –8, –9, –32767, –32768.
17. Припустимо, що при додаванні та відніманні чисел перенесення зі старшого розряду стає вмістом знакового розряду, а перенесення зі знакового розряду втрачається. Вказати значення виразу (величини *maxi* та *mini* позначають максимальне та мінімальне цілі числа):
   а) *maxi* + 1;
   б) *mini* – 1.
18. Обчислити мінімальне та максимальне за модулем скінченні дійсні числа, що зображуються за допомогою:
   а) 4 байтів з $d = 8$, $r = 23$;
   б) 8 байтів з $d = 11$, $r = 52$;
   в) 10 байтів з $d = 16$, $r = 63$.

   Тут $r$ — розрядність мантиси, а $d$ — розрядність порядку.