# 4 Програмування засобів КІСУ

## 4.1 **Види програмного забезпечення**

Програмне забезпечення розділяється на системне і прикладне. **Системне програмне забезпечення** призначене для керування роботою складових обчислювального пристрою (комп'ютера, контролера, автоматизованого робочого місця) та обміном даними між цими складовими, діагностування та усунення недоліків у роботі пристрою, автоматизації процесу обробки даних, організації обміну даними між користувачем і технічними засобами. Сюди відносяться:

- Операційні системи
- Драйвери
- Утиліти, тобто програми для виконання службових функцій
- Антивірусні програми
- Програми-архіватори

**Операційна система (ОС) –** організована сукупність програмних засобів, призначена для керування ресурсами обчислювальної системи. Операційні системи діють як інтерфейс між апаратурою і користувачами і займають найважливіше місце в сукупності сучасних системних програмних засобів. Вони є основою організації обчислювального процесу і визначають ефективність як використання апаратних компонентів системи, так і вирішення поставлених задач.

**Прикладне програмне забезпечення (ППЗ)**— це програми, що призначені для реалізації конкретних задач опрацювання даних. Їх поділяють на прикладні програми загального і спеціального призначення.

До прикладних програм **загального призначення** відносять програми, які можуть застосовуватися в різних галузях людської діяльності для опрацювання текстів, зображень, баз даних, програми статистичної обробки інформації, експертні системи підтримки прийняття управлінських рішень і т.п..

Прикладні програми **спеціального призначення** використовуються для реалізації завдань опрацювання даних у певній галузі діяльності, на конкретному підприємстві, в тій чи іншій організації.

У КІСУ до ППЗ спеціального призначення відносяться:

- ППЗ контролерів: непроцедурні технологічні мови, що дозволяють легко реалізовувати логічні операції; конфігуратор і бібліотека програмних модулів (модулі математичних функцій, первинної обробки інформації, регулювання). Особливостями ППЗ контролерів є: простота використання технологічних мов; наявність у бібліотеці модулів сучасних досконалих алгоритмів (алгоритми самонастроювання регуляторів, адаптивного керування, нечіткого регулятора й ін.). Деякі контролери можуть виконувати програми, написані мовами високого рівня;
- ППЗ пультів і АРМ операторів.

Розробка прикладного програмного забезпечення пультів оператора може здійснюватися двома шляхами: з використанням традиційних мов програмування (C++, Java, Python та ін.) або з використанням існуючих готових інструментальних проблемно-орієнтованих засобів.
Процес створення ППЗ з нуля з використанням традиційних мов програмування для КІСУ є неприпустимо тривалим і складним.

## 4.2 Операційні системи реального часу

На теперішній час найбільш поширеними ОС є Microsoft Windows, UNIX, MacOS. Під їх керуванням працюють більшість серверів і робочих станцій. Але в керуючих обчислювальних пристроях на перший план виходять вимоги роботи в режимі реального часу.

**Системою реального часу** називається будь-яка система, в якій суттєву роль грає час генерації вихідного сигналу. Часова затримка від одержання вхідного сигналу до видачі вихідного повинна бути малою, щоб забезпечити прийнятний час реакції на незаплановані події.

**ОС РЧ** (англ. Real-Time Operating System, RTOS) – це система, яка обслуговує зовнішні процеси, які розвиваються у пристроях, що мають суворі обмеження на час відповіді. Діями ОС РЧ керують переривання від зовнішніх процесів; якщо вони не будуть вчасно оброблені, то хід зовнішнього процесу може спотворитися.
Ці системи часто проектуються для окремого застосування, наприклад, для керування конкретним технологічним процесом. Проте поняття «система реального часу» не треба ототожнювати з поняттям «швидка система». Час затримки реакції СРЧ на подію буває не так і важливий (він може досягати декількох секунд). Головне, щоб цього часу було досить для конкретної ситуації.

Операційна система, яка може забезпечити необхідний час виконання завдання реального часу навіть в найгірших випадках, називається операційною системою **жорсткого реального часу**. Система, яка може забезпечити необхідний час виконання завдання реального часу в середньому, називається операційною системою **м'якого реального часу**.

Порівняння ОСРЧ і звичайних операційних систем:

| | ОС реального часу | ОС загального призначення |
|---|---|---|
| **Основна задача** | Встигнути зреагувати на події, що відбуваються на устаткуванні | Оптимально розподілити ресурси комп'ютера між користувачами та задачами |
| **На що орієнтована** | Обробка зовнішніх подій | Обробка дій користувача |
| **Як позиціонується** | Інструмент для створення конкретного апаратно-програмного комплексу реального часу | Сприймається користувачем як набір додатків, готових до використання |
| **Кому призначена** | Кваліфікований розробник | Користувач середньої кваліфікації |

Загальні вимоги, що пред’являються до систем реального часу:

- своєчасна реакція – після того як відбулася подія, реакція має бути не пізніше, ніж через необхідний час, перевищення цього часу розглядається як серйозна помилка;
- одночасна обробка інформації, яка характеризує зміну процесу декількох подій. Це означає, що система реального часу повинна мати вбудований паралелізм. Паралелізм досягається використанням декількох процесорів в системі і/або багатозадачним підходом.

Основу втілення системи реального часу складає ОС РЧ чи спеціалізовані керуючі програми, об'єднані в спеціалізовану ОС. Використання універсальних ОС РЧ у повному обсязі як на основі багатопроцесорної системи, так і у формі автоматизованого робочого місця з контролерами, є недоцільним. Причиною є ієрархічна надмірність універсальних ОС. Тому використовуються спеціалізовані ОС РЧ, розраховані на вирішення вузького кола задач.

ОСРЧ містить ядро і лінійку додаткових компонентів.

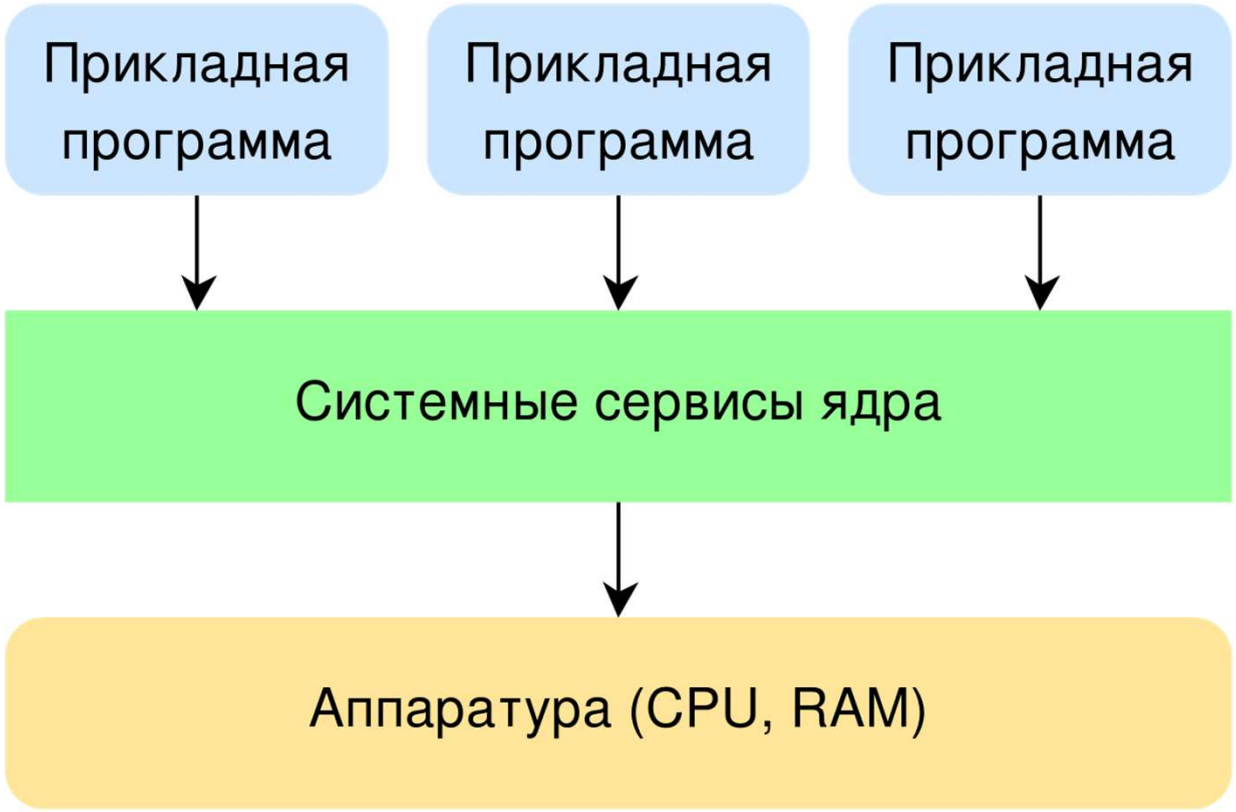
- Ядро ОСРЧ забезпечує функціонування проміжного абстрактного рівня ОС, який приховує від прикладного ПО специфіку технічного пристрою процесора (декількох процесорів) і пов'язаного з ним апаратного забезпечення.

- додаткових компонентів для організації таких високорівневих понять, як файлова система, мережева взаємодія, управління мережею, управління базою даних, графічний, призначений для користувача, інтерфейс і т. д.

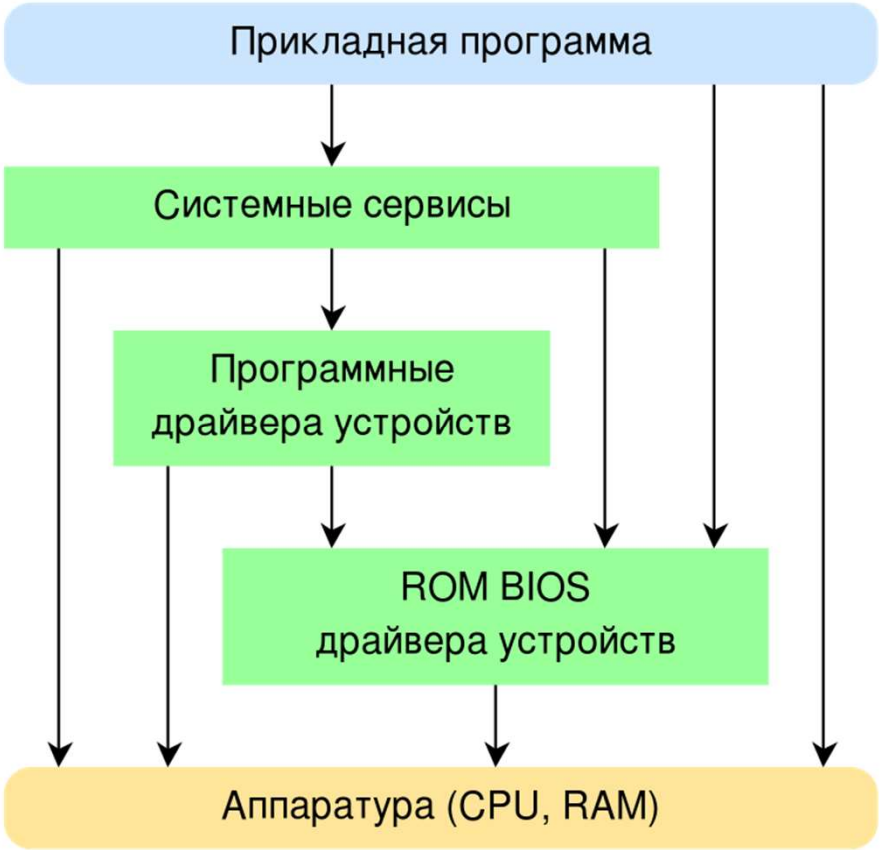
Кожен з таких компонентів включається через вбудовану систему тільки тоді, якщо вони необхідні для виконання вбудованої програми і тільки для того, щоб звести витрати пам'яті до мінімуму.

# Архітектури ОСРЧ

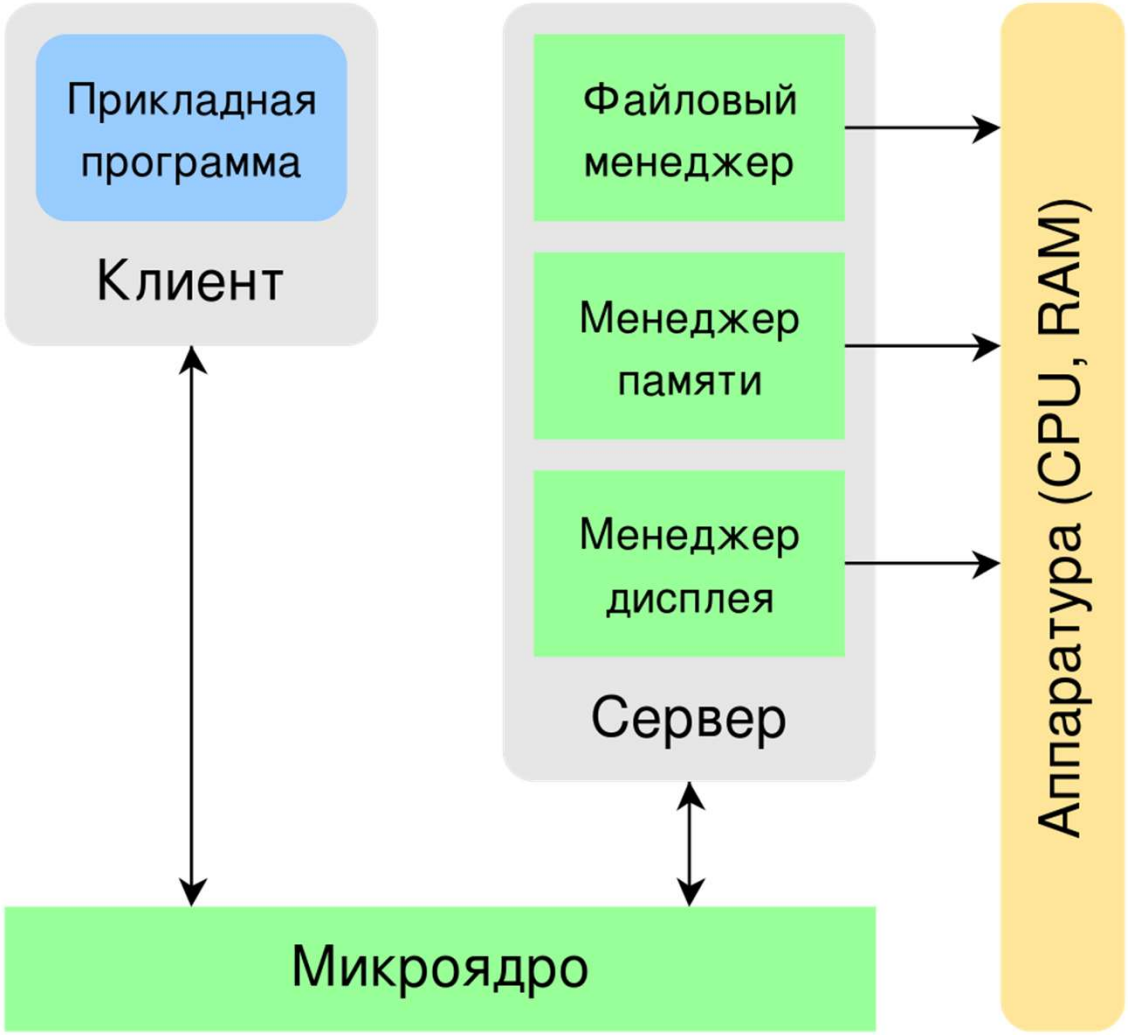
- **Монолітна архітектура**. ОС визначається як набір модулів, взаємодіючих між собою всередині ядра системи і надають прикладному ПО вхідні інтерфейси для звернень до апаратури. Основний недолік цього принципу побудови ОС полягає в поганій передбаченості її поведінки, викликаної складною взаємодією модулів між собою.



- **Рівнева (шарова) архітектура**. Прикладне ПО має можливість отримати доступ до апаратури не тільки через ядро системи та її сервіси, а й безпосередньо. У порівнянні з монолітною така архітектура забезпечує значно більший ступінь передбачуваності реакцій системи, а також дозволяє здійснювати швидкий доступ прикладних програм до апаратури.



- **Архітектура «клієнт-сервер»**. Основний її принцип полягає у винесенні сервісів ОС у вигляді серверів на рівень користувача та виконанні мікроядром функцій диспетчера повідомлень між клієнтськими програмами користувача і серверами — системними сервісами.



Переваги архітектури «клієнт-сервер»:

- підвищена надійність, через те, що кожен сервіс є, по суті, самостійним процесом і його легше налагодити і відстежити помилки;
- покращена масштабованість, оскільки непотрібні сервіси можуть бути виключені з системи без шкоди до її працездатності;
- підвищена відмовостійкість, оскільки сервіс, який "завис ", може бути перезавантажений без перезавантаження системи.

Особливі вимоги до ОС РВ *керуючих систем*:

а) оперативна взаємодія (обмін) із блоком керування на базі зворотного зв'язку за результатами керування;
б) підтримка стратегії керування з мінімально можливим простоєм основного устаткування системи керування;
в) можливість реконфігурації структури і перерозподіл задач між процесорами.

У режимі оперативної роботи повинні розв’язуватись наступні спеціальні задачі:

а) видача керуючих сигналів на контролери чутливих елементів і виконавчих пристроїв;

б) приймання запитів на обслуговування віддалених користувачів;

в) взаємодія з іншими комп'ютерами мережі, що забезпечують технологічну підготовку й інформування про стан керованого технологічного процесу;

г) тестування стану обчислювального комплексу і пристроїв системи керування;

д) реініціалізація (тобто перезапуск) ОС при виявленні особливих ситуацій.

На сьогодні найбільш широко розповсюдженими ОС РЧ є QNX Neutrino, RTOS, RTEMS, ChorusOS, RTX для Windows NT, INtime, TinyOS, OSEK / VDX, Contiki, pSOS, INTEGRITY, LynxOS, Microware OS-9, GRACE-OS, C EXECUTIVE, CMX-RTX, Inferno.

Розглянемо деякі промислові ОСРЧ.

**VxWorks AE 1.1**

Операційна система VxWorks побудована на принципах монолітної операційної системи. Вона підтримує так звану пріоритетну витісняючу багатозадачність в комбінації з карусельною багатозадачністю. Має технологію захищених доменів (protected domains), які є деякими контейнерами, зі своїм адресним простором і, залежно від налаштування, видимі або не видимі один для одного. Поява захищених доменів дозволила здійснити вищу захищеність даних і коду прикладних застосувань. Іншою позитивною рисою захищених доменів, в порівнянні з класичними процесами, являється можливість установки діапазону пріоритетів, які наслідуватимуть потоками цього домена. Таким чином, компоненти системи, що не вимагають реального часу, можуть бути легко перенесені в область пріоритетів, де вони не зможуть викликати конфліктів з потоками реального часу.

QNX 6.21

Ця система будується на базі мікроядра з організованими за технологією клієнт-сервер сервісами, винесеними на рівень призначених для користувача додатків. Мікроядро системи виступає диспетчером повідомлень, переадресовуючи системні виклики прикладних програм клієнтів до відповідних сервісів серверам і назад.

У системі QNX є ієрархія привілеїв виконання програм. Мікроядро виконується на рівні привілеїв 0 процесора, системні сервіси (менеджери) запускаються на рівні привілеїв 1, драйвери пристроїв - 2 і призначені для користувача застосування на 3 рівні привілеїв. Подібний розподіл призводить до вищої надійності і відмовостійкості системи, оскільки при «зависанні» окремих драйверів або сервісів, вони можуть бути перезапущені без перезавантаження системи. Також в ОС QNX реалізована модель віртуальної пам'яті для кожного процесу, що забезпечує високу міру захищеності даних і коду прикладних програм і системи.

# 4.3 Програмування ПЛК

Однією з важливих особливостей сучасного програмного забезпечення ПЛК є те, що можна використовувати різні технологічні мови програмування. Стандартом МЕК1131-3 визнані 5 мов програмування:

а) мова крокових діаграм LD;
б) мова функціональних блокових діаграм FBD;
в) мова послідовних функціональних схем SFC;
г) мова структурованого тексту ST;
д) мова інструкцій IL.

Кожна з них має свої особливості і пристосована для розв’язання тих чи інших задач.

# Мова крокових діаграм LD

Інші назви: мова релейно-контактної логіки, мова східчастої логіки. Англійською – Ladder diagram.

Синтаксис мови є зручним для заміни логічних схем, виконаних на релейній техніці й розрахований на знайомих з нею інженерів з автоматизації. Подає логічні операції як електричні кола із замкнутими та розімкненими контактами.

Основними елементами мови є контакти. Пара контактів ототожнюється з логічною змінною, а стан цієї пари — із значенням цієї змінної.

*Вхідні ланки (контакти)*:

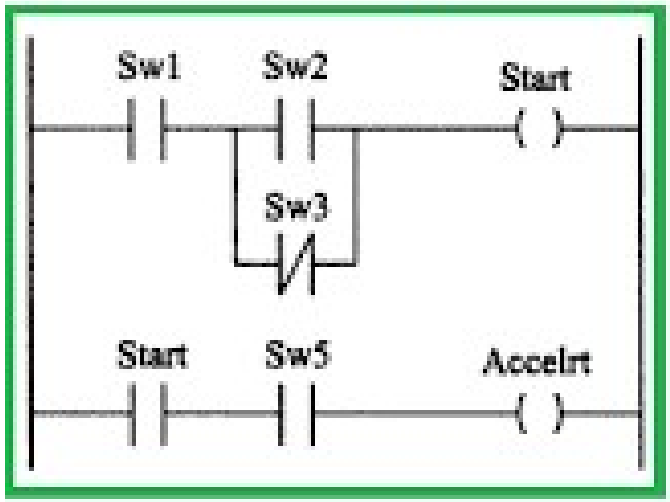
—[ ]— — нормально розімкнутий контакт, замикається при значенні *істина*;

—[\]— — нормально замкнутий контакт, розмикається при значенні *істина*.

*Вихідна ланка (котушка)* — результат логічного ланцюжка, що копіюється у цільову змінну, яку називають *котушка*.

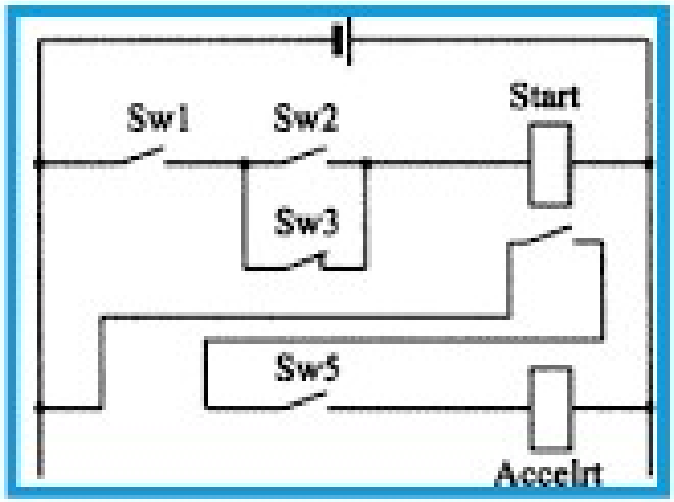
—( )— — котушка  реле

Ladder diagram




=

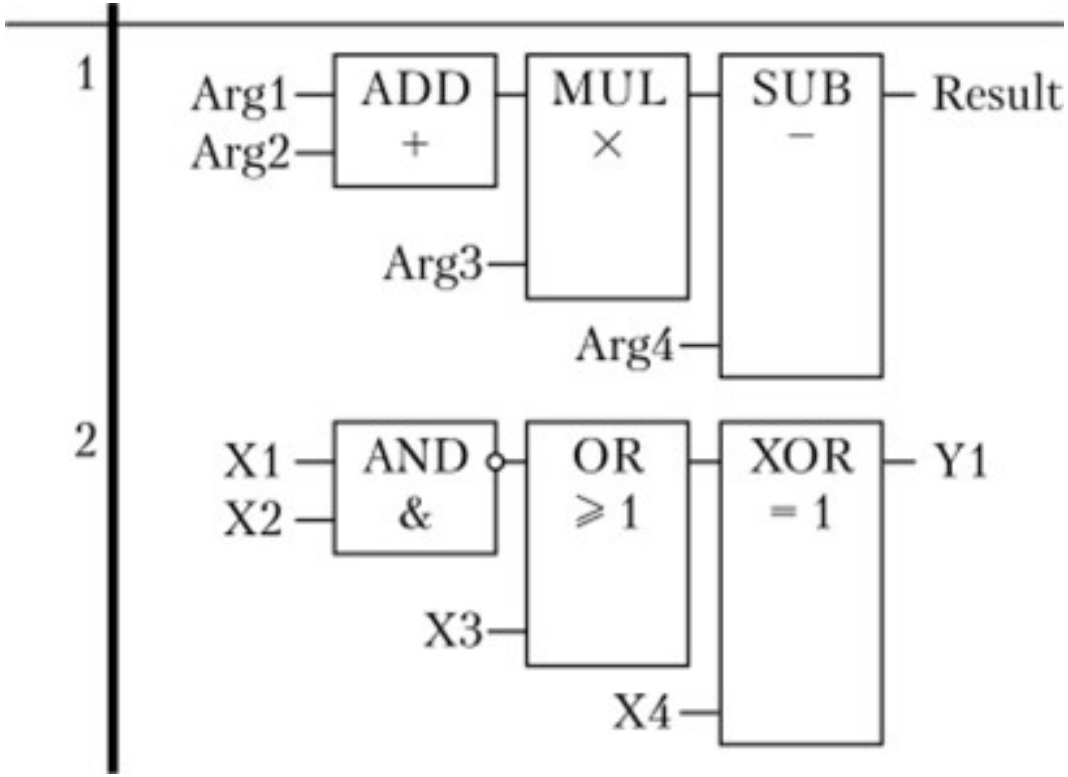
Релейна схема



# Мова функціональних блокових діаграм FBD

Англійською – Functional Block Diagram.
FBD-мова дозволяє створити програму практично будь-якої складності на основі стандартних блоків (арифметичні, тригонометричні та логічні блоки, ПІДрегулятори і т.д.), які сполучаються між собою за допомогою входів і виходів
FBD-програма (діаграма) – це блок-схема з певної кількості елементарних функцій та функціональних блоків, які зображаються прямокутниками.
Функції – це комбінаційні схеми. Вони не мають внутрішніх умов, тому при кожному їх виконанні одному і тому ж значенню вхідних величин відповідають ті ж самі значення вихідної величини. Функція має декілька входів і тільки один вихід.
Входом FBD- блока може бути константа, будь-яка внутрішня вхідна або вихідна змінна. Виходом – будь-яка внутрішня або вихідна змінна, або ім'я функції (тільки для функцій).

На відміну від схем LD, де з'єднання між елементами «передають» напругу живлення, тут з'єднання можуть передавати будь-які види інформації.




Рядки програми виконуються послідовно зверху вниз, елементи в рядку виконуються також послідовно зліва направо.
Так само, як і мова LD, мова FBD не передбачає будь-якої організації програми, крім виконання переходів.

# Мова послідовних функціональних схем SFC

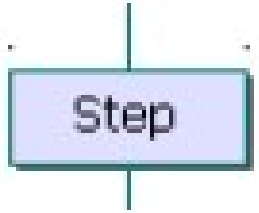
Англійською **–** Sequential Function Chart.

Мова SFC використовується для опису алгоритму у вигляді набору зв’язаних пар – **крок** (step) і **перехід** (transition). Крок є набором операцій над змінними. Перехід – набір логічних умовних виразів, визначаючих передачу управління до наступної пари крок-перехід. На вигляд опис SFC-мовою нагадує добре відомі логічні блок-схеми алгоритмів.

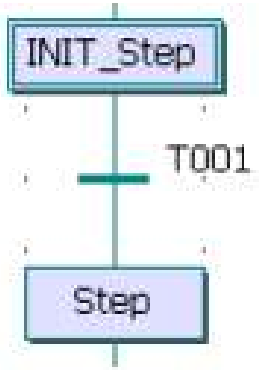
Строго кажучи, SFC-мова не є мовою програмування. Це засіб проектування прикладного програмного забезпечення, що складається з комплексу великої кількості одиниць: програм, функціональних блоків, функцій.

Основними компонентами SFC-мови є кроки, переходи, орієнтовані сполучення, стрибок на крок, сходження і розбіжності.

**Крок** зображається прямокутником. Кожному кроку привласнюється ім'я, написане усередині прямокутника. Початковий крок позначається графічним символом з подвійною рамкою. Кроки не можуть слідувати один за одним. Обов’язково поміж ними має бути перехід.




**Переходи** зображаються маленькими горизонтальними смужками, які перетинають лінії сполучення. Кожному переходу присвоюється ім’я, яке записується поряд.



**Дії** у SFC-програмі зображуються у вигляді прямокутників, розташованих праворуч від символу кроку і приєднаних до них лінією. У лівій частині прямокутника знаходиться класифікатор, можливо, з константою часу, а права містить ім’я дії або логічної змінної. Кроки можуть мати декілька дій.

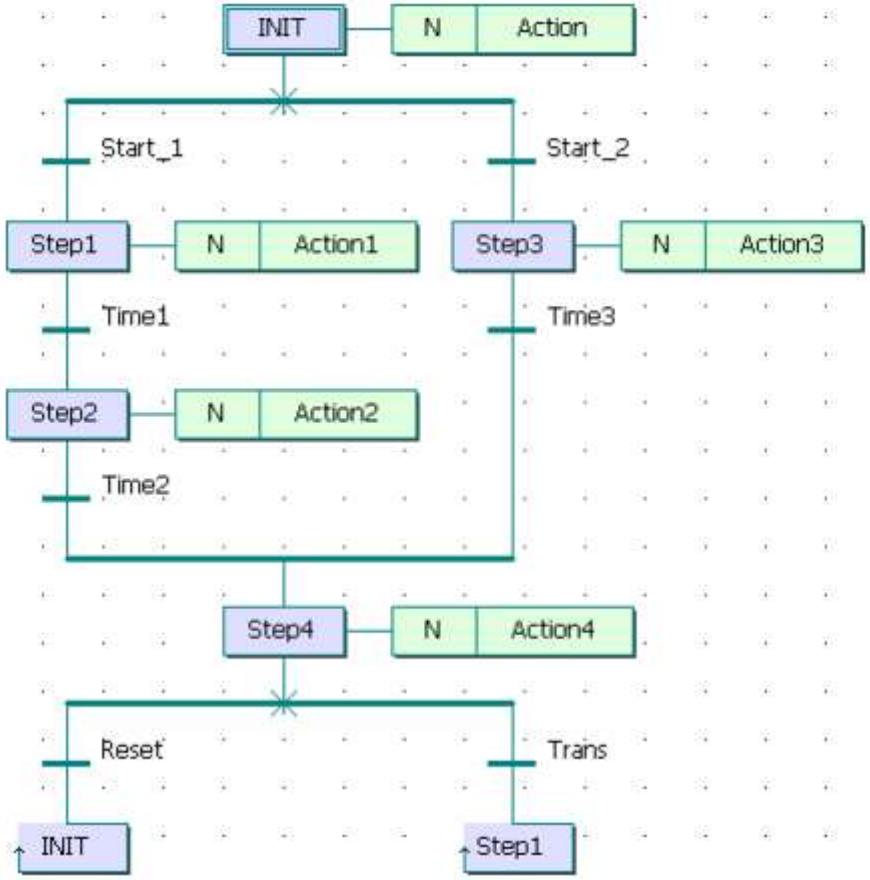

Основні типи дій:

- булеві привласнюють вихідним або внутрішнім логічним змінним значення TRUE або FALSE;
- імпульсні дії, які виконуються тільки одного разу при активізації кроку;
- SFC дії (управління дочірніми SFC програмами);
- виклик функцій, функціональних блоків і підпрограм, створених будь-якою мовою, окрім SFC.

**Розбіжності** - це множинні сполучення від одного символу SFC (Кроку або Переходу) до багатьох інших символів.

**Сходження** - це множинні сполучення від більш ніж одного символу SFC до одного іншого символу. Сходження і розбіжності можуть бути одиночними або подвійними.

Приклад програми:



# Мова структурованого тексту ST

Англійською Structured Text.

Це мова, яка відноситься до класу мов високого рівня, схожа на Паскаль. У цій мові підтримуються масиви (також і багатовимірні), контроль перетворення типів, є такі конструкції, як DO-WHILE, REPEAT UNTIL, FOR-TO-DO, IF-THEN-ELSE, CASE-OF, а також інші зрозумілі будь-якому програмісту оператори. Зазвичай мова ST використовується тільки для програмування послідовних кроків і переходів SFC-мови.

Основою ST-програми є вирази. Вираз складається із операндів (змінних, констант, функцій і функціональних блоків), розділених операторами. Результат обчислення виразу, як і у Паскалі, привласнюється змінній за допомогою оператора «:=». Кожний вираз обов'язково закінчується крапкою з комою «;»

# Мова інструкцій IL

Англійською Structured Text.

Список інструкцій IL (Instruction List) або командний список є текстовою мовою низького рівня. Це найпростіша мова мнемонічних інструкцій, яка нагадує мову Асемблера і призначена для програмування контролерів малої потужності. Програми, що створені IL-мовою, легко транслюються в машинні коди будь-якого процесора і це дозволяє створювати дуже швидкі програми. Проте, реально IL-мова не має ніяких переваг перед ST або FBD-мовами, тому самостійного значення не має і використовується тільки спільно з SFCмовою.

Синтаксис інструкції має вигляд: < Мітка:> < оператор> < модифікатор>< операнд> (* коментар*)

Інструкції завжди відносяться до поточного результату, який знаходиться в IL-регістрі (в асемблері це зветься аккумулятор).
**Мітка** не є обов’язковою, використовується при необхідності переводу виконання програми на вказану інструкцію за допомогою операції стрибка, а в якості операнда вказати мітку.
**Оператор** визначає операцію, яка має бути виконана з поточним результатом в IL-регістрі  і операндом. Результат операції знову запам'ятовується в IL- регістрі.
**Модифікатори** 'C', 'N' і '(' модифікує значення інструкції. Символ модифікатора завершує ім'я оператора без пропуску, наприклад: JMPC, ANDN, CALCN.
Модифікатор 'N' (negation) інвертує значення операнда до виконання інструкції. Модифікатор 'C' (condition) вказує на те, що відповідна інструкція має бути виконана тільки за умови, що поточний результат має значення TRUE. Модифікатор відкриваюча дужка '(' вказує на те, що оцінка інструкції має бути затримана до тих пір, поки не зустрінеться оператор закриваюча дужка ')', який виконує затриману операцію.

**Операнд** і поточний вміст акумулятора (IL-регістра) повинні мати однаковий тип даних. Якщо є потреба опрацювати операнди різних типів даних, то спочатку здійснюється перетворення типів.

При IL-програмуванні можна використовувати функції та функціональні блоки. Виклик функції в IL (написаної на будь-якій з мов IL, ST, LD, FBD або 'C') використовує її ім'я як оператора. При цьому перший параметр виклику має бути збережений в IL-регістрі перед викликом. Решта параметрів записується в полі операнда, розділяючись комами. Після виконання функції, значення що нею повертається, запам'ятовується у поточному результаті IL-регістра.

Список команд має завжди починатися з оператора LD (команда завантаження IL- регістра) і закінчуватися оператором збереження ST у деякій змінній. Приклад:

```
LD 20
SUB 8
ST M
```

Раніше основним технічним засобом для програмування ПрК були спеціалізовані пульти програмування, Вони створювались як у вигляді вбудованих в ПрК блоків, так і у вигляді окремих технічних засобів. Ці пульти використовувались також при відлагоджуванні програм, виконанні процедур діагностики і тестування ПЛК.

Тепер для програмування контролерів використовується спеціальне програмне забезпечення персональних комп'ютерів, основним завданням якого є створення максимальних зручностей для програмування і відлагодження програм. Процедури програмування замінюються на більш прості процедури конфігурування, що дає можливість користувачу, не входячи в тонкощі програмування, забезпечувати виконувати досить складних процедур.

Програмні термінали дають можливість використовувати для різних за алгоритмами задач різні технологічні мови. А вже при записі програми у ПЛК різні фрагменти програми поєднуються в один машинний код.

# 4.4 Програмні засоби верхнього рівня КІСУ

## Концепція інтеграції програмного забезпечення

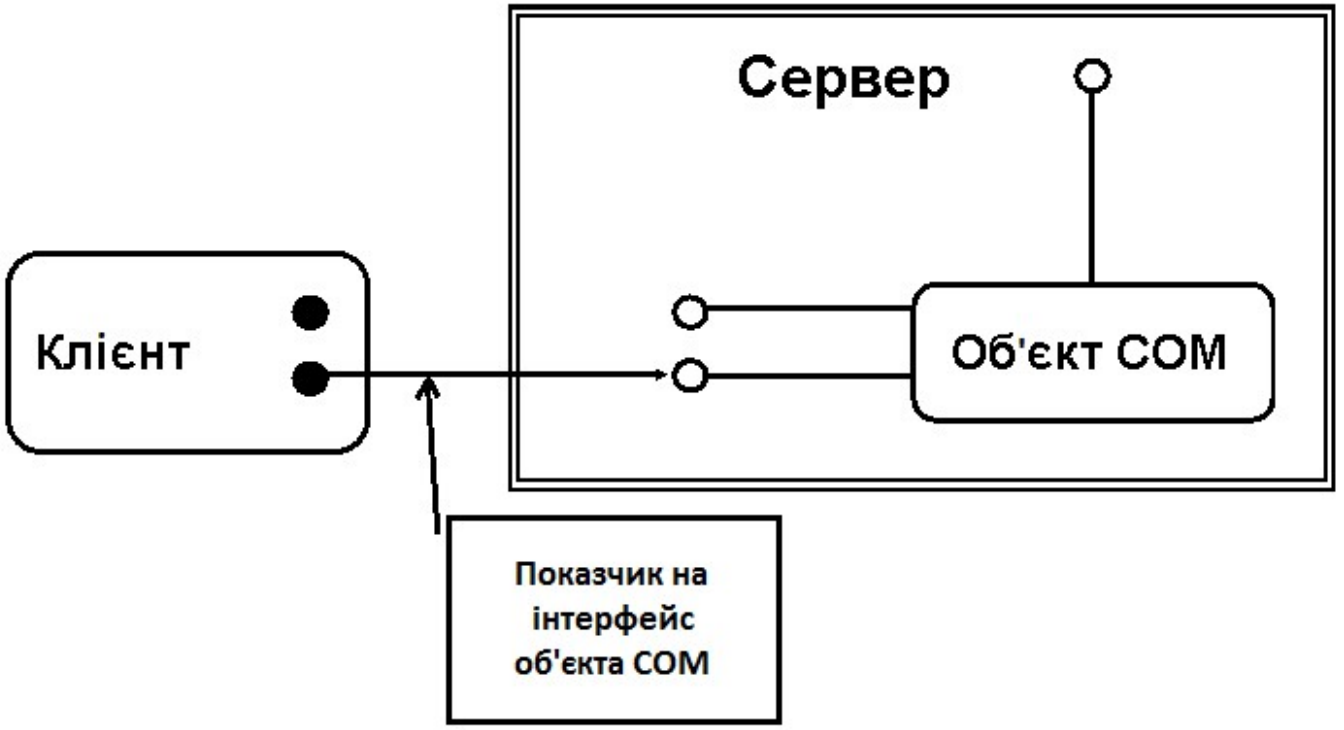
Інтеграція передбачає взаємодію всіх рівнів програмного забезпечення. Різні програмні системи, створені за допомогою різних засобів, встановлених на різних платформах, які працюють на різних комп'ютерах, знають, як запитати один в одного дані і як послати один одному "вказівки".

У компанії Microsoft є глобальна інтеграційна тенденція, що реалізувалася у створенні COM-технології. COM – Component Object Model (модель складових об'єктів) – і її мережеве розширення DCOM – Distributed COM (розподілена COM) – це технологія, введена спочатку Microsoft для інтеграції різних офісних додатків. Модель COM оперує **об'єктами**, дуже схожими на об'єкти в об'єктно-орієнтованих мовах програмування.

Але сама технологія COM не є мовою програмування. Вона тільки регламентує поведінку своїх об'єктів. Об'єкт після створення надає свою функціональність процесу, який його викликав, а після використання – знищується.

Об'єкти COM передають свою функціональність через інтерфейси. Інтерфейс в COM (не плутати з тим, що зазвичай розуміється під словом інтерфейс) об'єднує групу взаємопов'язаних функцій, що надаються об'єктом. Саме інтерфейс, вірніше, покажчик на нього є тим, з чим працює процес, що викликає даний об'єкт. Об'єкт COM – сторона пасивна. Він лише передає через інтерфейси свої функції. Об'єкт завжди працює у складі **COM-сервера**. Сервер може бути динамічною бібліотекою або файлом, що виконується. Об'єкт може мати власні властивості та методи або використовувати дані та служби сервера

Програма, що робить запит до сервера, відповідно, називається COM-клієнт. Але це не виключає того, що обидві програми одночасно можуть бути і COM-серверами, і COM-клієнтами.




Щоб створити об'єкт, потрібно знати, де він знаходиться. Інформація про всі доступні в даній ОС класи COM зібрана у спеціальній бібліотеці COM. У Windows для цього використовується реєстрація об'єктів в системному реєстрі. У реєстрі реєструються також доступні інтерфейси.

При цьому кожен COM-предмет реєстрації має унікальний ідентифікатор, званий GUID (Globally Unique Identifier – глобальний унікальний ідентифікатор). Реєстрація робить доступною інформацію про розташування об'єктів всіх програм, що є на обчислювальному пристрої.

Для мережевих рішень існує DCOM – розширення COM, що дозволяє добиратися до об'єктів на інших комп'ютерах. Важливим є те, що з точки зору програмування нічого не змінюється: DCOM – це системний сервіс, який робить COM прозорим в локальних мережах. При цьому від клієнта приховано, де саме розташований об'єкт:

- в адресному просторі того самого процесу,
- в іншому процесі або
- на іншому комп'ютері.