Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

О.В. Зубков

ПРОГРАМУВАННЯ ПРОМИСЛОВИХ КОНТРОЛЕРІВ SIEMENS
в прикладах і задачах

Рекомендовано
як навчальний посібник для студентів

Харків
“ ”
2011

УДК 519.689.4:658.51.011.56(075.8)

Рецензенти

докт. техн. наук, професор Хаханов В.І., декан факультету КІУ Харківського національного університету радіоелектроніки, керівник науково-методичної секції №2 Харківського національного університету радіоелектроніки

докт. техн. наук, професор Руденко О.Г., завідувач кафедри «Електронні обчислювальні машини» Харківського національного університету радіоелектроніки

Зубков О.В.

Програмування промислових контролерів Siemens в прикладах і задачах: Навч. посіб. – Харків:, 2011. – 122 с. ISBN

Стисло викладено технічні характеристики процесорів та модулів вводу-виводу Siemens. Детально розглянуті синтаксис та команди мов програмування LAD та STL. В кожному розділі наведені приклади програм керування та виконання базових функцій обробки сигналів, які супроводжуються коментаріями. З метою закріплення знань, набутих на практичних заняттях, подаються умови задач для самостійної роботи, які можуть також використовуватись як елементи контрольних завдань для студентів заочної форми навчання.

Значна увага приділяється вивченню розділів обробки цифрових сигналів, роботі з таймерами та лічильниками, непрямій адресації, як найбільш важливим з точки зору практичного використання.

Для студентів усіх форм навчання освітнього напряму “Радіотехніка”, спеціальностей “Радіотехніка”, “Апаратура радіозв’язку, радіомовлення і телебачення” і може бути корисним студентам інших напрямів в області електрозв’язку та технічної кібернетики.

Іл. 79 Бібліогр. назв. 2

УДК 519.689.4:658.51.011.56(075.8)
О.В. Зубков
. , 2011

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І УМОВНИХ ПОЗНАЧЕНЬ

AI – аналоговий вхід;
AO – аналоговий вихід;
BCD – двійково-десятковий формат даних;
CC0, CC1 – біти з регістру статусу, що характеризують результат математичної операції (нульовий, позитивний і т.і.);
CPU – центральний процесорний пристрій;
DB – блок даних;
DI – цифровий вхід;
DO – цифровий вихід;
FBD – функціональний блок (користувальницька підпрограма);
FC – функцій (користувальницька підпрограма);
IM – інтерфейс ний модуль;
LAD – мова релейно-контактних символів для програмування промислових контролерів;
OB – організаційний блок, у якому розташовують одну з головних програм;
OV – біт переповнення результату математичної операції, що знаходиться в регістру статусу;
PS – блок живлення;
RLO – біт результату логічної операції у регістрі статусу;
SCADA – пакет програм для збору, обробки, відображення та архівації інформації об об’єктах керування;
SFC – системна функція, що надається разом з програмним забезпеченням Step7;
STA – результат арифметичних операцій;
STL – мова інструкцій для програмування промислових контролерів Siemens;
MES – пакет програм аналізу, синхронізації, координації та оптимізації випуска продукції на підприємстві;
ERP – пакет програм керування внутрішніми та зовнішніми ресурсами підприємства.

АСКТП – автоматизована система керування технологічними процесами;
ПЕОМ – персональна електронно-обчислювальна машина.

ВСТУП

Сучасна промисловість – це об’єднання найновітніших технологій, високотехнологічного обладнання та автоматизованих систем керування технологічними процесами (АСКТП). Використання АСКТП дозволяє виготовляти продукцію відповідно до стандартів якості, зменшити долю браку, зберігати енергетичні ресурси та знизити шкідливі викиди виробництва. Будь-яке сучасне промислове підприємство використовує АСКТП, які розроблені на обладнанні відомих у світі фірм-виробників, таких як Siemens, Schnider, GE Fanuc, Alan Bradley та ін. Відповідно до вимог світових стандартів таке обладнання повинно відповідати промисловим стандартам, тобто працювати в широкому діапазоні температур, вологості, при вібраціях, в умовах агресивних середовищ. Фірма-виробник повинна зберігати на своїх складах запасне обладнання ще багато років після закінчення його виробництва, щоб підприємства мали можливість відремонтувати свої автоматизовані системи. Фірма Siemens вже багато років знаходиться на ринку АСКТП. Їй належить понад 40% європейського ринку та велика повага з боку підприємств за якісну продукцію. Siemens виробляє промислові контролери керування, модулі вводу-виводу інформації, панелі оператора, промислові комп’ютери, інвертори для керування двигунами, датчики, реле та багато іншого, тобто практично увесь спектр сучасного обладнання АСКТП. Загальна структура автоматизованого моніторингу та керування сучасним підприємством наведена на рисунку 1

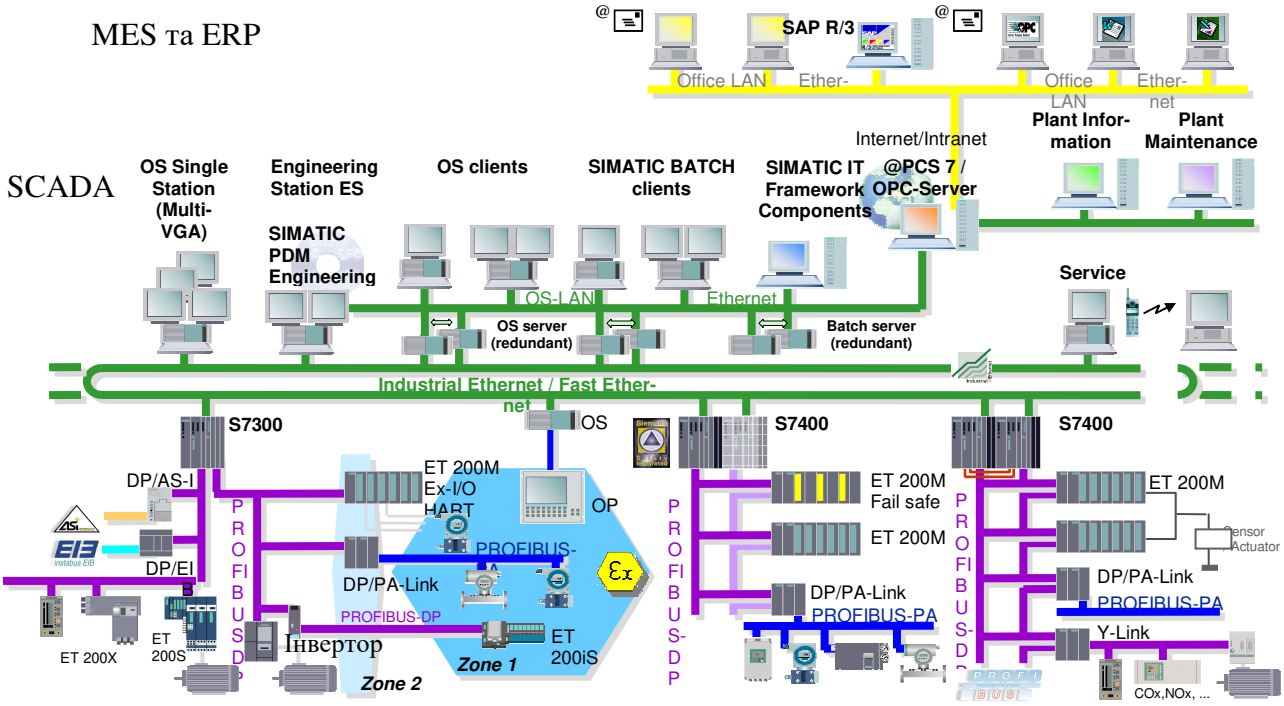


Рисунок 1 – Структура сучасної автоматизованої системи

В наведеній на рисунку 1 системі умовно можна виділити декілька рівнів: нижній рівень технологічного обладнання та керуючих контролерів, рівень

SCADA системи та рівні керування роботою підприємства в цілому, прогнозування роботи, оптимізації виробництва (MES та ERP).

На самому нижньому рівні знаходиться технологічне обладнання: електродвигуни, засувки, клапани, датчики і т.і. Це обладнання розташоване безпосередньо у цехах заводу, на трубопроводах в технологічних зонах. Для керування кожним з таких пристроїв потрібен спеціалізований пристрій керування. Наприклад, для керування частотою та напрямком обертання асинхронного електродвигуна потрібен інвертор, що забезпечує плавний пуск та зупинку двигуна, керує обертами двигуна. Така продукція також випускається фірмою Siemens. Але кожен з таких пристроїв керування нічого самостійно не робить, чекаючи команд від головного контролера керування. Головними контролерами у наведеній на рисунку 1 схемі є контролери фірми Siemens серій S7-1200, S7-300, S7-400. Щоб зібрати вхідну інформацію від датчиків та контактів потрібні модулі вводу інформації, а щоб видати сигнали керування на виконавчий пристрій – потрібні модулі виводу. Щоб не прокладувати багато дуже довгих кабелів від датчиків до модулів вводу, що опитують їх, та навпаки кабелі керування виконавчими пристроями від модулів виводу до самих пристроїв, використовують комунікаційні модулі типу ET200. У цьому випадку контролери та комунікаційні модулі об’єднуються у єдину мережу передавання даних типу PROFIBUS чи Industrial Ethernet, що потребує лише прокладки мереженого кабелю. Обробку всієї інформації виконує контролер, а кабелі під’єднують до модулів вводу і виводу інформації, що підключені до комунікаційних модулів ET200.

Щоб людина могла безпосередньо слідкувати за технологічним процесом у виробничій зоні, вносити корекції в роботу обладнання та самого процесу, використовуються графічні панелі OP. На графічних екранах цих панелей відображуються мнемосхеми технологічного процесу, стан обладнання, предаварійні та аварійні ситуації, показання датчиків та параметри технологічного процесу. Про роботу кожного з пристроїв можна отримати повну оперативну інформацію (струм, напруга, частота обертання, режим роботи і т.і.), якщо це було закладено при розробці системи, а також внести корекції в роботу.

На наступному рівні (SCADA система) знаходяться: сервери, АРМ-серери, сервери баз даних історії стану технологічного процесу, клієнтські станції. Операторам SCADA системи у вигляді графічних мнемосхем відображується стан роботи різних дільниць виробництва усього заводу чи окремих його цехів. Оператори мають можливість дистанційно налаштовувати роботу обладнання та приймати запобіжні дії у разі предаварійних та аварійних ситуацій. Ці функції доступні на серверах, АРМ-серверах та клієнтських станціях, що фізично являють собою промислові комп’ютери з відповідним програмним забезпеченням (WinCC, PCS-7). Крім того на серверах та АРМ-серверах постійно архівуються показання найбільш важливих у технологічному процесі датчиків, на кожну дію оператора чи предаварійний або аварійний стан системи у базу даних автоматично записується текстове повідомлення. Це необхідно, щоб у разі аварії можна було зрозуміти причини її виникнення та відповідальність керуючого персоналу. Також ця інформація дає можливість проаналізувати якість

продукції і своєчасно виконувати профілактичні роботи з обслуговування технологічного обладнання. Для обміну інформацією з MES та ERP системами керування підприємством використовуються фізичні сервери баз даних з відповідним програмним забезпеченням (Oracle, MS SQL Server).

Рівні MES та ERP систем дають можливість керувати підприємством та оптимізувати його роботу. В них об'єднується та аналізується інформація від АСКТП, складів матеріалів та готової продукції, енергоресурсів, що були витрачені на виробництво продукції. Програмна обробка усієї зібраної інформації дає можливість скоротити витрату енергоресурсів завдяки оптимальному плануванню графіка виробництва, оптимізації кількості і розташування обладнання. Також прогнозуються поломки обладнання та приймаються заходи щодо його своєчасного обслуговування та ремонту. На цьому рівні також можуть використовуватися програмні продукти фірми Siemens, такі як Simatic IT, але частіше все ж використовуються спеціалізовані програмні продукти інших виробників.

Модернізація промисловості в Україні потребує молодих спеціалістів з розробки та експлуатації сучасних АСКТП. Основна перевага віддається випускникам вузів з електронно-технічною, радіотехнічною підготовкою, бо вони мають підготовку, як в галузі технічного обладнання, так і його програмування. Розуміючи, що велика частина українського ринку автоматизованих систем належить компанії Siemens, вищі навчальні заклади впроваджують в програму підготовки своїх студентів вивчення обладнання цієї компанії та його програмування. Для програмування контролерів керування фірма Siemens надає необхідне програмне забезпечення, що дозволяє програмувати на 5 міжнародно-стандартизованих мовах LAD, STL, FBD, SCL, SFC. Крім того, на відміну від більшості виробників промислових контролерів, фірма Siemens випускає програмний пакет для розробки програмного забезпечення панелей оператора, а також 2 SCADA системи WinCC та PCS 7 – пакети програм для ПЕОМ, що дозволяють візуалізувати роботу системи АСКТП, архівувати у реальному часі показання датчиків, тривоги та аварійні ситуації, створювати резервовані системи і багато іншого. Наявність серійного обладнання для створювання АСКТП та програмних пакетів для його програмування дуже скорочує час розробки і дозволяє виділити основний час на створення програмного забезпечення.

Відповідно до побажань багатьох харківських підприємств на кафедрі «Радіоелектронні системи» з 2007 року викладається дисципліна «Програмування промислових контролерів Siemens» для студентів денної і заочної форми навчання напрямку 6.050901 "Радіотехніка" спеціальностей 7.05090101 "Радіотехніка", 7.05090102 "Апаратура радіозв'язку, радіомовлення та телебачення". Метою дисципліни є надання студентам початкових знань з функціонування автоматизованих систем на виробництві та умінь програмування промислових контролерів, що входять до цих систем та керують ними.

Основними задачами курсу є:

1) вивчення характеристик, конструкції, принципів функціонування промислових контролерів Siemens, модулів вводу-виводу, промислових мереж передавання даних;

2) оволодіння двома основними мовами програмування процесорів Siemens – LAD (мова релейно-контактної логіки) та STL (асемблер контролерів Siemens), що входять до складу стандартної версії пакету програмування Step7 Standard;
3) оволодіння пакетом симуляції роботи розробленого програмного забезпечення S7-PLCSim, завдяки якому не маючи апаратури можна налагодити розроблене програмне забезпечення майже на 95%;
4) отримання практичних навичок з конфігурування та програмування обладнання лабораторного стенду.

Нажаль на українському та російському ринках технічної літератури немає будь-яких підручників з програмування контролерів Siemens. Підручники з програмування контролерів інших фірм-виробників використовувати не можна, бо кількість і номенклатура команд у кожного виробника контролерів дуже відрізняються, а мова STL використовується тільки для контролерів Siemens.

У даному навчальному посібнику будуть розглянуті 2 мови програмування промислових контролерів Siemens, що входять до стандартного пакету Step7, розглянуті принципи роботи промислових контролерів, наведені приклади програм.

1 Контролери Siemens

1.1 Технічні характеристики контролерів Siemens та модулів вводу-виводу

Контролери Siemens – це сучасні 32-бітні контролерні модулі з вбудованою в них операційною системою, промисловими комунікаційними інтерфейсами, оперативною пам'яттю та пам'яттю програм. Існує 2 серії таких контролерів S7-300 та S7-400 [1-3]. У кожній серії випускається понад 20 модифікацій контролерів. Контролери відрізняються кількістю пам'яті, швидкодією, кількістю інтерфейсів, оброблюємих сигналів. Для зчитування та видачі цифрових та аналогових сигналів використовуються відповідно цифрові та аналогові модулі вводу-виводу інформації, які під'єднуються до центрального контролеру внутрішньою шиною або через спеціальні комунікаційні модулі. Умовне розташування контролеру та модулів вводу-виводу в лінійці апаратури S7-300 наведено на рисунку 1.1

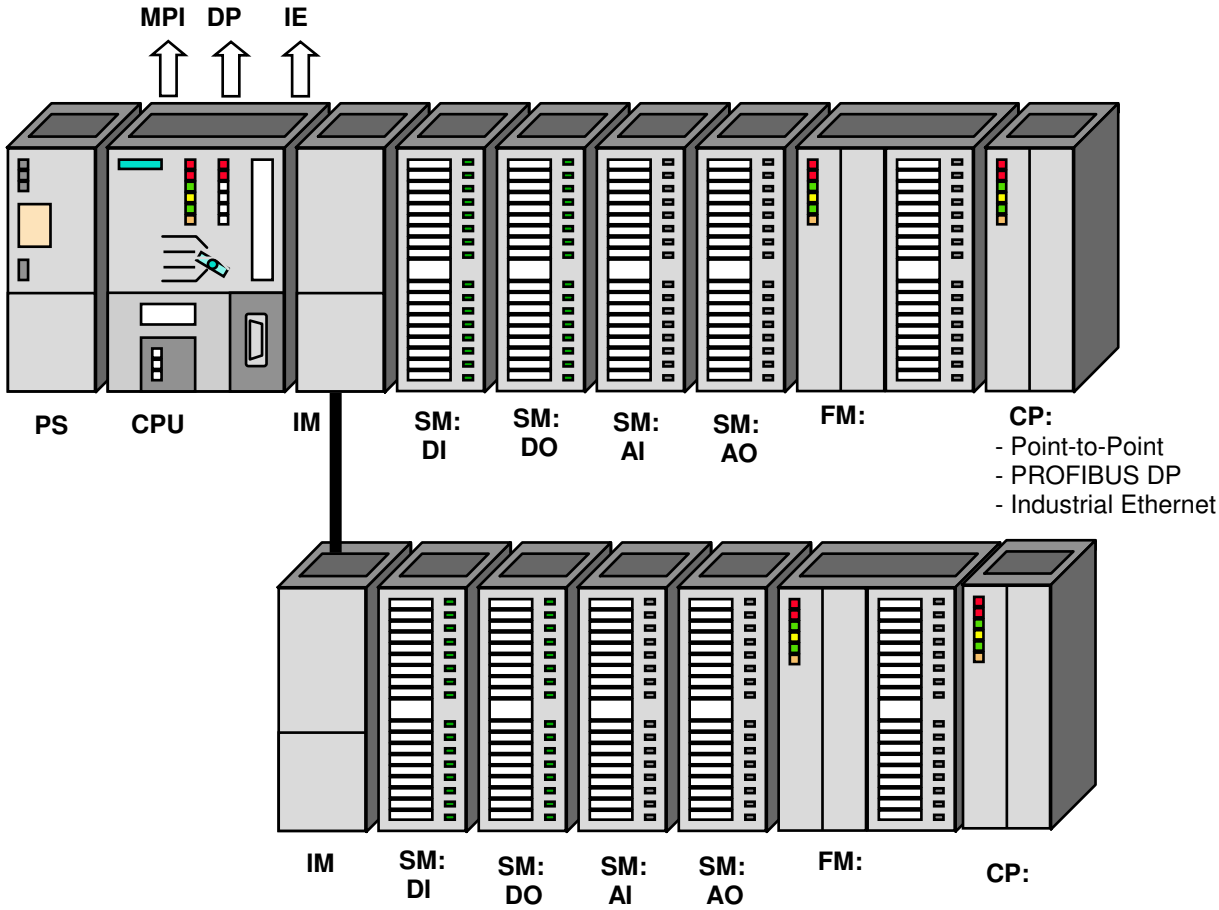


Рисунок 1.1 – Умовне розташування контролеру та модулів вводу-виводу в лінійці апаратури S7-300

Усі модулі апаратури керування розташовують послідовно на DIN рейці. На першому місці знаходиться блок живлення (PS). На другому контролер

(CPU), котрий виконує головні функції зі збору даних від інших модулів, її обробки та керування виконавчими пристроями. З усіх безпосередньо приєднаних до контролера модулей він збирає інформацію через послідовну системну шину, що з'єднує усі модулі на рейці. Кожен з контролерів також має один або декілька інтерфейсів: MPI, Profibus DP, Industrial Ethernet (IE). Основний інтерфейс для програмування контролера MPI, швидкість передавання даних через який до 12 Мбіт/с. Через цей інтерфейс іноді з'єднують контролер з графічними панелями, що відображають технологічний процес у зручному для людини вигляді. Декілька контролерів можуть бути з'єднані у мережу разом з графічними панелями та комп'ютерами з використанням інтерфейсу Profibus DP. Цей поширений промисловий інтерфейс має високу захищеність від завад, швидкість передавання даних до 12Мбіт/с та дозволяє організувати будь-яку конфігурацію мережі із довжиною до 11 км. Інтерфейс Industrial Ethernet на фізичному рівні є традиційним Ethernet зі швидкістю 100Мбіт/с по витій парі або оптичному каналу. На рівні протоколів використовуються спеціалізовані промислові протоколи, як модифікації традиційного протоколу TCP/IP. Інтерфейсний модуль (IM) встановлюється, коли загальна кількість різноманітних модулів включаючи контролер та блок живлення перевищує 10. Тоді модулі розташовують в декількох DIN рейках (рядах) загальною кількістю до чотирьох. Збір та передачу даних від модулів вводу-виводу до контролеру виконують в кожному ряду інтерфейсні модулі IM. Далі в кожному ряду розташовують у довільному порядку цифрові модулі вводу логічних сигналів (DI), виводу логічних сигналів (DO), модулі вводу та перетворення аналогових сигналів в цифрову форму (AI), формування та виводу аналогових сигналів (AO). В кінці розміщують високошвидкісні лічильники (FM), опозиціонери. Коли контролер не має деякого потрібного інтерфейсу чи необхідно з'єднати його з інтелектуальними датчиками використовують комунікаційні модулі (CP). В подальшому розробка програмного забезпечення починається з створення проекту з складом та розміщенням апаратури, що відповідають реальній системі керування.

Серія S7-400 має деякі принципові відзнаки від S7-300. В неї внутрішня шина паралельна, а не послідовна, завдяки чому існує можливість гарячої заміни модуля, що вийшов з ладу зі зберіганням працездатності всієї системи в загалом на відміну від S7-300, де для заміни треба вимкнути контролер, до якого під'єднаний цій модуль. Деякі контролери серії S7-400 завдяки програмно-апаратній підтримці дозволяють створювати резервовані системи, при цьому функції сумісної роботи та автоматичного передавання керування у випадку виходу з ладу одного з комплектів обладнання виконує сама операційна система контролера. Порівняльний аналіз технічних характеристик деяких контролерів серій S7-300 та S7-400 наведено у таблиці 1.1

Таблиці 1.1 – Технічні параметри контролерів Siemens серій S7-300 та S7-400

Параметри	Контролери S7-300		Контролери S7-400	
	CPU 312C	CPU 314C-2 DP	CPU 412-1	CPU 417-4
Час виконання 1000 команд, мкс	0,2-6	0,1-3	0,1-0,3	0,03-0,09
Кількість маркерної пам'яті, байт	128	256	4096	16384
Кількість ОЗУ блоків даних, кбайт	16	64	72	10000
Кількість оброблюємих дискретних входів/виходів	256	8192	32768	131072
Кількість оброблюємих аналогових входів/виходів	64	512	2048	8192
Кількість таймерів	128	256	2048	2048
Кількість лічильників	128	256	2048	2048
Кількість блоків даних, що можна створити	511	511	511	8191
Кількість функцій, що можна створити	512	512	256	6144
Кількість функціональних блоків, що можна створити	512	512	256	6144

Контролери Siemens мають 4 різновиду оперативної пам'яті: оперативна пам'ять, до якої після подачі живлення завантажується з пам'яті програм розроблене програмне забезпечення; швидкісна оперативна пам'ять (кеш контролера), якою може користуватися програміст – меркерна пам'ять; оперативна пам'ять блоків даних, якою користується програміст; оперативна пам'ять, що відповідає адресному простору входів-виходів. При написанні програми можна обробляти лише ті входи та виходи, що підключені до контролера. При цьому їх кількість не може перевищувати допустиму для конкретного контролера.

Для рахування інтервалів часу та імпульсів у кожній реальній програмі використовуються таймери та лічильники, кількість яких теж обмежена у кожного контролера. В операційній системі контролера вони реалізуються як програмні модулі.

Цифрові модулі вводу та обробки логічних сигналів мають загальну назву SM 321. Їх основні технічні характеристики наведені у таблиці 1.2

Таблиця 1.2 – Основні технічні характеристики модулів вводу SM 321

Параметр	Значення

Кількість входів в модулі	4,8,16,32
Максимальна довжина неекранованого кабелю до датчика, м	600
Максимальна довжина екранованого кабелю до датчика, м	1000
Рівень сигналу логічної одиниці, В	13…30
Рівень сигналу логічного нуля, В	-30…5
Потенційна розв'язка входів групами по	8 або 16
Параметризація (програмне налаштування) апаратних переривань (реакції на зміну стану логічного сигналу)	
Параметризація діагностичних переривань (обриву датчиків чи помилок у роботі модуля)	
Параметризація часу реакції на зміну вхідного сигналу для боротьби с завадами, мс	0,1…20

Слід відзначити, що переривання та налаштування часу реакції мають не всі модулі. В загалом цифрові модулі вводу призначені для опитування датчиків з цифровим виходом чи сухих контактів.

Цифрові модулі виводу мають загальну назву SM 322. Їх основні технічні характеристики наведені у таблиці 1.3

Таблиця 1.3 – Основні технічні характеристики модулів виводу SM 322

Параметр	Значення
Кількість виходів в модулі	4,8,16,32
Максимальна довжина неекранованого кабелю до навантаження, м	600
Максимальна довжина екранованого кабелю до навантаження, м	1000
Вихідний струм при логічній одиниці (24В), мА	5…600
Опір навантаження, Ом	48…4000
Потенційна розв'язка виходів групами по	8 або 16
Частота зміни сигналу при омічному навантаженні, Гц	100
Частота зміни сигналу при ламповому навантаженні, Гц	10
Частота зміни сигналу при індуктивному навантаженні, Гц	0,5
Параметризація діагностичних переривань	

В загалом цифрові модулі виводу призначені для керування реле, котрі в свою чергу комутують силові ланцюги. Існують різновиди модулів виводу, у яких вже вбудовані реле, що дозволяє безпосередньо комутувати на виходах напругу 220В, але обмежена кількість перемикань реле не дозволяє постійно їх перемикати, бо модуль вийде з ладу, а замінити вбудоване реле неможливо. Існують також модулі (SM 323), що містять входи та виходи разом.

Модулі вводу аналогових сигналів та їх перетворення у цифровий вид мають загальну назву SM 331. Їх основні технічні характеристики наведені у таблиці 1.4

Таблиця 1.4 – Основні технічні характеристики модулів вводу SM 331

Параметр	Значення
Кількість входів в модулі	2,4,8
Кількість розрядів у результаті перетворення	9…15 та знаковий
Параметризація вимірюваної величини (напруга, струм, опір, температура) та діапазону вимірювання	
Параметризація часу інтегрування результату вимірювання	
Параметризація діагностичних переривань	

При вимірюванні напруги можливе вимірювання однополярних чи двополярних сигналів у діапазонах від -80мВ…80мВ до -10В…+10В. Вимірювання струму виконується в двох основних діапазонах 0…20 мА та 4…20мА. Опір вимірюється до 600 Ом. Діапазон вимірювання температури залежить від датчика температури. Для запропонованих виробником контролерів датчиків можливе отримання результату у вигляді фізичної величини, а для інших датчиків у вигляді коду. Причому, щоб не вимірювалося – напруга, струм, опір, температура, існує нормальний діапазон результату вимірювання. Він становить від -27648 до +27648 для двополярних сигналів та від 0 до 27648 для одно полярних сигналів. Діапазон не змінюється в залежності від кількості розрядів внутрішнього аналого-цифрового перетворювача модуля.

В даному навчальному посібнику наведено дуже обмежену інформацію про технічні характеристики контролерів та деяких типів модулів, що підключаються до контролерів. На кожний контролер та модуль вводу-виводу існує докладна технічна інформація, яку можна отримати на сайті офіційного представника компанії Siemens [3].

Людина, яка розробляє програмне забезпечення для контролера, що є складовою частиною автоматизованої системи керування технологічним процесом обов’язково повинна на початку роботи ознайомитися з технічним описом контролера та приєднаних до нього модулів, щоб правильно їх параметризувати та обробляти отриману від них інформацію.

1.2 Поняття основного скану роботи контролера

В основі роботи контролерів Siemens знаходиться поняття основного циклу сканування. Цикл починається з того, що операційна система контролера опитує усі входи, що підключені до нього, та зчитує інформацію з входів в оперативну пам’ять входів, далі передаються комунікаційні дані. Після цього починає виконуватися програма з самим низьким пріоритетом – головна програма, що розташована у організаційному блоці OB1. Її роботу можуть переривати виклики інших програм з більш високим пріоритетом. Після виконання основної програми операційна система передає дані з області пам’яті, що відповідає виходам, безпосередньо на виходи. Слід пам’ятати, що коли в основній програмі Ви змінюєте біти пам’яті, що відповідають виходам, то ці зміни набирають сили тільки після закінчення всієї програми. Проаналізувати складне завдання

автоматизації - це значить розділити це завдання на ряд дрібних завдань або функцій та функціональних блоків у відповідності зі структурою процесу, для якого необхідно побудувати систему керування. Потім необхідно спланувати ті окремі функціональні частини, які отримані при розбивці загального завдання автоматизації процесу, щоб визначити закладені в них функції і сигнали зв'язку (інтерфейсу) із процесом та іншими функціональними частинами. Це поділ загального завдання керування на окремі функціональні частини може бути виконано у Вашій програмі. Таким чином, структура Вашої програми повинна відповідати структурі завдання автоматизації. Структурованість програми користувача спрощує конфігурування програми. Така програма може бути написана окремими блоками (у тому числі і декількома програмістами, у випадку великого об'єма). І, зрештою, не менш важливий момент - структурована користувальницька програма завжди є більш простою в роботах з пуску та налагодження. Структурованість програми залежить від її розміру і її функцій.

Розрізняються три варіанти структури програми: лінійна програма, фрагментована програма, структурована програма.

Лінійна програма характерна тим, що основна програма цілком розташовується в організаційному блоці ОВ 1. Програма розбивається на сегменти (networks). При редагуванні або налагодженні програми Ви можете посилатися на сегмент безпосередньо по його номеру.

Фрагментована програма характерна тим, що, власне кажучи залишаючись лінійної, програма розбивається на окремі блоки. Причиною такої розбивки програми можуть бути або занадто великий розмір програми для розміщення її в організаційному блоці ОВ 1, або необхідність поліпшення читаності програми. Блоки програми викликаються послідовно. Програма в окремому блоці може бути в такий же спосіб розбита, як і в організаційному блоці ОВ 1. Такий спосіб програмування дозволяє викликати окремі програми, пов'язані з окремими функціями процесу, з різних блоків. Перевага такої структури програми полягає також у тім, що незважаючи на те, що програма залишається лінійної, її можна налагоджувати і виконувати окремими фрагментами (просто додаючи або пропускаючи окремі виклики тих або інших блоків).

Структурована програма використовується, якщо: концептуальне формулювання завдання керування особливо широке, необхідно повторно використати функції програми, потрібно вирішувати складні завдання. Структурування програми означає розбивка програми на частині (блоки), які містять у собі незалежні функції, або які мають особливе функціональне призначення і при цьому обмінюються мінімально можливою кількістю сигналів з іншими блоками. Призначення для кожного блоку програми особливої функції (пов'язаної із процесом), дозволяє легко читати програмувальні блоки, спрощує інтерфейс з іншими блоками програми.

1.3 Типи даних та методи адресації в контролерах Siemens

Усі типи даних можна розділити на 4 основні групи: прості, складні, користувацькі та для параметрів. Змінні простих типів не можуть перевищувати 32 біт. До складних типів відносяться масиви, структури, рядки. Ці типи даних використовуються у мові SCL. До користувацьких типів відносяться блоки даних, а до параметричних таймери, лічильники, функції, функціональні блоки, невизначений тип даних та ін.

Зупинимося біль докладно на простих типах даних. До них відносяться: двійкове число BOOL (1 біт), однобайтне ціле без знаку BYTE (8 біт), двобайтне ціле без знаку WORD (16 біт), чотирибайтне ціле без знаку DWORD (32 біта), двобайтне ціле зі знаком INT (16 біт), чоритирибайтне ціле зі знаком DINT (32 біта), речовинне REAL (32 біта), часові змінні для таймерів S5TIME (16 біт), дата DATE (16 біт), системний час в IEC-форматі TIME (32 біта), системний час TIME_OF_DAY (32 біта) [1-2].

При запису значень констант та змінних типів цілих чисел можна використовувати десятковий формат (Наприклад, 10, -8 для цілих довжиною 16 біт та L#120000 для цілих довжиною 32 біта), двійковий (наприклад, 2#1000101100001111 для чисел довжиною 16 біт та 2#00100100000000111110001011000011111 для чисел довжиною 32 біта), шістнадцатиричний (Наприклад, W#16#0F21 для чисел довжиною 16 біт та DW#16#D0A521CE06 для чисел довжиною 32 біта). Інші типи даних будуть розглянуті більш докладно у відповідних розділах.

Змінні усіх типів знаходяться в оперативній пам’яті. У вивчаємих мовах програмування програміст повинен вказати місце в оперативній пам’яті, де буде розташовано змінну. При цьому використовується пряма та непряма адресація. Пряма адресація дозволяє звертатися до кожного біта, байта, слова та подвійного слова в будь-якій частині оперативної пам’яті. Звертання до області пам’яті входів починається з літери I (input – вхід), до області пам’яті виходів Q (quite – вихід) та маркерної пам’яті M (merker). Нижче наведено таблицю прикладів звертання до біта, байта, слова та подвійного слова.

Таблиця 1.5 – Приклади звертання до біта, байта, слова та подвійного слова

Звертання до біта	Звертання до байта	Звертання до слова	Звертання до подвійного слова
I 0.0 – нульовий біт у нульовому байті області входів	IB 0 – нульовий байт області входів	IW 0 – нульове слово області входів	ID 0 – нульове подвійне слово області входів
Q 1.2 – другий біт у першому байті області входів	QB 1 – перший байт області виходів	QW 1 – перше слово області виходів	QD 1 – перше подвійне слово області виходів
M 10.5 – п’ятий біт десятого байта меркерної пам’яті	MB 10 – десятий байт меркерної пам’яті	MW 10 – десяте слово меркерної пам’яті	MD 10 – десяте подвійне слово меркерної пам’яті

Слід звернути увагу, що адреси аналогових входів – це завжди слова, звертатися до яких потрібно як PIW *адреса* (наприклад, PIW 752 – 752-ге слово області входів).

Слід пам’ятати, що кожне слово займає у пам’яті 2 байта, а кожне подвійне слово – 4 байта. Пам’ять будь-яких змінних не повинна перехрещуватися. Тому, наприклад, якщо Ви використовуєте слово у маркерній пам’яті MW0, що займає нульовий та перший байти, то наступне слово, яке можна використовувати буде MW2, що займає 2-й та 3-й байти (рисунок 1.2).

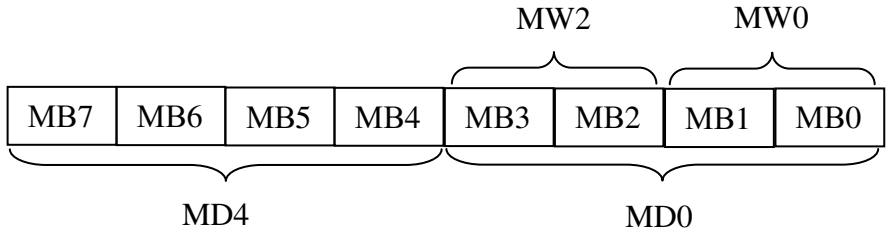


Рисунок 1.2 – Розташування слів та подвійних слів у пам’яті.

Непряма адресація буде розглянута пізніше у відповідному розділі 16.

Контролери Siemens можна програмувати 5-ти мовами, але тільки 3 з них (LAD, STL, FBD) входять до стандартного пакету STEP7. Інші дві можна придбати за додаткову платню. Ми розглянемо 2 основні мови програмування LAD та STL.

Перша мова – графічна. Програма цією мовою складається з графічних елементів, що є у бібліотеці. Інструкції записуються у рядки (networks) та виконуються зліва направо, а рядки зверху до низу. Друга мова – це набір асемблерних команд контролера. Кожна команда записується з нового рядка, а програма виконується тільки зверху до низу. Ця мова, на відміну від LAD, має більше різноманітних команд, більш гнучка, компактна.

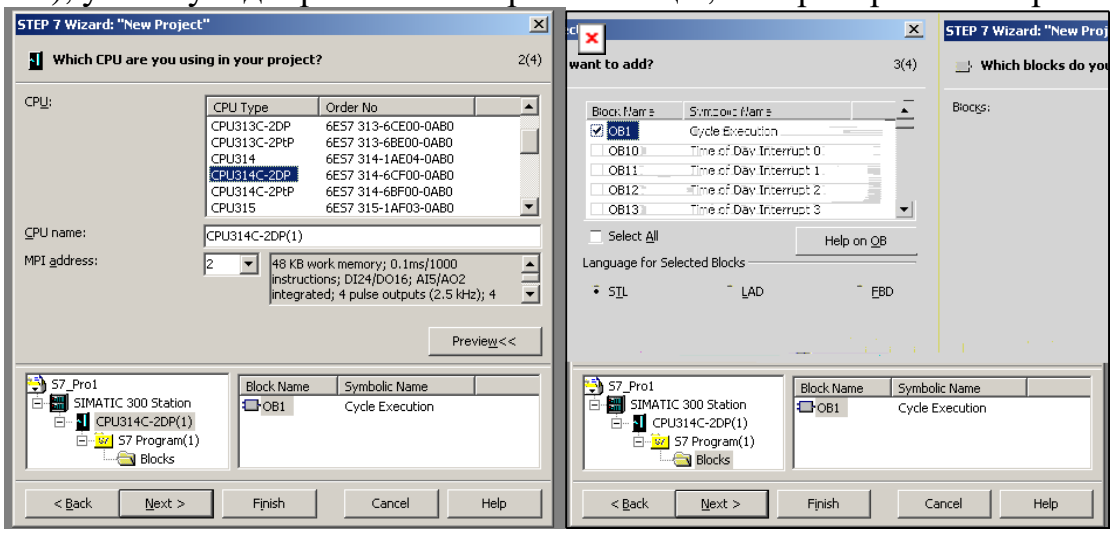
2 Ознайомлення з пакетом програмування контролерів Siemens – Step7

Фірма – виробник Siemens – систем S7-300 поставляє спеціальне середовище програмування **SIMATIC MANAGER** і середовище симулювання розроблених програм – S7-PLCSIM. **SIMATIC MANAGER** є повністю автономним середовищем, містить всі необхідні вбудовані засоби для набору вихідного тексту програми мовами STL та LAD і його компіляції, а симулятор S7-PLCSIM дозволяє налагодити введену програму та переглянути результати її виконання.

Для того, щоб почати роботу в пакеті **SIMATIC MANAGER** необхідно виконати наступну послідовність операцій.

Після запуску програми **SIMATIC MANAGER** запустити майстер створення проектів (пункт меню File\New project\Wizard). У вікнах, що відкриваються послідовно, вибрати тип використовуваного контролера (наприклад CPU314C-2DP, рисунок 2.1 а), програмні модулі, з яких буде складатися програма (OB1...) і мову програмування (STL,LAD і т.і.) (рисунок 2.1 б). При виборі програмних модулів обов'язковим є лише блок OB1, що містить програму основного робочого циклу обробки. У наступному вікні, що з'являється, потрібно задати ім'я проекту (наприклад, ім’я «Test», рисунок 2.2). Після завершення

формування проекту користувачу відкривається вікно вмісту проекту (рисунок 2.3), у якому відображається обрана станція, контролер і вміст проекту.



а б

Рисунок 2.1 – Створення проекту в автоматичному режимі

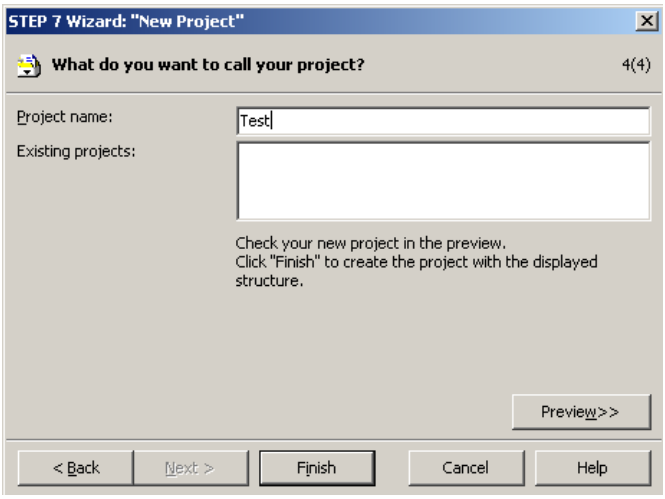
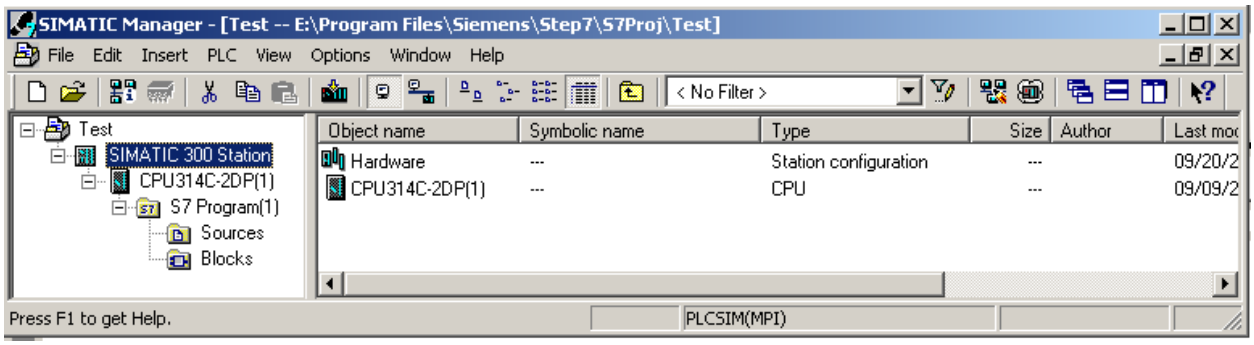
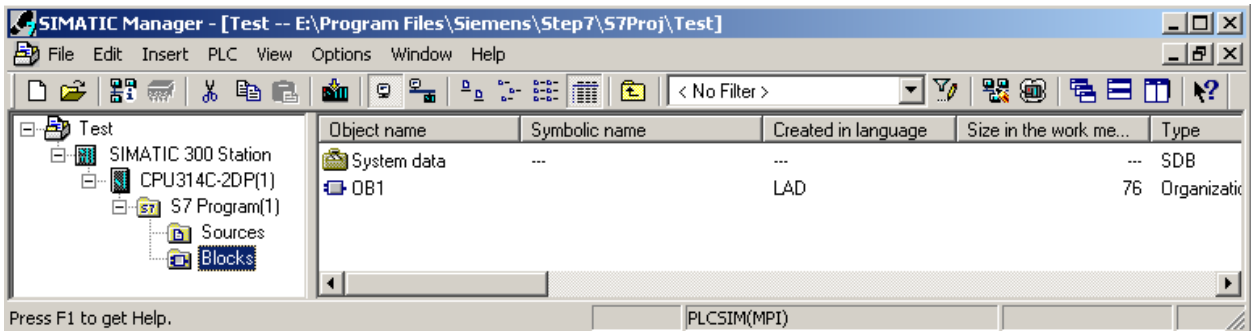


Рисунок 2.2 – Графічне вікно вводу імені проекту



а



б

Рисунок 2.3 – Вікно конфігурації створеного проекта: а – складові обладнання; б – програмні модулі

Далі слід повністю сконфігурувати обладнання, що підключене до контролера. Для цього необхідно ліворуч вибрати робочу станцію (наприклад, Simatic 300 Station рисунок 2.3 а), а у вікні праворуч запустити програму – Hadware Config (HW config). Після цього Вам відкриється вікно конфігурування станції (рисунок 2.4).

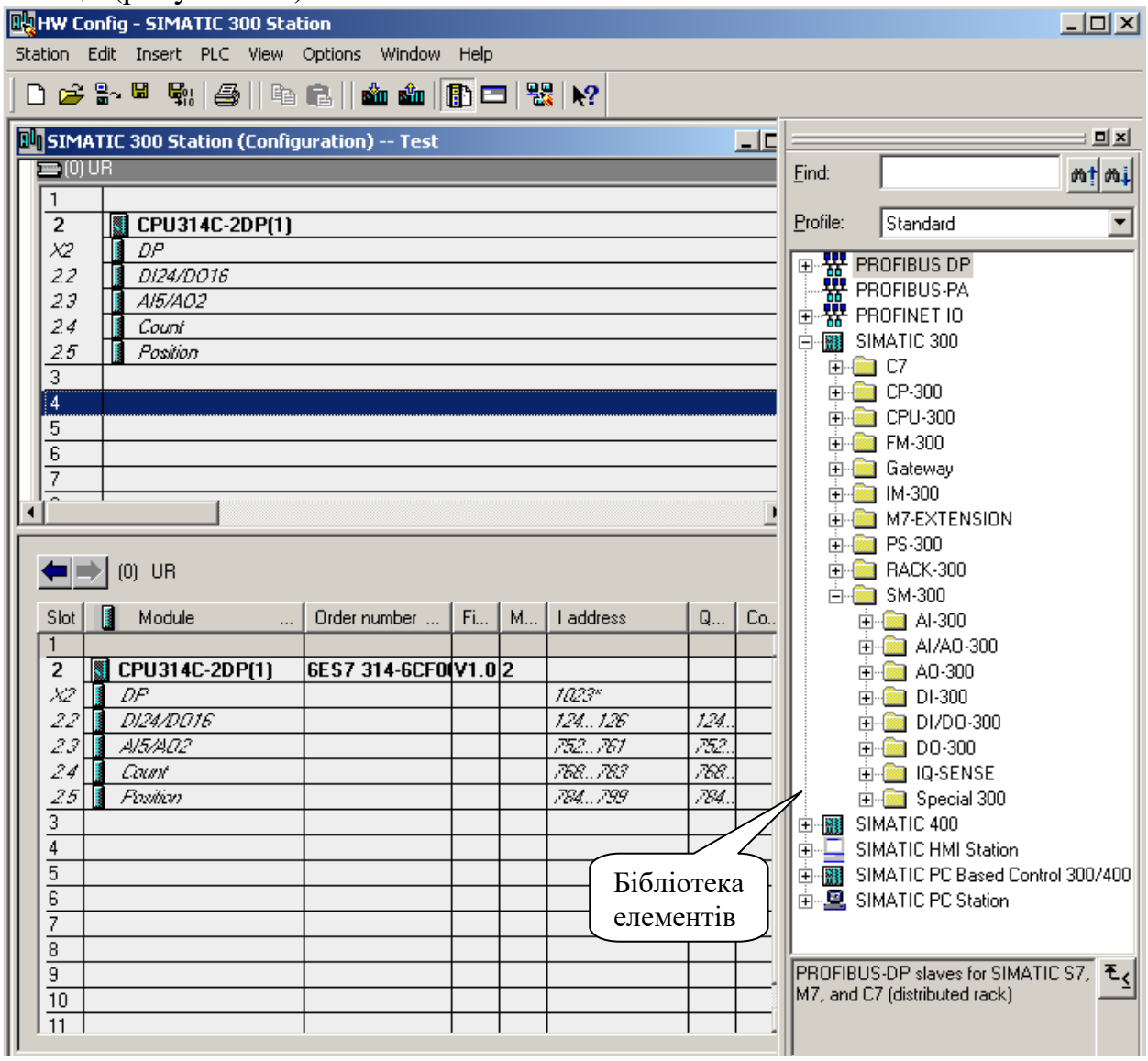


Рисунок 2.4 – Вікно конфігурування устаткування

В лівій частині умовно зображена DIN рейка (у вигляді таблиці), на якій у слоту №2 знаходиться контролер. Цей контролер має вбудовані в нього цифрові та аналогові входи та виходи, швидкісний лічильник, позиціонер, що відповідно можна побачити на рисунку 2.4 (цифрові входи DI, виходи DO, аналогові входи AI, аналогові виходи AO, швидкісний лічильник Count, позиціонер Position). Лише деякі контролери мають вбудовані входи чи виходи. Коли їх не достає чи зовсім не існує, то до контролера через системну шину під'єднують модулі з цифровими чи аналоговими входами, виходами та багато інших пристроїв. У програмі Hadware ці модулі можна додати з бібліотеки, що міститься праворуч. Модулі станції S7-300 містяться у розділі SIMATIC 300, модулі станції S7-400 – у розділі SIMATIC 400 і т.і. Крім контролера обов'язково встановити модуль живлення з бібліотеки SIMATIC 300/PS-300 (тільки слот 1). Інші модулі обираються та встановлюються за необхідністю: контролери з бібліотеки SIMATIC 300/CPU-300 (тільки слот 2); аналогові модулі вводу з SIMATIC 300/SM-300/AI-300; аналогові модулі виводу з SIMATIC 300/SM-300/AO-300; цифрові модулі вводу з SIMATIC 300/SM-300/DI-300; цифрові модулі виводу з SIMATIC 300/SM-300/DO-300 і т.і. Ці модулі встановлюються у слоти з четвертого по одинадцятий. Усі входи чи виходи у модулях мають свої адреси, що можна дізнатися у полях I address та Q address відповідно. У третій слот можна встановити тільки комунікаційний модуль, що дозволяє підключити до контролера ще декілька DIN реєк з модулями вводу-виводу чи іншими пристроями. Ці модулі знаходяться у бібліотеці SIMATIC 300/IM300 для 300-ї серії контролерів.

Коли до контролера необхідно підключити винесену периферію, тобто модулі вводу-виводу чи інші пристрої, що знаходяться на значній від нього відстані, то застосовують комунікаційні модулі серії ET200. Таке технічне рішення є дуже зручним у багатьох реальних ситуаціях, коли необхідно обробляти інформацію від великої кількості датчиків чи керувати багатьма приладами, що розташовані на значній відстані від центрального контролеру. Проводити до контролеру багато довгих кабелів від датчиків і приладів незручно та економічно необгрунтовано. Тому кабелі підключають до винесеної переферії (модулі вводу-виводу), що розташовують близько від датчиків та об'єктів керування. Структура мережі у цьому випадку така. До модуля ET200 підключають будь-які модулі чи пристрої збору і виводу інформації, але він не здійснює керування, а лише організує передавання зібраних даних у мережах PROFIBUS-DP, PROFINET до центрального контролера та від нього. Якщо модуль розрахований на передавання даних у мережі PROFIBUS-DP, то він знаходиться у бібліотеці PROFIBUS DP, а коли на роботу у мережі PROFINET, то у бібліотеці PROFINET IO.

Щоб встановити будь-який модуль його слід перетягнути мишкою з бібліотеки у відповідний слот. Усього слотів у одному ряду може бути не більше 11, а коли їх потрібно встановити більше, то декілька рядків поєднують з центральним контролером комунікаційними модулями (наприклад IM360). При цьому кожний контролер і комунікаційний модуль у мережі повинні мати уні-

кальні адреси. На рисунку 2.5 наведено приклад конфігурації обладнання з використанням контролера CPU 315-2 DP.

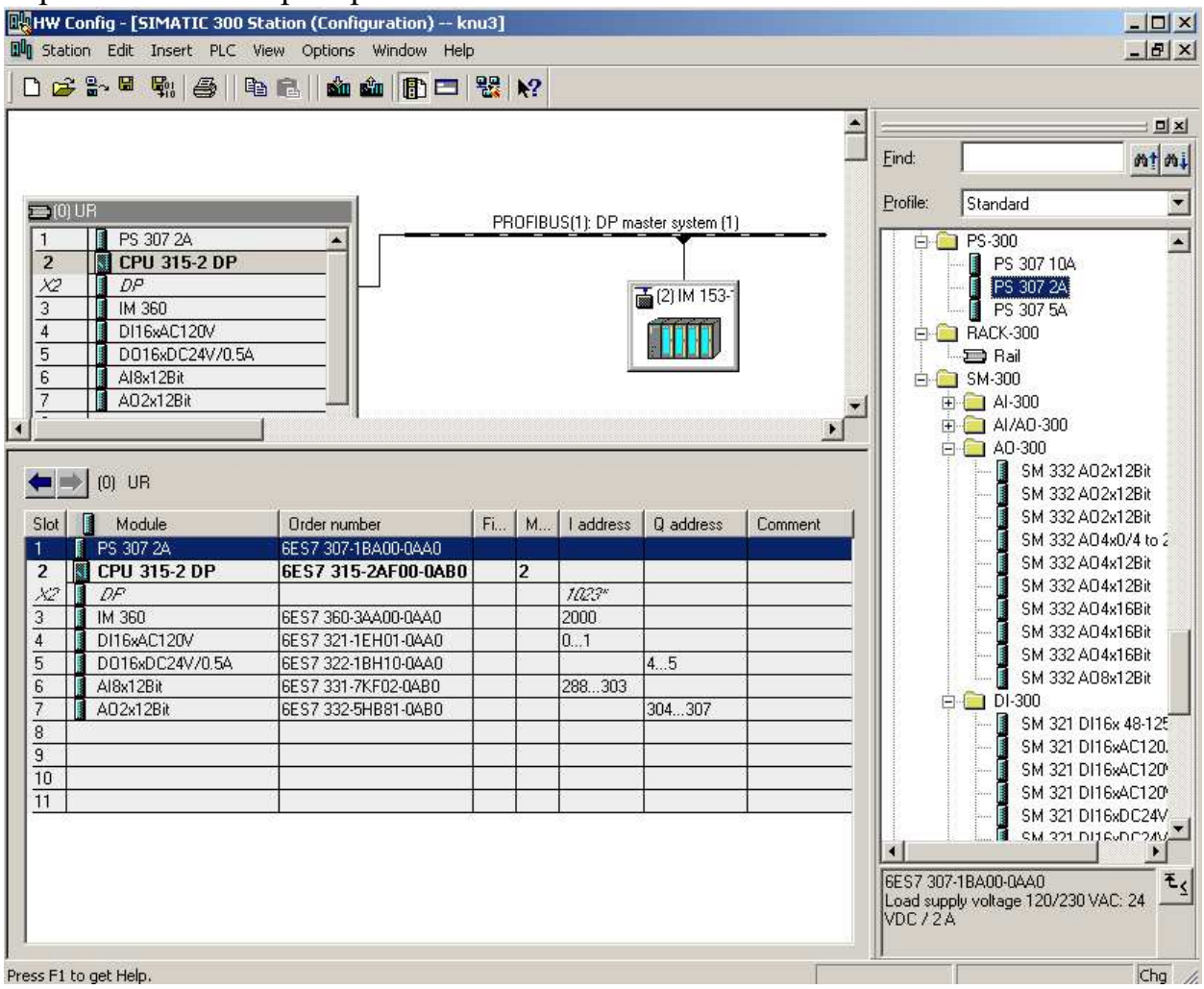


Рисунок 2.5 – Підключення до мережі PROFIBUS інтерфейсного модуля IM153-1

У структурі обладнання використано блок живлення PS 307 2A, інтерфейсний модуль IM 360 для підключення ще трьох DIN-рейок, цифровий модуль вводу DI16xAC120V з 16 входами (байти адреси 0…1), цифровий модуль виводу з 16 виходами DO16xDC24V/0.5A (байти адреси 4…5), аналоговий модуль вводу AI8x12Bit з 8 входами (байти адреси 288…303) та аналоговий модуль виводу AO2x12Bit з двома виходами (байти адреси 304…307). Підключення винесеної периферії (модулів, що не мають власного контролера та територіально далеко від головного контролера) здійснюється завдяки модулю IM-153-1, що забезпечує передавання даних інтерфейсом PROFIBUS-DP. Підключення модулів вводу-виводу до IM-153-1 здійснюється відповідно підключенню модулів до головного контролера.

Після закінчення набору модулів, з яких буде складатися станція S7-300, необхідно скомпілювати конфігурацію і завантажити її в проект. Для цього слід нажати відповідну кнопку компіляції  , а також кнопку завантаження у симулятор чи реальний контролер  . Якщо у Вас немає реального контролера, то

попередньо необхідно завантажити програму симуляції роботи контролера. Для завантаження симулятора S7-PLCSIM необхідно натиснути кнопку , у головному вікні проекту (рисунок 2.3), а також перевести контролер у режим RUN-P у вікні S7-PLCSIM, що відкрилося (рисунок 2.6).

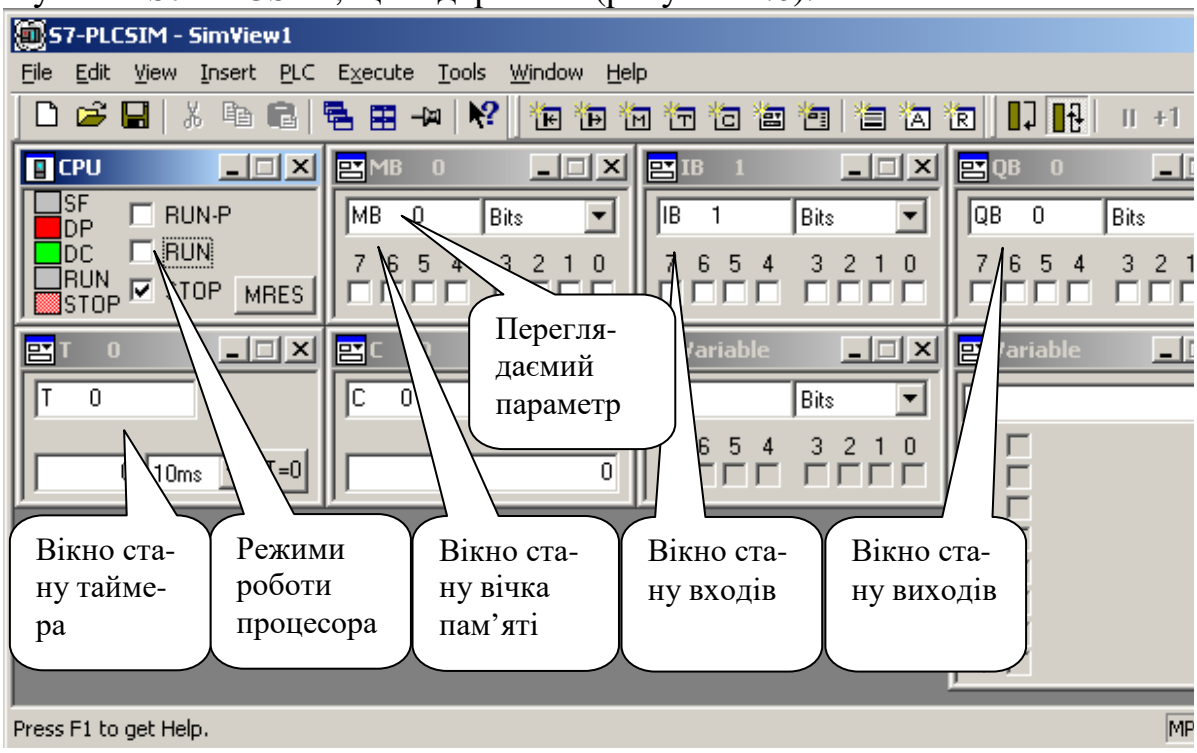


Рисунок 2.6 – Вікно симуляції роботи програми

Для перегляду стану входів, виходів, пам'яті і т.і. в симуляторі існують відповідні вікна, котрі можна викликати натисканням кнопок: (входи), (виходи), (меркерна пам'ять), (таймер), (лічильник). Крім того у будь-якому вікні перегляду стану можна змінити параметр, значення котрого треба переглянути, ручним вводом відповідної назви входу, виходу і т.д.

Для налаштування інтерфейса програматора при завантаженні конфігурації обладнання у реальний контролер необхідно:

1) у головному вікні проекту (рисунок 2.3) обрати пункт меню Options/Set PG/PC Interface. Далі з'являється вікно, зображене на рисунку 2.7, що містить перелік інтерфейсів для програмування контролеру;
2) налаштувати параметри програматора CP5611 при використанні інтерфейсів програмування MPI чи PROFIBUS-DP (у випадку коли в комп'ютер встановлено саме цей програматор). При можливості програмування контролеру через інтерфейс Ethrnet необхідно обрати вбудований в комп'ютер адаптер Ethrnet з позначкою auto.

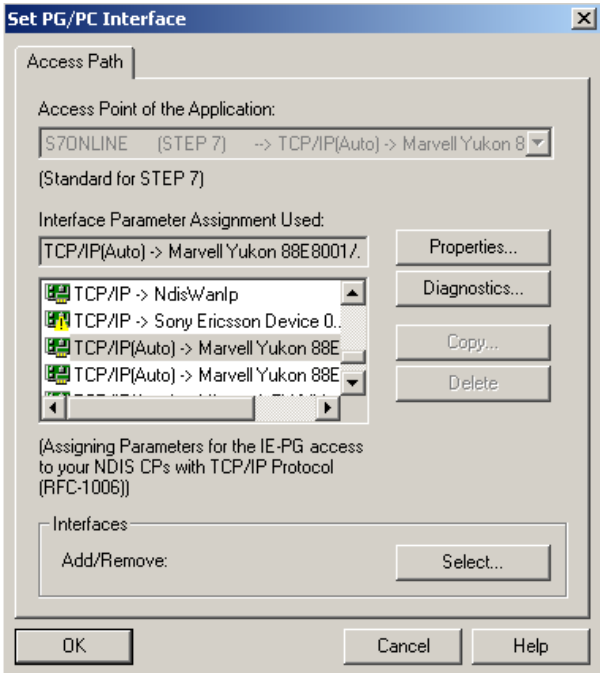


Рисунок 2.7 – Графічне вікно налаштування інтерфейсу програмування контролеру

При будь-якому способі програмування можна протестувати підключене обладнання та налаштувати передавання даних після натискання кнопок Diagnostics та Properties .

По закінченню компіляції та завантаженню вікно конфігурації станції (рисунок 2.5) необхідно закрити.

У цьму навчальному посібнику будуть вивчатися дві основні мови програмування контролерів Siemens – LAD та STL, що входять до складу стандартного пакету програмування Step7 Standard. При вивченні матеріалу посібника за допоміжною інформацією можна звертатися до літературних джерел [1-2], а т акож до internet-ресурсу [3].

Для вводу своєї програми в модуль OB1 чи інші сформовані блоки необхідно вибрати папку Blocks (рисунок 2.3) із вмістом проекту і подвійним клацанням миші на обраному блоці праворуч відкрити вікно створення програми (рисунок 2.8).

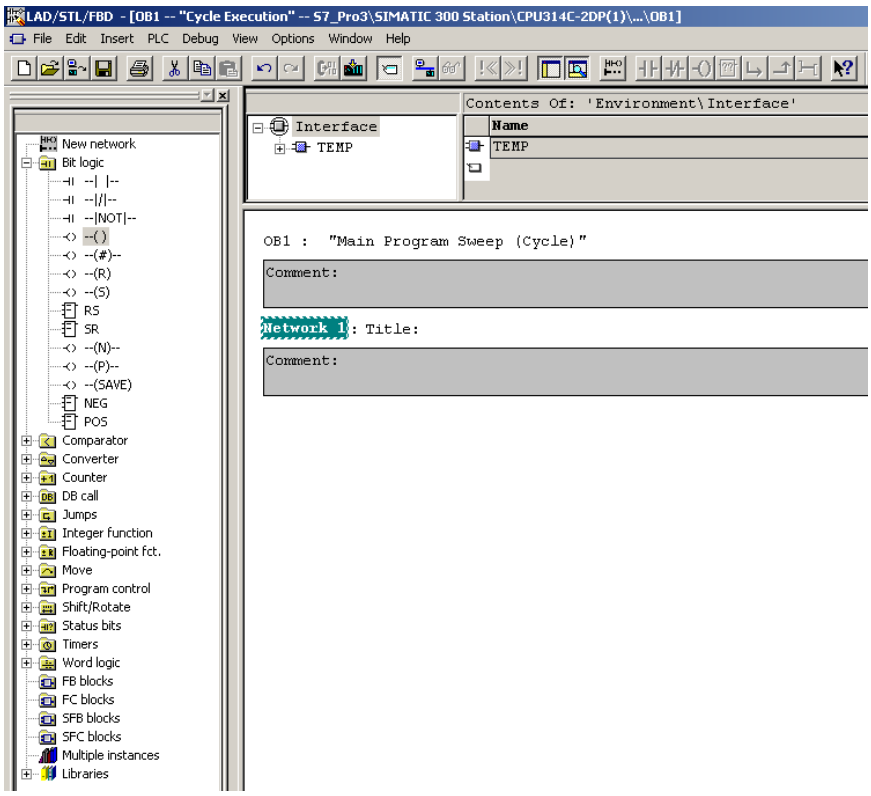


Рисунок 2.8 – Приклад розробленої програми

У представленому вікні ліворуч відображена бібліотека стандартних інструкцій-елементів середовища розробки, а праворуч у вигляді послідовного з'єднання цих елементів представлений фрагмент коду програми (рисунок 2.9). При наборі коду програми у вікно праворуч перетаскують відповідні елементи з бібліотеки. Для послідовного з'єднання декількох елементів (мова LAD) необхідно виділити курсором частину лінії та перетягнути на виділену частину лінії новий елемент (рисунок 2.9).

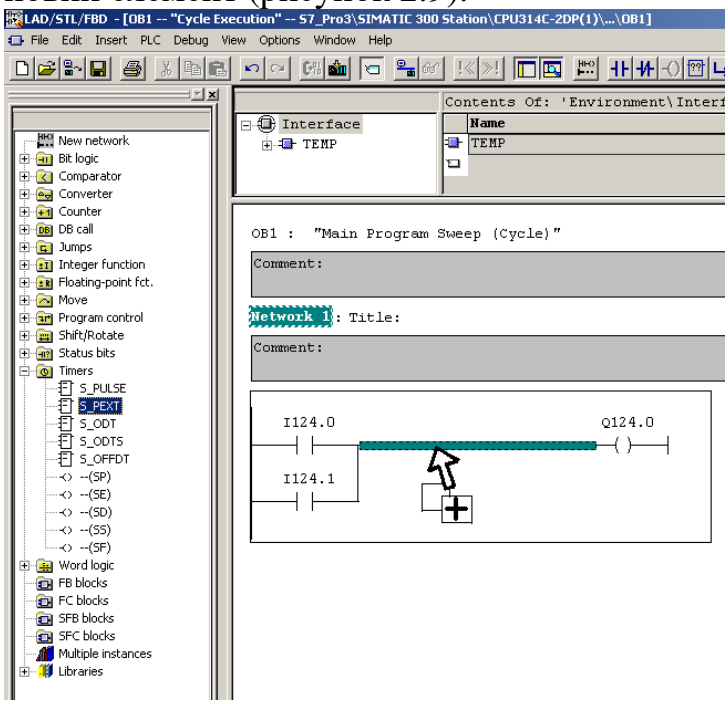
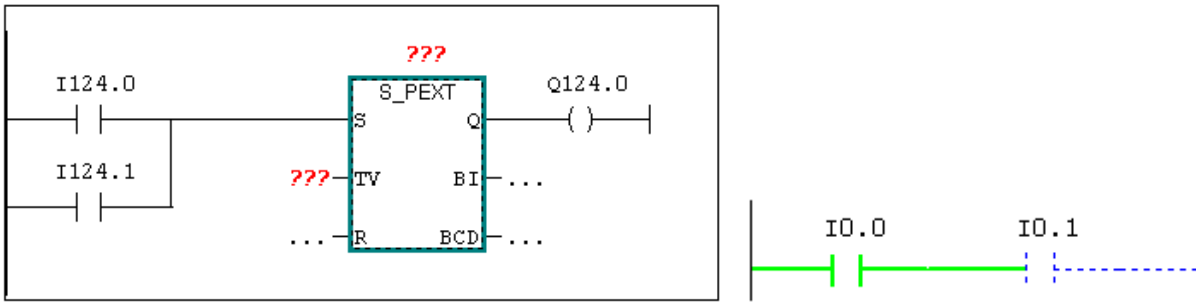


Рисунок 2.9 – Набір програми мовою LAD

Замість виділеної лінії може використовувати вхід чи вихід раніше встановленого елементу. Після набору ланцюжка інструкцій-елементів програми необхідно задати вхідні, вихідні адреси, меркери пам'яті і т.і. При вводі назв входів, виходів, меркерів пам'яті можуть використатися їх прямі адреси (I 0.0, Q 4.0, M 1.3) або умовні (“Key1”, Key2”,”Light”). Якщо програма набрана вірно, то в програмі не повинно бути позначок компілятора червоним кольором (рисунок 2.10 а).



а б

Рисунок 2.10 – Приклад програми з синтаксичними помилками

При наявності таких позначок необхідно виправити синтаксичні помилки в програмі.

Після закінчення набору програми необхідно перейти в меню проекту (рисунок 2.3).

Для початку симуляції необхідно натиснути кнопку  . Далі, треба виділите всі блоки, що входять у проект, і завантажте їх у контролер, натиснувши кнопку  . Якщо до цього все виконано вірно, то з'явиться вікно симулятора, що наведено на рисунку 2.6.

Для початку симуляції необхідно вибрати пункт RUN-P в CPU. В інших відкритих вікнах задаються вхідні впливи, відображаються вихідні, стан пам'яті і т.і.

Результати поточного виконання програми можна переглянути після натискання кнопки  (рисунок 2.10 б). При перегляді, з'єднання, що позначені чорною пунктирною лінією, відповідають логічному 0, а зеленою - логічній 1. Стан входів, виходів, пам’яті що необхідно контролювати можна також переглядати у відповідних вікнах стимулятора (рисунок 2.6). Крім того в цих вікнах можна змінювати вручну стан будь-якого вічка пам’яті, входу, виходу, імітуючи зміни, необхідні для налагодження програми.

У випадку перегляду результатів виконання програми у реальному контролері та стану його входів, виходів, пам’яті використовується зображене на рисунку 2.11 вікно, яке можна викликати через пункт меню PLC\Monitor/Modifai variables вікна розробки програми

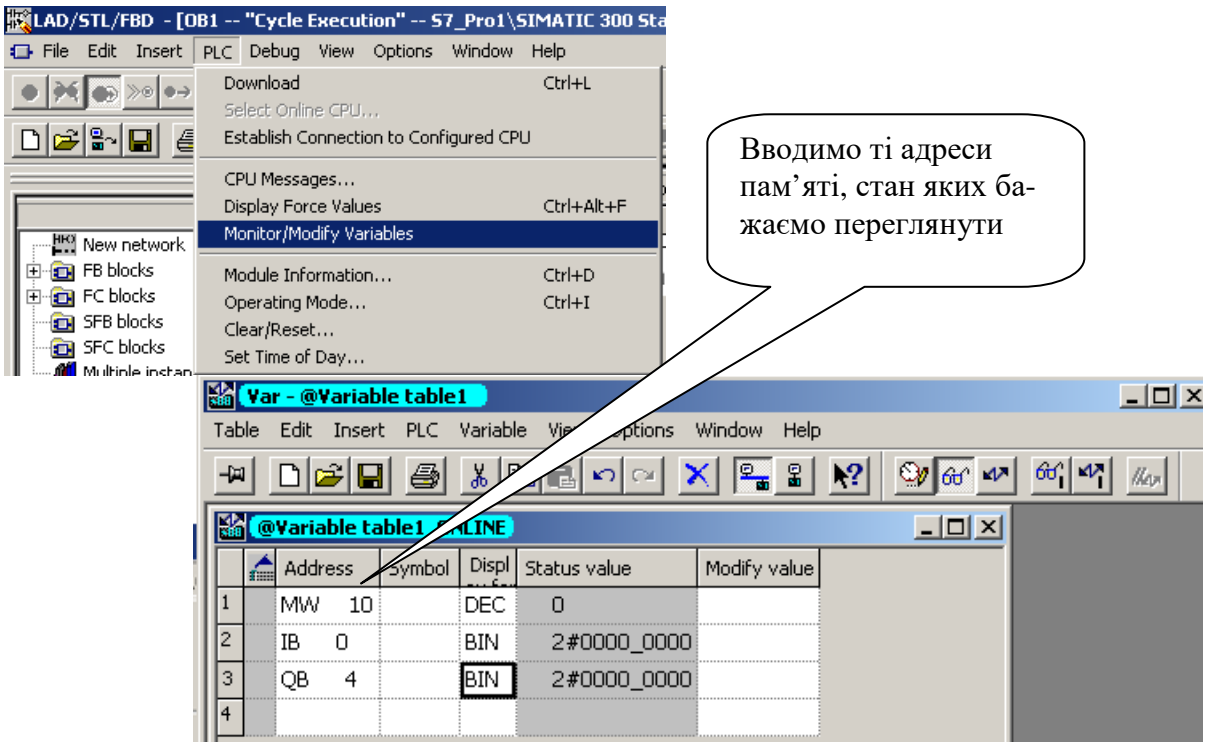


Рисунок 2.11 – Виклик вікна перегляду стану пам'яті контролера

Після виклику вікна перегляду стану пам'яті контролера відкривається таблиця variable table, у рядках якої вводимо необхідні нам адреси пам'яті. Після натискання кнопки перегляду стану  у стовпчику Status value відображується значення, що записане за цією адресою. Нове значення користувач може ввести в стовпчику Modify value. Натиснувши далі кнопку передавання введеного значення  , змінюємо значення за введеною адресою. Основною відміною від роботи з симулятором є неможливість зміни стану входів, бо операційна система контролера сама опитує входи та змінює стан пам'яті, відповідної до входів.

Наведене на рисунку 2.11 вікно дуже зручне та наглядне при перегляді та зміні стану пам'яті контролера.

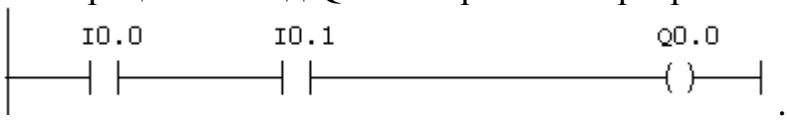
3 Вивчення інструкцій роботи з бітами у мовах LAD та STL

3.1 Вивчення інструкцій роботи з бітами у мові LAD

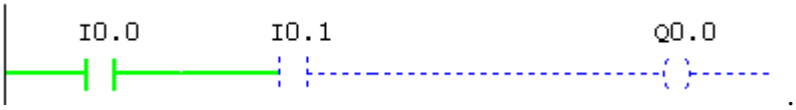
Для опитування одиночних входів чи будь-яких бітів в оперативній пам'яті контролера використовуються команди «нормально розімкнутий» I0.3 ┤ ├ та «нормально замкнений» I0.1 ┤/├ контакти. Над графічним зображенням команд вказується адреса входу чи біта пам'яті, який ми опитуємо. На виході елемента «нормально розімкнутий контакт» логічна одиниця, якщо на його вході логічна

одиниця та у біті пам'яті, що ми опитуємо. На виході елемента «нормально замкнений контакт» логічна одиниця, якщо на його вході логічна одиниця та нуль у біті пам'яті, що ми опитуємо. Комбінації з цих елементів дозволяють створити будь-яку логічну схему. Результат логічних операцій в кінці рядка програми (network) потрібно записувати на вихід, або у інший біт оперативної пам'яті. Для цього використовуються елементи «обмотка реле» Q0.0 ( ) або «установити» Q0.5 (S) чи «зняти» Q0.4 (R). Над графічним зображенням кожної з команд вказується адреса виходу чи біта пам'яті, стан якого ми хочемо змінити. Команда «обмотка реле» передає на вихід 1 тільки поки на її вході логічна одиниця.

Приклад 1. Розробимо програму, що опитує два логічні входи (I0.0 та I0.1), виконує логічну операцію «ТА» з результатами опитування та передає результат операції на вихід Q0.0. Розроблена програма має наступний вигляд

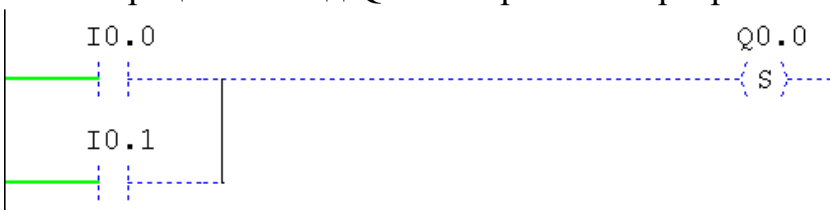


Та ж програма, але в режимі перегляду результатів виконання (після завантаження до контролеру чи симулятору і натискання кнопки ) має вигляд



Безперервною лінією відображено сигнал логічної «1», а пунктирною сигнал логічного «0». З результату виконання можна бачити, що на вході I0.0 зараз логічна «1», на вході I0.1 – логічний «0». Результат виконання логічної операції – «0», який і передається на вихід Q0.0.

Приклад 2. Розробимо програму, що опитує два логічні входи (I0.0 та I0.1), виконує логічну операцію «АБО» з результатами опитування та передає результат операції на вихід Q0.0. Розроблена програма має наступний вигляд



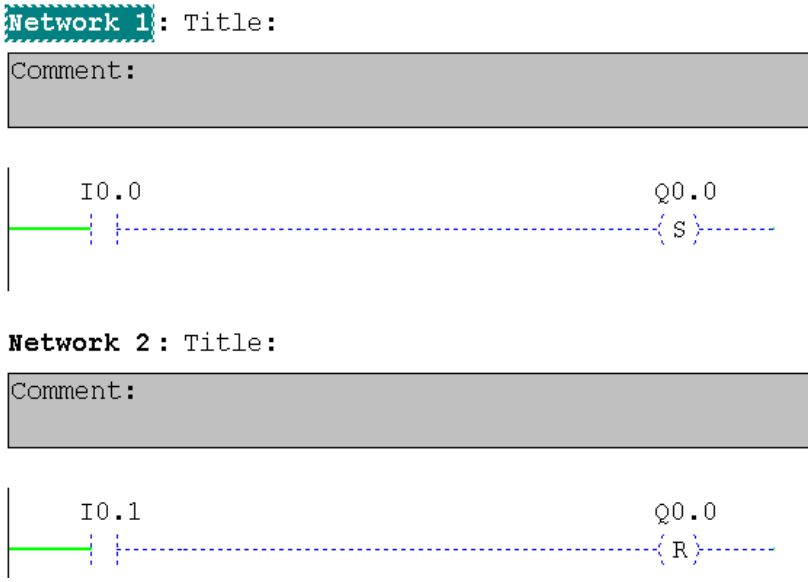
Операція «АБО» реалізується паралельним з'єднанням двох елементів «нормально розімкнутий» контакт. Щоб створити паралельне з'єднання елементів необхідно в рядку встановити перший «нормально розімкнутий» контакт, потім мишкою виділити лінію перед ним та натиснути кнопку  з панелі меню. У отриманій паралельній вітці програми встановлюємо другий елемент «нормально розімкнутий» контакт та за допомогою кнопки  чи мишкою з'єднуємо два паралельних ланцюга програми.

Команда «установити» 

видає на вихід одиницю коли на її вході з'являється «1» та утримує її на виході поки на елемент «зняти» 

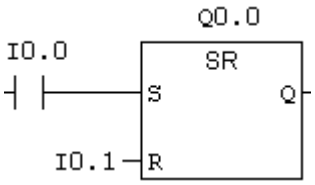
з тою ж адресою не буде подана логічна одиниця.

Приклад 3. Розробимо програму, що керує станом виходу Q0.0 при натисканні двох кнопок без фіксації. Назвемо ці кнопки Start та Stop. Підключимо їх до входів контролера I0.0 та I0.1 відповідно. Після натискання кнопки Start та її наступного розмикання на виході повинен зберігатися стан логічної «1» до натискання кнопки Stop.



Дана програма складається з двох рядків. У першому опитуємо кнопку Start, що підключена до входу I0.0 та подаємо на вихід логічну «1». У другому рядку опитуємо кнопку Stop, що підключена до входу I0.1 та подаємо на вихід логічний «0».

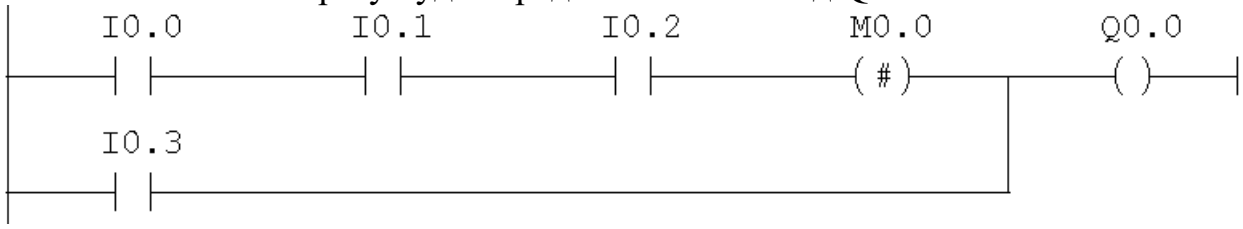
Якщо команди «установити» (S) та «зняти» (R) можуть бути записані компактно у один рядок програми, то використовують їх об'єднання – тригери SR та RS. Над графічним зображенням цих команд записують адресу виходу, яким керують, а на входах S та R відповідні логічні умови для керування станом виходу. Нижче наведено розв'язання попереднього приклада з використанням тригеру SR.

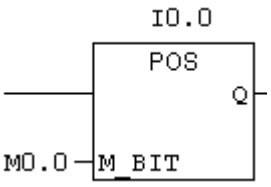


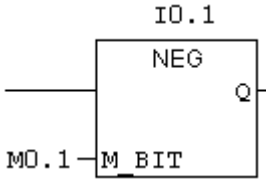
Слід зауважити, що коли на обидва входа SR тригера подано логічну «1», то на виході Q0.0 логічний «0». Це пояснюється тим, що операційна система контролера оновлює стан виходів тільки після виконання всієї програми. В наведеній програмі в одному скані спочатку виконується команда, що подає на вихід «1», а потім команда, що подає «0». Останньою є команда скидання виходу. Тому вона і виконується, а логічна одиниця на виході ніколи не з'являється.

Крім зазначених вище інструкцій –( )–, –(S)–, –(R)–, що дозволяють змінювати стан будь-якого біта оперативної пам’яті контролера, але повинні записуватися лише у кінці рядка програми, ще використовують інструкцію зберігання проміжного результату логічної операції –(#)–. Ця інструкція може встановлюватися у будь-якому місці рядка програми, крім його кінця.

Наведемо приклад програми, що реалізує операцію «ТА» з логічними сигналами з входів I0.0, I0.1, I0.2 та наступну логічну операцію «АБО» з попереднім результатом та логічним сигналом з входу I0.3. Проміжній результат логічної операції «ТА» буде зберігатися у біті маркерної пам’яті M0.0, а результат всього логічного виразу буде передаватися на вихід Q0.0.



До команд роботи з бітами відносяться також елементи виділення позитивного (з 0 в 1) 

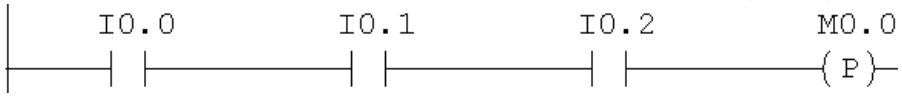
та негативного (з 1 в 0) 

фронтів сигналу. У мові LAD над цими елементами потрібно вказати адресу сигналу, позитивний чи негативний фронт якого потрібно зареєструвати. На вході M_BIT потрібно вказати біт пам’яті у якому буде знаходитися попереднє значення стану аналізуємого сигналу. Кожного разу, коли поточний стан сигналу, що аналізують, порівнюється з попереднім, що міститься у біті пам’яті з входу M_BIT, він автоматично стає попереднім. На вхід без назви потрібно подати логічну 1, яка дозволяє роботу цих елементів. На виході цього елементу формується логічна одиниця, коли знайдено фронт сигналу. Ця одиниця утримується на виході тільки протягом 1 циклу роботи контролера.

Різновидом команд виділення фронту сигналу є команди

та 

. Сигнал, фронт якого треба знайти подається на вхід цих команд. Над кожною з команд вказується біт пам’яті контролера, у якому зберігається попередній стан логічного сигналу, що подається на вхід команди. Основною відмінністю цих команд від POS та NEG є можливість аналізувати не тільки стан вхідного сигналу, а і стан результату логічної операції будь-якої складності. Наведемо фрагмент програми, у якому виділяється фронт сигналу, що є результатом логічної операції «ТА» із значеннями сигналів на входах I0.0, I0.1, I0.2.



Для інвертування стану логічного сигналу призначена інструкція 

.

3.2 Вивчення інструкцій роботи з бітами у мові STL

В контролері існує регістр стану, котрий складається з бітів, стан яких залежить від результатів виконання тієї чи іншої команди будь-якої мови програмування. Один із бітів цього регістра – RLO (результат виконання логічної операції). Коли ми зчитуємо з одного дискретного входу його стан, то це значення попадає у біт RLO, так само і будь-які логічні операції виконуються із деяким бітом пам'яті та RLO. Результат логічних операцій завжди знаходиться у RLO.

У мові STL існують три логічні операції над бітами «ТА», «АБО», «ВИКЛЮЧНЕ АБО» і відповідні їм інструкції A, O, X. При використанні цих інструкцій вказується біт у пам'яті контролера, до якого слід застосувати логічну операцію (Наприклад, A I 124.0). Якщо інструкція використана одноразово, то значення біта, що опитують, копіюється у біт RLO контролера і з ним можна виконувати будь-які дії, а якщо одна і та ж інструкція записана послідовно двічі, то виконується відповідна логічна операція. Наприклад, програма, що реалізує логічну операцію «АБО» над бітами, що зчитані з входів I0.0 та I0.1 має вигляд

```
O   I 0.0 //Логічна операція «АБО» з сигналами з входів I0.0 та I0.1
O   I 0.1
```

В даному прикладі і в подальшому коментарі в програмі записуються після символів //.

Коли треба зчитувати інверсне значення біта використовуються інструкції AN, ON, XN, що відповідають інструкції «нормально замкнений контакт» мови LAD. Для запису на вихід, чи у будь-який біт пам'яті, результату логічної інструкції використовується інструкція =, що відповідає інструкції «обмотка реле»  мови LAD. Для реалізації складних логічних виразів разом з логічними інструкціями використовуються дужки, що задають послідовність виконання логічних інструкцій. Наприклад, у програмі, що наведена нижче, спочатку реалізується операція «АБО» над значеннями біт I124.0 та I124.1, потім операція «АБО» над значеннями біт I124.2 та I124.3, а наприкінці операція «ТА» над результатами попередніх логічних операцій. Кінцевий результат зберігається у біті M10.0 меркерної пам'яті.

			RLO	STA	STANDARD
A(			1	1	0
O	I	124.0	1	1	0
O	I	124.1	1	0	0
)			1	1	0
A(			1	1	0
O	I	124.2	0	0	0
O	I	124.3	1	1	0
)			1	1	0
=	M	10.0	1	1	0

Рисунок 3.1 – Текст програми складного логічного виразу та результати її виконання

Праворуч від тексту програми наведено вікно результатів її виконання, що з'являється після натискання кнопки  у панелі меню (рисунок 3.1). Перший стовпчик у цьому вікні – значення біта RLO після виконання кожного рядка програми. Другий (STA) – результат логічної операції. З даного вікна ми бачимо, що при виконанні програми на входах I124.0, I124.2 та I124.3 логічна одиниця. Результати виконання кожної з операцій «АБО» – «1», результат виконання операції «ТА» теж «1». Цей результат записано у біт M10.0.

Для встановлення на виході чи у біті пам'яті стану логічної «1» з утриманням стану виходу використовуються інструкції S. Для скидання стану виходу чи біта пам'яті використовується інструкція R.

Наприклад, програма, що використовує результат операції «АБО» над значеннями біт I124.0 та I124.1 для встановлення виходу Q 0.0, а результат операції «АБО» над значеннями біт I124.2 та I124.3 для скидання виходa Q 0.0, має вигляд

```
O   I 124.0 //Виконання логічної операції «АБО» із сигналами з входів
O   I 124.1 // I 124.0 та I 124.1
S   Q 0.0 // Встановлення виходу Q 0.0 в стан логічної «1»
O   I 124.2 // Виконання логічної операції «АБО» із сигналами з входів
O   I 124.3 // I 124.2 та I 124.3
R   Q 0.0 // Скидання виходу Q 0.0
```

На відміну від мови LAD спеціальної команди збереження проміжного результату логічних операцій не існує, бо у будь-якій частині програми можна зберегти поточний стан біту RLO у будь-який біт пам'яті контролера. При цьому цю операцію можна виконувати необмежену кількість разів. Наприклад результат логічної операції «ТА» збережемо у біті пам'яті M0.1 та виведемо на вихід Q124.0

```
A   I0.0 //Виконуємо логічну операцію «ТА» з сигналами з входів
A   I0.1 // I0.0 та I0.1
```

= M0.1 //*Зберігаємо результат логічної операції у біті пам'яті M0.1*
= Q124.0 //*Передаємо той ж самий результат на вихід Q124.0*

Для виявлення позитивного чи негативного фронтів сигналу використовуються команди FP та FN відповідно. Разом з командою вказується біт у пам'яті контролера де автоматично зберігається попереднє значення сигналу, що аналізуємо. Перед використанням цих інструкцій слід опитати біт, зміну якого ми знаходимо. Наприклад,

A I 0.0 //*Опитуємо вхід I 0.0*
FP M 0.0 //*Знаходимо позитивних фронт сигналу з входу I 0.0.*

Крім зазначених вище інструкцій в мові STL використовуються також інструкції безпосереднього встановлення (SET) та скидання (CLR) стану біта RLO, а також інвертування стану RLO – команда NOT.

Порівнюючи інструкції роботи з бітами мов LAD та STL можна зробити висновок, що істотних переваг при використанні однієї чи другої мови немає, бо об'єм програми та її швидкодія при використанні однієї чи другої мови можуть відрізнятися не більше ніж на 5%. Програма, розроблена мовою LAD, більш наглядна. При реалізації складних логічних умов перевагу слід надавати мові STL.

Контрольні питання.

1) Для чого використовуються команди «нормально розімкнений» та «нормально замкнений» контакти? Яка між ними різниця?

2) Для чого використовуються команди «обмотка реле», «установити», «зняти»? Яка між ними різниця? Наведіть приклади практичного використання.

3) Який буде стан сигналу на виході Q 124.5 контролера, коли в програмі спочатку виконується команда «установити», а потім «зняти» до цього виходa? Чи з'явиться на цьому виході імпульс на час між моментами виконання команд «установити» та «зняти» ?

4) Для чого використовуються команди виділення позитивного та негативного фронтів сигналу? Який їх інтерфейс? Наведіть приклади практичного використання.

5) Як реалізувати логічні операції «ТА», «АБО», «Виключне АБО» мовами LAD та STL? Яка різниця в цих мовах при реалізації даних логічних операцій?

6) Як реалізувати складні логічні вирази за допомогою команд A, O, X мови STL?

Контрольні завдання

Задача 3.1. Для керування світлом в коридорі на його початку та в кінці встановлено два перемикача з двома станами сигналу на виході – логічна «1» та «0». Сигнали з них подаються на входи I0.0 та I0.1. Якщо перемикачи знаходяться в однакових положеннях, то світло повинно бути вимкнено, коли в різ-

них – включено. Керування світлом здійснюється логічним сигналом з виходу Q0.0 контролера. При подачі на вихід логічної «1» світло вмикається. Програму реалізувати мовою LAD. Тип контролера та необхідні модулі обрати самостійно.

Задача 3.2. Контролер повинен керувати вмиканням та вимиканням трьохфазного контактора з свого виходу Q0.0. Вмикати контактор можна, коли натискають кнопку Start та є напруга у всіх трьох фазах. Вимикаємо контактор при натисканні кнопки Stop чи зникання напругу у однієї чи більше фазах. Кнопки Start та Stop без фіксації. Вони підключені до входів I0.0 та I0.1 відповідно. При їх натисканні на входах з'являється сигнал логічної «1». Наявність напруги у фазах контролюється датчиками з цифровим виходом, що приєднані до входів I0.2, I0.3, I0.4 контролера відповідно та видають сигнал логічної «1» при наявності напруги. Програму реалізувати мовами LAD та STL. Тип контролера та необхідні модулі обрати самостійно.

Задача 3.3. Написати програму, що вмикає світло в приміщенні при вході людини до нього та вимикає при виході завдяки розташуванню двох світлових датчиків перетинання у дверному пройомі. Датчики приєднані до входів I0.0 та I0.1 контролера. При перетинанні проміня датчика людиною на логічному виході датчика з'являється «1». Осцилограми сигналів, що відповідають входу та виходу людини зображені на рисунку 3.2

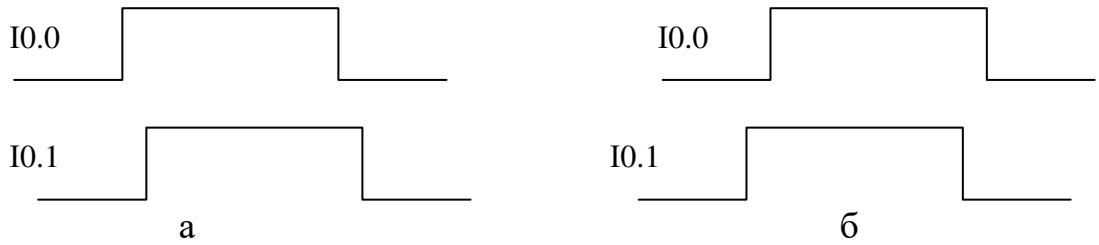


Рисунок 3.2 – Осцилограми сигналів на виходах датчиків, що відповідають: а) входу людини, б) виходу.

Керування світлом здійснюється з виходу Q0.0 контролера. Програму реалізувати мовами LAD та STL. Тип контролера та необхідні модулі обрати самостійно.

4 Вивчення інструкцій роботи з акумуляторами

В контролерах Siemens існує 4 акумулятори, перші два з яких (ACCU1, ACCU2) використовуються при будь-яких операціях в роботі з байтами, словами чи подвійними словами, а інші 2 виконують роль стеку акумуляторів. Тобто в третьому та четвертому акумуляторах можна тимчасово зберігати стан перших двох акумуляторів. Кожен з акумуляторів – це 32-х розрядний регістр. Його молодші 16 розрядів використовуються в інструкціях, призначених для обробки слів та байт, а цілком акумулятор використовується інструкціями обробки 32-х розрядних даних. Для завантаження даних у перший акумулятор використовується інструкція L, разом з якою вказується константа чи адреса пам'яті

контролера, звідки відбудеться завантаження. Ця інструкція, як і інші, існують тільки у мові STL, а у інших мовах вони є складовою частиною багатьох команд. При виконанні інструкції L дані завантажуються до першого акумулятора, те що було у ньому завантажується до другого, а те, що було у другому, – зникає. За допомогою перших двох акумуляторів реалізуються усі арифметичні операції. Зчитування байт, слів, подвійних слів із областей цифрових чи аналогових входів виконується до першого акумулятора. Нижче наведено приклади завантаження констант та даних з пам'яті

Команди	Пояснення команд
L 5	*//Завантаження 16-ти розрядної константи 5;*
L L#100	*//Завантаження 32-х розрядної константи 100;*
L IB0	*//Завантаження байту даних з області входів;*
L QW4	*//Завантаження слову даних з області виходів;*
L MD10	*//Завантаження десятого подвійного слову даних з області меркерної пам'яті;*
L PIW752	*//Завантаження коду рівню сигналу на аналоговому вході за адресою 752.*

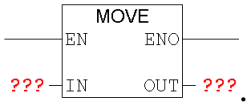
Наведемо ще приклад послідовного завантаження до акумуляторів двох констант 5 та 7

```
L  5
L  7
```

При виконанні першої інструкції (L 5) до першого акумулятора завантажується число 5. При виконанні другої команди (L 7) константа 5 переписується у другий акумулятор, а у першій акумулятор завантажується константа 7. Тобто, щоб завантажити у першій та другий акумулятор потрібні числа, їх треба завантажувати у певній послідовності. Це стає потрібно при виконанні арифметичних операцій та у багатьох інших випадках, що будуть розглянуті пізніше.

Для вивантаження даних з першого акумулятора використовується інструкція T, разом з якою вказується адреса у пам'яті куди передаються дані. При зчитуванні даних з першого акумулятора інструкцією T, на відміну від інструкції L, автоматичного копіювання даних з одного акумулятора в інший не відбувається. Тобто те що знаходилося у першому та другому акумуляторах там і залишається.

Інструкції L та T не використовуються поодинці у графічних командах мови LAD, а є їх складовою частиною. Вони обов'язково входять до складу арифметичних команд, команд порозрядної обробки, зсуву і т.і. Такі команди будуть розглянуті пізніше. В даному розділі ознайомимося лише з командою MOVE мови LAD, що є об'єднанням команди завантаження L та вивантаження T. Команда MOVE призначена для копіювання даних (типи байт, слово, подвійне слово) з одного вічка пам'яті в інше. Команда має наступний графічний



вигляд . На вході IN вказується адреса байта, слова чи подвійного слова, яке треба скопіювати. На виході OUT вказується адреса, куди треба записати дані. Вхід EN призначений для дозволу виконання операції, а на виході ENO при успішному виконанні команди з'являється логічна «1».

Для обміну вмісту перших двох акумуляторів між собою використовується інструкція ТАК. Інструкція PUSH копіює дані з 3-го акумулятора до 4-го, з 2-го до 3-го, з 1-го до 2-го. Інструкція ENT копіює дані з 3-го акумулятора до 4-го, з 2-го до 3-го. Інструкція POP копіює дані з 2-го акумулятора до 1-го, з 3-го до 2-го, з 4-го до 3-го. Інструкція LEAVE копіює дані з 3-го акумулятора до 2-го, з 4-го до 3-го.

Інструкції INC та DEC дозволяють відповідно збільшити чи зменшити вміст 1-го акумулятора на значення вказане з інструкцією. Ці інструкції дозволяють змінювати значення молодшого байту першого акумулятора в межах від 0 до 255. Наприклад, для збільшення вмісту першого акумулятора на 2 слід написати

INC 2.

Іноді наприкінці програми використовуються інструкції NOP 0 та NOP 1, що нічого не виконують, але займають час, відповідний одному такту контролера. Приклад їх використання буде розглянуто у восьмому розділі.

Контрольні питання.

1) Для чого використовуються кожен з чотирьох акумуляторів у контролерах Siemens?

2) Як працюють команди завантаження (L) та вивантаження (T) з константами та даними у пам'яті? Наведіть приклади використання.

3) Чи можливо виконання команди INC 260? Чому?

Контрольні завдання

Задача 4.1. Напишіть програму мовою STL, що копіює слово даних з області входів IW124 до маркерної пам'яті MW230. Як ця програма буде написана мовою LAD.

Задача 4.2. Напишіть мовою LAD ту ж саму програму, але при умові, що копіювання повинно виконуватися лише з появою на вході I0.0 сигналу логічної «1».

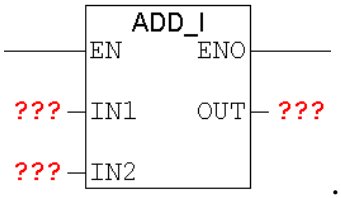
5 Вивчення арифметичних інструкцій у мовах LAD та STL

5.1 Вивчення арифметичних інструкцій у мові LAD

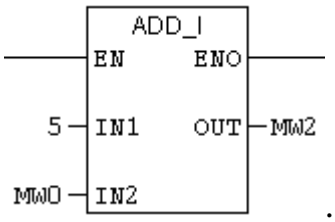
У мові LAD реалізовані інструкції для обробки 16-ти та 32-х розрядних цілих значень та 32-х розрядних речовинних значень. Перелік цих інструкцій наведено у таблиці 5.1

Таблиця 5.1 – Перелік основних арифметичних інструкцій мови LAD

Найменування інструкції	Інструкція з розрядністю даних		
	Цілі 16 біт	Цілі 32 біта	Речовинні 32 біта
додавання	ADD_I	ADD_DI	ADD_R
вирахування	SUB_I	SUB_DI	SUB_R
множення	MUL_I	MUL_DI	MUL_R
ділення	DIV_I	DIV_DI	DIV_R
залишок від ділення		MOD_DI	



Кожна з інструкцій має спільний з іншими вигляд . Зверху у зображенні інструкції вказується математична операція та тип даних, з якими вона виконується (наприклад, ADD_I). В усіх інструкціях є вхід дозволу виконання операції – EN; два входи IN1 та IN2, на які записують константи чи адреси вічків пам'яті контролера, з якими слід виконати арифметичну операцію; вихід OUT на якому міститься результат арифметичної операції та вихід ENO дозволу подальшого виконання інструкцій у рядку (network). Наприклад, інструкція додавання до даних з вічка маркерної пам'яті MW0 константи 5 має вигляд

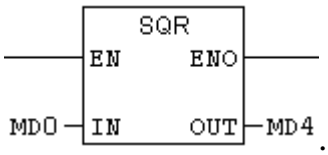


. Результат запам'ятовується у слові маркерної пам'яті MW2. Слід звернути увагу – коли в арифметичних інструкціях використовується константи типу даних REAL, то їх треба обов'язково записувати з знаками після коми (наприклад 3.5, 5.0).

Обмеженням використання арифметичних інструкцій у мові LAD є неможливість їх виконання з даними типу байт.

У контролерах Siemens також є більш складні математичні інструкції: sin, cos, tan, asin, acos, atan, exp, ln, sqrt, sqr обчислення відповідних математичних функцій sin, cos(x), tan(x), arcsin(x), arccos(x), arctan(x), e^x, ln(x), $\sqrt{x}$, x^2. Тип значення x тільки REAL. Ця група інструкцій також має спільний вигляд: вхід дозволу виконання операції – EN; вхід IN на яких подається значення операнда x; вихід OUT на якому міститься результат арифметичної операції та вихід ENO дозволу подальшого виконання інструкцій у рядку (network). Наприклад, інструкція, що обчислює значення функції x^2, коли x знаходиться у подвійному

слові MD0, а результат операції у MD4, має наступний вигляд



.

Розглянемо приклад програми, що розраховує значення виразу $y = 3175 \cdot (a + b)$, де a та b подвійні слова маркерної пам'яті, розташовані за адресами MD26 та MD30. Результат необхідно зберігати у MD34. Для написання програми будуть потрібні дві арифметичні інструкції роботи з подвійними словами: додавання та множення. Результат поточного виконання програми наведено на рисунку 5.1

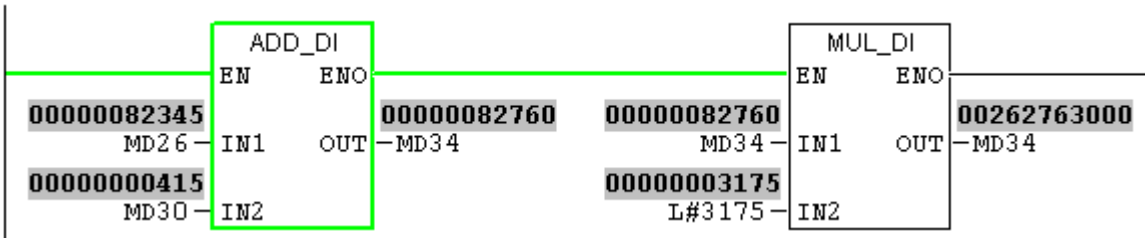
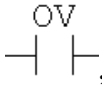


Рисунок 5.1 – Результат поточного виконання програми

При реалізації програми спочатку додаються подвійні слова MD26 та MD30. Проміжній результат зберігається у тому ж подвійному слові, що і кінцевий результат (MD34). Це дозволяє раціонально використовувати в програмі меркерну пам'ять. Константа 3175 записана у форматі подвійного слова L#3175.

При виконанні арифметичних інструкцій результат може перевищити допустимі межі. При цьому автоматично встановлюється в «1» значення біта OV у регістру статусу контролера. Обробку таких ситуацій повинен передбачити програміст. При їх обробці необхідно опитати стан біта OV. Для цього призначена команда опитування 

, що знаходиться у розділі Status bits бібліотеки графічних елементів мови LAD.

5.2 Вивчення арифметичних інструкцій у мові STL

Для реалізації арифметичних інструкцій у мові STL попередньо значення двох операндів повинні бути завантажені у перші два акумулятора, для чого використовується інструкція L. У мові STL реалізовані інструкції для обробки 16-ти та 32-х розрядних цілих значень та 32-х розрядних речовинних значень. Перелік цих інструкцій наведено у таблиці 5.2

Таблиця 5.2 – Перелік основних арифметичних інструкцій мови STL

Найменування інструкції	Інструкція з розрядністю даних		
	Цілі 16 біт	Цілі 32 біта	Речовинні 32 біта
додавання	+I	+D	+R

вирахування	-I	-D	-R
множення	*I	*D	*R
ділення	/I	/D	/R
залишок від ділення		MOD	

Слід звернути увагу, що при реалізації операцій вирахування – з вмісту другого акумулятора вираховується вміст першого, як і при діленні вміст другого акумулятора ділиться на вміст першого. Тому два числа, з якими виконується арифметична операція, повинні завантажуватися до акумуляторів у послідовності, відповідній їх розміщенню у двох акумуляторах.

Результат будь-якої арифметичної інструкції завжди знаходиться у першому акумуляторі. Другий акумулятор автоматично обнуляється.

Приклад 1 Програма, що реалізує ділення значення подвійного слова MD0 на значення подвійного слова MD4 та зберігає результат у MD8

```
L MD0
L MD4
/R
T MD8
```

Приклад 2 Програма, що обчислює значення виразу $y = 3175 \cdot (a + b)$, коли змнінні a та b розташовані у подвійних словах маркерної пам’яті за адресами MD26 та MD30

```
L MD26
L MD30
+D
L L#3175
*D
T MD34
```

Одною з переваг мови STL при виконанні арифметичних інструкцій є можливість роботи з даними типу байт. В такому випадку також використовуються інструкції +I, -I, *I, /I. Наприклад знайдемо суму двох байт з маркерної пам’яті, що розташовані за адресами MB0 та MB1. Результат збережемо у слові маркерної пам’яті MW10, тому що він може перевищувати розміри одного байту

```
L   MB0
L   MB1
+I
T   MW10
```

При виконанні арифметичних операцій, як і в мові LAD, ясніє можливість перевищення результатом допустимих меж для типу даних, що порівнюються. У такому випадку встановлюється в «1» біт OV регістра статусу контролера, контролювати котрий можна командою

```
A  OV
```

У мові STL також є більш складні математичні інструкції: sin, cos, tan, asin, acos, atan, exp, ln, sqrt, sqr обчислення відповідних математичних функцій sin(x), cos(x), tan(x), arcsin(x), arccos(x), arctan(x), e^x, ln(x), $\sqrt{x}$, x^2. Для їх використання попередньо треба завантажити значення аргумента у перший акумулятор. Наприклад

```
L 5.8
SQRT
```

Порівнюючи мови LAD та STL можна зробити висновок, що арифметичні інструкцій в обох мовах ідентичні, але їх можливості відрізняються. В мові STL існує можливість обробляти байти, що має важливе значення при реалізації внутрішніх циклів та в інших задачах. Деяка перевага мови STL у швидкості розробленої кінцевої програми існує завдяки можливості використання акумуляторів для зберігання проміжних дій, замість маркерної пам'яті та блоків даних. З цієї ж причини можна зменшити кількість використаної маркерної пам'яті. Але ці останні переваги можуть бути значущими лише для процесорів з мінімальними апаратними ресурсами.

Контрольні питання.

1) Об'ясніть призначення входів і виходів в загальному вигляді графічних команд додавання, вирахування, множення та ділення.

2) Об'ясніть, які інструкції використовуються для реалізації арифметичних операцій з цілими числами в мовах LAD та STL?

3) Об'ясніть, які інструкції використовуються для реалізації арифметичних операцій з речовинними числами в мовах LAD та STL?

Контрольні завдання

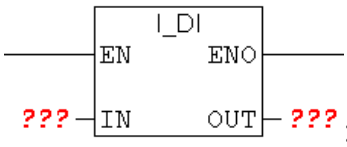
Задача 5.1. Розробити програму мовою LAD, що повинна одноразово розрахувати середнє арифметичне трьох речовинних чисел, що знаходяться у маркерній пам'яті за адресами MD24, MD28, MD32, коли на трьох логічних входах I2.0, I2.1 та I2.2 з'явиться логічна «1». Результат треба записати у подвійне слово MD100.

Задача 5.2. Написати програму, що логарифмує значення току з аналогового входу PIW754, на який подається токовий сигнал з аналогового датчику тиску. На виході датчика значення току знаходиться в межах 0...20мА, що відповідає діапазону тиску від 50 до 850 кПа. Результат слід зберігати у подвійному слові MD200.

Задача 5.3. Розробити програму, що складає два подвійних слова з маркерної пам'яті MD10, MD12. Результат слід записати у подвійне слово MD200. При перевищенні результату максимально допустимого $\pm 2^{31}$-1 слід записати логічну одиницю в біт маркерної пам'яті M35.0.

6 Вивчення інструкцій перетворення типів даних у мовах LAD та STL

6.1 Вивчення інструкцій перетворення типів даних у мові LAD

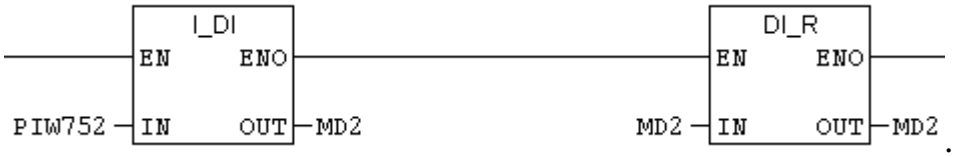
Досить часто аргументи, що входять до математичних виразів, мають різні типи даних. У мові LAD усі команди перетворення типів даних мають спільний вид 

: вхід EN дозволу виконання операції; вхід IN, на який подаємо значення, що необхідно перетворити; вихід OUT, на якому отримуємо результат перетворення та логічний вихід ENO дозволу виконання наступних інструкцій у рядку (network). Скорочення назви перетворення вказується зверху команди. Існують наступні перетворення:

- BCD_I : Перетворення BCD- коду в Integer;
- I_BCD : Перетворення Integer в BCD-код;
- BCD_DI : Перетворення BCD - коду в Double Integer;
- I_DI : Перетворення Integer в Double Integer;
- DI_BCD : Перетворення Double Integer в BCD;
- DI_R : Перетворення Double Integer в Real;
- INV_I : Інверсія числа типу Integer;
- INV_DI : Інверсія числа типу Double Integer;
- NEG_I : Додатковий код числа типу Integer (зміна арифметичного знаку);
- NEG_DI : Додатковий код числа типу Double Integer (зміна знаку);
- NEG_R : Інверсія знака числа типу Real (зміна арифметичного знаку);
- ROUND : Округлення до подвійного цілого за загальними математичними правилами (наприклад, числа 3.1, 3.5 будуть округлені до 3, а число 3.6 до 4);
- TRUNC : Виділення цілої частини (наприклад, числа 3.1, 3.5, 3.9 округляються до 3, число -0.1 округляється до 0);
- CEIL : Округлення до найближчого більшого (наприклад, числа 3.1, 3.5, 3.9 округляються до 4);
- FLOOR: Округлення до найближчого меншого (наприклад, числа 3.1, 3.5, 3.9 округляються до 3, число -0.1 округляється до -1).

Розглянемо перетворення, що є дуже поширеним в практиці програмування. Наприклад, аналоговий модуль вводу видає ціле значення коду на виході PIW752, що відповідає значенню реальної температури. Щоб отримати це речовинне значення температури треба провести обчислення складовою части-

ною яких є перетворення типу даних коду температури до речовинного значення. Частка програми, що реалізує це перетворення має вигляд



Результат опитування аналогового входу PIW752 (слово) за допомогою інструкції I_DI спочатку перетворюється у подвійне слово, що зберігається у маркерній пам'яті MD2, а потім інструкцією DI_R перетворюється у речовинне подвійне слово, що знову зберігається у маркерній пам'яті MD2.

Розглянемо ще один приклад, у якому треба знайти цілочисельний залишок від ділення слова MW34 на MW36. Значення залишку зберігається у слові MW40

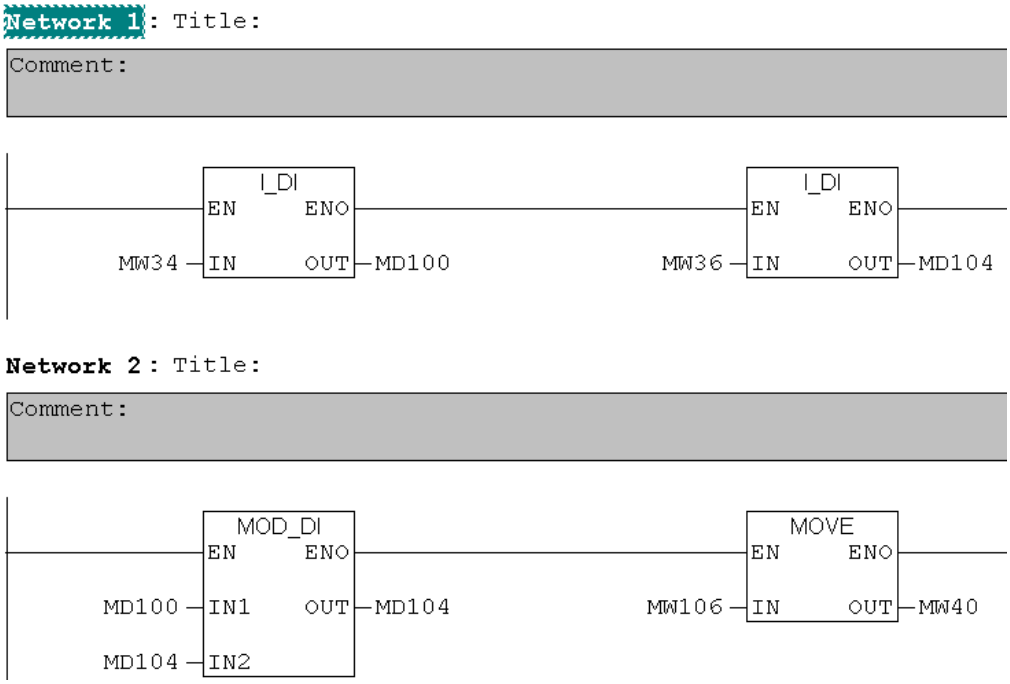


Рисунок 6.1 – Приклад програми знаходження цілочисельного залишку від ділення MW34 на MW36.

При реалізації програми, спочатку, слова MW34 на MW36 перетворюються у подвійні слова командою I_DI та зберігаються у тимчасових подвійних словах маркерної пам'яті MD100 на MD104. Така операція необхідна, бо команда знаходження залишку від ділення виконується лише з подвійними словами. Використовуючи команду MOD_DI знаходимо залишок від ділення, який зберігаємо у подвійному слові MD104. Подальшого перетворення із подвійного цілого до цілого не існує. Тому для його реалізації треба зрозуміти де знаходиться залишок. Зрозуміло, що залишок не перевищує слова, бо вихідні операнди також були словами. Тому в подвійному слові MD104 два старші байти нульові. Тобто залишок в двох молодших байтах, а саме в 107 та 106, бо послідовність байтів в словах та подвійних словах зворотня. Далі командою MOVE копіюємо залишок зі слова MW106 до слова MW40.

6.2 Вивчення інструкцій перетворення форматів у мові STL

У мові STL число, формат якого треба перетворити, потрібно завантажити до першого акумулятора. Далі використати одну з інструкцій перетворення форматів. Результат перетворення буде знаходитися у першому акумуляторі. До інструкцій перетворення форматів належать наступні:

• BTI Перетворення числа в BCD- коді в ціле число (16-біт);
• ITB Перетворення цілого числа (16-біт) в BCD- код;
• BTD Перетворення числа в BCD- коді в подвійне ціле число (32-біта);
• ITD Перетворення цілого числа (16-битий) у подвійне ціле число (32-біта);
• DTB Перетворення подвійного цілого числа (32-біта) в BCD- код;
• DTR Перетворення подвійного цілого числа (32-біта) у число із плаваючою крапкою (32-біта по IEEE-FP);
• INVI Інверсія біт числа типу Integer (16-біт);
• INVD Інверсія біт числа типу Double Integer (32-біта);
• NEGI Інверсія знака числа типу Integer (16- біт);
• NEGD Інверсія знака числа типу Double Integer (32-біта);
• NEGR Інверсія знака числа із плаваючою крапкою (32-біта, IEEE-FP);
• CAW Зміна порядку байт у молодшому слові ACCU 1-L (16- біт);
• CAD Зміна порядку байт в ACCU 1 (32-біта);
• RND Округлення до подвійного цілого;
• TRUNC Виділення цілої частини;
• RND+ Округлення до найближчого більшого;
• RND- Округлення до найближчого меншого.

Приклад попередньої програми перетворення даних типу Integer у Real, що була наведена у розділі 6.1, буде мати наступний вигляд

```
L PIW 752
ITD
DTR
T MD2
```

Спочатку завантажують до першого акумулятора дані з аналогового входу PIW 752. Потім, ціле 16-ти розрядне значення перетворюють до подвійного цілого інструкцією ITD. Розуміючи, що результат цього перетворення знаходиться у першому акумуляторі виконують друге перетворення з подвійного цілого до речовинного інструкцією DTR. Результат передається у меркерну пам’ять за адресою MD2.

Другий приклад попередньої програми знаходження залишку від ділення має вигляд

```
L MW34 //Завантаження значення з MW34
ITD //Перетворення завантаженого значення до подвійного цілого
```

L MW36 //Завантаження значення з MW36
ITD //Перетворення завантаженого значення до подвійного цілого
MOD //Знаходження залишку від ділення
T MW40 //Збереження результату

На відміну від програми мовою LAD після завантаження слова MW34 та перетворення його формату на подвійне ціле немає необхідності зберігати проміжний результат, бо наступна інструкція завантаження L цей результат передає до другого акумулятора, де він і зберігається. Маючи перший результат перетворення у другому акумуляторі, а другий результат у першому виконуємо інструкцію MOD, знаходячи залишок. Молодші 16 біт цього залишка з першого акумулятора автоматично копіюються до MW40 командою T MW40.

Приклад перетворення цілого (16 біт) до двійково-десяткового формату

L MW0
ITB
T MW20

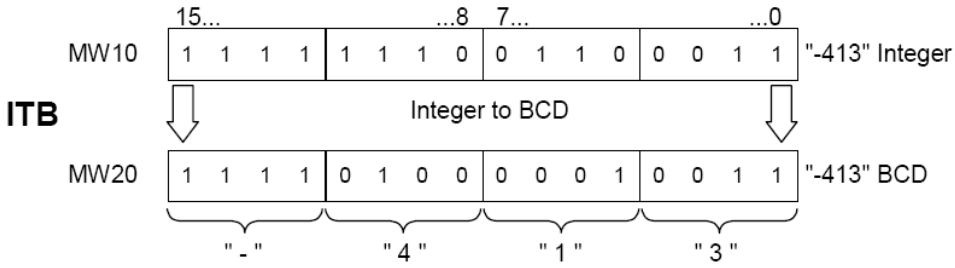


Рисунок 6.2 – Відповідність між десятковим та двійково-десятковим форматами чисел

Порівняльний аналіз мов LAD та STL при реалізації перетворень форматів даних дозволяє зробити висновок, що в цілому мови ідентичні як за бистродією так і за використаними ресурсами процесора. Переваги мови STL незначні. Основною відзнакою мови STL є наявність інструкцій CAW та CAD для зміни порядку байт в словах та подвійних словах, що іноді можуть бути потрібні при роботі з SCADA системою.

Контрольні питання

1) Для чого необхідні інструкції перетворення форматів у мовах LAD та STL?

2) Як буде записане число 381 у двійково-десятковому форматі?

3) Яку послідовність дій (команд) треба виконати, щоб перемножити слово даних з маркерної пам’яті на речовинну константу 17.8?

Контрольні завдання

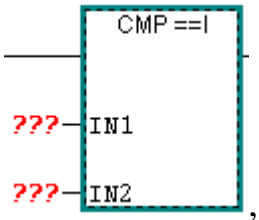
Задача 6.1. Написати програму, що зчитує дані з аналогового входу PIW752, перемножує отриманий результат на 0.75 та округлює результат до подвійного цілого.

Задача 6.2. Написати програму знаходження середнього арифметичного трьох слів у пам’яті MW30, MW34, MW38. Результат у речовинному форматі слід зберегти у подвійному слові маркерної пам’яті MD60.

7 Вивчення інструкцій порівняння у мовах LAD та STL

7.1 Вивчення інструкцій порівняння у мові LAD

У мові LAD існують команди, що дозволяють порівнювати два значення одного з типів даних Integer, Double Integer та Real. Усі команди порівняння

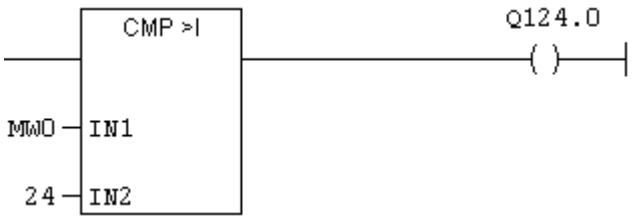


мають спільний вид , а саме два входа In1 та In2 на яких порівнюються два числа. Ці числа можуть бути константами чи зчитуватися з пам’яті контролера. Також є вхід дозволу виконання команди та логічний вихід, на якому логічний 0, коли умова порівняння не виконана та логічна 1 в іншому випадку. У верхній частині графічного зображення команд порівняння вказана умова порівняння та тип даних, що порівнюються. Існують наступні умови порівняння:

== тотожно рівно;
<> не рівно;
> більше;
< менше;
>= більше або рівно;
<= менше або рівно.

Порівнюються дані трьох типів: Integer (умовне позначення – I), Double Integer (D) та Real (R).

Приклад програми, що перевіряє, чи більше значення з маркерної пам’яті MW20 від константи 24 і керує виходом Q124.0



В наведеній програмі, якщо число з MW0 більше 24, то на вихід Q124.0 подається логічна «1».

Обмеженням мови LAD є неможливість порівняння даних типу байт.

7.2 Вивчення інструкцій порівняння у мові STL

В мові STL для порівняння двох чисел їх спочатку треба завантажити в перші 2 акумулятора контролера, а потім порівняти за допомогою команд:

порівняння слів – >I, >=I, <I, <=I, ==I, <>I;

порівняння подвійних слів >D, <D, >=D, <=D, ==D, <>D;

порівняння даних типа real >R, <R, >=R, <=R, ==R, <>R.

При виконанні будь-якої інструкції порівняння змінюється значення біта RLO, тобто при виконанні умови порівняння значення біта RLO встановлюється в «1».

При виконанні операцій >, < порівнюється значення, що знаходиться у другому акумуляторі з числом, що знаходиться у першому акумулятора. Тобто для виконання операції порівняння числа, що порівнюємо, треба завантажувати до двох акумуляторів у певній послідовності. Розглянемо приклад порівняння значень двох слів з маркерної пам'яті MW0, MW2 та результати виконання програми.

			RLO	STA	STANDARD
L	MW	0	0	1	5
L	MW	2	0	1	2
>I			1	1	2

У наведеній програмі спочатку до першого акумулятора завантажується число 5 з MW0. При завантаженні числа 2 з MW2 число 5 передається до другого акумулятора, а число 2 записується до першого акумулятора. Результатом порівняння є значення біта RLO=1.

Умовні позначення порівнянь відповідають наведеним у розділі 7.1. На відміну від мови LAD можна завантажити до акумуляторів та порівняти два байти.

Наведемо приклад програми, що зчитує подвійне слово даних MD20 з маркерної пам'яті та порівнює його значення з константою 25.0 на відповідність. У разі відповідності на вихід Q 124.0 подається логічна одиниця

```
L MD20
L 25.0
==R
= Q124.0
```

Використання мов LAD та STL при реалізації порівнянь цілком ідентичне бо перелік інструкцій та їх можливості в обох мовах цілком співпадають.

Контрольні питання.

1) Об'ясніть призначення входів та виходів графічного зображення команд порівняння.

2) Дані яких типів можна порівнювати у мовах LAD та STL? Чи можна порівнювати дані різних типів? Як це зробити?

3) Чи залежить у мові STL результат порівняння від розташування двох значень, що порівнюють, у двох акумуляторах?

Контрольні завдання

Задача 7.1. Розробіть програму порівняння амплітуди вхідного сигналу у діапазоні від 0В до 10 В на аналоговому вході PIW752 з пороговим значенням 4.8В. При перевищенні порога на цифровий вихід Q 0.0 треба подати логічну «1». При реалізації проекту слід самостійно обрати необхідний контролер та модулі вводу і виводу.

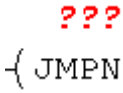
Задача 7.2. Напишіть програму мовою STL, що в кожному скані роботи контролера розраховує послідовно значення одного з 64 відліків періоду синусоїдального сигналу при амплітуді сигналу 200. Розраховане значення треба завантажувати за адресою PQW754.

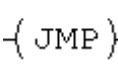
8 Вивчення інструкцій умовних та безумовних переходів у мовах LAD та STL

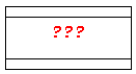
8.1 Вивчення інструкцій умовних та безумовних переходів у мові LAD

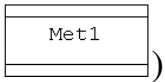
Програмні розгалуження є невід'ємною складовою будь-якої програми. В мовах LAD та STL розгалуження реалізують як сукупність команди, що дозволяє перестрибнути з деякого місця програми на інше, яке позначене міткою.

Для реалізації програмних розгалужень у мові LAD використовуються команди перестрибування

та 

, які записуються у кінці рядка Network. Разом з командою вказується ім'я мітки, на яку перестрибують (Наприклад, 

). Команда JMP дозволяє переходити на мітку коли значення біту RLO дорівнює 1, а команда JMPN – коли RLO дорівнює 0. Щоб поставити у програмі мітку використовується команда 

(Lable). Ім'я мітки може складатися з літер та цифр. Довжина імені не може біти більше чотирьох символів (

).

Якщо до команд JMP та JMPN у рядку програми немає інших команд, то такі переходи є безумовними, а коли до них знаходяться команди, що змінюють стан біту RLO, то такі переходи є умовними. До таких команд можуть відноситися: опитування біт, порівняння, лічильників, таймерів і т.і.

Наведемо приклад програми, у якій відповідно до результатів порівняння слова MW30 у маркерній пам'яті із константою 30 здійснюється розгалуження з

використанням команди JMP. Коли значення, що знаходиться у MW30 менше ніж 30, то переходимо на мітку m1 (початок рядка Network3) і пропускаємо команду завантаження до MW30 числа 0. У іншому випадку завантажуємо до MW30 число 0 та переходимо до виконання команд з рядку Network3.

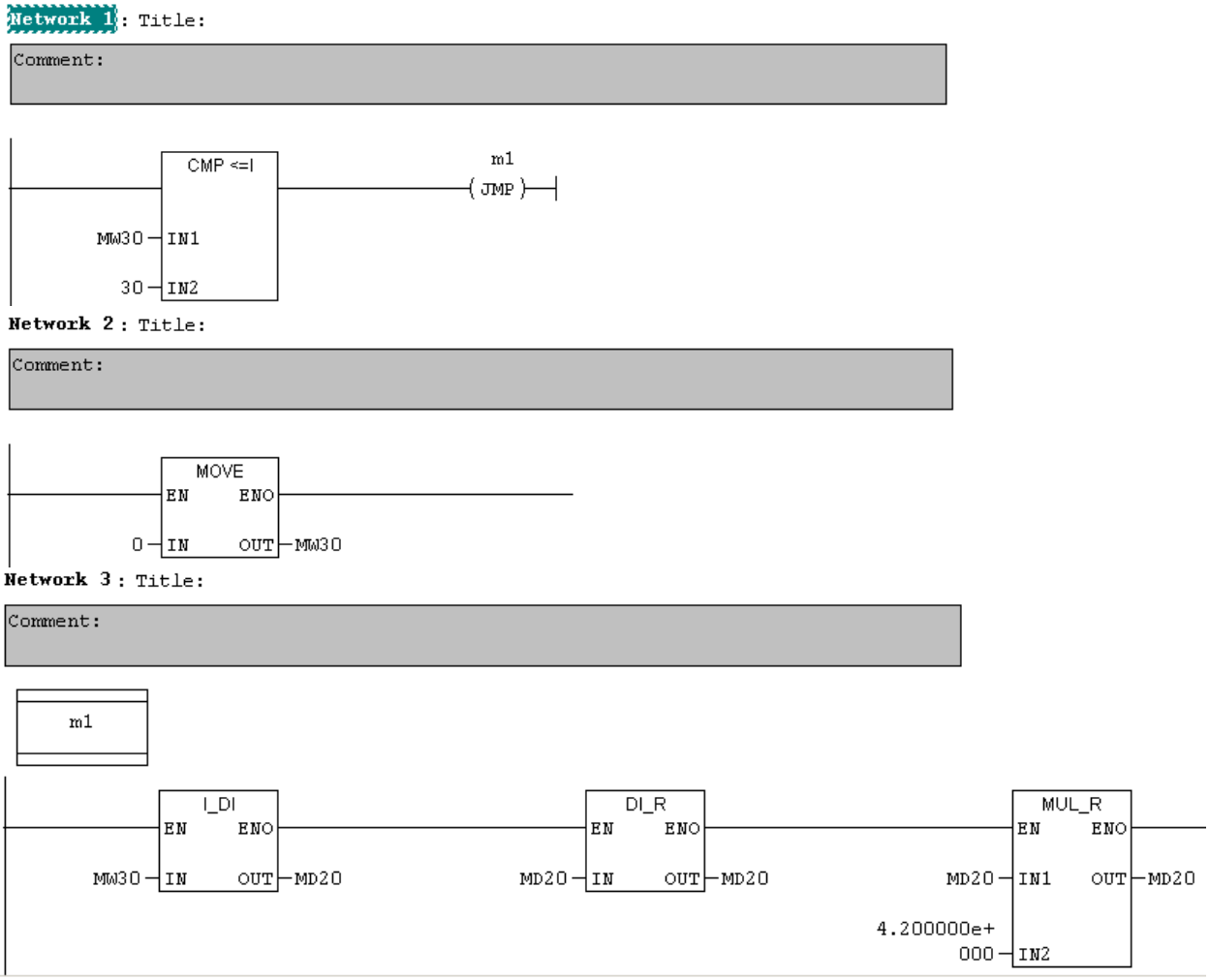


Рисунок 8.1 – Приклад програми реалізації розгалуження

8.2 Вивчення інструкцій умовних та безумовних переходів у мові STL

Програмні розгалуження у мові STL бувають умовними, безумовними та розподіленими. Нижче наведено перелік команд, що здійснюють перехід на мітку відповідно до виконання умови:

JC –перехід при RLO = 1;

JCN –перехід при RLO = 0;

JCB –перехід при RLO = 1 зі збереженням значення RLO в біт BR. Біт BR знаходиться у регістрі статуса контролера, але контролер їм автоматично не керує. Цей біт призначено для довільного використання при програмуванні;

JNB –перехід при RLO = 0 зі збереженням значення RLO в біт BR;

JBI –перехід при значенні біта BR = 1;

JNBI –перехід при значенні біта BR = 0;

JO –перехід при значенні біта OV = 1;

JOS –перехід при значенні біта OS = 1;
JZ –перехід при нульовому результаті математичної операції;
JN –перехід при ненульовому результаті математичної операції;
JP –перехід при позитивному результаті математичної операції;
JM –перехід при негативному результаті математичної операції;
JPZ –перехід при ненегативному результаті математичної операції;
JMZ –перехід при негативному або нульовому результаті математичної операції;
JUO –перехід при недійсному результаті математичної операції.

Команда безумовного переходу лише одна – JU. При її виконанні обов'язково переходимо на вказану мітку

Команда розподіленого переходу JL використовується у сукупності з командами безумовного та умовного переходів. Для її використання попередньо у перший акумулятор треба завантажити цілочисельне значення в межах від 0 до 255, тобто байт. Далі записується команда JL з зазначенням імені мітки, куди треба перейти, якщо завантажене раніше число у перший акумулятор не відповідає одному з перелічених нижче. Нижче записують команди безумовного переходу з зазначенням міток, на які треба перейти, коли завантажене у перший акумулятор число дорівнює 0, 1, 2 ...

Мітка – це сполучення не більше ніж чотирьох літер та цифр, яке обов'язково починається з літери. Після мітки ставиться двокрапка та має бути команда.

Приклад програми №1. Треба порівняти слово MW24 з константою 370. Якщо це слово менш ніж константа, то обчислюють MW30=MW24*5-14, а у іншому випадку MW30=MW24*2-138

```
    L MW 24 //Завантаження до акумуляторів слова MW24 та
    L 370 // константи 370
    <I // Порівняння слова MW24 з константою 370
    JC m1 // Перехід на мітку m1 якщо слово MW24 менш ніж константа
    L MW24 //Завантаження до акумулятора слова MW24
    L 2 //Завантаження до акумулятора константи 2
    *I //Множення слова MW24 та константи 2
    L 138 //Завантаження константи 138
    -I //Вирахування константи 138 з отриманого перед тим результата
    JU m2 //Безумовний перехід на мітку m2 (у кінець програми)
m1: L MW24 // Завантаження до акумулятора слова MW24
    L 5 // Завантаження до акумулятора константи 5
    *I // Множення слова MW24 та константи 5
    L 14 // Завантаження константи 14
    -I // Вирахування константи 14 з отриманого перед тим результата
m2: T MW30 //кінець програми та збереження результату обчислення
```

Приклад програми №2. Реалізація переходу, якщо результат арифметичної операції вирахування з MW36 константи 54 менше ніж 0

```
        L MW36
        L 54
        -I
        JM Met1
        ...
    Met1: ...
```

Приклад програми №3. Реалізація розподіленого переходу

```
    L MB0 //Завантаження номера перехода до першого акумулятора
    JL LSTX //Перехід на мітку LSTX, якщо завантажене значення більш ніж
3
    JU SEG0 // Перехід на мітку SEG0, якщо з MB0 завантажено 0
    JU SEG1 // Перехід на мітку SEG1, якщо з MB0 завантажено 1
    JU COMM // Перехід на мітку SEG0, якщо з MB0 завантажено 2
    JU SEG3 // Перехід на мітку SEG0, якщо з MB0 завантажено 3
    LSTX: JU COMM
SEG0: ... //Деяка частина програми
    ...
    JU COMM //Перехід у кінець програми
SEG1: ... // Деяка частина програми
    ...
    JU COMM
SEG3: ... // Деяка частина програми
...
    JU COMM
COMM: ...
```

Знаком ... умовно позначено деякий блок команд.

Порівнюючи мови LAD та STL при реалізації переходів можна зробити висновок, що мова STL має значно розширені можливості та гнучкість. Це досягається завдяки великій кількості команд переходів, які використовують результати арифметичних операцій, а також завдяки розподіленим переходам. Тому цією мовою рекомендовано розробляти програми із складними розподіленими переходами. Не зважаючи на це мовою LAD також можна реалізувати будь-які переходи, хоча розроблена програма буди значно більше за об’ємом ніж розроблена мовою STL.

Контрольні питання.

1) За допомогою яких команд у мові LAD можна реалізувати умовні та безумовні переходи?

2) Які інструкції у мові STL використовують значення біта RLO для прийняття рішення о переході на метку?

3) Які інструкції у мові STL використовують результати арифметичних інструкцій при прийнятті рішення о переході на метку?

4) Як реалізувати розподілений перехід?

Контрольні завдання

Задача 8.1. Розробити програму, що опитує три аварійні датчики з цифровим виходом. При відсутності на виходах датчиків логічної «1» слід перейти на мітку з ім’ям m_1 у кінці програми. У іншому випадку слід видати на вихід Q0.0 логічну одиницю. Програму реалізувати мовою STL.

Задача 8.2. Розробити програму, що знаходить максимальне з чисел, які знаходяться у словах MW20, MW22, MW24 меркерної пам’яті, та записує максимальне з них у слово MW50.

9 Вивчення інструкцій логічного та циклічного зсувів у мовах LAD та STL

9.1 Вивчення інструкцій логічного зсуву у мові LAD

Загальний вигляд інструкцій логічного і циклічного зсувів наведено на рисунку 9.1(а,б)

а б

Рисунок 9.1 – Загальний вигляд інструкцій логічного та циклічного зсувів

За допомогою інструкцій зсуву Ви можете побітно зсунути вміст входу IN вліво або вправо. Зсув на n біт вліво відповідає помноженню вмісту входу IN на 2n; зсув на n біт вправо ділить цілочисельно вміст входу IN на 2n. Наприклад, якщо Ви зсуваєте значення 3 у двійковому коді на 3 біти вліво, то отримуєте десяткове значення 24. Якщо Ви зрушуєте десяткове значення 16 на 2 біти вправо, то отримуєте двійковий еквівалент десяткового значення 4. Параметр N у графічному зображенні команди задає, на скільки біт повинен виконуватися зсув. Його значення може бути змінною чи константою. Якщо це значення константа, то її треба записувати у шістнадцятирічному форматі. Результат зсуву зчитується з виходу OUT. Операція зсуву виконується при поданні на вхід EN логічної одиниці.

При обробці слів та подвійних слів існує можливість логічного та циклічного зсуву на будь-яку кількість двійкових розрядів в межах довжини слова чи

подвійного слова. Схематично логічний зсув числа без знаку може бути пояснено за допомогою рисунка 9.2

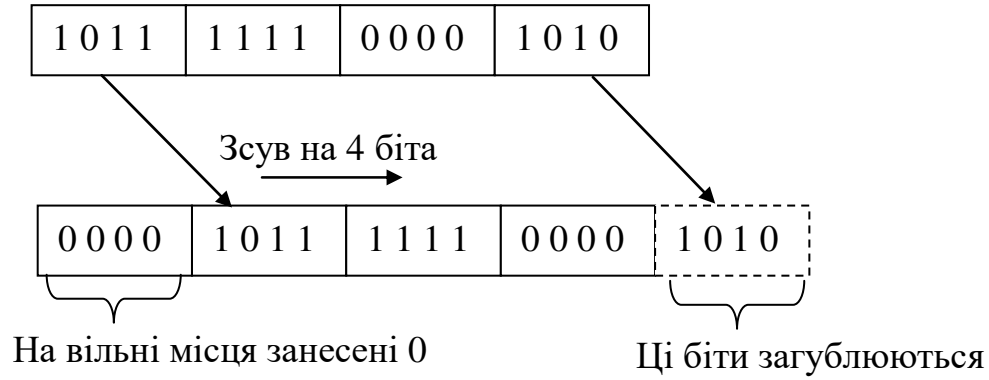


Рисунок 9.2 – Умовне зображення операції логічного зсуву

На відміну від приклада на рисунку 9.2, коли виконується логічний зсув вправо числа із знаком, то вільні місця заповнюються значенням знакового біта. У мові LAD існують наступні графічні блоки логічного зсуву:

• SHR_I : Зсув вправо числа типу Integer
• SHR_DI : Зсув вправо числа типу Double Integer
• SHL_W : Зсув слова вліво
• SHR_W : Зсув слова вправо
• SHL_DW : Зсув подвійного слова вліво
• SHR_DW : Зсув подвійного слова вправо

Циклічний зсув може бути пояснено за допомогою рисунка 9.3

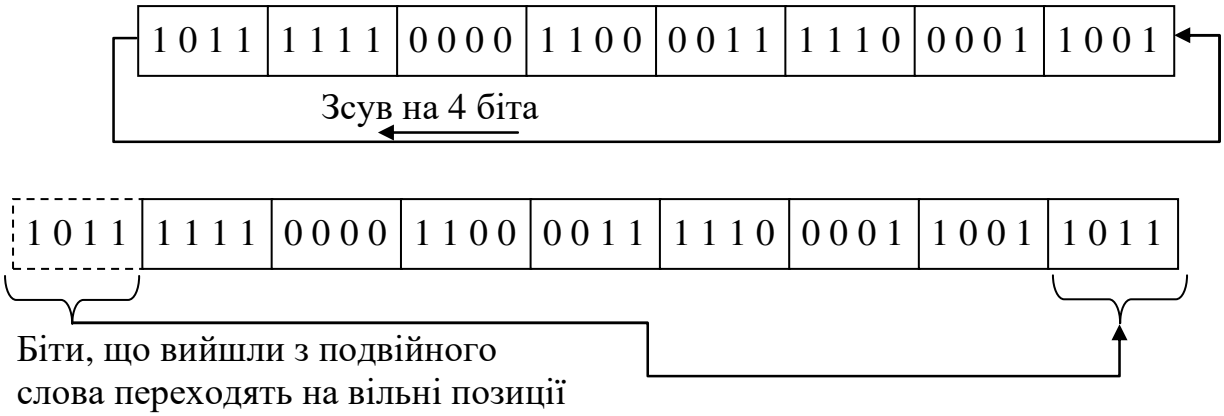


Рисунок 9.3 – Умовне зображення операції циклічного зсуву

Циклічний зсув виконується тільки з подвійними словами У мові LAD існують наступні графічні блоки логічного зсуву:

• ROL_DW : Циклічний зсув подвійного слова вліво (рисунок 9.3)
• ROR_DW : Циклічний зсув подвійного слова вправо

Приклад програми з використанням інструкції зсуву

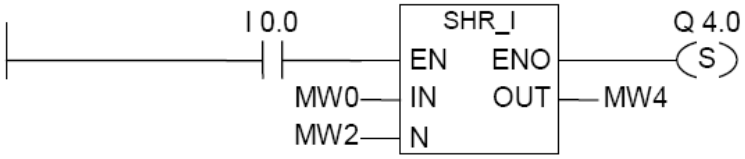


Рисунок 9.4 – Приклад програми з використанням інструкції зсуву

Зсув значення слова маркерної пам'яті MW0 виконується при появі на вході I0.0 логічної одиниці. Число розрядів, на яке виконується зсув знаходиться у слові MW2, результат зсуву записується у слово MW4. Якщо зсув виконано коректно, то на виході Q4.0 з'являється логічна одиниця.

9.2 Вивчення інструкцій логічного зсуву у мові STL

У мові STL інструкції логічних та циклічних зсувів виконуються за загальними правилами (рисунок 9.2 та рисунок 9.3). Для виконання зсуву треба завантажити значення, що треба зсунути, до першого акумулятора. Потім виконується відповідна інструкція зсуву. Результат знаходиться у першому акумуляторі. Існують наступні інструкції логічного та циклічного зсуву

- SSI Зсув вправо числа типу Integer
- SSD Зсув вправо числа типу Double Integer
- SLW Зсув слова вліво
- SRW Зсув слова вправо
- SLD Зсув подвійного слова вліво
- SRD Зсув подвійного слова вправо
- RLD Циклічний зсув подвійного слова вліво (32-біта)
- RRD Циклічний зсув подвійного слова вправо (32-біта)
- RLDA Циклічний зсув ACCU1 вліво через біт CC1 (32-біта)
- RRDA Циклічний зсув ACCU1 вправо через біт CC1 (32-біта)

За кожною інструкцією вказується на скільки розрядів треба виконати зсув.

Приклад

L MW4 //*Завантаження значення з MW4 до першого акумулятора*
SLW 5 //*Зсув біт у першому акумуляторі на п'ять розрядів вліво*.
T MW8 //*Збереження результату в MW8.*

	ACCU1			
№ біта	15...	.	..	…0
Перед виконанням інструкції SLW 5	0101	1111	0110	0100
Після виконання інструкції SLW 5	1110	1100	1000	0000

На відміну від мови LAD з’явились дві нові інструкції RLDA та RRDA, що виконують циклічний зсув подвійного слова через біт CC1 на один розряд вліво чи вправо відповідно. При зсуві подвійного слова вліво старший розряд попадає у біт CC1, а те, що було у ньому, передається у молодший біт цього подвійного слова. Інструкції RLDA та RRDA виконують зсув тільки на 1 розряд. За винятком інструкції RLDA та RRDA можливості мов LAD та STL при реалізації зсувів не відрізняються.

Контрольні питання

1) Які інструкції використовуються в мовах LAD та STL для виконання логічних зсувів?

2) Які інструкції використовуються в мовах LAD та STL для виконання циклічних зсувів?

3) У чому різниця при виконанні команд логічного та циклічного зсувів?

Контрольні завдання

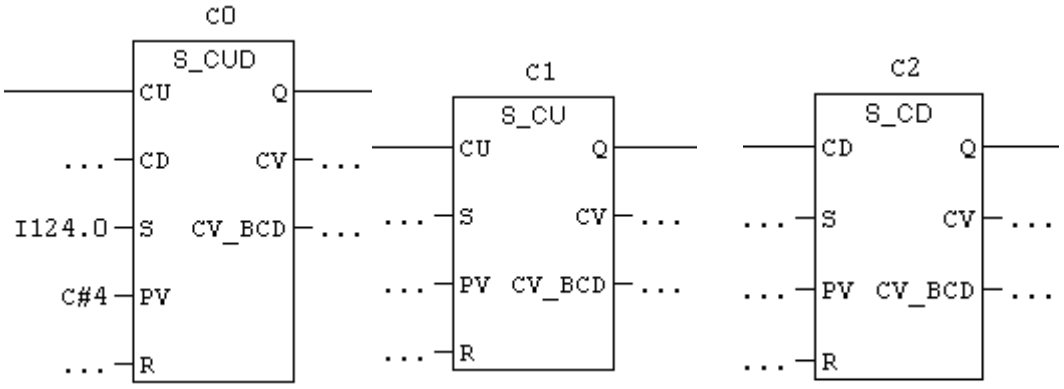
Задача 9.1. Розробити програму, що у кожному скані роботи виконує циклічний зсув подвійного слова MD10 та копіює значення нульового біта на вихід Q4.1.

Задача 9.2. Розробити програму, що виконує логічний зсув слова MW10 на один розряд кожного разу, коли натискаємо кнопку Start, що приєднана до входу I0.0. На початку та через кожні 16 зсувів у слово MW10 потрібно завантажувати константу 14821.

10 Вивчення інструкцій лічильників у мовах LAD та STL

10.1 Вивчення інструкцій лічильників у мові LAD

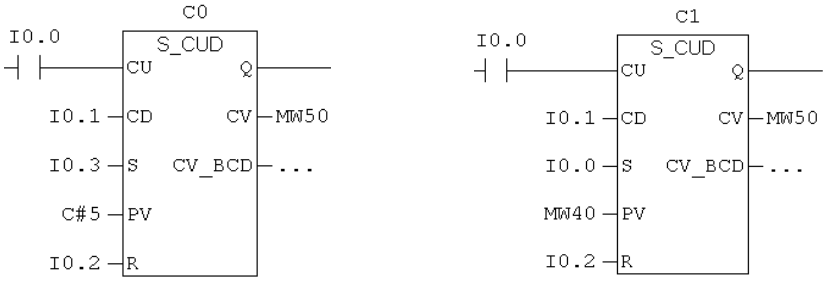
Лічильники – це програмні модулі, що дозволяють рахувати зміну якогонебудь біта в областях пам’яті входів, виходів, маркерної пам’яті і т.і. Лічильники мають область, зарезервовану для них у пам'яті CPU. Ця область пам'яті резервує по одному 16-бітному слову для кожної адреси лічильника. У ПЛК Siemens можливо використовувати в програмі до 128…512 лічильників залежно від їх модифікацій. Нумерація лічильників у програмі починається з 0. Кожен з лічильників має назву C0…C511 та рахує від 0 до 999 або навпаки. Приклади графічного зображення лічильників у мові LAD наведені на рисунку 10.1



а б в

Рисунок 10.1 – Лічильники, що рахують у обох напрямках (S_CUD), на збільшення (S_CU), на зменшення (S_CD)

На рисунку 10.1 а зображено універсальний лічильник S_CUD, що рахує в обох напрямках – на збільшення та на зменшення. Зверху записана назва лічильника – C0. Коли на вході CU логічний сигнал змінюється з «0» на «1» до поточного значення лічильника додається 1. Це поточне значення видається на вихід CV у форматі Integer та на вихід CV_BCD у двійково-десятковому форматі BCD. Коли поточне значення більше 0, на логічному виході Q логічна одиниця. Якщо поточне значення досягло 999, а на вході CU логічний сигнал знов змінюється з «0» на «1», то поточне значення залишається рівним 999. Це означає, що лічильник не переповнюється та не скидається автоматично до нуля. Коли на вході CD логічний сигнал змінюється з 0 на 1 від поточного значення лічильника вираховується 1. Якщо поточне значення досягло 0, а на вході CD логічний сигнал знов змінюється з «0» на «1», то поточне значення залишається рівним 0. Необов'язково, щоб лічильник рахував з 0. Його можна предвстановити у завдане значення. Для цього треба подати це значення на вхід PV і змінити логічний сигнал на вході S з «0» в «1». Значення на вході PV може бути константою та змінною величиною. При завданні константи формат запису має вигляд C#число (рисунок 10.1 а). Якщо значення на вході PV є змінною величиною, то воно повинно зберігатися у слові в двійково-десятковому форматі. Щоб негайно скинути поточне значення лічильника у 0 треба змінити логічний сигнал на вході R з «0» в «1». На рисунку 10.2 наведено приклади лічильників з повним керуванням.



а б

Рисунок 10.2 – Приклади лічильників

На обох лічильниках (С0 та С1) збільшення та зменшення поточного значення відбувається при змінах логічних сигналів I0.0 та I0.1 відповідно на входах CU та CD. Скидання лічильників відбувається зміною логічного сигналу I0.2, що подається на вхід R. Сигнал із входу I0.3 подається на вхід S лічильника та його зміна з «0» в «1» предвстановлює лічильник у початкове значення. На рисунку 10.2 а початкове значення – це константа 5 (C#5), а на рисунку 10.2 б, початкове значення зчитується з слова маркерної пам'яті MW40. Результат рахування зберігається у слові маркерної пам'яті MW50.

Два інших різновидів лічильників S_CU та S_CD (рисунок 10.2) можуть рахувати тільки на збільшення та зменшення відповідно. Тому у першому з них відсутній вхід CD, а у другому – вхід CU. У іншому вони повністю ідентичні лічильнику S_CUD.

Існують також прості інструкції роботи з лічильниками: 1) завантаження поточного значення $\underset{\text{поточне значення}}{(SC)}^{\text{N лічильника}}$; 2) збільшення значення лічильника $(CU)^{\text{N лічильника}}$; зменшення значення лічильника $(CD)^{\text{N лічильника}}$. Усі ці інструкції використовуються тільки вкінці рядка програми (network). Кожна з інструкцій виконується при зміні логічного сигналу на їх вході з «0» в «1».

Приклад програми, що рахує імпульси з входу I 124.0. При досягненні поточного значення в лічильнику рівного 20 лічильник автоматично скидається.

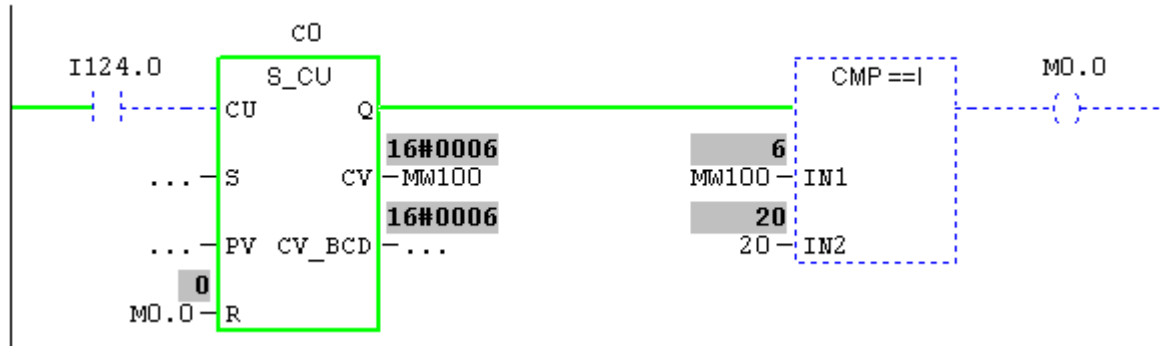


Рисунок 10.3 – Графічний вид програми з використанням лічильника

У цій програмі спочатку опитуємо вхід I 124.0 і подаємо з нього сигнал на вхід CU лічильника С0. Поточне значення лічильника зберігається у слові меркерної пам'яті MW100. Далі значення слова MW100 порівнюється у компараторі CMP==I з константою 20. Стан логічного виходa компаратора передається у біт M0.0. Поки поточне значення лічильника менш ніж 20 на виході компаратора логічний 0, а коли поточне значення стає рівним 20 на виході з'являється 1. Коли біт M0.0 стає рівним 1 лічильник С0 скидається у 0, бо значення цього біта подається на вхід R лічильника. Але скидання лічильника здійснюється тільки при наступному виконанні програми (скану), бо встановлення біта M0.0 в «1» здійснюється у кінці програми, а скидання лічильника С0 бітом M0.0 виконується на початку програми.

10.2 Вивчення інструкцій лічильників у мові STL

У мові STL лічильники також можуть рахувати на збільшення та на зменшення. Для цього використовуються інструкції CU та CD. Формат запису цих інструкцій CU №_лічильника, CD №_лічильника.

Наприклад

CU C5 //*Команда зміни значення п'ятого лічильника*

Перед використанням цих інструкцій у попередньому рядку програми повинен опитуватися біт пам'яті контролера, зміни якого з 0 в 1 будуть рахуватися лічильником.

Наприклад

A I 0.0 //*Опитування біта I 0.0*
CU C5 //*Збільшення поточного стану лічильника при зміні біта I 0.0 з «0» в «1»*

Для завантаження до лічильника деякого поточного значення це значення потрібно, спочатку, завантажити до акумулятора, а далі використати інструкцію S №_лічильника.

Наприклад

L C#8 //*Завантаження до акумулятора поточного значення 8*
S C5 //*Завантаження цього значення до лічильника C5*

Для скидання лічильника необхідно опитати біт пам'яті, зміна якого з «0» в «1» скине лічильник та використати інструкцію R №_лічильника. Наприклад

A M 1.1 //*Опитування біта M1.1*
R C4 //*Скидання лічильника C4 при зміні біта M 1.1 з 0 в 1*

При зчитуванні поточного значення лічильника це значення треба завантажити до акумулятора, а потім у необхідне слово пам'яті

L C3 //*Зчитування поточного стану лічильника C3*
T MW120 //*запис цього значення у слово MW120*

Порівняльний аналіз мов LAD та STL при використанні лічильників дозволяє зробити висновок, що можливості мов цілком співпадають.

Контрольні питання

1) Які типи лічильників існують та у чому їх різниця?

2) У яких межах рахує лічильник? Чи скидається він до нуля при переповненні?

3) Скільки байт займає лічильник у пам’яті контролера та скільки байт займає його поточне значення?

4) Поясніть призначення входів та виходів лічильника S_CUD мови LAD?

5) Як і у якому форматі завантажити до лічильника початкове значення у мовах LAD та STL?

6) Які дії потрібно виконати перед використанням інструкцій лічильників?

5) Як зчитати поточне значення лічильника у мові STL?

Контрольні завдання

Задача 10.1. Написати програму, що з використанням лічильників рахує імпульси на логічному вході I124.0, зберігає нараховане значення у 150 слові маркерної пам’яті та автоматично скидає лічильник при досягненні порогу у 87 імпульсів. Програму розробити мовою STL.

Задача 10.2. Розробити програму, що з використанням лічильників рахує імпульси на вході I1.0 та зберігає поточне значення у слові маркерної пам’яті за адресою MW56. Максимальне значення кількості імпульсів 3000

Задача 10.3. Розробити програму, відповідну задачі №3 з розділу 3, при умові що у кімнату може увійти більше, ніж одна людина

Задача 10.4. Розробити програму з використанням лічильника, що виконує логічний зсув слова MW10 на один розряд кожного разу, коли натискаємо кнопку Start, що приєднана до входу I0.0. На початку та через кожні 16 зсувів у слово MW10 потрібно завантажувати константу 14821.

11 Вивчення інструкцій таймерів у мовах LAD та STL

11.1 Вивчення інструкцій таймерів у мові LAD

Таймер – це програмний модуль, що дозволяє відміряти часові інтервали протягом від 10 мс до 2 годин 46 хвилин 30 секунд. Таймер ідентичний лічильнику бо рахує від 0 до 999, але замість чисельного рахування здійснює часове рахування у реальному часі. Тобто таймер рахує часові інтервали по 0.01, 0.1, 1 або 10 с, що мають назву – база часу. Наприклад, коли база часу дорівнює 0.01 с, то таймер може відміряти час від 10 мс до 9,99 с, бо діапазон рахування від 0 до 999. Відповідність між базою часу та можливим часом, що відміряє таймер наведена у таблиці 11.1

Таблиця 11.1 – Зв'язок значення бази часу та часу роботи таймера

База часу	Код бази часу	Час роботи таймера
10 мс	00	0.01…9.99 с
0.1 с	01	0.1с…1хв_39с _900мс
1 с	10	1с…16хв _39с
10 с	11	10с…2год_46хв_30с

Таймери мають область, зарезервовану для них у пам'яті. Ця область пам'яті резервує одне 16-бітне слово для кожної таймерної адреси. При програмуванні підтримуються від 128 до 512 таймерів відповідно модифікації контролера, що мають назви T0…T511. 16-бітне таймерне слово містить у 12 молодших розрядах значення часу у двійково-десятковому форматі та базу часу у дванадцятому та тринадцятому бітах цього слова. Коли 16-бітне таймерне слово задається константою, то база часу завантажується автоматично, а коли значення часу є змінною величиною, то базу часу необхідно визначити та завантажити самостійно. При завданні часу константою використовується тип даних S5TIME, а константа записується у вигляді S5T#час.

Наприклад, S5T#5s250ms.

Цей час у 5 секунд та 250 мс буде відповідати у таймерному слові числу 525, бо база часу автоматично буде дорівнювати 10 мс (тому, що значення часу менше ніж 10 с). У мові LAD існують 5 типів таймерів з загальним виглядом (рисунок 11.1), але різним принципом роботи.

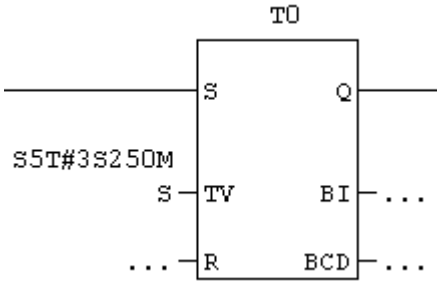


Рисунок 11.1 – Загальний вигляд таймера

Початок роботи таймера визначається зміною з 0 в 1 логічного сигналу на вході S. Значення часу для таймера подається на вхід TV. Подання логічної 1 на вхід R перериває роботу таймера та скидає його у первісний стан. На виходи BI та BCD видається поточне значення таймера у форматах Integer та BCD відповідно. Це час, що залишився до закінчення роботи таймера, бо він рахує від завданого часу до 0. Якщо на виходах BI чи BCD вказати слово у пам’яті контролера, то у нього автоматично буде зберігатися значення поточного часу. Вихід Q є логічним. Його стан краще за все можна пояснити за допомогою наступних осцилограм

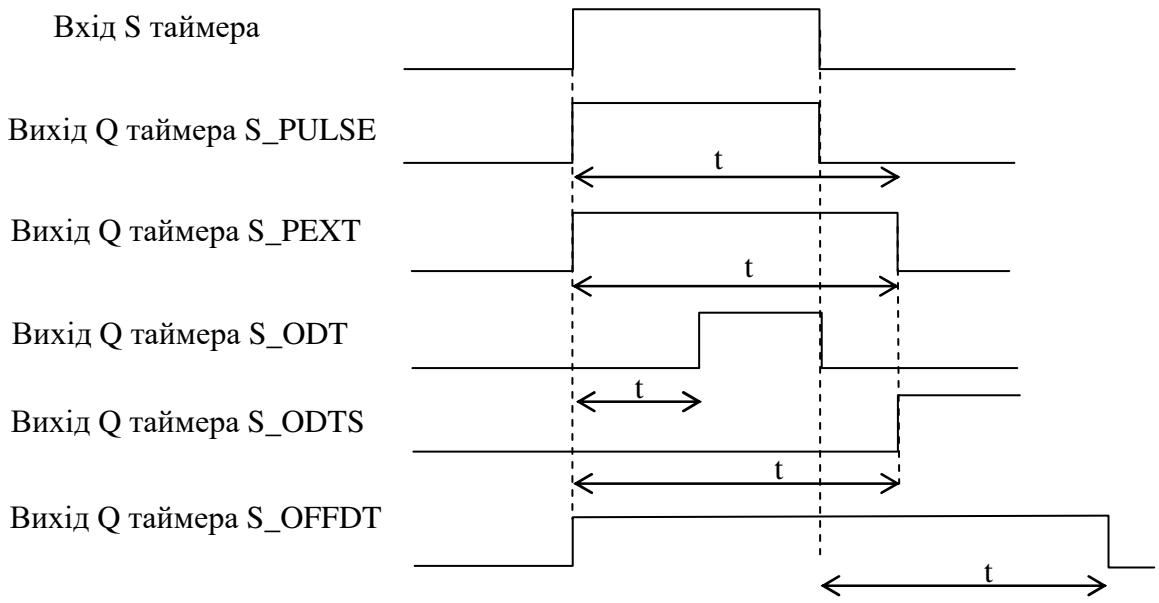


Рисунок 11.2 - Осцилограми стану логічного виходу таймерів

Перша з осцилограм відповідає вхідному сигналу, що запускає таймер. Таймер S_PULSE (Імпульс) працює після запуску поки на вході логічна 1 та не скінчився час, рівний t. Таймер S_PEXT (Імпульс із пам'яттю) після запуску завжди працює впродовж часу t, навіть при скиданні вхідного сигналу у 0. Впродовж роботи таймера на його виході логічна «1». Таймер S_ODT (Затримка включення) починає рахувати час t з появою вхідного сигнала, видає на вихід логічну 1 по закінченню цього часу і наявності на вході S логічної одиниці та змінює стан вихода на 0 зі скиданням вхідного сигнала. Таймер S_ODTS (Затримка вимикання з пам'яттю) по закінченню часу t видає на вихід логічну 1, яку можна скинути тільки поданням на вхід скидання R логічну 1. Таймер S_OFFDT (Затримка вимикання) видає на вихід Q логічну 1 з появою на вході S логічної 1, утримує його впродовж існування вхідного сигнала та впродовж часу t після скидання вхідного сигнала.

Наведені вище таймери, як команди, в деяких випадках надлишкові, бо не всі входи та виходи використовуються. У цих випадках можна розробляти програму з використанням спрощених варіантів таймера. Вони мають інші назви. Відповідність назв вже перелічених вище таймерів та спрощених команд таймерів наведено у таблиці 11.2

Таблиця 11.2 – Відповідність назв таймерів у мовах LAD та STL

S_PULSE	SP
S_PEXT	SE
S_ODT	SD
S_ODTS	SS
S_OFFDT	SF

Приклад спрощеного варіанта таймера наведено на рисунку 11.3

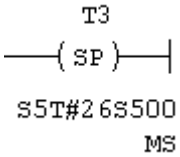


Рисунок 11.3 – Спрощений варіант таймера

Над командою вказують номер таймера, знизу – час, що таймер буде рахувати. Запускається таймер зміною з «0» в «1» логічного сигнала, що подається на вхід таймера. Спрощені команди таймерів можна розташовувати тільки у кінці рядка програми (network).

Головне призначення таймерів – це виконання деяких дій (програми) по закінчення необхідного інтервалу часу. Але програмні таймери не можуть забезпечити точне вимірювання часу, бо програмна реакція на закінчення рахування часу здійснюється тільки після виконанні рядка програми, де опитують таймер. Тому максимальна можлива погрішність часу може дорівнювати часу основного скану. Це можна пояснити так, наприклад. При опитуванні таймера він ще не закінчив рахування, але за час виконання декількох наступних команд рахування закінчилося. Знову стан таймера ми опитаємо лише у наступному скані програми, де і буде виконана реакція на цю подію. Для задач, де потрібне точне вимірювання часу слід користатися апаратним таймером та перериваннями від нього. Дана тема буде розглянута у розділі 17.

Наведемо приклад програми, що формує періодичну послідовність імпульсів на виході Q0.0, з тривалістю імпульсу 0,2 с та інтервалом між імпульсами 2 с.

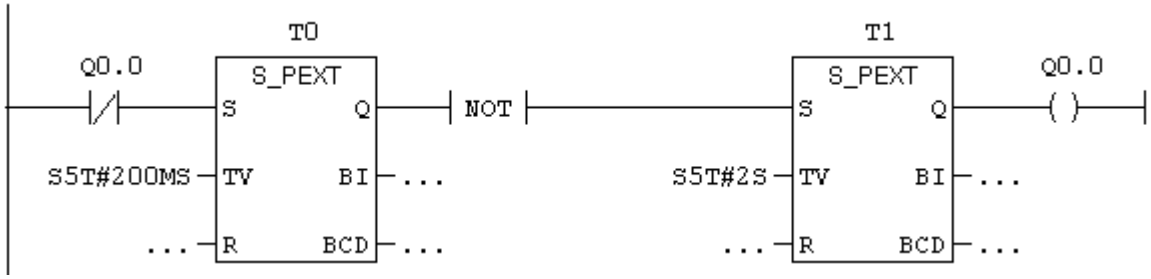


Рисунок 11.4 – Приклад програми формування періодичної послідовності імпульсів

В програмі використані 2 таймери T0 та T1. Перший з них відміряє тривалість імпульсу 0,2 с, а другий – інтервал між імпульсами. Спочатку програми опитується інверсне значення стану виходу Q0.0, яке при запуску контролера має значення 0. Тому на вхід S таймера T0 подається логічна 1 і він починає рахувати. Поки таймер T0 рахує, на його виході логічна 1, а на вході S таймера T1 логічний 0, бо стан вихода таймера T0 інвертується. Коли таймер T0 закінчив рахувати на вході S таймера T1 з'являється 1 і він починає рахувати. При цьому на вихід Q0.0 подається логічна 1. З закінченням рахування таймера T1 вихід Q0.0 скидається у 0. При наступному скані (виконанні програми) почне знов рахувати таймер T0, бо для нього інверсне значення сигналу Q0.0 на вході S зміниться з 0 в 1.

Розглянемо ще один приклад програми, що генерує пачку з 5 імпульсів на виході Q124.0, при умові що тривалість імпульсу 320 мс та інтервал часу між імпульсами 2,8 с. Генерування повинно починатися з натисканням кнопки Start, що приєднана до входу I124.0. Ця кнопка без фіксації.

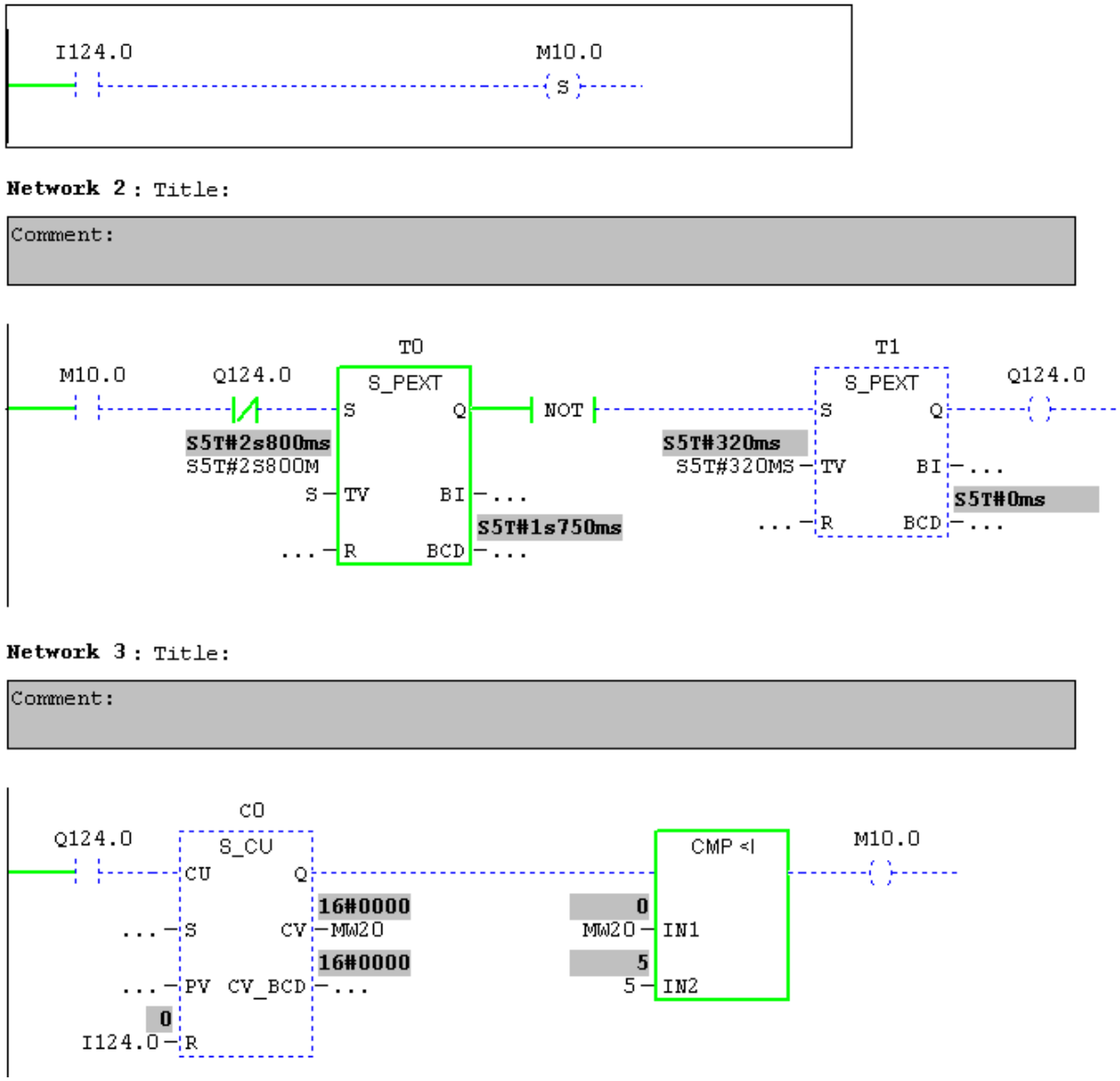


Рисунок 11.5 – Приклад програми з використанням таймерів та лічильників

Наведена на рисунку 11.5 програма складається з трьох рядків. У першому ми опитуємо кнопку Start, що під'єднана до входу I124.0. Якщо ця кнопка нажата, то результатом опитування буде логічна «1», яка дозволить встановити біт маркерної пам'яті M10.0 у «1». Завдяки використанню команди -(s)-, після відпускання кнопки стан біта M10.0 не зміниться.

У другому рядку спочатку опитуємо біт M10.0 та інверсне значення сигналу на виході Q124.0. Якщо стан біту M10.0 – логічна «1», а стан виходу Q124.0 – логічний «0», то на вхід таймера T0 прийде логічна «1» і він почне рахувати час 2,8с. По закінченню роботи таймера T0 на вході S таймера T1 з'являється «1», бо сигнал з виходу Q таймера T0 інвертується. Логічний сигнал з виходу Q таймера T1 формує на виході Q124.0 імпульс впродовж 320 мс. Змі-

на логічного сигналу з «1» в «0» на виході Q таймера T1 знову запускає таймер T0.

У третьому рядку програми встановлено лічильник C0, що рахує імпульси на виході Q124.0, бо цей сигнал подано на його вхід CU. Лічильник скидається у нульове значення при натисканні кнопки, що підключена на вхід I124.0. Нараховане лічильником число імпульсів зберігається у слові маркерної пам'яті MW20. У подальшому це значення порівнюється компаратором з константою 5. Поки нараховане значення менше 5 на виході компаратора логічна «1», що записується у біт M10.0. Тобто поки лічильник не нарахував 5 імпульсів стан біту M10.0 «1» – таймери працюють і формуються імпульси на виході Q124.0. Коли лічильник нарахує 5 імпульсів у біт M10.0 записуємо «0» і формування імпульсів припиняється.

11.2 Вивчення інструкцій таймерів у мові STL

Типи таймерів, їх призначення та можливості описані в розділі 11.1. Розглянемо принципи керування таймером.

Для запуску таймера необхідно опитати біт пам'яті, зміна якого з 0 в 1 дозволяє запустити таймер, одного з 5 типів. Наступний крок – завантаження до першого акумулятора значення часу, впродовж якого таймер повинен працювати. Далі використовуємо одну з 5 інструкцій запуску таймерів: SP, SE, SD, SS, SF, що відповідають інструкціям з таблиці 11.2 та містять у формату запису номер таймера. Наприклад

```
A I 124.3 //Опитування входа I 124.3
L S5T#2s300ms //Завантаження до акумулятора значення часу 2с300мс
SF T2 //Запуск таймера T2 при зміні сигналу на вході I 124.3 з 0 в 1
```

Для передчасного скидання таймера необхідно опитати біт пам'яті, зміна якого спричиняє це скидання та використати інструкцію скидання R №_таймера. Наприклад

```
A I 124.0 //Опитування входа I 124.0
R T2 //Скидання таймера T2
```

Для зчитування стану з логічного виходу таймера використовується інструкція A номер_таймера, а для зчитування часу до закінчення роботи таймера слід використати інструкцію L номер_таймера. Зчитане значення часу буде завантажено до першого акумулятора, а з нього це значення можна скопіювати у пам'ять.

```
A T3 //Опитування стану логічного виходу таймера T3
L T3 //Зчитування значення часу, що залишився до кінця роботи
T MW10 //Запис значення часу до слова маркерної пам'яті MW10
```

Приклад програми що формує періодичну послідовність імпульсів на виході Q0.0 з тривалістю імпульсу 0,2 с, та інтервалом між імпульсами 2 с.

```
AN Q0.0 //Опитування інверсного значення виходу Q0.0
L S5T#200ms //Завантаження до акумулятора значення часу 200ms
SE T0 //Запуск таймера T0 зі значенням часу 200ms
A T0 //Опитування стану логічного виходу таймера T0
NOT //Інвертування опитаного значення логічного стану виходу
L S5T#2s //Завантаження до першого акумулятора значення часу 2s
SE T1 //Запуск таймера T1 зі значенням часу 2s
A T1 // Опитування стану логічного виходу таймера T1
=Q0.0 //Передання логічного стану на вихід Q0.0
```

Приклад програми запуску таймера, коли значення часу завантажується з слова маркерної пам'яті

```
A I 124.3 //Опитування входа I 124.3
L MW20 //Завантаження до акумулятора значення часу з MW20
SE T2 //Запуск таймера T2
```

Порівняльний аналіз мов LAD та STL при використанні таймерів показує, що обидві мови мають однаковий набір команд таймерів з однаковими можливостями. Основною перевагою мови STL є можливість запуску та опитування поточного стану таймеру у різних частинах програми, що не можна реалізувати мовою LAD, але є дуже необхідним у деяких програмних конструкціях.

Контрольні питання

1) В яких межах та з яким часовим кроком можуть рахувати програмні таймери?

2) Що таке база часу та як її задають?

3) Скільки пам'яті займає таймер у контролері та у якому форматі до таймера завантажуються константи часу?

4) Як вірно записувати змінне значення часу у меркерній пам'яті, щоб потім його завантажувати до таймера?

5) Поясніть призначення входів та виходів п'яти програмних таймерів мови LAD.

5) Які різновиди таймерів існують і які вони генерують логічні сигнали на своєму логічному виході Q?

6) Як запустити таймер у мові STL? У якому напрямку рахує таймер?

7) Як визначити час, що залишився до кінця роботи таймера?

Контрольні завдання

Задача 11.1. Контролер повинен керувати роботою групи з 8 клапанів. Сигнали керування клапанами подаються з виходів контролера Q0.0…Q0.7. На

кожен клапан по черзі повинні подаватися імпульси з тривалістю 40 мс. Інтервал часу між відкриттями двох клапанів 1,2 с. Процес наступного відкриття групи клапанів починається через час 40 с. Процес роботи системи починається з натискання кнопки без фіксації Start, що приєднана до входу I0.0. Осцилограми сигналів, що треба сформувати, зображені на рисунку 11.6

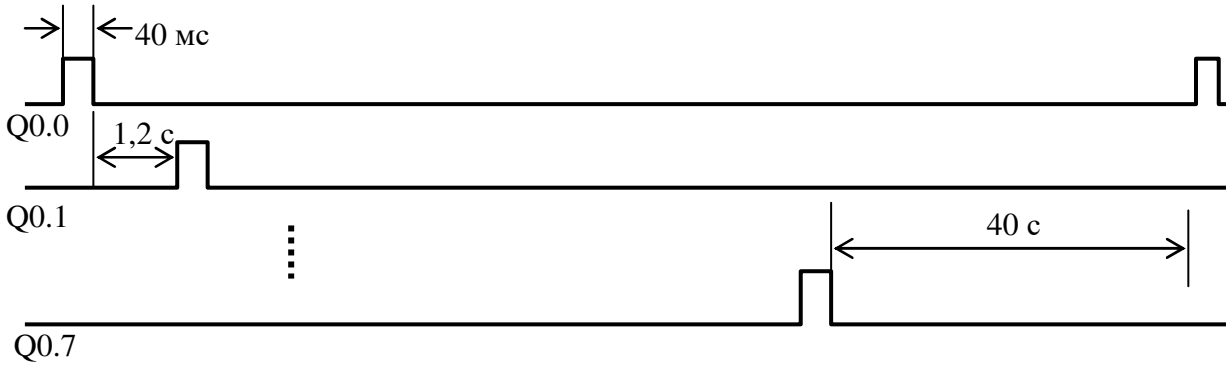


Рисунок 11.6 – Осцилограми сигналів керування клапанами

Задача 11.2. У вічку маркерної пам’яті MW50 знаходиться змінне значення часу для запуску таймера T5. Значення часу може бути від 1 до 80 с і зберігається у цілочисельному форматі. Розробити програму, що реалізує запуск таймера T5 із змінним часом роботи при натисканні кнопки Start, що приєднана до входу I124.0.

Задача 11.3. У вічку маркерної пам’яті MD120 знаходиться змінне значення часу для запуску таймера T0. Значення часу може бути від 0.1 до 15.0 с і зберігається у речовинному форматі. Розробити програму, що реалізує запуск таймера T0 із змінним часом роботи при натисканні кнопки Start, що приєднана до входу I124.0.

Задача 11.4. Внести корективи в програму, що є рішенням задачи 1 так, щоб усі значення часу були змінними. При цьому тривалість імпульсу знаходиться в межах 20...100 мс, інтервал часу між імпульсами 1.0...4.0 с, інтервал часу між повторами роботи групи клапанів від 20 до 100 с. Адреси для зберігання значень часу у маркерній пам’яті обрати самостійно.

Задача 11.5. Розробити програму генерування періодичних трикутних імпульсів на аналоговому виході PQW752. Період сигналу 1,4 с. Генерування повинно починатися автоматично із вмиканням контролера.

Задача 11.6. Розробити програму, що генерує періодичну послідовність імпульсі із періодом повторення 1с та тривалістю імпульсів від 20мс до 120 мс. Тривалість імпульсів визначається амплітудою аналогового сигналу на аналоговому вході PIW752.

Задача 11.7. Розробити програму, що періодично опитує аналоговий вхід PIW752, зчитує з нього значення току (датчик 4...20мА) та обчислює значення ковзного середнього (середнього арифметичного) за трьома останніми відліками. Періодичність відліків 230 мс. Результат розрахунку повинен зберігатися у слові маркерної пам’яті MW60.

Задача 11.8. Розробити програму, що вимірює тривалість імпульсів, що подаються на вхід I0.0. При цьому тривалість імпульсів не перевищує 3 с. Результат вимірювання повинен бути представлений у речовинному форматі.

12 Робота з таблицями символів

Раніше, при написанні програми, ми використовували абсолютні адреси дискретних входів, виходів, маркерної пам'яті (наприклад I0.1, Q124.0, MW10). Але такий спосіб завдання назв незручний, бо коли їх багато, то програміст повинен співвідносити адреси входів, виходів з їх призначенням, що затримує написання програми. Більш зручно використовувати назви входів, виходів і т.і. типа «Вхід 1-го клапана».

Введення подібних назв здійснюється з використанням таблиці символів, яка уявляє собою відповідність умовних назв та реальних адрес входів, виходів, маркерної пам'яті контролера. Для відкриття таблиці символів необхідно у головному вікні проекту обрати пункт меню SIMATIC 300 Station/CPU314C-2DP(1)/S7Program(1) та обрати програму Symbols (рисунок 12.1)

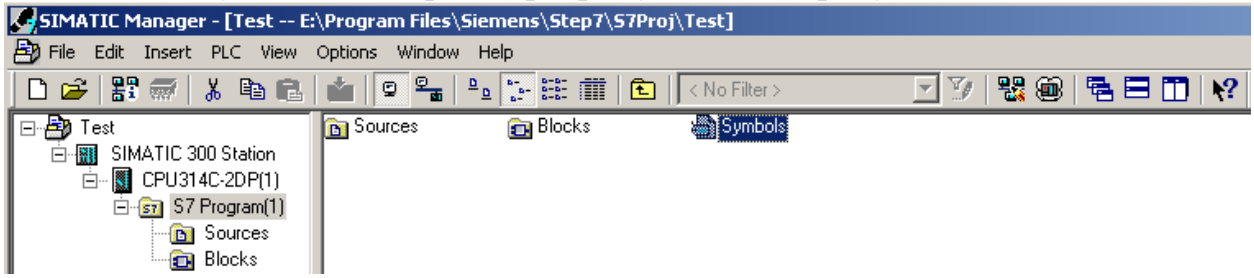
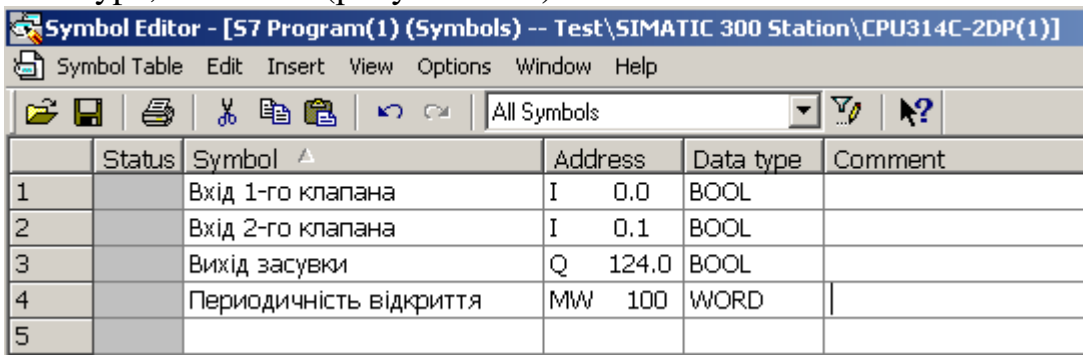


Рисунок 12.1 – Виклик таблиці символів з головного вікна проекту

У вікні, що буде відкрито, з'являється таблиця з полями Symbol, Address, Data type, Comment (рисунок 12.2)



	Status	Symbol	Address	Data type	Comment
1		Вхід 1-го клапана	I 0.0	BOOL	
2		Вхід 2-го клапана	I 0.1	BOOL	
3		Вихід засувки	Q 124.0	BOOL	
4		Периодичність відкриття	MW 100	WORD	
5					

Рисунок 12.2 – Приклад символьної таблиці

У полі Symbol вводять умовну назву входа, виходa і т.і. Далі, у полі Address, вводять адресу входа, виходa і т.і. Тип даних з'являється автоматично, але його можна корегувати, бо інколи автоматичне значення не відповідає необхідному. Наприклад, при вводу MW100 автоматичне значення типу даних – WORD (слово без знаку), але цієї адресі може відповідати тип даних із знаком INT.

Коли набір таблиці символів закінчено, то необхідно зберегти зміни.

У подальшому, при написанні програми, необхідно вводити лише першу літеру адреси (наприклад I), фактично визначив лише область пам'яті. При цьому, з'являється таблиця з усіма символьними назвами, що відповідають за типом тому, що можна вводити (рисунок 12.3).

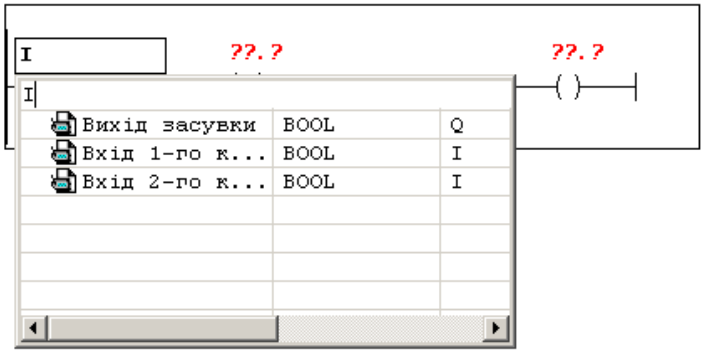


Рисунок 12.3 – Приклад вводу програми з використанням символьних назв

Обрав одну з назв отримаємо програму з використанням символьних назв (рисунок 12.4)

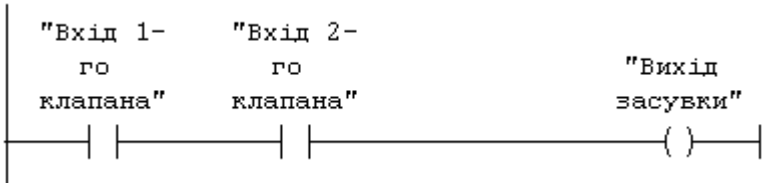


Рисунок 12.4– Приклад програми з використанням символьних назв

Контрольні питання

1) Для чого і коли використовуються таблиці символів

2) Чи займає додаткову пам'ять контролера використання таблиці символів

Контрольні завдання

Задача. Розробити з використанням символьної таблиці програму, що керує засувкою. Засувка відкривається при натисканні кнопки «Відкрити», що приєднана до цифрового входу I4.0, та закривається при натисканні кнопки «Закрити», що приєднана до входу I4.1. При натисканні кнопок на входах з'являється логічна «1». Засувка керується двома сигналами «Вмикання» та «Вимикання», що подаються з виходів контролера Q0.0 та Q0.1. При натисканні кнопки «Вмикання» на вихід Q0.0 подаємо сигнал логічної «1». При цьому, спочатку, засувка сходить з кінцевого вимикача впродовж 0,2 с, потім відкривається впродовж 2 с і, в кінці, замикається кінцевий вимикач, що говорить о закінченні процесу відкриття. При закритті засувки спочатку вона сходить з кінцевого вимикача впродовж 0,2 с, потім закривається впродовж 2 с, в кінці, замикається кінцевий вимикач, що говорить о закінченні процесу відкриття. Логічні сигнали з двох кінцевих вимикачів подаються на входи I4.2 та I4.3. Розімкнений стан кінцевого вимикача відповідає сигналу логічного «0» на вході

контролера, а замкнений стан – логічній «1». При перевищенні часу сходу з кінцевого вимикача при відкритті у подвійному слові маркерної пам’яті MD70 треба встановити в одиницю біти 16 та 31. При перевищенні часу сходу з кінцевого вимикача при закритті у подвійному слові маркерної пам’яті MD70 треба встановити в одиницю біти 17 та 31. При перевищенні часу відкриття засувки у подвійному слові маркерної пам’яті MD70 треба встановити в одиницю біти 18 та 31. При перевищенні часу відкриття засувки у подвійному слові маркерної пам’яті MD70 треба встановити в одиницю біти 19 та 31.

13 Блоки даних

Блок даних – це структура даних, тобто об’єднання декілька змінних у єдину групу за змістом чи іншим принципом, як це буде потрібно програмісту. Така структура не має загальної назви. При звертанні з програми до записів у цій структурі вказується абсолютна адреса вічка у цій структурі, тобто символьних імен записів не може бути. Блоки даних при роботі контролера розміщуються у оперативній пам’яті, а коли живлення вимикається, то автоматично інформація з блоків даних завантажується у flesh пам’ять контролера завдяки автономному живленню від внутрішньої батарейки. При наступному включенні контролера автоматично інформація в блоках даних відновиться та програма почне роботу із збереженими значеннями змінних з блоків даних. Крім цього інформація з блоків даних доступна панелями оператора чи SCADA системам. Блоки даних бувають глобальні (Shared DB), тобто доступні з будь якої програми проекту та залежні (Instance DB), що використовуються у сукупності з функціональними блоками. Залежні блоки даних теж доступні з будь-якою частини програми чи підпрограми. Структуру глобальних блоків даних розробляє програміст, а структура залежних блоків створюється автоматично із переліком вхідних та вихідних змінних функціонального блоку даних.

Для створення глобального блоку даних необхідно обрати вкладку проекту Blocks. Щигликом правої кнопки миші в полі блоків у спливаючему меню слід обрати пункт InsertNewObject/DataBlock (рисунок 13.1).

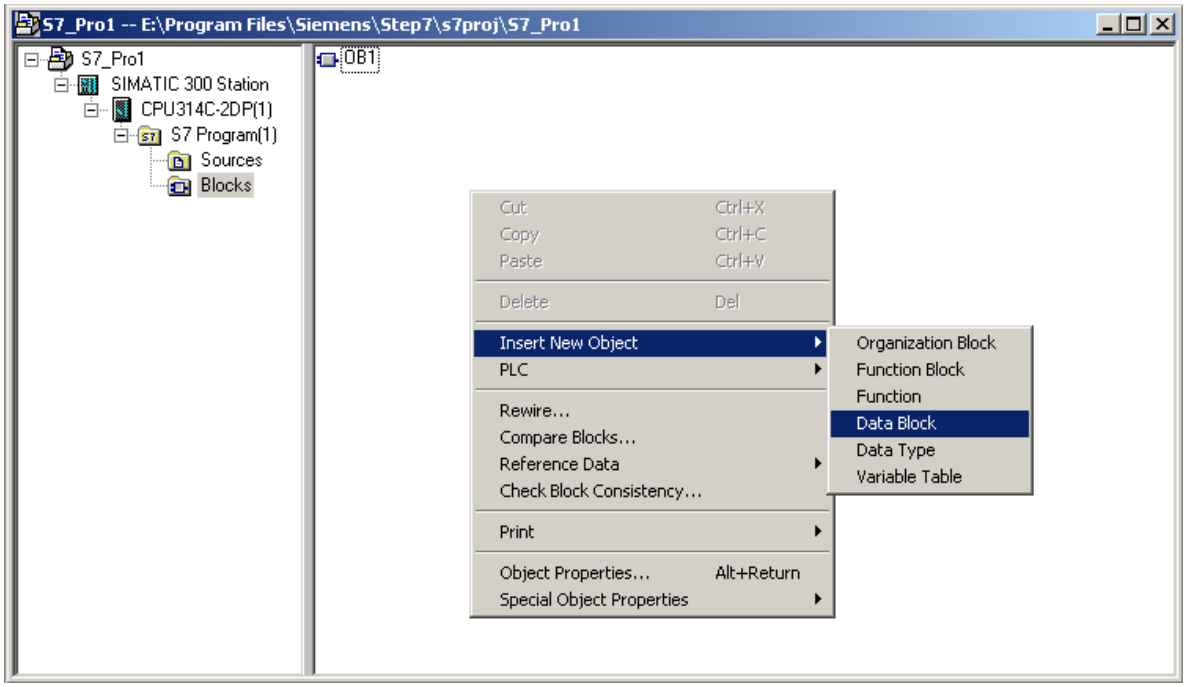


Рисунок 13.1 – Меню створення блоку даних

У вікні, що відкривається, в полі Name and type слід ввести ім'я блоку даних, які обов'язково повинно складатися з літер DB та номеру блоку. Нумерація блоків починається з одиниці – DB1, DB2 …(рисунок 13.2). Максимальний номер блоку даних визначається конкретною модифікацією контролера. У меню вибору обираємо тип Shared DB (глобальний блок даних). Блок даних може мати символьне ім'я котре вводимо у полі Symbolic name.

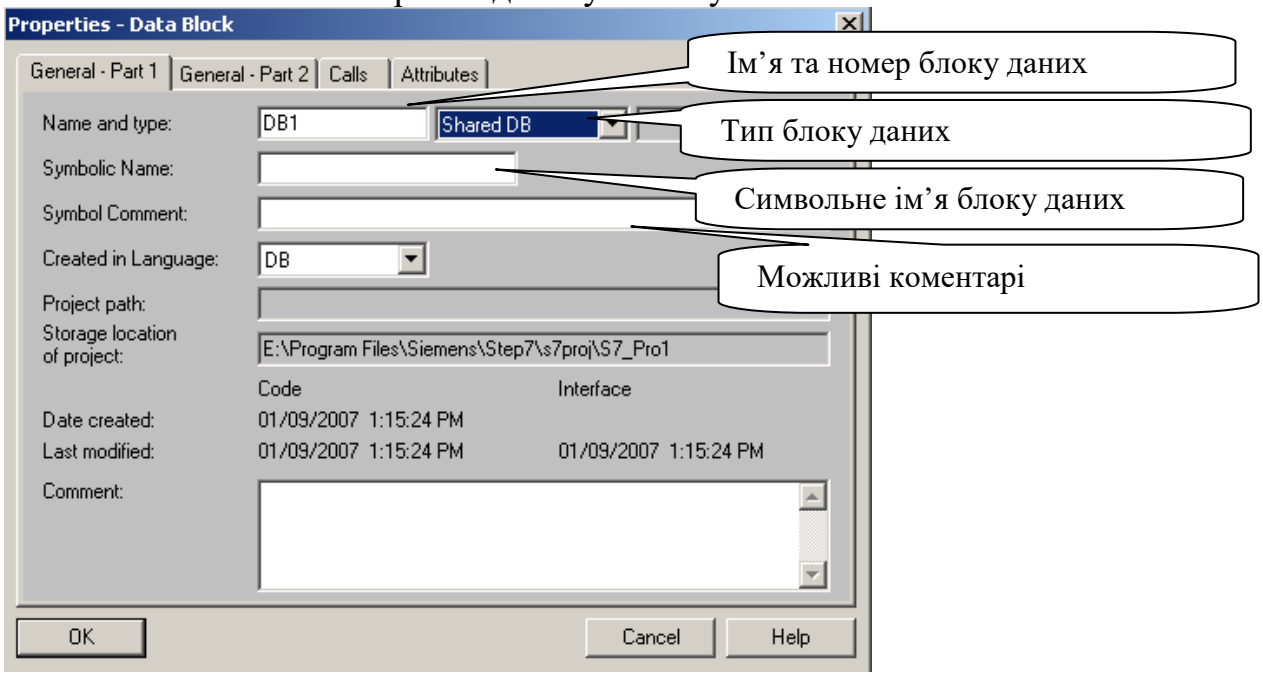


Рисунок 13.2 – Меню вибору параметрів блоку даних

Для закінчення створення нового блоку слід натиснути кнопку OK. Після цього відкривається таблиця (рисунок 13.3), у яку можна записувати ім'я змінних (поле Name), їх типи (поле Type), початкові значення (Initial value) та коме-

нтарії (поле Comment). Таблиця параметрів заповнюється послідовно, для вибору типу параметра використовується спливаюче меню (рисунок 13.3)

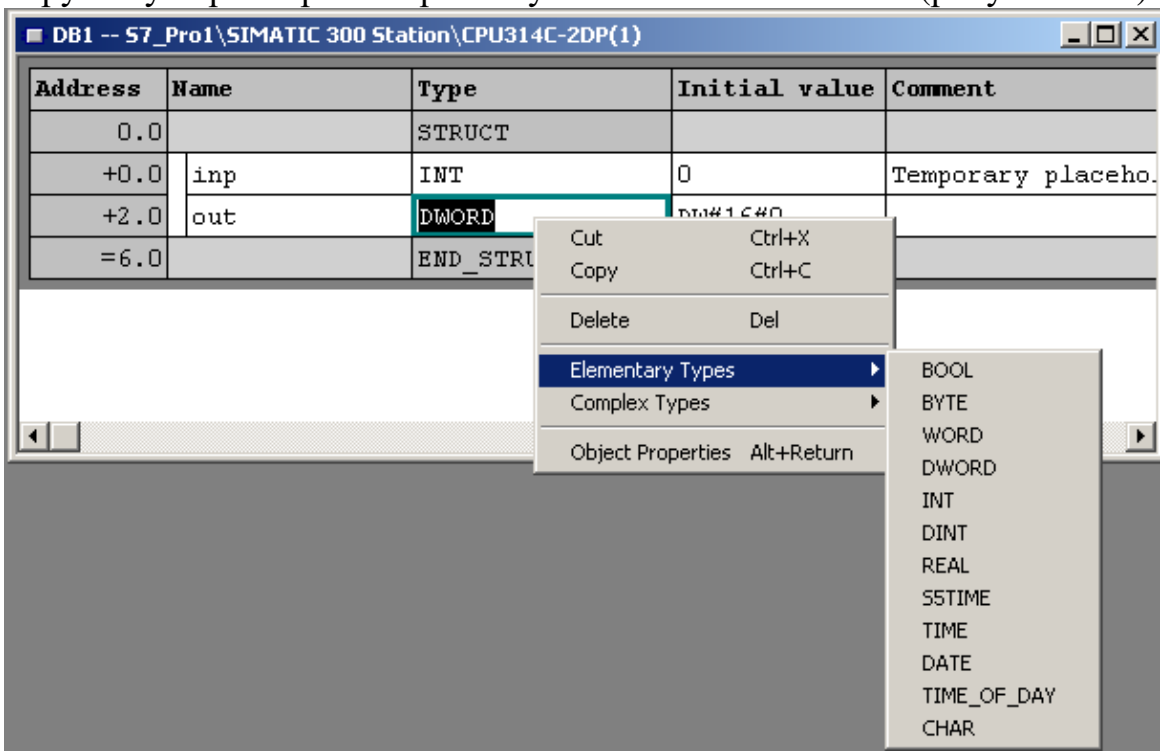


Рисунок 13.3 – Таблиця змінних блоку даних

Початкові значення змінних в блоці даних необхідно завантажувати у відповідному до типу форматі (наприклад, INT – 0, WORD – W#16#0). Після завершення формування структури блоку даних його слід зберегти та завантажити у симулятор чи реальний контролер, натиснувши кнопку  .

13.1 Використання блоків даних у мові LAD

Записи у блоках даних мають ті ж самі типи, що і змінні з областей входів, виходів, маркерної пам'яті (bool, byte, int, word, dint, dword, real і т.і.). Тому для обробки цих записів використовуються всі раніше вивчені команди у розділах 3-11. При роботі з записами у блоках даних використовуються два основних спосіба доступу до них.

13.1.1 Доступ до записів блока даних з вказанням номера блока і адреси запису

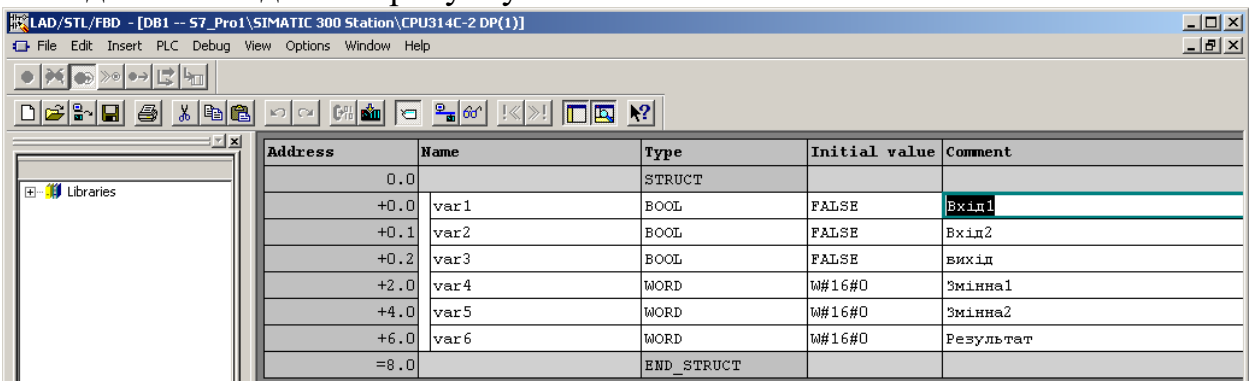
Розглянемо перший спосіб доступу до записів у блоку даних. При його використанні вказується номер блока даних, до якого йде звертання, а також адреса запису у цьому блоці даних у форматі

№_блока_даних.Адреса_запису_у_блоці.

Замість *№_блока_даних* пишуть DB1, DB2… Замість *Адреса_запису_у_блоці* пишуть:

1) DBX №_байта.№_біта – при доступу до біта з номером №_біта, який розташовано у байті з номером №_байта (наприклад, DBX3.0 – нульовий біт третього байта);
2) DBB №_байта – при доступу до байта з номером №_байта (наприклад, DBB2);
3) DBW №_слова – при доступу до змінних типів WORD та INT з блоку даних (наприклад, DBW4);
4) DBD №_подвійного_слова – при доступу до змінних типів DWORD, DINT та REAL з блоку даних (наприклад, DBD8).

Наведемо приклади програм з використанням першого спосіба доступу до змінних блоків даних. Для цього, спочатку, розробимо блок даних, у якому міститься три змінних типу BOOL та три змінних типу WORD. Розроблений блок даних наведено на рисунку 13.4



Address	Name	Type	Initial value	Comment
0.0		STRUCT		
+0.0	var1	BOOL	FALSE	Вхід1
+0.1	var2	BOOL	FALSE	Вхід2
+0.2	var3	BOOL	FALSE	вихід
+2.0	var4	WORD	W#16#0	Змінна1
+4.0	var5	WORD	W#16#0	Змінна2
+6.0	var6	WORD	W#16#0	Результат
=8.0		END_STRUCT		

Рисунок 13.4 – Структура блоку даних DB1

В блоці DB1 змінні var1, var2, var3 розташовані у бітах 0,1,2 нульового байту та мають тип BOOL, змінні var4,var5, var6 розташовані у 2-му, 4-му та 6-му словах та мають тип WORD.

Приклад 1. Розробимо програму, що виконує операцію логічного «ТА» із значеннями змінних var1 та var2. Результат логічної операції записуємо у змінну var3. При вводі адрес змінних у програму (рисунок 13.5) достатньо ввести лише початкову літеру D назви блока DB1 і у нас з’являється спливаюче меню з блоками даних і записами у них, що відповідають за типом входу команди.

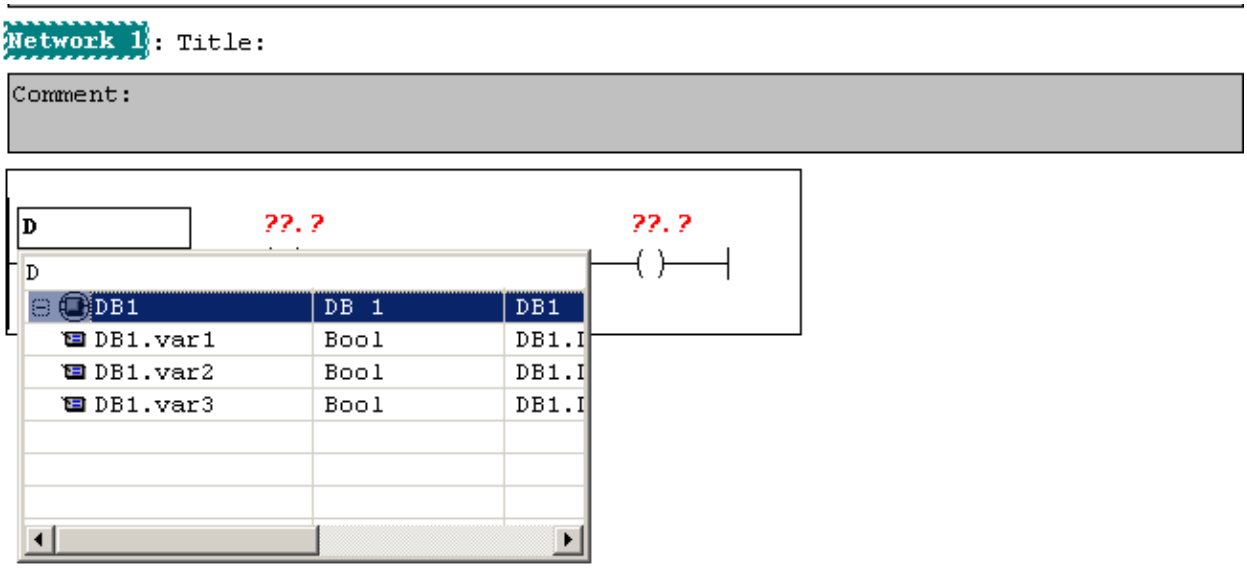


Рисунок 13.5 – Спливаюче меню при звертанні до блоку даних.

Обираючи необхідну змінну блоку даних, чи вводячи повну адресу запису з блоку даних, маємо вигляд програми, що наведено на рисунку 13.6

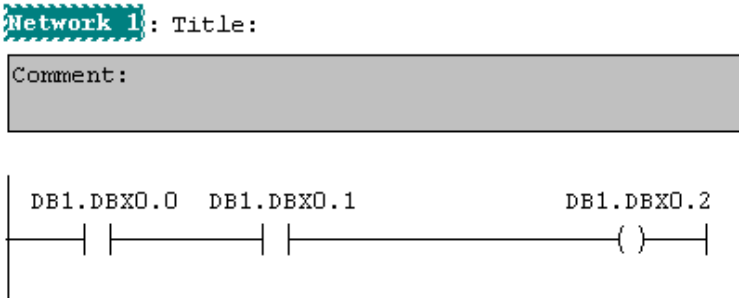


Рисунок 13.6 – Приклад програми, що виконує операцію логічного «ТА» з записами блоку даних.

Слід звернути увагу, що у введеній програмі (рисунок 13.6) відсутні назви змінних, а відображуються лише їх адреси (DB1.DBX0.0 …).

Приклад 2. Розробимо програму, що виконує складання змінних var4 та var5 з блоку даних DB1. Результат розрахунку записуємо у змінну var6 того ж блока даних.

Розроблену програму наведено на рисунку 13.7

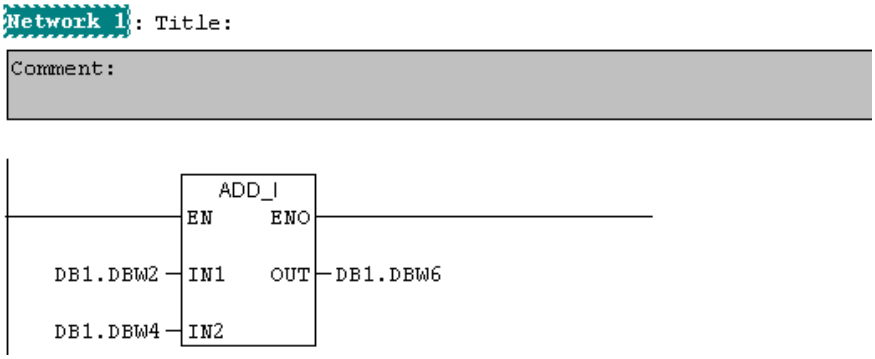


Рисунок 13.7 – Програма складання двох змінних з блоку даних DB1.

При написанні цієї програми у спливаючому меню структура боку даних DB1 не відображується.

13.1.2 Доступ до записів блока даних з використанням команди OPN

При реалізації данного способу доступу до записів блоку даних використовується команда відкриття блоку даних $-\overset{???}{(\text{OPN})}-$. Над цією командою вказується номер блока даних, що потрібно відкрити. Далі у тексті програми, при звертанні до записів у блоці даних, вказується лише адреса запису:

1) DBX №_байта.№_біта – при доступу до біта з номером №_біта, який розташовано у байті з номером №_байта;
2) DBB №_байта – при доступу до байта з номером №_байта;
3) DBW №_слова – при доступу до змінних типів WORD та INT з блоку даних;
4) DBD №_подвійного_слова – при доступу до змінних типів DWORD, DINT та REAL з блоку даних.

Слід звернути увагу, що при використанні команди (OPN) чи при першому способі звертання до інформації в глобальному блоці даних номер відкритого блока завантажується до регістру DB у контролері. При відкритті залежного блока даних його номер завантажується до регістру DI у контролері. Тому одночасно можна відкрити лише один глобальний та один залежний блок даних і він залишається відкритим, поки не буде відкритий інший блок даних такою ж командою (OPN). Поки блок даних залишається відкритим можлива будь-яка кількість програмних звертань до записів цього блоку.

Такий спосіб доступу до записів з блоку даних є універсальним, бо використовується при прямій та непрямій адресації (дивись розділ 16). Крім того цей спосіб більш швидкісний, бо команда відкриття блоку даних виконується лише один раз до обробки змінних відкритого блоку, в протилежність використанню повної форми звертання до змінної блока даних (наприклад DB1.DBX0.0), коли команда відкриття блока даних виконується кожного разу при подібному звертанні.

Приклад3. Розробимо програму, що відкриває розроблений блок даних DB1 та виконує операцію логічного «І» із значеннями змінних var1 та var2. Результат логічної операції записуємо у змінну var3.

Розроблену програму наведено на рисунку 13.8

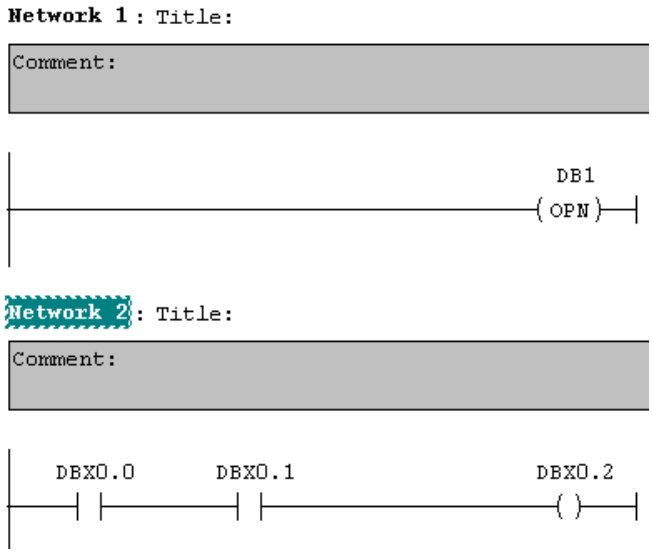


Рисунок 13.8 – Приклад програми, що виконує операцію логічного «І» з записами блоку даних.

У першому рядку програми відкривається перший блок даних командою OPN, що розташована у кінці рядка. Такий виклик є безумовним, бо до команди OPN не має ні однієї команди, що може змінювати стан біта RLO контролера. Безумовний виклик є найбільш поширеним на практиці. У випадку умовного відкриття блоку обов'язковим є відкриття одного з сукупності блоків даних, бо далі виконується програма з обробки записів у блоці.

Приклад 4. Розробимо програму, що відкриває блок даних DB1 та виконує складання змінних var4 та var5. Результат розрахунку записуємо у змінну var6 того ж блока даних.

Розроблену програму наведено на рисунку 13.9

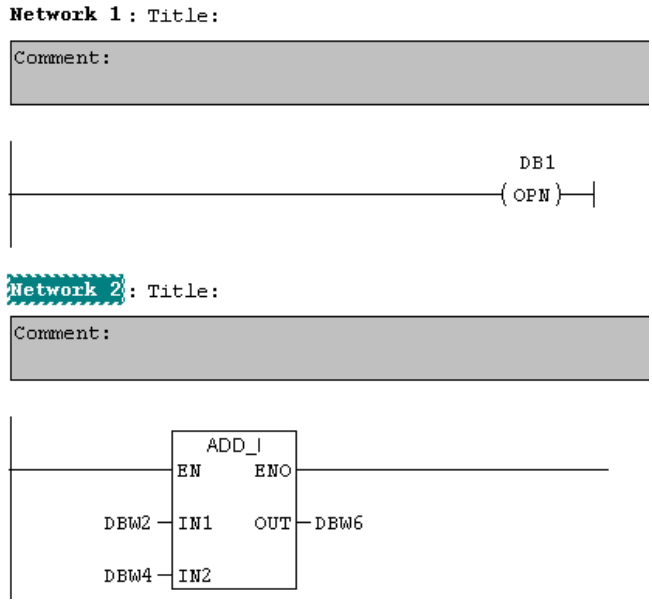


Рисунок 13.9 – Програма складання двох змінних з блоку даних DB1.

13.2 Використання блоків даних у мові STL

У розділі 13.1 були описані два способи доступу до записів у блоці даних. Вони однаково застосовуються і в мові STL з усіма командами цієї мови.

Команда OPN, що відкриває блок даних має формат

OPN №_блока_даних.

При її використанні у внутрішні регістри контролера DB чи DI завантажується номер відкритого глобального чи залежного блоку даних відповідно.

Нижче наведемо приклади програм виконання логічної операції «ТА» із значеннями змінних var1 та var2. Результат логічної операції записуємо у змінну var3.

Приклад програми з використанням формата повної адреси

```
A    DB1.DBX   0.0 //Логічна операція «ТА»
A    DB1.DBX   0.1 //із змінними за адресами DBX 0.0 та DBX 0.1
=    DB1.DBX   0.2 //Збереження результату у змінній за адресою DBX 0.2
```

Приклад програми з використанням команди OPN

```
OPN  DB    1 //Відкриття блоку даних
A    DBX   0.0 //Логічна операція «ТА»
A    DBX   0.1 //із змінними за адресами DBX 0.0 та DBX 0.1
=    DBX   0.2 //Збереження результату у змінній за адресою DBX 0.2
```

Нижче наведемо приклади програм, що виконують складання значень змінних var4 та var5 з блоку даних DB1 при використанні повної адреси та команди OPN. Результат розрахунку записуємо у змінну var6 того ж блока даних.

Приклад програми з використанням формата повної адреси

```
L    DB1.DBW   2 //Завантаження до акумулятора слова DBW 2 з блоку DB1
L    DB1.DBW   4 //Завантаження до акумулятора слова DBW 4 з блоку DB1
+I               //Складання значень у двох акумуляторах
T    DB1.DBW   6 //Збереження результату у слові DBW 6 з блоку DB1
```

Приклад програми з використанням команди OPN

```
OPN  DB    1 //Відкриття блоку даних
L    DBW   2 //Завантаження до акумулятора слова DBW 2 з блоку DB1
L    DBW   4 //Завантаження до акумулятора слова DBW 4 з блоку DB1
+I           //Складання значень у двох акумуляторах
T    DBW   6 //Збереження результату у слові DBW 6 з блоку DB1
```

13.3 Формування переліка змінних для SCADA систем

У реальних системах автоматизації виробництва робота контролерів невід'ємно пов'язана з візуалізацією (графічним відображенням) технологічного процесу, а також з архівуванням стану системи та аварійних ситуацій в роботі обладнання. Такі задачі реалізуються з використанням промислових комп'ютерів, на яких встановлено спеціалізований програмний продукт – SCADA систему. Існує багато виробників таких програмних продуктів, серед яких і корпорація Siemens. Корпорацією Siemens пропонується для використання SCADA система – WinCC, яка є передовим програмним продуктом та дуже добре зарекомендувала себе на виробництві. Для графічного відображення роботи автоматизованої системи розробник з бібліотеки графічних елементів розробляє її мнемосхему. Цей процес дуже нагадую розробку зображення в графічних редакторах Corel Draw, Adobe Photoshop та ін. Потім кожен графічний об'єкт зв'язують із змінними, що передаються у SCADA систему з вічків пам'яті контролера. У кожної змінної є ім'я та відповідна у пам'яті контролера адреса. При роботі SCADA системи зміна значень у пам'яті контролера призводить до змін кольору, положення графічних об'єктів, ці значення відображуються оператору, він може їх змінювати, тобто керувати технологічним процесом.

Одною з переваг використання SCADA системи WinCC є можливість автоматичного формування за допомогою пакета Step7 переліка змінних, для використання їх в SCADA системі. Такий спосіб дає можливість значно скоротити час створення мнемосхеми технологічного процесу і уникнути помилок при вказуванні адрес у пам'яті контролера, що відповідають змінним. Розглянемо цей спосіб детальніше.

Для початку на комп'ютері разом з пакетом Step7 повинна бути встановлена SCADA система WinCC.

Далі, у головному вікні проекту слід створити графічну станцію, тобто новий проект для SCADA системи. Для цього треба виділити назву проекту, викликати спливаюче меню та у ньому обрати пункт Insert New Object/OS (рисунок 13.10)

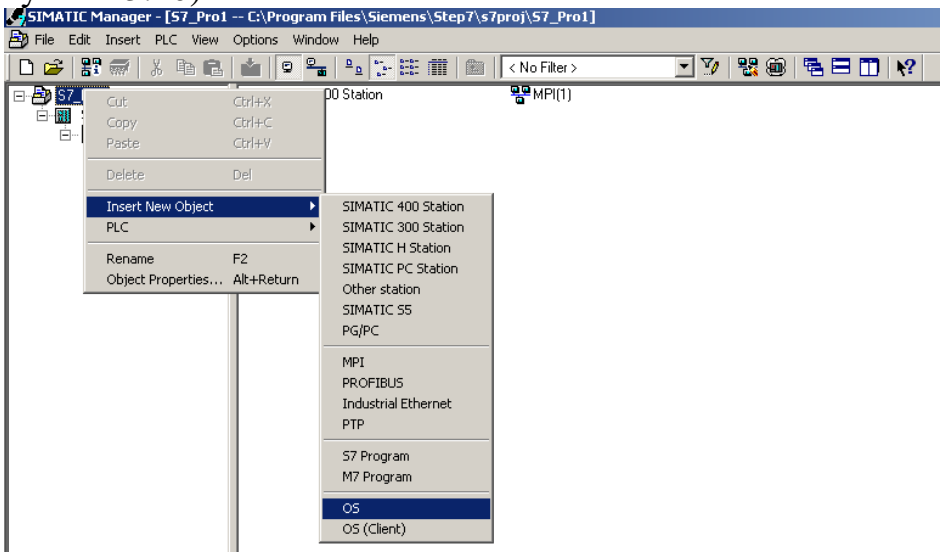


Рисунок 13.10– Створення графічної станції

Автоматично буде створена графічна станція з ім’ям OS(1) (рисунок 13.11)

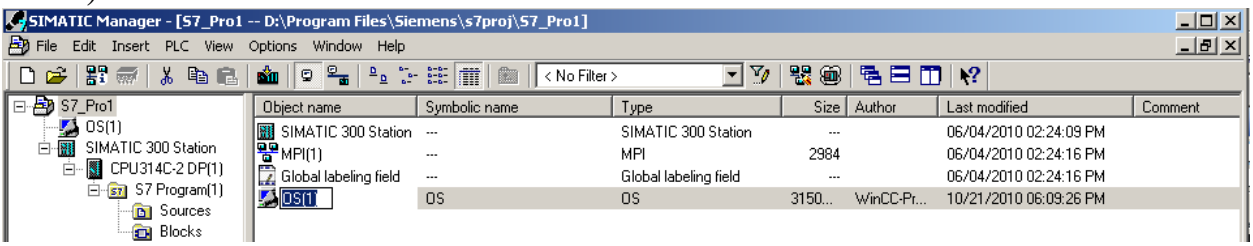


Рисунок 13.11 – Створена графічна станція з автоматично отриманим ім’ям OS(1)

Ім’я станції можна змінити на інше традиційним способом, виділивши його перед тим мишкою. У наведеному на рисунку 13.12 вікні ім’я графічної станції змінено на Drive

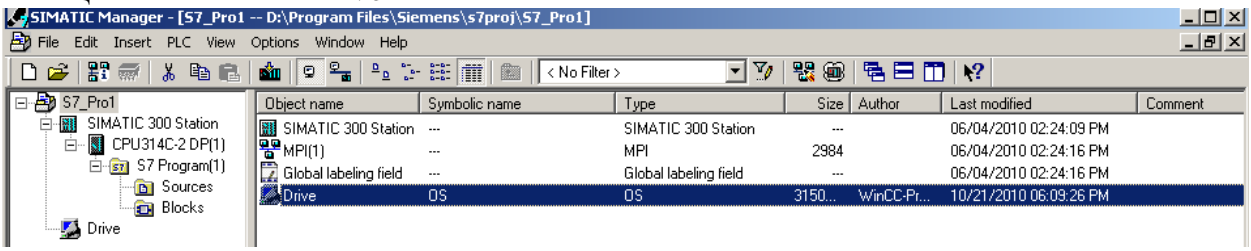


Рисунок 13.12 – Зміна імені станції на Drive

Далі треба розробити блоки даних та їх структури, відповідно розділу 13.1. У відкритому блоці необхідно для кожної змінної, значення якої буде передаватися у SCADA систему, встановити атрибут S7_m_c в значення true. Для цього треба клацнути правою кнопкою миші на полі блока даних та обрати пункт спливаючого меню (рисунок 13.13).

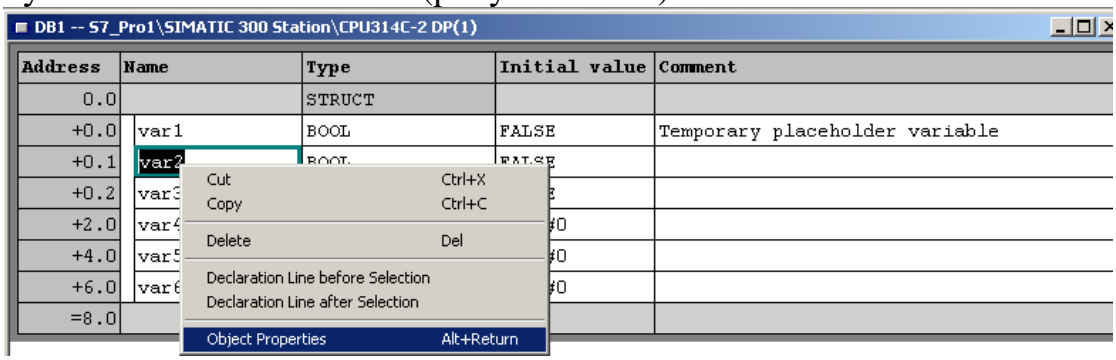


Рисунок 13.13– Виклик вікна встановлення атрибутів

У вікні, що з’явиться після цього треба ввести ім’я атрибута (S7_m_c) та його значення відповідно рисунку 13.14. Змінні, у яких цей атрибут вже встановлено, помічаються умовно прапорцем (рисунок 13.14).

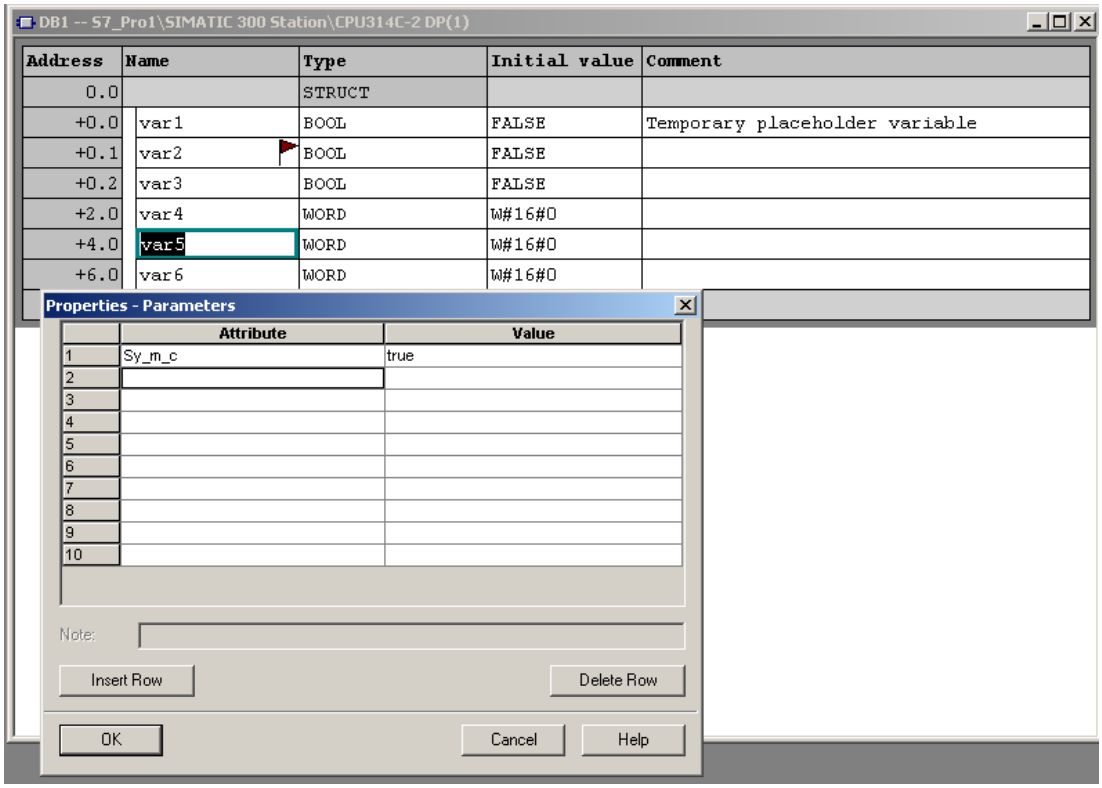
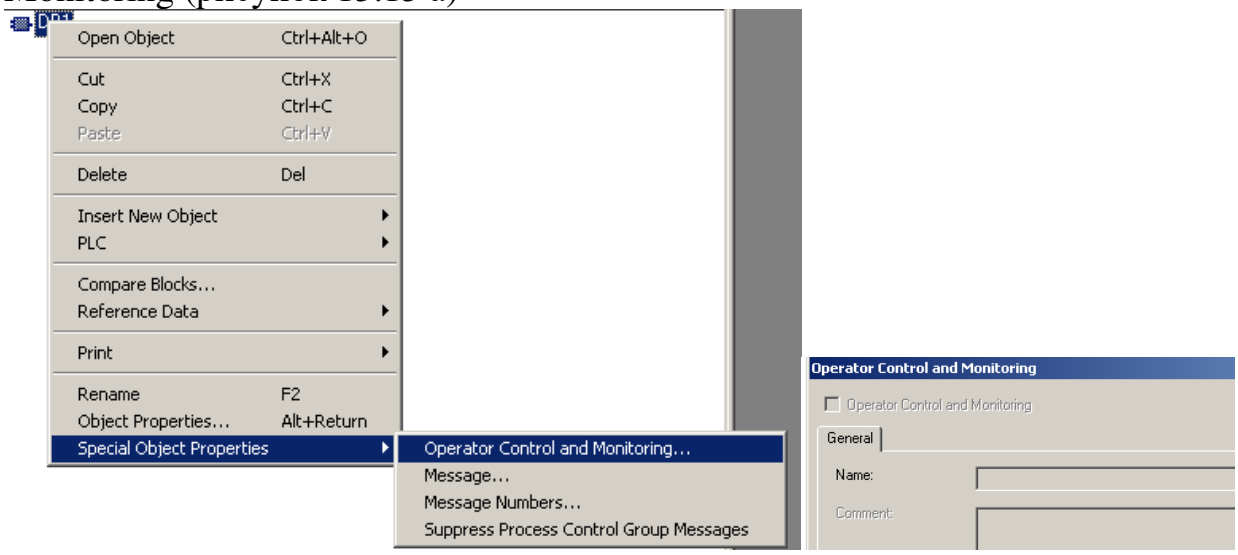


Рисунок 13.14 – Встановлення атрибуту S7_m_c для змінних блока даних

Далі, у головному вікні проекту, треба викликати спливаюче меню спеціальних властивостей блока даних Special Object Properties/Operator control and Monitoring (рисунок 13.15 а)



а б

Рисунок 13.15– Виклик меню спеціальних властивостей блока даних

У вікні, що з’явиться після цього треба поставити галочку біля параметру Operator control and Monitoring. На цьому налаштування змінних для передавання у SCADA систему закінчується

Для формування переліка змінних у SCADA системі треба у головному вікні проекту обрати пункт спливаючого меню Compile

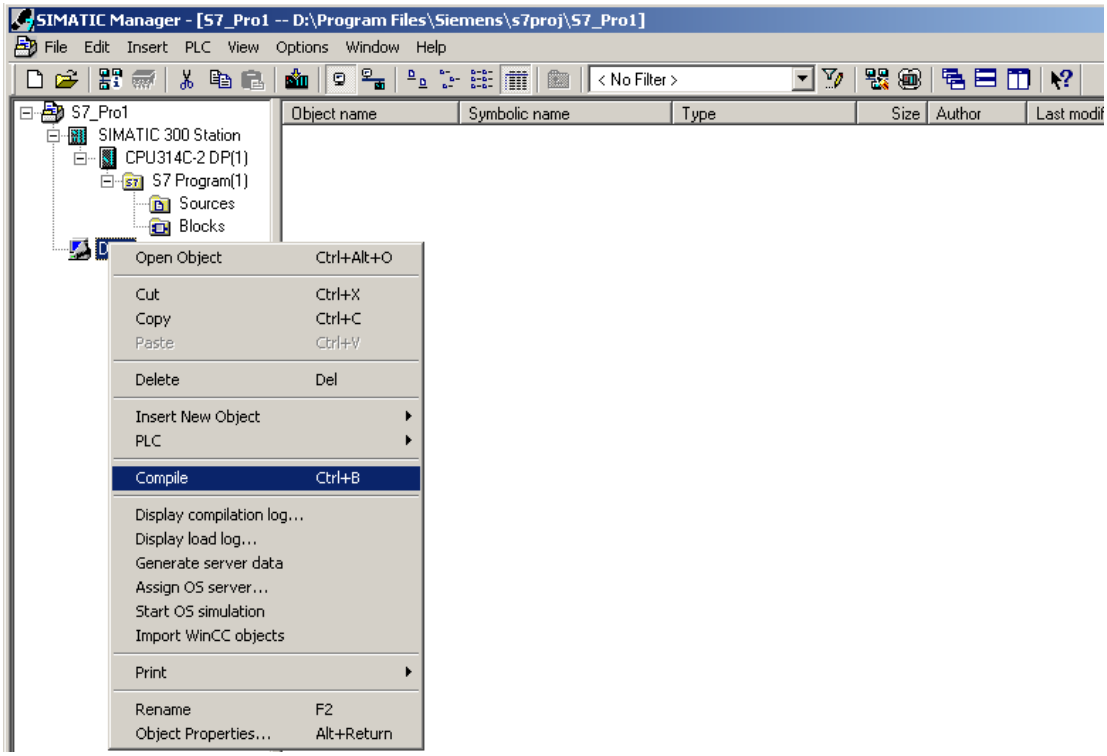


Рисунок 13.16 – Компіляція станції

Далі з’явиться вікно (рисунок 13.17), у якому можна обрати проекти для контролерів, дані з яких будуть передаватися SCADA систему. Тут слід сказати, що у одному проекті Step7 можна розробляти програмне забезпечення для декількох контролерів, що об’єднуються у єдину мережу. Тому при компіляції графічної станції користувачу надається можливість обрати ті контролері, котрі будуть передавати інформацію графічній станції

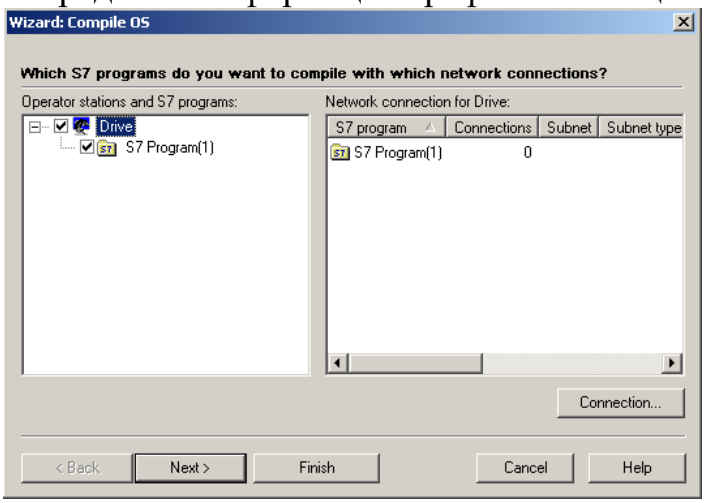


Рисунок 13.17 – Меню вибору контролерів для компіляції

Натиснувши кнопку Next переходимо до наступного графічного вікна (рисунок 13.18). У ньому користувачу надається можливість або повністю сформувати перелік всіх змінних для SCADA системи, або тільки зміни, що були введені після останньої компіляції.

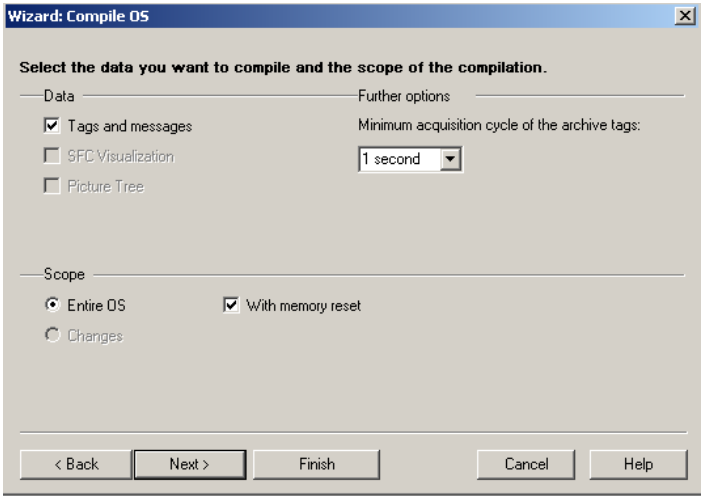
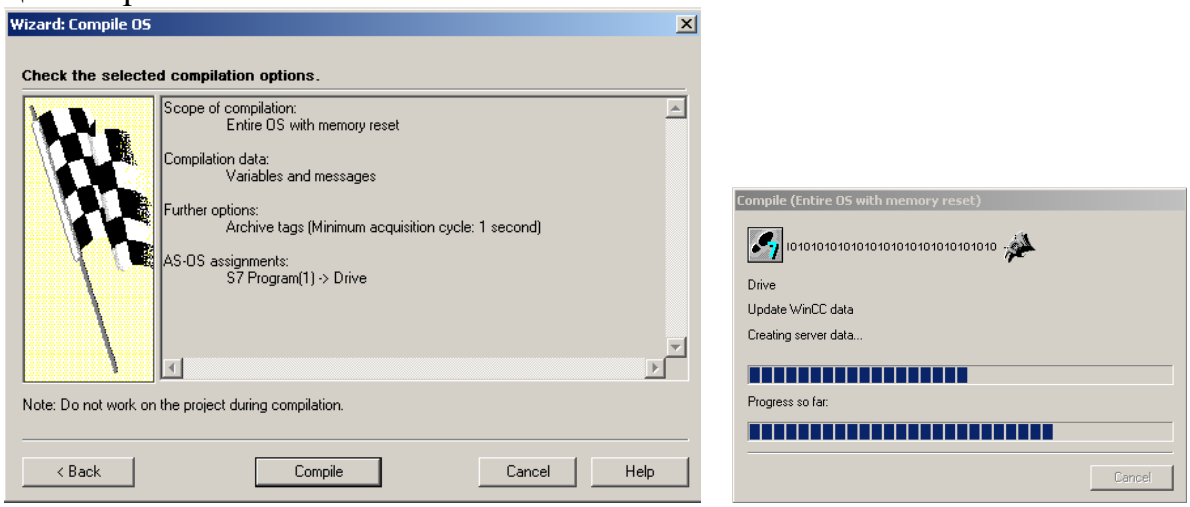


Рисунок 13.18– Меню вибору параметрів для компіляції

При першій компіляції такого вибору не існує (тільки пункт налаштування Entire OS) і формується повний перелік змінних. При наступних компіляціях стає доступним пункт налаштування Changes, що дозволяє скомпілювати лише зміни, що були введені.

Натиснувши кнопку Next переходимо к останньому пункту меню (рисунок 13.19 а), у якому для початка компіляції слід натиснути кнопку Compile. Після натискання цієї кнопки з'являється вікно (рисунок 13.19 б), що показує процес формування проекту графічної станції і створення переліка змінних для цього проекта.



а б

Рисунок 13.19 – Меню початку компіляції та передання даних у станцію

Порівняльний аналіз мов LAD та STL показує, що при роботі з блоками даних мови мають однаковий набір інструкцій. Тому їх можливості однакові, хоча використання мови STL дозволяє обробляти не тільки слова та подвійні слова, але і байти з блоків даних. Тому, при необхідності обробки батов, можна використовувати лише мову STL.

Контрольні питання

1) Для чого використовуються блоки даних у контролерах Siemens?

2) Чи існує обмеження на кількість і об’єм блоків даних у контролерах Siemens?

3) Структури із змінними яких типів даних можна створювати у блоках даних? Що відбувається з даними у блоках даних при вимиканні живлення контролера?

4) Які основні два способи доступу до даних у блоках даних існує у мовах LAD та STL?

5) Чи можна відкрити одночасно два глобальних блока даних? Поясніть відповідь?

6) Чи можна відкрити одночасно два залежних блока даних? Поясніть відповідь?

7) Як автоматично передати значення деяких змінних з блоку даних до SCADA системи WinCC? Що дає використання автоматичного формування переліку змінних для SCADA системи WinCC за допомогою пакету Step7?

Контрольні завдання

Задача 13.1. Створити два глобальних блока даних DB1, DB3 із п’ятьма змінними у них var1, var2, var3, var 4, var5. Типи цих змінних bool, byte, word, int, real. Початкові значення змінних False, 3, 5620, -6, 3.1. Розробити програми мовами LAD та STL, що копіюють дані із блока DB1 до DB3 при натисканні кнопки без фіксації, приєднаної до входу I0.0.

Задача 13.2. Розробити фільтр ковзного середнього за трьома відліками сигналу з аналогового входу PIW752, що зберігаються у трьох змінних sample1, sample2, sample3 з блоку даних DB1. Періодичність обробки вхідного сигналу 0,2 с. Результат фільтрації записується у четверте слово result того ж блоку даних.

14 Функції

Сучасним мовам програмування притаманний модульний принцип побудови програми. Сутність цього принципу у тому, що велика програма складається з багатьох модулів (дрібних підпрограм), що викликаються послідовно з головної програми.

Промислові мови програмування не є винятком. Так, при програмуванні контролерів Siemens, велика програма в модулі OB1 послідовно викликає підпрограми – функції та функціональні блоки. Не суттєво якою мовою написана кожна з підпрограм та якою мовою здійснюється виклик. Основні переваги та необхідність використання підпрограм полягають у наступному:

1) у контролерах Siemens об’єм коду кожної з програм не може перевищувати межу, яка визначається типом контролера та дорівнює 16-64 кБайт коду програми в залежності від контролера. Тому скласти одну велику програму неможливо, а викликати з головної програми багато дрібних підпрограм з великим сумарним об’ємом коду не складно;
2) перевіряти та налагоджувати одну велику програму значно складніше, ніж окремі підпрограми поодинці;

3) коли в тексті головної програми зустрічається декілька раз виконання одних і тих же дій з однаковими чи різними початковими даними. Прикладом може бути керування декількома насосами чи клапанами, котрі або зовсім не відрізняються друг від друга, або відрізняються технічними параметрами (струмом, напругою і т.і.). Керування такими приладами здійснюється по черзі. Наприклад, алгоритм керування усіма клапанами однаковий. Початкові данні – різні. Отже програму керування слід писати, як окрему підпрограму, і визивати за час основного скану стільки раз, скільки в системі є клапанів. При кожному з викликів підпрограми в неї треба передавати початкові дані відповідного клапану у вигляді вхідних параметрів (змінних). Результати роботи підпрограми передаються в головну програму у вигляді вихідних параметрів (змінних) – сигналів керування клапаном.

При написанні підпрограм можливо визивати з них інші підпрограми. Максимальна глибина вкладання підпрограм дорівнює 8 для контролерів серії S7-300 та 24 для контролерів серії S7-400.

При програмуванні контролерів Siemens існують два різновиди функцій: системні та користувацькі. Системні функції – це підпрограми, що розроблено компанією Siemens. Вони входять до складу бібліотек, що йдуть у дистрибутиві середовища програмування Step7. Такі функції завжди мають назву SFC та номер.

Користувальницькі функції розробляє автор-програміст і користується ними у подальшому. Такі функції мають назву FC та номер. Кількість функцій, що можна розробити, обмежується можливостями контролера. Для контролерів серії S7-300 їх не може бути більше ніж 2048, а для контролерів серії S7-400 їх не може бути більше 6144. Користувацькі функції можуть бути з вхідними та вихідними параметрами (змінними) чи без них.

Розглянемо процес створення користувацької функції.

Для створення функції необхідно у головному вікні проекту обрати вкладку Blocks. Після щиглика правою кнопкою миші в поле блоків у спливаючему меню виберіть пункт InsertNewObject/Function.

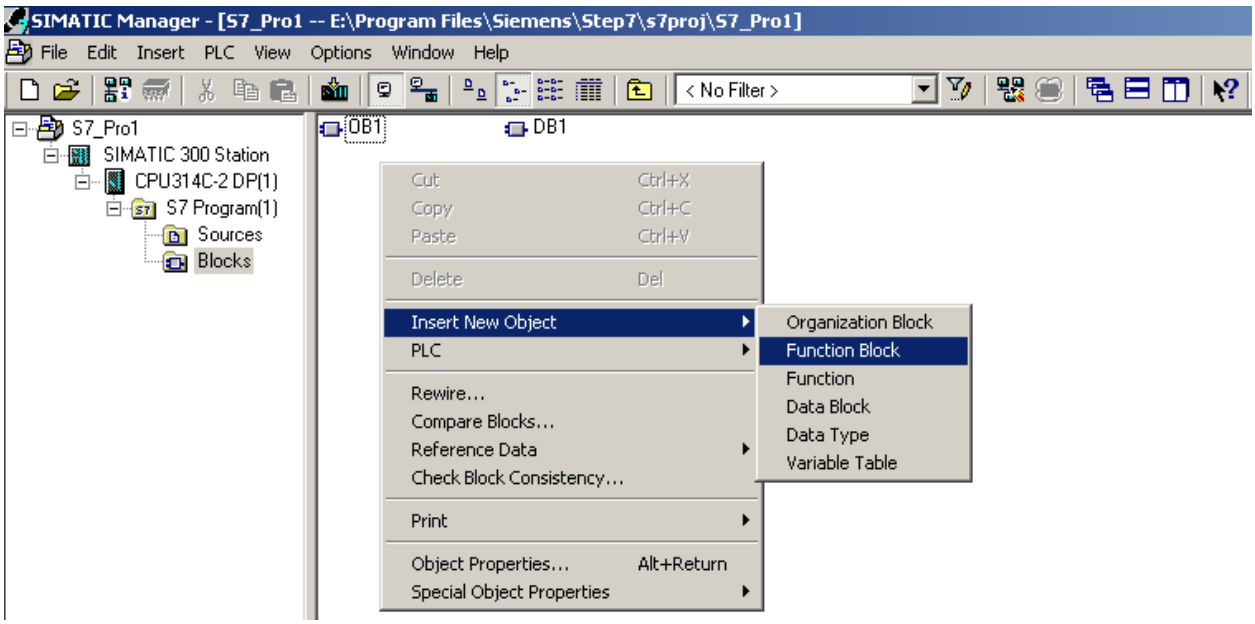


Рисунок 14.1 – Меню створення функції

Далі з'являється графічне вікно створення нової функції (рисунок 14.2)

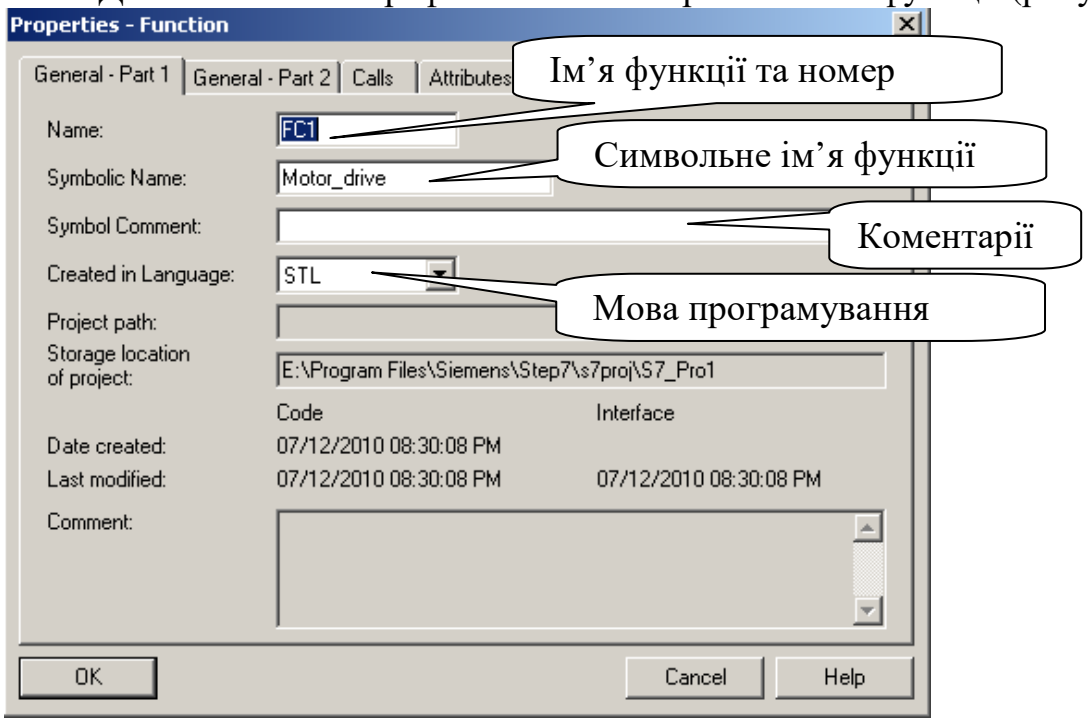


Рисунок 14.2 – Графічне вікно створення нової функції

Обов'язковим є ввод імені функції FC та номера. Нумерація функцій починається з 1. Символьне ім'я дозволяє ввести і в подальшому використовувати зручне для розуміння програміста ім'я функції. Якщо у наведеному на рисунку 14.2 прикладі функція FC1 має символьне ім'я Motor_drive, то і у тексті програми замість FC1 буде автоматично підставлено ім'я Motor_drive. Ввід символьного імені не обов'язковий. У тексті програми функції, що не мають символьного імені будуть записані як FC1, FC2 і т.і. У полі Symbol Comment можна ввести коментарі рідною мовою. Мову програмування можна обрати одразу чи змінити у подальшому. Для закінчення формування функції слід натиснути

кнопку ОК. Тепер нова функція з'явилася у папці Bloks головного вікна проекту.

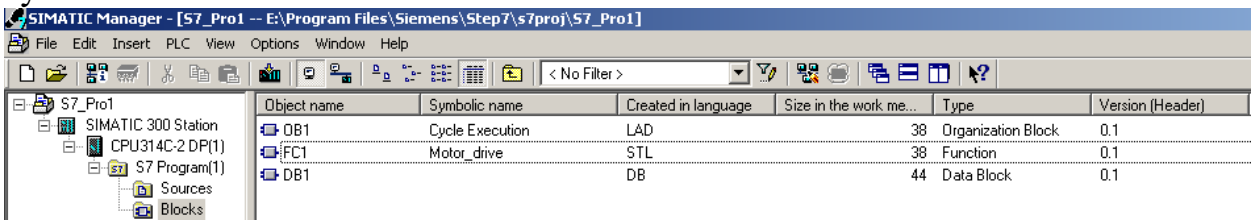


Рисунок 14.3 – Головне вікно програми після створення нової функції

Її можна відкрити подвійним клацанням миші і почати розробку підпрограми. Спочатку треба ввести усі вхідні, вихідні та тимчасові параметри у відповідних полях (рисунок 14.4)

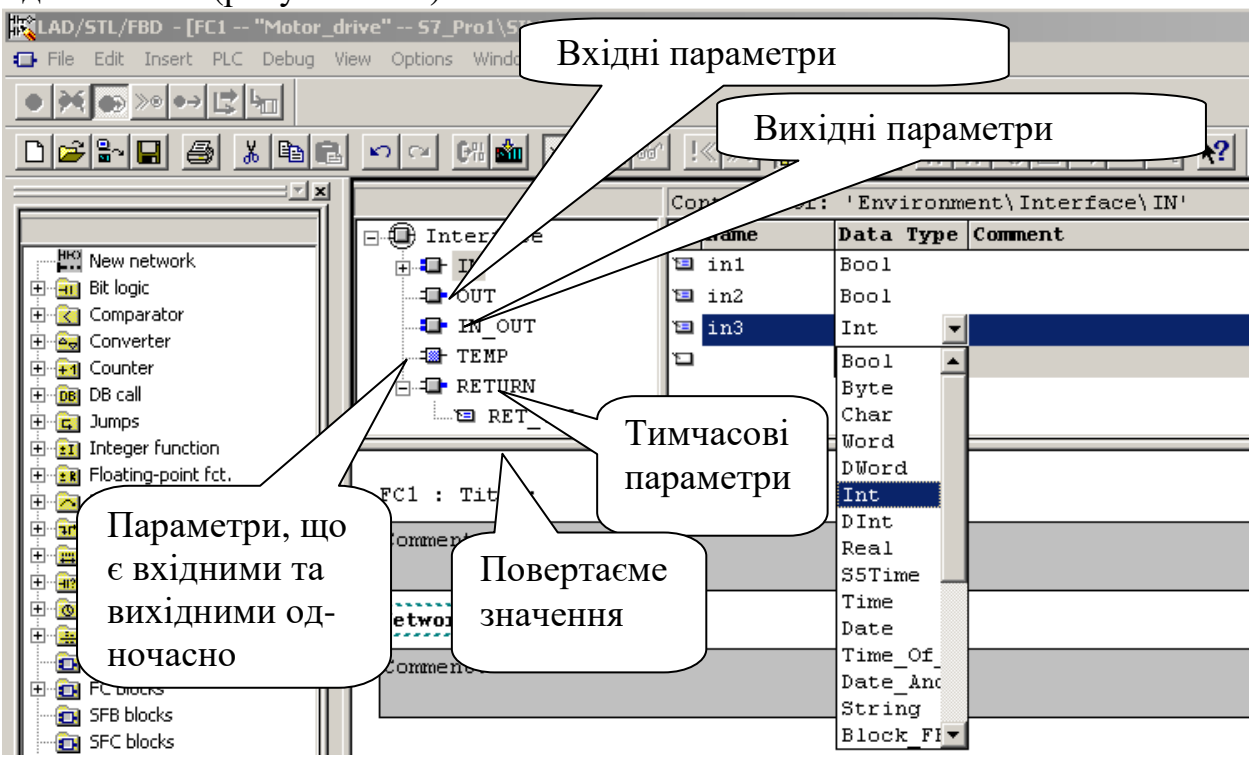


Рисунок 14.4 – Вікно редактору функції

У вкладці IN вводять вхідні параметри функції. Їх значення змінювати у тексті програми функції не дозволяється. У вкладці OUT вводять вихідні параметри. Значення цих параметрів не можливо опитувати. У вкладці IN_OUT вводять параметри, що є одночасно і вхідними і вихідними. Їх можна і опитувати і змінювати у тексті програми функції. Параметри TEMP створюють, щоб зберігати в них проміжні дані під час виконання функції. Коли функція має лише одне значення, що повертається, то його частіше за все пишуть у RET_VAL, хоча ніякої різниці між цим параметром та параметрами з вкладки OUT не існує.

Для створення параметрів слід обрати відповідну вкладку. Так на рисунку 14.4, обравши вкладу IN створюємо три вхідні параметра: in1,in2 типу Bool та in3 типу Int. Також створимо один вихідний параметр з ім'ям result типу Real.

При використанні параметрів у програмі функції перед їх назвою обов'язково ставиться символ #.

Розробимо програму мовою LAD, котра виконує множення вхідного параметра in3 на константу 3.2, коли на двох логічних входах in1, in2 функції логічна «1». Розроблену програму наведено на рисунку 14.5

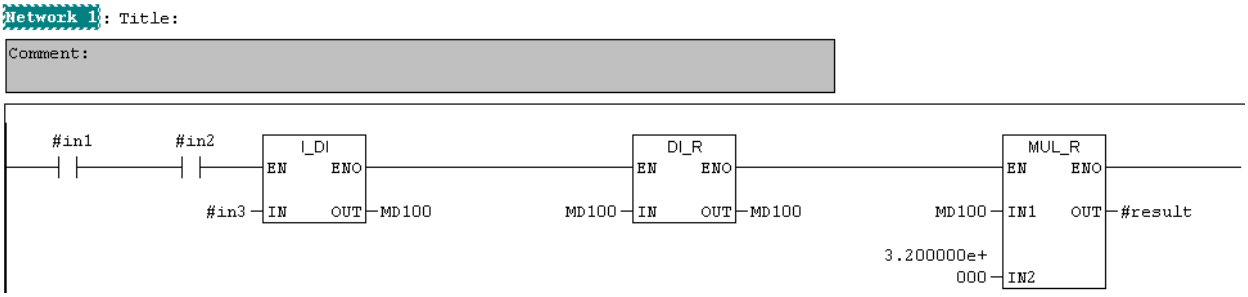


Рисунок 14.5 – Програма функції FC1.

Відповідно до програми спочатку перевіряється стан вхідних логічних параметрів in1, in2. Коли їх стан – логічна «1», то виконується послідовно три операції:

1) перетворення типу вхідного параметра in3 з Int в Dint (команда I_DI);
2) подальше перетворення формату з Dint в Real (команда DI_R);
3) множення результату перетворення на константу 3.2 (команда MUL_R) та передання результату у вихідний параметр result.

Слід звернути увагу, що для проміжних результатів у програмі використовувалося подвійне слово маркерної пам'яті MD100, а не тимчасовий параметр, який можна було зробити. Річ у тому, що тип тимчасової змінної не можна змінювати і, у наведеній на рисунку 14.5 програмі, було б необхідно дві тимчасові змінні (одна типу Dint, друга – Real) замість одного подвійного слова маркерної пам'яті MD100, у якому можуть зберігатися дані обох цих типів.

Таж програма функції FC1, але мовою STL має наступний вигляд.

```
A     #in1 //Логічна операція «ТА» з параметрами
A     #in2 //in1 та in2
JCN   m1 //Якщо обидва параметри in1 та in2 не дорівнюють«1», то
         //перехід на метку m1
L     #in3 //Завантаження до акумулятора значення параметра in3
ITD //Перетворення типу даних завантаженої змінної у DInt
DTR //Перетворення типу даних з DInt до Real
L   3.200000e+000 //Завантаження до акумулятора константи 3.2
*R //Множення константи та вхідного параметра in3
T     #result //Передання результату у вихідний параметр result
m1:NOP 0 //Мітка, на яку переходимо, якщо in1 та in2 не дорівнюють «1»
```

По закінченню написання функції слід зберегти її, натиснувши кнопку у панелі інструментів. Для перевірки роботи функції з використанням симулятора її необхідно завантажити до нього, натиснувши кнопку у панелі ін-

струментів. Автоматично з виконанням операції збереження, ім'я функції з'являється у вікні бібліотеки будь-якої мови програмування (рисунок 14.6).

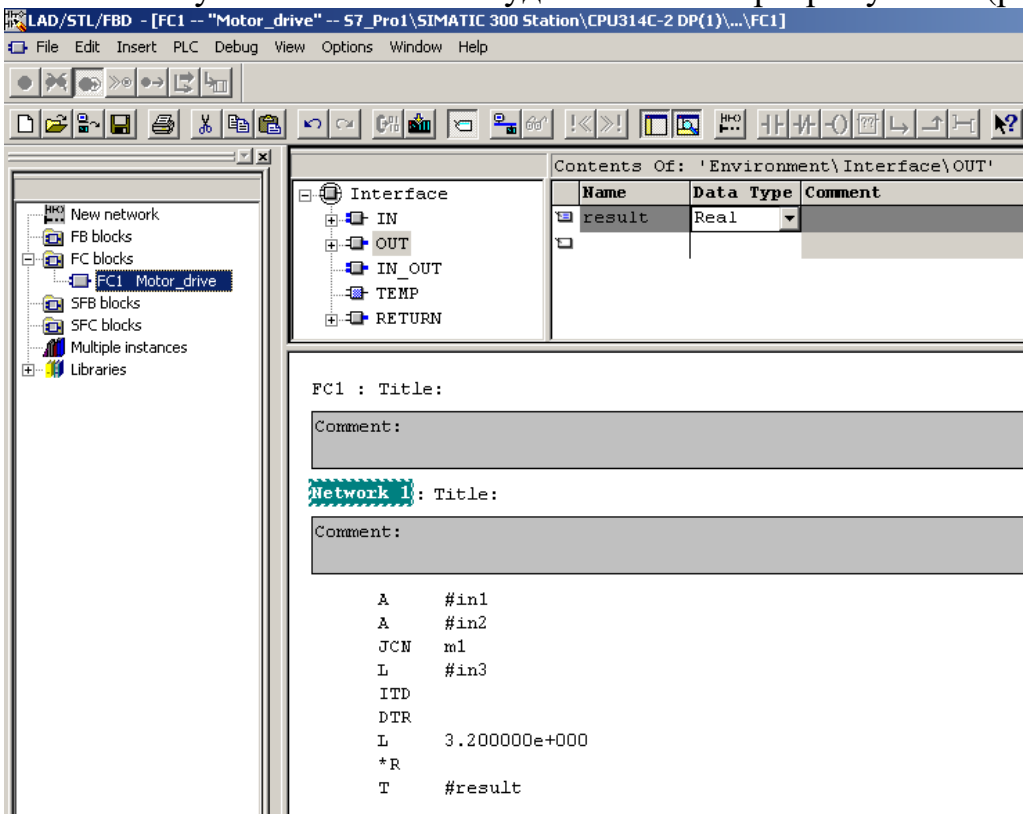
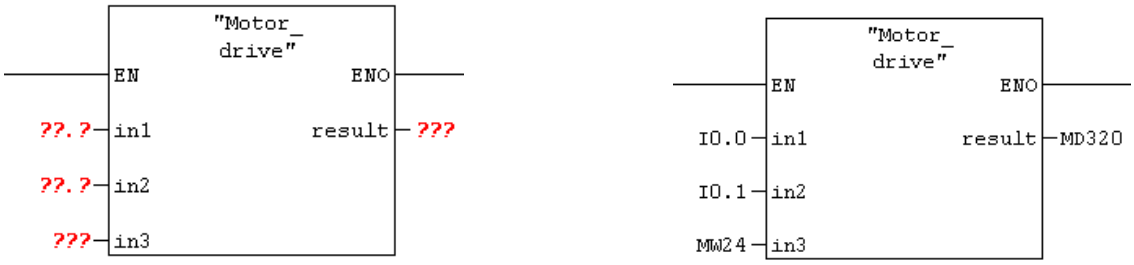


Рисунок 14.6 – Доступ до розробленої функції з бібліотеки

Для виклику розробленої функції з головної програми чи будь-якої іншої підпрограми достатньо перетягнути її з бібліотеки до самої програми. Виклик розробленої нами функції у головній програмі мовою LAD буде наступним



а б

Рисунок 14.7 – Загальний вигляд виклику функції FC1 (ім 'я Motor drive)

На рисунку 14.7а бачимо, який вид має функція після розміщення у програмі. Наявність знаків питання говорить, що для вхідних і вихідних параметрів функції обов'язково потрібно вказати адреси входів, виходів чи вічків оперативної пам'яті. На рисунку 14.7б зображено виклик функції з заповненими параметрами (параметру in1 присвоюється значення дискретного сигналу з входу I0.0 і т.д.). При виклику функції мовою LAD у неї завжди є вхід дозволу виконання (EN) та вихід (ENO). Сигнал логічної «1» на вході EN дозволяє виконання функції та автоматично передається на вихід ENO, дозволяючи виконання наступних за функцією команд у рядку програми.

При виклику функції мовою STL використовується команда CALL з номером функції. При перетягуванні функції з бібліотеки мишкою у текст програми виклик записується автоматично. При виклику нашої функції FC1 мовою STL, назви вхідних параметрів будуть автоматично виділені червоним кольором, що означає обов'язкове заповнення параметрів виклику функції.

```
CALL  "Motor_drive"
 in1   :=I0.0 //Сигнали, за якими дозволяється розрахунок у функції
 in2   :=I0.1 // Сигнали, за якими дозволяється розрахунок у функції
 in3   :=MW24 //З MW24 беремо слово для множення
 result:=MD320 //Результат записується у подвійне слово MD320
```

Слід пам'ятати, що при зміні кількості чи типів вхідних, вихідних параметрів функції, усі виклики, котрі були записані раніше в програмі, автоматично не зміняться. Робота контролера зупиниться (він перейде в режим Stop). Тому необхідно всі виклики функції видалити з програми і знов записати виклики цієї функції.

Крім команди CALL існують ще дві команди виклику функції:
CC – виклик функції при умові, що результат попередньої логічної операції дорівнює «1», тобто, що значення біта RLO дорівнює;
UC – безумовний виклик функції.

Але ці команди можна використовувати лише для функцій та функціональних блоків, що не мають вхідних та вихідних параметрів.
Приклад виклику функції командою CC

```
A I0.0 //Опитуємо стан вхідного сигналу I0.0
CC FC2 //Викликаємо функцію FC2 без параметрів, якщо стан вхідного сигнала I0.0 – логічна «1»
```

В функціях та функціональних блоках дуже часто використовуються програмні розгалуження, коли кожна з гілок розгалуження є закінченням підпрограми. Але обидві гілки – це послідовно записані частини підпрограми. Коли виконана одна з них, не повинна виконуватися інша. У таких випадках дуже зручно використовувати команди закінчення блоку програми.

Команда BEU негайно завершує виконання програми у тому місці де вона розташована, незважаючи на наявність за нею інших інструкцій. Це команда безумовного завершення підпрограми.

Команда BEC завершує виконання програми у тому місці де вона розташована, якщо значення біта RLO дорівнює «1». У цей біт записується результат логічних операцій, порівняння і т.і. Це команда умовного завершення підпрограми.

Прикладом використання цих команд може бути перероблена модифікація раніше розробленої нами програми функції, у котрій розгалуження реалізоване інструкцією умовного переходу та інструкцією NOP 0. У оновленій програмі, коли результат логічної операції вхідних параметрів in1 та in2 дорівнює

1, виконується інструкція умовного кінця блока BEC і подальше перемноження in3 на константу не відбувається.

```
AN    #in1 //Логічна операція «І» з інверсними значеннями параметрів
AN    #in2 //in1 та in2
BEC //Якщо обидва параметри in1 та in2 не дорівнюють«1», то
        //закінчення програми
L     #in3 //Завантаження до акумулятора значення параметра in3
ITD //Перетворення типу даних завантаженої змінної у DInt
DTR //Перетворення типу даних з DInt до Real
L   3.200000e+000 //Завантаження до акумулятора константи 3.2
*R //Множення константи та вхідного параметра in3
T     #result //Передання результату у вихідний параметр result
```

Порівнюючи мови LAD та STL при розробці функцій ми бачимо, що створення вхідних та вихідних параметрів функцій, їх використання не відрізняються у обох мовах. Але при реалізації розгалужень у тілі функції мова STL є більш гнучкою, бо в неї є команди BEU та BEC, які дозволяють ефективно закінчувати гілки програми. Мовою LAD це можна зробити теж, але об'єм програми збільшується.

При виклику функції переваг в використанні мов LAD та STL не існує.

Контрольні питання

1) Що таке функція? Які два види функцій існують? Чи існують обмеження на кількість функцій та їх об'єм у контролерах Siemens?

2) Як створити функцію?

3) Що таке вхідні та вихідні параметри функцій? Як створити ці параметри та як змінювати значення параметрів в програмі функції?

4) Як викликати функцію з головної програми мовами LAD та STL?

5) Які існують команди завершення програми у мові STL?

Контрольні завдання

Задача 14.1. Розробити функцію, що знаходить середнє арифметичне трьох вхідних параметрів та видає результат у вихідному параметрі. Вхідні параметри – це три слова з маркерної пам'яті, вихідний параметр – речовинне значення, що знаходиться у подвійному слові маркерної пам'яті.

Задача 14.2. Розробити функцію, що формує сигнал трапецеїдальної форми на аналоговому виході PQW752 при натисканні кнопки, приєднаної до цифрового входу I124.0. Тривалість фронтів сигналу 3,2 с, тривалість плоскої частини сигналу 8,6 с.

15 Функціональні блоки

Функціональні блоки, як і функції, – це підпрограми, що можна викликати з головної програми чи іншої підпрограми. Максимальна глибина вкладання функціональних блоків дорівнює 8 для контролерів серії S7-300 та 24 для контролерів серії S7-400.

Основна відмінність функціонального блока від функції полягає в тому, що кожному з функціональних блоків повинен відповідати хоча б один залежний від нього блок даних (Instance DB). У цьому залежному блоці даних знаходяться значення усіх вхідних та вихідних параметрів функціонального блоку. Завдяки цьому функціональний блок пам'ятає значення усіх параметрів від попереднього виклику і при наступному його виклику, не обов'язково задавати значення параметрів, що не змінилися. Якщо деякі значення параметрів не задали, то будуть узяті попередні. Така особливість функціональних блоків дуже зручна при програмній реалізації: контурів регулювання, рекурсивних розрахунках і т.і., коли значення процесу на теперішній момент часу визначаються з попереднього стану системи та результатів нових вимірювань.

В мовах програмування контролерів Siemens існують два різновиди функціональних блоків: системні та користувацькі. Системні функціональні блоки – це підпрограми, що розроблено компанією Siemens і входять до складу бібліотек. Ці бібліотеки є складовою частиною середовища програмування Step7. Системні функціональні блоки мають назву SFB з номером.

Користувацькі функціональні блоки розробляє автор програми и користується ними у подальшому. Вони мають назву FB з номером. Кількість функціональних блоків, що можна розробити, обмежується можливостями контролера. Для контролерів серії S7-300 їх не може бути більше ніж 2048, а для контролерів серії S7-400 їх не може бути більше 6144. Користувацькі функціональні блоки можуть бути з параметрами – вхідними та вихідними чи без них.

Розглянемо процес створення користувацького функціонального блока.

Для створення функціонального блоку необхідно у головному вікні проекту обрати вкладку Blocks. Після клацання правою кнопкою миші в полі блоків у спливаючому меню виберіть пункт InsertNewObject/Function Block.

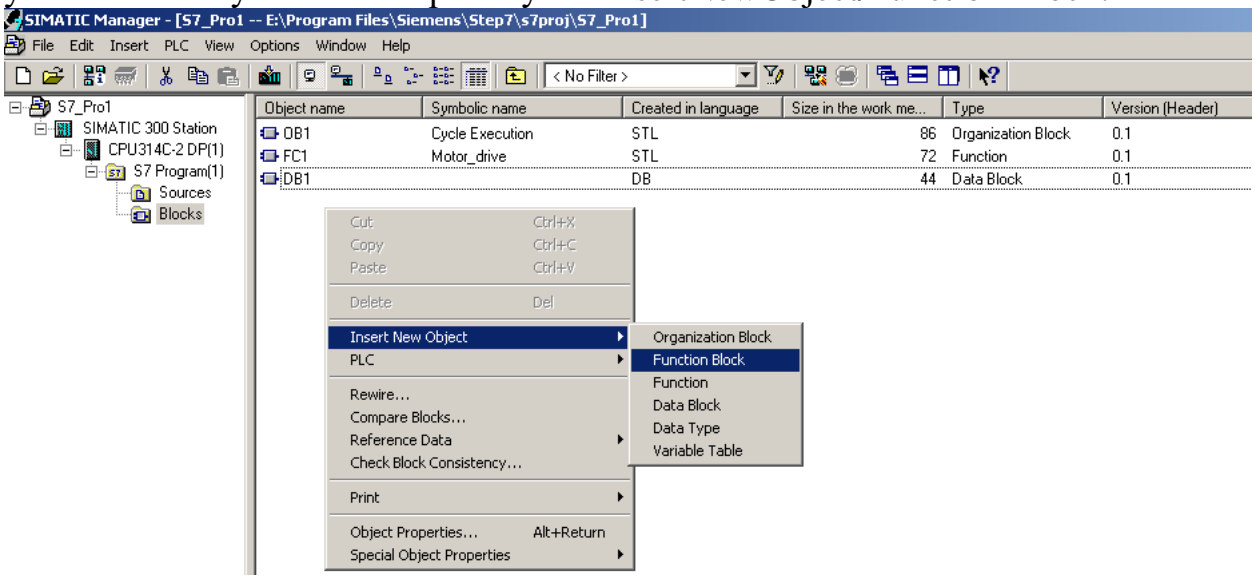


Рисунок 15.1 – Меню створення функціонального блока

Далі з’являється графічне вікно створення нового функціонального блока (рисунок 15.2)

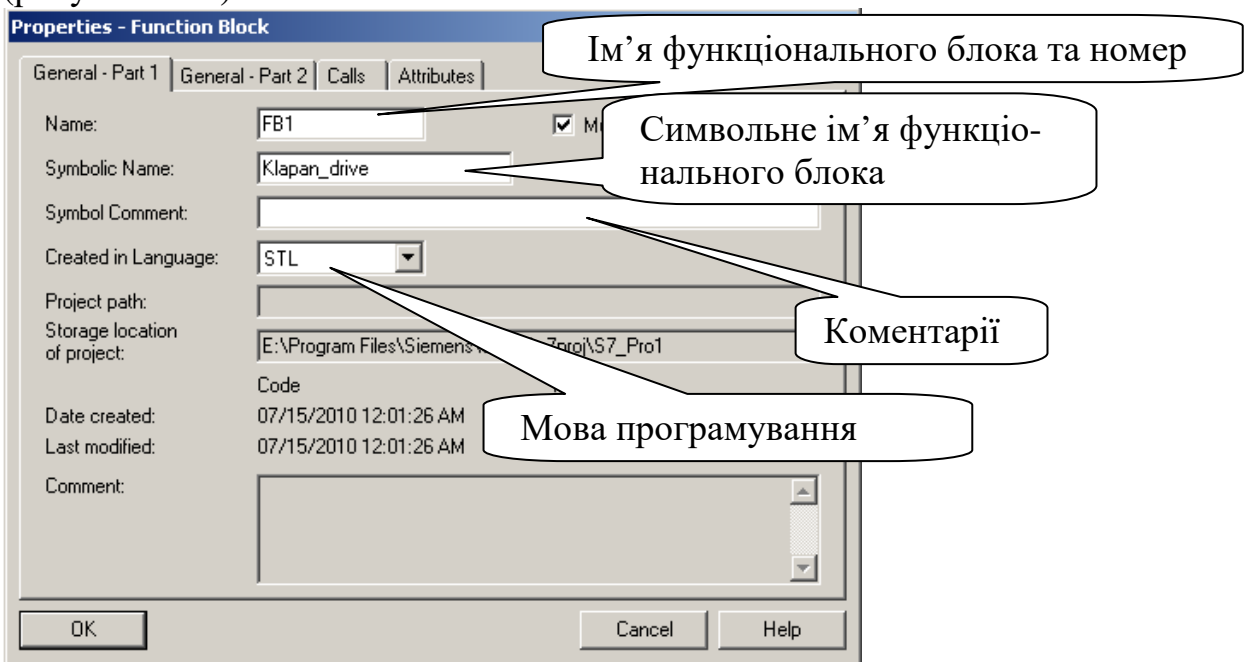


Рисунок 15.2 – Графічне вікно створення нового функціонального блока

Обов’язковим є ввод імені функціонального блока FB та номера. Нумерація починається з 1. Символьне ім’я дозволяє ввести і в подальшому використовувати зручне для розуміння програміста ім’я функціонального блока. Якщо у наведеному на рисунку 15.2 прикладі функціональний блок FB1 має символьне ім’я Klapan_drive, то і у тексті програми замість FB1 буде автоматично записано Klapan_drive. Ввід символьного імені не обов’язковий. У тексті програми функціональні блоки, що не мають символьного ім’я, будуть записані, як FB1, FB2 і т.і. У полі Symbol Comment можна ввести коментарії рідною мовою. Мову програмування можна обрати одразу чи змінити у подальшому. Для закінчення формування функціонального блоку слід натиснути кнопку OK. Тепер новий функціональний блок з’являється у папці Bloks головного вікна проекту.

Далі слід створити хоча б один блок даних (Instance DB), залежний від FB1. Для створення блоку даних необхідно у головному вікні проекту обрати вкладку Blocks. Після щиглика правою кнопкою миші в полі блоків у спливаючому меню виберіть пункт InsertNewObject/Data Block.

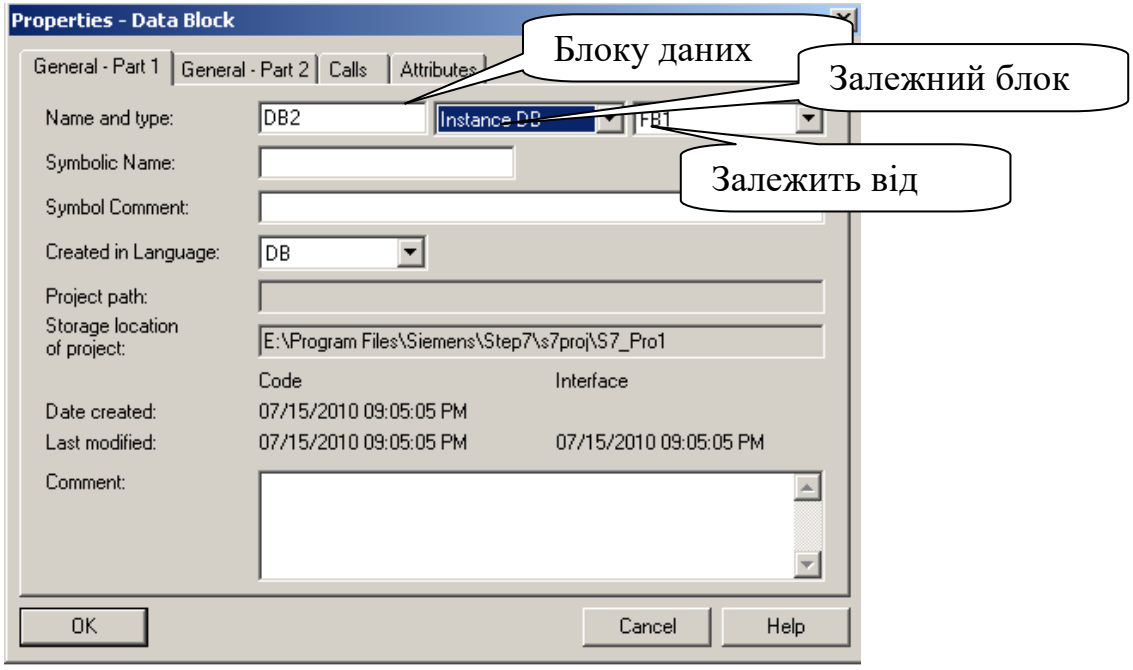


Рисунок 15.3 – Створення залежного блоку даних

У зображеному на рисунку 15.3 вікні в полі Name слід ввести номер блоку даних. Далі необхідно обрати тип блоку даних InstanceDB та вказати функціональний блок, від якого залежить цей блок даних. Для закінчення формування блока даних натисніть кнопку ОК. Разом з цією дією автоматично буде створено структуру блока даних, відповідну переліку вхідних та вихідних параметрів функціонального блока.

По закінченню створення функціонального блоку він з'являється у папці Blocks головного вікна проекту. Тепер його можна відкрити, створити вхідні і вихідні параметри, написати текст програми. Ці дії цілком аналогічні відповідним при створенні функції і розглянуті у розділі 14.

Для закріплення матеріалу у функціональному блоці FB1 самостійно слід створити вхідні параметри InA, InB та вихідний OutX типу Int, написати програму складання вхідних параметрів с занесенням результату у OutX.

По закінченню написання функціонального блоку слід зберегти його, натиснувши кнопку  у панелі інструментів. Для перевірки роботи з використанням симулятора, функціональний блок необхідно завантажити до нього, натиснувши кнопку  у панелі інструментів. Після збереження розробленої програми функціонального блоку він автоматично з'являється у бібліотеці функціональних блоків. Для його виклику будь-якою мовою достатньо перетягнути його з бібліотеки у програму.

Приклад виклику такого функціонального блоку мовою LAD наведено на рисунку 15.4.

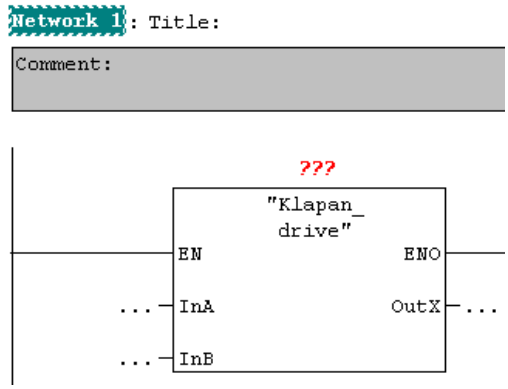


Рисунок 15.4 – Виклик функціонального блоку FB1 мовою LAD

При виклику FB1 обов'язково треба ввести замість знаків питання номер залежного блоку даних. Подавати на входи дані, чи зчитувати результат з виходу необов'язково. Приклад програми з подвійним викликом такого функціонального блоку і результатами виконання наведено на рисунку 15.5

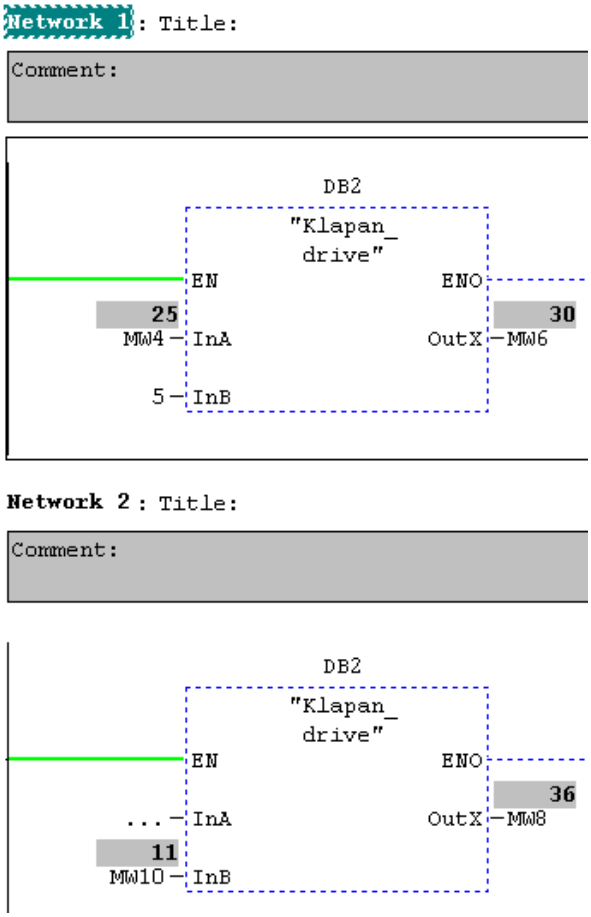


Рисунок 15.5 – Приклад програми з подвійним викликом функціонального блоку

При першому виклику FB1 складалося значення слова MW4, що дорівнювало 25, з константою 5. При наступному виклику складалося слово MW10 із попереднім значенням на вході InA завдяки використанню залежного блока даних DB2.

Приклад цієї ж програми мовою STL наведено нижче

```
CALL  "Klapan_drive" , DB2 //Перший виклик FB1
 InA :=MW4
 InB :=5
 OutX:=MW6
CALL  "Klapan_drive" , DB2 //Другий виклик FB1
 InA :=
 InB :=MW10
 OutX:=MW8
```

При розробці функціональних блоків істотних переваг в використанні мов LAD чи STL не має крім команд BEU та BEC, як і при розробці функцій.

Контрольні питання

1) Що таке функціональний блок? Чим він відрізняється від функції?

2) Як викликати функцію з основної програми мовами LAD та STL

3) Чи існують обмеження на кількість та об'єм функціональних блоків в контролерах Siemens

Контрольні завдання

Задача 15.1. Розробити функціональний блок, що вимірює період імпульсів на цифровому вході контролера. Значення періоду не перевищує 5с. Адреса входу, на який подаються імпульси є вхідним параметром функціонального блоку, а результат вимірювання – вихідний параметр.

Задача 15.2. Розробити функціональний блок, що реалізує фільтр ковзного середнього за чотирма відліками вхідного сигналу з аналогового входу PIW752. Значення амплітуди нового відліку та попередні повинні зберігатися у залежному блоці даних. Виклик функціонального блоку виконується кожні 300 мс. При виклику до функціонального блоку передається нове значення амплітуди сигналу з аналогового входу. Вихідний параметр функціонального блоку – результат фільтрації (середнє арифметичне).

16 Непряма адресація

У попередніх главах цього навчального посібника в усіх прикладах програм використовувалася пряма адресація, тобто при звертанні до входів, виходів, маркерної пам'яті чи блоків даних вказувався тип пам'яті та номер того вічка пам'яті, до якого зверталися. Наприклад: I0.1 – звертання до першого біта нульового байта області входів; MW10 – звертання до десятого слова меркерної пам'яті; DB1.DBD4 – звертання до четвертого подвійного слова у першому блоці даних. Перевагами прямої адресації є: простота написання та розуміння програми, використання в усіх мовах. Але існує багато практичних задач, у яких можливостей прямої адресації недостатньо. Наведемо приклад. Нехай тре-

бу керувати декількома однотипними клапанами. Коли пристрої, якими керують, однотипні, то і програма керування одна. Тому програму слід написати у вигляді окремої функції, котру будемо визивати з головної програми стільки разів, скільки є клапанів. Але при написанні такої програми ми стикаємося з такою проблемою – кожен з клапанів підключено до відповідних входів та виходів контролера, тому одна і та ж команда підпрограми повинна звертатися до різних входів чи виходів керуючи ними. У таких випадках використовуються методи непрямої адресації у мові STL. Непряма адресація дає можливість звернутися до адрес, котрі невідомі до початка виконання програми. У мовах LAD та FBD ці методи не працюють.

Сутність методів непрямої адресації полягає у тому, що адреса пам’яті у межах однієї області пам’яті може бути змінною. Тобто у одному рядку програми ми можемо звертатися до різних входів, у другому до різних виходів, у третьому до різних частин маркерної пам’яті. Але при використанні непрямої адресації у одному рядку програми не може змінюватися тип даних, до яких ми звертаємося. Це означає що у одному рядку програми не можна спочатку звернутися до байта, при наступному виконанні програми до слова, а потім до подвійного слова. Тобто тип даних є незмінним, а змінюється тільки адреса (номер вічка).

Методи непрямої адресації поділяються на адресацію через пам’ять та регістрову адресацію.

16.1 Непряма адресація через пам’ять.

При непрямій адресації за допомогою пам'яті адреса вказується за допомогою адресованого вічка пам'яті. Адреса повинна мати розмір подвійного слова, якщо потрібно використовувати покажчик на область, або ж він повинен мати розмір слова (WORD), якщо потрібно при непрямій адресації використовувати число, як покажчик. Адреса операнда може перебувати в одній з нижче перерахованих адресних областей:

1. область маркерної пам’яті, як абсолютна адреса або символьна змінна;
2. L-Стік (область тимчасових локальних даних), як абсолютна адреса або символьна змінна;
3. блок глобальних даних як абсолютна адреса; при використанні адресації глобальних даних користувач повинен забезпечити, щоб необхідний блок даних був відкритий з допомогою DB-регістра; якщо, наприклад, Ви вказуєте адресу адреси глобальних даних непрямим способом за допомогою подвійного слова в глобальних даних, то всі операції (обчислення адреси) повинні бути зроблені з обліком того, що всі дані перебувають в одному й тому ж блоці даних.
4. екземплярний блок даних, як абсолютна адреса або символьна змінна; існують певні обмеження при використанні екземплярних даних, як адреси.

Якщо Ви використовуєте екземплярні дані, як адреси в функціях, оброб-

ляйте їх точно в такий же спосіб, як і адреси глобальних даних; при цьому Ви повинні використовувати DI-регістр замість DB-регістра. Символьна адресація не допускається в цьому випадку.

16.1.1 Непряма адресація з покажчиком на область

Покажчик на область використовуваний для непрямої адресації за допомогою пам'яті, завжди є внутрізонним покажчиком. Це означає, що він містить адресу байта і адресу біта. Будь-які два суміжних біта завжди відрізняються адресою на 1. Два суміжних байта завжди відрізняються адресою на 8, два слова на 16, два подвійних слова на 32. При цьому в кожній з областей пам’яті контролера при непрямій адресації біти, байти, слова, подвійні слова починають рахувати з нуля. Наприклад адреса входу I0.0 дорівнює нулю, так саме, як і адреса виходу Q0.0, біта меркерної пам’яті M0.0, а також любого початкового біта у будь-якому блоці даних. Адреса байта IB0 дорівнює нулю, IB1 – восьми. Адреса слова IW0 дорівнює нулю, IW2 –16; адреса ID0 дорівнює нулю, ID4 – 32.

При реалізації метода непрямої адресація з покажчиком на область використовується наступний формат команди доступу до даних за адресою

Команда Область пам’яті[*подвійне слово з адресою*]

Для полегшення завантаження подвійних слів-констант до подвійних слів, що містять адресу, використовується наступний формат констант

P#*Кількість_байт.Кількість_біт*

Значення *Кількість_байт* обмежується розмірами області пам’яті чи блока даних, а значення *Кількість_біт* знаходиться в межах від 0 до 7.

Розробимо програму, котра виконує операцію логічного «ТА» з логічними сигналами з двох входів, результат передається на вихід. Адреси двох входів та виходу є змінними, тобто задаються за допомогою непрямої адресації. Для їх збереження використаємо відповідно три подвійних слова маркерної пам’яті MD100, MD104, MD108. У даній програмі завантажимо до подвійних слів адреси у вигляді констант. Наведемо та пояснимо текст програми

```
L P#20.0 //Завантаження до MD100 адреси нульового біта двадцятого
T MD100 //байта
L P#20.3//Завантаження до MD104 адреси третього біта двадцятого
T MD104 //байта
L P#1.6 //Завантаження до MD108 адреси шостого біта першого
T MD108 //байта
……..
A I [MD100] //Зчитування логічного стану з входу I 20.0
A I [MD104] //Зчитування логічного стану з входу I 20.3 та виконання ло-
```

гічної операції «ТА»

= Q[MD108] //*Передання результату логічної операції на вихід Q1.6*

Розробимо ще одну програму, у якій виконується складання двох слів з маркерної пам'яті, адреси котрих є змінними та знаходяться у подвійних словах маркерної пам'яті MD100, MD104. Адресу у перше подвійне слово завантажимо у вигляді константи, а при завантаженні у друге – розрахуємо. При цьому адреса другого слова буде відрізнятися від адреси першого на 32, що відповідає зміщення у адресному просторі на 4 байти. Результат розрахунку завантажимо до слова MW50. Наведемо та пояснимо текст програми

```
L P#16.0 // Завантаження до MD100 адресу шістнадцятого байта
T MD100
L 32 //Додаємо до адреси число 32. Результат 128+32=160
+I //(16 байт + 4 байти = 20 байт)
T MD104 //Розраховану адресу завантажуємо до MD104
……
L MW[MD100] //Завантажуємо до акумулятора значення слова MW16
L MW[MD104] //Завантажуємо до акумулятора значення слова MW20
+I //Додаємо значення двох слів MW16 та MW20
T MW50 //Результат завантажуємо до MW50
```

Розглянемо тепер на прикладах, як працювати з інформацією у блоках даних за допомогою непрямої адресації. У першому прикладі зчитаємо до акумулятора слово з першого глобального блоку даних, коли адреса цього слова знаходиться у подвійному слові меркерної пам'яті MD60.

```
L P#2.0 // Завантаження до MD60 адресу другого байта
T MD60 //
…….
OPN DB1 //Відкриваємо блок DB1
L DBW[MD60] //Зчитуємо з відкритого блоку DB1 слово, адреса якого
знаходиться у подвійному слові маркерної пам'яті MD60
```

У даному прикладі слід звернути увагу на те, що при використанні методів непрямої адресації завжди необхідно використовувати команду відкриття блоку даних OPN DB… перед тим, як працювати з його записами. Команди у вигляді L DB1.DBW[MD60] не допустимі.

У другому прикладі зчитаємо до акумулятора слово з маркерної пам'яті, коли його адреса знаходиться у нульовому подвійному слові глобального блоку даних DB1.

```
L P#40.0 // Завантаження до DB1.DBD 0 адресу сорокового байта
T DB1.DBD 0
……
```

OPN DB1 //*Відкриваємо блок DB1*
L MW[DBD0] //*Зчитуємо з MW40 дані*

Дивлячись на наведений приклад слід запам'ятати, що коли адреса знаходиться у глобальному блоці даних, то вона не може бути адресою вічка з іншого блока даних, тому що у програмі одночасно може бути відкритим лише один блок даних.

Дуже часто методи непрямої адресації використовуються у функціях чи функціональних блоках. Якщо, наприклад, функція керує декількома двигунами, то при кожному виклику такої функції (керуванні якимось двигуном) у вхідних параметрах треба вказувати входи контролера, через які він опитує приєднані до двигуна датчики та виходи, через які контролер керує роботою двигуна. Щоб керувати виходами та опитувати входи у даному випадку, необхідно використовувати методи непрямої адресації. Але така реалізація програми функції має деякі особливості. Розглянемо їх на прикладі.

Створимо функцію, що повинна зчитувати з аналогового входу значення напруги та порівнювати його з порогом. Якщо зчитана напруга менш ніж порогове значення, то на цифровий вихід подаємо логічний «0». В іншому випадку подаємо логічну «1». Адреси аналогового входу та цифрового виходу є вхідними та вихідними параметрами функції, тобто змінними. Реалізувати цю функцію можливо тільки з використанням методів непрямої адресації. Спочатку треба створити функцію (наприклад FC1), виконавши послідовність дій, наведену у розділі 14. У функції створимо вхідний параметр IN_analog та вихідний OUT_digit типу подвійне слово. Тип подвійне слово обрано тому, що обидва параметри містять 32-х розрядні адреси входу та виходу. Також вхідним параметром буде Porog, що містить порогове значення для порівняння. Основною особливістю методів непрямої адресації у функції є неможливість вводу традиційних записів типу

L MW[#IN_analog].

Вони не будуть сприйматися компілятором та недопустимі для вводу. Є можливість обійти цю проблему створивши допоміжні внутрішні змінні у розділі змінних TEMP (див. рисунок 16.1). Створимо дві такі змінні з назвами temp_an та temp_dig. Створені змінні та сама програма зображені на рисунку 16.1

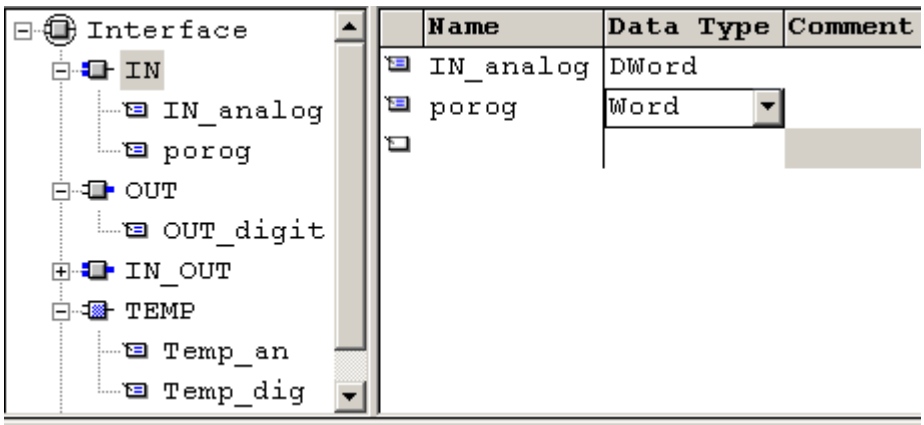


Рисунок 16.1 – Програма функції із використанням непрямої адресації

```
L    #IN_analog //Зчитування до акумулятора значення вхідної змінної
                  IN_analog
T    #Temp_an //Завантаження цього значення до тимчасової змінної
                Temp_an
L    #OUT_digit // Зчитування до акумулятора значення вхідної змінної
                  OUT_digit
T    #Temp_dig // Завантаження цього значення до тимчасової змінної
                 Temp_dig
L    PIW [#Temp_an] //Завантаження поточного значення з аналогового
                     входу, що визначається адресою у вхідному параметрі
L    #porog //Завантаження порогу
>I   //порівняння зчитаного з аналогового входу кода з пороговим
=    Q [#Temp_dig] //Передача на цифровий вихід логічної «1», коли
                    зчитане з аналогового входу значення більше порога
```

З наведеного прикладу можемо зробити висновок. Для використання значення вхідної змінної у якості адреси треба скопіювати це значення у внутрішню тимчасову змінну. Тільки її можна використовувати у якості зберігача адреси.

Те ж саме стосується і відкриття блоків даних, коли номер блока є вхідним параметром для функції чи функціонального блоку. Спочатку треба створити тимчасову змінну типу слова, далі завантажити до неї значення з вхідної змінної, а потів використовувати при відкритті блока даних.

16.1.2 Непряма адресація з використанням номера

Методами непрямої адресації можуть задаватися не лише адреси записів у блоках даних, а й номера блоків даних при їх відкритті. Для збереження номера блоку даних використовуються слова маркерної пам’яті, а не подвійні слова. В таких словах маркерної пам’яті повинні знаходитися цілі беззнакові числа від 1 до максимального, що визначається максимальною кількістю блоків даних для даної модифікації контролера. Мінімальне число 1 визначається ну-

мерацією блоків даних, що починається з 1, а не з 0. Формат команди відкриття блока даних з використанням методів непрямої адресації

OPN DB [*Слово_меркерної_пам'яті*]

Розглянемо приклад програми відкриття першого блоку даних з використанням методів непрямої адресації. Номер блока даних будемо зберігати у слові маркерної пам'яті MW 60.

```
L 1 //Завантаження до MW 60 константи 1, що є номером блока даних,
T MW 60 //котрий відкривається пізніше
……
OPN DB[MW60] //Відкриття першого блока даних
```

За допомогою непрямої адресації можна також задавати номер таймера, що запускається, чи визивати функцію. При цьому, як і при завданні номера блока даних, що відкривається, номер таймера чи номер функції повинен знаходитися в слові маркерної пам'яті.

Приклад запуску таймера

```
L 2 //Завантаження до MW140 номера таймера, що буде запущений
T MW140
……
A I 0.2 //Опитування входу, сигнал якого запустить таймер
L S5T#3s300ms //Визначення часу роботи таймера
SP T[MW140] //Запуск таймера
```

Викликання функції непрямою адресацією аналогічне. Слід лише пам'ятати, що для виклику функції можна використовувати лише команди UC або CC, а отже можна викликати лише функції без вхідних та вихідних параметрів.

Зараз були розглянуті приклади роботи з глобальними блоками даних з використанням методів непрямої адресації. Робота з залежними блоками даних ні чим не відрізняються від розглянутих прикладів.

16.2 Регістрова адресація

К контролерах Siemens існує два 32-розрядних регістри, котрі призначені для реалізації другого методу непрямої адресації. Ці регістри мають назви AR1 та AR2. При використанні вони цілком рівнозначні. Сутність регістрової адресації полягає в тому, що адреса вічка, до якого треба звернутися, є сумою адрес в одному з двох адресних регістрів та константи зсуву. Дані в адресних регістрах є змінними. Формат команд при використанні регістрової адресації наступний

Команда тип_пам'яті_та_тип_даних[AR1,P#*кількість_байт.кількість_біт*]
Команда тип_пам'яті_та_тип_даних[AR2,P#*кількість_байт.кількість_біт*]

Константа зсуву

Для завантаження даних до адресного регістра чи зміни його вмісту використовується декілька команд. Усі вони працюють з адресними регістрами та першим акумулятором. Наведемо перелік команд з їх описанням

Команда	Опис команди
LAR1	Дані з першого акумулятора завантажуються до першого адресного регістра
LAR2	Дані з першого акумулятора завантажуються до другого адресного регістра
TAR1	Дані з першого адресного регістра завантажуються до першого акумулятора
TAR2	Дані з другого адресного регістра завантажуються до першого акумулятора
+AR1	Складується вміст першого адресного регістра та першого акумулятора. Результат завантажується до першого адресного регістра.
+AR2	Складується вміст другого адресного регістра та першого акумулятора. Результат завантажується до другого адресного регістра.

Наведемо декілька прикладів програм, де використані методи регістрової адресації. Розробимо програму, що виконує логічну операцію «АБО» з двома входами, адреси яких завантажені до адресних регістрів.

```
L P#5.0 //Завантаження до AR1 константи, що відповідає адресі
LAR1 // нульового біта п'ятого байту
L P#5.1 //Завантаження до AR2 константи, що відповідає адресі
LAR2 // першого біта п'ятого байту
……
O I[AR1, P#0.0] //Виконання операції логічного «АБО»
O I[AR2, P#0.0] //Виконання операції логічного «АБО»
= Q4.0 //Передання результату логічної операції на вихід
```

В наведеній програмі слід звернути увагу, що адресна константа зсуву дорівнює нулю (P#0.0), бо потреби в неї не має. Фактично результуюча адреса повністю задається вмістом адресного регістра (AR1 або AR2).

Розробимо приклад фрагмента програми, що копіює три слова з однієї частини маркерної пам'яті в іншу, коли початкова адреса першої частини є змінною, а зсув між частинами пам'яті є постійним і дорівнює 50 байтам.

```
L P#40.0 //Завантаження до AR1 константи, що відповідає адресі
LAR1 // 40-го слова
…….
L MW[AR1,P#0.0] //Зчитування даних з 40-го слова маркерної пам'яті
T MW[AR1,P#50.0] //Завантаження даних у 90-е слово маркерної пам'яті
L P#2.0 //Додання до вмісту AR1 константі в одне слово (2 байти)
+AR1
L MW[AR1,P#0.0] //Зчитування даних з 42-го слова маркерної пам'яті
T MW[AR1,P#50.0] //Завантаження даних у 92-е слово маркерної пам'яті
L P#2.0 //Додання до вмісту AR1 константі в одне слово (2 байти)
+AR1
L MW[AR1,P#0.0] //Зчитування даних з 44-го слова маркерної пам'яті
T MW[AR1,P#50.0] //Завантаження даних у 94-е слово маркерної пам'яті
```

Розглянемо приклад роботи з блоками даних. Номер блока даних та адреса зчитаного з нього подвійного слова є змінними.

```
L 2 //Завантаження до слова MW80 константи 2, що є номером
T MW80 //відкритого в подальшому блока даних
L P#10.0//Завантаження до AR1 константи, що відповідає адресі
LAR1 //десятого подвійного слова
……
OPN DB[MW80] //Відкриття глобального блока даних DB2
L DBD[AR1,P#0.0] //Зчитування з відкритого блока десятого подвійного слова
```

Наведені приклади є показовими з використання команд регістрової адресації та переваг цього метода. До його переваг можна віднести:

1) можливість не змінюючи дані в адресному регістрі звертатися до різних вічок пам'яті, що віддалені між собою на фіксовану величину. Це дуже зручно при копіюванні даних та в деяких алгоритмах обробки;
2) не використовується додаткова меркерна пам'ять для зберігання адреси;
3) існують спеціальні команди для швидкої зміни вмісту адресних регістрів.

До недоліків регістрової адресації можна віднести:

1) обмеженість кількості адресних регістрів, де зберігаються адреси;
2) необхідність завжди запису адреси зсуву;
3) неможливість зберігання в адресних регістрах номерів блоків даних, що відкриваються за допомогою непрямої адресації.

Контрольні запитання

1) Що таке непряма адресація? Для чого вона використовується?
2) Що таке непряма адресація з покажчиком на область при звертанні до

входів, виходів, маркерної пам’яті?

3) Як використовуються методи непрямої адресації при відкритті блоків даних?

4) Як використовуються методи непрямої адресації при роботі з таймерами?

5) Що таке адресні регістри та як вони використовуються в методах непрямої адресації при доступі до входів, виходів, маркерної пам’яті?

6) Що відрізняє непряму регістрову адресацію від непрямої адресації з покажчиком на область?

7) Як використовувати адресні регістри при доступі до даних блока даних?

Контрольні завдання

Задача. Розробити функцію, що реалізує фільтр ковзного середнього за N відліками вхідного сигналу з аналогового входу. Кількість відліків вхідного сигналу, адреса аналогового входу є вхідними параметрами. Результат фільтрації – середнє значення (тип слово, вихідний параметр).

Задача 2. Розробити функцію, що керує роботою восьми клапанів. Керування здійснюється цифровими сигналами з цифрових виходів дискретних модулів виводу. Кожен клапан відкривається на час від 40 до 140 мс. Періодичність відкриття кожного клапану 320 с. Інтервал часу між відкриттям двох сусідніх клапанів 1с. Усі перераховані часові інтервали є вхідними параметрами функції. Також вхідним параметром функції є адреса першого дискретного виходу, який керує першим клапаном. Адреси інших виходів, що керують клапанами, відрізняються від адреси першого виходу на 1,2,3,4,5,6,7 відповідно.

17 Організація циклів в межах основної програми чи функцій

Існує багато практичних задач, у яких необхідно в межах основного скану роботи програми виконати циклічну обробку, тобто організувати циклічну обробку даних. В мові STL для цього існує спеціальна команда LOOP. Формат команди

LOOP *Метка*

Команда виконує три операції:

1) вирахування з вмісту першого акумулятора 1;
2) перевірка вміста першого акумулятора на нуль;
3) коли вміст акумулятора не дорівнює нулю, то переходимо на метку.

Алгоритм роботи програми зображено на рисунку 17.1

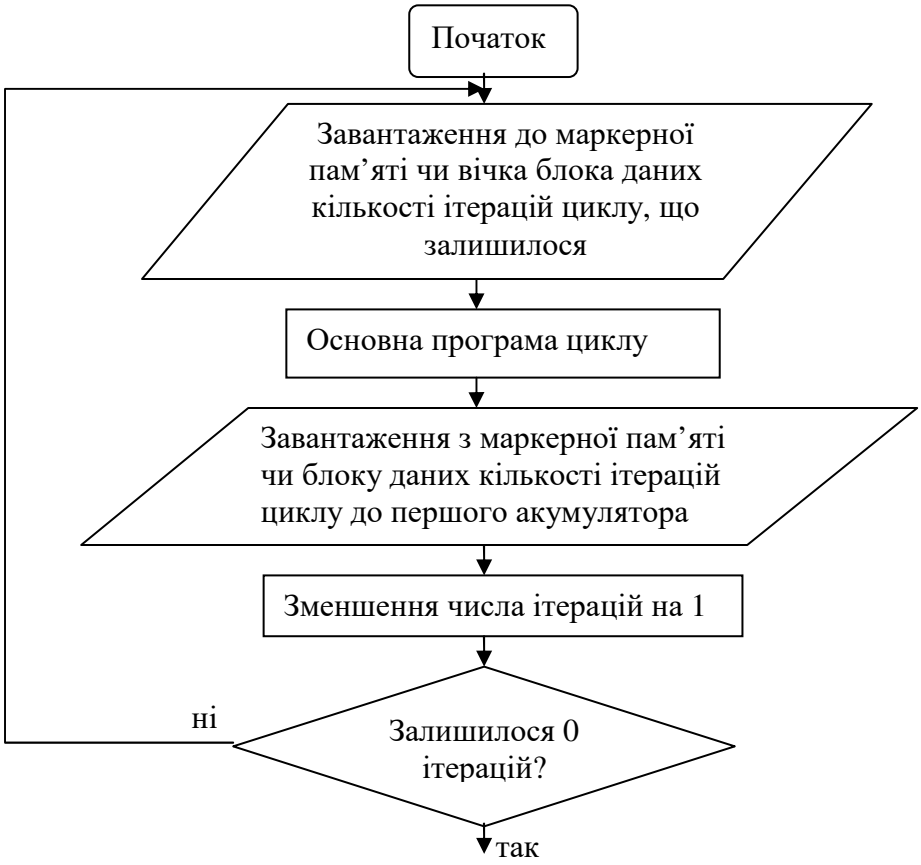


Рисунок 17.1 – Алгоритм організації циклу

Програмна реалізація циклу з 8 ітерацій наведена нижче. Поточне значення кількості ітерацій, що залишилися, знаходиться у MB100. Розміру байту цілком достатньо, бо велика кількість ітерацій може призвести до значеного збільшенні часу виконання скану, що буде не припустимо до технологічного процесу. Команда LOOP зменшує значення у першому акумуляторі на 1 та виконує перехід на метку next, коли вміст першого акумулятора не дорівнює нулю. Зберігання зменшеного значення виконується на початку ітерації циклу командою T MB 100.

Основна складність налагодження такої програми полягає в неможливості переглянути результати виконання кожної ітерації. Якщо перейти в режим перегляду програми маємо наступну інформацію про виконання

```
      Програма          RLO     STA     STANDARD
      L    8            0       1       8
next: T    MB   100     0       1       8
.....
      L    MB   100     0       1       8
      LOOP  NEXT
```

У крайньому справа стовпчику (STANDARD) відображується, скільки залишилося ітерацій циклу. Можна побачити, що відображуються результати виконання лише першої ітерації циклу. Усі наступні виконуються, але не відображуються.

У мові LAD команд, подібних до LOOP немає. Але реалізація алгоритму, що зображено на рисунку 17.1, цілком можлива. В такому випадку необхідно використовувати три команди: відрахування, порівняння та переходу на метку. Відповідна програма мовою LAD наведена на рисунку 17.2.

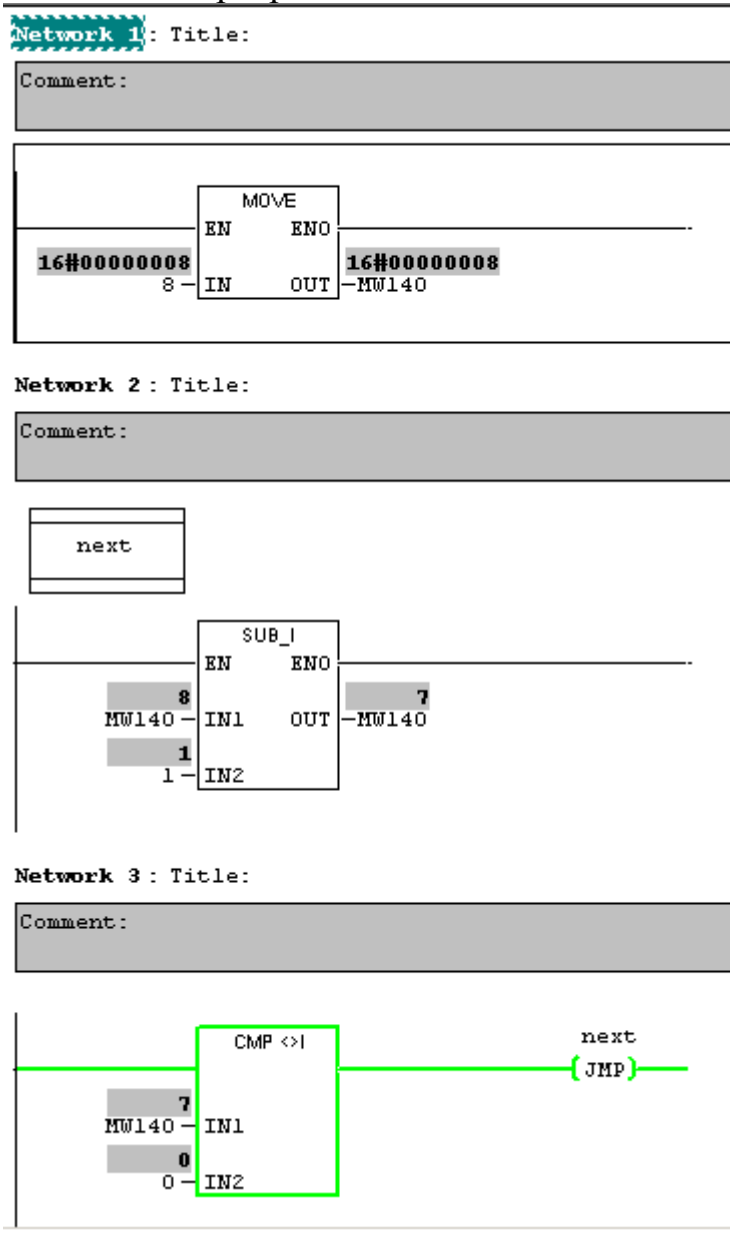


Рисунок 17.2 – Програмна реалізація циклу мовою LAD

Перед циклом до слова MW140 завантажується кількість ітерацій циклу. У мові LAD ми не можемо використовувати байти для збереження кількості ітерацій циклу, бо у подальшому, команди відрахування можуть працюють лише із словами чи подвійними словами. Цикл починається з метки *next*. У кінці циклу значення MW140 зменшується на 1 командою SUB_I, перевіряється на рівність 0 командою CMP<>I. Коли в MW140 число не дорівнює 0, то переходимо на метку *next* командою $\overset{\text{next}}{\text{JMP}}$.

Контрольні запитання

1) Для чого використовуються внутрішні цикли в межах основної про-

грами чи підпрограми Які існують обмеження при використанні внутрішніх циклів

2) Як реалізуються програмні цикли мовою STL

3) Як реалізуються програмні цикли мовою LAD

Контрольні завдання

Задача 17.1. Розробити програму функціонального блоку з використанням циклів та непрямої адресації, що знаходить суму N слів з маркерної пам'яті. Кількість слів та адреса першого з них є вхідними параметрами. Результат додавання – вихідний параметр (подвійне слово).

Задача 17.2. Розробити глобальний блок даних DB4, що містить 32 вічка типу слово. На початку виконання організаційного блока OB1 одноразово обчислити 32 відліка амплітуди одного періода синусоїдального коливання та записати ці розраховані значення у блок DB4. Амплітуда синусоїдального коливання не повинна перевищувати значення 255.

Задача 17.3. Розробити функцію, що керую роботою шістнадцяти насосів одночасно. Кожен насос має два цифрові входи для керування ним. Подання сигналу логічної «1» на першій вхід вмикає насос, подання сигналу логічної «1» на другий – вимикає. Кожен з сигналів повинен бути тривалістю 2 с. Сигнали вмикання під'єднані до виходів контролера QW0, сигнали вимикання до виходів QW2. Поточний стан насосів зчитується з їх цифрових виходів, що підключені до входів контролера IW0. Якщо насос працює на відповідному вході контролера сигнал логічної «1». Постійно повинні працювати 9 насосів. У випадку поломки насоса автоматично повинен включатися один з незадіяних на цей час насосів.

18 Обробка переривань

Обробка переривань – це припинення виконання поточної програми, що обумовлено подіями. Коли операційна система контролера одержує сигнал про те, що відбулася деяка подія, вона припиняє сканування основної програми і викликає програму, що відповідає цій конкретній події.

Після виконання цієї програми операційна система продовжує сканування основної програми із місця переривання. Такі переривання обробки програми можуть мати місце після кожної операції (оператора).

Згаданими подіями можуть бути як "переривання" так і помилки. Порядок, у якому обробляються переривання, регулюється планом пріоритетів. Кожна подія має свій пріоритет обробки.

Окремі переривання об'єднуються в класи пріоритетів. Кожна програма, пов'язана з подією-перериванням, повинна бути записана в організаційному блоці, з якого можуть бути викликані також і інші блоки. Подія з більш високим пріоритетом переривае програму в організаційному блоці з більше низьким пріоритетом. Користувач може викликати переривання процесу виконання програми подіями з високим пріоритетом, використовуючи системні функції.

У контролерах Siemens існують наступні типи переривань:

1) апаратні переривання (hardware interrupts) - переривання, викликувані модулем (за допомогою вхідного сигналу від процесу, або за допомогою сигналу, що генерується властиво в модулі);
2) таймерні переривання (watchdog interrupts) - переривання, що генеруються операційною системою із заданим періодом (інтервалом);
3) тимчасові (за часом доби) переривання (time-of-day interrupts) - переривання, що генеруються операційною системою мою в заданий час доби (однократні або періодичні);
4) переривання с затримкою обробки (time-delay interrupts) - переривання, що генеруються після закінчення заданого періоду часу (при цьому в системній функції можна визначати момент початку відліку часу заданого періоду);
5) переривання мультипроцесорного режиму (multiprocessor interrupts) - переривання, що генеруються іншими CPU у мережі, при мультипроцесорному режимі.

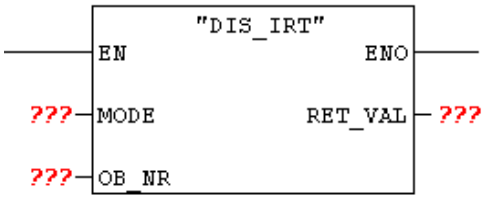
Кожному з переривань жорстко зарезервовані відповідні організаційні блоки (OB), у яких розміщають програми обробки переривань. Ця відповідність наведена у таблиці 18.1

Таблиця 18.1 – Відповідність переривань та організаційних блоків

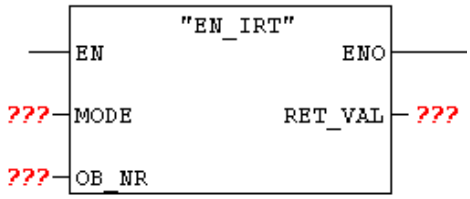
№ п/п	Назва переривання	Організаційні блоки	Пріоритети
1	За часом доби	OB10–OB17	2
2	С затримкою обробки	OB20–OB23	20-23
3	Таймерні (циклічні)	OB30–OB38	30-38
4	Апаратні	OB40–OB47	40-47
5	Мультипроцесорного режиму	OB60	60

Первинно усі переривання заборонені і не сконфігуровані. Тому, що відповідних організаційних блоків у створеному проекті немає, контролер не обробляє переривання. Необхідні у проекті переривання треба дозволити та сконфігурувати. Це можна зробити за допомогою утілити HW Config (див. рисунок 2.3) чи програмним шляхом. При використанні останнього спосіба існують системні функції, що можуть заблокувати деякі чи усі одночасно переривання (SFC 39 DIS_IRT чи SFC 41 DIS_AIRT), а також розблокувати будь-яке переривання чи усі (SFC 40 EN_IRT чи SFC 42 EN_AIRT).

Графічний вид системних функцій SFC 39 DIS_IRT та SFC 40 EN_IRT у мові LAD наведено на рисунку 18.1



а



б

Рисунок 18.1 – Графічний вид системних функцій SFC 39 DIS_IRT та SFC 40 EN_IRT

Призначення входів та виходів цих функцій описано у таблиці 18.2

Таблиця 18.2 – Призначення входів та виходів системних функцій SFC 39 DIS_IRT, SFC 40 EN_IRT

Вхід/вихід	Призначення
MODE (тип BYTE)	Визначає переривання та помилки, що будуть заборонені чи дозволені:1) 00_{hex} усі переривання та асинхронні помилки заборонені чи дозволені (синхронні помилки дозволені). Уся інформація пишеться у діагностичний буфер; 2) 01_{hex} лише один з типів переривань (при забороні чи дозволі переривань за часом доби в параметр OB_NR пишемо 10, переривань с затримкою обробки – 20; циклічних – 30; апаратних – 40; мультипроцесорного режиму – 60). Уся інформація пишеться у діагностичний буфер; 3) 02_{hex} конкретне переривання, що визначається параметром OB_NR. Уся інформація пишеться у діагностичний буфер; 4) 80_{hex} усі переривання та асинхронні помилки. Інформація не пишеться у діагностичний буфер; 5) 81_{hex} лише один з типів переривань, що визначається параметром OB_NR, інформація не пишеться у діагностичний буфер; 6) 82_{hex} конкретне переривання, що визначається параметром OB_NR, інформація не пишеться у діагностичний буфер.
OB_NR (тип INT)	Номер організаційного блока, що відповідає перериванню
RET_VAL (тип WORD)	Код помилки, що повертається за результатами виконання функції. Коли RET_VAL=0000_{hex}, помилок немає. RET_VAL=8090_{hex}, параметр OB_NR має недопустиме значення. RET_VAL=8091_{hex}, параметр MODE має недопустиме значення. RET_VAL=$8xyy_{hex}$, код помилки при виконанні функції (конкретне значення та його опис можна знайти в середовищі програмування Step7 чи технічній інформації контролеру). Значення цього параметра передають у слово маркерної пам'яті чи слово блоку даних.

Приклади виклику функцій SFC 39 DIS_IRT та SFC 40 EN_IRT у мові LAD наведено на рисунку 18.2

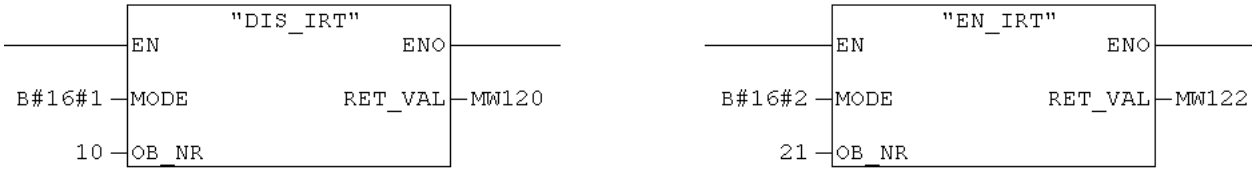


Рисунок 18.2 – Приклади виклику функцій SFC 39 DIS_IRT та SFC 40 EN_IRT

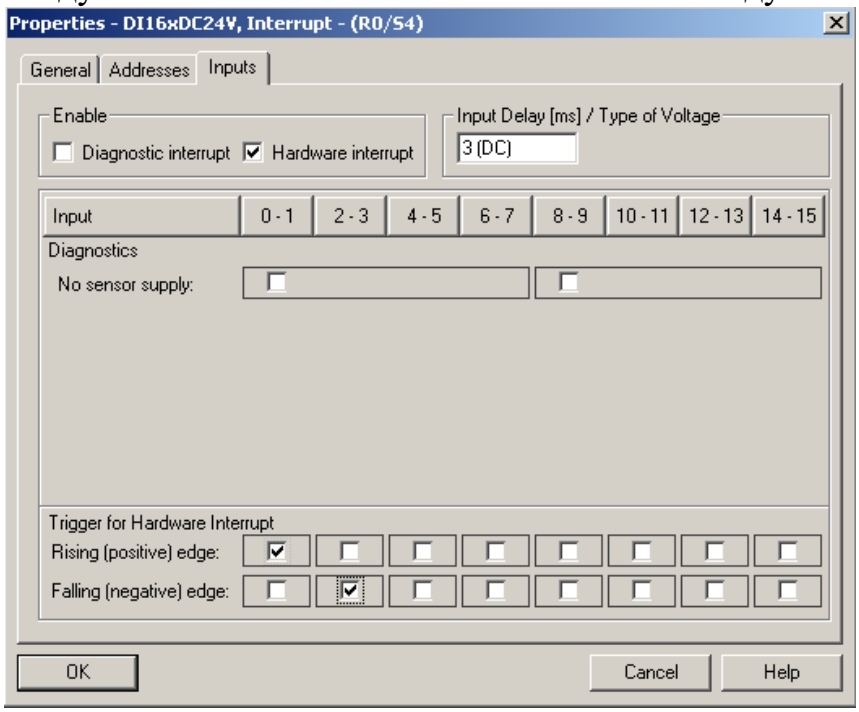
Функцією DIS_IRT забороняються усі переривання за часом доби. Результат виконання функції пишемо у слово маркерної пам'яті MW120. Функці-

єю EN_IRT дозволяємо переривання с затримкою обробки, що буде оброблятися програмою в організаційному блоці OB21. Результат виконання функції пишемо у слово маркерної пам’яті MW122.

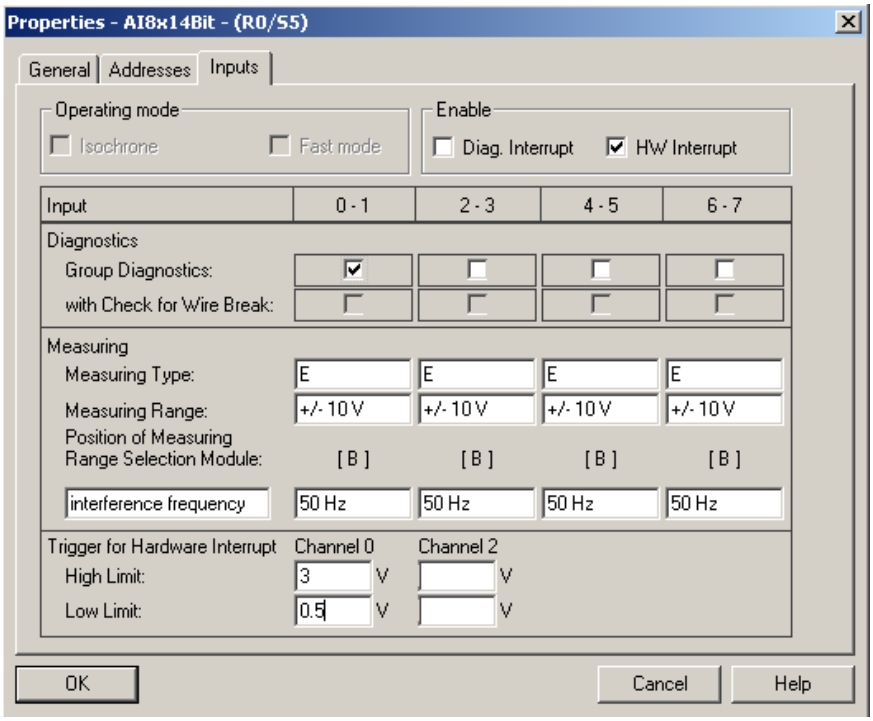
Функції SFC 41 DIS_AIRT та SFC 42 EN_AIRT мають лише один вихідний параметр RET_VAL, бо забороняють чи дозволяють усі переривання одночасно.

18.1 Апаратні переривання

Апаратні переривання можуть виникати при зміні логічного сигналу на логічному вході модуля вводу чи при виході показань аналогового сигналу на аналоговому вході за встановлені для нього межі. Спочатку усі апаратні переривання заборонені. Для їх конфігурування можна скористатися програмою HW Config, що завантажується з головного вікна відкритого проекту (див. рисунок 2.3). Після відкриття цієї програми подвійним клацанням по відповідному модулю вводу викликаємо його вікно властивостей. Наприклад, на рисунку 18.3 наведені вікна властивостей апаратних переривань дискретного модулю вводу SM 321-7BH00-0AB0 та аналогового модулю вводу SM 331-7HF00-0AB0.



а



б

Рисунок 18.3 – Вікна властивостей апаратних переривань дискретного модулю вводу SM 321-7BH00-0AB0 та аналогового модулю вводу SM 331-7HF00-0AB0

Щоб сконфігурувати апаратні переривання дискретного модуля вводу необхідно поставити дозволяючу галочку біля пункту налаштування Hardware Interrupt та галочки, відповідні входам і зміні вхідних сигналів (рисунок 18.3 а). Тепер автоматично із зміною логічного сигналу на сконфігурованих дискретних входах буде викликатися програма, що записана у одному з організаційних блоків OB40–OB47. Так на рисунку 18.3а дозволені апаратні переривання для перших чотирьох входів з 16. Для 0-го та 1-го входів дозволені переривання за наростаючим фронтом вхідного сигналу (Rising Edge), а для входів 2-го та 3-го за спадаючим фронтом вхідного сигналу (Falling Edge). Зміни сигналів на інших входах оброблятися не будуть.

Щоб сконфігурувати апаратні переривання аналогвого модуля вводу необхідно поставити дозволяючу галочки біля пункту налаштування Hardware Interrupt, біля відповідних груп входів і вказати робочий діапазон вхідних сигналів (рисунок 18.3 б). Так на рисунку 18.3б дозволені апаратні переривання для перших двох входів з 8. Для 0-го та 1-го входів дозволені переривання коли вимірювана напруга буде менше ніж 0.5В чи більше ніж 3В. Зміни сигналів на інших входах оброблятися не будуть. Тепер автоматично із виходом результатів вимірювання за межі 0.5…3В буде викликатися програма, що записана у одному з організаційних блоків OB40–OB47.

Щоб визначити, у яких організаційних блоках можна писати програми обробки апаратних переривань, треба відкрити подвійник кликанням по контролеру вікно його властивостей та вкладку Interrupts цього вікна.

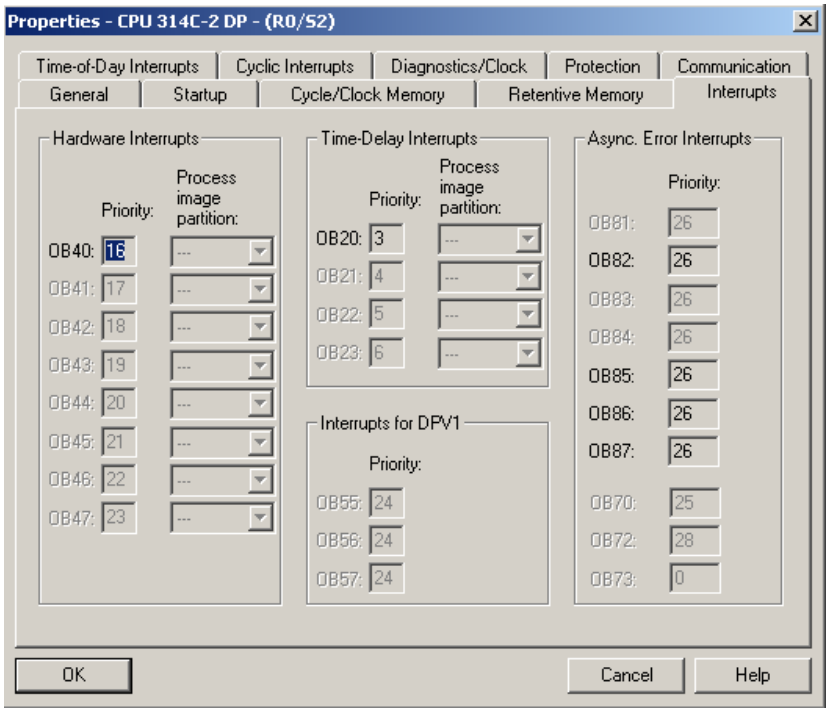


Рисунок 18.4 – Вкладка Interrupts вікна властивостей контролера

З рисунку 18.4 можна побачити, що у контролера 314C-2DP програмна обробка апаратних переривань здійснюється лише у організаційному блоці OB40. Інші блоки недоступні. Вони стають доступні в більш потужних контролерах. Тому при будь-якому перериванні буде викликатися одна і та ж програма обробки з OB40. Розпізнавання, зміна якого сигналу відбулася, буде здійснюватися у самій програмі обробки переривання. Там же буде і обробка зміни сигналу.

Потрібно враховувати, що при виконанні програми обробки переривань Ви повинні мати справу з поточними станами сигналів від I/O модулів, а не зі станами сигналів входів у пам'яті, які обновляються на початку виконання основної програми. Функціональні схеми модулів вводу передбачають спеціальні логічні ланцюги для реагування на зміни сигналів. Тому імітація апаратних переривань за допомогою програми симуляції S7-PLCSIM неможлива, бо у цій програмі імітуються зміни у області пам'яті контролера, що відповідна реальним входам.

Якщо ініційовано апаратне переривання, для якого в користувальницькій програмі немає організаційного блоку, операційна система викликає OB 85 (організаційний блок обробки помилок, виникаючих при виконанні програми). Якщо OB 85 не запрограмовано, CPU переходить у режим STOP. Тому обов'язково слід створювати блок OB 85, хоча він може не містити ніякої програми обробки помилок.

18.2 Таймерні переривання

Таймерні переривання – це виклик відповідної програми обробки пере-

ривання через рівні фіксовані інтервали часу. У відповідному організаційному блоці (OB30-OB38) може знаходитися, як програма обробки переривання так і головна програма. Якщо там знаходиться головна програма, то організаційний блок OB1 не створюють. Розміщення головної програми в OB30-OB38 характерно для систем керування з фіксованим інтервалом обробки даних. Але Ми не радимо Вам розміщувати головну програму у таймерному перериванні, бо іноді сукупність інших переривань та мережного передавання даних може значно збільшити час виконання програми в OB30-OB38 і цей час перевищить період виклику переривання. У таких випадках виникає помилка у роботі контролера та він переходить у режим STOP.

При параметризації CPU Ви можете задати параметри таймерного переривання. Таймерне переривання має три параметри: інтервал, фазовий зсув і пріоритет. Ви можете визначити всі три параметра. Значення, які Ви можете задати для перших двох з зазначених параметрів, лежать у діапазоні часу від 1 мілісекунди до 1 мінути із кроком, рівним 1 мілісекунді. Значення для пріоритету вибираються з діапазону значень від 2 до 24, а також можливо нульове значення (0), залежно від CPU. Якщо пріоритет прирівнюється нулю, то таймерне переривання не буде активно.

Спочатку усі таймерні переривання заборонені. Для їх конфігурування можна скористатися програмою HW Config, що завантажується з головного вікна відкритого проекту (див. рисунок 2.3). Після відкриття цієї програми подвійним клацанням по контролеру викликаємо його вікно властивостей Cyclic Interrupts (рисунок 18.5).У даному випадку відкрито вікно властивостей контролера CPU317F-2

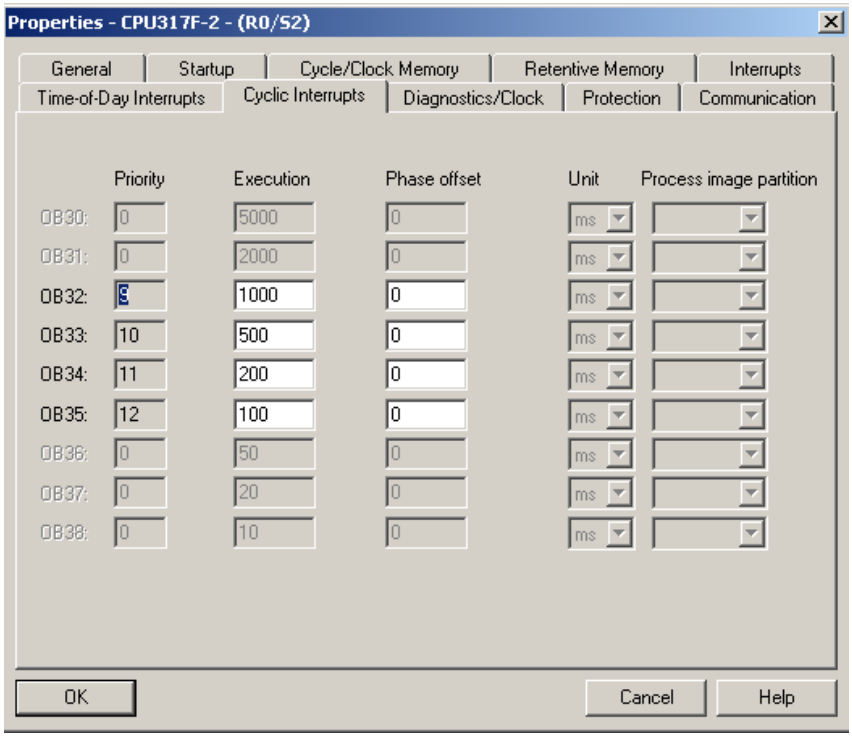


Рисунок 18.5 – Вікно властивостей контролера Cyclic Interrupts

З рисунку 18.5 ми бачимо, що для даного контролера дозволено лише чо-

тири таймерні переривання, обробка яких виконується у ОВ32-ОВ35. Для менш потужних контролерів буде дозволено менше переривань (для CPU314C-2DP дозволено лише одне переривання з обробкою у модулі ОВ35). Для більш потужних контролерів можуть бути дозволені усі переривання. За умовчуванням для обраного контролера завдано час виклику (для ОВ35 – 100мс і т.і.) та фазовий зсув, значення якого зараз для усіх переривань дорівнює нулю. Значення часу виклику переривання можна змінити на будь-яке з дозволеного інтервалу.

Параметр "Фазовий зсув" (Phase Offset) може бути використаний для забезпечення часового зсуву запуску підпрограм, що запускаються таймерними перериваннями, поза залежністю від того факту, що для цих програм в якості параметрів задана безліч однакових тимчасових періодів.

Використання фазового зсуву дозволяє уточнити процес періодичного запуску переривань. Наприклад, з рисунку 18.5, випливає, що через перші 100 мс роботи контролера буде викликана програма з ОВ35, через 200 мс одночасно треба викликати програми з ОВ35 та ОВ34, через 500 мс одночасно треба викликати програми з ОВ35, ОВ34, ОВ33 і т.д. Але одночасне виконання декількох програм неможливо.

Параметри "Час старту" і "Фазовий зсув" визначають момент переходу від режиму запуску у режим виконання організаційного блоку. Момент виклику ОВ таймерного переривання визначається як сума часового інтервалу і фазового зсуву. На рисунку 18.6 показаний приклад використання часу виклику і фазового зсуву.

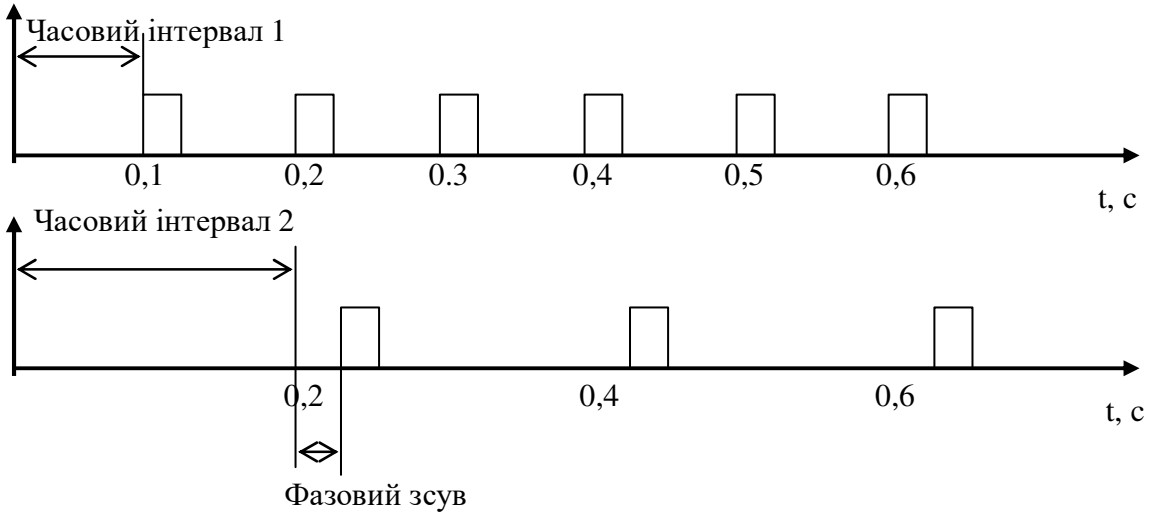


Рисунок 18.6 – Моменти часу викликів переривань у ОВ35 та ОВ34.

Як можна побачити з рисунка 18.6, для часового інтервалу 1 параметр "Фазовий зсув" не встановлюється (або дорівнює 0); часовий інтервал 2 має у два рази більшу величину, чим часовий інтервал 1. Тому що часовий інтервал 2 має ненульовий фазовий зсув, ОВ34 для часового інтервалу 2 і ОВ35 для часового інтервалу 1 не можуть бути викликані одночасно.

Отже, ОВ34 з більше низьким пріоритетом не буде примушений очікувати поки виконається ОВ35 з більше високим пріоритетом, і буде оброблятися у свій строк, точно витримуючи заданий часовий інтервал.

Конфігурування таймерних переривань, як і усіх інших, потребує компіляції результатів , завантаження оновленої конфігурації до реального контролера чи симулятора, створення у головному вікні проекту відповідних блоків OB32-OB35 та завантаження їх до контролера.

Якщо знову генерується то ж саме таймерне переривання, для якого в користувальницькій програмі вже виконується відповідний організаційний блок, то операційна система викликає. Якщо OB 80 не запрограмований, CPU переходить у режим STOP. Тому обов'язково слід створювати блок OB 80, хоча він може не містити ніякої програми обробки помилок.

18.3 Переривання за часом доби

Переривання за часом доби дозволяють викликати відповідні їм програми однократно чи періодично у відповідні часи та дні року. Для розміщення програм обробки цих переривань передбачені організаційні блоки з OB 10 по OB 17. Проте, те, які з даних восьми організаційних блоків доступні залежить від типу CPU.

Обробка переривань за часом доби може бути запрограмована при конфігуруванні встаткування (у програмі HW Config) або ж може бути виконана за допомогою використання системних функцій під час робочого режиму RUN. На рисунку 18.7 наведено графічне вікно властивостей переривань за часом доби для контролера CPU314C-2DP (Time-of-Day Interrupts).

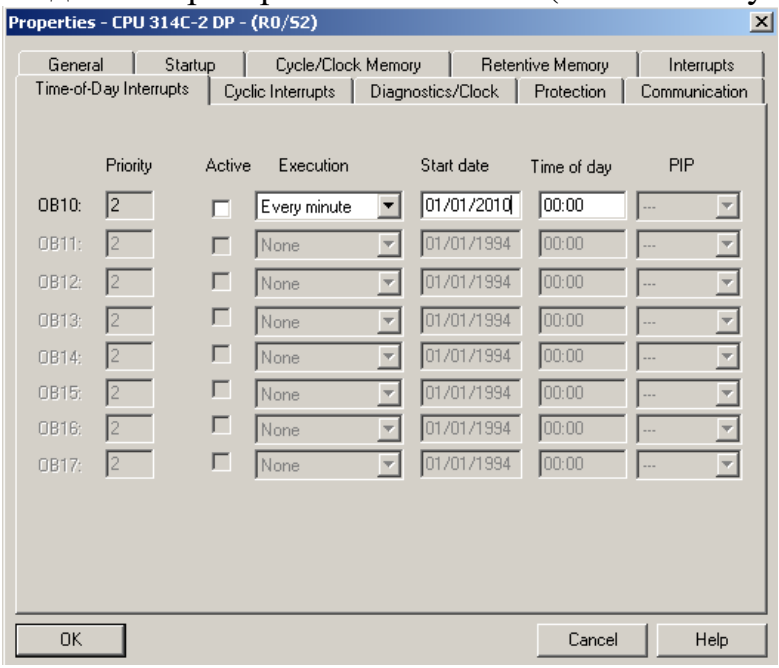


Рисунок 18.7 – Графічне вікно властивостей переривань за часом доби для контролера CPU314C-2DP.

З рисунку 18.7 можна бачити, що для переривань за часом доби необхідно настроювати час (Time of day) та день року (Start date) з якого вини почнуть виконуватися, періодичність виклику (Execution) та активувати переривання

встановленням галочки у полі Active.

Передумовою правильного виконання обробки переривань за часом доби є коректна установка годин реального часу CPU (real-time clock).

Ви можете ініціювати переривання за часом доби (time-of-day interrupts) окремо за допомогою конфігурування встаткування або за допомогою використання системних функцій. Необхідно відзначити, що при активації за допомогою конфігурування встаткування переривання запускається автоматично після параметризації CPU.

Ви можете запустити переривання за часом доби двома способами (параметр Execution):

- однократний запуск – відповідний організаційний блок викликається один раз у заданий час;
- періодичний запуск – залежно від значень параметрів OB викликається щохвилини, щогодини, щодня, щотижня, щомісяця, в кінці місяця або щорічно.

Для контролерів серії S7-300 для блоків обробки встановлений за замовчуванням фіксований пріоритет 2. Для серії S7-400 для всіх можливих організаційних блоків (у відповідності зі специфікаціями CPU) Ви можете встановлювати значення пріоритету з діапазону від 2 до 24 або рівне 0.При встановленні для блоку значення пріоритету 0 блок перестане виконуватися. Ніколи не призначайте той самий пріоритет двічі, тому що при цьому можуть бути загублені переривання в випадку, якщо більше, ніж 12 переривань із однаковим пріоритетом будуть ініційовані одночасно.

Для закінчення конфігурування устаткування необхідно скомпілювати конфігурацію  , завантажити оновлену конфігурацію до реального контролера  чи симулятора, створити у головному вікні проекту відповідні блоки OB10-OB17 та завантаження їх до контролера.

Для програмного керування перериваннями за часом доби передбачено наступні системні функції:

- SFC 28 SET_TINT функція для налаштування переривання за часом доби;
- SFC 29 CAN_TINT функція для скасування переривання за часом доби;
- SFC 30 ACT_TINT функція для активації переривання за часом доби;
- SFC 31 QRY_TINT функція для запиту про стан переривання за часом доби.

Усі функції знаходяться у бібліотеці Libraries/Standard Library/System Function Blocks.

Параметри для вищевказаних системних функцій представлені в таблиці 18.3.

Таблиця 18.3 – Опис інтерфейсів функцій SFC 28, SFC 29, SFC 30, SFC 31

SFC	Параметр	Вхід/ вихід	Тип	Опис параметра
28	OB_NR	INPUT	INT	Номер OB для виклику в заданий час
	SDT	INPUT	DT	Дата й час запуску у форматі DATE_AND_TIME
	PERIOD	INPUT	WORD	Період запуску переривання (базовий період для часу запуску [start time]): W#16#0000 = однократний запуск W#16#0201 = запуск щохвилини W#16#0401 = запуск щогодини W#16#1001 = запуск щодня W#16#1201 = запуск щотижня W#16#1401 = запуск щомісяця W#16#2001 = запуск в кінці місяця W#16#1801 = запуск щороку
	RET_VAL	OUTPUT	INT	Інформація про помилки
29	OB_NR	INPUT	INT	Номер OB, стартовий час якого необхідно скасувати.
	RET_VAL	OUTPUT	INT	Інформація про помилки
30	OB_NR	INPUT	INT	Номер OB, якому необхідно активувати.
	RET_VAL	OUTPUT	INT	Інформація про помилки
31	OB_NR	INPUT	INT	Номер OB, про стан якого необхідно зробити запит.
	RET_VAL	OUTPUT	INT	Інформація про помилки
	STATUS	OUTPUT	WORD	Стан переривання за часом доби

Системна функція **SFC 28 SET_TINT** визначає стартовий час (час запуску) для переривання за часом доби. Вона дозволяє тільки задати параметр "стартовий час". Для того щоб запустити переривання за часом доби, Ви повинні будете ще й активувати це переривання, викликавши системну функцію SFC 30 ACT_TINT. "Стартовий час" визначається в параметрі SDT у форматі DATE_AND_TIME. Операційна система ігнорує секунди і мілісекунди та установлює для них нульові значення. При установці стартового часу для переривання за часом доби старе значення, якщо воно було задано, замінюється новим значенням.

При цьому активність переривання за часом доби анулюється, що означає, що Ви після установки нового часу повинні будете активізувати переривання за допомогою функції SFC 30 ACT_TINT.

Особливістю використання системної функції SFC 28 є те, що серед простих типів даних немає типу. Тобто ми не можемо задати константу цього типу, хоча можемо створити змінну цього типа. Цей тип є об'єднанням двох простих типів даних DATE та TIME. Для завдання у програмі значення змінної типу DATE_AND_TIME використовується системна функція FC3, що входить у пакет Step7. Вона знаходиться у бібліотеці Libraries/stdlibs/iec. Приклад програми

налаштування переривання за часом доби з використанням системних функцій FC3 та SFC 28 мовою LAD наведено на рисунку 18.8

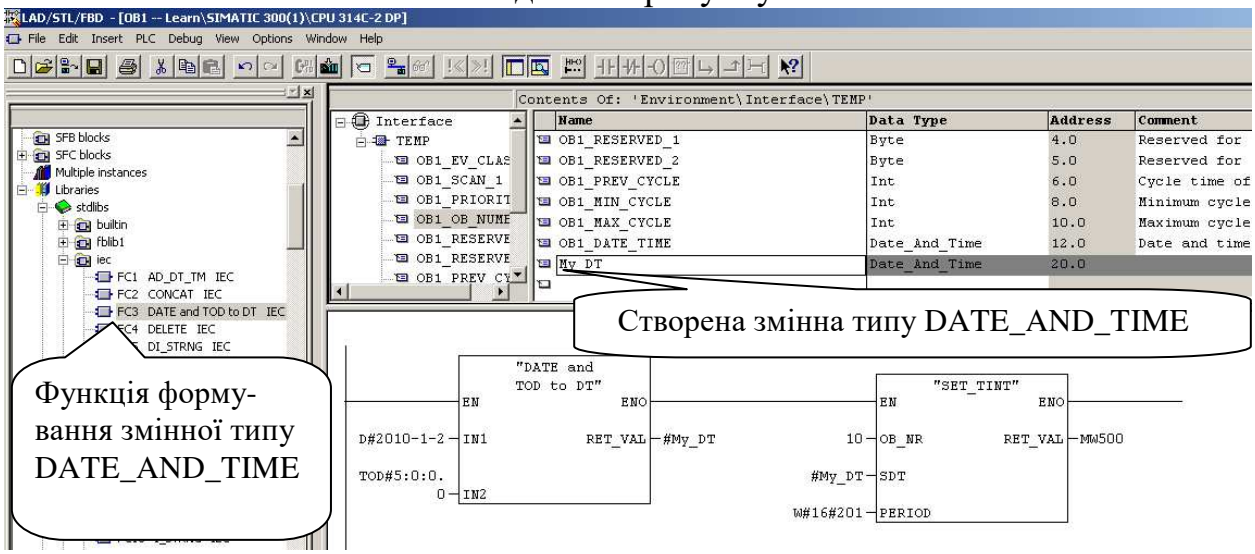


Рисунок 18.8 – Приклад налаштування переривання за часом мовою LAD

У наведеному на рисунку 18.8 прикладі в організаційному блоці OB1 у рядку програми послідовно викликані функції FC3 та SFC 28. Їх перетягнуто до рядку з відповідних бібліотек. Функція FC3 має два вхідних параметра IN1, IN2 та вхід дозволу виконання функції EN. Параметр IN1 це дата, IN2 – час. В наведеному прикладі переривання почне виконуватися з п’ятої години 2 січня 2010 року, бо завдан час TOD#5:0:0.0 (5 годин: 0 хвилин: 0 секунд. 0 мілісекунд) та дата D#2010-1-2 (2 січня 2010 року). Вихідний параметр функції RET_VAL потребує передавання результату функції у змінну відповідного типу DATE_AND_TIME. Тому серед тимчасових змінних TEMP організаційного блоку OB1 створена змінна My_DT типу DATE_AND_TIME. Якщо б функція FC3 викликалася у іншому організаційному блоці чи іншій підпрограмі, то необхідно б було створити тимчасову змінну типу DATE_AND_TIME в розділі тимчасових змінних тієї підпрограми.

Функція SFC 28 настроює переривання за часом доби, що буде оброблятися в організаційному блоці OB10 починаючи з часу та дати, що задає змінна My_DT кожну хвилину, бо на вході PERIOD задана константа W#16#201. Код помилки у разі її виникнення чи нуль у разі коректного виконання функції записується у слово маркерної пам’яті MW500.

Мовою STL ця ж програма має наступний вид

```
CALL  "DATE and TOD to DT" //Виклик функції FC3
IN1    :=D#2010-1-2 //Завантаження дати
IN2    :=TOD#5:0:0.0 //Завантаження часу
RET_VAL:=#My_DT //Збереження дати та часу у змінній My_DT
CALL  "SET_TINT" //Виклик функції FC28
OB_NR  :=10 //Завантаження номера організаційного блока, у
                    //якому буде оброблятися переривання за часом доби
SDT    :=#My_DT //Завантаження часу та дати початку обробки пере-
```

ривання

PERIOD :=W#16#201 //*Завантаження періоду обробки переривання*

RET_VAL:=MW500 //*Збереження інформації про виконання функції*

Особливістю набору коду програми мовою STL є те, що функції FC3 та SFC 28 при їх першому використанні необхідно перетягувати у текст програми з бібліотек. При цьому вини копіюються до проекту.

Після налаштування переривання необхідно створити відповідний організаційний блок (у даному випадку це блок OB10) та програму обробки переривання у середині блока. Далі блок OB10 слід завантажити до симулятора чи реального контролера.

Щоб переривання за часом доби почало виникати його треба активувати функцією **SFC30 ACT_TINT**. Якщо переривання активоване, то вважається, що параметр "стартовий час" ("start time") встановлений для переривання за часом доби. Якщо Ви активували переривання за часом доби в режимі однократного виклику та при цьому заданий стартовий час (start time) вже пройшов на сучасний момент, то SFC 30 видасть повідомлення про помилку. Коли Ви активували переривання в режимі періодичного виклику та при цьому заданий стартовий час вже пройшов, то операційна система викличе для обробки відповідний OB у наступному підходящому періоді в заданий момент часу. Якщо переривання за часом доби в режимі однократного виклику на сучасний момент виникло, то це переривання для подальшого застосування більш недоступно. Якщо потрібно знову його використовувати (з іншим значенням стартового часу) Ви повинні його перезавантажити і потім знову активувати.

Для скасування вже активованого переривання за часом доби використовується функція **SFC29 CAN_TINT** для якої треба вказати організаційний блок, який не буде оброблятися та адресу вічка оперативної пам’яті де буде збережена інформація про помилки при виконанні функції.

Отримати інформацію про стан виконання переривання за часом доби можна за допомогою системної функції **SFC 31 QRY_TINT**. Функція SFC 31 QRY_TINT повертає необхідну інформацію в параметрі STATUS.

Якщо відповідний біт параметра STATUS має сигнал із значенням "1", то це означає:

0 - переривання за часом доби заблоковано операційної системою;

1 - нове переривання за часом доби відкинуто;

2 - переривання за часом доби не активоване та не завершено;

3 - зарезервований;

4 - завантажений OB переривання за часом доби;

5 - переривання за часом доби не заблоковано.

Якщо викликається OB обробки переривання за часом доби, але даний OB не запрограмований, то операційна система викликає OB 85. Якщо OB 85 не запрограмований, CPU переходить у режим STOP. Однократного виклику, і якщо при цьому заданий стартовий час уже пройшло на сучасний момент (по годинниках реального часу системи), то операційна система викличе OB 80. Якщо OB 80 недоступний, те CPU перейде в режим STOP.

Якщо викликається ОВ обробки переривання за часом доби, для якого в користувальницькій програмі вже виконується відповідний організаційний блок (режим періодичного запуску), те операційна система викликає ОВ 80. Коли ОВ 80 і ОВ обробки переривання за часом доби будуть виконані, ОВ обробки переривання за часом доби буде перезапущений.

Тому обов'язково слід створити ОВ 85 та ОВ 80. Програми обробки помилок в них не обо'язкові.

18.4 Переривання с затримкою обробки

Переривання із затримкою обробки (time-delay interrupts) – це такі переривання, що виконують функції таймера затримки без використання стандартних функцій таймера. В STEP 7 передбачено організаційні блоки з ОВ 20 по ОВ 23 спеціально для обслуговування переривання із затримкою обробки. Проте, те, які з даних чотирьох організаційних блоків доступні, залежить від типу CPU. Пріоритети ОВ обробки переривань із затримкою обробки програмуються при конфігуруванні. Головними особливостями даного типу переривань є:

1) запуск та настроювання переривання виконується лише відповідними системними функціями;
2) переривання виконується лише один раз. Для наступного його виконання необхідно знову запустити його;
3) є можливість викликати переривання через час від 1мс.

Основною перевагою даного переривання перед програмними таймерами є можливість точного вимірювання часу від 1 мс. Це дуже важливо для деяких систем керування. Прикладом можуть бути швидкісні клапани, котрі треба відкривати на час від 1 мс. Програмні таймери, по-перше, не можуть запуститися на час менший ніж 10 мс, а, по-друге, обробка закінчення роботи програмного таймера затримується до моменту виконання інструкції опитування таймера в програмі, що при часі виконання головної програми 100-150мс має те ж саме мінімальне значення часу. В даному випадку використання переривань с затримкою обробки дає точний відлік часу з обробкою переривання у тому з скані роботи контролера, якщо він ще не закінчився з моменту запуску переривання.

Для керування перериваннями с затримкою обробки передбачені наступні системні функції:

- SFC 32 SRT_DINT функція для установки затримки запуску ОВ переривання;
- SFC 33 CAN_DINT функція для скасування переривання із затримкою обробки;
- SFC 34 QRY_TINT функція для запиту про стан переривання із затримкою обробки.

Таблиця 18.4 – Інтерфейс функцій SFC 32, SFC 33, SFC 34

SFC	Параметр	Вхід/ вихід	Тип	Опис параметра
32	OB_NR	INPUT	INT	Номер ОВ для виклику

	DTIME	INPUT	TIME	Часовий інтервал затримки (припустимі значення – в діапазоні: T#1ms…T#1m)
	SIGNINPUT	INPUT	WORD	Ідентифікатор завдання в стартовій інформації відповідного ОВ при його виклику (довільні символи)
	RET_VAL	OUTPUT	INT	Інформація про помилки
33	OB_NR	INPUT	INT	Номер ОВ, виклик якого необхідно скасувати.
	RET_VAL	OUTPUT	INT	Інформація про помилки
34	OB_NR	INPUT	INT	Номер ОВ, про стан якого необхідно зробити запит.
	RET_VAL	OUTPUT	INT	Інформація про помилки
	STATUS	OUTPUT	WORD	Стан переривання затримки обробки

Переривання із затримкою обробки (time-delay interrupt) запускається за допомогою системної функції SFC 32 SRT_DINT. При виклику функції SFC 32 SRT_DINT починається відлік часу запрограмованої затримки.

Ідентифікатор завдання визначається в параметрі SIGN для системної функції SFC 32. Значення затримки (періоду часу) установлюється із кроком збільшення, рівним 1 мс. Точність для параметра "затримка обробки" також становить 1 мс. Необхідно відзначити, що процес виконання блоку ОВ, пов'язаного з перериванням із затримкою обробки, може бути сам затриманий, якщо організаційні блоки з більш високими пріоритетами почнуть оброблятися в той же період часу, що й викликаний ОВ.

Стан переривання із затримкою обробки перевіряється системною функцією SFC 34 QRY_DINT. Переривання при цьому вибирається за допомогою параметра OB_NR (номер блоку). Функція SFC 34 QRY_DINT повертає необхідну інформацію в параметрі STATUS. Якщо відповідний біт містить сигнал зі значенням "1", то це означає наступне:

0 - переривання із затримкою обробки заблоковано операційної системою;

1 - нове переривання із затримкою обробки відкинуто;

2 - переривання із затримкою обробки не активоване та не завершено;

3 - зарезервований;

4 - завантажений ОВ переривання із затримкою обробки;

5 - переривання із затримкою обробки не заблоковано;

6 - і наступні зарезервовані.

Якщо викликається ОВ обробки переривання із затримкою обробки, але даний ОВ не запрограмований, то операційна система викликає ОВ 85. Якщо ОВ 85 не запрограмовано, CPU переходить у режим STOP. Якщо минув інтервал затримки і викликається ОВ обробки даного переривання, для якого в користувальницькій програмі вже виконується відповідний організаційний блок, то операційна система викликає ОВ 80 або переходить у режим STOP, якщо ОВ 80 недоступний у користувальницькій програмі.

18.5 Переривання мультипроцесорного режиму

Переривання мультипроцесорного режиму дозволяє забезпечити синхронну реакцію на подію у всіх CPU, що працюють в мультипроцесорному режимі. Переривання мультипроцесорного режиму запускається за допомогою SFC 35 MP_ALM. Для обслуговування переривання мультипроцесорного режиму передбачений організаційний блок OB 60, для якого встановлений фіксований пріоритет, рівний 25. Виклик системної функції SFC 35 MP_ALM запускає виконання організаційного блоку обробки переривання мультипроцесорного режиму. Якщо CPU працює в однопроцесорному режимі, то організаційний блок OB 60 запускається негайно. Якщо режим роботи мультипроцесорний, то блок OB 60 запускається одночасно на всіх CPU мультипроцесорної системи, що означає, що CPU, в якому викликана функція SFC 35 відкладе запуск блоку OB 60 до моменту, коли всі інші CPU повідомлять про свою готовність.

Функція SFC 35 MP_ALM має один вхідний параметр типу BYTE, який є однаковим для усіх контролерів в мультипроцесорному режиму та передається до них. Значення параметра від 1 до 15. Також у функції є один вихідний параметр RET_VAL типу INT який містить код помилки при її виникненні.

Контрольні питання.

1) Що таке апаратні переривання? Як налаштовувати апаратні переривання на цифрових та аналогових входах модулів вводу?
2) У яких організаційних блоках обробляються апаратні переривання? Скільки таких блоків може існувати та від чого залежить їх чисельність?
3) Що таке таймерні переривання? У чому їх призначення Як налаштувати таймерні переривання?
4) У яких організаційних блоках обробляються таймерні переривання? Скільки таких блоків може існувати від чого залежить їх чисельність?
5) Що таке переривання за часом доби? Для чого вони використовуються? Як налаштовуються переривання за часом доби у програмі Hadware config пакету Step7?
6) Як налаштовуються переривання за часом доби за допомогою системних функцій? Об’ясніть призначення цих функцій та їх вхідних і вихідних параметрів.
7) У яких організаційних блоках обробляються переривання за часом доби? Скільки таких блоків може існувати від чого залежить їх чисельність?
8) Для чого використовуються переривання с затримкою обробки? Які часові інтервали можна запрограмувати для цього типу переривань?
9) Як налаштовуються переривання с затримкою обробки за допомогою системних функцій? Об’ясніть призначення цих функцій

та їх вхідних і вихідних параметрів?

10) У яких організаційних блоках обробляються переривання с затримкою обробки? Скільки таких блоків може існувати від чого залежить їх чисельність?

Контрольні завдання

Задача 18.1. Необхідно сформувати періодичну послідовність імпульсів на виході Q124.4 після натискання кнопки START, що під'єднана до входу I2.0. Формування імпульсів закінчується при натисканні кнопки STOP під'єднаної до входу I2.1. Період імпульсів 0,4 с, тривалість імпульсу 5 мс. Формування імпульсів реалізувати за допомогою переривання с затримкою обробки. Необхідне обладнання для реалізації проекту обрати самостійно.

Задача 18.2. У обробнику таймерного переривання розробити програму, що керує роботою 8 клапанів за допомогою цифрових сигналів, що подаються з виходів QB124. На кожен клапан повинні подаватися імпульси тривалістю 60 мс. Інтервал часу між подачею імпульсів на два сусідні клапани 2 с. Періодичність процесу повторювання відкриття клапанів – 600 с. Необхідне обладнання для реалізації проекту обрати самостійно.

ПЕРЕЛІК ПОСИЛАНЬ

1. Hans Berger Automating With STEP 7 In LAD And FBD: Programmable Controllers SIMATIC S7-300/400:Publicis Mcd Werbeagentur Gmbh. 2008, – 440 p.
2. Hans Berger Automating With STEP7 In STL And SCL: Programmable Controllers SIMATIC S7-300/400: Wiley-vch Verlag Gmbh. 2009, – 544 p.
3. Офіційний сайт представництва компанії Siemens в Росії [Електронний ресурс]. – Режим доступу : http://old.automation-drives.ru/as/products

ГЛОСАРІЙ

Акумулятор (*аккумулятор, accumulator register*) — регістр, у котрий завантажуються дані для виконання арифметичних та логічних операцій, таймерів, лічильників, а також знаходяться результати цих операцій.
Апаратні переривання (*аппаратные прерывания, hardware interrupt*) – припинення виконання основної програми чи підпрограми при зміні логічного сигналу на входах дискретного модуля вводу чи при зміні аналогового сигналу на вході аналогового модуля вводу.
АСКТП (*АСУТП, process control system*) автоматизована система керування технологічним процесом— комплекс програмних та технічних засобів, призначених для автоматизації керування технологічним обладнанням на підприємстві. Зазвичай АСКТП має єдину систему операторського керування технологічним процесом у вигляді одного чи декількох пультів керування, засобів обробки та архівування інформації.
Блок даних (*блок данных, data block*) – набір однотипних чи різних за типом змінних, що розміщуються у оперативній пам’яті контролера та використовуються у користувацькій програмі, але зберігаються при вимиканні живлення.
Користувацька функція (*пользовательская функция, user function*) – це розроблена програмістом АСКТП підпрограма, що має назву та викликається з головних програм чи інших підпрограм стільки разів скільки необхідно.
Лічильник (*счётчик числа импульсов, counter*) — пристрій, на арифметичних виходах котрого формується значення кількості підрахованих імпульсів у десятковому чи двійково-десятковому форматах.
Меркерна пам’ять (*меркерная пам'ять, merker memory*) – найбільш швидка оперативна пам’ять програмованого контролеру, інформація у якій зникає при вимиканні живлення.
Модуль вводу інформації (*модуль ввода информации, input module of the information*) – спеціалізований пристрій для опитування дискретних чи аналогових входів, збору від них даних та передачі цих даних для обробки на програмований контролер.
Модуль виводу інформації (*модуль вывода информации, output module of the information*) – спеціалізований пристрій для виводу на цифрові чи аналогові виходи даних, переданих від програмованого контролера.
Непряма адресація (*косвенная адресация, indirect addressing*) – можливість однією командою звертатися до даних за різними адресами оперативної пам’яті контролера, коли ці адреси знаходяться у пам’яті контролера чи адресних регістрах.
Область входів (*область входов, area of inputs*) – область оперативної пам’яті контролера, у яку його операційна система записує стани сигналів, що отримані з реальних фізичних входів.

Область виходів (*область выходов, area of outputs*) – область оперативної пам'яті контролера, у якій користувацька програма змінює дані щоб операційна система контролера потім передала ці дані на реальні фізичні виходи.
Організаційний блок (*организационный блок, OB*) – одна з головних програм, що викликається та виконується автоматично операційною системою програмованого контролера.
Переривання за часом доби (*прерывание по времени суток, time of day interrupt*) – виклик розробленої програми одноразово чи періодично у фіксований час доби.
Переривання затримки обробки (*прерывание задержки обработки, time delay interrupt*) – одноразовий виклик розробленої програми по закінченню роботи апаратного таймеру.
Програмований контролер (*программируемый контроллер, programmable logic controller, PLC*)— спеціалізований процесорний пристрій, що використовується для автоматизації технологічних процесів. Працює без особливого обслуговування при несприятливих умовах навколишнього середовища.
Регістр статусу (*регистр статуса, status register*) – регістр, кожен біт якого автоматично змінюється за результатами виконання арифметичних чи логічних команд. Ці біти використовуються для виконання порівнянь, переходів і т.і.
Симулятор (*симулятор, simulater*) – спеціалізована програма, що дозволяє при відсутності реального контролера імітувати роботу розробленого програмного забезпечення у складі з моделлю програмованого контролера, що вибраний для реалізації керування у автоматизованій системі.
Системна функція (*системная функция, system function*) – це розроблена компанією Siemens підпрограма, що має назву та викликається з головних програм чи інших підпрограм стільки разів скільки необхідно.
Скан роботи контролера (*скан работы контроллера, controler scan*) – періодичне опитування входів, що під'єднані до контролеру, виконання користувацької програми, зміна стану виходів, виконання комунікаційних функцій.
Таймер (*таймер, timer*) – об'єкт, що викликає подію по закінченню заданого інтервалу часу. Подією є: зміна сигналу на дискретному виході чи у пам'яті, виклик функції і т.і.
Функціональний блок (*функциональный блок, function block*) – це підпрограма що має назву та викликається з головних програм чи інших підпрограм стільки разів скільки необхідно. На відміну від функції функціональний блок має залежний від нього блок даних, що дозволяє зберігати значення вхідних та вихідних параметрів від виклику до виклику.
Функція (*функция, function*) —це підпрограма, що має назву та викликається з головних програм чи інших підпрограм стільки разів скільки необхідно.
Циклічні переривання (*циклические прерывания, cyclic interrupt*) – припинення виконання головної програми та виклик іншої розробленої програми через рівні інтервали часу операційною системою контролеру.