Принцип на работа на програмируемите логически контролери. Програмируемите логически контролери работят под управлението на операционна система за реално време (ОСРВ), която осигурява циклично изпълнение на логическата програма.

Операционната система управлява последователното изпълнение на няколко основни действия. Тази последователност, както и самото й изпълнение, се нарича оперативен цикъл (operating cycle) на промишления контролер. Основните действия на промишления контролер са 5. Те се изпълняват последователно и се наричат още „фази на оперативния цикъл”. Поредните номера и означенията на фазите са както следва:

1 – IN – сканиране на входовете (input scan)
2 – PRG – сканиране на програмата (program scan)
3 – OUT – сканиране на изходите (output scan)
4 – COM – обслужване на комуникацията (service communications)
5 – H&O – системни функции (housekeeping and overhead)

Принцип на работа на програмируемите логически контролери.

Принцип на работа на програмируемите логически контролери.

Фаза 1 се нарича сканиране на входове (Input Scan). Физическите места, където постъпват входните сигнали в промишления контролер се наричат входни точки (Input Points). За всяка входна точка е определен по един бит в оперативната памет – входен бит (Input Bit). По време на тази фаза се прочитат входните точки и информацията от тях се записва във входните битове. Създава се „образ" на входните въздействия към дадения момент в областта на входните битове на оперативната памет (Process Image Inputs).

Фаза 2 е сканиране на програмата (Program Scan). По време на нея се изпълняват всички инструкции от логическата програма по отработване на входните въздействия и формиране на изходните реакции. Програмируемите логически контролери са последователностни устройства, в които при формирането на изходните реакции освен входните променливи участват и вътрешни за контролера променливи (таймери, броячи, компаратори и т.н.). В много ограничени случаи, програмируем логически контролер може да изпълнява само комбинационни логически функции (изходните реакции се определят еднозначно само от входните въздействия). Продължителността на тази фаза зависи от броя и вида на използваните инструкции.

Принцип на работа на програмируемите логически контролери.

Фаза 3 се нарича сканиране на изходите (Output Scan). Физическите места, през които промишленият контролер изпраща изходните сигнали към управлявания обект се наричат изходни точки (Output Points). Както при входните точки, така и за всяка изходна точка е определен по един бит в оперативната памет – изходен бит (Output Bit). В тази област от оперативната памет се създава „образ” на изходните реакции в дадения момент (Process Image Outputs). По време на тази фаза се извършва прочитане на изходните битове и прехвърлянето на информацията от тях в изходните точки (физическите адреси на изходите).

Фаза 4 е за обслужване на комуникацията (Service Communication). По време на нея се извършва комуникация с други устройства, например програмиращо устройство, централен компютър, устройство за интерфейс човек-машина (операторски пулт, панел, терминал) и др. В повечето програмируеми логически контролери, комуникацията се обслужва главно от операционната система на контролера, според специални директиви, записани в управляващата програма. Някои контролери позволяват директно управление на комуникацията от потребителската програма чрез съответната системна функция.

Принцип на работа на програмируемите логически контролери.

Фаза 5 е за изпълнение на системни функции (Housekeeping and Overhead). По време на нея се извършват разнообразни действия, по-важните от които са за управление на:

– паметта;

– вътрешните регистри на микропроцесора;

– вградените специални апаратно-програмни функции (Firmware Function) като броячни входове, импулсни входове и изходи, допълнителни комуникационни портове и др.;

– вградените логически функции, които имитират работата на хардуерни устройства, използувани за целите на логическото управление като устройства за времезадържане (таймери), устройства за отброяване на външни или вътрешни за контролера събития (броячи) и др.;

– тестване на апаратно-програмните функции на промишления контролер (Self Diagnostic Test).

Принцип на работа на програмируемите логически контролери.

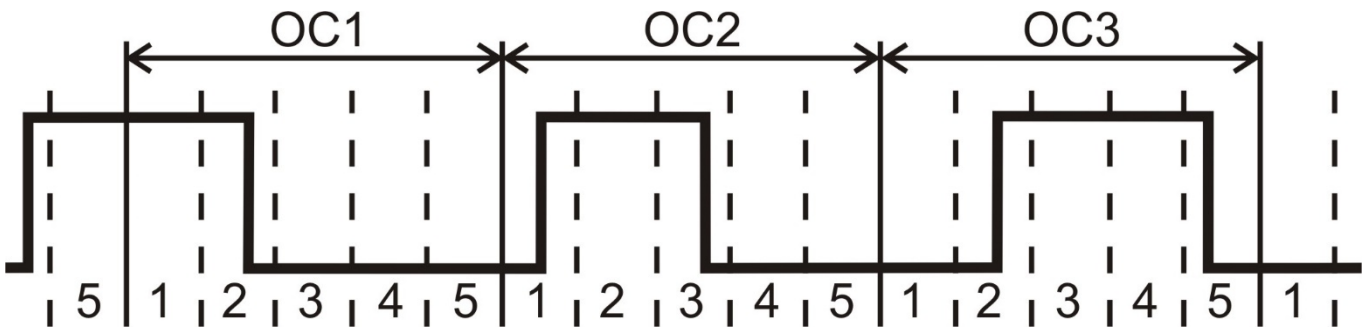
Последователното изпълнение на отделните фази води до някои ограничения, които трябва да се имат предвид при избора на подходящ контролер за дадено приложение. Двете основни ограничения които съществуват са минималната продължителност на входните сигнали и максималното време на реакция.

Минимална продължителност на входните сигнали е параметър, който определя избора на контролер и на подходящи алгоритмични и апаратни подходи за правилното четене на входовете. Обстоятелствата, които налагат взимането на подобни решения са:

- асинхронно активиране на входните точки спрямо оперативния цикъл;
- логическата програма няма достъп до физическите адреси на входните точки, а работи с областта от паметта Process Image Inputs;
- операционната система обновява тази област само във фазата Input Scan.

Принцип на работа на програмируемите логически контролери.

Показаните входни въздействия са с продължителност, по-малка от максималната продължителност на оперативния цикъл TOC_{max}. В първия оперативен цикъл, входното въздействие попада във времето на сканирането на фазата IN и ще бъде отчетено. Във втория оперативен цикъл, входното въздействие се променя по време на фазата на сканиране на входовете и може да бъде или да не бъде отчетено, в зависимост от моментът на стробиране на входната информация. В третия оперативен цикъл входното въздействие се променя извън фазата на сканиране на входовете и не може да бъде отчетено.

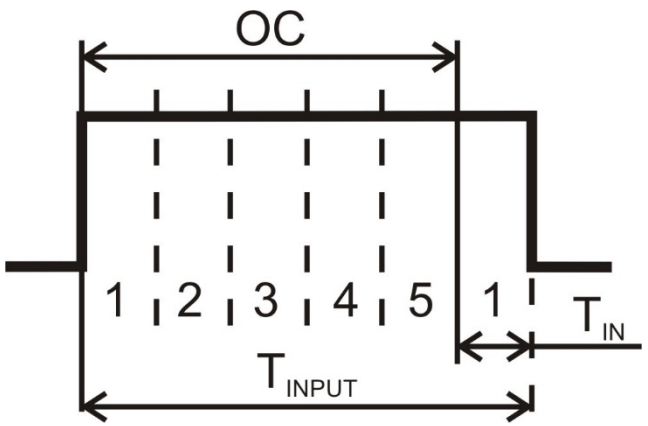


Принцип на работа на програмируемите логически контролери.

Анализът на тези три случая показва, че за да попадне дадено въздействие във фазата на сканиране на входовете е необходимо неговата продължителност да бъде по-голяма от максималната продължителност на оперативния цикъл.

За да се гарантира правилното възприемане на входното въздействие е необходимо то да бъде активно през поне една цяла фаза по сканиране на входовете, а това е възможно само ако минималната продължителност на входното въздействие е равна на максималното време на оперативния цикъл плюс максималното време за сканиране на входовете:

$T_{INPUT} = TOC_{max} + T_{IN}$.



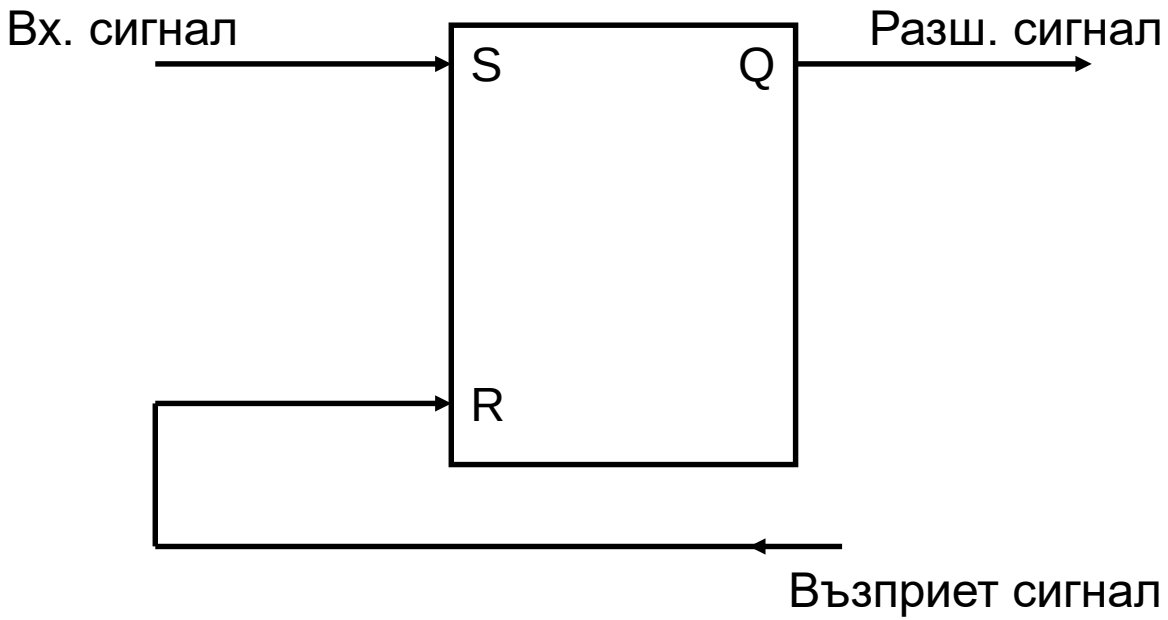
Принцип на работа на програмируемите логически контролери.

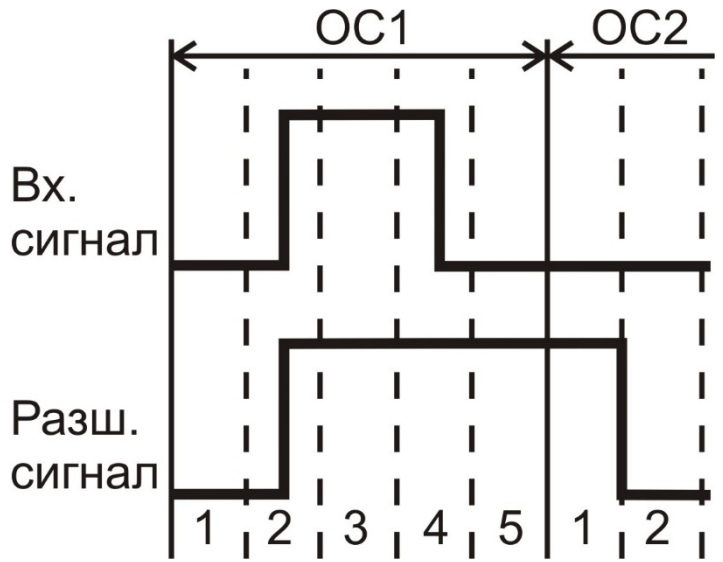
Следователно входни въздействия с продължителност, по-малка от минималната не могат да бъдат обработени от стандартните средства на операционната система на промишления контролер. За такива случаи се въвеждат специални методи, които най-често се свеждат до два.

Първият е чрез „разтягане на импулса”, което се реализира по апаратно-програмен път. Активният фронт на входното въздействие се запомня в апаратен тригер докато премине фазата на сканиране на входовете. След това тригерът се нулира по програмен начин, само ако е възприето въздействието.

Вторият начин е чрез използване на прекъсвания. Активирането на входния сигнал прекъсва оперативния цикъл, независимо в каква фаза на изпълнение се намира, и се изпълнява програма за обработка на прекъсването. След завършването на нейното действие изпълнението на оперативния цикъл се възстановява от мястото на прекъсването.

Принцип на работа на програмируемите логически контролери.



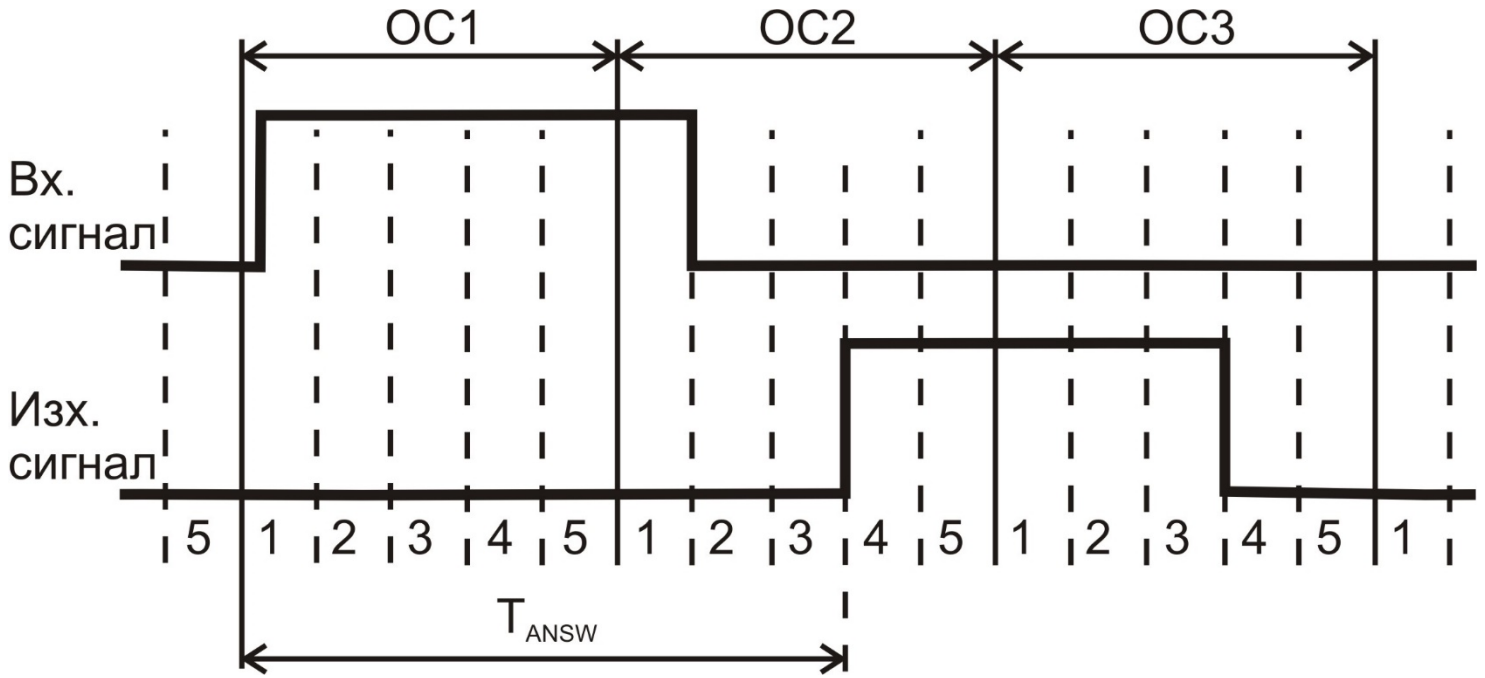


Апаратно – програмно разтягане на импулса

Принцип на работа на програмируемите логически контролери.

Друг важен параметър на промишлените контролери е максималното време на реакция. То се определя от момента на постъпване на входното въздействие до изработването на изходна реакция, съгласно алгоритъма за управление на обекта или процеса. От направените по-горе разглеждания за минимална продължителност на входното въздействие следва, че в най-лошия случай активирането на входа ще се отчете във втория оперативен цикъл, т.е. може да се запише:

$T_{ANSW} = T_{INPUT} + T_{PRG} + T_{OUT} = 2TOC - T_{H\&O} - T_{COM}$.



Процесорът на контролера S7-1200 поддържа следните видове блокове с код, които позволяват създаването на ефективна структура на потребителската програма:

● Организационните блокове (OB) определят структурата на програмата.

● Функциите (FC) и функционалните блокове (FB) съдържат програмния код, който съответства на специфични задачи или комбинации от параметри. Всеки FC или FB осигурява набор от входни и изходни параметри за споделяне на данни с блока, който ги извиква. FB също използва свързан блок за данни (DB), за да поддържа състоянието на стойностите по време на изпълнението, които може да бъдат използвани от други блокове в програмата. FC или FB може да се извика от OB или от други FC или FB, до следните дълбочини на извикване:

- 16 от програмния цикъл или от стартов OB
- 4 от прекъсване

● Блоковете от данни (DBs) съхраняват данни, които могат да бъдат използвани от програмните блокове.

Организационният блок (OB) отговаря за конкретно събитие в CPU и може да прекъсне изпълнението на потребителската програма. По подразбиране за цикличното изпълнение на потребителската програма (OB 1) осигурява основната структура на потребителската програма. Ако се включат други OB в програмата, тези OB прекъсват изпълнението на OB 1. Другите OB изпълняват специфични функции, като например за задачи в STARTUP режим, за обработка на прекъсвания и грешки, или за изпълнение на специфичен програмен код в определени интервали от време.

Функционален блок (FB) е подпрограма, която се изпълнява при извикване от друг блок с код (OB, FB или FC).

Функцията (FC) е подпрограма, която се изпълнява при извикване от друг блок с код (OB, FB или FC). Тя не се асоциира с конкретна DB. Повикващият блок предава параметри към FC. Изходните стойности от FC трябва да бъдат записани на адрес в паметта или в глобална DB.

Процесорният модул на контролера S7-1200 има три режима на работа: режим STOP, режим STARTUP и режим RUN.

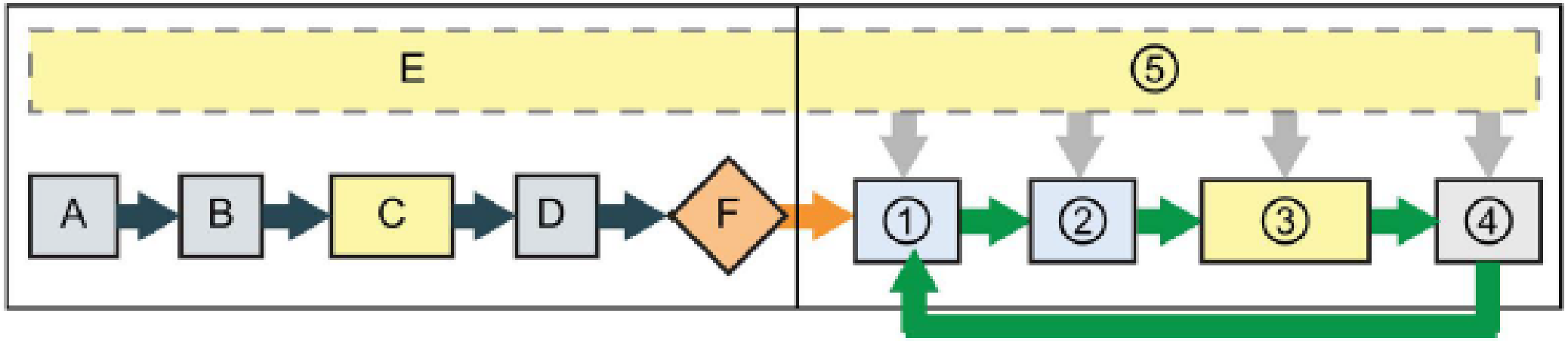
Текущият режим на работа се индицира от светодиоди на лицевия панел.

● В режим STOP процесорът не изпълнява програмата.

● В режим STARTUP стартовите организационни блокове - OBs (ако са налични) се изпълняват веднъж. Прекъсвания не се обработват по време на този режим.

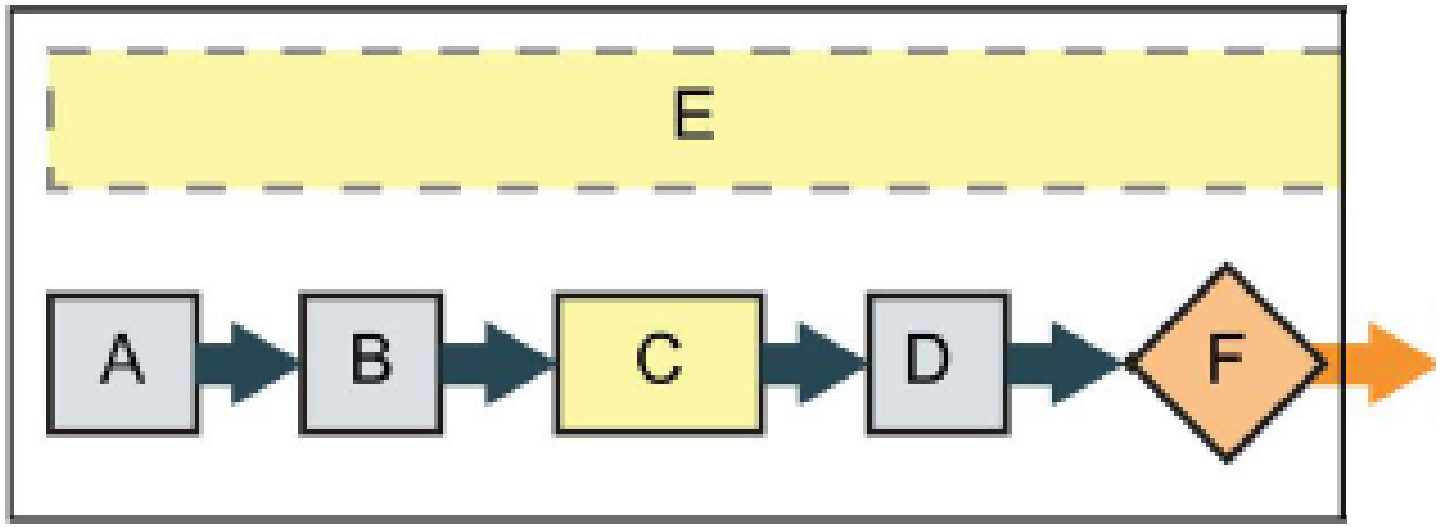
● В режим RUN оперативният цикъл се изпълнява многократно. Прекъсвания се приемат и се обработват във всяка фаза на оперативния цикъл в режим RUN.

Последователност на изпълнението на задачите на модул CPU на контролера S7-1200 в режим STARTUP и в режим RUN



STARTUP RUN

STARTUP

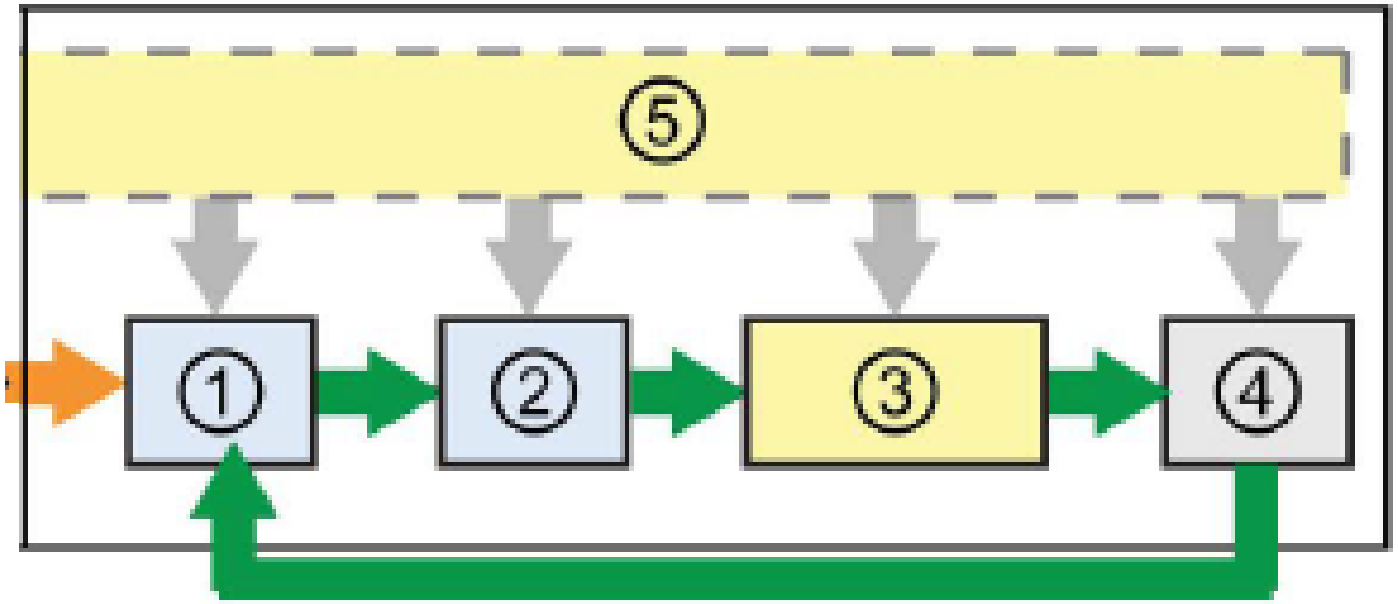


A – Нулиране на областта I от паметта.
B – Инициализиране областта Q от паметта с нула, последната стойност, или заместващата стойност, както е конфигурирана.
C – Инициализиране областта M от паметта и блоковете данни с тяхната първоначална стойност и разрешение на конфигурираното циклично прекъсване.
Изпълнение на началните OB.
D – Копиране състоянието на физическите входове в областта I от паметта.
E – Запазване на всички заявки за прекъсване в опашка за обработка след въвеждане Режим RUN.
F – Разрешение на копирането от областта Q на паметта във физическите изходи.

Когато режимът на работа се промени от STOP на RUN, процесорът нулира паметта на входните битове и обработва стартовите OBs. Всяко четене на входните битове чрез инструкции в стартовите OBs ще прочете по-скоро нула отколкото текущата физическа стойност. Ето защо, за да се прочете текущото състояние на физически вход по време на стартирането, трябва да се извърши непосредствено четене (immediately read - I_:P). Стартиращите OBs и всякакви други свързани FC и FB се изпълняват. Ако има повече от един стартиращ OB, те се изпълняват по ред според номера, най-ниският номер се изпълнява първи.

В режим RUN програмния цикъл се изпълнява циклично. Основният блок на програмата е програмен цикъл OB. В него са инструкциите, които управляват програмата и от където се извикват допълнителни потребителски блокове.

RUN



1 – Копиране от областта Q на паметта във физическите изходи.
2 – Копиране състоянието на физическите входове в областта I от паметта.
3 – Изпълнение на OB от програмния цикъл.
4 – Изпълнение на процедури за самодиагностика.
5 – Обработване на заявките за прекъсване и комуникациите по време на всяка фаза на цикъла.

За всеки цикъл на сканиране (оперативен цикъл), CPU записва изходите, чете входовете, изпълнява потребителската програма, актуализира комуникационните модули и обработва потребителски прекъсвания и заявки за комуникация. Заявките за комуникация се обработват периодично в целия цикъл.

Тези действия (с изключение на потребителски прекъсвания) се обслужват редовно и последователно. Потребителските прекъсвания, които са активирани, се обслужват според приоритета в реда в който се случват. При прекъсване, процесорът чете входовете, изпълнява OB и след това записва изходите, като използва съответния дял на образа на процеси (PIP), ако е приложимо.

Системата гарантира, че цикълът на сканиране ще бъде завършен за период, наречен максимално време на цикъла. В противен случай се генерира събитие за грешка във времето.

Времето, което е необходимо на процесора за изпълнение на цикъла се нарича време на оперативния цикъл.

Cycle time	Range (ms)	Default
Maximum scan cycle time[1]	1 to 6000	150 ms
Minimum scan cycle time[2]	1 to maximum scan cycle time	Disabled

CPU модулът непрекъснато следи оперативния цикъл и реагира когато времето е надвишено.

В случаите когато CPU модулът завърши нормалния оперативен цикъл за по-кратко от зададеното време, останалото време се използва за самодиагностика и/или обработка на заявки за комуникации.

За изравняване на времето на оперативния цикъл се използва минималното време на цикъла. Когато то е зададено, CPU модулът изпълнява закъснение след завършване на оперативния цикъл докато изтече минималното зададено време на цикъла.

Всеки цикъл на сканиране започва с извличане на текущите стойности на цифровите и аналоговите изходи от образа на изходните реакции и след това запис във физическите изходи на CPU, SB, и SM модули, конфигурирани за автоматична актуализация - I/O update (конфигурация по подразбиране). Когато физически изход се промени чрез инструкция, стойността му както в образа на изходните реакции, така и на самия физически изход се актуализира.

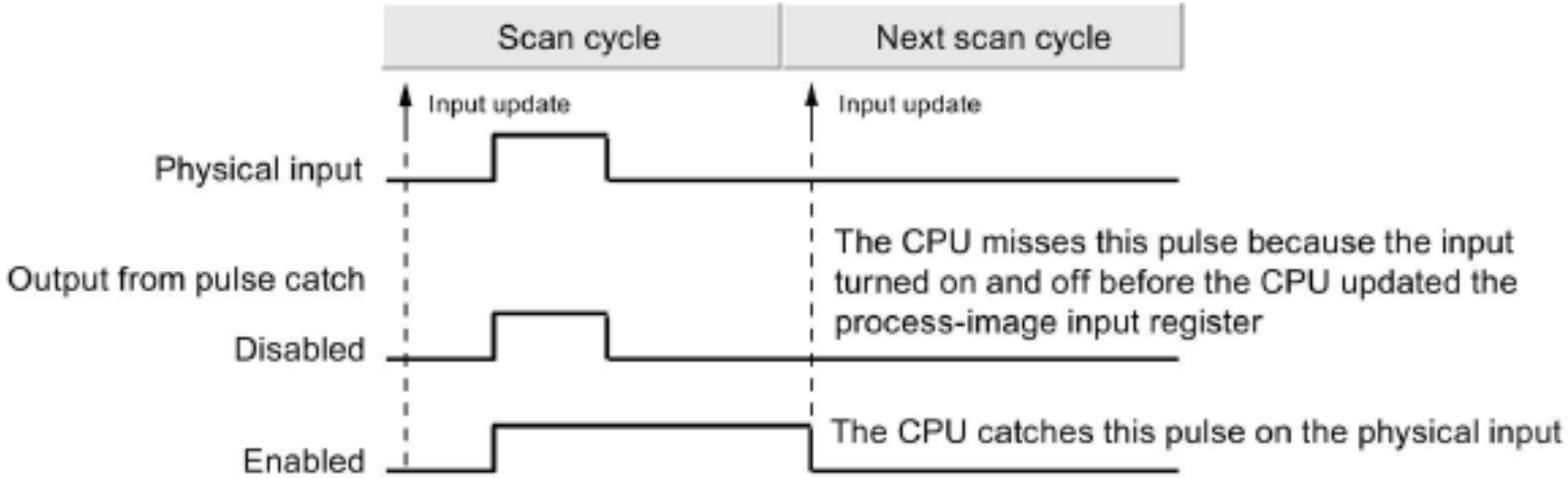
Цикълът на сканиране продължава с четене на текущите стойности на цифровите и аналоговите входове от CPU, SB и SM, конфигурирани за автоматична актуализация (конфигурация по подразбиране), и след това запис на тези стойности в образа на входните въздействия. При четене на физически вход чрез инструкция, стойността на физическия вход се обработва от инструкцията, но образът на входните въздействия не се актуализира.

След прочитане на входовете, потребителската програма се изпълнява от първата инструкция до крайната инструкция. Това включва всички организационни блокове (OBs) на програмния цикъл и всичките свързани с тях FC и FB. Организационните блокове на програмния цикъл се изпълняват по реда на техните номера - OB номер с най-ниския номер се изпълнява първи.

Обработката на комуникациите се осъществява периодично по време на цикъла, може и с прекъсване изпълнението на потребителската програма.

Самодиагностиката включва периодични проверки на състоянието на системата и на I/O модули.

Прекъсванията могат да възникнат по време на която и да е част от цикъла на сканиране и са предизвикани от събития. Когато постъпи събитие, процесорът прекъсва цикъла на сканиране и извиква OB, който е конфигуриран да обработва това събитие. След като OB завърши обработката на събитието, процесорът възобновява изпълнението на потребителска програма от точката на прекъсване.



CPU модулът на S7-1200 има възможност за разширяване на кратки входни импулси, които няма да бъдат регистрирани, ако промяната на сигнала се извършва извън фазата за четене на входовете.

Когато „pulse catch“ опцията за даден цифров вход е разрешена логическото ниво след активният преход на сигнала се запомня и се задържа до следващия оперативен цикъл. Това гарантира, че кратък импулс ще бъде регистриран и задържан докато CPU модулът прочете входовете.

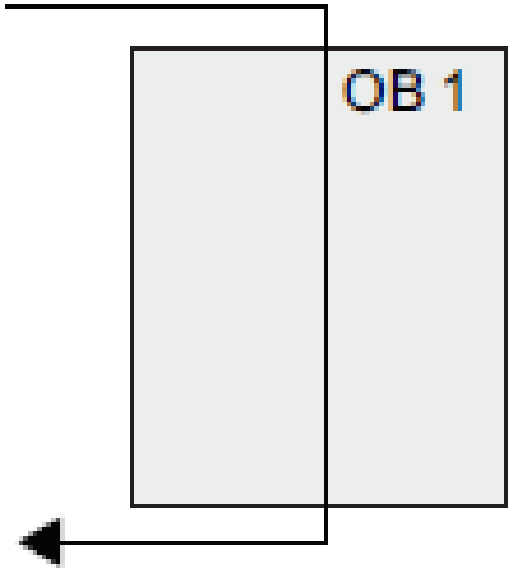
Структура на потребителската програма. Според изискванията на приложението, може да се избере линейна структура или модулна структура за създаване на потребителската програма:

● Линейната програма изпълнява всички инструкции на задачите за автоматизация последователно, една след друга. Обикновено линейната програма поставя всички инструкции на кода в един OB за циклично изпълнение на програмата (OB 1).

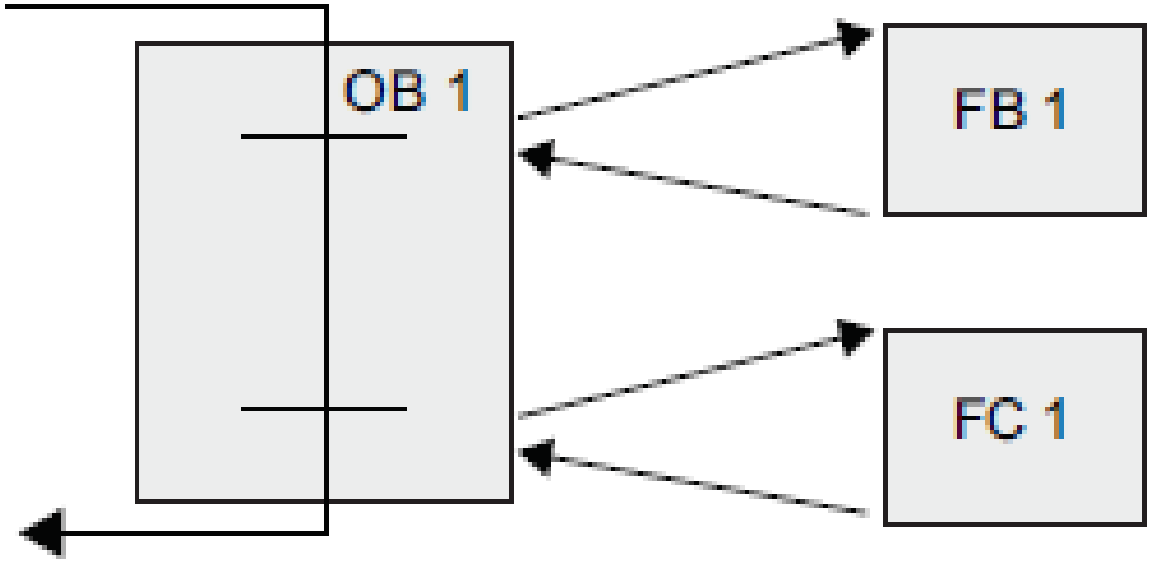
● Модулната програма извиква специфични блокове код, които изпълняват специфични задачи. За да се създаде модулна структура, сложната задача за автоматизация се разделя на по-малки подчинени задачи, които съответстват на технологичните функции на процеса. Всеки блок код осигурява програмния сегмент за всяка подчинена задача. Програмата се структурира чрез извикване на даден блок код от друг блок.

Чрез създаване на типови блокове код, които могат да бъдат използвани повторно в рамките на потребителската програма, може да се опрости проектирането и изпълнението на потребителската програма.

Структура на потребителската програма



Линейна
структура



Модулна
структура

Структура на потребителската програма

Използването на генерични (типови) блокове код има следните предимства:

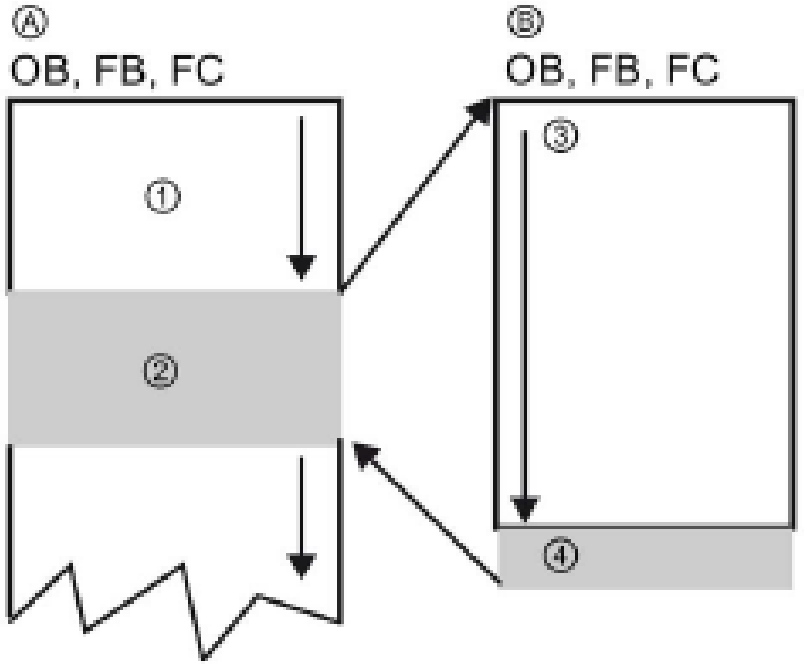
● Може да се създават блокове с код за многократна употреба за стандартни задачи, като например за управление на помпа или двигател. Може също така тези типови блокове код да се съхраняват в библиотека, от която да се използват за различни приложения или решения.

● Когато потребителската програма се структурира чрез използване на модулни компоненти, които са свързани с функционалността на задачите, проектирането на потребителската програма става по-лесно за разбиране и управление. Модулните компоненти не само помагат за стандартизиране на проектирането на програмата, но също така помагат актуализирането или модифицирането на програмния код да се извърши по-бързо и по-лесно.

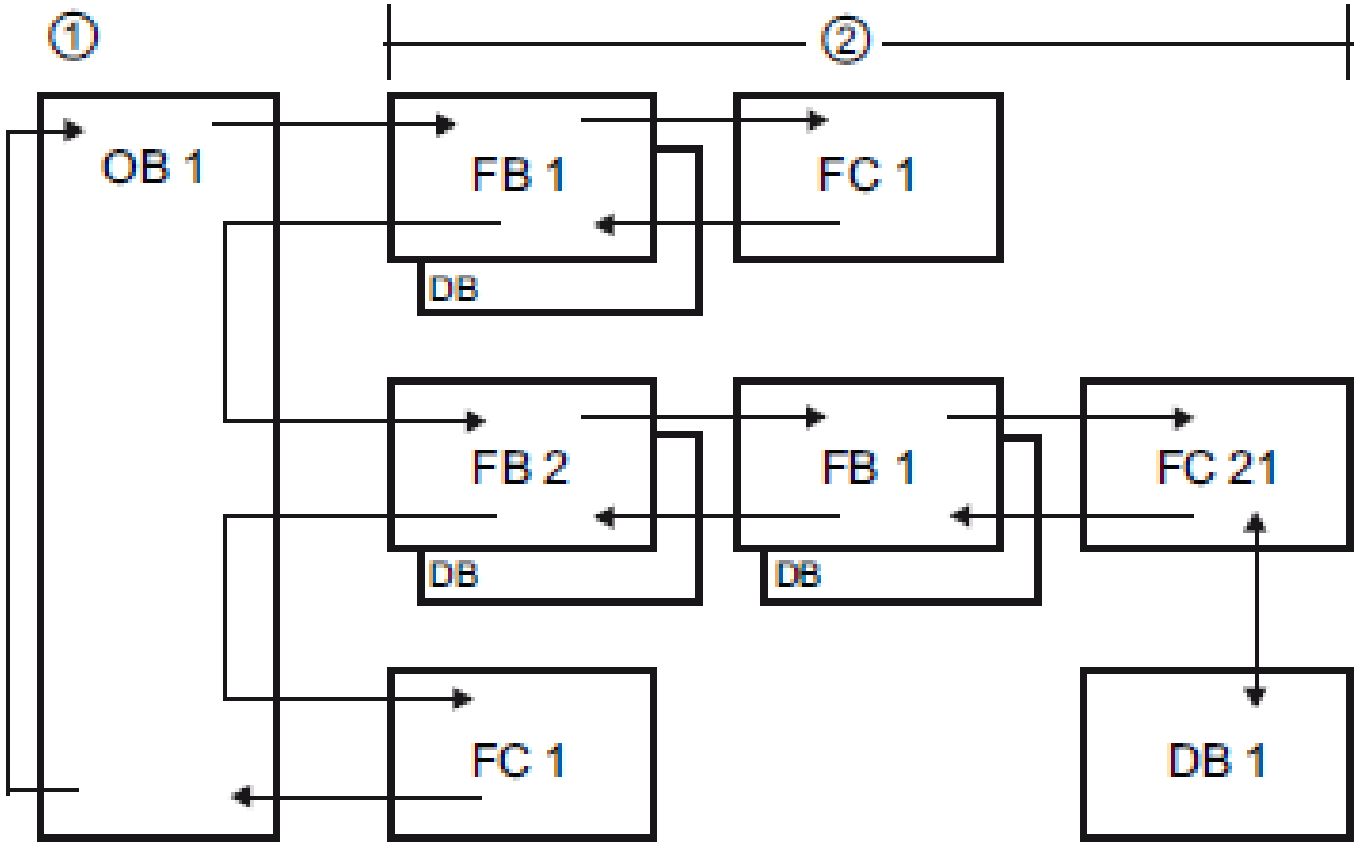
● Създаването на модулни компоненти опростява отстраняването на грешки в програмата. Структурирането на пълната програма като набор от модулни сегменти позволява да се тества функционалността на всеки блок код, когато е разработен.

Структура на потребителската програма

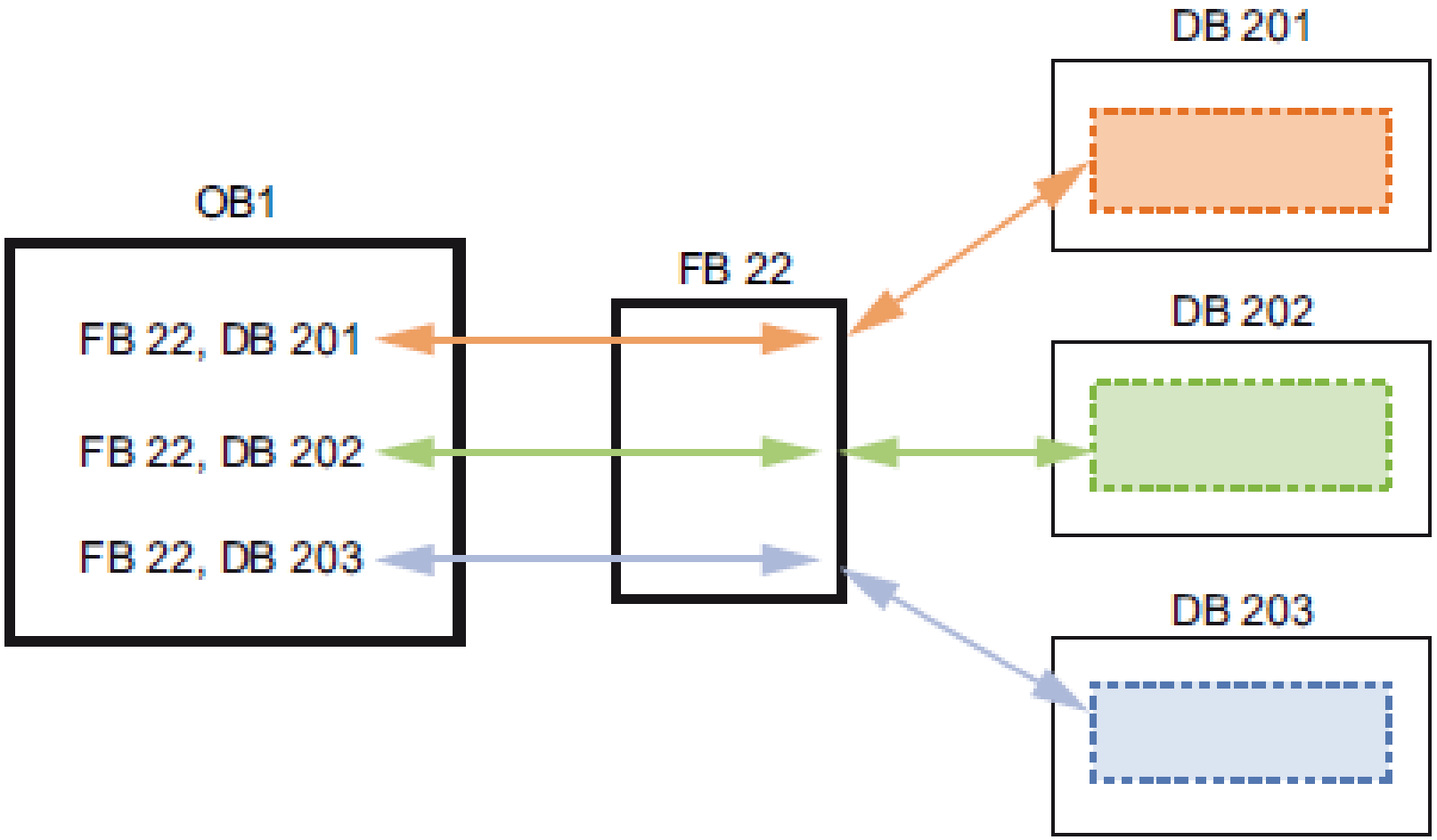
Когато един блок код извиква друг блок код, процесорът изпълнява програмния код в извикания блок. След като изпълнението на извикания блок код е завършено, процесорът възобновява изпълнението на инструкциите от повикващия блок. Обработката продължава с изпълнението на инструкция, която следва извикването.



A – извикващ блок
B – извикан (или прекъсващ) блок
① – изпълнение на програмата
② – инструкция или събитие, което инициира изпълнението на друг блок
③ – изпълнение на програмата
④ – край на извикания блок (връщане към извикващия блок)



Модулна структура – дълбочина на вложени блокове



Изпълнение на един и същ функционален блок с различни блокове данни

Повикващият блок предава параметри на FB и също така идентифицира специфичен блок данни (DB), който съхранява данните за конкретното повикване или заявка от този FB. Промяната на зададената база данни позволява на FB с общо приложение да управлява работата на набор от устройства.

Тази структура позволява на един общ FB да контролира няколко подобни устройства, като например двигатели, чрез присвояване на различен блок данни за всяко повикване за различните устройства.

Всяка DB база данни съхранява данните (като скорост, време за ускоряване и общо време на работа) за отделно устройство.

В този пример FB 22 управлява три отделни устройства, като DB 201 съхранява оперативната база от данни за първото устройство, DB 202, съхранява оперативните данни за второто устройство, и DB 203 съхранява оперативните данни за третото устройство.

Обикновено FB се използват за управление на задачи или устройства, които не завършват действието си в рамките на един цикъл на сканиране. За съхраняване на работните параметри, така че да могат да бъдат бързо достъпни от едно сканиране към следващо, всеки FB в потребителската програма има една или повече оперативни бази данни.

Основни групи инструкции

Размерът на потребителската програма, данните и конфигурацията са ограничени от наличната памет за зареждане и работната памет в CPU. Няма конкретно ограничение за броя на отделните блокове OB, FC, FB и DB. Въпреки това, общият брой на блоковете е ограничен до 1024.

1. Инструкции от контактен тип (контактни инструкции). Това са инструкции за обработка на отделен бит. С тях се обработва т.нар. „контактни” битове от данновата област, съответстващи на логически елементи с две състояния – включено и изключено. Такива са инструкциите за двоичните логически операции AND, OR, XOR и др. Към този тип инструкции спадат и инструкциите, чрез които се кодират операциите за логическо установяване Set и нулиране Reset на бит от данновата област.

2. Инструкции за работа с числови операнди, представени чрез байт, дума, дълга дума и числа с плаваща запетая. Делят се на три вида:

– математически инструкции, чрез които се извършват математически операции върху операндите;

– логически инструкции, към които се причисляват операциите по преместване и ротиране на операндите;

– инструкции за преобразуване на типа на информацията, като например преобразуване от двоичен в двоично-десетичен вид и т.н.

3. Инструкции за сравняване, които сравняват операндите и изработват като изход еднобитов резултат. Резултатът от сравнението може да се използува като бит в изразите на контактните инструкции.

4. Инструкции за управление на хода на потребителската програма като условни разклонения, прескачане на отделни части от програмата, предаване на управлението, извикване на подпрограми и др.

5. Инструкции за ползване на системните функции, имитиращи работата на хардуерни устройства като таймери, броячи и др. Към тях се отнасят и инструкциите за управление на специализирани хардуерни блокове и др.

Ладер диаграми (Ladder logic - LAD)

Ладер диаграмата представлява графично описание на логически инструкции, в които булевите променливи са изразени чрез „контакти” и практически повтаря електрическата схема на изгражданите на базата на релейно-контактни схеми устройства. Състоянията на отделните битове се определя от две основни инструкции – „изпълни при затваряне” (examine if close –| |–) и „изпълни при отваряне” (examine if open –|/|–). Тези две инструкции напълно съответстват на релейните схеми „нормално отворен контакт” и „нормално затворен контакт”. Всъщност те изразяват по какъв начин участват булевите променливи в логическото уравнение – с правата или с инверсната си стойност.

По аналогия с релейно-контактните схеми, всяко логическо уравнение се представя чрез последователни или паралелни връзки между логическите променливи („контакти”) и формира т.нар. „стъпало” (contact network) в ладер диаграмата. Началото на всяко стъпало започва с условна „захранваща линия” (power line). Резултантното състояние на всеки изходен (или междинен) бит се представя чрез „бобина“ (coil – ( )), чийто логически смисъл е присвояване на стойност.

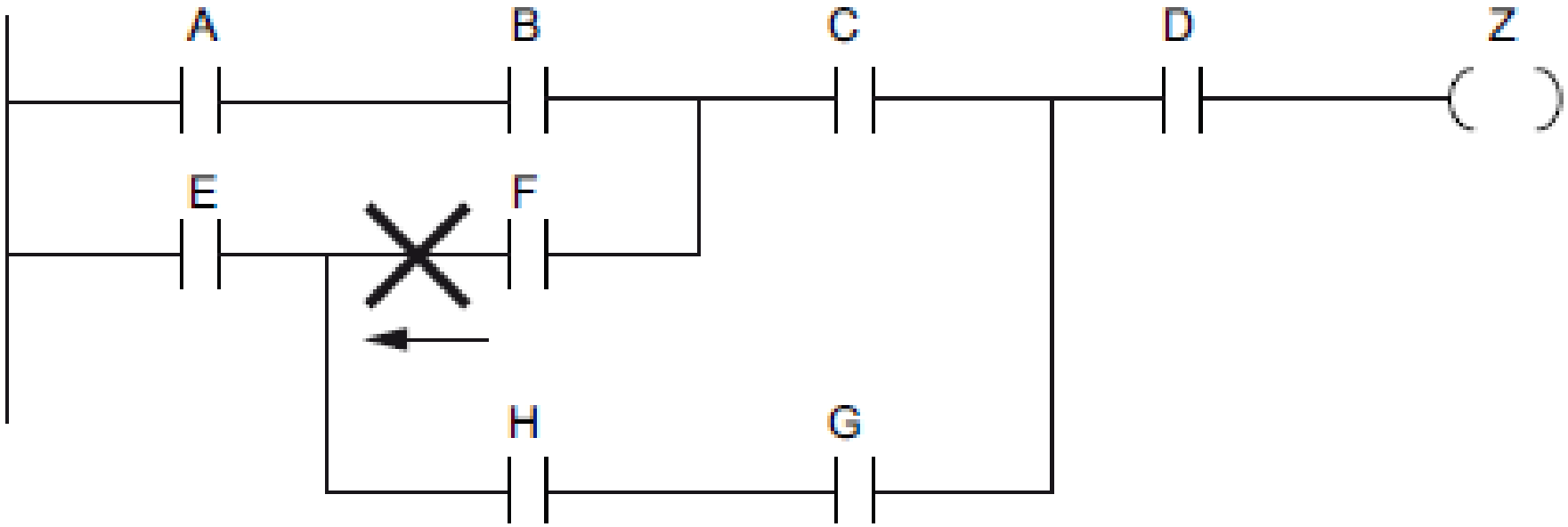
Ладер диаграми (Ladder logic - LAD)

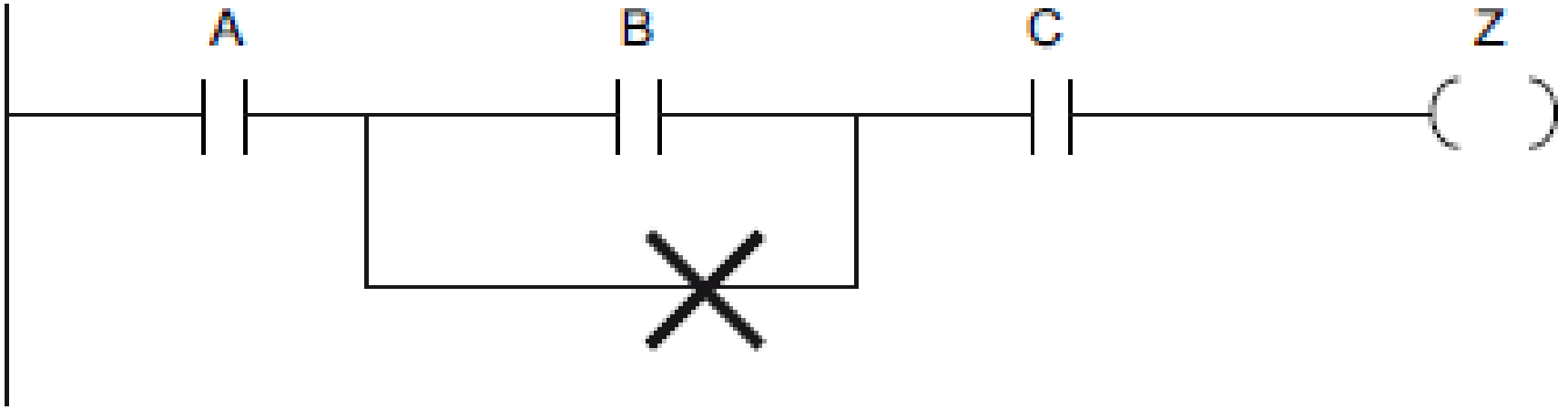
Отделните стъпала в ладер диаграмите се обработват от операционната система на промишления контролер последователно отгоре надолу. Логическата комбинация между отделните контакти в ладер диаграмите се нарича „тестова зона“ или „логически условия“, а участващите контактни инструкции – „входни инструкции”.

Резултатът от логическата операция може да служи като условие за извършването на определена логическа операция. За тези цели се ползват т.нар. „специални бобини” (special coil – (xxx)). Те могат да бъдат:

1. Функции на схемите:
   – установяване на бит в 1 – Set – (S) ;
   – нулиране на бит – Reset – (R) .
2. Функции за организиране на програмата:
   – безусловен преход към етикет – Jump to Label – (J) ;
   – извикване на подпрограма с номер n – Call Subroutine – (Call n);
   – връщане от подпрограма – Return from Subroutine – (Ret) ;
   – край на потребителска програма – End of Program – (END) .

Ладер диаграми (Ladder logic - LAD)





Ладер диаграми (Ladder logic - LAD)

Условната „бобина“ се нарича още „изходна инструкция“, което означава, че в зависимост от резултата на изпълнението на логическото уравнение могат да се изпълняват значително по-сложни инструкции, като например „бокс инструкции“ (box).

Бокс инструкциите съчетават действието на контактните и числовите инструкции. Те се разделят на две основни групи:

– бокс изходни инструкции за системни функции (system function box), които са вградени в операционната система, като таймери, броячи и др.

– бокс изходни функции за операционни символи (operational box), които представляват логически и математически операции над числови операнди, както и операции за прехвърляне на променливи в различните даннови области.

Нормално бокс инструкциите могат да се използват само като изходни инструкции. Изключение е бокс инструкцията за сравнение (comparison box instruction), която извършва операция над числови операнди и формира двоичен (булев) резултат

Някои инструкции (напр. математически или за преместване) предоставят параметри за разрешение (EN и ENO). Тези параметри се отнасят до подаването на захранване в LAD или FBD и определят дали инструкцията ще се изпълнява по време на това сканиране.

● EN (Enable In) е двоичен вход. На този вход трябва да има захранване (EN = 1), за да се изпълни бокс инструкцията. Ако EN входът на LAD бокс е свързан директно към захранването, инструкцията винаги ще бъде изпълнена.

● ENO (Enable Out) е двоичен изход. Ако боксът получава захранване на входа EN и изпълнява функцията си без грешка, тогава изходът на ENO подава захранване (ENO = 1) на следващия елемент. Ако се установи грешка при изпълнението на инструкцията, тогава захранването се прекратява (ENO = 0) в бокс инструкцията, която генерира грешката.

● Нормално отвореният контакт е затворен (ON), когато стойността на зададения бит е равна на 1.

● Нормално затвореният контакт е затворен (ON), когато стойността на зададения бит е равна на 0.

● Контакти, свързани последователно реализират И логическа функция.

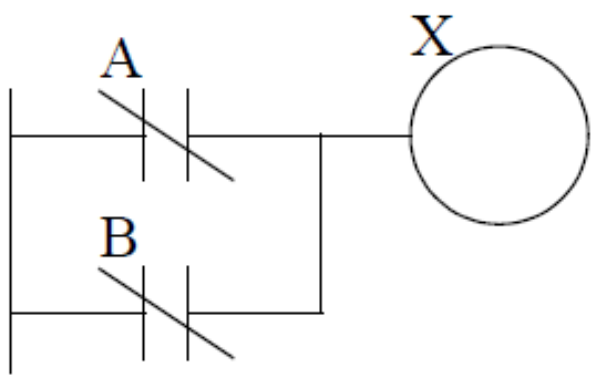
● Свързаните в паралел контакти създават ИЛИ логическа функция.

Parameter	Data type	Description
IN	Bool	Assigned bit

NAND

$$X = \overline{A \cdot B}$$

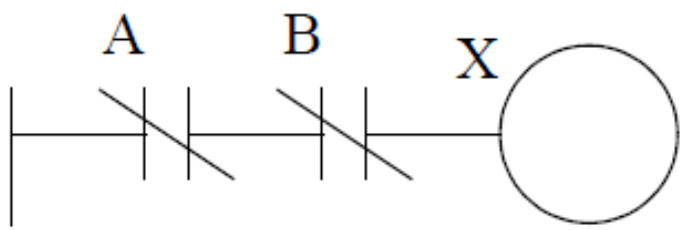
$$X = \overline{A} + \overline{B}$$



NOR

$$X = \overline{A + B}$$

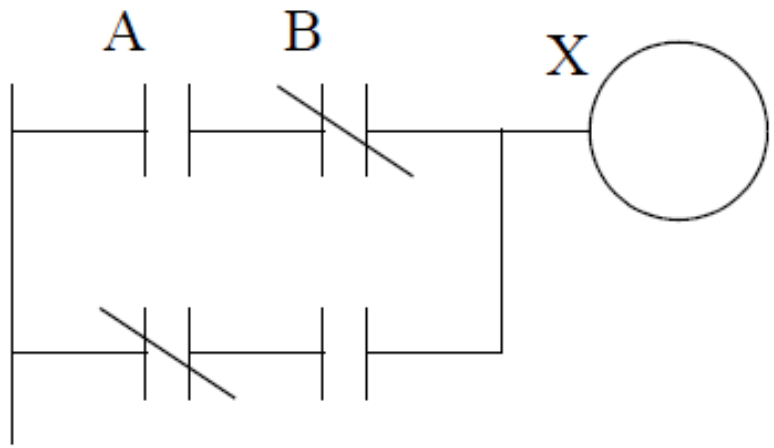
$$X = \overline{A} \cdot \overline{B}$$



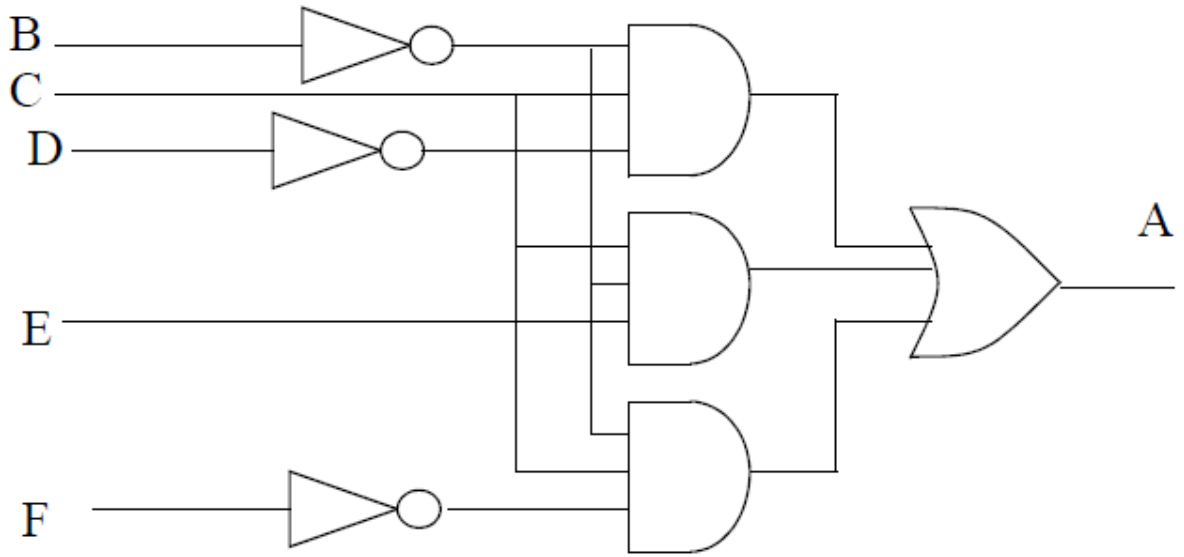
EOR

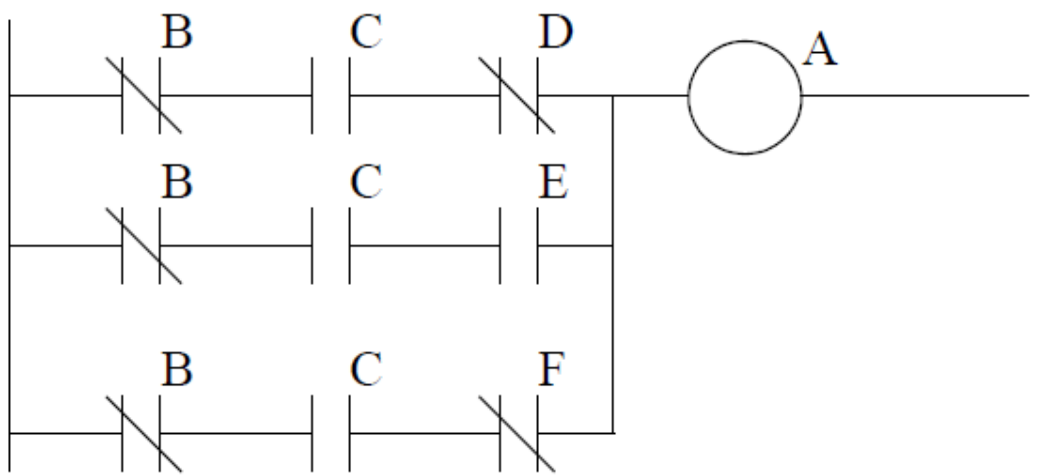
$$X = A \oplus B$$

$$X = A \cdot \overline{B} + \overline{A} \cdot B$$

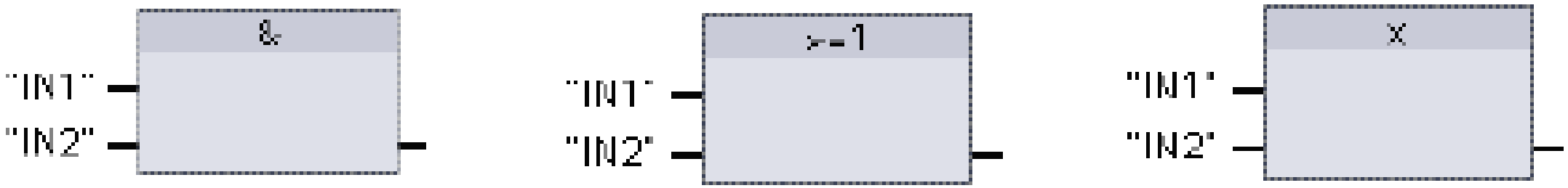


$$A = (\bar{B} \cdot C \cdot \bar{D}) + (\bar{B} \cdot C \cdot E) + (\bar{B} \cdot C \cdot \bar{F})$$





При програмирането с използването на FBD, LAD контактните стъпала се трансформират в AND (&), OR (> = 1) и XOR (x) бокс инструкции, където може да се задават стойности на битовете за входовете и изходите. Може също така да се свързват с други логически бокс инструкции и да се създават собствена комбинационна логика.

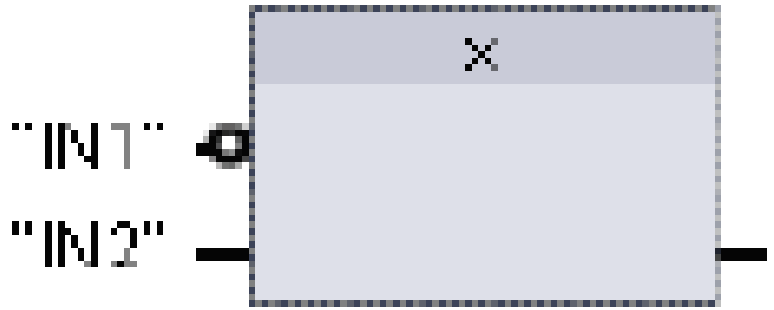


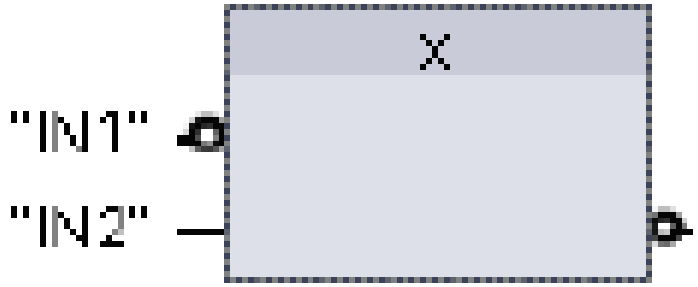
Parameter	Data type	Description
IN1, IN2	Bool	Input bit

Контактът LAD NOT инвертира логическото състояние на входа.

1) Ако в контакта NOT не постъпва захранване, тогава се подава захранване навън.

2) Ако в NOT контакт постъпва захранване, тогава не се подава захранване навън.



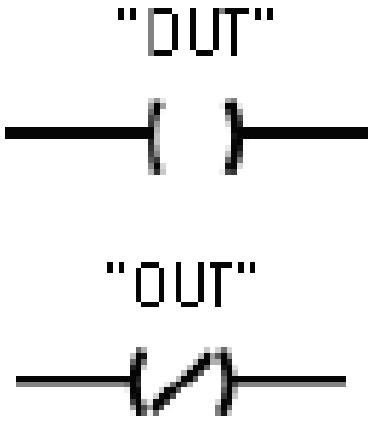


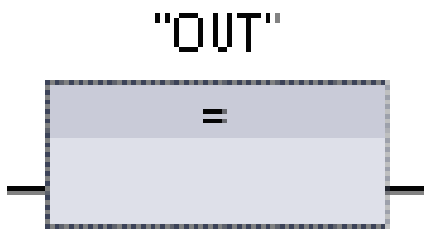
● Ако има захранване към изходна бобина или FBD "=" бокс инструкция е разрешена, тогава изходният бит е равен на 1.

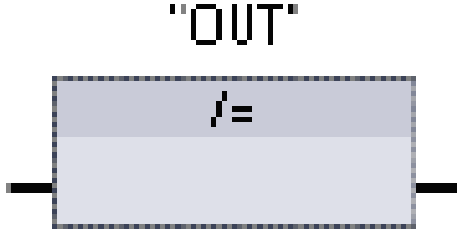
● Ако няма захранване към изходна бобина или FBD "=" бокс инструкция не е разрешена, тогава изходният бит е равен на 0.

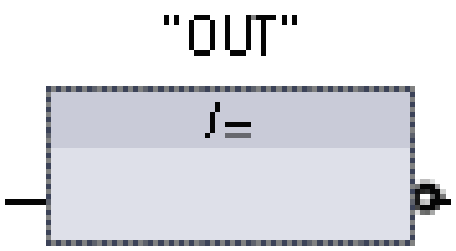
● Ако има захранване към инвертирана изходна бобина или FBD "/ =" бокс инструкция е разрешена, тогава изходният бит е равен на 0.

● Ако няма захранване към инвертирана изходна бобина или FBD "/ =" бокс инструкция не е разрешена, тогава изходният бит е равен на 1.







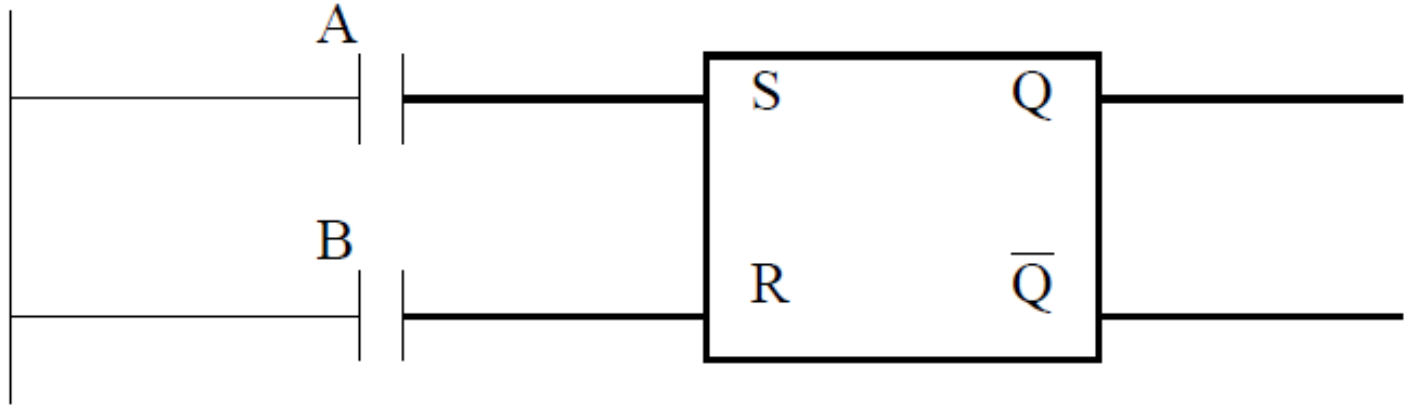


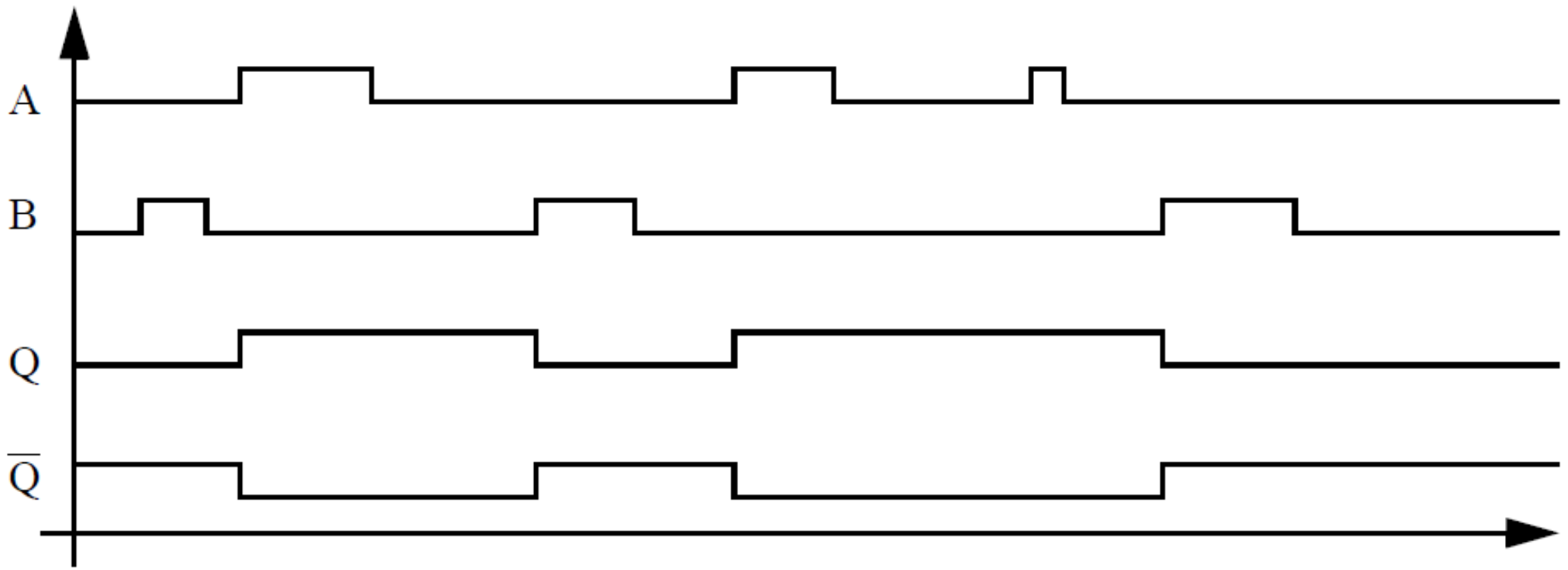
Parameter	Data type	Description
OUT	Bool	Assigned bit

Когато S (Set) е активиран, стойността на данните в адреса OUT е равна на 1. Когато S не е активиран, стойността на OUT не се променя.

Когато се активира R (Reset), стойността на данните в адреса OUT е равна на 0. Когато R не е активиран, стойността на OUT не се променя.

Parameter	Data type	Description
IN (or connect to contact/gate logic)	Bool	Bit location to be monitored
OUT	Bool	Bit location to be set or reset


A
B
S
R
Q
$\overline{Q}$

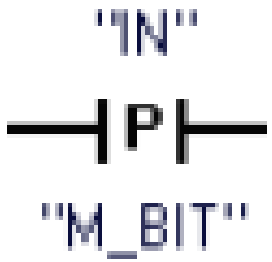

A
B
Q
$\overline{Q}$

Instruction	S1	R	"OUT" bit
RS	0	0	Previous state
	0	1	0
	1	0	1
	1	1	1
	S	**R1**	
SR	0	0	Previous state
	0	1	0
	1	0	1
	1	1	0

"OUT"

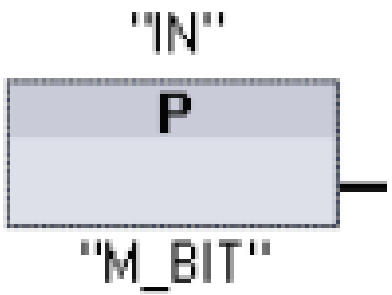
"OUT"

Parameter	Data type	Description
S, S1	Bool	Set input; 1 indicates dominance
R, R1	Bool	Reset input; 1 indicates dominance
OUT	Bool	Assigned bit output "OUT"
Q	Bool	Follows state of "OUT" bit

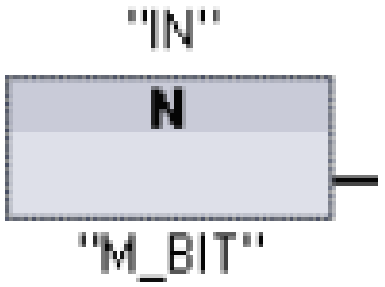


Състоянието на този контакт е TRUE при откриване на положителен преход (OFF-to-ON) на зададения бит "IN".

Състоянието на този контакт е TRUE при откриване на отрицателен преход (ON-to-OFF) на зададения бит "IN".



Изходното логическо състояние е TRUE при откриване на положителен преход (OFF-to-ON) на зададения входен бит.



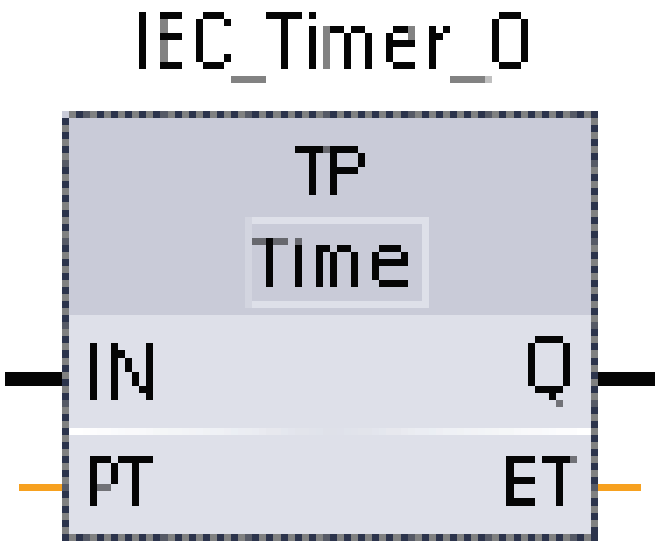
Изходното логическо състояние е TRUE при откриване на отрицателен преход (ON-to-OFF) на зададения входен бит.

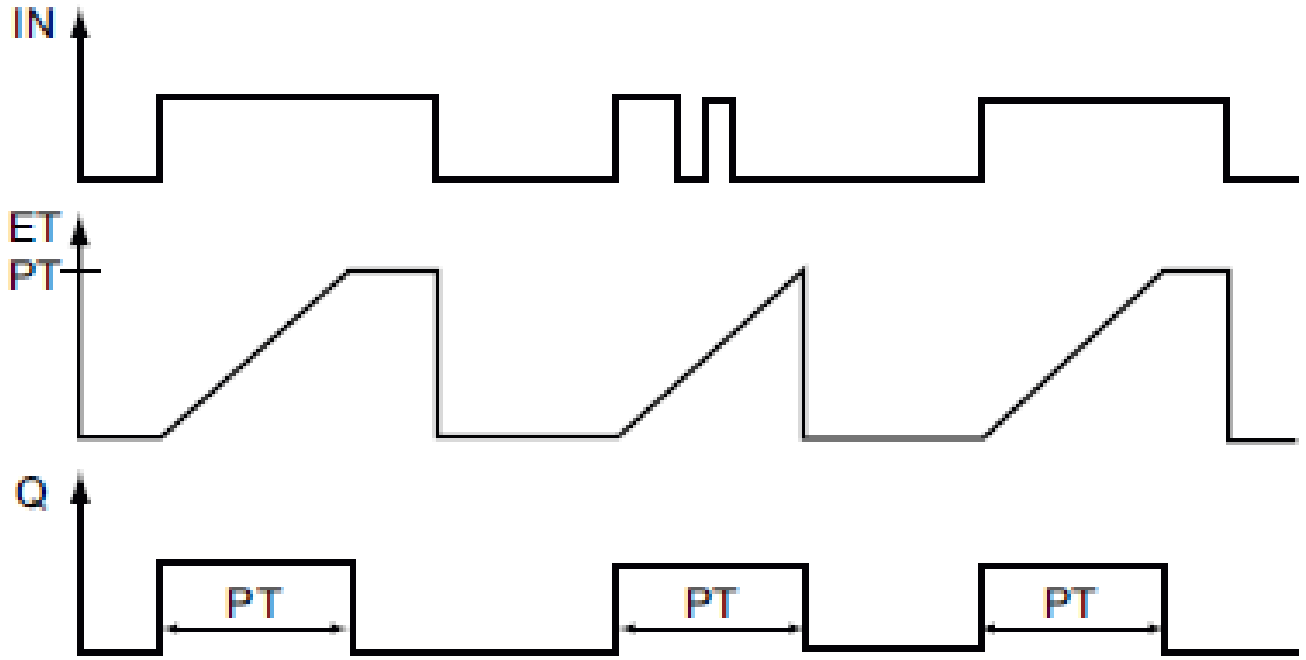
Зададеният бит "OUT" е TRUE, когато е открит положителен преход (OFF-to-ON) на захранването към бобината.

Зададеният бит "OUT" е TRUE, когато е открит отрицателен преход (ON-to-OFF) на захранването към бобината.

Зададеният бит "OUT" е TRUE, когато е открит положителен преход (OFF-to-ON) на входа на бокс инструкцията.

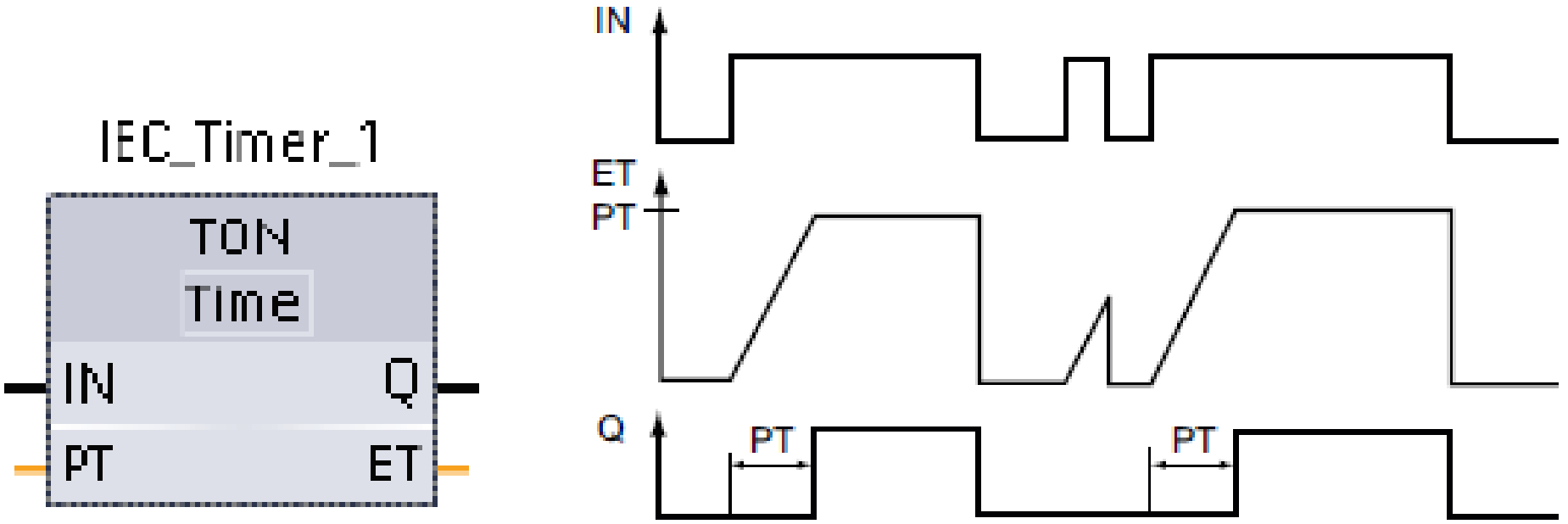
Зададеният бит "OUT" е TRUE, когато е открит отрицателен преход (ON-to-OFF) на входа на бокс инструкцията.





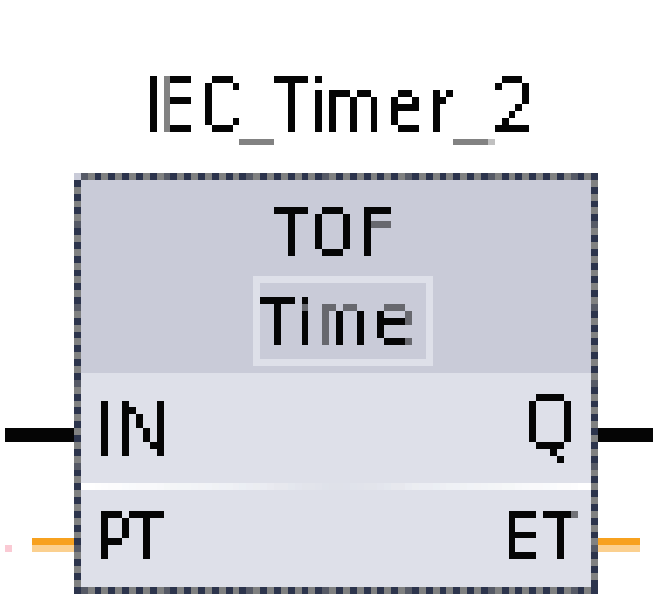
TP: Pulse timer

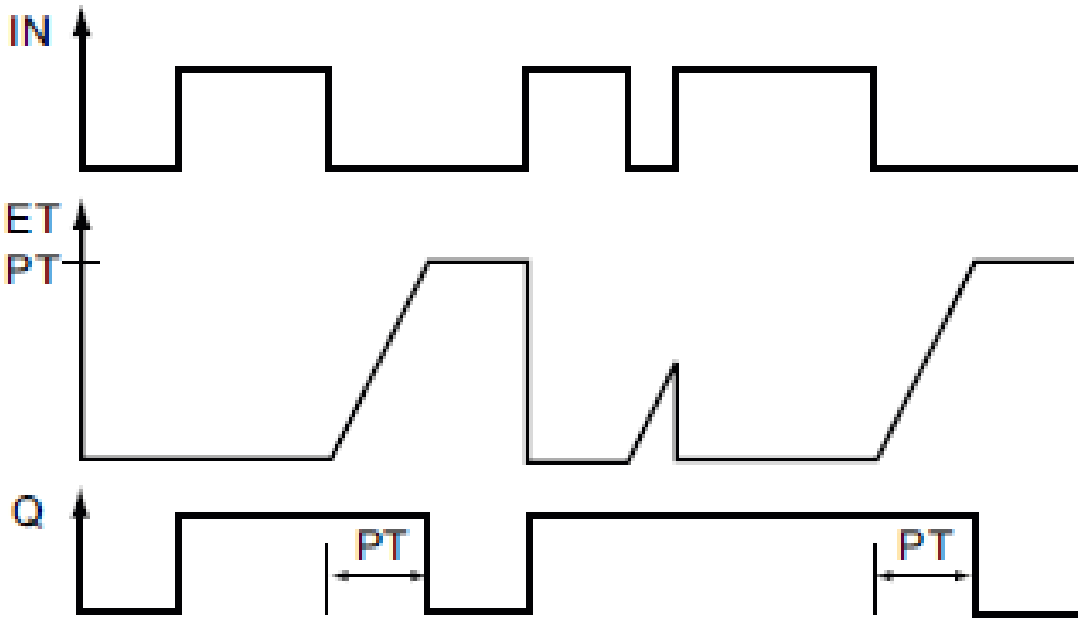
TP таймерът генерира импулс с предварително зададена продължителност.



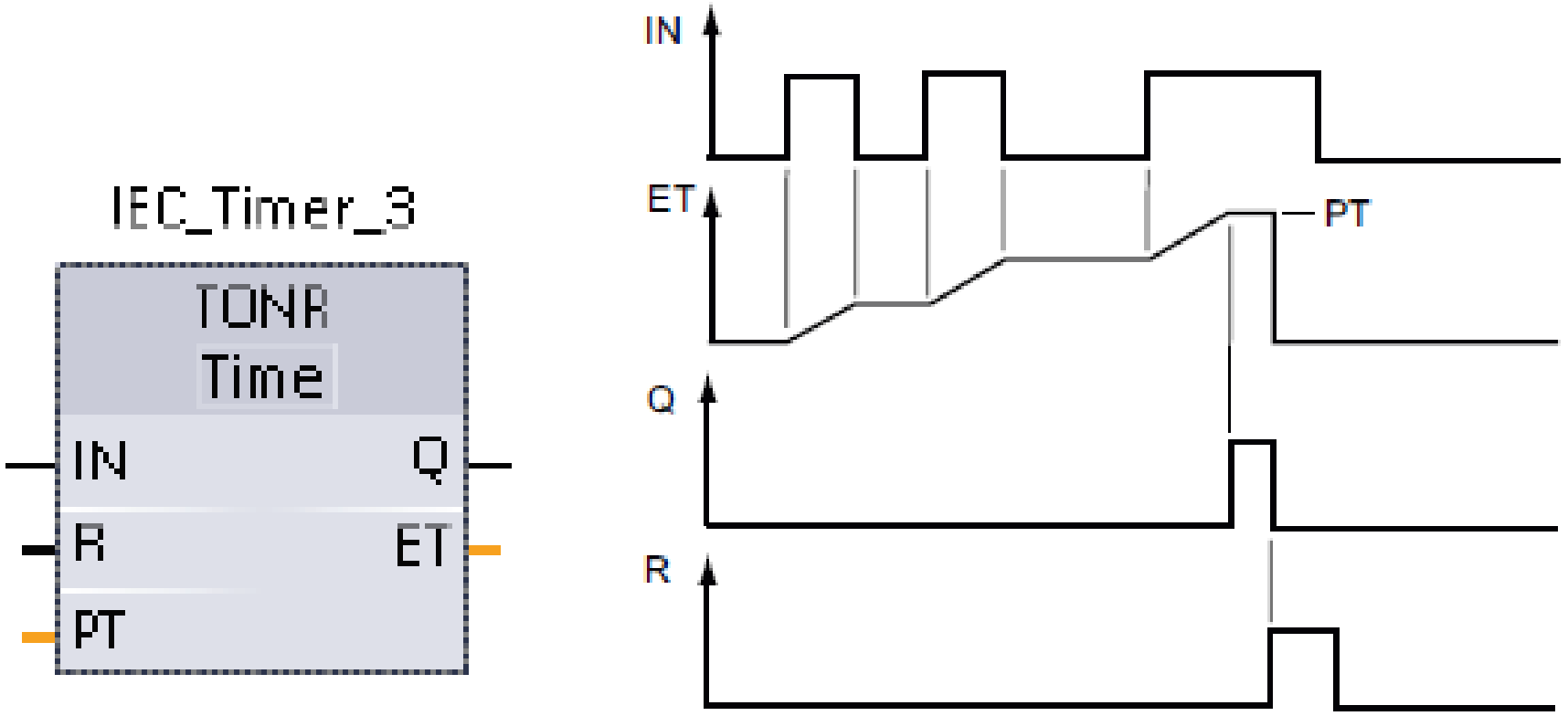
TON: ON-delay timer

TON таймерът установява изхода Q в състояние ON след предварително зададено закъснение.



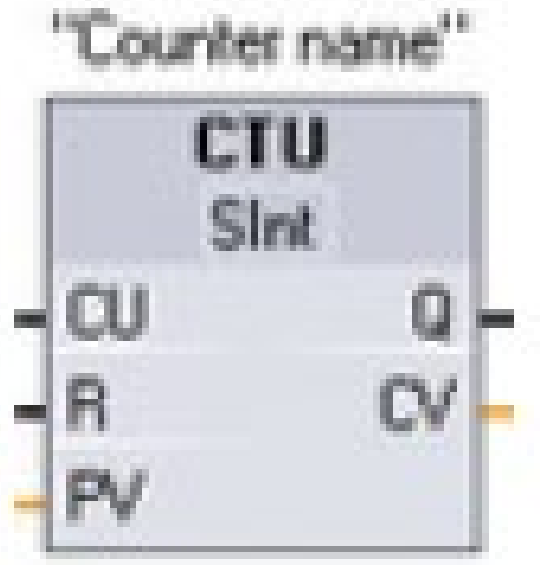


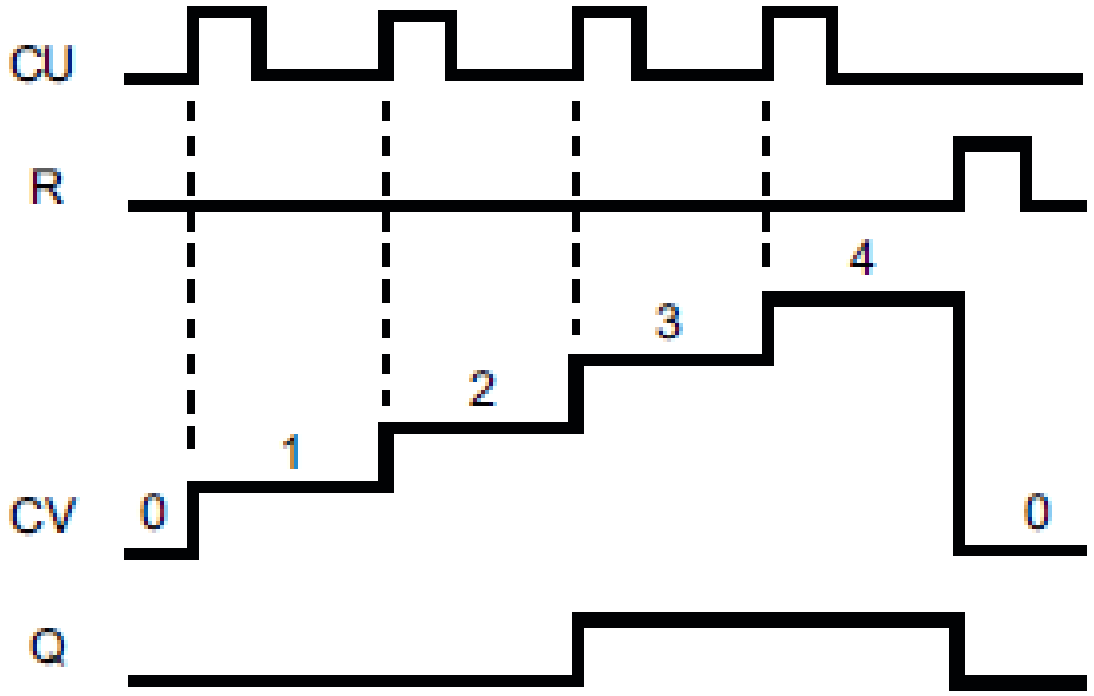
TOF: OFF-delay timer
TOF таймерът нулира изхода Q след предварително зададено закъснение.



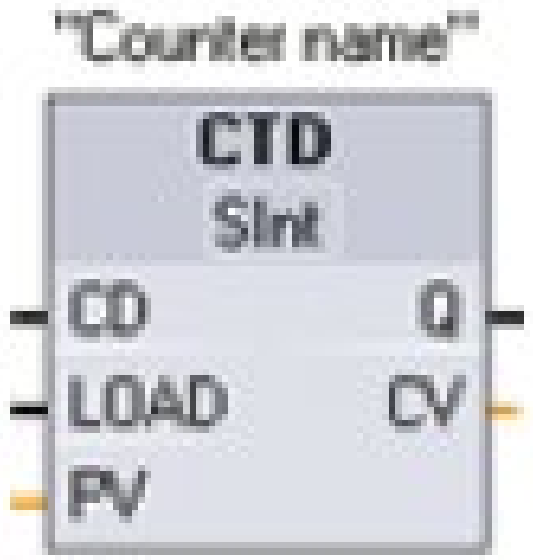
TONR: ON-delay Retentive timer

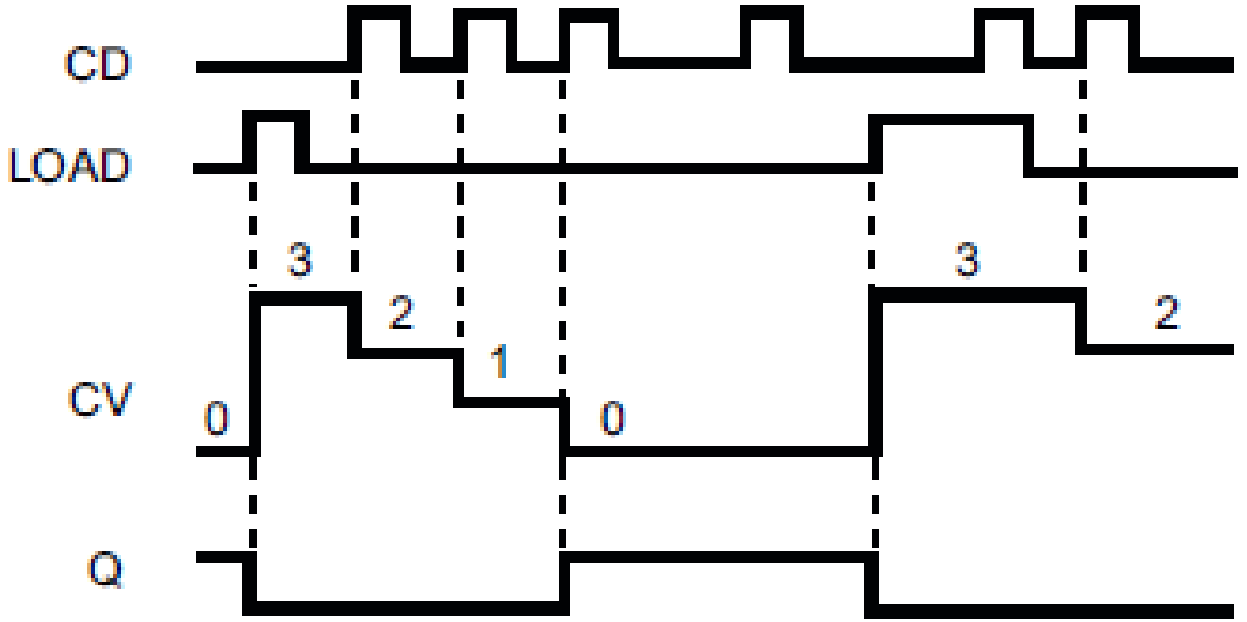
TONR таймерът установява изхода Q в ON след предварително зададено закъснение. Изтеклото време се акумулира в множество времеви периоди докато входа R не бъде използван за нулиране на изтеклото време.



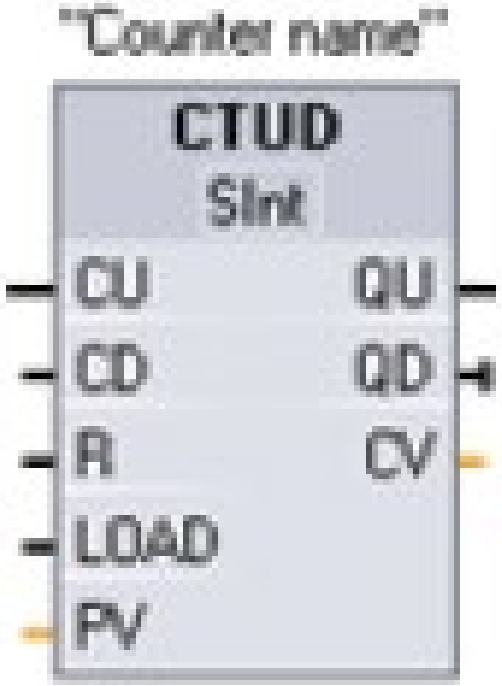


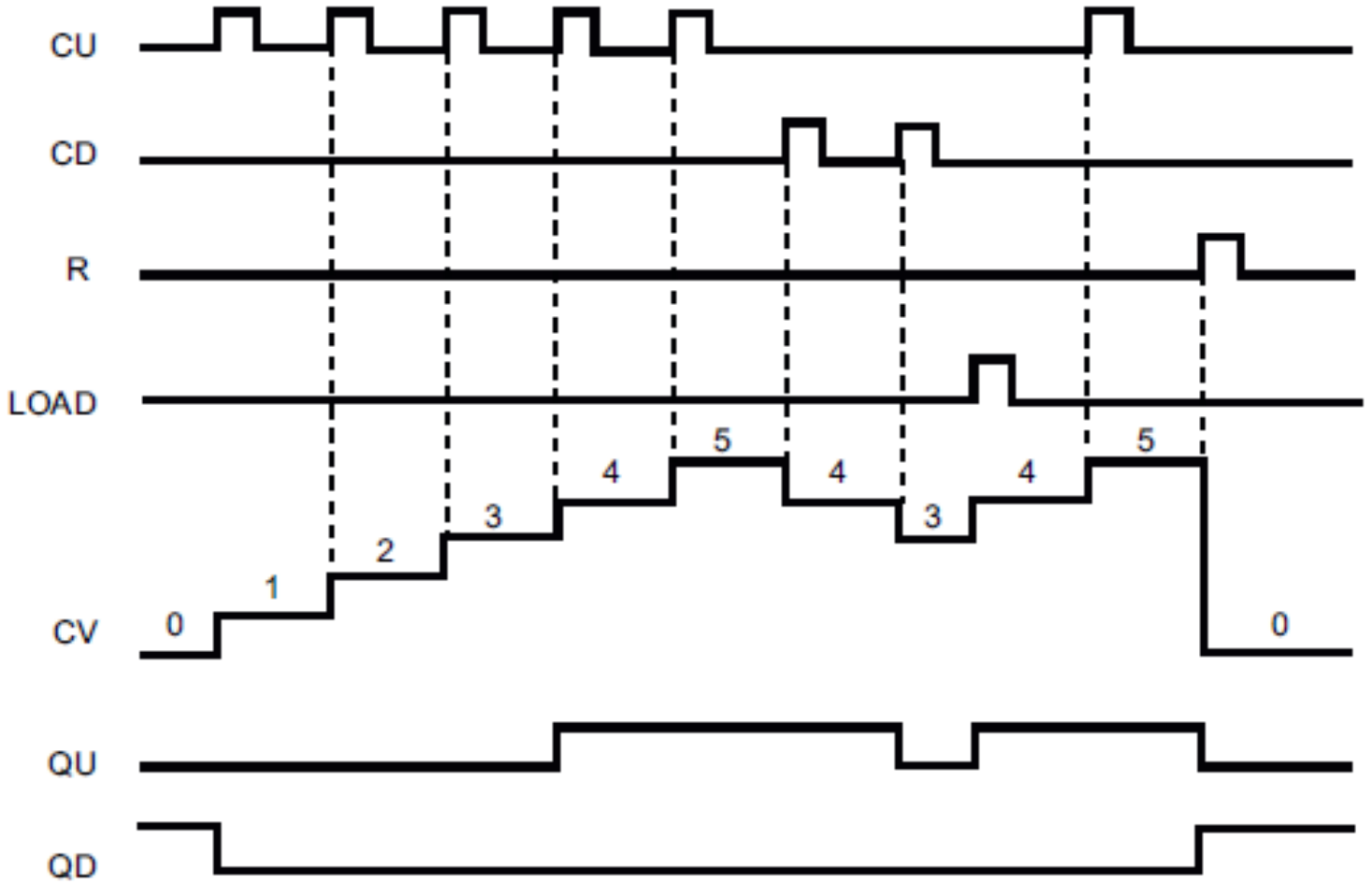
Сумиращ брояч





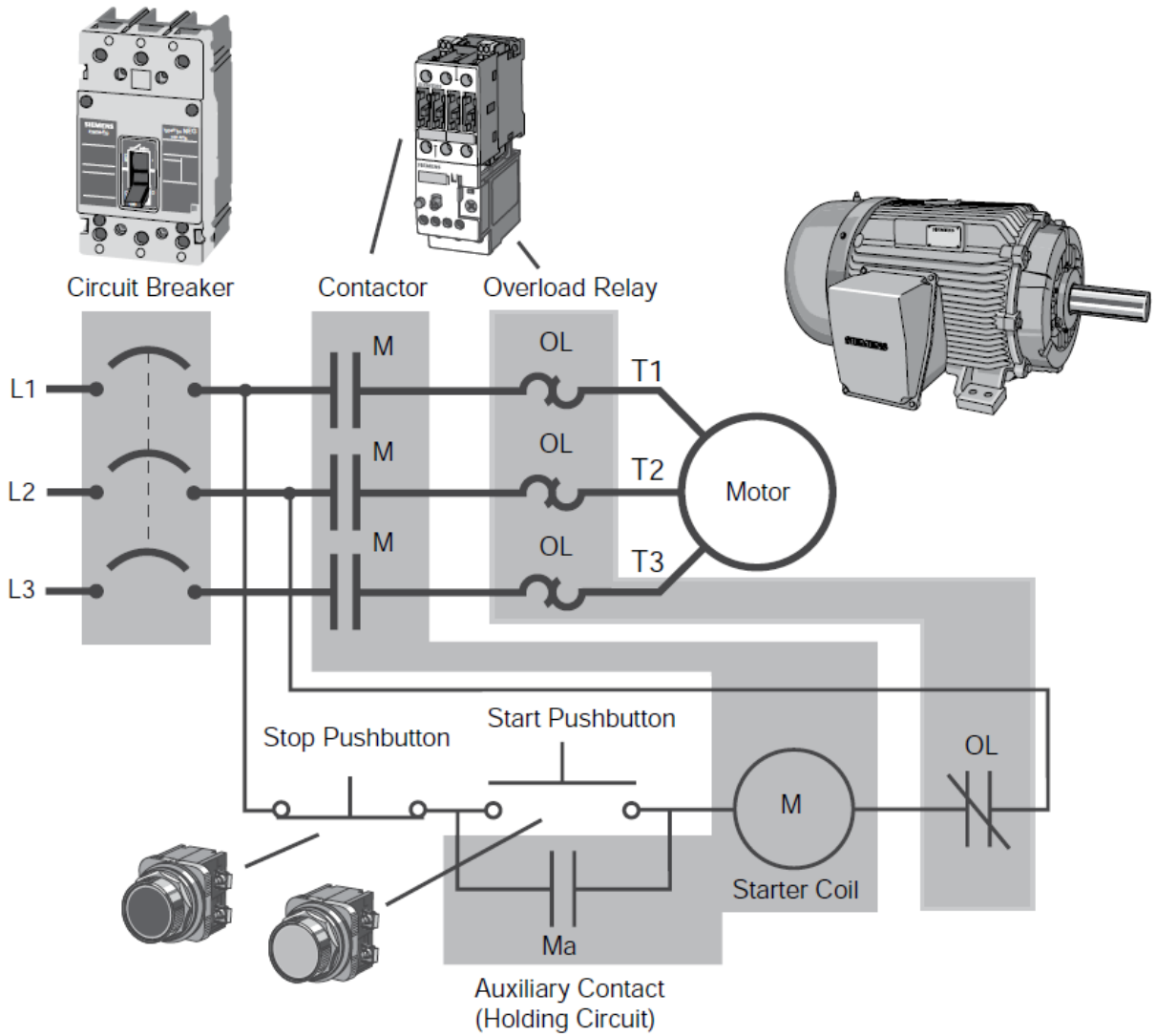
Изваждащ брояч



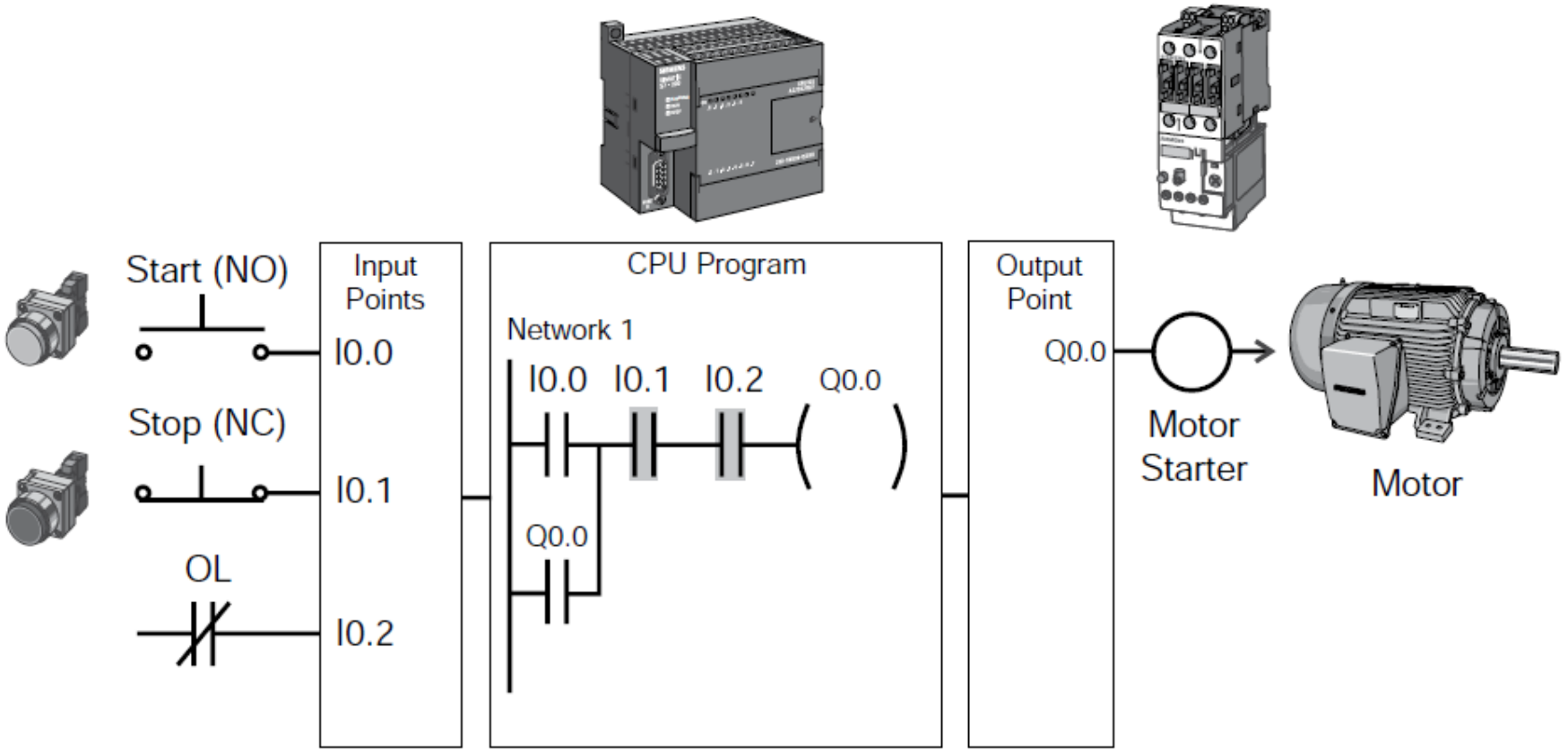


Реверсивен брояч

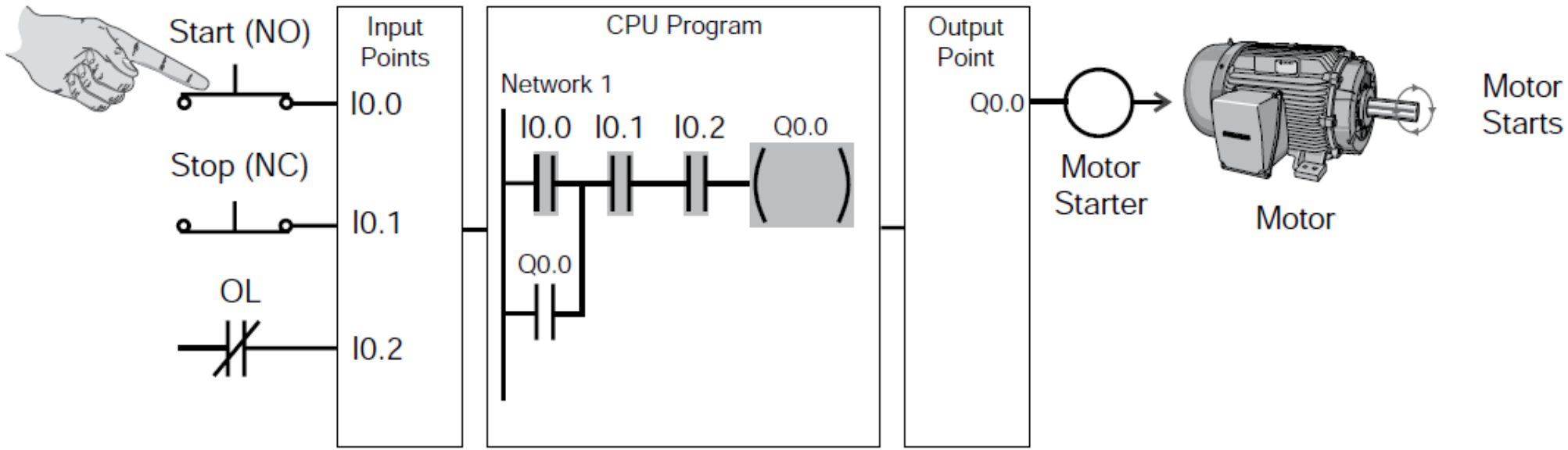
Реализиране на старт-стоп



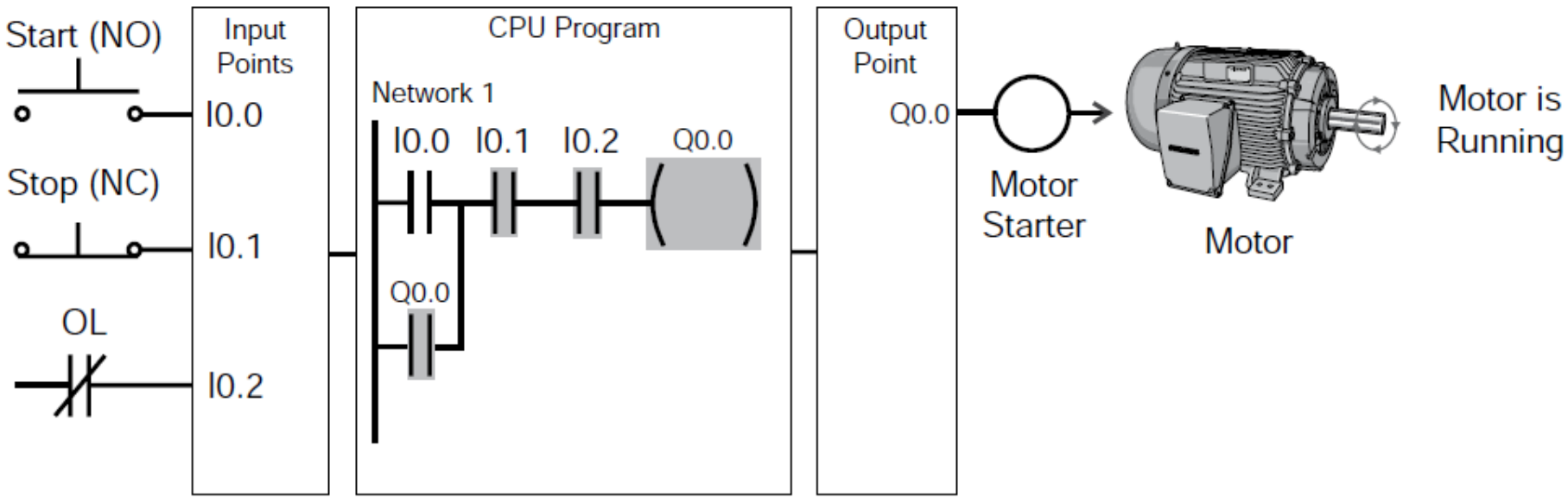
Реализиране на старт-стоп



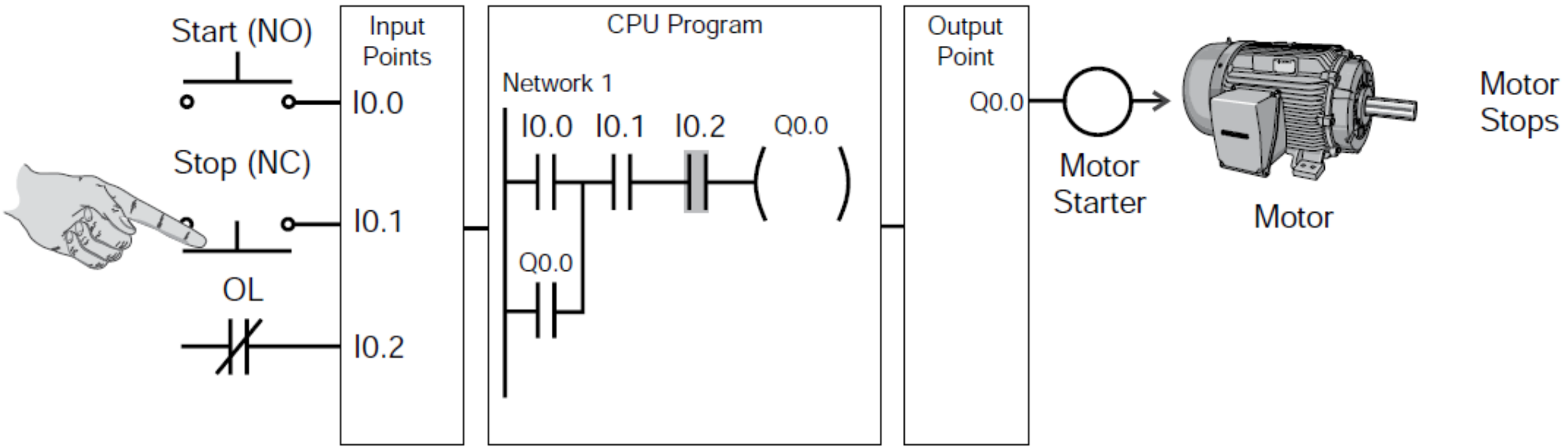
Реализиране на старт-стоп



Реализиране на старт-стоп

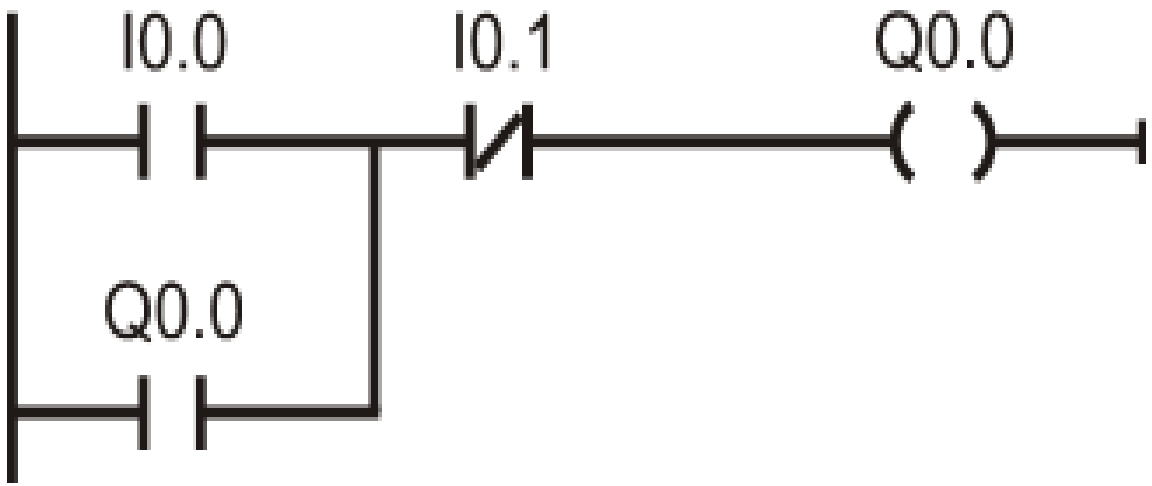


Реализиране на старт-стоп



Реализиране на старт-стоп

Ладер диаграми (Ladder logic - LAD)



Реализиране на старт-стоп

Функционални блокови схеми (Function Block Diagram - FBD)

Реализиране на старт-стоп

SCL (structured control language) – програмен език от високо ниво, базиран на PASCAL

“On” := (“Start” OR “On”) AND NOT “Stop”;