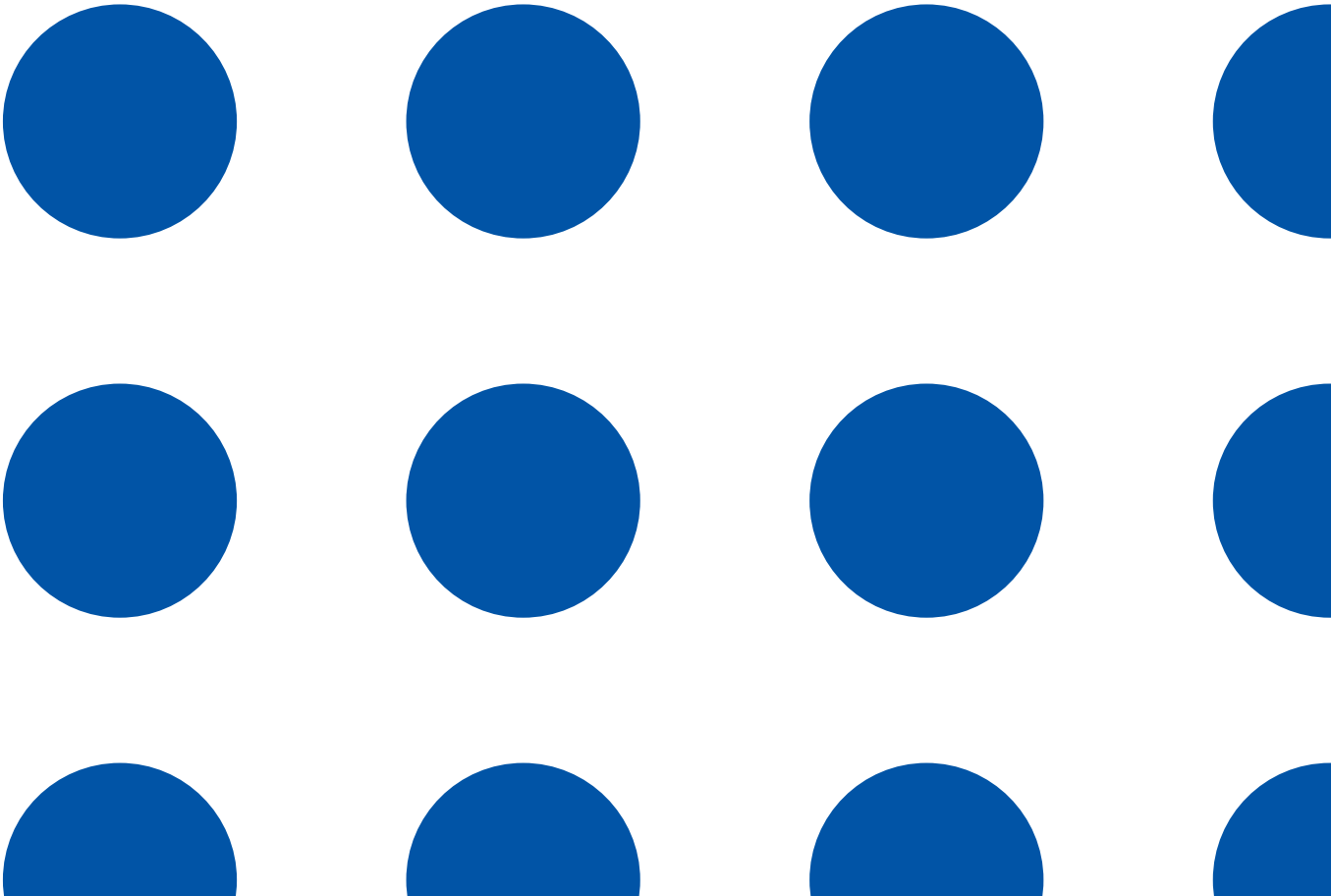
STU FEI
SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

SPEKTRUM STU

EDÍCIA SKRÍPT

LADISLAV KÖRÖSI, LEO MRAFKO, EVA MIKLOVIČOVÁ

PROGRAMOVATEĽNÉ LOGICKÉ AUTOMATY



LADISLAV KÖRÖSI, LEO MRAFKO, EVA MIKLOVIČOVÁ

PROGRAMOVATEĽNÉ LOGICKÉ AUTOMATY

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
2025

Recenzenti: Ing. Marián Filka
Ing. Marián Pavlík

Schválilo Vedenie fakulty elektrotechniky a informatiky STU v Bratislave.

ISBN 978-80-227-5509-2

Obsah

Zoznam skratiek

Kapitola 1

Úvod

Skriptá Programovateľné logické automaty (PLC) sú určené predovšetkým pre študentov 3. ročníka bakalárskeho štúdia študijného programu Robotika a kybernetika na FEI STU v Bratislave, ale aj pre širokú verejnosť. Skriptá sa zameriavajú na základné princípy a programovanie PLC, s ktorými sa študenti stretnú na predmete Riadiace systémy.

Elektronické skriptá sú členené do 4 kapitol. V úvodnej kapitole je zhrnuté teoretické minimum. V druhej kapitole je uvedená syntax príkazov v jazyku rebríkových schém, ktoré študenti využijú na praktických cvičeniach. Kapitola 3 sa zameriava na prezentáciu príkladov na lepšie pochopenie syntaxe. Posledná kapitola, ktorá je zároveň prílohou, obsahuje virtuálne procesy aj so zadaniami na precvičenie úloh rôzneho typu a rôznej náročnosti.

Veríme, že skriptá budú vhodnou pomôckou pre každého, kto sa rozhodne pracovať s programovateľnými logickými automatmi.

1.1 Riadiace systémy

Riadiacim systémom je všeobecne súbor elektronických prístrojov a zariadení, ktoré sa používajú na zabezpečenie stability, presnosti a plynulých dynamických prechodov procesu alebo výrobnej činnosti. Riadiaci systém môže mať rozmanitú podobu a líši sa implementáciou podľa toho, či ide napr. o elektráreň, o jednoúčelový stroj, či iný objekt [19].

Z historického pohľadu môžeme rozdeliť riadiace systémy na programovateľné logické automaty (PLC), distribuované riadiace systémy (DCS) a supervízorové riadenie a zber dát (SCADA) [15, 20, 18, 24].

Spočiatku sa programovateľný logický automat chápal ako riadiaci systém, ktorý je pripojený k digitálnym vstupno-výstupným jednotkám na riadenie procesov samostatnej montáže [1, 10]. V tom čase PLC dominovali v riadení diskrétnych procesov. Dôvodom ich vzniku bola snaha efektívne a lacno nahradiť logiku realizovanú pomocou relé. PLC sa používajú aj v súčasnosti na riadenie diskrétnych udalostných systémov (strojov). Charakteristickým znakom PLC je ich relatívne jednoduchý operačný systém navrhnutý tak, aby vykonával minimálne také úlohy, ako je načítanie vstupov, realizácia programov a aktualizácia výstupov. Táto jednoduchá štruktúra operačného systému znamená rýchle vykonávanie programov. Časy spracovania sú rýchle, lebo sú prirodzene bližšie k zariadeniam, ktoré ovládajú motory, čerpadlá, spínače atď. Z pohľadu spracovania meraných údajov dokážu zobrazovať aktuálne informácie na operátorských obrazovkách (HMI, PC, a pod.) [6, 21].

Druhou skupinou sú distribuované riadiace systémy, ktoré vznikli v 70. rokoch minulého storočia s cieľom riadenia spojitých procesov [1]. S rozvojom elektroniky a náhradou starých (veľkých) počítačov minipočítačmi bolo ekonomicky efektívne zabezpečiť redundanciu nasadením dvoch počítačov. Prvým distribuovaným riadiacim systémom bol TDC 2000 spoločnosti Honeywell. Charakteristickou vlastnosťou systému bola spoľahlivosť založená na redundancii procesorov, ako aj redundancii v komunikácii a v operátorskom rozhraní. Tieto riadiace systémy sú navrhnuté na riadenie spojitých procesov v aplikáciách napr. pre ropné, plynárenské a chemické závody. Tieto procesy sú spravidla geograficky rozsiahle, pozostávajú z viacerých menších systémov (procesov), ktoré sú riadené stovkami riadiacich slučiek. Riadenie takýchto procesov si vyžaduje spoľahlivosť a nepretržitú prevádzku. Z pohľadu spracovania snímaných veličín dokážu okrem zobrazenia aktuálnych informácií aj predikovať ich vývoj.

Napriek rozdielom medzi PLC a DCS došlo počas uplynulých desaťročí k ich priblíženiu. Pre moderné výkonné PLC s ich inštrukčnou sadou sa zaviedol pojem PAC (Programmable Automation Controller – programovateľný automatizačný regulátor), ktorý má vlastnosti PLC a DCS. PLC v porovnaní s DCS sú rýchlejšie a dokážu zvládnuť zložitejšie riadenie. Stali sa tiež spoľahlivejšími, vďaka čomu sa viac podobajú DCS. DCS sa stali postupom času tiež rýchlejšími a prispôsobivejšími ako PLC. V určitých prípadoch sa používajú namiesto PLC. Náklady na DCS sú stále vyššie ako na typické PLC.

SCADA systém možno zadefinovať ako systém, ktorý dozerá na geograficky rozložené procesy a riadi ich. Niekedy sa označuje aj ako tzv. Telecontrol system, ktorého názov pochádza z kombinácie telemetrie a zberu údajov. Veľkosť riadených procesov je od niekoľko tisíc vstupov-výstupov až po takmer milión vstupov-výstupov. Z hardvérového pohľadu sa SCADA skladá zo supervízneho systému, decentrálnych riadiacich systémov a klientskych staníc. Supervízny systém je počítačový systém, ktorý zbiera údaje a posiela riadiace príkazy. Súčasťou SCADA systému sú PLC a vzdialené koncové jednotky (RTU). Na rozdiel od PLC má RTU telemetrický hardvér na posielanie údajov do supervízneho systému s lokálnym autonómnym riadením. Podporujú štandardné telemetrické protokoly (napr. DNP3) pre pridelenie časových značiek, synchronizáciu času a pod. Klientske stanice sa pripájajú k databázam supervízneho systému na zobrazenie aktuálnych a historických údajov, alarmov atď. [1, 17].

1.2 Programovateľné logické automaty

Programovateľný logický automat (anglicky Programmable Logic Controller (PLC) nemecky Speicherprogrammierbare Steuerung (SPS)) je prístroj, ktorý slúži na riešenie komplexných úloh riadenia v automatizácii. Ide o používateľom programovateľný číslicový počítač, ktorý má oproti bežným počítačom niektoré špecifické vlastnosti [15, 20, 22].

1.2.1 História

Koncom minulého storočia sa americký výrobca automobilov General Motors zaujímal o použitie počítačov, ktoré by nahradili reléové systémy používané pri riadení automatizovanej

výroby automobilov. Vtedajšie riadiace systémy sa vyznačovali mnohými nevýhodami – najmä finančnou náročnosťou, množstvom kabeláže, relé a časovačov zvyšujúcich poruchovosť systému, a tým aj náročnosť údržby a komplikácie pri zmenách riadiaceho algoritmu. Automobilový priemysel potreboval flexibilnejšie riešenie, aby bolo možné realizovať každoročnú zmenu výrobného programu efektívnejším spôsobom. Dve nezávislé spoločnosti, Bedford Associates (neskôr Modicon, dnes Schneider Electric) a Allen Bradley reagovali na špecifikáciu spoločnosti General Motors. Každý z nich vyrobil počítačový systém, ktorý sa len málo podobal komerčným minipočítačom tej doby. Samotný počítač, nazývaný centrálny procesor, bol navrhnutý tak, aby pracoval v priemyselnom prostredí, a bol spojený s vonkajším svetom prostredníctvom rámov, do ktorých sa dali zapojiť vstupné alebo výstupné moduly. V týchto prvých počítačoch boli v podstate len digitálne vstupné a výstupné moduly so 16 kanálmi [21, 15, 7].

Najradikálnejšou myšlienkou však bol programovací jazyk na báze reléových schém. Vstupy reprezentujúce koncové spínače, tlačidlá, prepínače boli realizované inštrukciami „spínací kontakt“ a „rozpínací kontakt“ (angl. contacts). Výstupy reprezentujúce žiarovky, relé, štartér motora tvorili skupinu inštrukcií „cievky“ (angl. coils).

Program sa zadával cez programovací terminál s klávesmi zobrazujúcimi symboly (spínacie/rozpínacie kontakty, cievky, časovače, počítadlá, paralelné vetvy atď.), ktoré boli zrozumiteľné pre elektrikárov. Vzniknutý programovací jazyk sa dodnes používa v riadiacich systémoch pod názvom rebríková schéma (ladder (logic) diagram) [21].

Rýchly pokrok nielen v informatike, ale aj v oblasti automatizácie priniesol veľmi skoro potrebu implementácie analógového riadenia do PLC. Novovyvinuté moderné riadiace systémy, založené na PLC sa začali nazývať PAC.

1.2.2 Cyklus PLC

K riadiacim systémom sa technologické prvky (snímače a akčné členy) pripájajú pomocou vstupov a výstupov. Tie môžu byť digitálne alebo analógové. Snímače pripojené k digitálnym alebo analógovým vstupným modulom predstavujú informáciu, resp. spätnú väzbu z riadeného procesu, kým akčné členy pripojené k digitálnym a analógovým výstupným modulom ovládajú proces. Procesor (CPU) PLC vykonáva tzv. programový cyklus, ktorý v zjednodušenej podobe pozostáva z troch krokov [3, 5, 4, 2, 6, 7, 8, 21]:

- Krok 1: Načítanie vstupov – V prvom kroku CPU pomocou zbernice komunikuje stav vstupov zo vstupných modulov. Tieto hodnoty sa lokálne aktualizujú príslušným modulom a odosielajú do procesora. Stav vstupov je uložený do pamäte vstupov (process image of inputs).

- Krok 2: Vykonanie programu – Vykonajú sa časti programu vytvorené v rôznych programovacích jazykoch. Počas vykonávania programu je obraz vstupov nemenný, t. j. aj keď sa v priebehu vykonania programu zmení stav vstupu, v aktuálnom cykle to program neovplyvní. Program môže meniť hodnoty v rôznych pamäťových oblastiach ako sú výstupy a vnútorné pamäťové premenné. Tieto premenné môžu byť viacnásobne prepísané, ale na zabezpečenie efektívnej diagnostiky sa odporúča premenné zapisovať len raz (napr. v prípade binárnych premenných dvakrát s využitím inštrukcií set a reset).

- Krok 3: Aktualizácia výstupov – V treťom kroku procesor zapíše pamäťovú oblasť výstupov, tzv. obraz výstupov (process image of outputs) na výstupné moduly.

Procesor PLC má všeobecne dva režimy, RUN a STOP. V režime RUN sa vykonáva program, kým v režime STOP nie. Implementácia režimov závisí od výrobcov ako aj rodiny PLC. Najčastejšou implementáciou je načítanie vstupov bez ohľadu na režim PLC. Výhodou je možnosť neustáleho monitorovania vstupov a diagnostika. Ak je CPU v režime RUN, vykonajú sa kroky 2 a 3. V opačnom prípade sa program nevykonáva a výstupy sú v preddefinovanom (vypnutom) stave.

Okrem základných krokov CPU zabezpečuje komunikáciu (decentrálne zariadenia, HMI, programovací softvér, ...) a diagnostiku (komunikácie, hardvéru, softvéru, ...).

V procesore PLC sa konfigurujú tzv. úlohy (angl. tasks), ktoré delíme na cyklické, periodické a udalostné. Tieto úlohy realizujú vyššie opísané tri kroky programového cyklu [3, 5, 4, 2, 7].

- Cyklická úloha – Kroky 1 až 3 sa vykonajú v uvedenom poradí a po dokončení cyklu sa hneď pokračuje 1. krokom. Úloha má najnižšiu prioritu, t. j. v prípade viac úlohového projektu sa preruší úlohou s vyššou prioritou, preto nie je vhodná pre spojité riadenie. Používa sa na riadenie diskrétnych udalostných systémov.

- Periodická úloha – Úloha sa začne vykonávať v presne definovanej perióde. Poradie krokov 1 - 2 - 3, resp. 3 - 1 - 2 závisí od konkrétnej implementácie. Z pohľadu spojitého riadenia je výhodnejšia implementácia 3 - 1 - 2, lebo aktualizácia vstupov a výstupov má pevnú periódu vzorkovania. Nakoľko úlohy majú svoju prioritu, môže byť periodická úloha prerušená periodickou úlohou s vyššou prioritou alebo udalostnou úlohou. Periodická úloha je vhodná na spojité riadenie. Periodické úlohy majú vždy vyššie priority ako cyklické.

- Udalostná úloha – Má najvyššiu prioritu, preto preruší vykonanie cyklickej aj periodickej úlohy. Udalosť, ktorá vyvoláva spustenie udalostnej úlohy, môže byť hardvérové alebo časovo aktivované prerušenie.

K vstupom a výstupom v niektorých riadiacich systémoch možno pristupovať aj počas vykonávania programu, napríklad zistiť aktuálny stav vstupu na svorkách, alebo zapísať konkrétny výstup na výstupný modul. Nevýhodou tohto riešenia je predĺženie programového cyklu z dôvodu nadbytočnej komunikácie zbernicou, ktorá je najčastejšie sériová.

Norma IEC 61131 zavedená v roku 1993 definuje softvérovú a hardvérovú štruktúru PLC. Norma sa skladá zo siedmich častí

1. Základné informácie – Definície základných pojmov a koncepcií.

2. Technické požiadavky a testy – Elektronická a mechanická konštrukcia zariadení a overovacie testy.

3. Programovacie jazyky – Štruktúra programov pre PLC, programovacie jazyky a opis vykonávania programov.

4. Používateľská príručka – Pomoc pri výbere, inštalácii a údržbe PLC.

5. Špecifikácia komunikačných služieb – Softvérové možnosti komunikácie s inými zariadeniami používajúcimi MAP protokol.

6. Fuzzy logika.

7. Technická správa.

Norma definuje najmenšie jednotky používateľského programu (Program Organisation Unit - POU). Existujú tri druhy POU [3, 5, 4, 2, 7, 21, 22]:

- Funkcia (FC) – je ucelená časť programu, ktorú môže program opakovane volať. Na rovnaké vstupy dáva ako odpoveď rovnaké výstupy, t. j. je statická (neobsahuje vnútornú pamäť). Dáta, s ktorými FC pracuje, sa ukladajú do lokálneho dátového zásobníka a po ukončení FC sa zmažú. Pri každom volaní funkcie je nutné priradiť formálnym parametrom (ak existujú) skutočné parametre. Formálne parametre, ktoré sa používajú priamo vo funkcii, sú iba ukazovateľmi na skutočné parametre logického bloku, z ktorého bola funkcia volaná. Programový kód funkcie sa v CPU nachádza len raz [13, 11, 14]. Rozlišujeme dve skupiny funkcií:
  - Štandardné funkcie – dostupné zo štandardných knižníc programovacieho softvéru (napr. ADD – sčítanie, SQRT – odmocnina, MAX – maximálna hodnota, atď.).
  - Používateľom definované funkcie so vstupno-výstupným rozhraním alebo bez neho.

- Funkčný blok (FB, DFB) – na rozdiel od FC obsahuje vnútornú pamäť, ktorá sa nazýva inštančný dátový blok (DB). V tomto DB sú uložené všetky premenné konkrétneho FB, ktoré ostávajú v DB aj po ukončení vykonávania FB. Výstup funkčného bloku závisí od vstupov, ako aj od interných premenných, preto na rovnaké vstupy nemusí dávať rovnaké výstupy. Programový kód funkčného bloku sa v CPU nachádza len raz [16, 12]. Existujú dve skupiny funkčných blokov:
  - Štandardné funkčné bloky – dostupné zo štandardných knižníc programovacieho softvéru - napr. časovače (TON – oneskorené zapnutie, TOF – oneskorené vypnutie, ...), počítadlá, inštrukcie PID – proporcionálny, integračný a derivačný regulátor atď.
  - Používateľom definované funkčné bloky – označenie funkčných blokov sa v rôznych riadiacich systémoch líši. V riadiacich systémoch Siemens má názov FB, ale u Schneider Electric je zaužívaný názov DFB (Derived Function Block - odvodený funkčný blok) [8, 23].

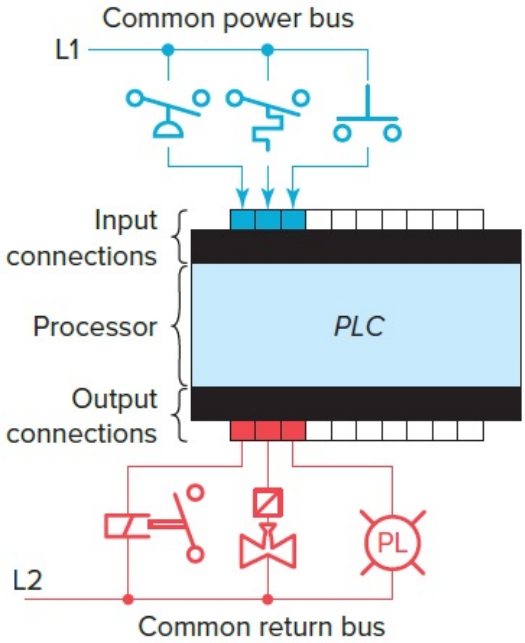
- Program (PRG) – je najvyššia úroveň POU, ktorá pristupuje k vstupom a výstupom a z ktorej sú volané funkcie a funkčné bloky. V riadiacich systémoch Siemens je program totožný s tzv. organizačným blokom (OB), ktorý je súčasne aj úlohou.

1.2.3 Vonkajšia štruktúra PLC

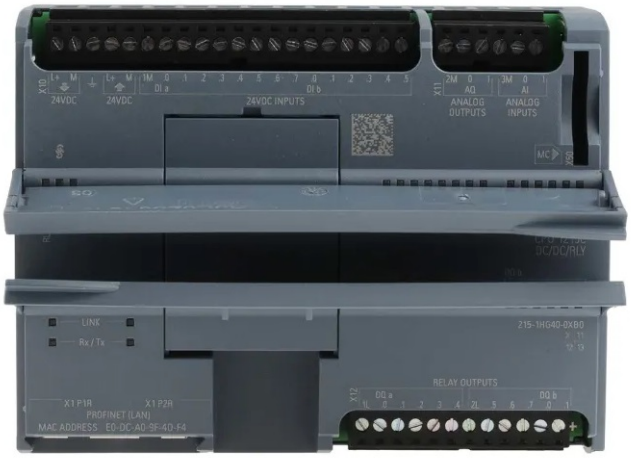
Z pohľadu vonkajšej štruktúry môžeme PLC rozdeliť na kompaktné a modulárne riadiace systémy [22]. Ich spoločným prvkom je procesor. Procesor použitý v PLC je štandardným

procesorom, ktorý sa nachádza aj v iných mikroprocesorom riadených systémoch (napr. v Siemens CPU S7-317 je použitý 32 bitový Infineon TriCore). Voľba procesora závisí od požiadaviek (rýchlosť vykonania aritmetických a logických operácií, presuny pamäťových blokov, používateľské rozhranie, komunikačné možnosti, bezpečnostné funkcie, redundancia, ...).

Kompaktné riadiace systémy sú zvyčajne v kompaktnom vyhotovení, vyznačujú sa nižším výkonom a menším počtom pripojiteľných vstupov a výstupov. Preto sú aj finančne menej náročné. Pri kompaktnom prevedení CPU obsahuje integrované vstupy a výstupy priamo na CPU. Najčastejšie má 1 až 2 analógové vstupy a výstupy, kým digitálnych máva okolo 10 až 14. Príklad integrovaných vstupov a výstupov kompaktného PLC s elektrickým zapojením vstupov a výstupov je na Obr. 1.1. V hornej časti CPU Input connections reprezentujú vstupné svorky. V spodnej časti CPU Output connections reprezentujú výstupné svorky. Na Obr. 1.2 je fotografia PLC Siemens Simatic S7-1200.



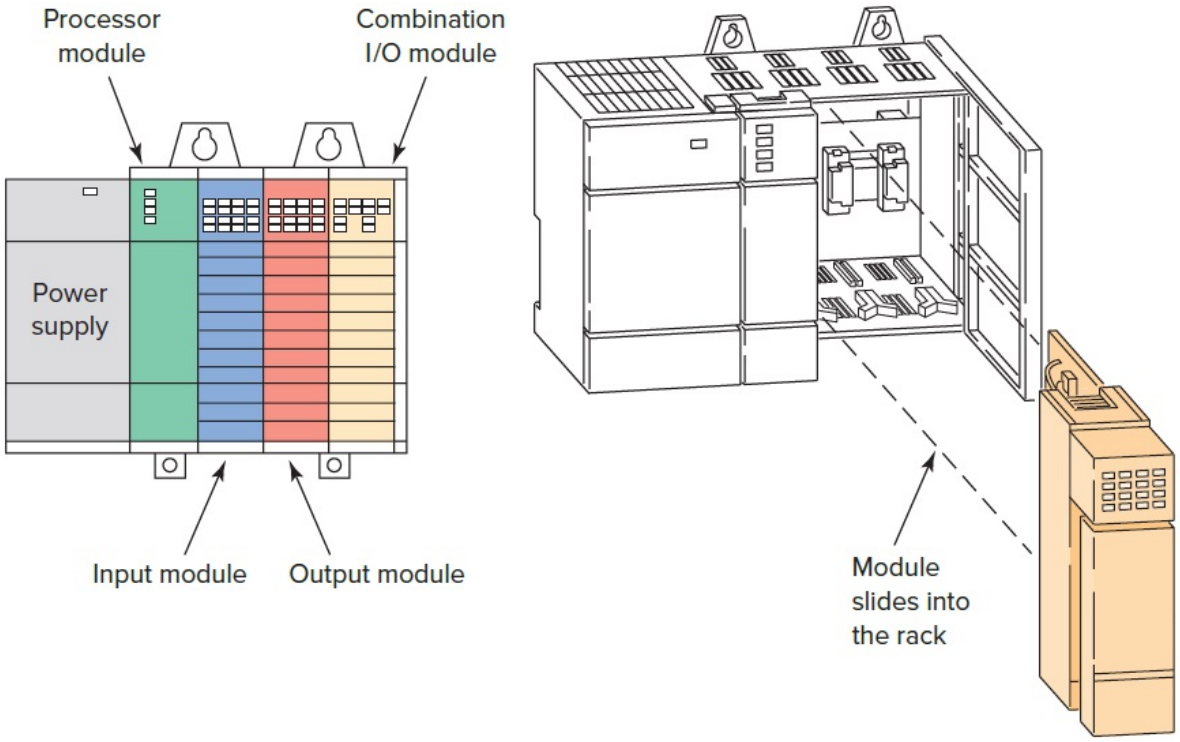
Obr. 1.1. Príklad integrovaných vstupov a výstupov kompaktného PLC s elektrickým zapojením vstupov a výstupov [22]



Obr. 1.2. Fotografia S7-1200 s viditeľnými svorkovnicami

Modulárne riadiace systémy sú adaptívne riadiace systémy pozostávajúce z modulov, voľbou ktorých vznikne riadiaci systém vhodný na riadenie konkrétneho procesu. Pozostáva zo zdroja, procesora, ktorý nemá integrované vstupy a výstupy a z ďalších modulov, ako digitálne a analógové vstupné a výstupné moduly, komunikačné moduly, rýchle čítače, budiče krokových motorov, vážiace moduly a pod. Historicky sa moduly pripájali do tzv. rámu alebo lišty (angl. rack). V súčasnosti sa upúšťa od rámov s pevným počtom modulov. Susedné moduly sú prepojené pomocou kontaktov integrovaných v module, t. j. zbernica je vedená priamo cez modul. V prípade odpojenia modulu dochádza k znefunkčneniu zvyšných modulov napravo od aktuálneho modulu, kým pri pevnom ráme zvyšné moduly sú naďalej funkčné. Zbernica zabezpečuje privedenie napájania pre pripojené moduly ako aj komunikáciu medzi modulmi. Čím je komunikácia po zbernici rýchlejšia, tým sa rýchlejšie načítajú vstupy a zapíšu výstupy, a teda cyklus PLC môže byť kratší. V súčasnosti existujú zbernice na báze Ethernetu (tzv. ePAC - Modicon M580), ktoré majú vyššiu rýchlosť a priepustnosť.

Na Obr. 1.3 vľavo je zobrazený modulárny systém pozostávajúci z CPU, vstupného, výstupného a komunikačného modulu. V pravej časti je 3D vyobrazenie modulárneho systému. Tu je naznačený príklad vsunutia modulu do rámu, ktorý má v zadnej časti konektor zbernice. Na Obr. 1.4 sa nachádza modulárny systém PLC Modicon M340 od spoločnosti Schneider Electric.



Obr. 1.3. Príklad modulárneho PLC [22]

Obr. 1.4. Fotografia modulárneho PLC Modicon M340

Analógové moduly rozdeľujeme podľa typu merania do 2 kategórií:

- Napäťové a prúdové moduly – označované ako U/I s pripojením zariadení pomocou unifikovaných napäťových alebo prúdových signálov (0 - 5V, 0 - 10V, +/- 5V, 0 - 20mA, 4 - 20mA, +/- 20mA atď.). Konfigurácia sa vykonáva v programovacom nástroji v časti hardvérovej konfigurácie.
- Odporové moduly – označované ako RTD/TC s pripojením termočlánkov (TC) rôznych typov a odporových teplotných detektorov (RTD - ako napr. PT100, PT500, PT1000, Ni100 atď.).

Samozrejme existujú aj moduly podporujúce všetky typy meraní.

Analógové vstupné moduly transformujú signál pripojený k svorkám kanálu na rozsah modulu (napr. 0 - 10 000, 0 - 27 648). Rozsah závisí od výrobcu a rodiny PLC. Rýchlosť spracovania a presnosť závisí od konkrétnej voľby modulu. Všeobecne má projektant možnosť voľby z dvoch úrovní presnosti (počtu bitov prevodníka) a dvoch úrovní rýchlosti spracovania signálov. Riadiace systémy Siemens ponúkajú vstupné analógové moduly s označením základné (BA - Basic), štandardné (ST - Standard), s vyššou funkcionalitou (HF - High Feature) a s vysokou rýchlosťou (HS - High Speed) [3, 5, 4, 2, 6, 7].

1.2.4 Druhy PLC

Riadiace systémy PLC sú navrhnuté do priemyselného prostredia, sú odolné voči zvýšenej teplote na pracovisku, vibráciám, prašnosti a pod. Okrem uvedených prevádzkových podmienok, ktoré majú vplyv na výber riadiacich systémov, majú PLC vlastnosti, ktoré nezávisia od prostredia. Podľa účelu nasadenia delíme PLC na klasické, bezpečnostné a odolné voči poruchám [22].

- Klasické PLC – Sú to klasické kompaktné alebo modulárne riadiace systémy s upravenými vlastnosťami pre priemysel vykonávajúce programový cyklus v zmysle predchádzajúcej kapitoly.

- Bezpečnostné PLC (angl. failsafe alebo safety PLC) – Pri návrhu riadenia procesu je potrebné brať do úvahy bezpečnosť pracoviska (osôb a materiálu). Na základe analýzy súčasného stavu sa musia určiť riziká. Ak sú riziká vysoké, je potrebné prijať opatrenia na zníženie rizík. Okrem pasívnych ochrán (oplotenie, múr, ...) a zavedení aktívnych ochranných prvkov (snímačov ako laserový skener, optická bariéra a pod.) sa nasadzujú tzv. bezpečnostné riadiace systémy. Zložitosť bezpečnostných PLC sa líši od jednoduchého bezpečnostného relé až po konfigurovateľné bezpečnostné systémy. Bezpečnostné PLC v prípade výskytu nebezpečných podmienok dovedú proces do bezpečného stavu. Snímače a akčné členy sa pripájajú k vstupno-výstupným modulom dvoj- alebo viackanálovo, čo zabezpečuje vyššiu robustnosť a odolnosť spracovania signálov voči poruchám samotného riadiaceho systému. Modul procesora pozostáva z viacerých CPU, v ktorých je program skompilovaný odlišným spôsobom. Výsledok výpočtov z viacerých CPU sa vzájomne porovnáva s cieľom diagnostiky CPU a RAM. V praxi sa takéto bezpečnostné systémy (moduly) označujú žltou alebo červenou farbou. Bezpečnostné PLC dokáže vykonávať klasický aj bezpečnostný program zároveň. Dokáže komunikovať pomocou klasického priemyselného ethernetu (prípadne zbernice) alebo využívať bezpečnostnú nadstavbu (napr. PROFIsafe). Pre diskrétnu automatizáciu pomocou PLC platia iné bezpečnostné normy ako pre DCS systémy.

- PLC odolné voči poruchám (angl. fault tolerant PLC) – Porucha riadiaceho systému alebo akýchkoľvek jeho komponentov môže viesť k nákladným prestojom vo výrobe. Je potrebné vziať do úvahy aj náklady na opätovné spustenie kontinuálneho procesu a skutočné výrobné straty vyplývajúce z poruchy. Použitím komponentov odolných voči poruchám sa môže minimalizovať riziko zlyhania výroby. PLC odolné voči poruchám sú navrhnuté tak, aby pokračovali v prevádzke aj v prípade poruchy jedného alebo viacerých komponentov. Nato sa využíva princíp redundancie. PLC systém má

redundantné moduly (zdroje, CPU, vstupno-výstupné moduly, komunikáciu a pod.) ku ktorým sa môžu pripájať redundantné snímače a akčné členy. Nasadením riadiacich systémov odolných voči poruchám sa zvyšuje celková dostupnosť systému. Do skupiny PLC odolných voči poruchám sa radia tzv. standby riadiace systémy. Hot standby riadiaci systém pozostáva z dvoch identických PLC zostáv. Primárny systém riadi proces a sekundárny (standby) „čaká“ na poruchu primárneho systému. CPU sú synchronizované a v prípade výskytu poruchy záložný systém automaticky preberá riadenie.

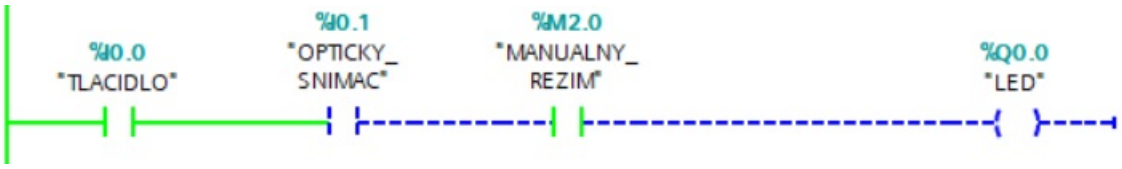
V určitých prípadoch je nutné zaručiť bezpečnosť, ale aj vysokú dostupnosť. Sú aj riadiace systémy, ktoré kombinujú vyššie opísané vlastnosti riadiacich systémov.

1.2.5 Programovacie jazyky PLC

Programovacie jazyky PLC používajú inžinieri, technici a údržbári. Vychádzajú preto skôr z techník používaných v priemysle než z techník používaných v počítačovom programovaní. V tejto kapitole uvedieme jazyky, ktorými je možné programovať PLC od rôznych výrobcov podľa normy IEC 61131 [5, 3, 5, 4, 2, 10, 23].

Jazyk rebríkových schém (Ladder Diagram – LD, LAD)

Grafický jazyk LD vznikol v USA, kde prvé PLC pre automobilový priemysel nahradili reléovú logiku. Základom jazyka boli symboly použité na výkresoch: -] [- pre spínací (NO) kontakt, -]/[- rozpínací (NC) kontakt a -( )- pre zápis digitálneho výstupu. Nad uvedené symboly sa zadávali adresy vstupov, výstupov a vnútorných premenných ako argumenty inštrukcií. Postupom času bola sada inštrukcií rozšírená o nové bitové inštrukcie, časovače, počítadlá, matematické inštrukcie a pod. Výhodou jazyka je dobrá diagnostika. Nevýhodou je náročnejšia implementácia komplexných matematických vzťahov a opakujúcich sa častí kódu (napr. inštrukcie for, while, repeat). Program vytvorený v jazyku LD sa vykonáva zhora nadol a v rámci priečky rebríka zľava doprava a zhora nadol. Rebrík (názov sa líši podľa výrobcu: ladder, network, ...) predstavuje skupinu navzájom prepojených inštrukcií, ktoré sú pripojené k ľavej a pravej časti editora. Graficky takto vytvárajú priečky rebríka. Príklad programu v jazyku LD je na Obr. 1.5. [3, 5, 4, 2, 7, 9]

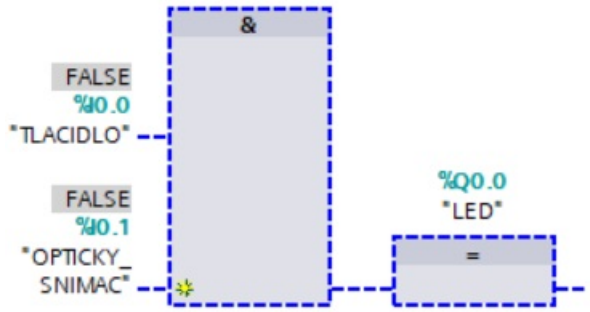


Obr. 1.5. Príklad programu v jazyku LD v TIA Portal

Diagram funkčných blokov (Function Block Diagram – FBD)

Grafický jazyk FBD je v Európe veľmi rozšírený. Skladá sa z grafických blokov, ktoré sú navzájom pospájané podobne ako logické obvody (AND, OR, XOR, ...). Symboly inštrukcií sú podobné niektorým symbolom v jazyku LD. Na rozhraní symbolov je vstup EN,

ktorý povoľuje vykonanie inštrukcie, a výstup ENO, ktorý povoľuje výstup. Takto zreťazené inštrukcie môžu tvoriť jeden celok. Poradie vykonania inštrukcií závisí od umiestnenia na pracovnej ploche. Inštrukcie sa vykonávajú v rovnakom poradí, ako keby sme čítali slovenský text zhora nadol zľava doprava. Výhodou jazyka je dobrá diagnostika a kompaktné inštrukcie v porovnaní s jazykom LD. Kým v jazyku LD, AND viacerých podmienok môže presahovať šírku obrazovky, tak v jazyku FBD zaberie len časť obrazovky. Nevýhody jazyka sú zhodné s jazykom LD. Príklad programu v jazyku FBD je na Obr. 1.6. [3, 5, 4, 2, 9]



Obr. 1.6. Príklad programu v jazyku FBD v TIA Portal

Štruktúrovaný text (Structured Text – ST)

Ide o vysokoúrovňový textový jazyk podobný jazykom Basic alebo Pascal, špeciálne vyvinutý pre aplikácie priemyselného riadenia. Umožňuje použitie zložitých matematických príkazov a príkazov na podmienené a opakované vykonávanie častí programu (REPEAT-UNTIL, DO-WHILE, FOR, IF-THEN-ELSE, CASE a pod.). Nevýhodou jazyka je horšia diagnostika, a to aj napriek novým animáciám v jednotlivých softvéroch, ako je zobrazenie binárnych stavov farbami červená (stav FALSE) a zelená (stav TRUE). Výhodou jazyka je pestrosť možností implementácie algoritmov a možnosť rýchleho kopírovania a modifikácie častí programu. Príklady ekvivalentných programov v jazyku ST sú na Obr. 1.7 [3, 5, 4, 2, 9].

```
1  (* PROG 1 *)
2  IF "A" AND NOT ("B") AND "C" AND "D" THEN
3      "E" := TRUE;
4  ELSE
5      "E" := FALSE;
6  END_IF;
7
8  (* PROG 2 *)
9  "E":="A" AND NOT ("B") AND "C" AND "D";
```

Obr. 1.7. Ekvivalentné príklady programov v jazyku ST v TIA Portal

Zoznam inštrukcií (Instruction List – IL)

Textový programovací jazyk, ktorý sa podobá na assembler. V každom riadku je umiestnená jedna inštrukcia pozostávajúca z názvu inštrukcie (prípadne názvu inštancie funkč-

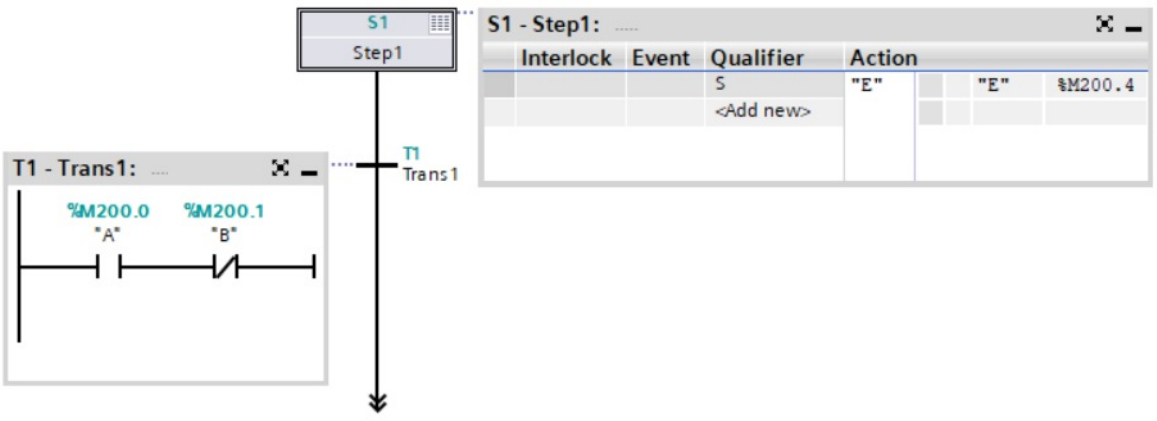
ného bloku) a jedného alebo viacerých operandov. Program sa vykonáva líniovo zhora nadol. Ak vyžadujeme vykonanie určitých inštrukcií v závislosti od splnenia alebo nesplnenia podmienok, je potrebné naprogramovať skoky na návestia a preskočiť tak časti kódu. Jazyk IL sa používal tam, kde sa mal program vykonávať vysokou rýchlosťou (ide o strojový jazyk) alebo sa vyžadoval prístup k pamäťovým oblastiam bez obmedzení aké boli stanovené v iných programovacích jazykoch. Nevýhodou tohto jazyka je náročná diagnostika najmä pri tvorbe zložitých podmienok a iný náhľad na programovanie ako v grafických jazykoch. Výhodou je rýchle vykonávanie kódu s cieľom zabezpečiť čo najkratší programový cyklus. Príklad programu v jazyku IL je na Obr. 1.8. [3, 5, 4, 2, 9]

```
1     A     "A"
2     AN    "B"
3     A     "C"
4     A     "D"
5     =     "E"
```

Obr. 1.8. Príklad programu v jazyku IL v Unity Pro

Sekvenčný diagram (Sequential Function Chart – SFC)

Ide o grafický programovací jazyk definovaný podľa normy IEC 61131 ako príprava funkčných schém pre riadiace systémy. Je založený na jazyku GRAFCET (z franc. Graphe Fonctionnel de Commande Étapes/Transitions – funkčný graf riadenia krokov a prechodov), ktorý vychádza z Petriho sietí. Grafické objekty sa podobajú na vývojový diagram pozostávajúci z krokov a prechodov. Krok reprezentuje stav v rámci sekvencie, počas ktorého sa vykonáva určitá časť programu prislúchajúca tomuto stavu. Po splnení prechodovej podmienky nasledujúcej za aktívnym krokom sa aktivita (riadenie) presúva na ďalší krok, kde sa začne vykonávať program definovaný pre nový stav. Aj podľa názvu možno usúdiť, že je vhodný na sekvenčné riadenie procesov, pri ktorých sa program procesu dá rozdeliť na niekoľko krokov. Výhodou jazyka je veľmi dobrá diagnostika s možnosťou vnorenia programov v iných programovacích jazykoch. Pri programovaní v SFC je potrebné iné programátorské myslenie ako v iných programovacích jazykoch. Jazyk je vhodný na riadenie diskrétnych sekvenčných udalostných systémov a dávkových procesov. Príklad programu v jazyku SFC je na Obr. 1.9 [3, 5, 4, 2, 9].



Obr. 1.9. Príklad programu v jazyku SFC v TIA Portal

Kapitola 2

Premenné

Táto kapitola je venovaná premenným, ktoré sú v oblasti riadiacich systémov nazývané tiež tagmi. V skriptách budeme naďalej používať pojem premenná. V podkapitolách budú opísané pamäťové oblasti riadiacich systémov Siemens, ich rozdelenia, dátové typy a pod.

2.1 Tag

Premenná (tag) je pamäťové miesto v riadiacom systéme, ktoré reprezentuje konkrétnu hodnotu v jednom čase. Operácie s premennou môžeme rozdeliť na:

- Čítanie - hodnota sa nemení (napr. inštrukcie normally open contact, normally closed contact, porovnávacie bloky, vstupy napr. inštrukcií ADD, SUB, ..., t. j. vstupné premenné funkcií a funkčných blokov)
- Zápis - hodnota sa mení (napr. inštrukcie set, reset, výstupy matematických inštrukcií ADD, SUB, ..., t. j. všeobecne výstupné premenné funkcií a funkčných blokov)

Tagy sú jednoznačne zadefinované:

- Dátovým typom a fyzickou (tzv. absolútnou) adresou. Napríklad:
  - %M0.0, BOOL
  - %MW0, INT
  - %I0.0, BOOL
- Dátovým typom a symbolickým názvom. Napríklad:
  - Pocet, INT
  - Priemer, REAL
  - Koniec, BOOL
  - Teplota, REAL
  - Cas, TIME

V praxi tag môže mať symbolický názov, dátový typ a aj fyzickú adresu. Napríklad:

- TLACIDLO, BOOL, %I0.0
- MOTOR_START, BOOL, %Q0.0
- AKCNA_VELICINA, INT, %QW100
- MERANA_TEPLOTA, INT, %IW95

2.2 Dátové typy

Prehľad vybraných elementárnych a štrukturovaných dátových typov je v tab. 2.1 až 2.9. Hodnoty niektorých dátových typov v stĺpci Rozsah začínajú písmenom (pozri Tab. 2.1). Toto písmeno je skratkou dátového typu (B - Byte, W - Word, ...). Za ním nasleduje znak # ako oddeľovač. Zápis 16# definuje hodnotu v hexadecimálnej sústave. Za ňou nasleduje konkrétna hodnota v hexadecimálnom zápise (napr. FF). Písmeno D znamená Double. Napríklad v dátovom type DWORD znamená dvojslovo (dvojnásobná dĺžka slova), teda 16-bitovú celočíselnú premennú (v hexadecimálnom tvare). Písmeno L znamená Long. Napríklad v dátovom type LWORD ide o dvojnásobnú dĺžku dvojslova, tzn. 64-bitovú celočíselnú premennú (v hexadecimálnom tvare). Písmeno S znamená Small, t. j. napr. v dátovom type SINT (Tab. 2.3) definuje polovicu veľkosti INT, teda 8-bitovú celočíselnú premennú. BCD (Binary Coded Decimal) je binárne kódované decimálne číslo (Tab. 2.5). Pri tomto kóde je každá číslica hodnoty premennej zakódovaná pomocou štyroch bitov. BCD reprezentácia znakov a číslic sa používala pri zobrazovačoch (displejoch).

Tabuľka. 2.1. Bitové sériové dátové typy

Názov	**Bity**	**Rozsah**
BOOL	1 bit	0, 1, FALSE, TRUE
BYTE	8 bitov	B#16#00 až B#16#FF
WORD	16 bitov	W#16#0000 až W#16#FFFF
DWORD	32 bitov	DW#16#0000 0000 až DW#16#FFFF FFFF
LWORD	64 bitov	LW#16#0000 0000 0000 0000 až LW#16#FFFF FFFF FFFF FFFF

2.3 Vybrané pamäťové oblasti

Táto kapitola sa zameriava na vybrané pamäťové oblasti Siemens PLC:

Tabuľka. 2.2. Celočíselné dátové typy bez znamienka (U - Unsigned)

Názov	**Bity**	**Rozsah**
USINT	8 bitov	0 až 255
UINT	16 bitov	0 až 65 535
UDINT	32 bitov	0 až 4 294 967 296
ULINT	64 bitov	0 až 18 446 744 073 709 551 615

Tabuľka. 2.3. Celočíselné dátové typy so znamienkom

Názov	**Bity**	**Rozsah**
SINT	8 bitov	–128 až +127
INT	16 bitov	–32 768 až +32 767
DINT	32 bitov	–2 147 483 648 až +2 147 483 647
LINT	64 bitov	–9 223 372 036 854 775 808 až +9 223 372 036 854 775 807

Tabuľka. 2.4. Čísla s pohyblivou rádovou čiarkou

Názov	**Bity**	**Rozsah**
REAL	32 bitov	približne $\pm 1,18 \times 10^{-38}$ až $\pm 3,40 \times 10^{38}$
LREAL	64 bitov	približne $\pm 2,23 \times 10^{-308}$ až $\pm 1,80 \times 10^{308}$

- Oblasť vstupov (%I) - hodnoty sa získavajú v každom programovom cykle zo snímačov (napr. vstupných analógových a digitálnych modulov). Táto oblasť sa nepoužíva na uloženie pomocných stavov riadeného procesu. Ide o spätné väzby z procesu.
- Oblasť výstupov (%Q) - hodnoty sa modifikujú počas programového cyklu a zapisujú na konci cyklu na výstupy (napr. výstupné analógové a digitálne moduly), aby ovládali akčné členy. Ide o akčné veličiny.

Tabuľka. 2.5. BCD čísla

Názov	**Bity**	**Rozsah**
BCD16	16 bitov	–999 až +999
BCD32	32 bitov	–9 999 999 až +9 999 999

Tabuľka. 2.6. Dĺžky trvania

Názov	**Bity**	**Rozsah**
TIME	32 bitov	T#–24d20h31m23s648ms až T#+24d20h31m23s647ms
LTIME	64 bitov	LT#–106751d23h47m16s854ms775us808ns až LT#+106751d23h47m16s854ms775us807ns

Tabuľka. 2.7. Časové body (dátum a čas dňa)

Názov	**Bity**	**Rozsah**
DATE	32 bitov	D#1990-01-01 až D#2168-12-31
TOD	64 bitov	TOD#00:00:00.000 až TOD#23:59.59.999

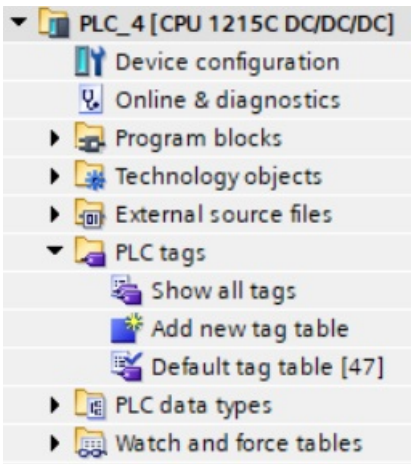
Tabuľka. 2.8. Znaky v ASCII

Názov	**Bity**	**Rozsah**
CHAR	8 bitov	’a’, ’A’, ’1’, . . .

- Vnútorná pamäťová oblasť (%M) - hodnoty čítame a modifikujeme na uchovávanie stavov procesu. Sú to napr. počty objektov, žiadané hodnoty, stavy či stroj pracuje, skončil, fázy procesu, . . . a ich vizualizácie pomocou HMI. Vstupné, výstupné a pamäťové premenné definujeme v symbolických tabuľkách, ktoré nájdeme v stromovej štruktúre projektu v časti PLC tags (2.1). Príklady premenných sú na Obr. 2.2.

Tabuľka. 2.9. Štruktúrované dátové typy

Názov	**Bity**	**Rozsah**
DATE_AND_TIME	8 bajtov	Dátum a čas (presnosť milisekundy) Príklad: DT#1990-01-01-00:00:00
STRING	2 + n bajtov	Reťazec s *n* znakmi. Príklady: 'Ahoj svet!', 'Motor zapnutý'
ARRAY	premenlivé	Kombinácia niekoľkých ekvivalentných dátových typov. Príklad: Premenná *vektor* má dátový typ ARRAY[1..10] OF INT Individuálne premenné sú vektor[1], vektor[2] , ..., vektor[10]
STRUCT	premenlivé	Kombinácia niekoľkých rôznych dátových typov. Príklad: Premenná *motor* má dátový typ STRUCT Individuálne premenné môžu byť motor.zapnuty, motor.vypnuty, motor.alarm a pod.



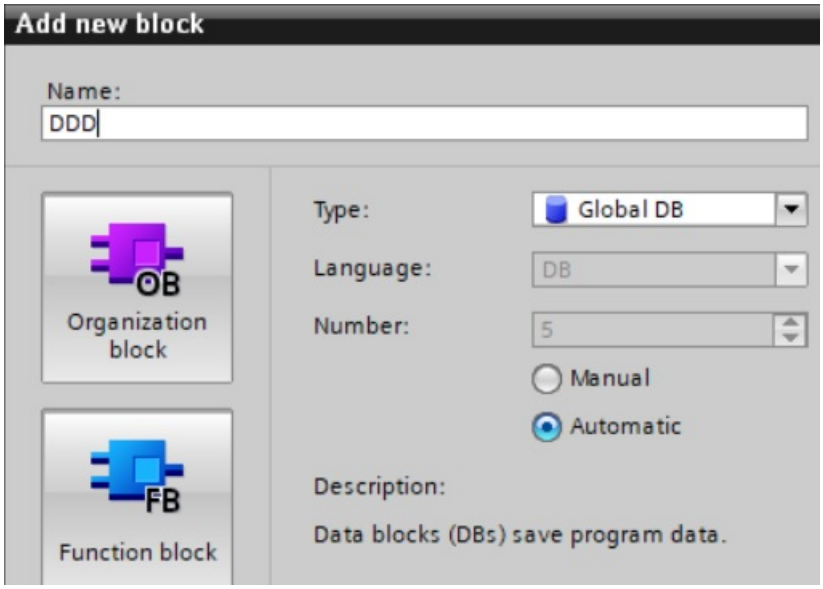
Obr. 2.1. Editor vstupných, výstupných a pamäťových premenných

Default tag table

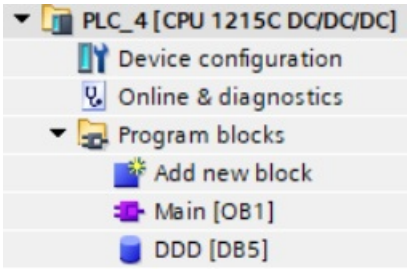
	Name	Data type	Address
1	OS1	Bool	%I0.0
2	OS2	Bool	%I0.1
3	KS1	Bool	%I0.2
4	M1_ZIADANY_POCET	Int	%MW10
5	M1_AKTUALNY_POCET	Int	%MW12
6	OS2_SB	Bool	%M1.1

Obr. 2.2. Príklady premenných v symbolickej tabuľke Default tag table

- Dátové bloky (%DB) - sú vo svojej podstate štruktúry premenných s oddelenou a dynamicky pridelovanou pamäťovou oblasťou. Na rozdiel od pamäťovej oblasti %M, ktorej miesto premennej je fixné a spoločné pre všetky dátové typy, doplnenie premennej medzi už definované v %DB nevyžaduje posun adries všetkých za ňou. Umožňujú definovanie polí (ARRAY), štruktúr (STRUCT), počiatočných hodnôt atď. Navyše umožňujú definovanie polí (ARRAY), štruktúr (STRUCT), počiatočných hodnôt, atď. Rozhranie pre vytvorenie nového dátového bloku je na Obr. 2.3. Vľavo hore pod *Name:* sa vypĺňa symbolický názov dátového bloku. V časti *Type:* sa volí globálny alebo inštančný dátový blok (viac v texte nižšie). V riadku *Number:* sa automaticky určuje číslo dátového bloku, ktoré po prepnutí z Automatic na Manual možno zmeniť na iné číslo. Číslo dátového bloku sa v stromovej štruktúre projektu zobrazuje vedľa symbolického názvu DB (Obr. 2.4). Číslo dátového bloku je pritom v rámci programu unikátne, nesmú existovať viaceré dátové bloky s rovnakým číslom.



Obr. 2.3. Rozhranie na vytvorenie nového dátového bloku



Obr. 2.4. Názov a číslo dátového bloku v stromovej štruktúre projektu

V závislosti od riadiaceho systému (Siemens, Schneider Electric, ...) existujú rôzne implementácie pamäťových oblastí. Zameriame sa na Siemens riešenie. Nižšie uvedené mapovanie %M je rovnaké aj pre oblasti %I, %Q a %DB.

- %Ma.b - definuje bit v pamäťovej oblasti M, kde a je číslo bajtu a b je číslo bitu (od 0 po 7) v bajte. Bit 0 je najnižší bit bajtu (LSB - least significant bit) a bit 7 je najvyšší bit bajtu (MSB - most significant bit).
  - Príklad %M0.0. Čítaj ako pamäťový bit 0.0.
- %MBa - definuje bajt (slabiku) v pamäťovej oblasti M, kde a je číslo bajtu.
  - Príklad %MB0. Čítaj ako pamäťový bajt 0. Bity uvedeného bajtu sú %M0.0 až %M0.7, t. j. zmenou hodnoty bitov zmeníme hodnotu bajtu, resp. opačne lebo ide o rovnakú pamäťovú oblasť.
- %MWa - definuje word (slovo) v pamäťovej oblasti M, kde a je číslo slova. Slovo %MWa sa skladá z bajtov %MBa a %MB(a+1), pričom nižší bajt slova je %MB(a+1) a vyšší bajt je %MBa.
  - Príklad 1: %MW0 čítaj ako pamäťové slovo 0. %MW0 sa skladá z bajtov %MB0 a %MB1 pričom %MB0 je vyšší a %MB1 je nižší bajt. Najnižší bit (bit 0) slova %MW0 je preto %M1.0 a najvyšší bit slova (bit 15) je %M0.7.
  - Príklad 2: %MW1 sa skladá z bajtov %MB1 a %MB2 pričom %MB2 je nižší a %MB1 je vyšší bajt. Najnižší bit (bit 0) slova %MW1 je preto %M2.0 a najvyšší bit slova (bit 15) je %M1.7. Z uvedených dvoch príkladov je zrejmé, že %MW0 a %MW1 majú spoločný bajt %MB1 a preto zmenou %MW0 ovplyvňujeme %MW1 a opačne. Ak nechceme, aby sa oblasti prepisovali, slová adresujeme s indexom +2, t. j. napr. %MW0, %MW2, %MW4, ...
- %MDa - definuje double word (dvojslovo) v pamäťovej oblasti M, kde a je číslo dvojslova. Dvojslovo %MDa sa skladá z dvoch nezávislých slov %MWa a %MW(a+2), kde %MW(a+2) je nižšie slovo %MW(a) je vyššie, t. j. skladá sa bajtov %MBa, %MB(a+1), %MB(a+2) a %MB(a+3).
  - Príklad 1: %MD0 čítaj ako pamäťové dvojslovo 0. %MD0 sa skladá z slov %MW0 a %MW2 pričom %MW0 je vyššie a %MW2 je nižšie slovo. Skladá sa z bajtov %MB0, %MB1, %MB2 a %MB3. Najnižší bit (bit 0) dvojslova %MD0 je preto %M3.0 a najvyšší bit slova (bit 31) je %M0.7.

– Príklad 2: %MD1 sa skladá z bajtov %MB1, %MB2, %MB3 a %MB4. Ide o nezávislé slová %MW1 a %MW3 pričom %MW1 je vyššie slovo a %MW3 je nižšie slovo. Najnižší bit (bit 0) dvojslova %MD1 je preto %M4.0 a najvyšší bit dvojslova je %M1.7. Z uvedených dvoch príkladov je zrejmé, že %MD0 a %MD1 majú spoločné bajty %MB1, %MB2 a %MB3 a preto zmenou hodnôt %MD0 ovplyvňujeme %MD1 a opačne. Ak nechceme, aby sa oblasti prepisovali, dvojslová adresujeme s indexom +4, t. j. napr. %MD0, %MD4, %MD8, ...

Dátové bloky možno rozdeliť podľa rôznych hľadísk. Prvým je závislosť alebo nezávislosť premenných v dátovom bloku od štruktúry funkčného bloku:

- Globálne - premenné definujeme editáciou dátového bloku. Obsahujú len skupinu statických premenných.

DDD

		Name	Data type	Start value	Retain
1		Static			
2		A	Bool	false	
3		INT_1	Int	0	
4		R	Real	0.0	
5		Z	Bool	false	
6		GGG	Byte	16#0	
7		H	Bool	false	

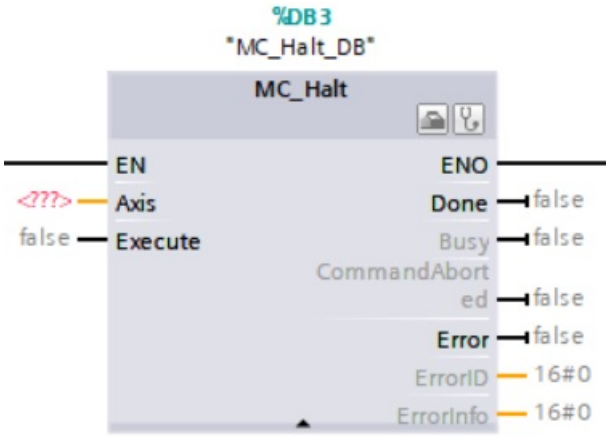
Obr. 2.5. Príklad globálneho DB s názvom DDD

- Inštančné - štruktúra závisí od rozhrania funkčného bloku. Nové premenné sa pridávajú editáciou FB a vygenerovaním nových inštančných DB. Inštančné sú napr. dátové bloky časovačov, počítadiel, ... ako aj DB vlastných FB. Obsahujú skupiny premenných Input, Output, ... Na Obr. 2.6 je inštančný dátový blok funkčného bloku MC_Halt (Obr. 2.7).

MC_Halt_DB

		Name	Data type	Start value	Retain
1		Input			
2		Axis	TO_SpeedAxis		
3		Execute	Bool	false	
4		Output			
5		Done	Bool	false	
6		Busy	Bool	false	
7		CommandAborted	Bool	false	
8		Error	Bool	false	
9		ErrorID	Word	16#0	
10		ErrorInfo	Word	16#0	
11		InOut			
12		Static			
13		FB_ID	DInt	0	

Obr. 2.6. Príklad inštančného DB (MC_Halt)



Obr. 2.7. Volanie inštrukcie MC_Halt s dátovým blokom MC_Halt_DB

Inštančné DB delíme na:

- Jednoinštančný (Single Instance) - Vytvorí inštančný dátový blok pre zvolený funkčný blok s adresou %DBa, kde a je číslo dátového bloku. Tento dátový blok (napr. pre časovač) má len jednu inštanciu. V TIA Portal sa ukladá do System blocks. Ak by sme ho použili vo FB, inštancia by bola volaná viac krát a stavy by sa prepisovali.
- Viac-inštančný (Multi-Instance) - Vytvorí inštančný dátový blok ako statickú premennú vo FB. Ide o inštančný dátový blok v inštančnom dátovom bloku. Pri volaní toho istého FB s rôznymi inštančnými DB sú (vnorené) multiinštančné DB nezávislé.

Rozdelenie dátových blokov na základe spôsobu organizácie poradia premenných:

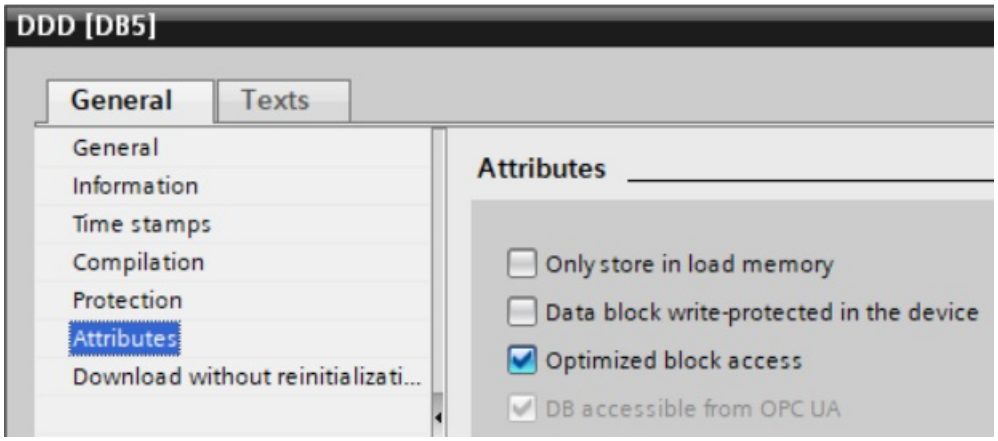
- Optimalizované - nie je možné pristupovať k fyzickým adresám premenných, ale len k symbolom (napr. DDD.A, DDD.INT_1, ...). Pri každej kompilácii môže dôjsť k preusporiadaniu premenných za účelom minimalizácie veľkosti pamäte. Príklady optimalizovaných DB sú na Obr. 2.5 a 2.6.
- Neoptimalizované - Premenné sú uložené podľa zoznamu v DB (Obr. 2.8). Umožňujú pristupovať k fyzickým adresám (pre dátový blok číslo 5 fyzická adresa %DB5.DBX0.0 adresuje symbolickú premennú DDD.A). Fyzické adresy premenných sú dané stĺpcom Offset. Celé číslo je počiatočným bajtom adresy.

DDD

	Name	Data type	Offset	Start value	Retain
1	Static				
2	A	Bool	0.0	false	
3	INT_1	Int	2.0	0	
4	R	Real	4.0	0.0	
5	Z	Bool	8.0	false	
6	GGG	Byte	9.0	16#0	
7	H	Bool	10.0	false	

Obr. 2.8. Príklad neoptimalizovaného DB

Zmenu medzi optimalizovaným a neoptimalizovaným DB vykonáme vo vlastnostiach DB v menu *Attributes* zmenou nastavenia *Optimized block access* (Obr. 2.9).



Obr. 2.9. Príklad neoptimalizovaného DB

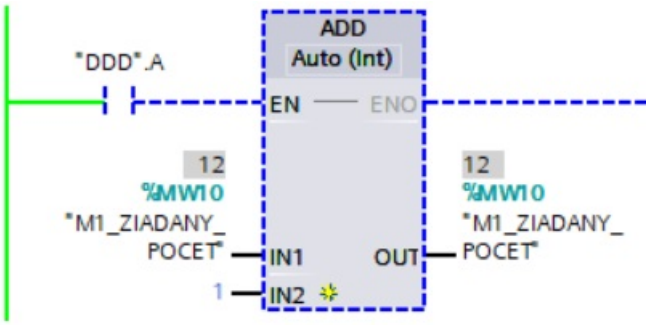
Fyzické adresy neoptimalizovaných DB sú jednoznačne definované počiatočným číslom bajtu:

- %DBa.DBXb.c - je bit (BOOL) č. *c* v dátovom bloku č. *a* uložený v bajte č. *b*.
  - Príklad: V predchádzajúcom Obr. 2.8 fyzická adresa %DB5.DBX0.0 referuje na premennú DDD.A. Bity %DB5.DBX0.1 až %DB5.DBX0.7 a %DB5.DBX1.0 až %DB5.DBX1.7 nie sú využité. Je to príklad neoptimálneho využitia pamäte.
- %DBa.DBBb - je bajt č. *b* v dátovom bloku č. *a*.
  - Príklad: V Obr. 2.8 je %DB5.DBB9 premenná DDD.GGG. Adresa %DB5.DBB0 adresuje bajt 0, ktorého súčasťou je aj vyššie spomenutý bit %DB5.DBX0.0. Ide o najnižší bit bajtu, čiže mapovanie pamäte je analogické oblasti %M.
- %DBa.DBWb - je word č. *b* v dátovom bloku č. *a*.
  - Príklad: V Obr. 2.8 adresa %DB5.DBW2 referuje na premennú DDD.INT_1. Tu je potrebné poznamenať, že kým dátový typ WORD pracuje so 16 bitmi, tak dátový typ INT interpretuje jeden z bitov ako znamienko. Fyzické adresy sa používajú najmä pri nulovaní pamäťových oblastí prípadne pri presune bloku dát v pamäti.
- %DBa.DBDb - je double word č. *b* v dátovom bloku č. *a*.
  - Príklad: V Obr. 2.8 je %DB5.DBD4 premenná DDD.R.

2.4 Monitorovanie tagov

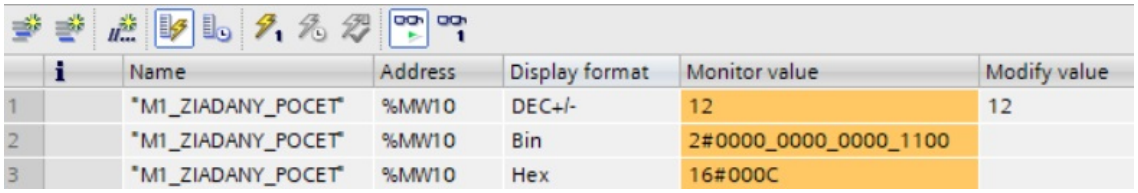
Existuje niekoľko nástrojov monitorovania a zmeny hodnôt tagov v riadiacich systémoch:

- Programový editor - V editore programu sa v závislosti od programovacieho jazyka zobrazujú hodnoty premenných. Hodnoty tagov sa zobrazujú na vstupoch a výstupoch inštrukcií alebo v tabuľkovej podobe pri textových programovacích jazykoch. Príklad monitorovanie binárnej a celočíselnej premennej je na Obr. 2.10. Binárna premenná DDD.A má hodnotu FALSE (nie je zafarbená na zeleno) a hodnota premennej M1_ZIADANY_POCET je 12. Podrobnejšie monitorovanie bude vysvetlené pri inštrukciách a riešených príkladoch.



Obr. 2.10. Príklad monitorovania hodnôt premenných v programovom editore

- V tabuľke - ktorá má rôzne názvy ako napr. variable table (VAT tabuľka), watch table (v TIA Portal). V tabuľkách sa vytvorí zoznam premenných s cieľom monitorovania a zmeny hodnôt bez potreby HMI prípadne sledovania viacerých častí programu. Príklad monitorovania tej istej premennej v rôznych sústavách je na Obr. 2.11.



	i	Name	Address	Display format	Monitor value	Modify value
1		"M1_ZIADANY_POCET"	%MW10	DEC+/-	12	12
2		"M1_ZIADANY_POCET"	%MW10	Bin	2#0000_0000_0000_1100	
3		"M1_ZIADANY_POCET"	%MW10	Hex	16#000C	

Obr. 2.11. Príklad monitorovania hodnôt vo Watch Table

- Iné objekty - okrem vyššie uvedených existujú nástroje na záznam hodnôt ako Trace, HMI, ktoré zobrazujú hodnoty v číselnej alebo grafickej podobe a ďalšie.

Hodnoty premenných monitorujeme alebo zadávame v rôznych sústavách:

- základ 2 (binary) **2**#1111111111111111
- základ 8 (octal) **8**#177777
- základ 10 (decimal) 65535 (pozn. 10# nie je potrebné)
- základ 16 (hex) **16**#**1**FFFF.

Všetky vyššie uvedené hodnoty sú rovnaké.
Hodnoty diskrétnych ale i analógových vstupov a výstupov možno vynútiť.

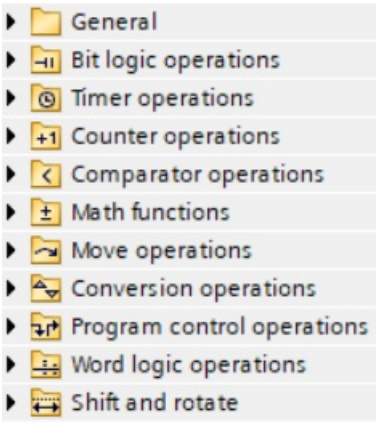
- Force („Forceovanie“) - vynútenie hodnoty TRUE alebo FALSE binárnej premennej. Program už nie je prioritný, t. j. programátor nastavuje hodnotu binárnej premennej bez ohľadu na program. V TIA Portal sa vynútenie realizuje vo Force Table.
- Unforce - zrušenie vynútenia. Ak je premenná zapísaná programom, tak jej hodnota bude závisieť od programu. V opačnom prípade bude jej hodnota TRUE alebo FALSE podľa stavu poslednej vynútenej hodnoty. Ak bola vynútená na hodnotu TRUE, a po zrušení sa ničím hodnota premennej neprepisuje, jej hodnota bude 1. Ak bola vynútená na hodnotu FALSE, a po zrušení sa ničím hodnota premennej neprepisuje, jej hodnota bude 0.

Vynútenia sú užitočné vo fáze testovania vstupov a výstupov počas oživovania zariadenia, dočasného preklemovania blokujúcich stavov a pod.

Kapitola 3

Syntax príkazov

V tejto kapitole si opíšeme vybrané príkazy (SFC a SFB) programovateľných logických automatov od spoločnosti Siemens v jazyku rebríkových schém pre softvérový nástroj TIA Portal 15.1. Detailná syntax príkazov je zdokumentovaná v manuáloch k príslušným rodinám a softvérom riadiacich systémov. Zoznam skupín inštrukcií v základnej knižnici je na Obr. 3.1. Pred vysvetlením samotných inštrukcií je nutné pochopiť poradie vykonávania inštrukcií v rebríkových schémach. Danej téme sa venuje podkapitola 3.1.



Obr. 3.1. Zoznam skupín v základnej knižnici

3.1 Poradie vykonania inštrukcií v jazyku LD

Inštrukcie v jazyku LD sa vykonávajú postupne podľa poradia rebríkov zhora nadol. V rámci rebríka sa inštrukcie vykonávajú zľava doprava a zhora nadol. V nižšie uvedených príkladoch budú nazvané doposiaľ neznáme inštrukcie podľa ich operandov (premenných). Vo všetkých príkladoch budeme uvažovať vykonanie všetkých inštrukcií. Je dôležité poznamenať, že inštrukcie sa vykonajú raz v jednom cykle.

3.1.1 Poradie inštrukcií - Príklad 1

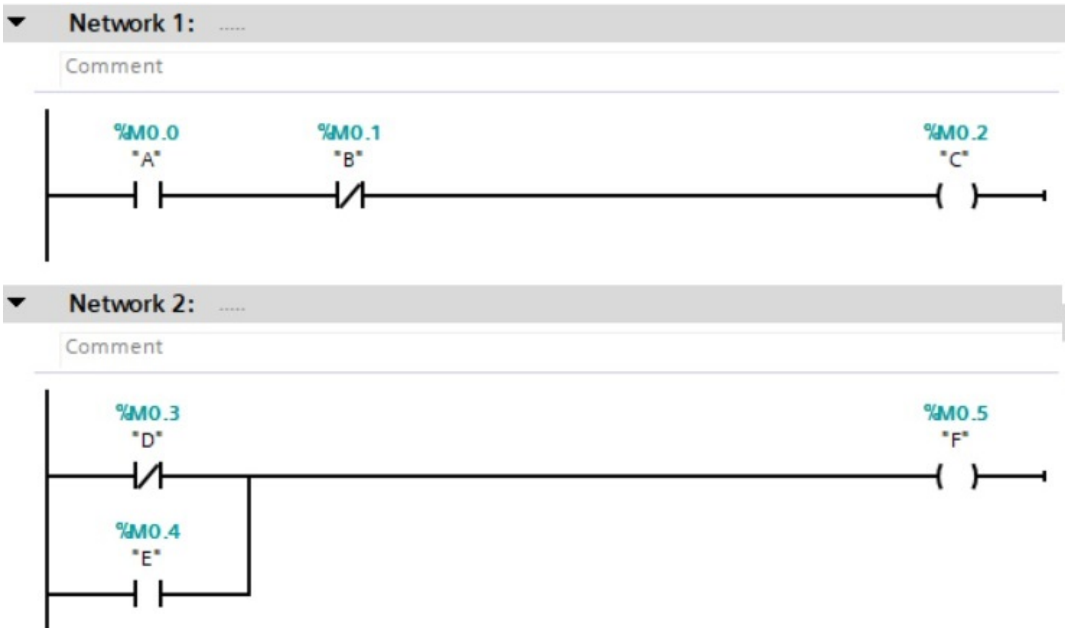
V príklade na Obr. 3.2 je zobrazený jeden rebrík, ktorý sa v riadiacich systémoch Siemens nazýva Network. Je označený ako Network 1. V ňom sú dve inštrukcie s operandami "A" a "B". Inštrukcie sa začnú vykonávať zľava doprava, t. j. prvou vykonanou inštrukciou bude

spínací kontakt testujúci stav bitu “A“ a druhou zápis binárnej hodnoty do premennej “B“. Vykonanie zhora nadol sa nerealizuje lebo príklad neobsahuje vetvenia. V ďalších príkladoch sa v texte budeme odvolávať na inštrukcie podľa im priradených operandov - napr. “A“, “B“.

Obr. 3.2. Príklad vykonania dvoch inštrukcií

3.1.2 Poradie inštrukcií - Príklad 2

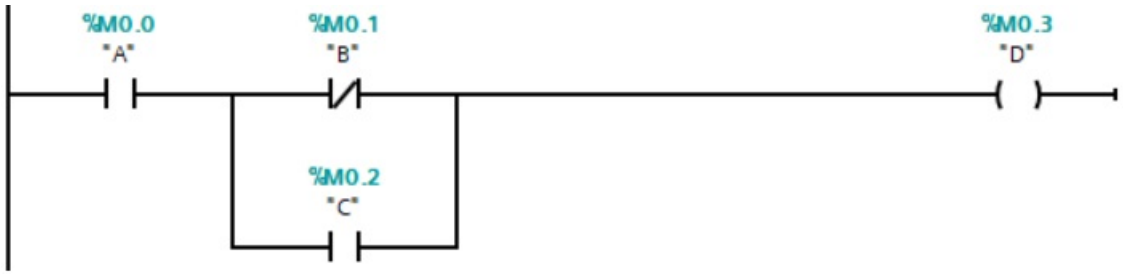
V príklade na Obr. 3.3 sú inštrukcie usporiadané v dvoch rebríkoch (Network 1 a 2). Ako prvý sa vykoná Network 1 a po ňom Network 2. V prvom rebríku nie sú vetvenia, preto vykonanie je zľava doprava v poradí “A“, “B“ a “C“ (pozn. “A“ a “B“ predstavujú spojenie AND, ktoré bude vysvetlené neskôr). V druhom rebríku (Network 2) postupujeme zľava doprava a zhora nadol. Po inštrukcii “D“ máme vetvenie, ktoré predstavuje OR vetvu. Tu program postupuje zhora nadol, preto druhou inštrukciou v 2. rebríku je “E“. Po dokončení spodného vetvenia sa pokračuje tam kde program pôvodne prebiehal zľava doprava. Poslednou vykonanou inštrukciou bude “F“. V ďalších príkladoch budú kvôli úspore miesta vystrihnuté programy z rebríkov bez uvedenia čísla rebríkov a komentárov (pozn. v niektorých prípadoch aj graficky editované, aby sa zmestili na šírku strany).



Obr. 3.3. Príklad vykonania inštrukcií v dvoch rebríkoch

3.1.3 Poradie inštrukcií - Príklad 3

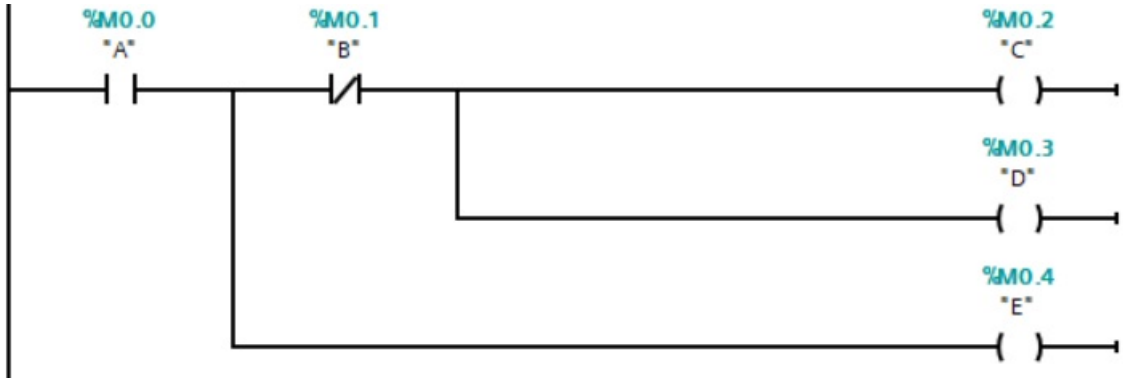
V príklade na Obr. 3.4 sa začnú vykonávať inštrukcie zľava doprava inštrukciou "A". Pri vetvení sa pokračuje hornou časťou rebríka "B". Pred pokračovaním sa dokončí vetvenie, t. j. pokračuje sa zhora nadol inštrukciou "C" a dokončí vetva inštrukciou "D".



Obr. 3.4. Príklad vykonania inštrukcií v rebríku

3.1.4 Poradie inštrukcií - Príklad 4

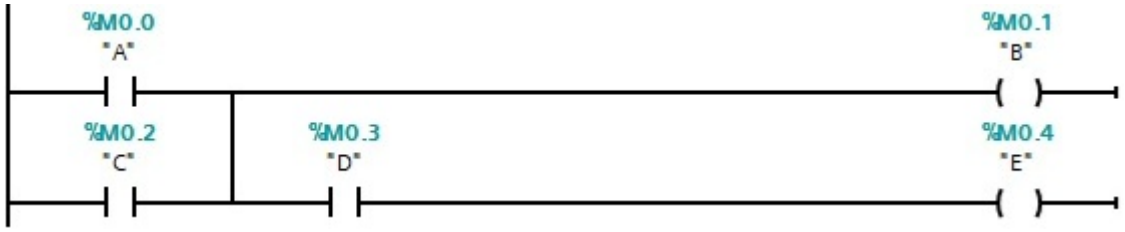
V príklade na Obr. 3.5 po vykonaní inštrukcie "A" na základe pravidla zľava doprava a až potom zhora nadol pokračuje program vykonaním "B" a nie inštrukciou "E". Po "B" sa program vetví, ale poradie je zhora nadol, t. j. "C" a potom "D". Poslednou inštrukciou je "E".



Obr. 3.5. Príklad vykonania inštrukcií v rebríku

3.1.5 Poradie inštrukcií - Príklad 5

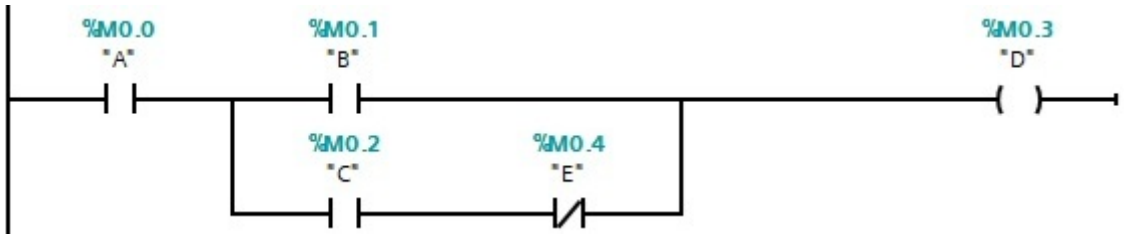
V príklade na Obr. 3.6 sa najprv vykoná "A". Za inštrukciou je vetvenie, ktoré je potrebné dokončiť, preto nasleduje "C" (ide o spojenie "A" OR "C"). Pokračuje sa v hornej vetve "B" a potom v dolnej vetve "D" a nakoniec "E".



Obr. 3.6. Príklad vykonania inštrukcií v rebríku

3.1.6 Poradie inštrukcií - Príklad 6

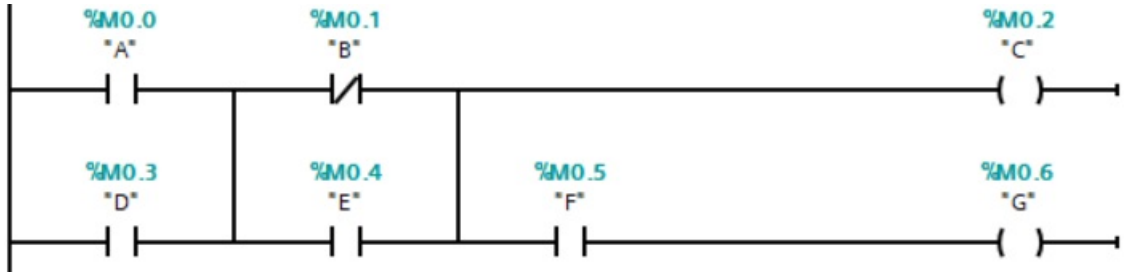
Majme upravený príklad z Obr. 3.4 v podobe ako je na Obr. 3.7. Prvou vykonanou inštrukciou je "A" po ktorej je vetvenie. Najprv sa vykonajú inštrukcie v hornej vetve, t. j. "B" a potom v dolnej vetve v poradí "C" a "E". Poslednou inštrukciou bude "D".



Obr. 3.7. Príklad vykonania inštrukcií v rebríku

3.1.7 Poradie inštrukcií - Príklad 7

V príklade na Obr. 3.8 sa vykoná "A". Pokračuje sa zhora nadol pre dokončenie OR vetvenia inštrukciou "D". Program pokračuje v hornej časti tam kde prestal (pozn. medzi "A" a "D"), t. j. vykoná sa "B". Opäť sa pokračuje zhora nadol kvôli OR vetveniu, t. j. "E" a pokračuje v hornej vetve dokončením "C". Po vykonaní "A", "D", "B", "E" a "C" program pokračuje zhora nadol inštrukciou "F" a zľava doprava "G".

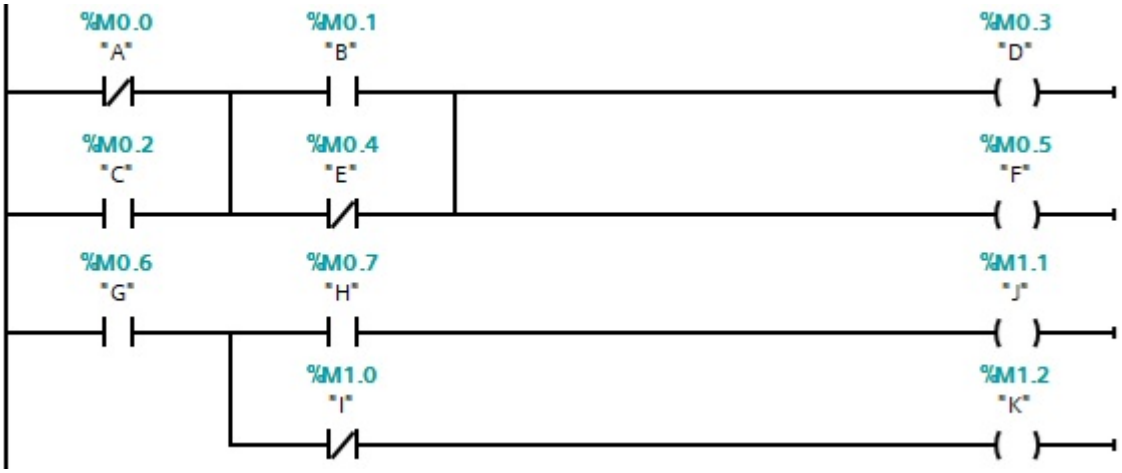


Obr. 3.8. Príklad vykonania inštrukcií v rebríku

3.1.8 Poradie inštrukcií - Príklad 8

Majme upravený príklad z Obr. 3.4 v podobe ako je na Obr. 3.9. Prvou vykonanou inštrukciou je "A" po ktorej je vetvenie. Najprv sa vykoná inštrukcia "C" v spodnej vetve a pokračuje sa v hornej časti rebríka za inštrukciou "A". Vykoná sa "B" pri ktorej je opäť vetvenie, preto sa dokončí spodná OR vetva (inštrukcia "E") a pokračuje sa v hornej vetve.

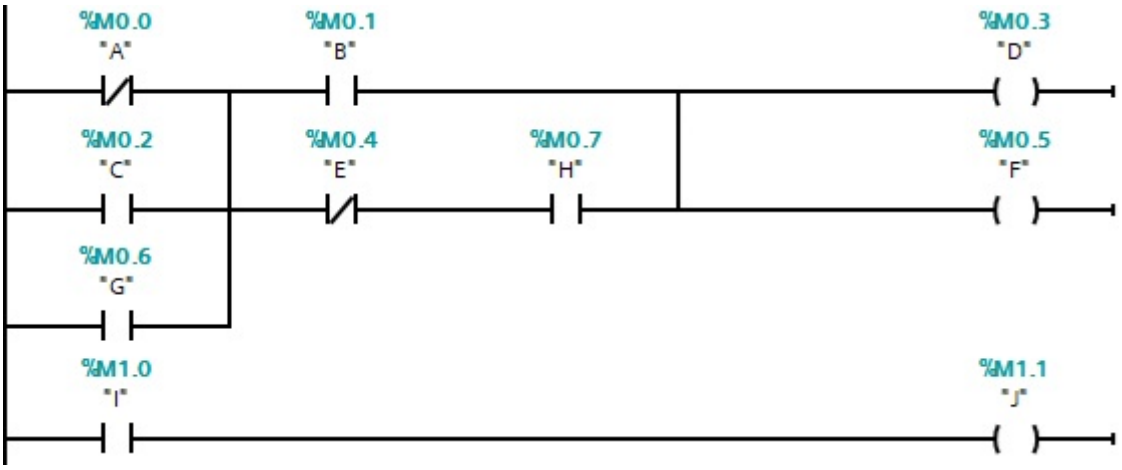
Dokončí sa “D“ a v spodnej vetve “F“, lebo sa inštrukcie vykonávajú zhora nadol. V ďalšej časti sa začína inštrukciami v poradí “G“, “H“ a “J“ v hornej vetve a až potom program pokračuje spodnou vetvou “I“ a nakoniec “K“.



Obr. 3.9. Príklad vykonania inštrukcií v rebríku

3.1.9 Poradie inštrukcií - Príklad 9

Majme upravený príklad na Obr. 3.10. Prvou vykonanou inštrukciou je “A“ po ktorej je vetvenie. Aby sa dokončilo OR vetvenie, vykonajú sa “C“ a “G“. Program pokračuje v hornej vetve inštrukciou “B“, ktorá končí vetvením, preto sa dokončí spodná OR vetva inštrukciami “E“ a “H“. Opäť sa pokračuje v hornej vetve inštrukciou “D“ a dokončí sa spodná vetva “F“. Na záver máme ešte inštrukcie v poradí “I“ a “J“.

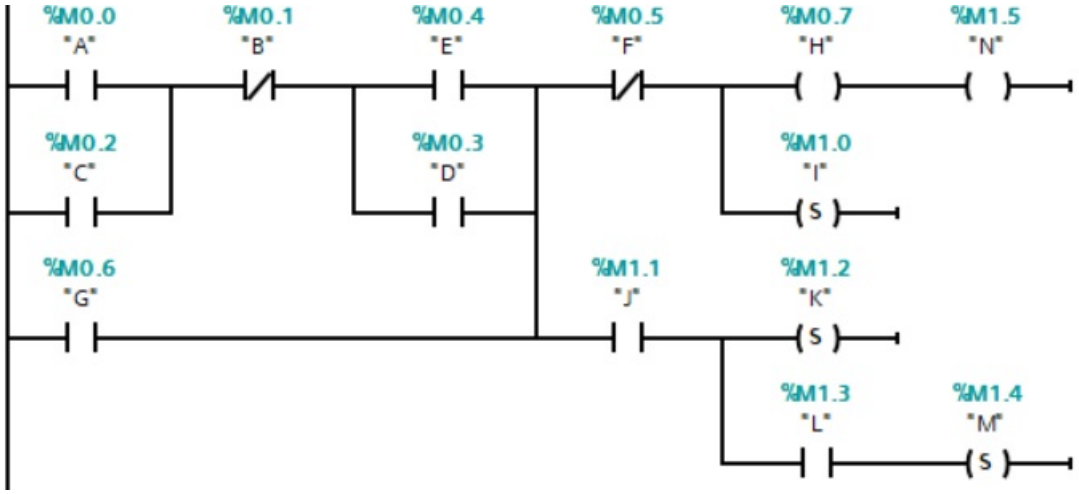


Obr. 3.10. Príklad vykonania inštrukcií v rebríku

3.1.10 Poradie inštrukcií - Príklad 10

Majme zložitejší príklad (Obr. 3.11). Program začína vľavo hore, t. j. inštrukciou “A“, vykoná vetvenie “C“, pokračuje “B“, “E“, dokončí vetvenie “D“. Na tomto mieste vidíme ďalšie vetvenie k doterajším inštrukciám, t. j. program sa vykonáva zhora nadol inštrukciou

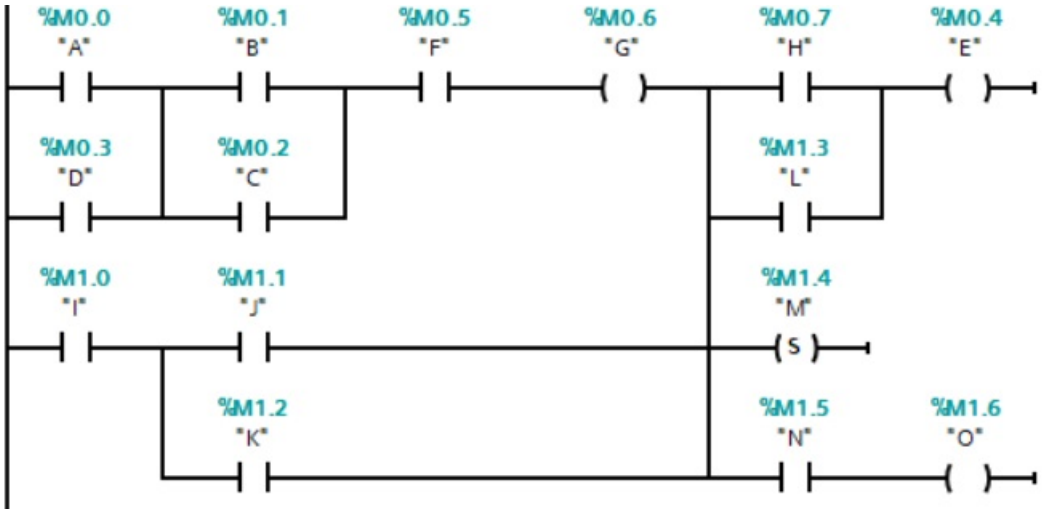
"G". Túto časť by sme mohli zhrnúť vzťahom ((A OR C) AND NOT(B) AND (E OR D)) OR G. Po tejto časti máme vetvenie začínajúce "F" a vetvenie začínajúce inštrukciou "J". Program pokračuje v hornej časti (princíp zhora nadol) inštrukciou "F". V tomto bode máme nové vetvenie. Program pokračuje najprv hornou vetvou a potom spodnou (princíp zhora nadol). V hornej vetve zľava doprava, preto "H" a "N". V spodnej vetve "I". Až teraz program pokračuje v druhom vetvení "J". Po "J" je opäť vetvenie, pokračujeme hornou časťou "K", potom dolnou časťou "L" a "M".



Obr. 3.11. Príklad vykonania inštrukcií v rebríku

3.1.11 Poradie inštrukcií - Príklad 11

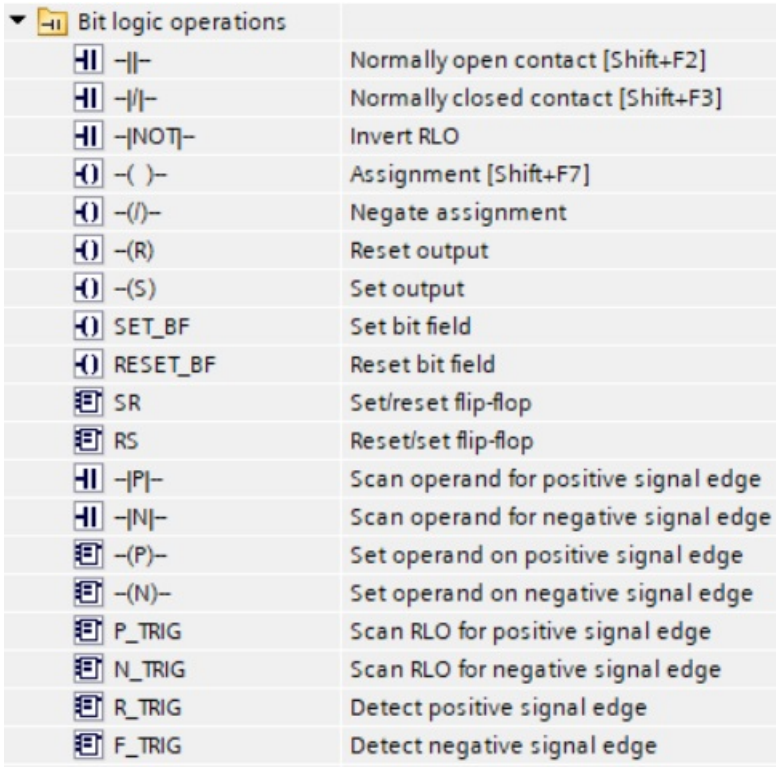
Majme ešte jeden zložitejší príklad (Obr. 3.12). Program vykonáva inštrukcie v poradí "A", "D" (OR vetva), "B", "C" (ďalšia OR vetva), "F", "G". Na tomto mieste je vetvenie k vyššie opísaným inštrukciám, preto sa pokračuje "I", "J" a "K". Po dokončení tejto časti sa pokračuje hore nedokončenou vetvou "H", "L" (OR vetva), "E", "M" a nakoniec "N" a "O".



Obr. 3.12. Príklad vykonania inštrukcií v rebríku

3.2 Bitové inštrukcie (Bit logic operations)

Pod inštrukciami bitovej logiky rozumieme také inštrukcie, ktoré pracujú s booleovskými (bitovými) premennými, t. j. dokážu spracovať len dva logické stavy FALSE (“0“) a TRUE (“1“). Zoznam bitových inštrukcií v TIA Portal pre CPU S7-1200 je na Obr. 3.13.



Obr. 3.13. Zoznam bitových inštrukcií

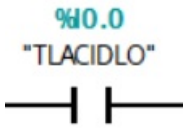
Bitové inštrukcie patria k najčastejšie používaným inštrukciám v riadení diskrétnych udalostných systémov. Môžeme ich zadeliť do dvoch kategórií.

Prvá skupina inštrukcií (tzv. kontakty) sa používa na vyhodnotenie stavov digitálnych vstupných signálov (snímačov, tlačidiel, prepínačov atď.), ale aj pamäťových bitov, ktoré reprezentujú stavy riadeného systému. Táto skupina príkazov reprezentuje podmienkovú časť programu.

Druhá skupina (tzv. cievky) modifikuje hodnoty operandov na TRUE a FALSE v závislosti od podmienkovej časti programu. Tieto inštrukcie ovládajú digitálne výstupy (spustenie / zastavenie - motorov, čerpadiel, ...; otvorenie / zatvorenie - ventilov, bezpečnostných brán, úpiniek, ...), ale aj stavy riadeného procesu.

3.2.1 Spínací kontakt

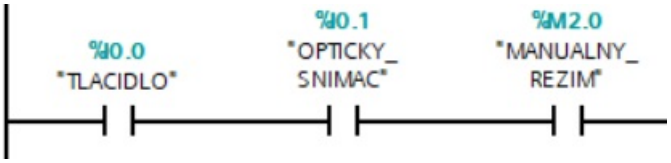
Prvou v sérii bitových inštrukcií je *spínací kontakt* (Obr. 3.14).



Obr. 3.14. Symbol inštrukcie *spínací kontakt*

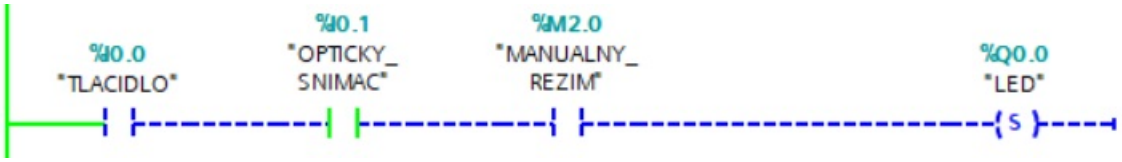
Inštrukcia má jeden operand, ktorý sa zadáva nad symbolom inštrukcie. Aktivácia spínacieho kontaktu závisí od stavu priradeného operandu. Keď má operand hodnotu TRUE, spínací kontakt sa zopne a stav signálu na výstupe sa nastaví na hodnotu vstupu (hodnotu TRUE). Keď má operand hodnotu FALSE, spínací kontakt nie je aktivovaný a stav signálu inštrukcie je nastavený na hodnotu vstupu (hodnotu FALSE). Zjednodušene povedané, výstupom inštrukcie je hodnota jej operandu (bitu). [11] V príklade na Obr. 3.14 je operandom digitálny vstup TLACIDLO s fyzickou adresou %I0.0.

Dva alebo viacero spínacích kontaktov (alebo výstupov iných inštrukcií, ktoré ovplyvňujú stav RLO - výsledok logickej operácie) zapojených do série predstavuje logické spojenie AND. Na Obr. 3.15 je uvedený príklad zapojenia troch inštrukcií do série. Ukážka programu na obrázku je bez monitorovania stavov premenných. Prvé dve inštrukcie majú operandy digitálne vstupy (%I) a tretia inštrukcia pamäťový bit (%M) reprezentujúci stav systému (režim činnosti).



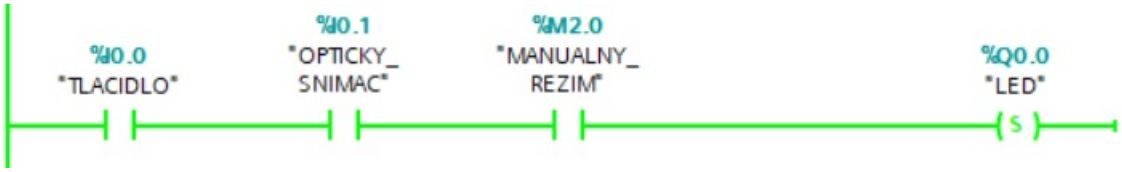
Obr. 3.15. Príklad zapojenia inštrukcií spínací kontakt do podmienky AND

Pri zapojení AND podmienková časť umožní vykonanie príkazu, ak sú všetky kontakty zopnuté (zatvorené). Na Obr. 3.16 majú operandy nasledovné stavy TLACIDLO = FALSE, OPTICKY_SNIMAC = TRUE a MANUALNY_REZIM = FALSE. Stav druhej inštrukcie s operandom OPTICKY_SNIMAC indikuje zelenou farbou výstup inštrukcie, ktorý má hodnotu TRUE. Ľavá časť rebríkovej schémy animovaná zelenou farbou nie je spojená s inštrukciou SET s operandom LED, preto sa inštrukcia SET nevykoná.



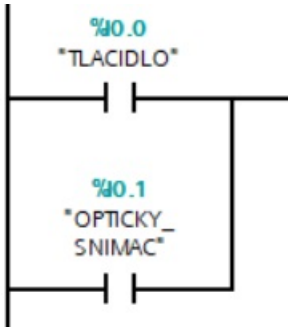
Obr. 3.16. Príklad neplatnej podmienky AND

Na Obr. 3.17 majú všetky operandy hodnotu TRUE, preto výstupy všetkých troch normálne otvorených kontaktov sú v stave TRUE. Ľavá časť rebríkovej schémy je spojená s inštrukciou SET, ktorá sa v aktuálnom cykle monitorovania vykoná.



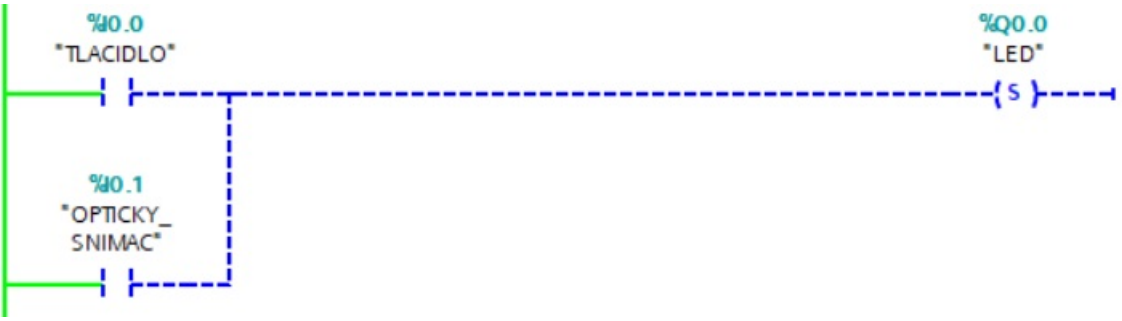
Obr. 3.17. Príklad platnej podmienky AND

Paralelné zapojenie dvoch a viac normálne otvorených kontaktov (príp. iných inštrukcií) vedie k logickému spojeniu OR. Na Obr. 3.18 je uvedený príklad zapojenia dvoch inštrukcií v podmienke OR.



Obr. 3.18. Príklad zapojenia normálne otvorených kontaktov do podmienky OR

Na Obr. 3.19 majú všetky operandy hodnotu FALSE, preto výstupy všetkých normálne otvorených kontaktov sú v stave FALSE. Inštrukcia SET nie je spustená.

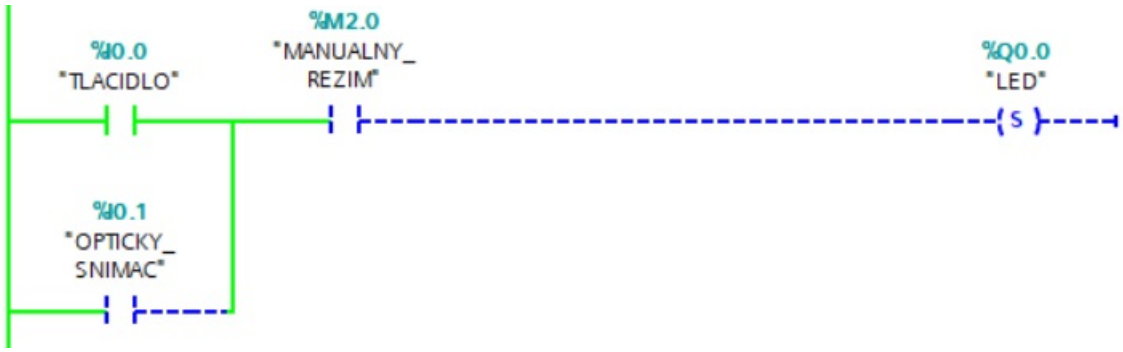


Obr. 3.19. Príklad neplatnej podmienky OR

Na Obr. 3.20 sú hodnoty operandov TLACIDLO = FALSE a OPTICKY_SNIMAC = TRUE. Inštrukcia SET je spustená lebo platí aspoň jedna z OR vetiev (v príklade spodná vetva).

Obr. 3.20. Príklad platnej podmienky OR

Inštrukcie možno kombinovať sériovo a paralelne, a tak pomocou nich vytvárať kombinácie AND a OR podmienky. Na Obr. 3.21 má TLACIDLO hodnotu TRUE a zvyšné premenné hodnoty FALSE. Podmienka OR platí, lebo v hornej vetve je výstupom inštrukcie TRUE. Výsledok OR podmienky je prvou časťou AND podmienky. Druhou časťou je spracovanie operandu MANUALNY_REZIM inštrukciou *normálne otvorený kontakt*. Nakoľko výstupom tejto inštrukcie je FALSE, podmienka AND neplatí a inštrukcia SET nie je vykonaná.



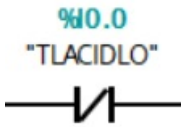
Obr. 3.21. Príklad neplatnej kombinácie podmienok OR a AND

Ekvivalentný program v jazyku ST je:

```
IF (TLACIDLO OR OPTICKY_SNIMAC) AND MANUALNY_REZIM THEN
      LED:=TRUE;
END_IF;
```

3.2.2 Rozpínací kontakt

Ďalšou inštrukciou je *rozpínací kontakt* (Obr. 3.22).



Obr. 3.22. Symbol inštrukcie *rozpínací kontakt*

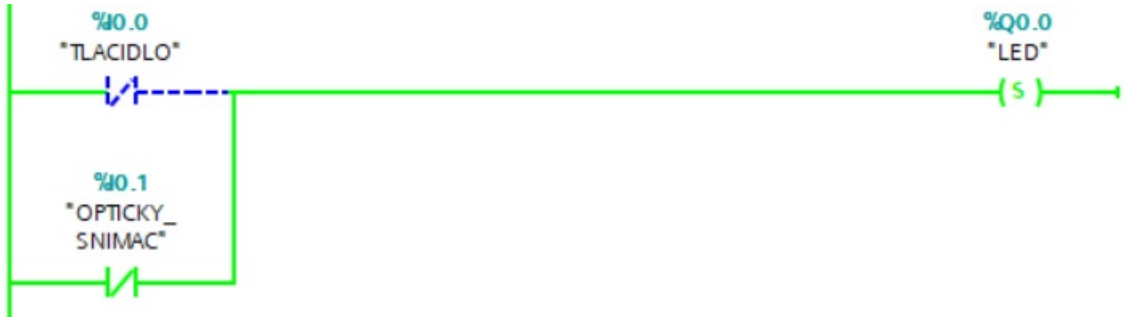
Inštrukcia má jeden operand, ktorý sa podobne ako pri spínacom kontakte zadáva nad symbolom inštrukcie. Aktivácia rozpínacieho kontaktu závisí od stavu priradeného operandu. Keď má operand hodnotu FALSE, rozpínací kontakt sa zopne a stav signálu na výstupe sa nastaví na negovanú hodnotu vstupu (hodnotu TRUE). Keď má operand hodnotu TRUE, rozpínací kontakt nie je aktivovaný a stav signálu inštrukcie je nastavený na negovanú hodnotu vstupu (hodnotu FALSE). Zjednodušene povedané, výstupom inštrukcie je negovaná hodnota jej parametra. [11] Príklady použitia inštrukcie sú na Obr. 3.23 a 3.24. Na prvom obrázku má vstup TLACIDLO hodnotu TRUE, preto jeho výstup je FALSE. Inštrukcia SET nie je vykonaná. Na druhom obrázku má vstup TLACIDLO hodnotu FALSE. Jeho výstup je hodnota TRUE (animovaná zelenou farbou), preto sa inštrukcia SET vykoná.

Obr. 3.23. Prvý príklad výstupu *rozpínacieho kontaktu*

Obr. 3.24. Druhý príklad výstupu *rozpínacieho kontaktu*

Na Obr. 3.25 a 3.26 sú uvedené zapojenia inštrukcie v podmienkach AND a OR. Na prvom obrázku má vstup TLACIDLO hodnotu TRUE a vstup OPTICKY_SNIMAC hodnotu FALSE. Podmienka AND neplatí, preto inštrukcia SET nie je vykonaná. Na druhom obrázku majú vstupy rovnaké hodnoty ako v predchádzajúcim príklade. Druhá (spodná) časť OR podmienky je platná, preto sa inštrukcia SET vykoná.

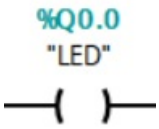
Obr. 3.25. Príklad neplatnej podmienky AND



Obr. 3.26. Príklad platnej podmienky OR

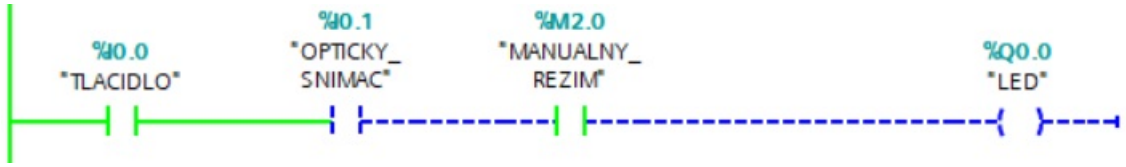
3.2.3 Priradenie

Symbol inštrukcie *priradenie* je na Obr. 3.27. Slúži na modifikáciu hodnoty bitovej premennej. Operand, ktorého hodnota sa má nastaviť, sa zadáva nad symbol inštrukcie. Ak má výsledok RLO na vstupe inštrukcie (cievky) stav signálu TRUE, zadaný operand sa nastaví na TRUE. Ak je stav signálu na vstupe cievky FALSE, bit zadaného operandu sa nastaví na FALSE. Inak povedané, kým platí podmienka pred inštrukciou, operand má hodnotu TRUE, inak FALSE. Inštrukcia nemá vplyv na RLO. RLO zo vstupu inštrukcie sa posiela priamo na jej výstup.

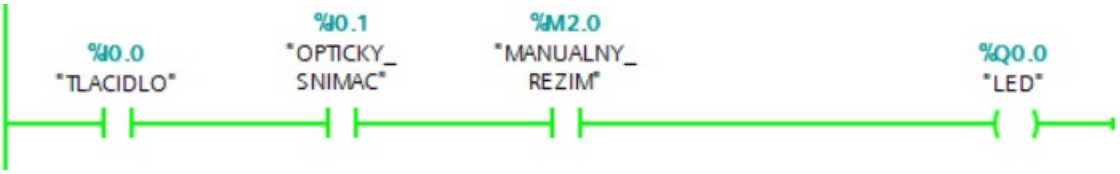


Obr. 3.27. Symbol inštrukcie *priradenie*

Na Obr. 3.28 majú premenné TLACIDLO a MANUALNY_REZIM hodnoty TRUE a premenná OPTICKY_SNIMAC hodnotu FALSE. Podmienka AND neplatí, preto stav RLO je FALSE. Hodnota operandu LED sa nastaví na FALSE. V ďalšom príklade (Obr. 3.29) podmienka AND platí lebo všetky premenné v podmienkovej časti majú hodnotu TRUE. Hodnota operandu LED sa nastaví na TRUE.



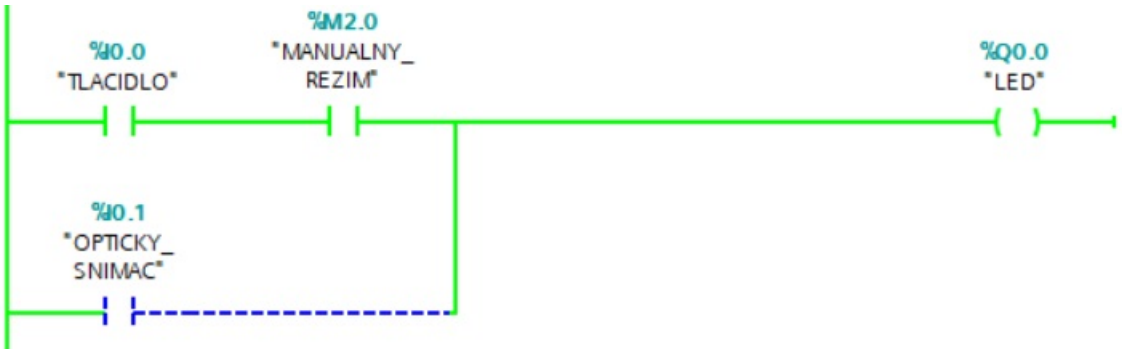
Obr. 3.28. Príklad neplatnej podmienky RLO pre *priradenie*



Obr. 3.29. Príklad platnej podmienky RLO pre *priradenie*

Neodporúča sa viacnásobný zápis bitového operandu pomocou tejto inštrukcie lebo na hodnotu premennej má vplyv len posledná podmienka. Príklad nesprávneho použitia inštrukcie *priradenie* je na Obr. 3.30. V hornej vetve platí podmienka AND, ktorá zapíše do LED hodnotu TRUE, ale v spodnej časti je ďalšou inštrukciou *priradenie* prepísaná na FALSE. Namiesto viacnásobného zápisu je vhodné použiť *priradenie* len raz s vhodnou kombináciou vstupných podmienok. Úprava programu z Obr. 3.30 je na 3.31.

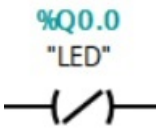
Obr. 3.30. Príklad nesprávneho použitia inštrukcie *priradenie*



Obr. 3.31. Príklad správneho použitia inštrukcie *priradenie*

3.2.4 Negované priradenie

Symbol inštrukcie *negované priradenie* je na Obr. 3.32. Inštrukcia invertuje výsledok RLO a priradí ho k určenému operandu. Keď je RLO na vstupe inštrukcie TRUE, operand sa vynuluje (zapíše sa na FALSE). Keď je RLO na vstupe inštrukcie FALSE, operand je nastavený na TRUE. Operand je zadávaný nad symbolom inštrukcie.



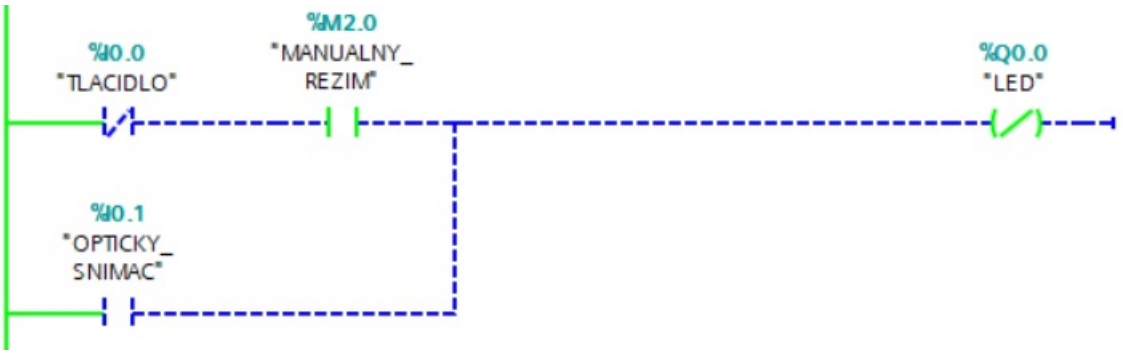
Obr. 3.32. Symbol inštrukcie *negované priradenie*

Na Obr. 3.33 je jednoduchý príklad použitia inštrukcie *negované priradenie*. Stav vstupu TLACIDLO je TRUE, preto hodnota LED je zapísaná na FALSE. Ekvivalentný program

je na Obr. 3.34. *Negované priradenie* má zmysel použiť pri zložitejšej podmienkovej časti programu. Príklad takého programu je na Obr. 3.35. Podmienka OR neplatí, preto hodnota bitu LED je zapísaná na TRUE.

Obr. 3.33. Príklad použitia inštrukcie *negované priradenie*

Obr. 3.34. Ekvivalentný program

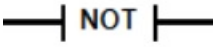


Obr. 3.35. Zložitejší príklad použitia inštrukcie *negované priradenie*

3.2.5 Invertovanie stavu RLO

Inštrukcia *invertovanie stavu RLO* (Obr. 3.36) invertuje hodnotu signálu RLO. Ak je hodnota signálu na vstupe inštrukcie TRUE, výstup inštrukcie má hodnotu FALSE. Ak je hodnota signálu na vstupe inštrukcie FALSE, výstup má hodnotu TRUE.

Jednoduché príklady sú na Obr. 3.37 a Obr. 3.38. V prvom príklade má TLACIDLO stav TRUE, preto výstupom inštrukcie *normálne otvorený kontakt* je TRUE. Stav RLO je nastavený na TRUE. Táto hodnota je invertovaná inštrukciou *invertovanie stavu RLO* na FALSE. Na výstup LED sa zapíše hodnota FALSE. V druhom príklade má TLACIDLO stav FALSE. Výstup inštrukcie *spínací kontakt* je FALSE, ktorý je aj stavom RLO. Táto hodnota je invertovaná na TRUE a zapísaná do LED. Tento príklad demonštruje stav, kedy inštrukcia *priradenie* nie je pripojená k ľavej časti rebríka (tok signálu je prerušený pri tlačidle), napriek tomu sa do operandu LED zapisuje hodnota TRUE.



Obr. 3.36. Symbol inštrukcie *invertovanie stavu RLO*

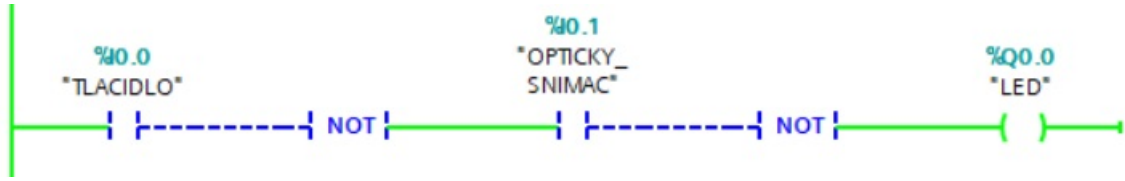
Obr. 3.37. Príklad animácie inštrukcie *invertovanie stavu RLO*

Obr. 3.38. Príklad animácie inštrukcie *invertovanie stavu RLO*

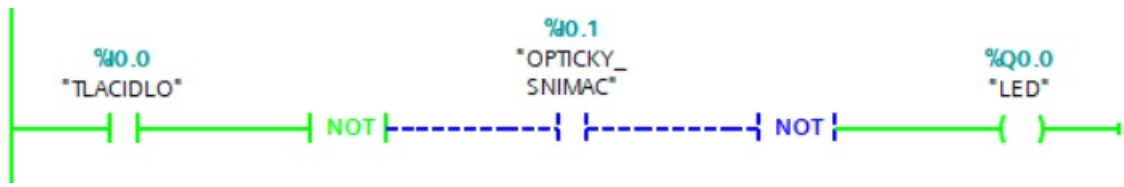
Viacnásobné nasadenie inštrukcie v tom istom rebríku môže zneprehľadniť pochopenie podmienkovej časti programu. Na Obr. 3.39 až 3.42 budú uvedené všetky kombinácie hodnôt vstupov:

- Na Obr. 3.39 sú hodnoty vstupov TLACIDLO = FALSE a OPTICKY_SNIMAC = FALSE.
- Na Obr. 3.40 sú hodnoty vstupov TLACIDLO = TRUE a OPTICKY_SNIMAC = FALSE.
- Na Obr. 3.41 sú hodnoty vstupov TLACIDLO = FALSE a OPTICKY_SNIMAC = TRUE.
- Na Obr. 3.42 sú hodnoty vstupov TLACIDLO = TRUE a OPTICKY_SNIMAC = TRUE.

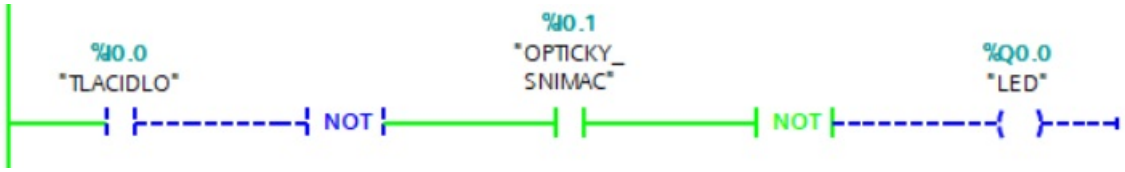
V ďalších častiach skrípt sa budeme snažiť danej inštrukcii vyhýbať. Dôvodom je aj jej absencia v iných programovacích nástrojoch od rôznych výrobcov riadiacich systémov.



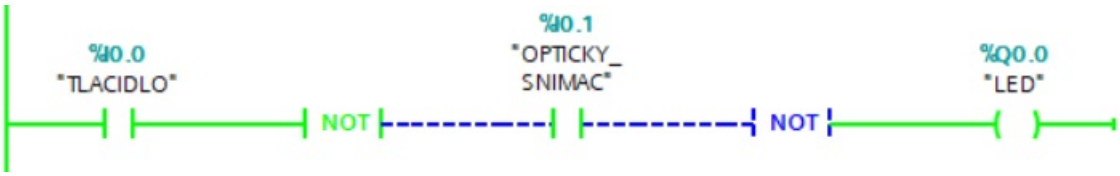
Obr. 3.39. Príklad animácie zložitejších podmienok s inštrukciou *invertovanie stavu RLO*



Obr. 3.40. Príklad animácie zložitejších podmienok s inštrukciou *invertovanie stavu RLO*



Obr. 3.41. Príklad animácie zložitejších podmienok s inštrukciou *invertovanie stavu RLO*



Obr. 3.42. Príklad animácie zložitejších podmienok s inštrukciou *invertovanie stavu RLO*

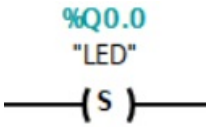
3.2.6 SET

Symbol inštrukcie *SET* je na Obr. 3.43. Inštrukcia sa používa na nastavenie hodnoty operandu na TRUE. Operand sa zadáva nad symbol inštrukcie. Inštrukcia sa vykoná len vtedy, ak je výsledok logickej operácie (RLO) na vstupe inštrukcie (cievky) TRUE. Ak signál prúdi do cievky (RLO = TRUE), špecifikovaný operand je nastavený na TRUE. Ak je RLO na vstupe cievky FALSE (t. j. do cievky nepreteká žiadny signál), stav signálu zadaného operandu zostáva nezmenený. Na Obr. 3.44 až 3.46 je uvedený program v rôznych fázach monitorovania:

1. Na Obr. 3.44 je počiatočný stav, keď TLACIDLO má hodnotu FALSE inštrukcia *SET* zatiaľ nebola vykonaná. Hodnota operandu LED je FALSE.

2. Na Obr. 3.45 má vstup TLACIDLO hodnotu TRUE (napr. tlačidlo pripojené na vstupné svorky digitálneho vstupného modulu ako spínací kontakt je práve stlačené). Inštrukcia *SET* sa vykonáva v každom cykle pokiaľ má TLACIDLO hodnotu TRUE. Hodnota LED je nastavená na TRUE.

3. Na poslednom Obr. 3.46 má TLACIDLO opäť hodnotu FALSE. Inštrukcia *SET* sa nevykonáva, ale hodnota operandu LED sa nemení. Stále má hodnotu TRUE (animované zeleným symbolom inštrukcie *SET*).



Obr. 3.43. Symbol inštrukcie *SET*

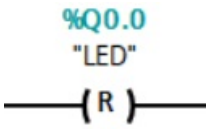
Obr. 3.44. Prvá fáza animácie programu s inštrukciou *SET*

Obr. 3.45. Druhá fáza animácie programu s inštrukciou *SET*

Obr. 3.46. Tretia fáza animácie programu s inštrukciou *SET*

3.2.7 RESET

Symbol inštrukcie *RESET* je na Obr. 3.47. Inštrukcia sa používa na resetovanie stavu signálu zadaného operandu na FALSE. Operand sa zadáva nad symbol inštrukcie. Inštrukcia sa vykoná len vtedy, ak je výsledok logickej operácie (RLO) na vstupe inštrukcie TRUE. Ak signál prúdi do inštrukcie (RLO = TRUE), špecifikovaný operand sa nastaví na FALSE. Ak je RLO na vstupe cievky FALSE (do cievky nepriteká žiadny signál), stav signálu zadaného operandu zostáva nezmenený.



Obr. 3.47. Symbol inštrukcie *RESET*

Na Obr. 3.48 až 3.50 je uvedený program v rôznych fázach monitorovania:

1. Na Obr. 3.48 je počiatočný stav, keď TLACIDLO má hodnotu FALSE a inštrukcia *RESET* zatiaľ nebola vykonaná. Hodnota operandu LED je TRUE (zelený symbol inštrukcie *RESET*). Hodnota bola nastavená napr. manuálne na stav TRUE. Túto hodnotu zmeníme v ďalšom kroku na FALSE.

2. Na Obr. 3.49 má vstup TLACIDLO hodnotu TRUE. Inštrukcia *RESET* sa vykonáva v každom cykle pokiaľ má TLACIDLO hodnotu TRUE. Hodnota LED je nastavená na FALSE.

3. Na poslednom Obr. 3.50 má TLACIDLO opäť hodnotu FALSE. Inštrukcia *RESET* sa nevykonáva, hodnota operandu je nezmenená (naďalej FALSE).

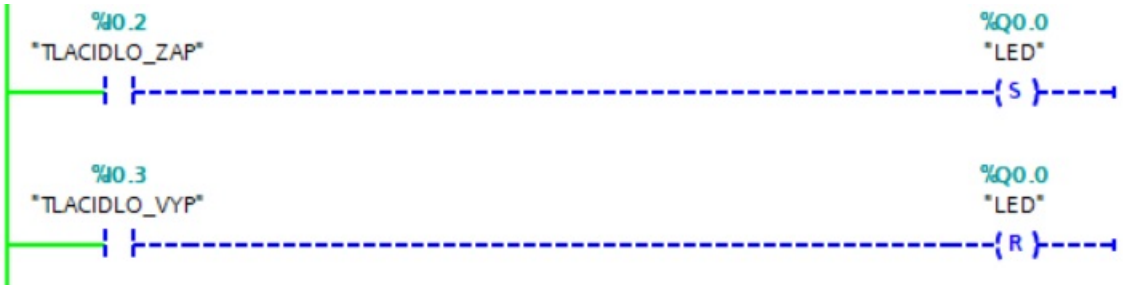
Obr. 3.48. 1. fáza animácie programu s inštrukciou *RESET*

Obr. 3.49. 2. fáza animácie programu s inštrukciou *RESET*

Obr. 3.50. 3. fáza animácie programu s inštrukciou *RESET*

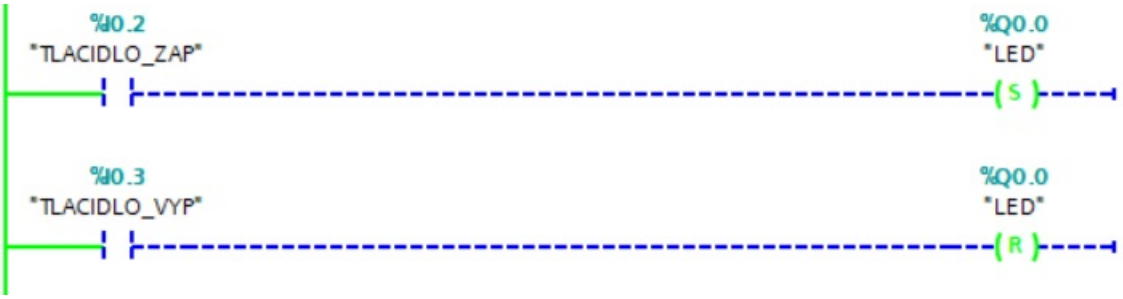
Inštrukcie *SET* a *RESET* sa používajú spolu. Ak sa pomocou inštrukcie *SET* nastaví hodnota premennej na TRUE, inštrukcia *RESET* sa použije na jeho opätovnú zmenu na FALSE. Neodporúča sa kombinácia inštrukcie *Priradenie* (príp. *Negované priradenie*) so *SET* a *RESET*, lebo dochádza k hazardným stavom. Jednoduchý príklad zapnutia a vypnutia výstupu pomocou dvoch tlačidiel pripojených ako spínacie kontakty je na Obr. 3.51 až 3.54:

1. Na Obr. 3.51 je stav keď všetky premenné majú hodnotu FALSE. Tlačidlá TLACIDLO_ZAP a TLACIDLO_VYP nie sú zatlačené a výstup LED je vypnutý.
2. Na Obr. 3.52 je zaznamenaný moment zatlačenia tlačidla TLACIDLO_ZAP. Počas cyklov keď je tlačidlo TLACIDLO_ZAP zatlačené sa neustále vykonáva inštrukcia *SET*. Výstup je aktivovaný (má stav TRUE).
3. Po uvoľnení tlačidla TLACIDLO_ZAP (Obr. 3.53) LED ostáva aktivované.
4. Zatlačením tlačidla TLACIDLO_VYP (Obr. 3.54) sa vykoná inštrukcia *RESET*, čím sa výstup deaktivuje (má hodnotu FALSE).
5. Uvoľnením tlačidla TLACIDLO_VYP sa inštrukcia *RESET* prestane vykonávať a výstup LED ostáva deaktivovaný. Stav sa zhoduje s 1. fázou (Obr. 3.51).
6. V prípade zatlačenia oboch tlačidiel sa najprv vykoná inštrukcia *SET* a následne *RESET* lebo sú platné obe podmienky na vstupoch inštrukcií. Nakoľko sa ako posledná inštrukcia vykoná *RESET*, LED má prepísaný stav z TRUE na FALSE. Na digitálny výstupný modul sa zapíše FALSE (viď animáciu inštrukcie *RESET* na Obr. 3.55).

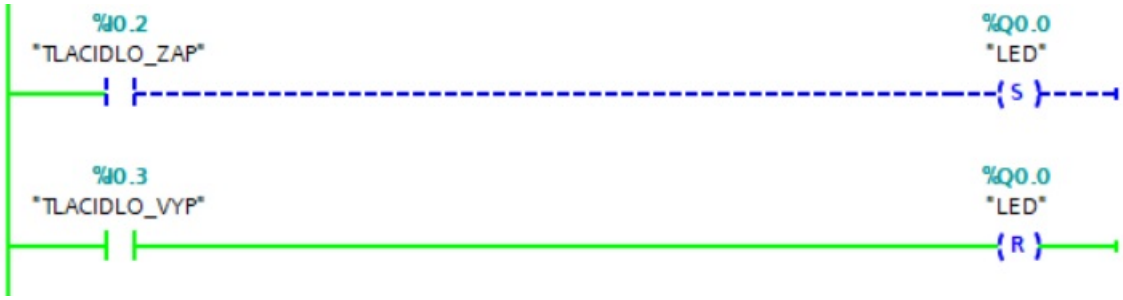


Obr. 3.51. 1. a 5. fáza animácie ovládania výstupu (LED)

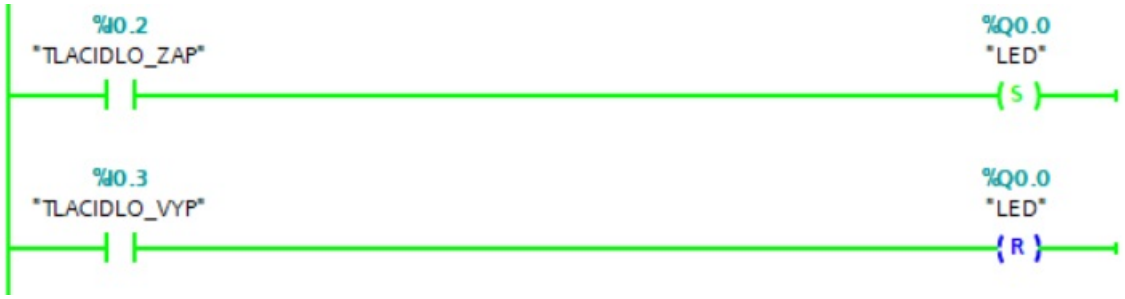
Obr. 3.52. 2. fáza animácie ovládania výstupu (LED)



Obr. 3.53. 3. fáza animácie ovládania výstupu (LED)



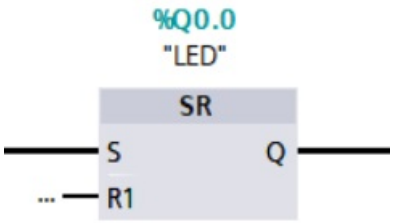
Obr. 3.54. 4. fáza animácie ovládania výstupu (LED)



Obr. 3.55. 6. fáza animácie ovládania výstupu (LED)

3.2.8 SET-RESET

V predchádzajúcich kapitolách sme opísali inštrukcie *SET* a *RESET*. Okrem uvedených samostatných inštrukcií existujú aj klopné obvody *SET-RESET (SR)* a *RESET-SET (RS)* integrujúce kombináciu základných inštrukcií. Z ich názvu vyplýva poradie vstupov a s tým súvisiace poradie vykonania inštrukcií. Symbol inštrukcie *SET-RESET* je na Obr. 3.56. Operand inštrukcie sa zadáva nad symbol inštrukcie. Výstup Q má hodnotu operandu a môžeme ním ovplyvňovať stav RLO. Vstup S reprezentuje *SET* a vstup R1 *RESET*. Číslica 1 indikuje dominantnosť inštrukcie. Ak sú aktivované oba vstupy, vykoná sa *SET* a po ňom *RESET*, preto operand bude nastavený na FALSE.



Obr. 3.56. Symbol inštrukcie *SET-RESET*

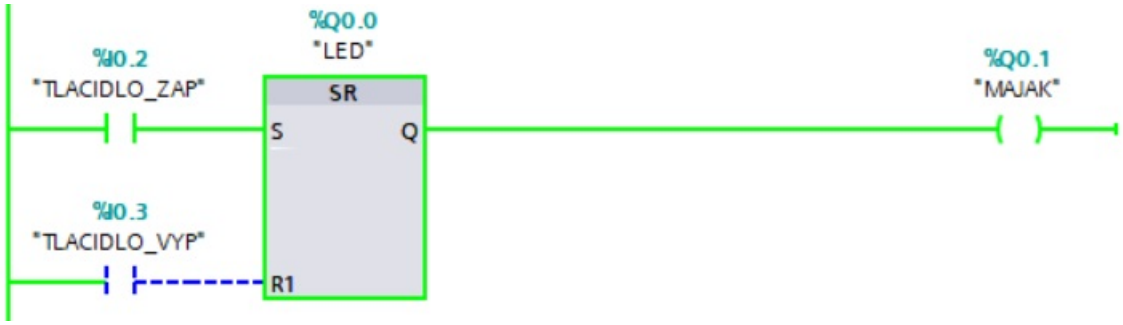
Postupnosť animácií programu s inštrukciou *SET-RESET* je na Obr. 3.57 až 3.59:

1. Na Obr. 3.57 je aktivované tlačidlo TLACIDLO_ZAP, ktorý zapína výstupy LED a MAJAK. Aktivované tlačidlo TLACIDLO_ZAP v každom cykle vykoná inštrukciu *SET*. Tlačidlo TLACIDLO_VYP nie je aktivované, preto sa inštrukcia *RESET* nevykonáva. Zápisom hodnoty TRUE do operandu LED bude výstup %Q0.0 aktivovaný. Na výstupe Q inštrukcie *SET-RESET* je stav TRUE, preto sa do výstupu MAJAK zapisuje hodnota TRUE.

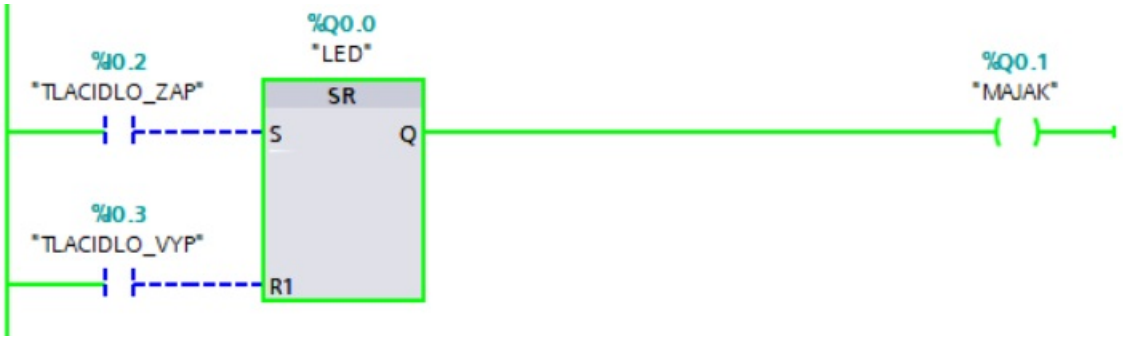
2. V ďalšom kroku je tlačido TLACIDLO_ZAP uvoľnené. Inštrukcie *SET* a *RESET* sa nevykonávajú. Výstupy sú naďalej aktívne (Obr. 3.58).

3. Na treťom obrázku (Obr. 3.59) je aktivácia vykonania *RESET* tlačidlom TLACIDLO_VYP. Výstupy sú nastavené na FALSE.

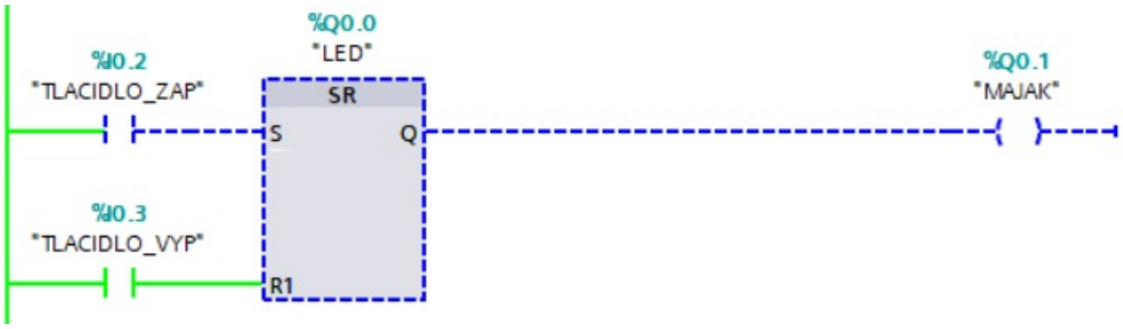
4. Na poslednom obrázku (Obr. 3.60) sú aktivované tlačidlá TLACIDLO_ZAP a TLACIDLO_VYP. V každom cykle sa najprv vykoná inštrukcia *SET* a po nej *RESET*. Poslednou vykonanou inštrukciou bol *RESET*, preto sú výstupy nastavené na FALSE.



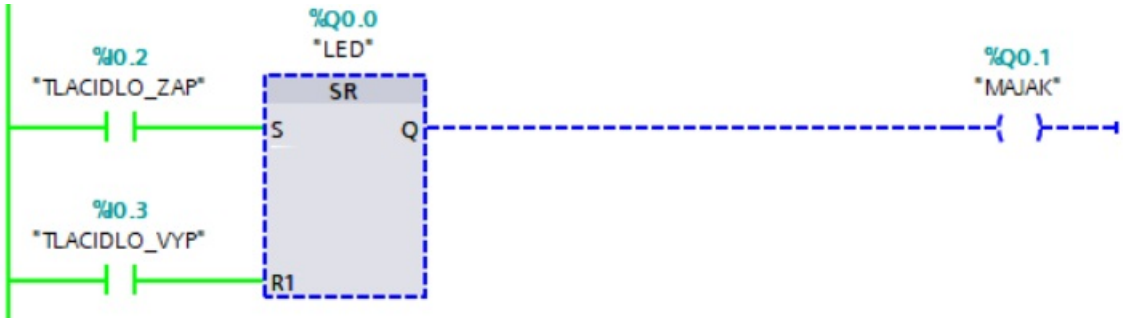
Obr. 3.57. 1. fáza animácie ovládania výstupov LED a MAJAK



Obr. 3.58. 2. fáza animácie ovládania výstupov LED a MAJAK



Obr. 3.59. 3. fáza animácie ovládania výstupov LED a MAJAK



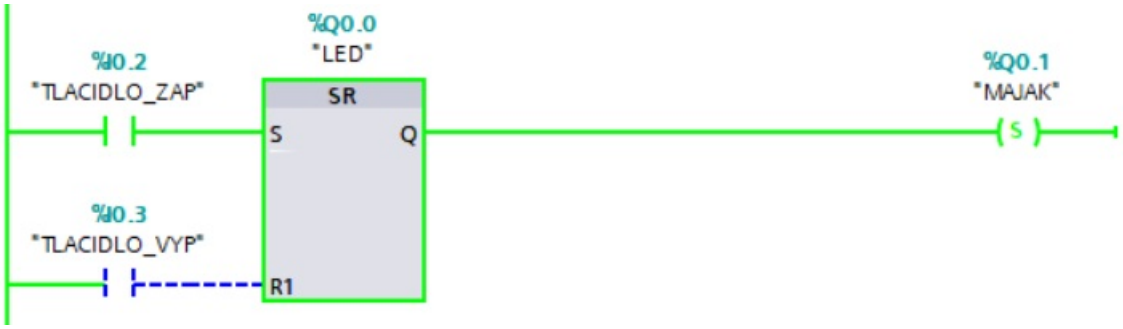
Obr. 3.60. 4. fáza animácie ovládania výstupov LED a MAJAK

Modifikovaný program s jeho animáciou je na Obr. 3.61 až 3.63:

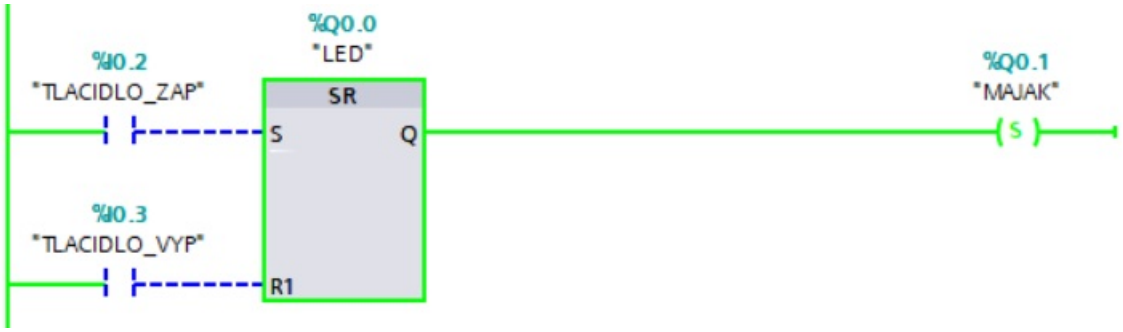
1. Na Obr. 3.61 je aktivované tlačidlo TLACIDLO_ZAP. Vykoná sa inštrukcia *SET*, preto hodnota operandu LED a výstup Q majú hodnotu TRUE. Zároveň sa vykonala aj inštrukcia *SET* pripojená na výstup Q inštrukcie *SET-RESET*. Hodnota výstupu MAJAK je TRUE.

2. V ďalšom kroku sa tlačidlo TLACIDLO_ZAP uvoľnilo. Hodnoty LED a MAJAK sú nezmenené (Obr. 3.62).

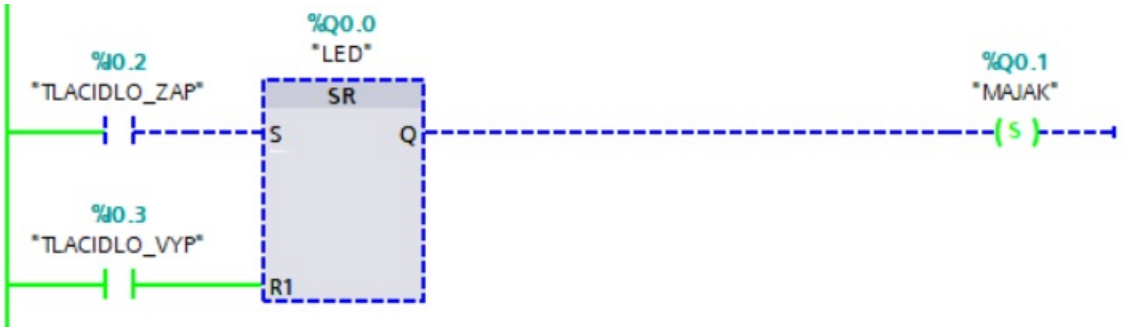
3. V poslednom kroku sa aktivoval vstup TLACIDLO_VYP (Obr. 3.63). Hodnota LED a výstupu Q sa zmenili na FALSE, ale hodnota výstupu MAJAK je stále TRUE. Na jeho zmenu na hodnotu FALSE by sme potrebovali inštrukciu *RESET*.



Obr. 3.61. 1. fáza animácie ovládania výstupov LED a MAJAK - modifikovaný program



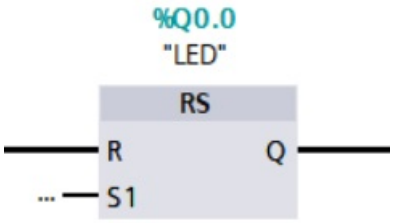
Obr. 3.62. 2. fáza animácie ovládania výstupov LED a MAJAK - modifikovaný program



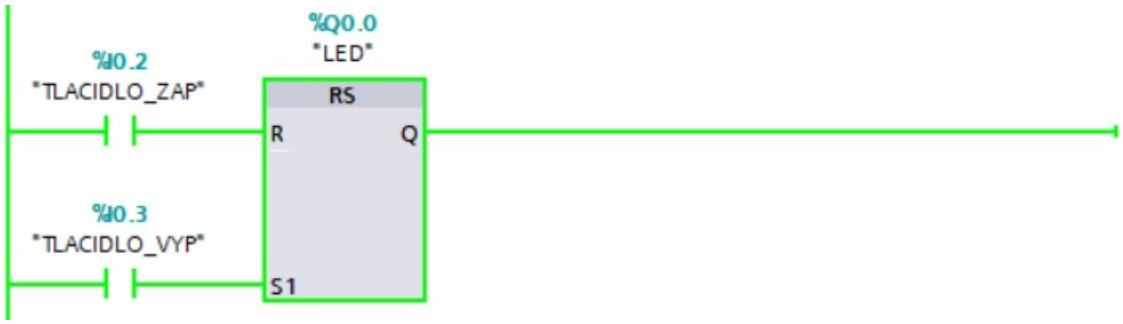
Obr. 3.63. 3. fáza animácie ovládania výstupov LED a MAJAK - modifikovaný program

3.2.9 RESET-SET

Symbol inštrukcie *RESET-SET* je na Obr. 3.64. Ide sa o kombináciu inštrukcií *SET* a *RESET*. Vstup *SET* je dominantný, lebo sa vykoná ako posledný pri aktivácii oboch vstupov inštrukcie. Operand sa zadáva nad symbolom inštrukcie. Výstup Q kopíruje stav operandu a ovplyvňuje stav RLO za inštrukciou. Príklad aktivácie oboch vstupov inštrukcie je na Obr. 3.65.



Obr. 3.64. Symbol inštrukcie *RESET-SET*



Obr. 3.65. Príklad použitia inštrukcie *RESET-SET*

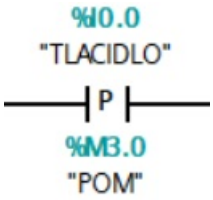
3.2.10 Nábežná hrana operandu

Inštrukcie detekcie nábežnej a dobežnej hrany detegujú zmenu operandu zo stavu FALSE na TRUE resp. TRUE na FALSE. Ich výstup je TRUE počas jedného cyklu PLC, keď operand inštrukcie zmenil svoju hodnotu. Používajú sa na jednorazové vykonanie inštrukcie alebo skupiny inštrukcií.

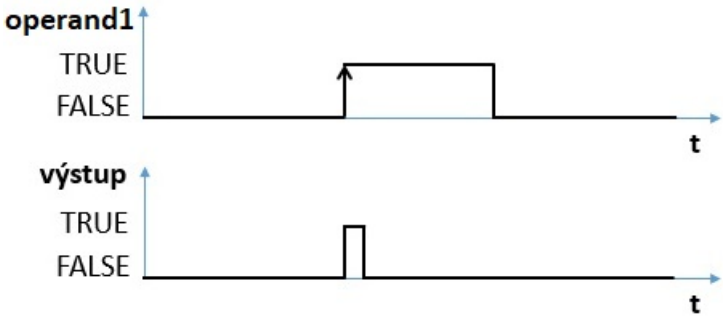
Symbol inštrukcie *nábežná hrana operandu* je na Obr. 3.66. Nad symbol inštrukcie sa zadáva operand, ktorého zmenu chceme sledovať. Pod inštrukciu je zadávaný pomocný operand (pamäťový bit alebo bit z dátového bloku). Ten slúži na porovnanie aktuálnej hodnoty prvého operandu (nad inštrukciou) so starou hodnotou prvého operandu (pod inštrukciou) z predchádzajúceho cyklu. Hodnota tejto pomocnej premennej je menená samotnou inštrukciou. Inštrukcia *nábežná hrana operandu* má na výstupe hodnotu TRUE ak došlo k zmene operandu z hodnoty FALSE na TRUE. V opačnom prípade je výstupom inštrukcie FALSE (Obr. 3.67). Inak povedané, ak operand1 (horný) má hodnotu TRUE a operand2 (dolný) má hodnotu FALSE, výstupom inštrukcie je TRUE, inak FALSE. Písmeno P v symbole inštrukcie znamená Positive (kladnú - nábežnú hranu).

Inštrukcia číta, ale aj zapisuje hodnotu druhého operandu. Táto premenná sa nesmie nikde používať (zapisovať) lebo inštrukcia *nábežná hrana operandu* nebude fungovať podľa očakávaní.

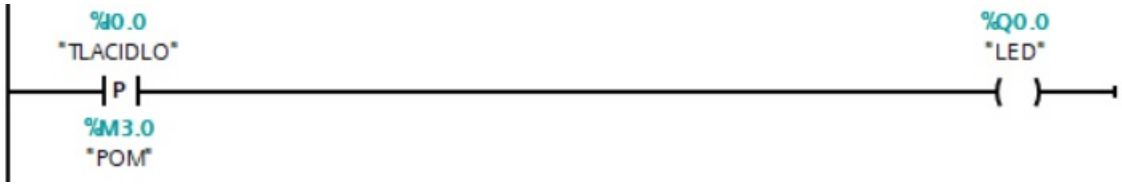
Príklad detekcie nábežnej hrany vstupu TLACIDLO je na Obr. 3.68. V momente zatlačenia tlačidla (ak uvažujeme tlačidlo ako spínací kontakt pripojený k digitálnemu vstupnému modulu) sa výstup LED nastaví na TRUE na jeden cyklus. Ekvivalentný program bez inštrukcie *nábežná hrana operandu* je na Obr. 3.69. Ďalšie príklady inštrukcie sú uvedené v kap. 3.4.1 Sčítanie (ADD).



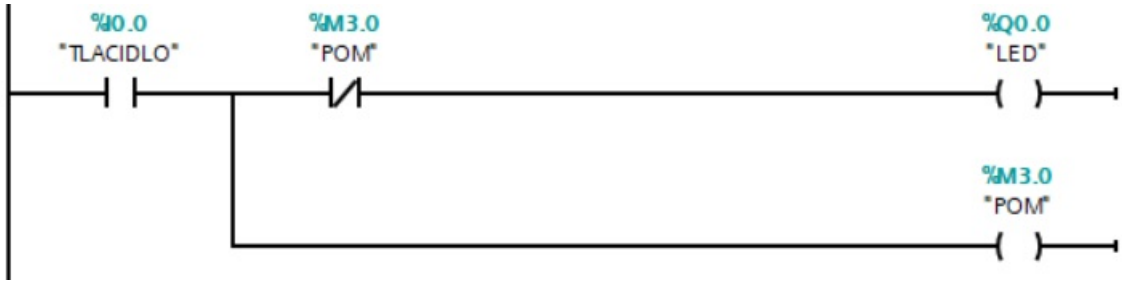
Obr. 3.66. Symbol inštrukcie *nábežná hrana operandu*



Obr. 3.67. Časový priebeh signálov inštrukcie *nábežná hrana operandu*



Obr. 3.68. Príklad použitia inštrukcie *nábežná hrana operandu*

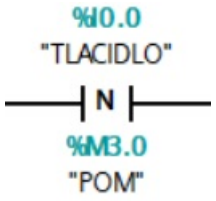


Obr. 3.69. Ekvivalentný program detekcie nábežnej hrany

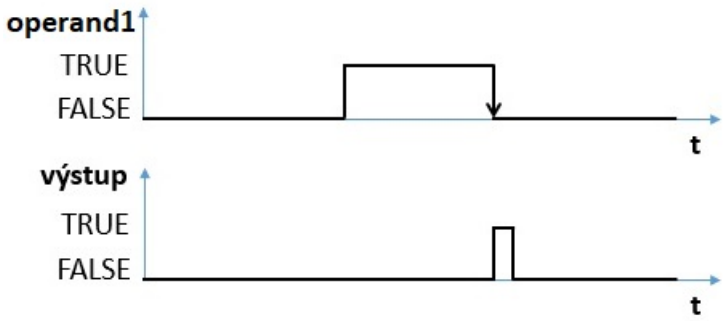
3.2.11 Dobežná hrana operandu

Symbol inštrukcie *dobežná hrana operandu* je na Obr. 3.70. Nad inštrukciu sa podobne ako pri inštrukcii *nábežná hrana operandu* zadáva operand, ktorého zmenu stavu chceme detegovať. Pod inštrukciu sa zadáva pomocný bitový operand, ktorý nesmie byť nikde inde prepisovaný. Inštrukcia má na výstupe hodnotu TRUE, ak horný operand zmení svoj stav z TRUE na FALSE. V opačnom prípade je výstupom hodnota FALSE (Obr. 3.71). Inak povedané, ak operand1 (horný) má hodnotu FALSE a operand2 (dolný) má hodnotu TRUE, výstupom inštrukcie je TRUE, v opačnom prípade FALSE. Písmeno N v symbole inštrukcie znamená Negative (záporná - dobežnú hranu).

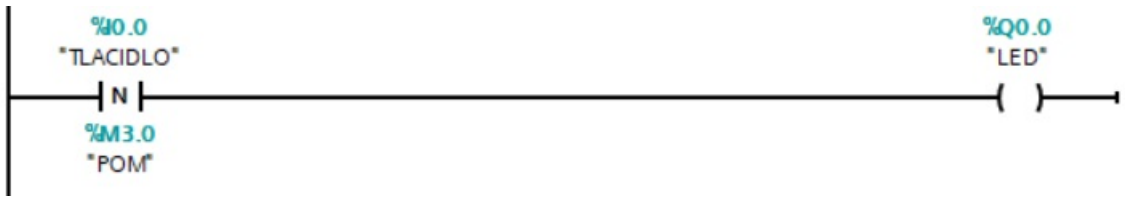
Príklad detekcie dobežnej hrany vstupu TLACIDLO je na Obr. 3.72. V momente uvoľnenia zatlačeného tlačidla (zmeny hodnoty z TRUE na FALSE) sa LED zapíše na jeden cyklus na TRUE. Keďže cyklus PLC je veľmi rýchly, hodnotu TRUE trvajúcu len jeden cyklus by sme voľným okom nepostrehli. Ekvivalentný program bez inštrukcie *dobežná hrana operandu* je na Obr. **??**. Príklad inštrukcie je uvedený v kap. 3.4.1 Sčítanie (ADD).



Obr. 3.70. Symbol inštrukcie *dobežná hrana operandu*



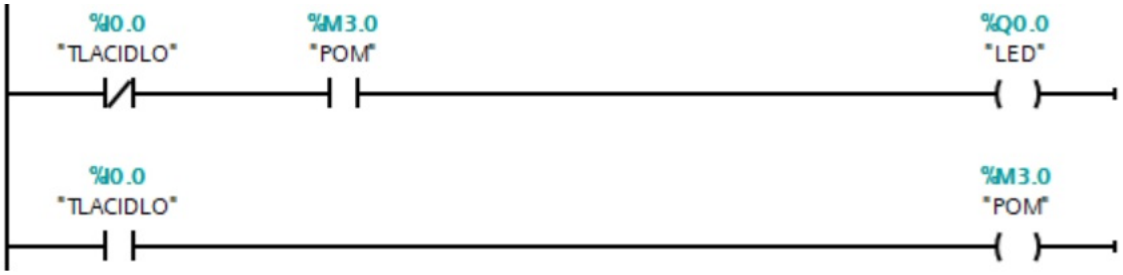
Obr. 3.71. Časový priebeh signálov inštrukcie *dobežná hrana operandu*



Obr. 3.72. Príklad použitia inštrukcie *dobežná hrana operandu*

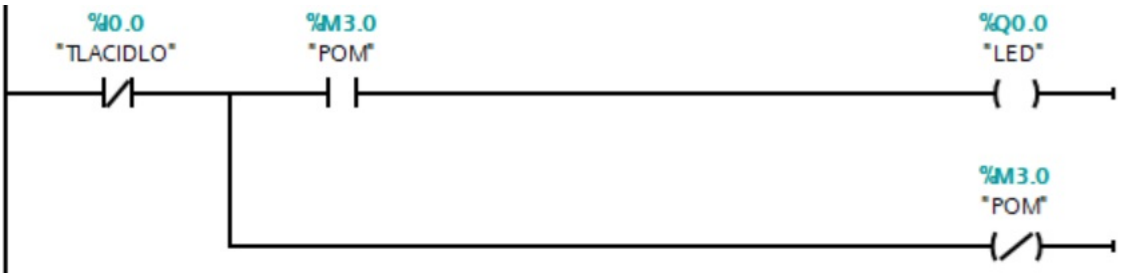
Výstup LED bude mať hodnotu TRUE, ak má vstup TLACIDLO stav FALSE (t. j. nie je stlačené) a premenná POM je TRUE. To znamená, že v predchádzajúcom cykle

bolo tlačidlo stlačené (lebo POM=TRUE) a aktuálne je uvoľnené (TLACIDLO=FALSE). V druhej vetve sa aktuálny stav vstupu TLACIDLO uloží do premennej POM na vyhodnotenie v ďalšom cykle.



Obr. 3.73. Prvý ekvivalentný program detekcie dobežnej hrany

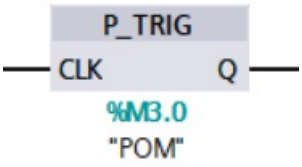
Podobným spôsobom je riešený aj druhý ekvivalentný príklad z hľadiska funkcionality. Rozdiel spočíva v spôsobe „uloženia“ stavu TLACIDLO do premennej POM. Používame vetvenie, avšak zápis do premennej POM prebieha pomocou negovaného priradenia, aby sa hodnota POM zhodovala s hodnotou TLACIDLO. Ak má tlačidlo stav TRUE, výstupom prvej inštrukcie je FALSE. Táto hodnota sa v spodnej vetve privedie k inštrukcii negovaného priradenia, a preto sa do premennej POM uloží opačný signál, teda TRUE. Naopak, ak má tlačidlo stav FALSE, výstup prvej inštrukcie je TRUE. Tento stav sa privedie k premennej POM, kde sa pomocou negovaného priradenia uloží opačný stav RLO, teda FALSE.



Obr. 3.74. Druhý ekvivalentný program detekcie dobežnej hrany

3.2.12 Nábežná hrana P_TRIG

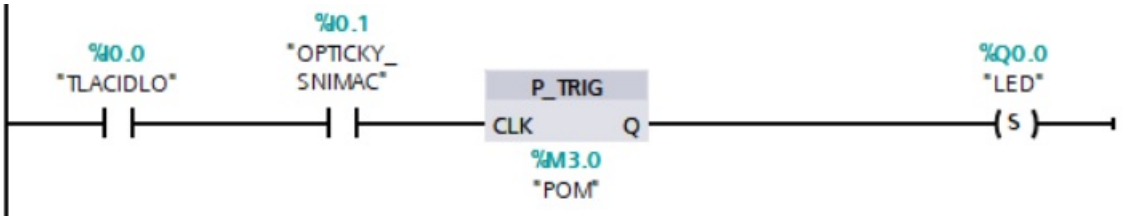
Symbol inštrukcie *nábežná hrana P_ TRIG* je na Obr. 3.75. Na rozdiel od inštrukcie *nábežná hrana operandu*, táto inštrukcia deteguje nábežnú hranu toku signálu RLO pripojeného na jeho vstup CLK. Pod inštrukciu sa zadáva pomocný bit, ktorý nesmie byť nikde inde zapisovaný. Inštrukcia si do daného bitu uchováva hodnotu vstupu pre vyhodnotenie nábežnej hrany v ďalšom cykle. Výsledok detekcie je indikovaný výstupom Q. Ak sa CLK zmení z hodnoty FALSE na TRUE, výstup Q je TRUE, inak FALSE.



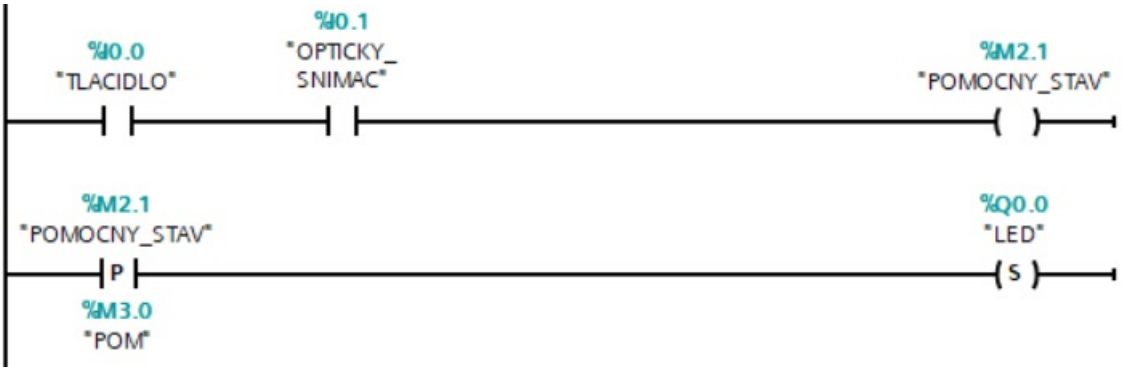
Obr. 3.75. Symbol inštrukcie *nábežná hrana P_ TRIG*

Príklad použitia inštrukcie je na Obr. 3.76. Na vstup CLK je privedený logický AND výstupu dvoch inštrukcií *spínací kontakt*. Ak podmienka neplatila a začne platiť (TLACIDLO a OPTICKY_SNIMAC nadobudnú hodnotu TRUE), na vstupe CLK dôjde k zmene stavu z FALSE na TRUE. Na jeden cyklus sa aktivuje výstup Q na TRUE. V tomto cykle sa zároveň vykoná inštrukcia *SET* pripojená za výstup Q.

Ekvivalentný program pomocou inštrukcie *nábežná hrana operandu* je na Obr. 3.77. Výsledok logickej operácie AND sa musí uložiť do pomocnej premennej (POMOCNY_STAV). Tá bude prvým operandom detekcie nábežnej hrany.



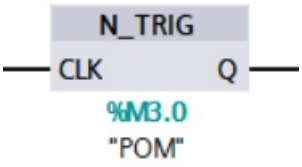
Obr. 3.76. Príklad použitia inštrukcie *nábežná hrana P_ TRIG*



Obr. 3.77. Ekvivalentný program detekcie nábežnej hrany

3.2.13 Dobežná hrana N_TRIG

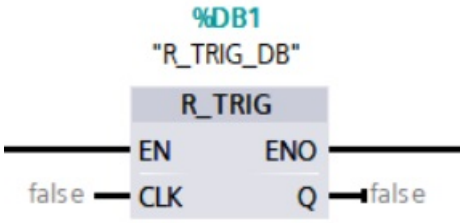
Symbol inštrukcie *dobežná hrana N_ TRIG* je na Obr. 3.78. Parametrizuje sa analogicky ako inštrukcia *nábežná hrana P_ TRIG*. Rozdielom je detekcia dobežnej hrany stavu RLO na vstupe CLK. Ak sa zmení stav vstupu CLK z hodnoty TRUE na FALSE, výstup Q sa nastaví na TRUE, inak na FALSE.



Obr. 3.78. Symbol inštrukcie *dobežná hrana N_ TRIG*

3.2.14 Nábežná hrana R_TRIG

Doteraz opísané bitové inštrukcie patrili do skupiny štandardných funkcií. Nasledujúce dve inštrukcie sú súčasťou štandardných funkčných blokov. Symbol inštrukcie *nábežná hrana R_ TRIG* je na Obr. 3.79. Podobne ako inštrukcia *nábežná hrana P_ TRIG* deteguje nábežnú hranu na vstupe CLK. Výsledok detekcie sa zapisuje na výstup Q. Rozdielom je absencia nutnosti definovania pomocného bitu. Ten sa uchováva v inštančnom dátovom bloku, ktorý sa zadáva nad symbol inštrukcie. Uprednostnením tejto inštrukcie pred inštrukciou *nábežná hrana P_ TRIG* sa vyhneme prípadnej zlej voľbe pomocného bitu. Štruktúra inštančného dátového bloku inštrukcie je na Obr. 3.80. Spomínaný pomocný bit (Stat_Bit) je v zozname statických premenných. Vstup EN (Enable Input) slúži na povolenie vykonania inštrukcie. Väčšinou sa vykonávanie inštrukcie nezakazuje. Výstup ENO (Enable output) kopíruje hodnotu vstupu EN. K výstupu Q nie je možné pripojiť inštrukcie.



Obr. 3.79. Symbol inštrukcie *nábežná hrana R_ TRIG*

R_TRIG_DB

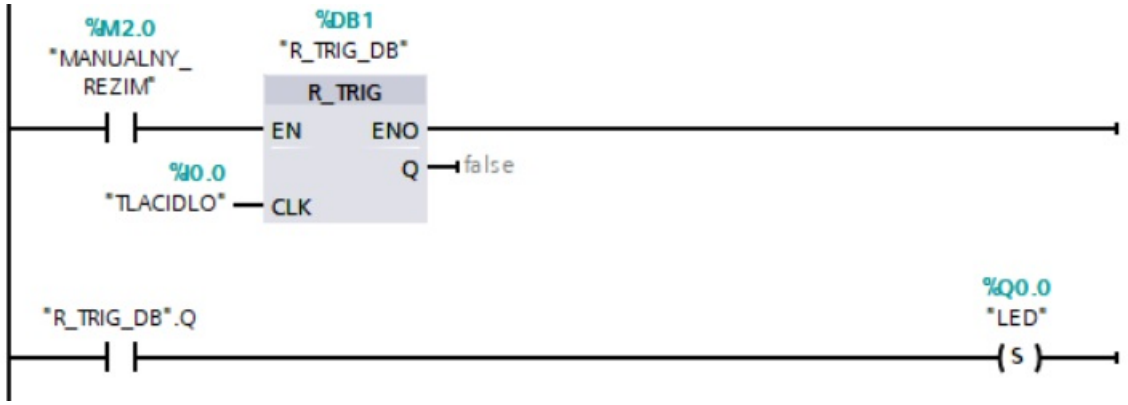
		Name	Data type	Start value	Retain
1		Input			
2		CLK	Bool	false	
3		Output			
4		Q	Bool	false	
5		InOut			
6		Static			
7		Stat_Bit	Bool	false	

Obr. 3.80. Štruktúra inštančného dátového bloku inštrukcie *nábežná hrana R_ TRIG*

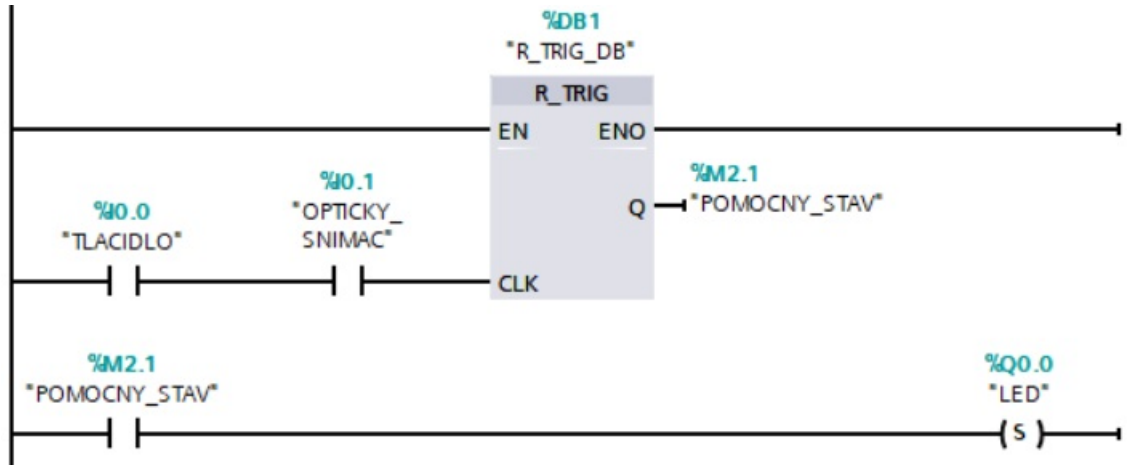
Na Obr. 3.81 je príklad použitia inštrukcie. Nábežná hrana sa deteguje nad vstupom TLACIDLO v prípade, že MANUALNY_REZIM je v stave TRUE. Ten povoľuje vykonanie inštrukcie. Ak je detegovaná nábežná hrana, vykoná sa inštrukcia *SET*. Nábežná

hrana zapísaná na výstup Q je teda v stave TRUE ak MANUALNY_REZIM je TRUE a TLACIDLO zmenilo stav z FALSE na TRUE.

Ďalší príklad je na Obr. 3.82. Na vstup CLK je privedený výsledok logickej operácie AND dvoch digitálnych vstupov. Detekcia hrany sa vykonáva v každom cykle (EN má na vstupe hodnotu TRUE pripojenú z ľavej strany rebríka). Výsledok detekcie Q je uložený do pomocnej premennej POMOCNY_STAV. Ten v novej vetve ovláda vykonanie inštrukcie *SET*.



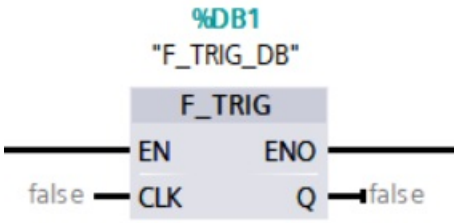
Obr. 3.81. 1. príklad použitia inštrukcie *nábežná hrana R_TRIG*



Obr. 3.82. 2. príklad použitia inštrukcie *nábežná hrana R_TRIG*

3.2.15 Dobežná hrana F_TRIG

Symbol inštrukcie *dobežná hrana F_TRIG* je na Obr. 3.83. Parametrizácia je analogická ako pri inštrukcii *nábežná hrana R_TRIG*. Inštrukcia deteguje dobežnú hranu na vstupe CLK.

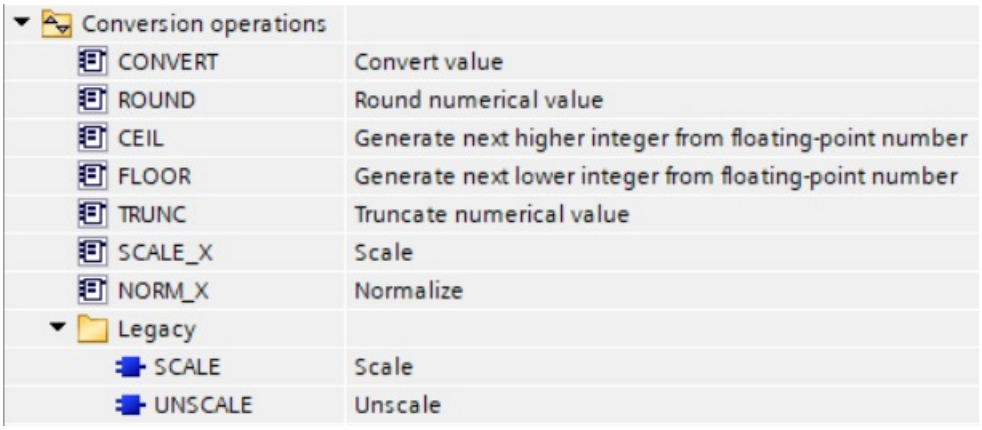


Obr. 3.83. Symbol inštrukcie *dobežná hrana F_ TRIG*

3.3 Konverzné inštrukcie (Conversion operations)

Zoznam inštrukcií konverzie v TIA Portal pre CPU S7-1200 a S7-1500 je na Obr. 3.84. Inštrukcie sú vykonané, ak ich vstup EN, je v stave TRUE. Inštrukcie z tejto skupiny sa používajú na

- Konverziu hodnoty premennej z jedného dátového typu na hodnotu iného dátového typu (CONVERT).
- Rôzne formy zaokrúhľovania hodnoty premennej (ROUND, CEIL, FLOOR a TRUNC).
- Lineárnu transformáciu hodnoty (SCALE_X, NORM_X, SCALE a UNSCALE)



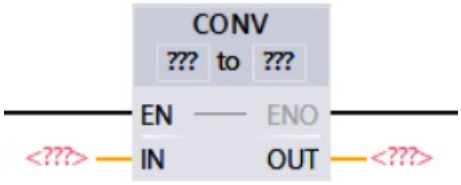
Obr. 3.84. Zoznam inštrukcií konverzie

3.3.1 Konverzia dátových typov (CONV)

Pri matematických operáciach (pozri 3.4 Matematické inštrukcie (Math functions)) často vstupujú do výpočtov premenné rôznych dátových typov. Aby nedošlo k strate presnosti a prípadným hazardom je potrebné dbať na konverziu hodnôt z jedného dátového typu na iný dátový typ. Ak nie je procesorom podporovaná automatická konverzia, je k dispozícii inštrukcia *CONV*. Symbol inštrukcie *CONV* je na Obr. 3.85. Inštrukcia načíta obsah parametra IN a skonvertuje ho podľa dátových typov zvolených v hornej časti inštrukcie. Prevedená hodnota je zapísaná na výstup OUT. Výstup ENO je v stave FALSE ak

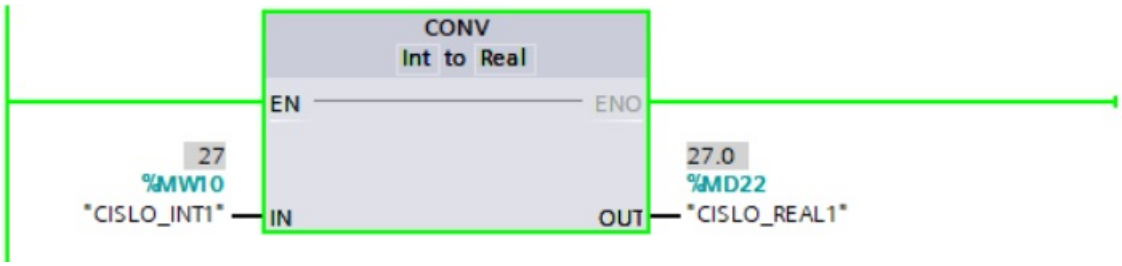
- Vstup EN je v stave FALSE.

- Počas vykonávania sa vyskytujú chyby, ako napríklad pretečenie.



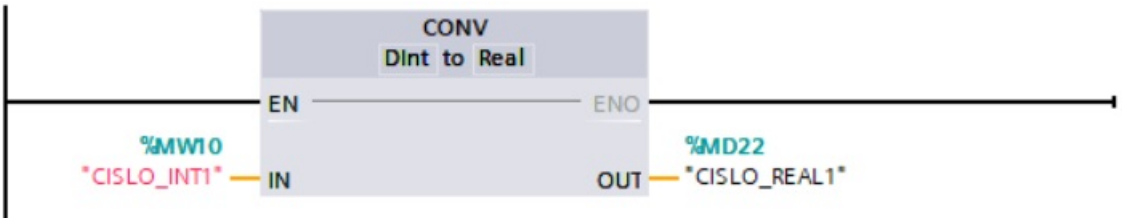
Obr. 3.85. Symbol inštrukcie *CONV*

Na Obr. 3.86 je prezentovaný príklad konverzie celočíselnej hodnoty 27 uloženej v premennej CISLO_INT1 dátového typu INT na dátový typ REAL (na hodnotu 27.0). Dátové typy sú uvedené pod názvom symbolu inštrukcie. Výstup konverzie (OUT) je priradený do premennej CISLO_REAL1 dátového typu REAL.



Obr. 3.86. 1. príklad použitia inštrukcie *CONV*

Na Obr. 3.87 je ukážka chybnej parametrizácie. Na vstup IN je pripojená premenná CISLO_INT1 dátového typu INT, ale očakávaný vstup by mal byť dátového typu DINT (pozri DInt pod názvom inštrukcie). Chyba je indikovaná červenou farbou názvu premennej. Program v takomto stave nie je možné skompilovať a nahrať do CPU.



Obr. 3.87. 2. príklad použitia inštrukcie *CONV*

V ďalšom príklade (Obr. 3.88) sa konvertuje hodnota premennej z dátového typu DINT na INT. Spodných 16 bitov premennej CISLO_DINT1 sa zapíše na výstup OUT. Horných 16 bitov sa počas tejto konverzie stráca. Hodnota premennej je schválne väčšia ako maximálna hodnota dátového typu INT. Hodnoty premenných v binárnej aj decimálnej podobe sú na Obr. 3.89.

Viac príkladov automatickej a manuálnej konverzie typov je v kap. 3.4.1 Sčítanie (ADD).

Obr. 3.88. 3. príklad použitia inštrukcie *CONV*

Name	Address	Display format	Monitor value
"CISLO_DINT1"	%MD14	Bin	2#0000_0000_0000_0001_0000_0000_0000_0011
"CISLO_INT1"	%MW10	Bin	2#0000_0000_0000_0011
"CISLO_DINT1"	%MD14	DEC+/-	65539
"CISLO_INT1"	%MW10	DEC+/-	3

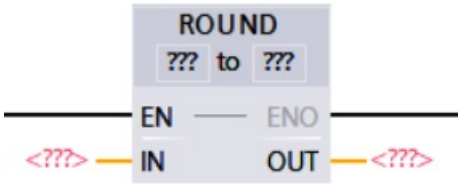
Obr. 3.89. Hodnoty premenných v 3. príklade

3.3.2 Zaokrúhlenie (ROUND)

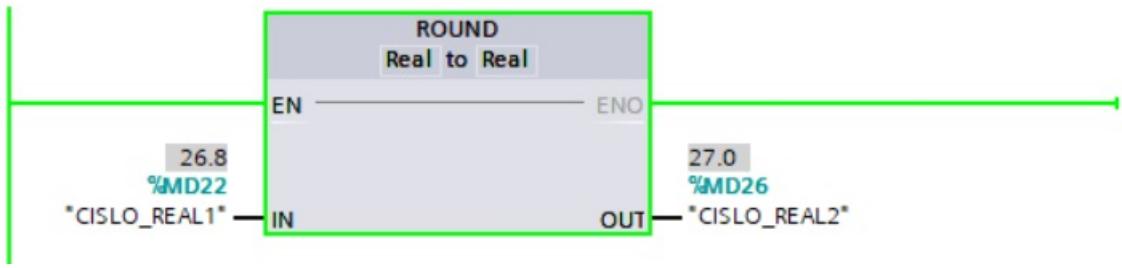
Symbol inštrukcie *ROUND* je na Obr. 3.90. Inštrukcia zaokrúhľuje hodnotu premennej z dátového typu REAL alebo LREAL na hodnotu zvoleného dátového typu. Zaokrúhľuje sa na najbližšie celé číslo.

Na Obr. 3.91 je príklad zaokrúhlenia hodnoty 26,8 uloženej v premennej CISLO_REAL1. Výsledok je uložený ako hodnota (27,0) dátového typu REAL do premennej CISLO _REAL2.

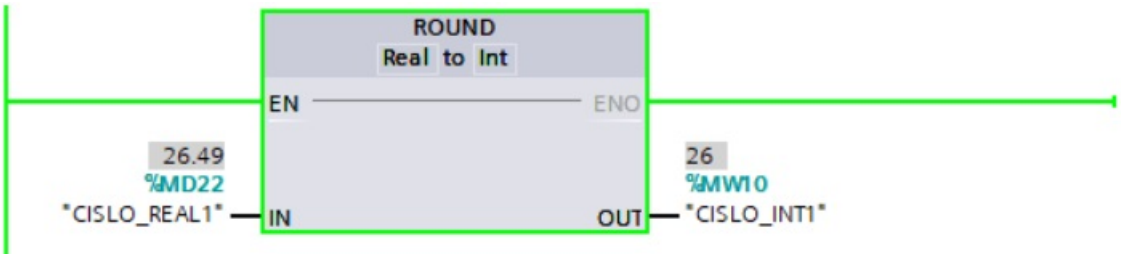
V druhom príklade (Obr. 3.92) je hodnota 26,49 zaokrúhlená smerom nadol a uložená ako celočíselná hodnota (26) dátového typu INT.



Obr. 3.90. Symbol inštrukcie *ROUND*



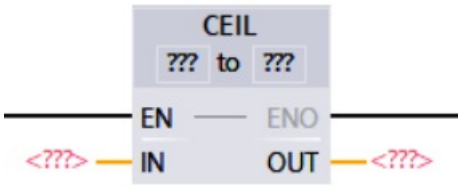
Obr. 3.91. 1. príklad použitia inštrukcie *ROUND*



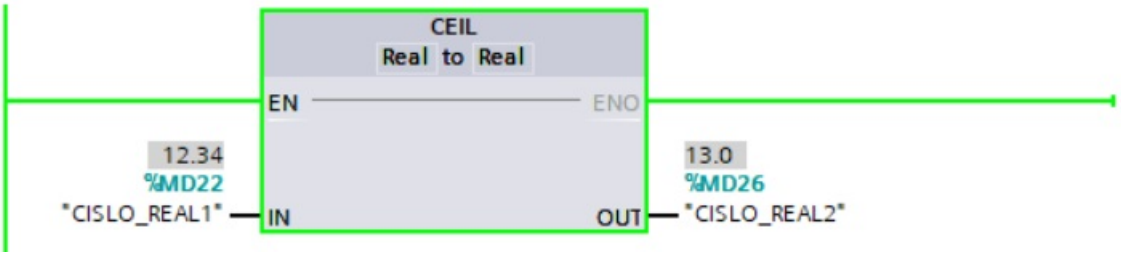
Obr. 3.92. 2. príklad použitia inštrukcie *ROUND*

3.3.3 Zaokrúhlenie smerom hore (CEIL)

Symbol inštrukcie *CEIL* je na Obr. 3.93. Inštrukcia zaokrúhľuje hodnotu pripojenú na vstup IN na najbližšie vyššie celé číslo. Príklad zaokrúhlenia hodnoty 12,34 na hodnotu 13,0 (dátového typu REAL) je na Obr. 3.94.



Obr. 3.93. Symbol inštrukcie *CEIL*



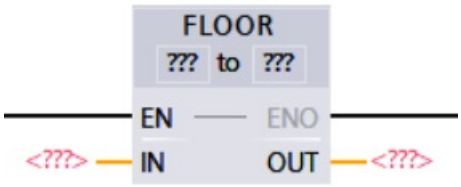
Obr. 3.94. Príklad použitia inštrukcie *CEIL*

3.3.4 Zaokrúhlenie smerom dole (FLOOR)

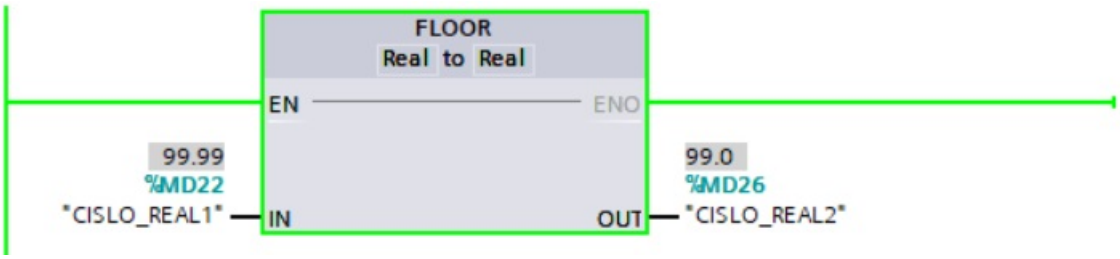
Symbol inštrukcie *FLOOR* je na Obr. 3.95. Inštrukcia zaokrúhľuje hodnotu pripojenú na vstup IN na najbližšie nižšie celé číslo.

Príklad zaokrúhlenia hodnoty 99,99 na hodnotu 99,0 (dátového typu REAL) je na Obr. 3.96.

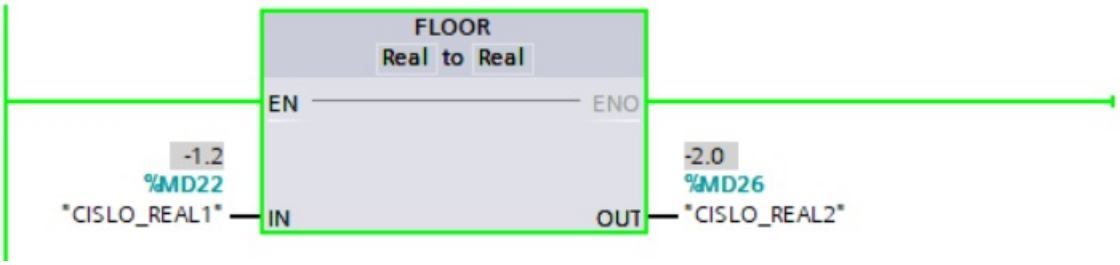
V druhom príklade (Obr. 3.97) sa zaokrúhľuje hodnota -1,2 smerom dole na najbližšie nižšie celé číslo dátového typu REAL. Výsledkom je hodnota -2,0.



Obr. 3.95. Symbol inštrukcie *FLOOR*



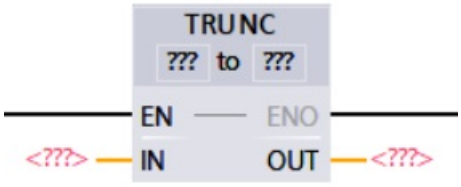
Obr. 3.96. 1. príklad použitia inštrukcie *FLOOR*



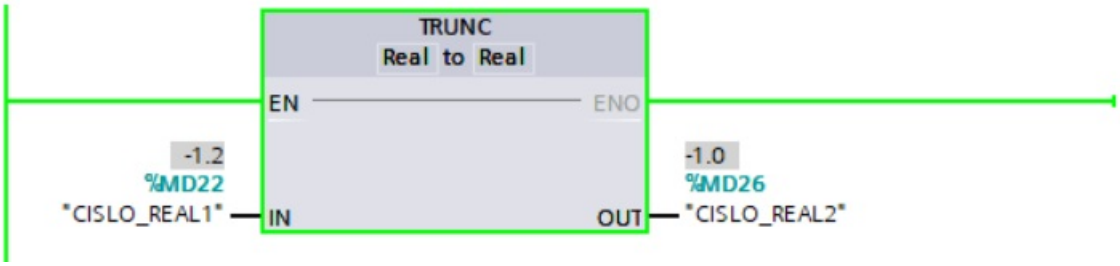
Obr. 3.97. 2. príklad použitia inštrukcie *FLOOR*

3.3.5 Celočíselná časť reálneho čísla (TRUNC)

Inštrukcia *TRUNC* vracia celočíselnú časť reálneho čísla. Symbol inštrukcie *TRUNC* je na Obr. 3.98. Príklad inštrukcie je na Obr. 3.99. Z hodnoty -1,2 pripojenej na vstup IN sa na výstup OUT vyberie celočíselná časť hodnoty, t. j. -1. Hodnota je spracovaná podľa zvoleného dátového typu. V príklade ide o dátový typ REAL, preto je výstupom hodnota -1,0.



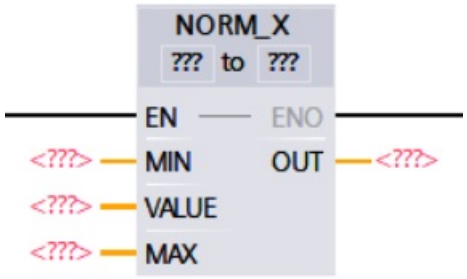
Obr. 3.98. Symbol inštrukcie *TRUNC*



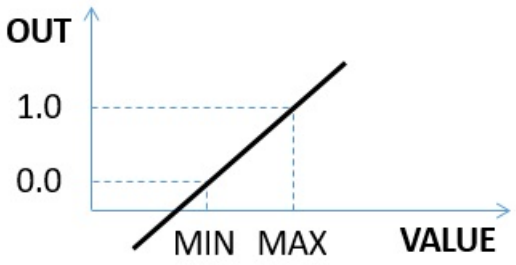
Obr. 3.99. Príklad použitia inštrukcie *TRUNC*

3.3.6 Normovanie (NORM_X)

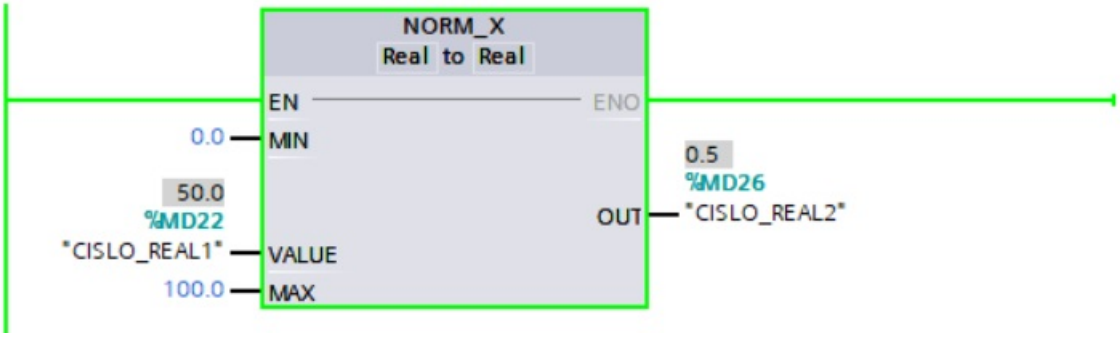
Funkcia *NORM_X* (Obr. 3.100) realizuje lineárnu transformáciu (Obr. 3.101) hodnoty VALUE z rozsahu MIN - MAX do rozsahu 0,0 - 1,0. Vzťah výpočtu výstupu je OUT = (VALUE – MIN) / (MAX – MIN). Výstup OUT nie je obmedzený. Inštrukcia sa môže s výhodou použiť v kombinácii s inštrukciou *SCALE_X* pri spracovaní analógových signálov. Príklad lineárneho prevodu je na Obr. 3.102. Hodnota 50,0 z rozsahu MIN = 0,0 až MAX = 100,0 sa lineárne transformuje do rozsahu 0,0 - 1,0. Výsledkom je hodnota 0,5. Ak by bola na vstupe hodnota 110,0, výstupom by bola hodnota 1,1.



Obr. 3.100. Symbol inštrukcie *NORM_X*



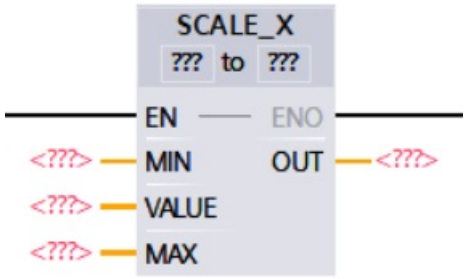
Obr. 3.101. Lineárny prevod inštrukciou *NORM_X*



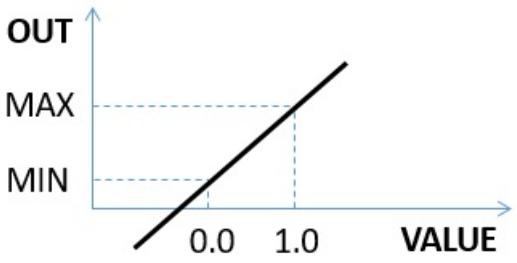
Obr. 3.102. Príklad použitia inštrukcie *NORM_X*

3.3.7 Škálovanie (SCALE_X)

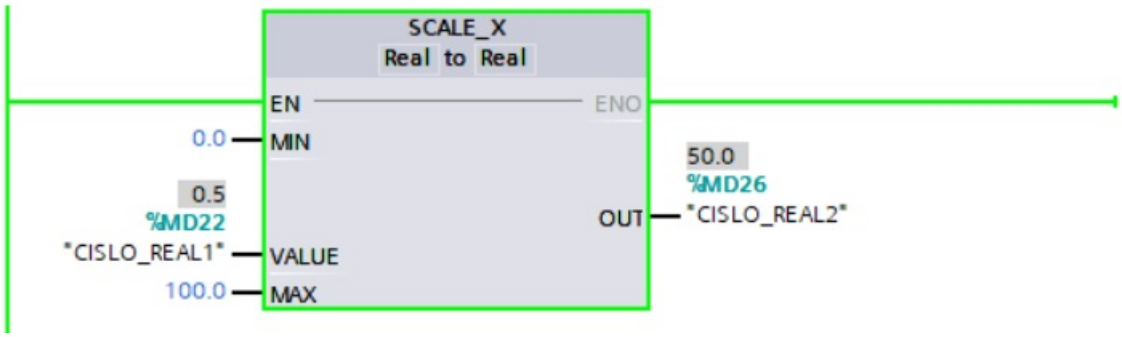
Inštrukcia *SCALE_X* (Obr. 3.103) realizuje opačnú lineárnu transformáciu (Obr. 3.104) ako inštrukcia *NORM_X*. Hodnota VALUE je transformovaná z rozsahu 0,0 - 1,0 do zadaného rozsahu MIN - MAX. Vzťah výpočtu výstupu je OUT = [VALUE * (MAX - MIN)] + MIN. Výstup OUT nie je obmedzený. Príklad lineárneho prevodu je na Obr. 3.105. Hodnota 0,5 z rozsahu 0,0 a 1,0 sa lineárne transformuje do rozsahu MIN = 0,0 až MAX = 100,0. Výsledkom je hodnota 50,0.



Obr. 3.103. Symbol inštrukcie *SCALE_X*



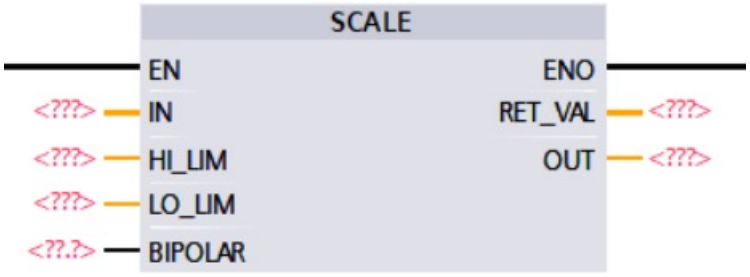
Obr. 3.104. Lineárny prevod inštrukciou *SCALE_X*



Obr. 3.105. Príklad použitia inštrukcie *SCALE_X*

3.3.8 Škálovanie z rozsahu analógových vstupov (SCALE)

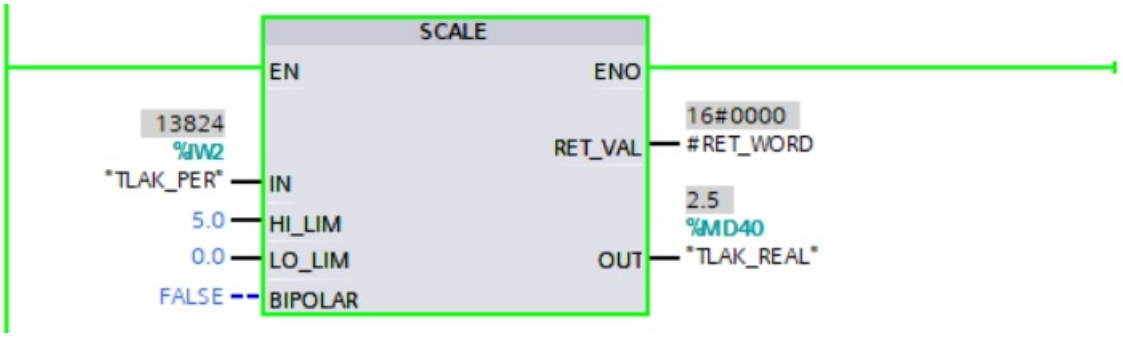
Inštrukcia *SCALE* nie je dostupná pre rodinu PLC S7-1200. Funkcia vykonáva lineárny prevod z rozsahu analógového vstupného modulu na inžinierske jednotky. Symbol inštrukcie je na Obr. 3.106. Hodnota v rozsahu 0 až 27 648 alebo v rozsahu -27 648 až 27 648 sa privádza na vstup IN. Vstup BIPOLAR rozhoduje o tom, ktorý rozsah sa má použiť vo výpočtoch. Ak má hodnotu FALSE, ide o unipolárny signál pripojený k analógovému vstupnému modulu (napr. 0 - 10V) v rozsahu 0 až 27 648. Ak má hodnotu TRUE, ide o bipolárny signál (napr. +- 10V) v rozsahu -27 648 až 27 648. Hodnota IN sa lineárnou transformáciou prevádza do rozsahu LO_LIM až HI_LIM. Hodnota v inžinierskych jednotách sa zapisuje na výstup OUT. Výstup RET_VAL dátového typu WORD indikuje prípadný výskyt chyby.



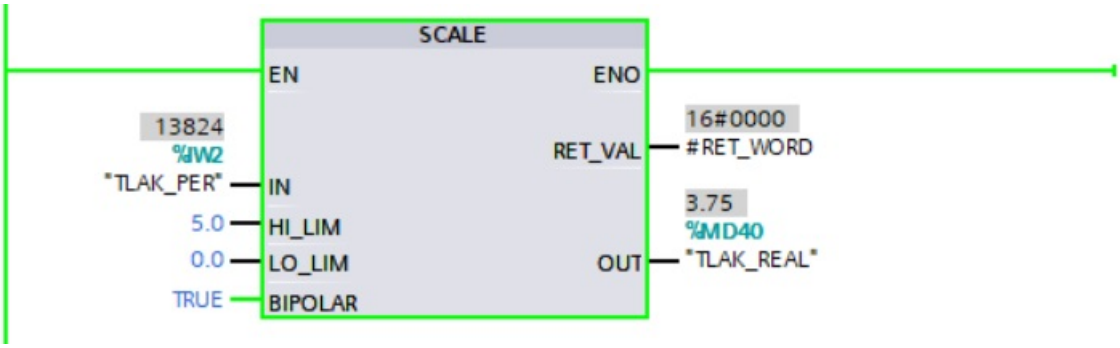
Obr. 3.106. Symbol inštrukcie *SCALE*

Prvý príklad použitia inštrukcie je na Obr. 3.107. Analógové meranie tlaku TLAK_PER s fyzickou adresou %IW2 je pripojené k vstupu IN. Podľa nastavenia vstupu BIPOLAR ide o unipolárne meranie v rozsahu 0 až 27 648. Aktuálna hodnota merania je 13 824 (polovica rozsahu). Hodnota merania sa má transformovať do rozsahu 0,0 až 5,0. Na výstupe je vypočítaná hodnota v inžinierskych jednotkách 2,5 (napr. barov).

Druhý príklad (Obr. 3.108) je modifikáciou prvého príkladu. V príklade je nastavený bipolárny signál. Rozsah merania je -27 648 až 27 648. Hodnota 13 824 predstavuje tri štvrtiny rozsahu, preto výsledkom sú tri štvrtiny z rozsahu 0,0 až 5,0, t. j. 3,75.



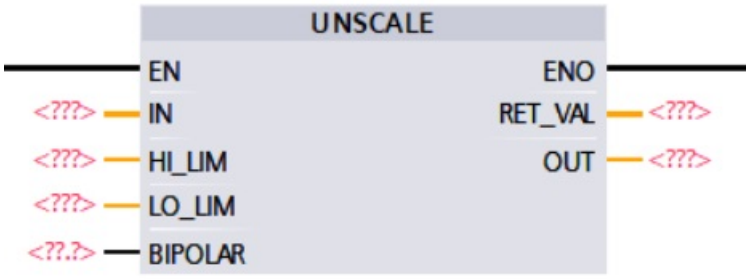
Obr. 3.107. 1. príklad použitia inštrukcie *SCALE*



Obr. 3.108. 2. príklad použitia inštrukcie *SCALE*

3.3.9 Škálovanie na rozsah analógových výstupov (UNSCALE)

Inštrukcia *UNSCALE* nie je dostupná pre rodinu PLC S7-1200. Inštrukcia *UNSCALE* prevádza desatinné číslo na rozsah analógového výstupného modulu. Symbol inštrukcie je na Obr. 3.109. Na vstup IN sa pripája hodnota, ktorá sa má previesť na rozsah analógového výstupného modulu. Vstup BIPOLAR a výstup RET_VAL majú rovnaký význam ako pri inštrukcii *SCALE*. Výstup OUT obsahuje transformovanú hodnotu v rozsahu analógového výstupného modulu (v rozsahu 0 až 27 648 alebo +- 27 648 v závislosti od nastavenia BIPOLAR). Vstupy LO_LIM a HI_LIM definujú rozsah premennej pripojenej na vstup IN z ktorého sa transformácia realizuje.

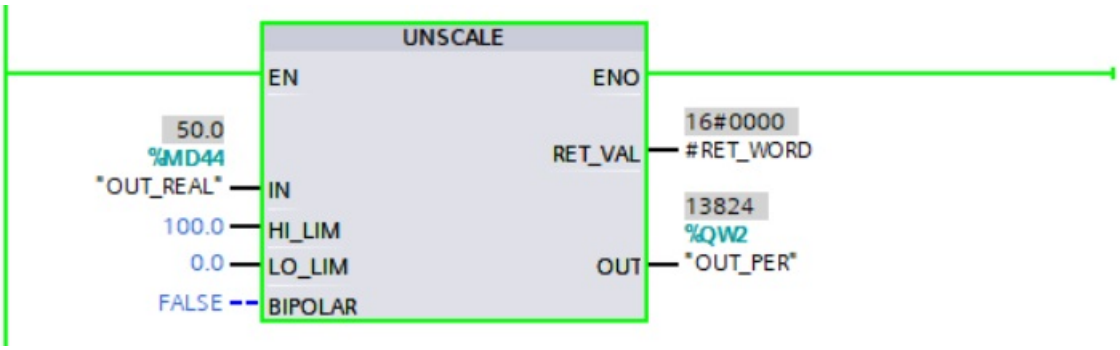


Obr. 3.109. Symbol inštrukcie *UNSCALE*

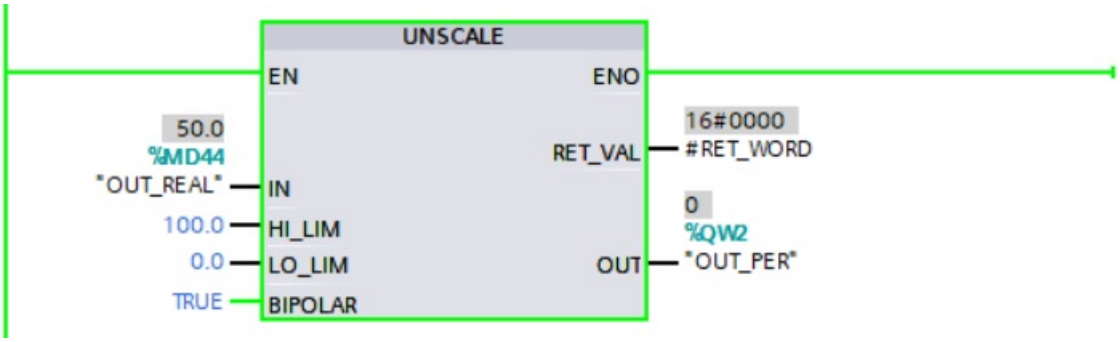
V prvom príklade (Obr. 3.110) sa transformuje akčná veličina OUT_REAL z rozsahu 0,0 až 100,0 % na rozsah analógového výstupného modulu OUT_PER. Rozsah výstupu je

unipolárny, preto sa hodnota 50,0 transformuje na hodnotu polovice rozsahu 0 - 27 648, t. j. 13 824.

V druhom príklade (Obr. 3.111) sa zmenil signál z unipolárneho na bipolárny. Rozsah výstupu je -27 648 až 27 648. Hodnota 50,0, ktorá reprezentuje polovicu rozsahu 0,0 až 100,0 je transformovaná na hodnotu polovice rozsahu modulu, t. j. 0.



Obr. 3.110. 1. príklad použitia inštrukcie *UNSCALE*



Obr. 3.111. 2. príklad použitia inštrukcie *UNSCALE*

3.4 Matematické inštrukcie (Math functions)

Často je potrebné v aplikáciách PLC realizovať matematické výpočty nad rôznymi dátami. Na to slúžia matematické inštrukcie, ktoré okrem základných bitových inštrukcií (so stavmi TRUE, FALSE) vykonávajú komplexnejšie operácie nad dátovými typmi ako napr. INT, UINT, USINT, DINT, UDINT, REAL. Jeden z dôvodov ich použitia je, že získané údaje z analógového vstupu nemusia byť v podobe akej to vyhovuje programátorovi, a preto je potrebné upraviť rozsah merania. Okrem toho sa môžu používať na počítanie udalostí, výrobkov, času ako aj na zložitejšie matematické výpočty, ktoré sa využívajú v oblasti riadenia alebo predikcii správania procesov a iné. [13] Zoznam matematických inštrukcií v TIA Portal je na Obr. 3.112. Všetky matematické inštrukcie sú vykonané, ak má ich vstup EN stav TRUE.

Math functions	
CALCULATE	Calculate
ADD	Add
SUB	Subtract
MUL	Multiply
DIV	Divide
MOD	Return remainder of division
NEG	Create twos complement
INC	Increment
DEC	Decrement
ABS	Form absolute value
MIN	Get minimum
MAX	Get maximum
LIMIT	Set limit value
SQR	Form square
SQRT	Form square root
LN	Form natural logarithm
EXP	Form exponential value
SIN	Form sine value
COS	Form cosine value
TAN	Form tangent value
ASIN	Form arcsine value
ACOS	Form arccosine value
ATAN	Form arctangent value
FRAC	Return fraction
EXPT	Exponentiate

Obr. 3.112. Zoznam matematických inštrukcií

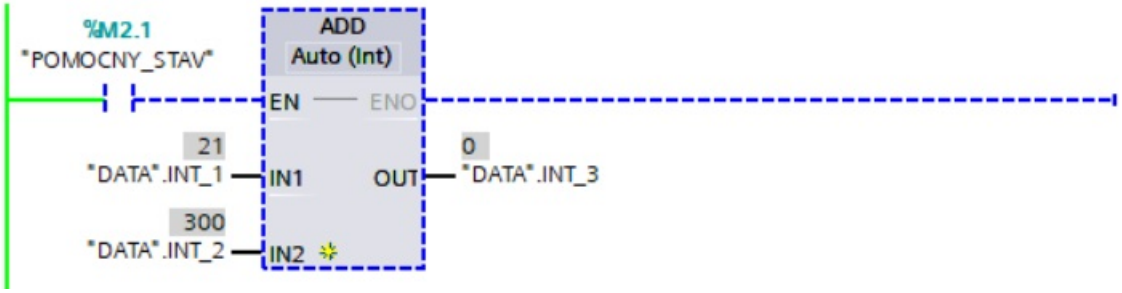
3.4.1 Sčítanie (ADD)

Inštrukcia sčítania (ADD - addition) sčíta vstupné operandy IN1 až INn a výsledok uloží do výstupu OUT. Základný symbol má predvolené dva vstupy IN1 a IN2 (Obr. 3.113 - symbol vľavo). Ďalšie vstupy sa pridajú cez menu kliknutím na posledný vstupný pin alebo kliknutím na žltú hviezdičku v spodnej časti symbolu inštrukcie. Príklad upraveného symbolu je na Obr. 3.113 - symbol vpravo. Pod názvom inštrukcie sa volí dátový typ. Po vložení prvého operandu sa automaticky nastaví podľa dátového typu premennej. Túto voľbu môžeme dodatočne zmeniť.

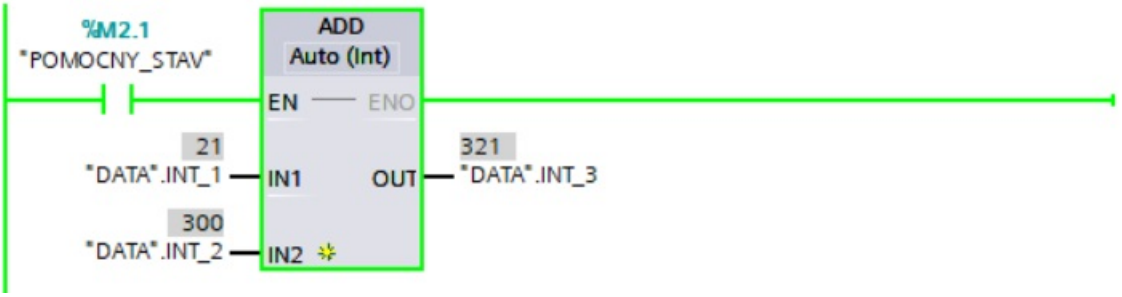
Obr. 3.113. Symbol inštrukcie *ADD*

Na Obr. 3.114 je na vstup EN inštrukcie *ADD* privedený signál FALSE. Inštrukcia sa nevykonáva. Hodnota premennej DATA.INT_3 sa nemení.

Na ďalšom Obr. 3.115 bol na vstup EN privedený stav TRUE. Inštrukcia sa vykonáva v každom cykle pokiaľ je EN aktivovaný. Do premennej DATA.INT_3 sa uložil súčet hodnôt premenných DATA.INT_1 a DATA.INT_2. Inštrukcia očakáva na vstupoch a výstupoch premenné dátových typov INT (viď INT pod názvom inštrukcie).



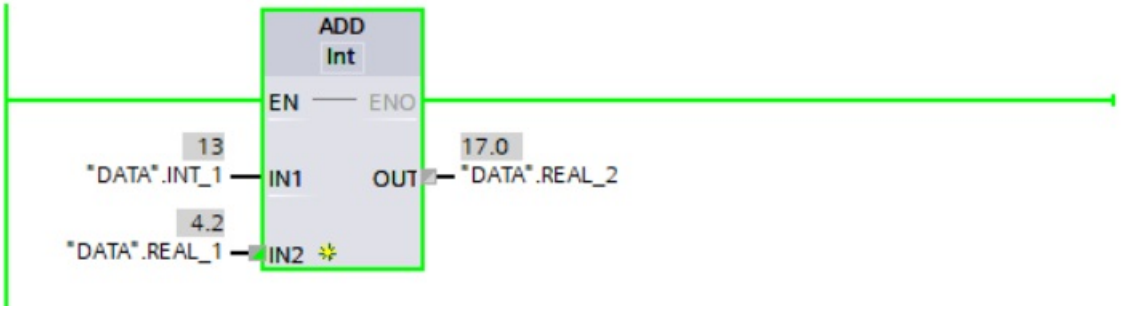
Obr. 3.114. 1. fáza monitorovania programu s inštrukciou *ADD*



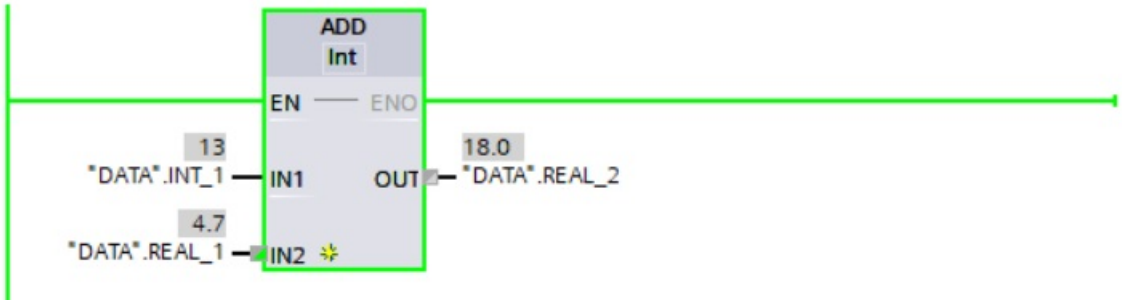
Obr. 3.115. 2. fáza monitorovania programu s inštrukciou *ADD*

Na Obr. 3.116 a 3.117 sú príklady súčtu dvoch operandov s automatickou konverziou dátových typov. Inštrukcia *ADD* má nastavený dátový typ INT. Premenné pripojené na vstup IN1 a výstup OUT vyhovujú danej voľbe. Premenná pripojená na vstup INT2 je dátového typu REAL. Nakoľko CPU podporuje automatickú konverziu dátových typov (štvorec na vstupnom pine IN2), hodnota je najprv konvertovaná z dátového typu REAL na INT a až potom sčítaná s hodnotou IN1. Konverzia dátového typu príslušného signálu je

vývojovým prostredím indikovaná prostredníctvom malého štvorčeka pri príslušnom vstupe/výstupe bloku. V prvom príklade sa hodnota 4,2 konvertuje na hodnotu 4 a v druhom príklade hodnota 4,7 na hodnotu 5.

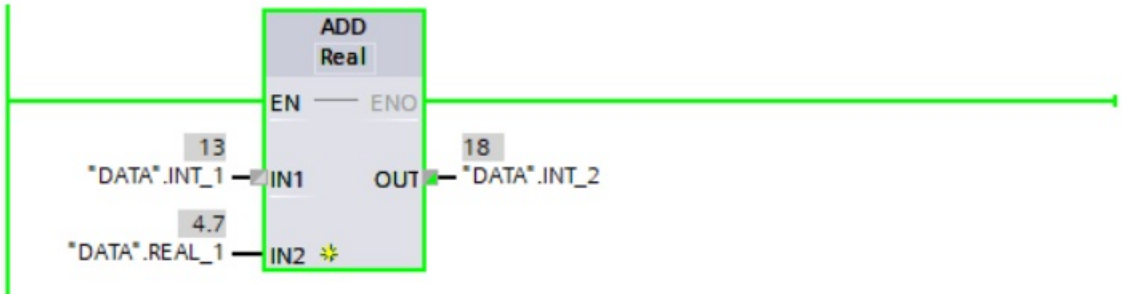


Obr. 3.116. Monitorovanie programu s automatickou konverziou dátových typov



Obr. 3.117. Monitorovanie programu s automatickou konverziou dátových typov

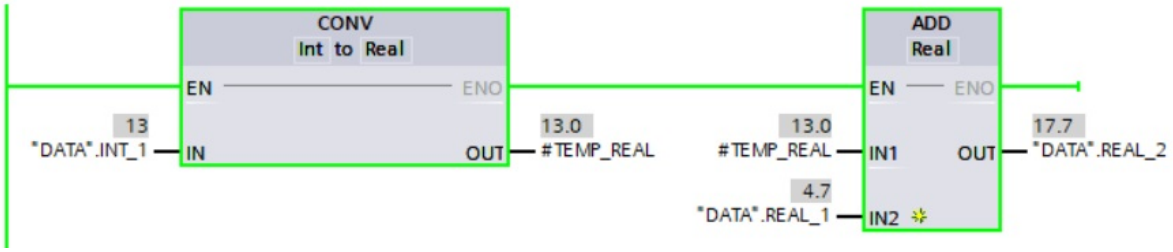
Na Obr. 3.118 je upravený program z predchádzajúcich príkladov. Dátový typ INT bol zmenený na REAL. Premenná pripojená na vstup IN1 je dátového typ INT. Jej hodnota 13 je konvertovaná na dátový typ REAL, na hodnotu 13,0. Na vstup IN2 je pripojená premenná dátového typu REAL. Konverzia nie je potrebná. Hodnota vstupu je 4,7. Súčet IN1 a IN2 je 17,7. Táto hodnota je uložená na výstup OUT ako desatinné číslo (dátový typ REAL). K výstupu OUT je pripojená premenná dátového typu INT, preto sa hodnota 17,7 zapíše do premennej ako celé číslo (hodnota 18).



Obr. 3.118. Monitorovanie programu s automatickou konverziou dátových typov

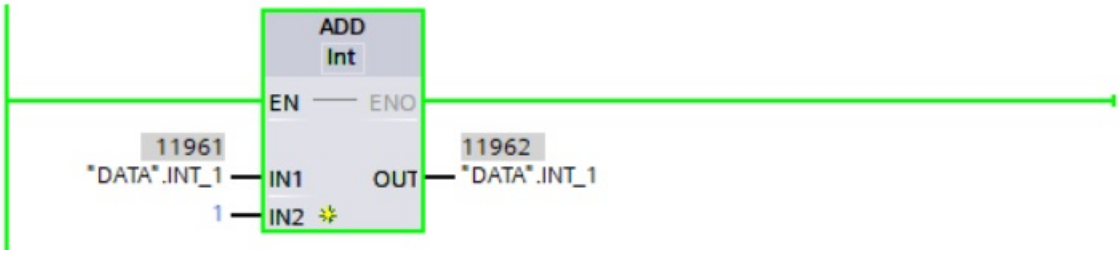
Ak automatická konverzia dátových typov nie je dostupná, použijeme inštrukciu *CONV* (Obr. 3.119). Hodnota premennej DATA.INT_1 je pretypovaná z INT na REAL. Výsledok

sa zapisuje do dočasnej premennej TEMP_REAL. Tá je vstupným operandom inštrukcie *ADD*.

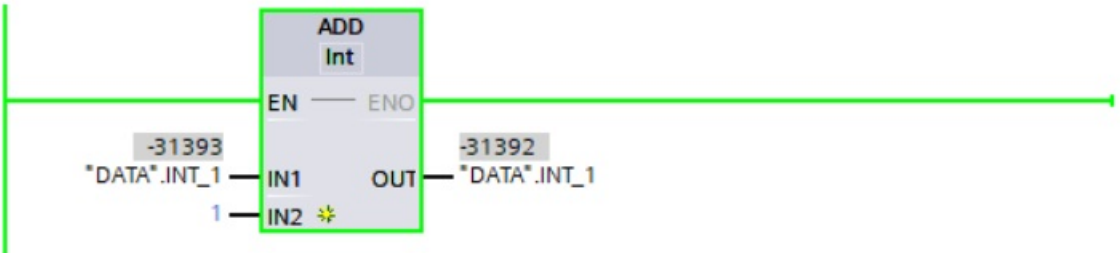


Obr. 3.119. Monitorovanie programu s manuálnou konverziou dátových typov

Inštrukcia *ADD* sa často používa pri navyšovaní počtov ako napr. počet výrobných cyklov, počet spustení motora, počet kusov objektov vo výrobnej linke, počet vyrobených kusov, počet nepodarkov atď. Na Obr. 3.120 je implementovaný vzťah DATA.INT1 = DATA.INT1 + 1. Nakoľko na vstupe inštrukcie je stále hodnota TRUE, inštrukcia *ADD* sa vykonáva v každom cykle. Ako vidno na obrázku, od vynulovania hodnoty premennej prešlo 11 961 cyklov. Postupným navyšovaním hodnoty dôjde k pretečeniu dátového typu. Na Obr. 3.121 je príklad, keď hodnota premennej dosiahla maximálnu hodnotu dátového typu INT (32 768), pretiekla na hodnotu -32 768 a postupne sa navyšovala. Monitorovanie bolo zaznamenané pri hodnote -31 393.



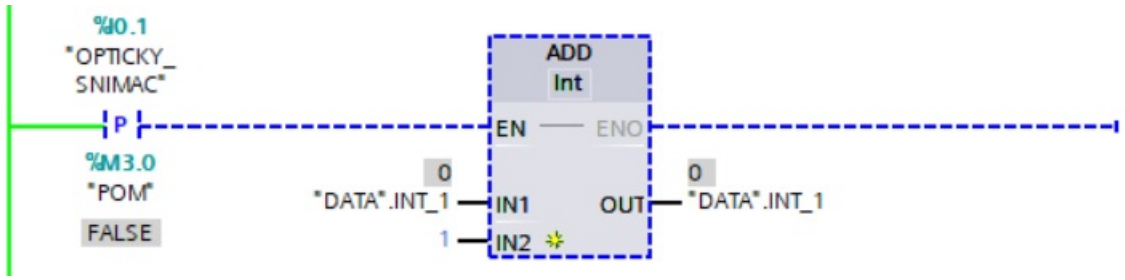
Obr. 3.120. 1. fáza monitorovania inkrementovania hodnoty premennej



Obr. 3.121. 2. fáza monitorovania inkrementovania hodnoty premennej

Ak má premenná DATA.INT_1 reprezentovať početnosť v závislosti od detekcie objektov snímačom OPTICKY_SNIMAC, použijeme napr. jednu z inštrukcií detekcie nábežnej hrany (Obr. 3.122). Ak hodnota snímača zmení stav z FALSE na TRUE, inštrukcia *ADD* sa vykoná len raz. Hodnota premennej DATA.INT_1 bude navýšená len raz. Nesprávnym riešením by bolo použitie napr. inštrukcie *spínací kontakt* s operandom OPTICKY_SNIMAC.

Snímač je v praxi aktivovaný niekoľko milisekúnd alebo sekúnd počas ktorých sa vykoná niekoľko cyklov. Inštrukcia *ADD* by sa nevykonala raz, podľa predpokladu.

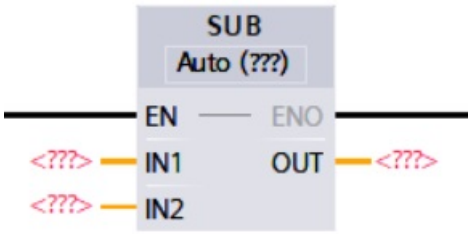


Obr. 3.122. Inštrukcia *ADD* s detekciou nábežnej hrany

V prípade, že jedným zo vstupných operandov inštrukcie *ADD* (jedným zo sčítancov) je konštanta 1, podobne ako v predchádzajúcom príklade, môžeme s výhodou hodnotu premennej inkrementovať aj s využitím inštrukcie *INC - Increment*.

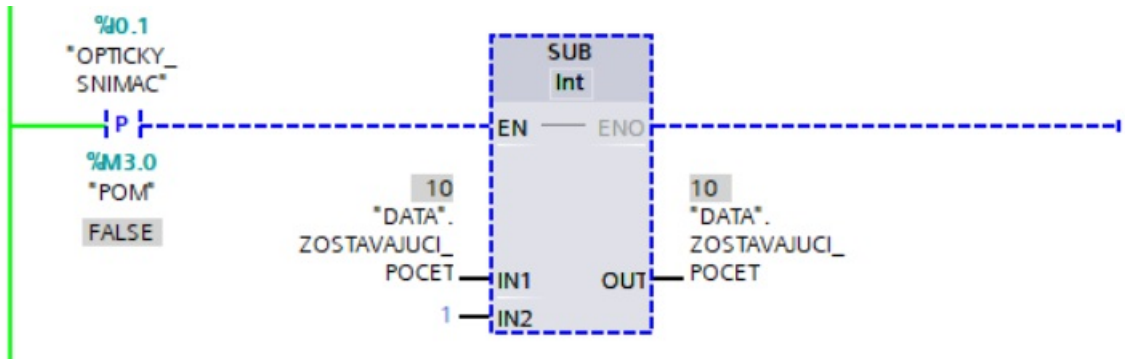
3.4.2 Odčítanie (SUB)

Inštrukcia realizuje matematickú operáciu rozdielu (SUB - subtraction) dvoch nebitových operandov podľa vzťahu OUT = IN1 - IN2. Symbol inštrukcie *SUB* je na Obr. 3.123. Inštrukcia má na rozdiel od inštrukcie *ADD* len dva operandy.

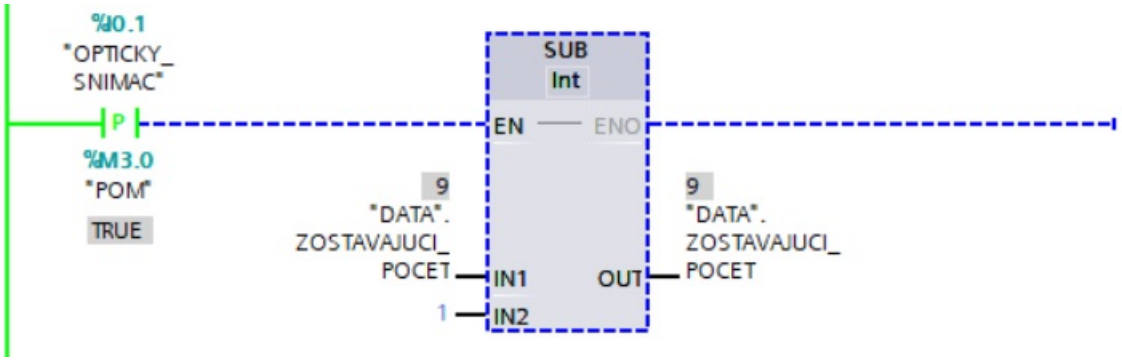


Obr. 3.123. Symbol inštrukcie *SUB*

V príklade na Obr. 3.124 bola manuálne nastavená hodnota premennej DATA.ZOST AVAJUCI_POCET na 10. Hodnota vstupu OPTICKY_SNIMAC je FALSE. Zmena stavu snímača z FALSE na TRUE (Obr. 3.125) jeden raz spustila inštrukciu [SUB], ktorá dekrementovala hodnotu premennej o 1.



Obr. 3.124. 1. fáza monitorovania programu s inštrukciou *SUB*



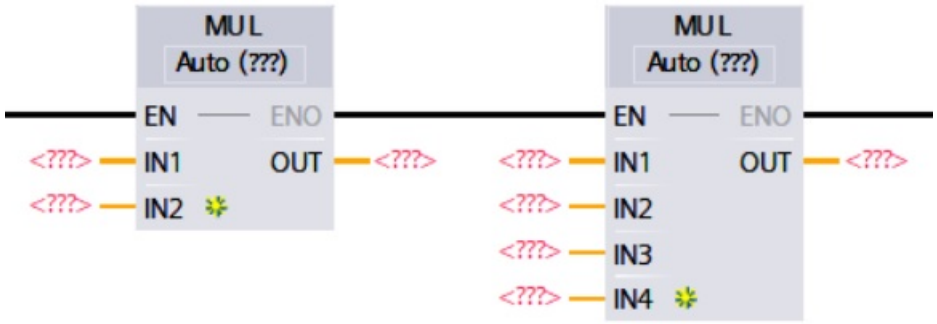
Obr. 3.125. 2. fáza monitorovania programu s inštrukciou *SUB*

V prípade, že menšiteľom na druhom vstupe inštrukcie *SUB* je konštanta 1, teda dekrementujeme hodnotu premennej o 1, môžeme s výhodou využiť aj inštrukciu *DEC* - *Decrement*.

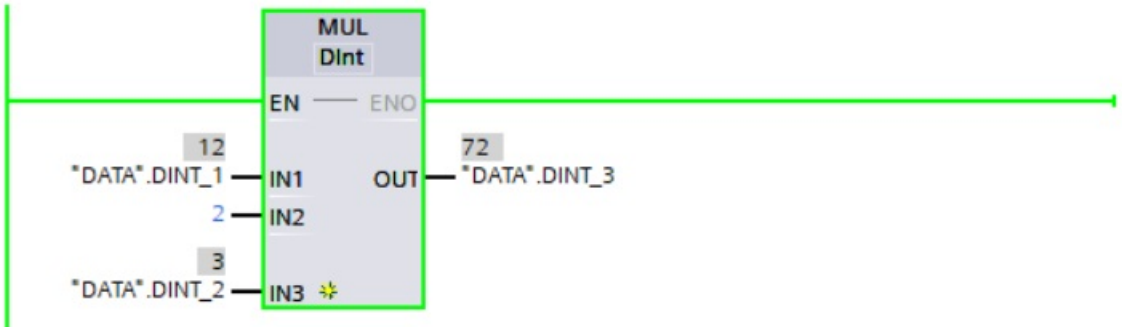
3.4.3 Násobenie (MUL)

Inštrukcia *MUL* (Obr. 3.126) reprezentuje matematickú operáciu násobenia (MUL - multiplication). Preddefinovaný počet vstupných operandov je dva (IN1 a IN2), ale podobne ako pri inštrukcii *ADD*, možno počet vstupov navýšiť (Obr. 3.126 - symbol vpravo). Výsledok multiplikácie vstupov (OUT = IN1 * IN2 * ... INn) sa zapisuje do OUT.

Príklad monitorovania hodnôt multiplikácie troch operandov je na Obr. 3.127.



Obr. 3.126. Symbol inštrukcie *MUL*



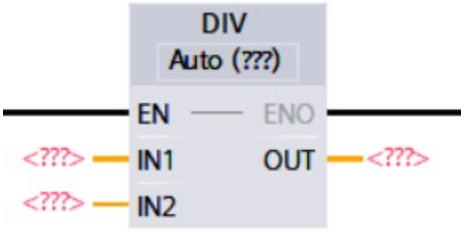
Obr. 3.127. Príklad inštrukcie *MUL*

3.4.4 Delenie (DIV)

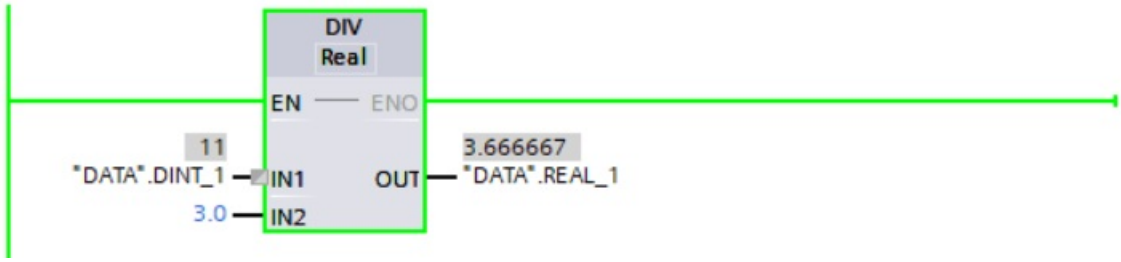
Symbol inštrukcie *DIV* je na Obr. 3.128. Inštrukcia realizuje operáciu delenia (DIV - division) podľa vzťahu OUT = IN1 / IN2.

Prvý príklad (Obr. 3.129) demonštruje implementáciu inštrukcie nad dátovým typom REAL. Premenná priradená k vstupu IN1 sa automaticky konvertuje na dátový typ REAL. Tá je predelená hodnotou 3,0. Výsledok 3,666 je zapísaný do premennej dátového typu REAL.

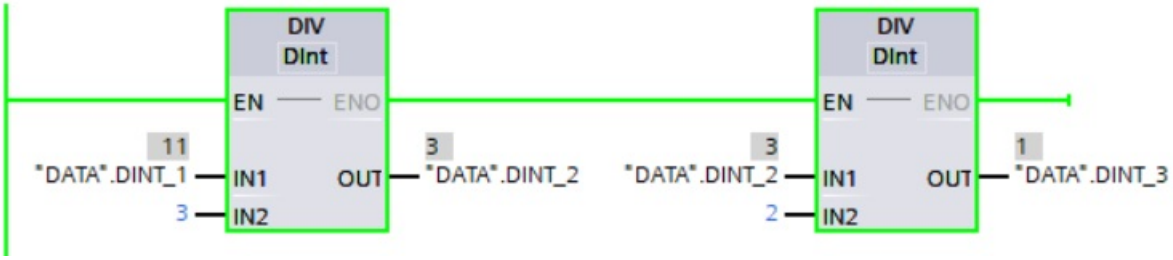
Príklad monitorovania programu s inštrukciou *DIV* je na Obr. 3.130. Ako dátové typy v inštrukcii sú nastavené DINT. Všimnite si výsledky celočíselného delenia. Výsledok delenia 11 / 3 = 3,666 je zapísaný na výstup OUT ako celočíselná hodnota 3 (stráca sa desatinná časť čísla ako pri inštrukcii *TRUNC*). Podobne je to aj v druhom príklade (*DIV* vpravo).



Obr. 3.128. Symbol inštrukcie *DIV*



Obr. 3.129. 1. príklad použitia inštrukcie *DIV*

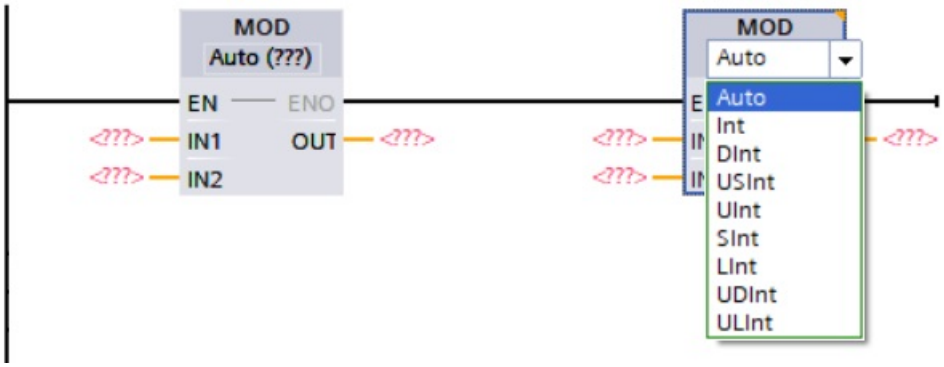


Obr. 3.130. 2. príklad použitia inštrukcie *DIV*

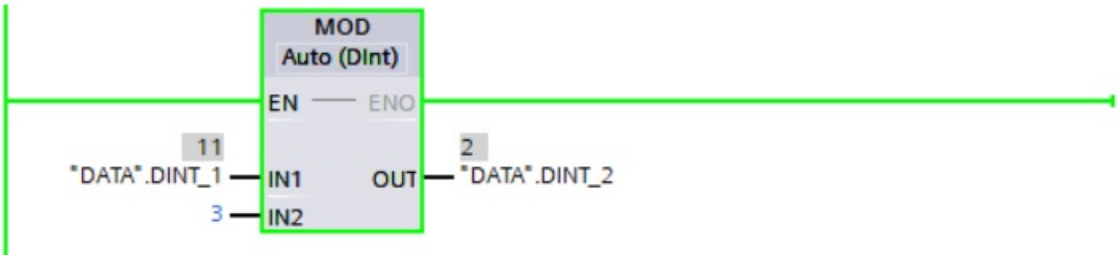
3.4.5 Zvyšok po delení (MOD)

Inštrukcia *MOD* vracia celočíselný zvyšok delenia dvoch celočíselných vstupných operandov (modulo). Symbol inštrukcie je na Obr. 3.131 (vľavo). V pravej časti obrázku je ponú-

kaný zoznam celočíselných dátových typov. Príklad použitia je na Obr. 3.132.



Obr. 3.131. Symbol inštrukcie *MOD*



Obr. 3.132. Príklad použitia inštrukcie *MOD*

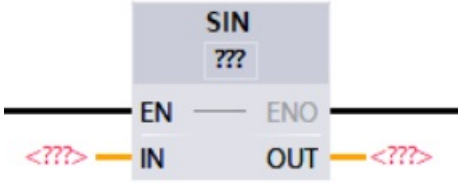
3.4.6 Sínus (SIN)

Symbol inštrukcie *SIN* je na Obr. 3.133. Inštrukcia sa používa na výpočet sínusu uhla zadaného v radiánoch na vstupe IN. Výsledok je uložený do výstupu OUT.

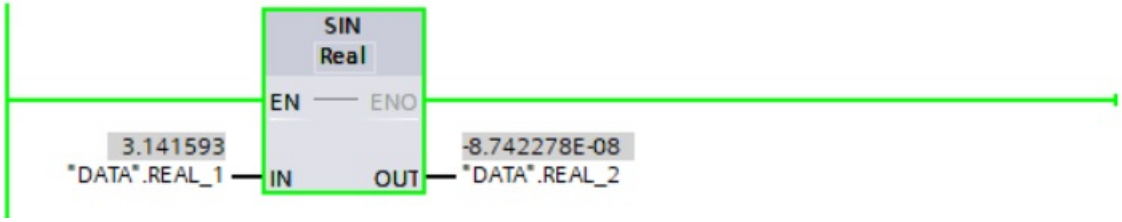
V príklade na Obr. 3.134 bola do premennej DATA.REAL_1 manuálne zapísaná hodnota 3,14159265358979238462. Ako vidno z monitorovania, do premennej sa zapísala hodnota s presnosťou 6 desatinných číslic. Výsledok výpočtu (-8,742278E-08) je približne rovný nule.

V druhom príklade (Obr. 3.135) boli na zvýšenie presnosti výpočtu použité premenné dátových typov LREAL. Monitorovanie hodnôt premenných z oboch príkladov je na Obr. 3.136.

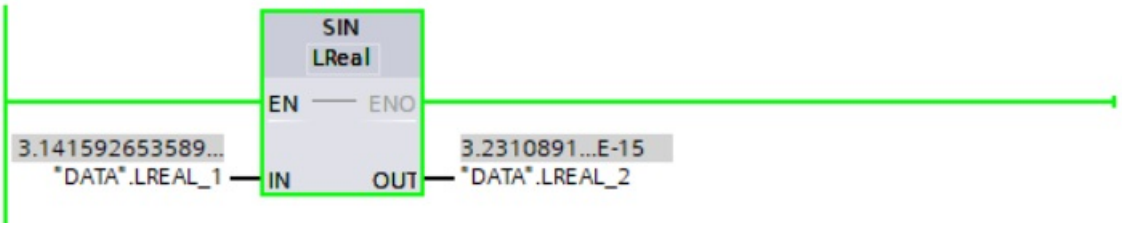
Priebeh funkcie na vybranom vstupnom intervale je na Obr. 3.137.



Obr. 3.133. Symbol inštrukcie *SIN*



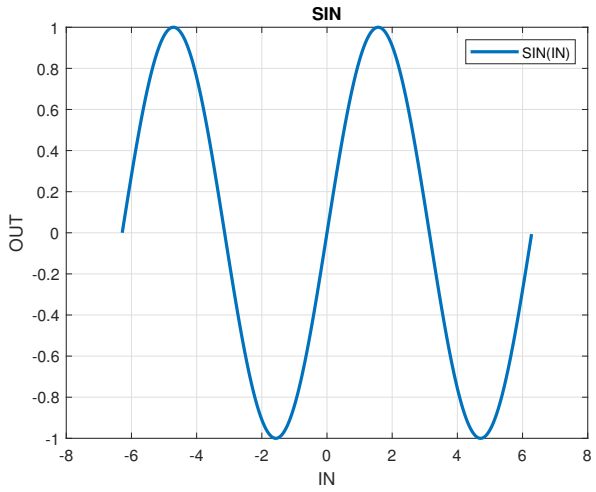
Obr. 3.134. Príklad použitia inštrukcie *SIN*



Obr. 3.135. Príklad použitia inštrukcie *SIN*

Name	Address	Display format	Monitor value
"DATA".REAL_1		Floating-point number	3.141593
"DATA".LREAL_1		Floating-point number	3.14159265358979
"DATA".REAL_2		Floating-point number	-8.742278E-08
"DATA".LREAL_2		Floating-point number	3.23108914886517E-15

Obr. 3.136. Hodnoty premenných



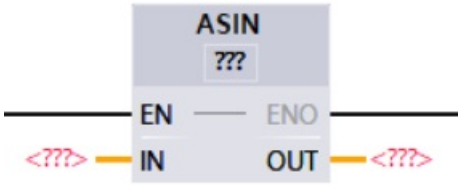
Obr. 3.137. Priebeh hodnoty sínusu (OUT) v závislosti od vstupu IN

3.4.7 Arkussínus (ASIN)

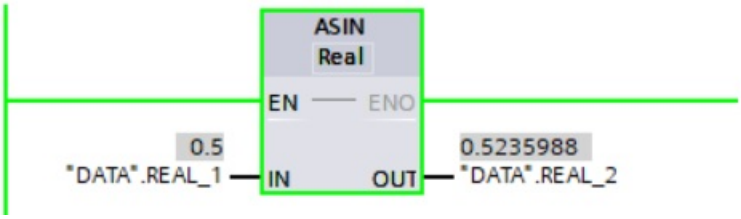
Symbol inštrukcie *ASIN* je na Obr. 3.138. Inštrukcia vypočíta hodnotu arkussínusu vstupnej hodnoty. Na vstupe IN možno zadať iba platné čísla s pohyblivou desatinnou čiarkou

v rozsahu -1,0 až +1,0. Vypočítaná veľkosť uhla je v radiánoch zapísaná na výstup OUT. Hodnota OUT je v rozsahu od $-\pi/2$ do $+\pi/2$. Príklad so hodnotou hodnotu 0,5 je na Obr. 3.139.

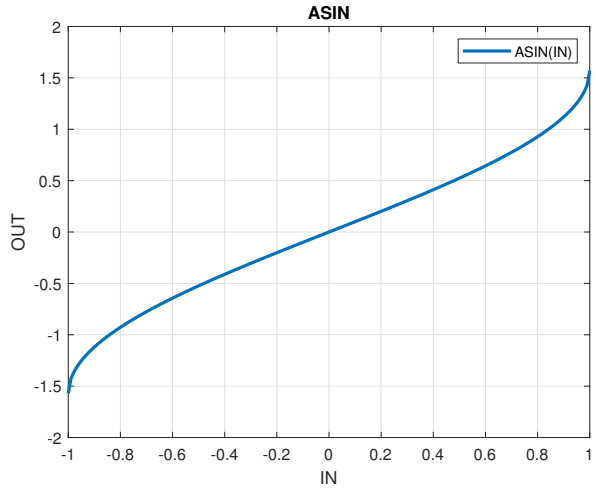
Priebeh funkcie je na Obr. 3.140.



Obr. 3.138. Symbol inštrukcie *ASIN*



Obr. 3.139. Príklad použitia inštrukcie *ASIN*

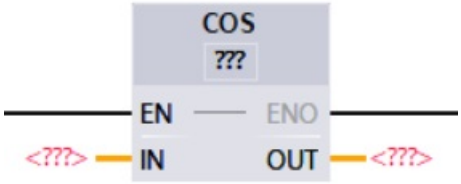


Obr. 3.140. Priebeh hodnoty arkussínus (OUT) v závislosti od vstupu IN

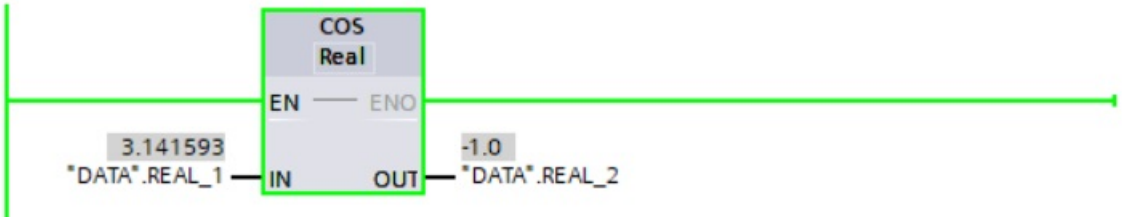
3.4.8 Kosínus (COS)

Symbol inštrukcie *COS* je na Obr. 3.141. Inštrukcia sa používa na výpočet kosínusu uhla zadaného v radiánoch na vstupe IN. Výsledok je uložený do výstupu OUT. Príklad so vstupnou hodnotou uhla 3,141592 je na Obr. 3.142.

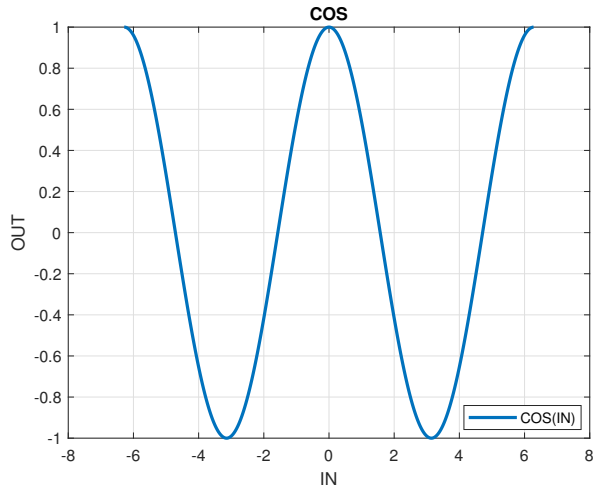
Priebeh funkcie je na Obr. 3.143.



Obr. 3.141. Symbol inštrukcie *COS*



Obr. 3.142. Príklad použitia inštrukcie *COS*

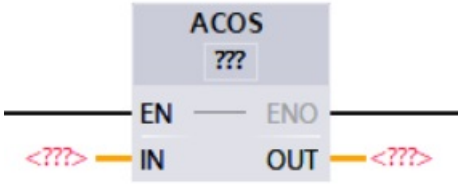


Obr. 3.143. Priebeh hodnoty kosínusu (OUT) v závislosti od vstupu IN

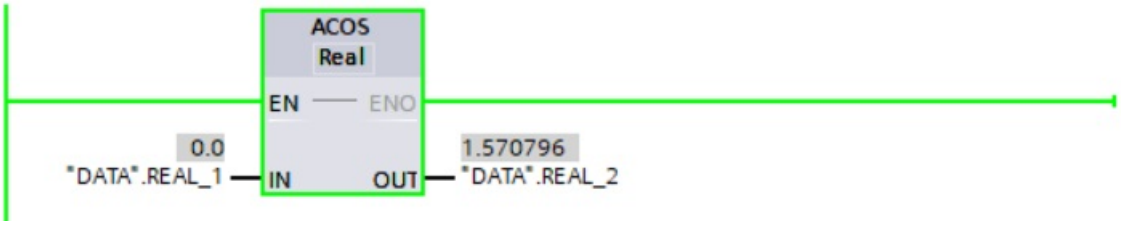
3.4.9 Arkuskosínus (ACOS)

Symbol inštrukcie *ACOS* je na Obr. 3.144. Inštrukcia vypočíta hodnotu arkuskosínusu vstupnej hodnoty. Na vstupe IN možno zadať iba platné čísla s pohyblivou desatinnou čiarkou v rozsahu -1,0 až +1,0. Vypočítaná veľkosť uhla je v radiánoch zapísaná na výstup OUT. Hodnota OUT je v rozsahu od 0 do π. Príklad so vstupnou hodnotou 0,0 je na Obr. 3.145.

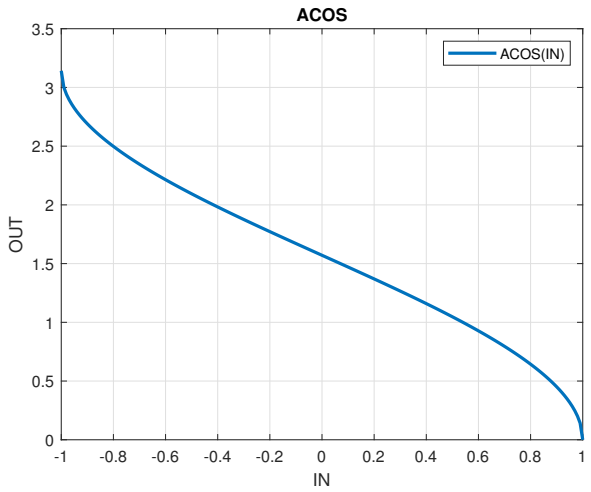
Priebeh funkcie je na Obr. 3.146.



Obr. 3.144. Symbol inštrukcie *ACOS*



Obr. 3.145. Príklad použitia inštrukcie *ACOS*



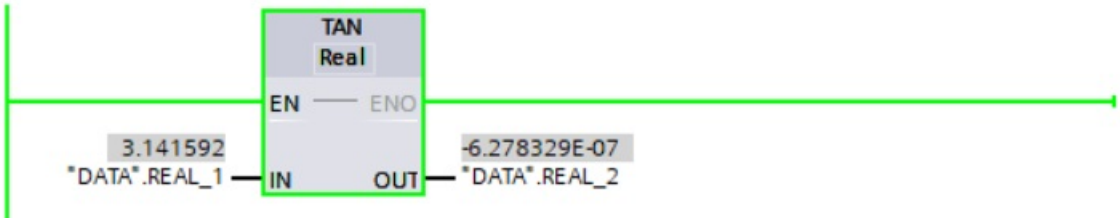
Obr. 3.146. Priebeh hodnoty arkuskosínusu (OUT) v závislosti od vstupu IN

3.4.10 Tangens (TAN)

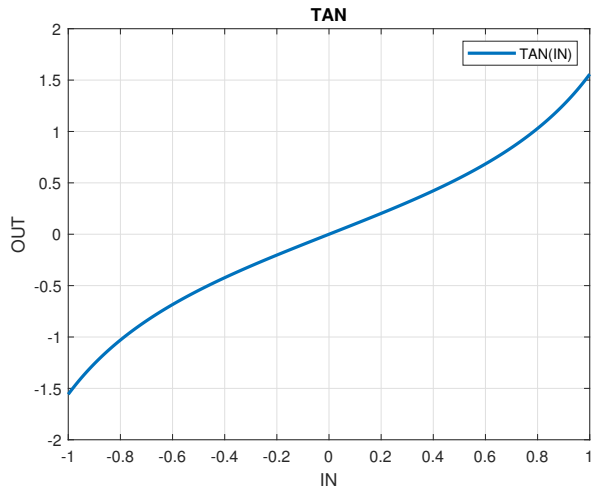
Symbol inštrukcie *TAN* je na Obr. 3.147. Inštrukcia vypočíta tangens uhla zadaného v radiánoch na vstupe IN. Výsledok je uložený do výstupu OUT. Príklad so vstupnou hodnotou uhla 3,141592 je na Obr. 3.148. Výsledkom je približne 0,0.

Priebeh funkcie na vybranom vstupnom intervale je na Obr. 3.149.

Obr. 3.147. Symbol inštrukcie *TAN*



Obr. 3.148. Príklad použitia inštrukcie *TAN*

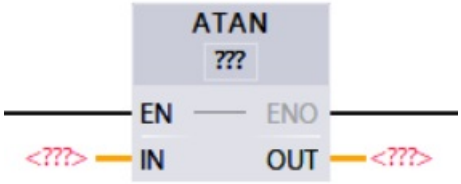


Obr. 3.149. Priebeh hodnoty tangensu (OUT) v závislosti od vstupu IN

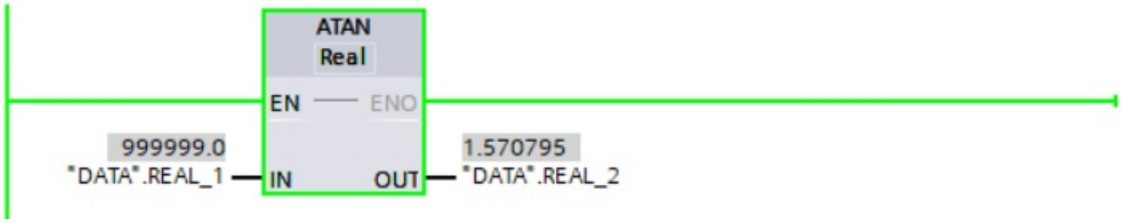
3.4.11 Arkustangens (ATAN)

Symbol inštrukcie *ATAN* je na Obr. 3.150. Inštrukcia vypočíta hodnotu arkustangensu vstupnej hodnoty. Na vstupe IN možno zadať čísla s pohyblivou desatinnou čiarkou. Vypočítaná veľkosť uhla je v radiánoch zapísaná na výstup OUT. Hodnota OUT je v rozsahu od $-\pi/2$ do $+\pi/2$. Príklad so vstupnou hodnotou 999999,0 je na Obr. 3.151. Výsledok je približne $+\pi/2$.

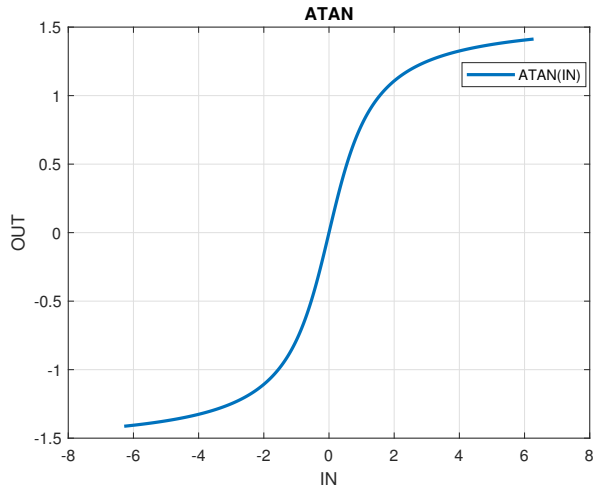
Priebeh funkcie je na Obr. 3.152.



Obr. 3.150. Symbol inštrukcie *ATAN*



Obr. 3.151. Príklad použitia inštrukcie *ATAN*

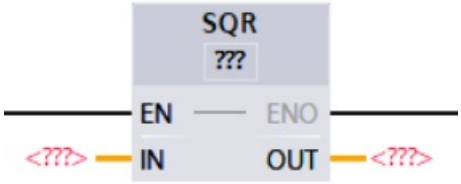


Obr. 3.152. Priebeh hodnoty arkustangensu (OUT) v závislosti od vstupu IN

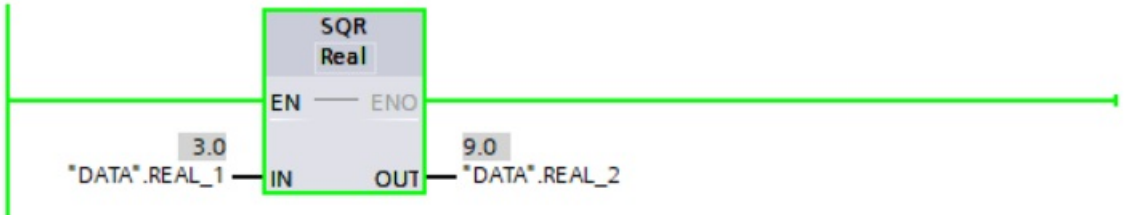
3.4.12 Druhá mocnina (SQR)

Symbol inštrukcie *SQR* je na Obr. 3.153. Inštrukcia vypočíta druhú mocninu hodnoty na vstupe IN a uloží ju na výstup OUT. Príklad inštrukcie je na Obr. 3.154. Ekvivalentný príklad pomocou inštrukcie *MUL* je na Obr. 3.155. Tretia mocnina by sa riešila pridaním tretieho vstupu s priradenou premennou DATA.REAL_1.

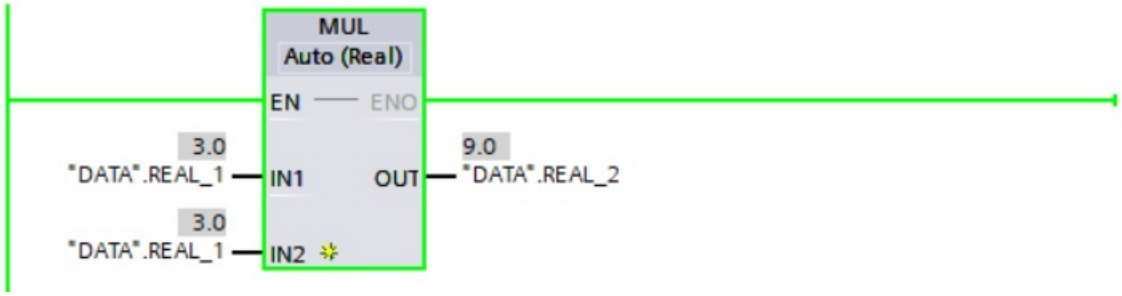
Priebeh funkcie na vybranom vstupnom intervale je na Obr. 3.156.



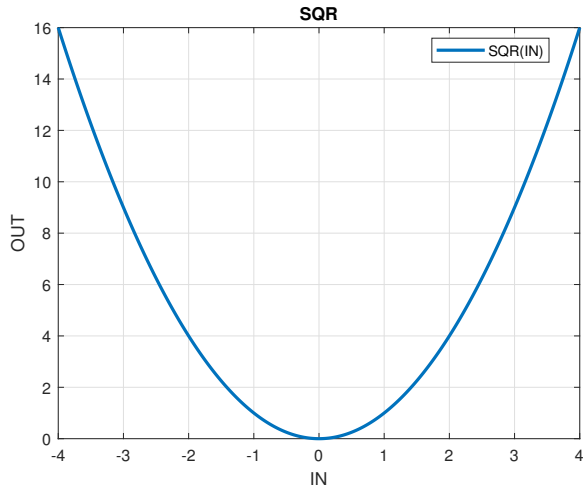
Obr. 3.153. Symbol inštrukcie *SQR*



Obr. 3.154. Príklad použitia inštrukcie *SQR*



Obr. 3.155. Ekvivalentný program



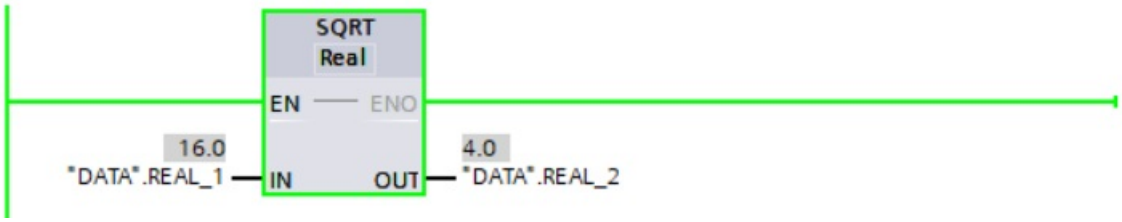
Obr. 3.156. Priebeh hodnoty druhej mocniny (OUT) v závislosti od vstupu IN

3.4.13 Druhá odmocnina (SQRT)

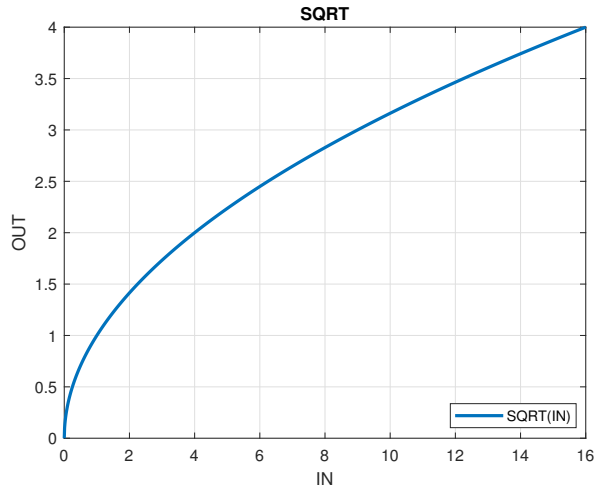
Symbol inštrukcie *SQRT* je na Obr. 3.157. Inštrukcia vypočíta druhú odmocninu hodnoty na vstupe IN a výsledok zapíše na výstup OUT. Ak je vstupná hodnota záporná, výsledkom bude neplatné číslo s pohyblivou desatinnou čiarkou. Príklad so vstupnou hodnotou 16,0 a výsledkom 4,0 je na Obr. 3.158. Pre hodnotu 0,0 je výsledok 0,0.

Priebeh funkcie na vybranom vstupnom intervale je na Obr. 3.159.

Obr. 3.157. Symbol inštrukcie *SQRT*



Obr. 3.158. Príklad použitia inštrukcie *SQRT*

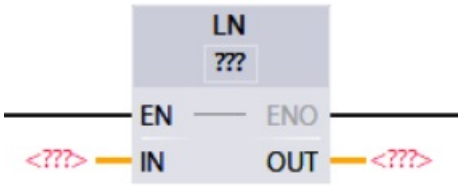


Obr. 3.159. Priebeh hodnoty druhej odmocniny (OUT) v závislosti od vstupu IN

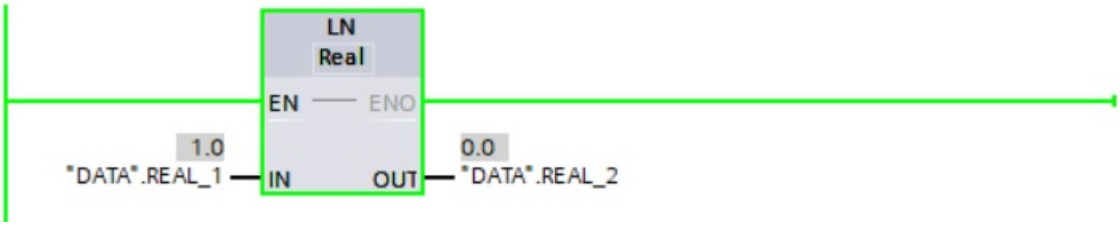
3.4.14 Prirodzený logaritmus (LN)

Na výpočet prirodzeného logaritmu so základom e ($e = 2,718282$) hodnoty na vstupe IN môžeme použiť inštrukciu *LN*. Symbol inštrukcie je na Obr. 3.160. Príklad so vstupnou hodnotou 1,0 je na Obr. 3.161.

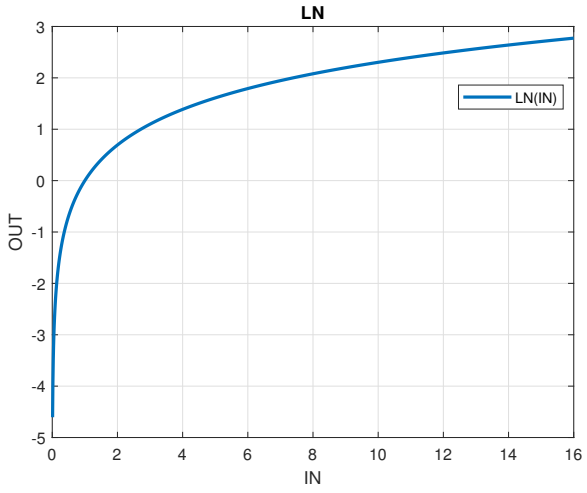
Priebeh funkcie na vybranom vstupnom intervale je na Obr. 3.162.



Obr. 3.160. Symbol inštrukcie *LN*



Obr. 3.161. Príklad použitia inštrukcie *LN*

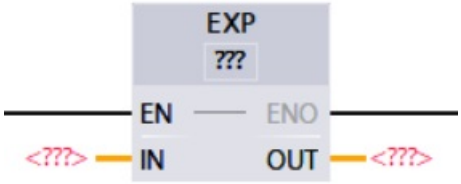


Obr. 3.162. Priebeh hodnoty prirodzeného logaritmu (OUT) v závislosti od vstupu IN

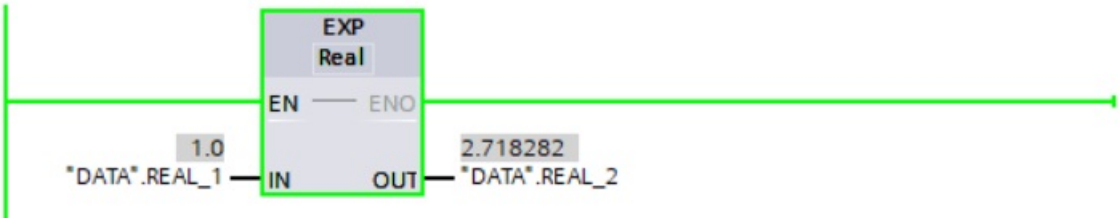
3.4.15 Prirodzená exponenciálna funkcia *e* (EXP)

Symbol inštrukcie *EXP* je na Obr. 3.163. Výpočet poskytuje hodnotu prirodzenej exponenciálnej funkcie, teda exponenciálnej funkcie so základom e. Vzťah výpočtu je $OUT = e^{IN}$. Príklad so vstupom 1,0 je na Obr. 3.164.

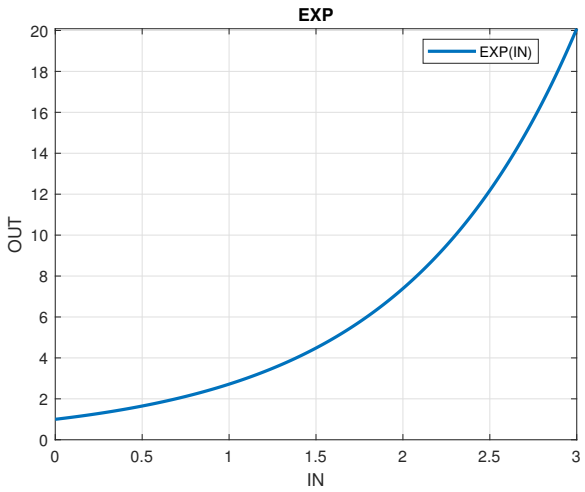
Priebeh funkcie na vybranom vstupnom intervale je na Obr. 3.165.



Obr. 3.163. Symbol inštrukcie *EXP*



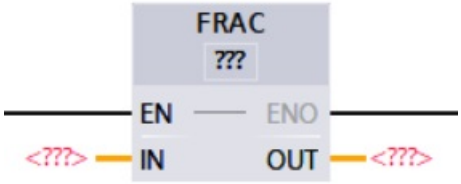
Obr. 3.164. Príklad použitia inštrukcie *EXP*



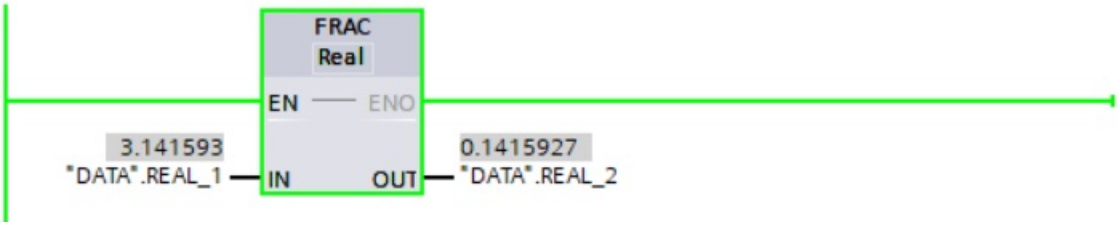
Obr. 3.165. Priebeh hodnoty exponenciálnej funkcie so základom e (OUT) v závislosti od vstupu IN

3.4.16 Desatinná časť čísla (FRAC)

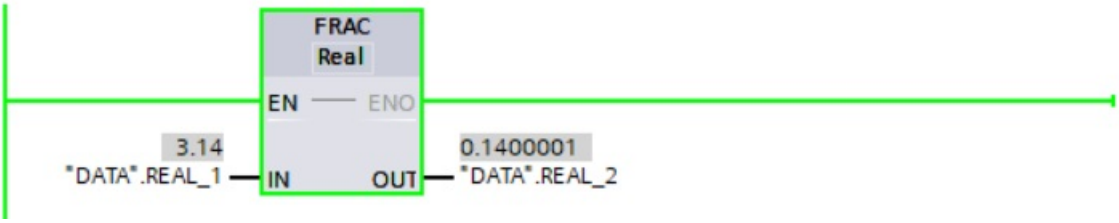
Inštrukcia vracia desatinnú časť reálneho čísla. Symbol inštrukcie *FRAC* je na Obr. 3.166. Ak má napríklad vstup IN hodnotu 123,4567, výstup OUT vráti hodnotu 0,4567. Na Obr. 3.167 a 3.168 vidno, že desatinná časť na výstupe OUT sa nezhoduje s desatinnou časťou čísla na vstupe IN. Nepresnosť je spôsobená obmedzeným rozsahom zobrazenia pri monitorovaní hodnoty premennej dátového typu REAL. Reálne čísla sú aproximované hodnoty čísiel.



Obr. 3.166. Symbol inštrukcie *FRAC*



Obr. 3.167. 1. príklad inštrukcie *FRAC*



Obr. 3.168. 2. príklad inštrukcie *FRAC*

3.5 Porovnávacie bloky (Comparator operations)

Porovnávacie inštrukcie sú riadiace bloky, ktoré sa môžu nachádzať na ľubovoľnej pozícii v rebríkovej schéme (okrem polohy úplne vpravo). Slúžia na porovnanie hodnôt nebinárnych operandov, ktoré sú zvyčajne premenné alebo konštanty. Ich logický výstup je závislý od typu porovnania a ovplyvňuje tok signálu v rebríkovej schéme (RLO). V tejto kapitole opíšeme najčastejšie používané inštrukcie, ktoré majú svoje ekvivalenty aj v programovacích nástrojoch od iných výrobcov riadiacich systémov ako Siemens. Zoznam porovnávacích inštrukcií v TIA Portal pre CPU S7-1200 je na Obr. 3.169.

Comparator operations	
CMP ==	Equal
CMP <>	Not equal
CMP >=	Greater or equal
CMP <=	Less or equal
CMP >	Greater than
CMP <	Less than
IN_Range	Value within range
OUT_Range	Value outside range
–|OK|–	Check validity
–|NOT_OK|–	Check invalidity
Variant	

Obr. 3.169. Zoznam inštrukcií porovnávacích inštrukcií

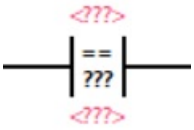
3.5.1 Rovná sa (==)

Symbol inštrukcie *rovná sa* je na Obr. 3.170. Inštrukcia testuje či sa operandy Operand 1 a Operand 2 rovnajú. Prvý operand sa zadáva nad symbolom inštrukcie a druhý operand pod symbol inštrukcie. Operandmi inštrukcie môžu byť premenné rôznych dátových typov alebo konštanty. Ak sa hodnoty operandov rovnajú, potom je výstupom inštrukcie hodnota TRUE, inak FALSE. V prípade použitia rôznych dátových typov je potrebná konverzia (automatická alebo manuálna inštrukciou *CONV*) na rovnaký dátový formát.

V prvom príklade (Obr. 3.171) sa porovnáva hodnota premennej s číslom 3. Hodnota premennej DATA.INT_1 je 0 a tá sa nerovná hodnote 3. Výsledkom porovnania je hodnota FALSE. Do premennej LED sa zapisuje hodnota FALSE.

V druhom príklade (Obr. 3.172) je podmienka platná lebo premenná nadobudla manuálnym nastavením hodnotu 3. Platnosť porovnania je animovaná zeleným symbolom inštrukcie. Do výstupu LED sa zapisuje výsledok porovnania, hodnota TRUE.

V poslednom príklade (Obr. 3.173) sú porovnané dve reálne hodnoty. V praxi sa neodporúča takýmto spôsobom porovnávať hodnoty reprezentované ako REAL, lebo ide o aproximované hodnoty.



Obr. 3.170. Symbol inštrukcie *rovný*

Obr. 3.171. 1. príklad inštrukcie *rovný*

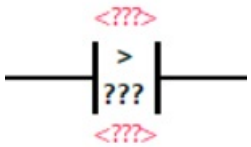
Obr. 3.172. 2. príklad inštrukcie *rovný*

Obr. 3.173. 3. príklad inštrukcie *rovný*

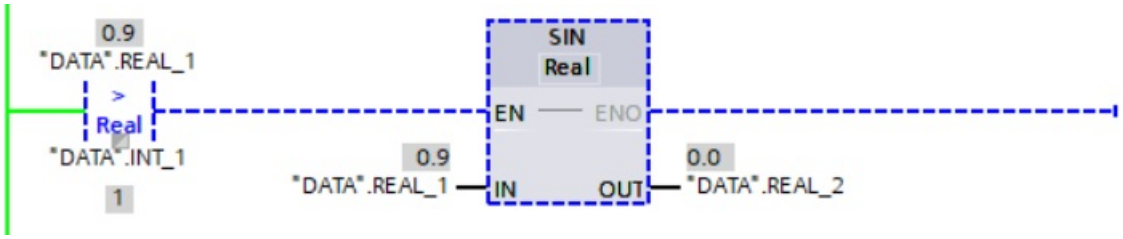
3.5.2 Väčší (>)

Inštrukcia *väčší* testuje kedy je prvý operand nad inštrukciou väčší ako druhý operand zadaný pod inštrukciou. Ak platí podmienka Operand 1 > Operand 2, potom výstupom inštrukcie je hodnota TRUE, inak FALSE. Symbol inštrukcie je na Obr. 3.174.

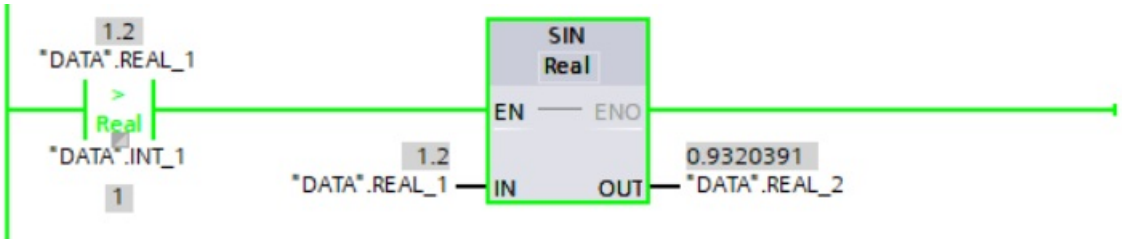
Príklad inštrukcie s automatickou konverziou dátových typov je na Obr. 3.175 a 3.176. Automatická konverzia je indikovaná štvorčekom v hornej alebo dolnej časti symbolu inštrukcie podľa toho, ktorý operand sa automaticky konvertuje. Nastavený dátový typ porovnania je REAL. Druhý operand DATA.INT_1 je dátového typu INT. Jeho hodnota 1 sa prevedie na dátový typ REAL (1,0), ktorá sa porovná s horným operandom. Na Obr. 3.175 je príklad neplatnej podmienky a na Obr. 3.176 je príklad platnej podmienky. Inštrukcia výpočtu sínusu sa vykonáva ak platí porovnanie, v opačnom prípade sa inštrukcia *SIN* nevykoná a hodnota DATA.REAL_2 nie je modifikovaná programom.



Obr. 3.174. Symbol inštrukcie *väčší*



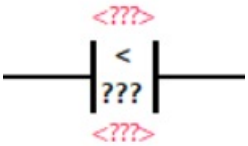
Obr. 3.175. Príklad inštrukcie *väčší* s automatickou konverziou dátových typov



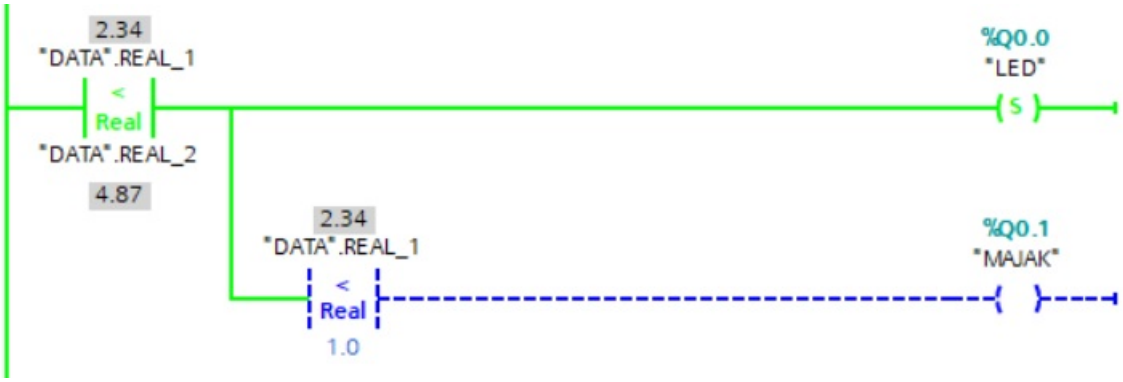
Obr. 3.176. Príklad inštrukcie *väčší* s automatickou konverziou dátových typov

3.5.3 Menší (<)

Symbol inštrukcie *menší* je na Obr. 3.177. Ak má horný operand menšiu hodnotu ako dolný operand, výsledkom porovnania je TRUE, inak FALSE. Príklad použitia inštrukcie je na Obr. 3.178. Prvá inštrukcia porovnáva hodnoty dvoch premenných. Výsledok porovnania je platný. Signál RLO je TRUE, preto sa v hornej vetve vykoná inštrukcia *SET*. Program pokračuje v spodnej vetve. Hodnota premennej sa porovná s konštantou 1,0. Výsledok porovnania je FALSE. Následnosť porovnávacích inštrukcií tvorí logickú operáciu AND. Prvá (horná) časť je platná, ale dolná nie je platná. Stav RLO za druhou porovnávacou inštrukciou je FALSE, preto sa do premennej MAJAK zapíše hodnota FALSE.



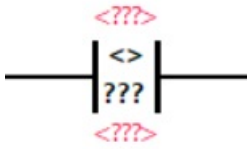
Obr. 3.177. Symbol inštrukcie *menší*



Obr. 3.178. Príklad inštrukcie *menší*

3.5.4 Nerovný (<>)

Symbol inštrukcie *nerovný* je na Obr. 3.179. Ak má horný operand inú hodnotu ako dolný operand, výsledkom porovnania je TRUE, inak FALSE. Na Obr. 3.180. sú porovnané hodnoty dvoch premenných dátových typov DINT. Hodnoty nie sú rôzne, preto výstupom inštrukcie je hodnota FALSE. Inštrukcia *RESET* nie je vykonaná.

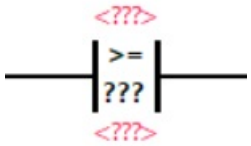


Obr. 3.179. Symbol inštrukcie *nerovný*

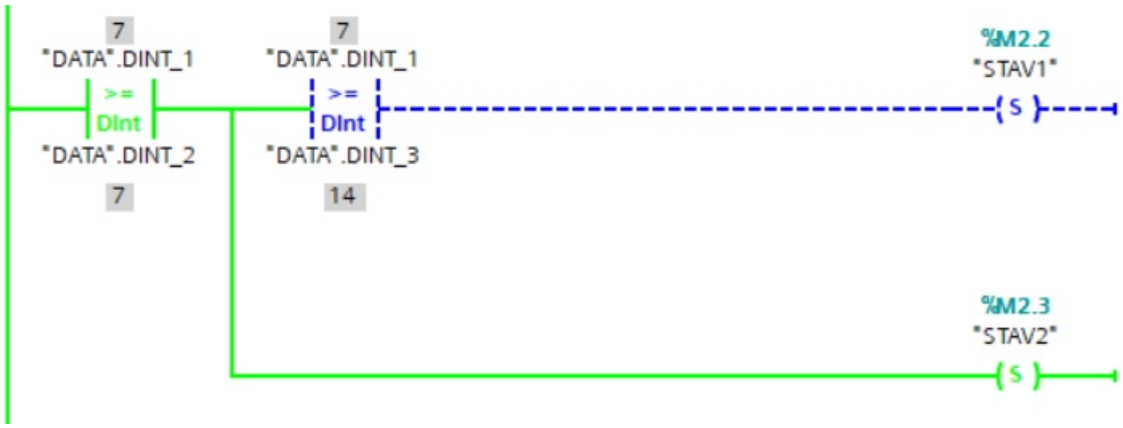
Obr. 3.180. Príklad inštrukcie *nerovný*

3.5.5 Väčší alebo rovný (>=)

Symbol inštrukcie *väčší alebo rovný* je na Obr. 3.181. Ak je horný operand *väčší alebo rovný* ako dolný operand, výsledkom porovnania je TRUE, inak FALSE. Na Obr. 3.182 sú prvou porovnávacou inštrukciou porovnané hodnoty premenných DATA.DINT_1 a DATA.DINT_2. Výsledok porovnania je TRUE. Ďalšia inštrukcia porovnáva hodnoty premenných DATA. DINT_1 a DATA.DINT_3. Výsledok porovnania je FALSE. Logický AND nie je platný, preto inštrukcia *SET* s operandom STAV1 nie je vykonaná. Program pokračuje v spodnej vetve, kam prichádza vyhodnotený signál prvého porovnania (TRUE). Inštrukcia *SET* s operandom STAV2 je spustená.



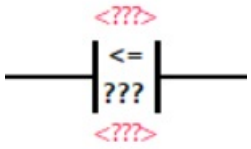
Obr. 3.181. Symbol inštrukcie *väčší alebo rovný*



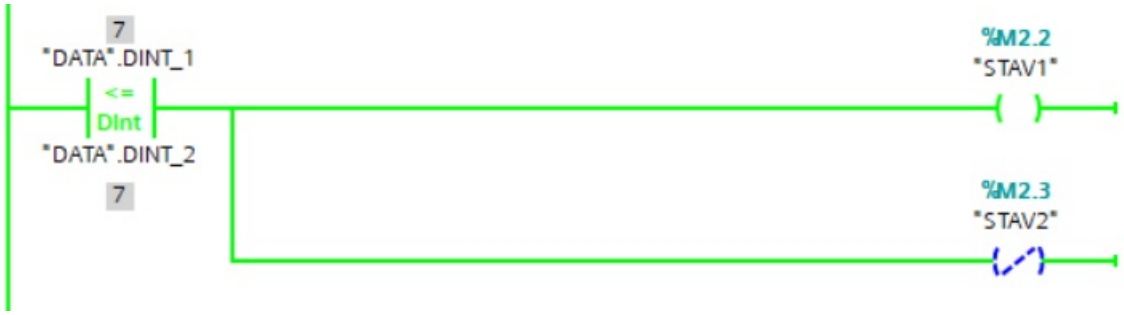
Obr. 3.182. Príklad inštrukcie *väčší alebo rovný*

3.5.6 Menší alebo rovný (<=)

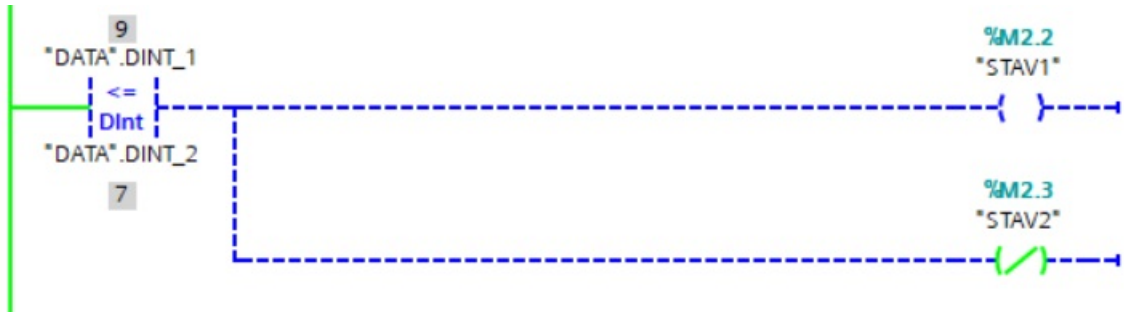
Symbol inštrukcie *menší alebo rovný* je na Obr. 3.183. Ak má horný operand menšiu alebo rovnakú hodnotu ako dolný operand, výsledkom porovnania je TRUE, inak FALSE. Na Obr. 3.184 sú porovnané hodnoty dvoch operandov. Výsledok porovnania je platný (TRUE). Do premennej STAV sa zapisuje hodnota TRUE. Hodnota TRUE prichádza na vstup inštrukcie *negované priradenie*. Do premennej STAV2 sa zapisuje hodnota FALSE. Hodnota premennej STAV1 = TRUE znamená, že výsledok porovnania je platný. Hodnota premennej STAV2 = TRUE znamená, že výsledok porovnania nie je platný (pozri Obr. 3.185).



Obr. 3.183. Symbol inštrukcie *menší alebo rovný*



Obr. 3.184. Príklad inštrukcie *menší alebo rovný*



Obr. 3.185. Príklad inštrukcie *menší alebo rovný*

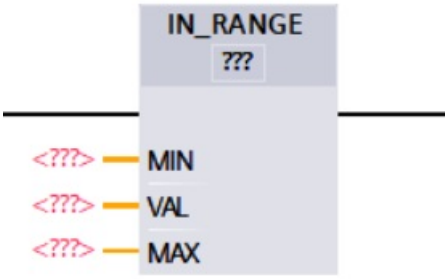
3.5.7 Hodnota v rozsahu (IN_RANGE)

V určitých prípadoch je potrebné testovať či je hodnota premennej v stanovenom rozsahu. Na tento účel môžeme použiť inštrukciu *IN_RANGE*. Symbol inštrukcie je na Obr. 3.186. Hranice rozsahu určujú vstupy MIN a MAX. Testovaná hodnota sa pripája na vstup VAL.

Ak hodnota na vstupe VAL vyhovuje porovnaniu MIN <= VAL a VAL <=MAX, výstup inštrukcie má stav TRUE.

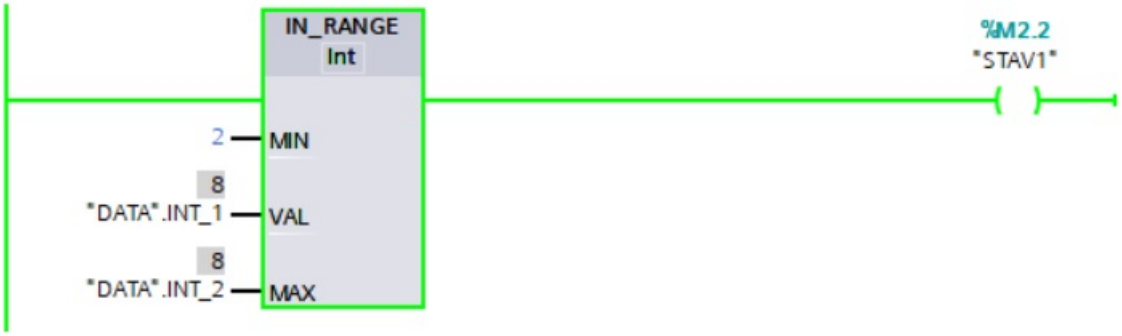
Na Obr. 3.187 sa testuje hodnota premennej DATA.INT_1. Dolná hodnota intervalu je určená konštantnou hodnotou 2 (pozri dátový typ Int). Horná hodnota intervalu je určená hodnotou premennej DATA.INT_2. Hodnota VAL = 0 nie je v intervale <MIN = 2, MAX = 8>, preto výstupom inštrukcie je hodnota FALSE.

V druhom príklade (Obr. 3.188) sa testovaná hodnota VAL = 8 rovná hodnote hornej hranice intervalu MAX = 8. Výstup inštrukcie je TRUE. Ekvivalentné riešenie pomocou základných porovnávacích inštrukcií je na Obr. 3.189. Využíva sa logické spojenie AND inštrukcií *väčší alebo rovný* a *menší alebo rovný*.



Obr. 3.186. Symbol inštrukcie *IN_RANGE*

Obr. 3.187. Príklad inštrukcie *IN_RANGE*

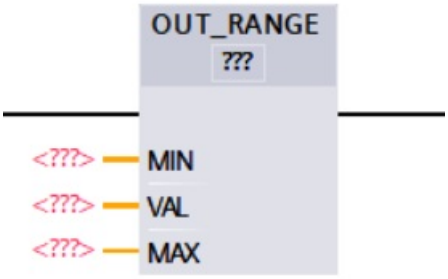


Obr. 3.188. Príklad inštrukcie *IN_RANGE*

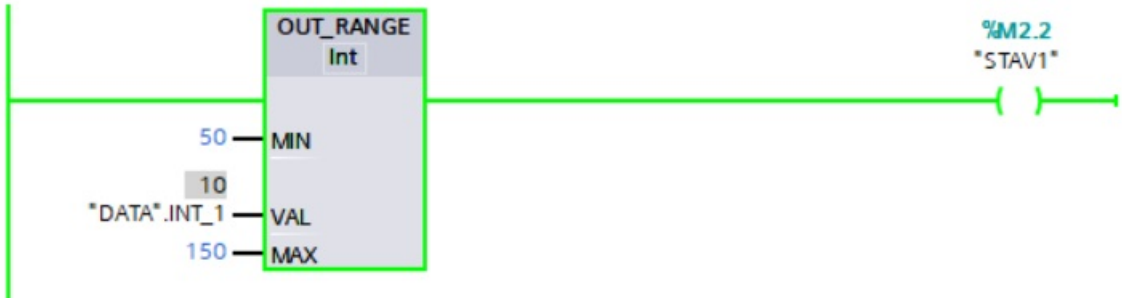
Obr. 3.189. Ekvivalentné riešenie

3.5.8 Hodnota mimo rozsah (OUT_RANGE)

Opakom inštrukcie *IN_RANGE* je inštrukcia *OUT_RANGE*. Symbol inštrukcie je na Obr. 3.190. Ak je hodnota VAL<MIN alebo VAL>MAX, t. j. hodnota VAL nie je v rozsahu <MIN, MAX>, výstupom inštrukcie je TRUE. Príklad monitorovania premenných je na Obr. 3.191. Ekvivalentné príklady sú na Obr. 3.192 a 3.193. Prvý je riešený pomocou základných porovnávacích inštrukcií. Druhý využíva negovanú hodnotu výstupu inštrukcie *IN_RANGE*.

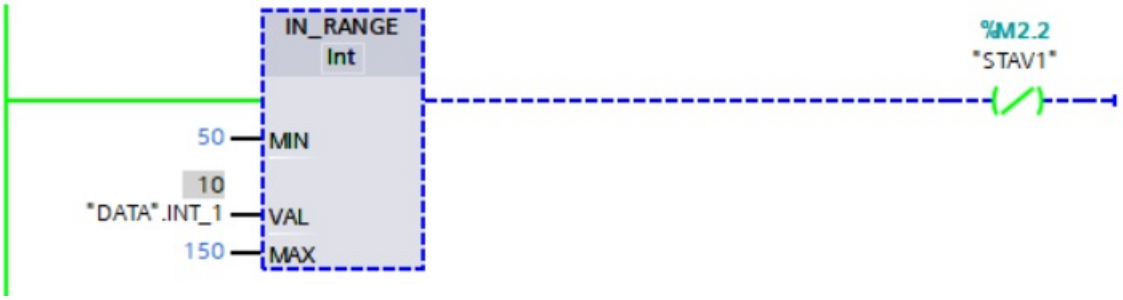


Obr. 3.190. Symbol inštrukcie *OUT_RANGE*



Obr. 3.191. Symbol inštrukcie *OUT_RANGE*

Obr. 3.192. 1. ekvivalentný program

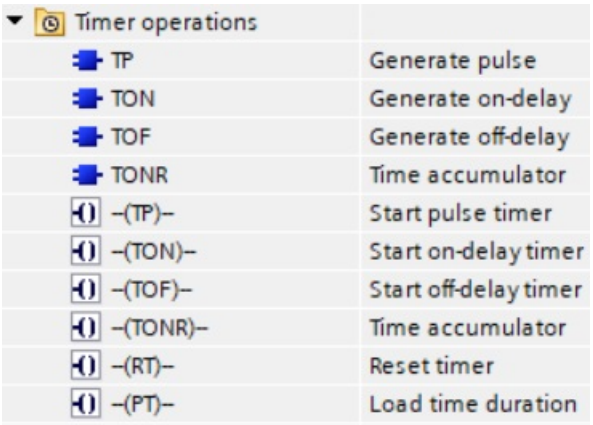


Obr. 3.193. 2. ekvivalentný program

3.6 Časovače (Timer operations)

Časovače patria do skupiny štandardných funkčných blokov, ktoré majú vlastnú inštanciu (dátovú pamäť). Časovače sú dôležité v aplikáciách keď je dôležitý cyklus stroja alebo keď sú potrebné oneskorenia medzi sekvenciami algoritmu. Tieto inštrukcie sú výstupné inštrukcie a vznikli s cieľom nahradenia časových relé.

Zoznam časovačov dostupných v TIA Portal pre CPU S7-1200 je na Obr. 3.194. V tejto kapitole opíšeme prvé štyri inštrukcie z uvedeného zoznamu. Ide o najčastejšie používané časovače v praxi. Ich ekvivalenty nájdeme aj v softvérových nástrojoch iných riadiacich systémov.



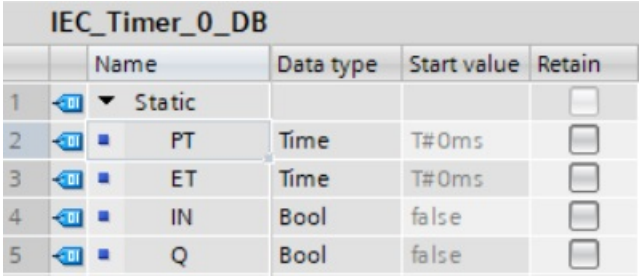
Obr. 3.194. Zoznam inštrukcií časovačov

Časovače a počítadlá používajú vopred definované dátové bloky nezávislé od typu časovača alebo počítadla. Existuje viacero dátových blokov. Najnovšie verzie TIA Portalu pracujú s dátovým typom IEC_TIMER. Okrem toho existujú aj napr. TON_TIME, TOF_TIME a TP_TIME, ktoré boli priradené časovačom *TON*, *TOF* a *TP*. Prístup k štruktúre je cez bodkovú syntax vo forme Názov.Parameter. Na úvod si všeobecne opíšeme dátové štruktúry časovačov a neskôr konkrétne inštrukcie, ktoré využívajú tieto dátové štruktúry. Symbolické názvy dátových blokov sa zadávajú nad symbolom časovača. Štruktúra časovača (Obr. 3.195) je takáto:

- PT - Preset Time - prednastavený čas dátového typu TIME, ktorý má časovač dosiahnuť počas časovania.
- ET - Elapsed Time - aktuálna hodnota časovača. Jej hodnota sa navyšuje od 0ms po hodnotu PT počas časovania. Hodnota sa často využíva na časové spínanie udalostí.
- IN - Input - Vstup ku ktorému sa pripája logický signál opúšťajúci časovač. Význam je závislý od konkrétneho typu časovača.
- Q - Output - Výstup časovača. Význam je závislý od konkrétneho typu časovača.

V starších verziách TIA Portalu boli dostupné premenné:

- ST - Uložený čas CPU pri spustení časovača. Uplynutý čas sa počíta podľa vzťahu ET = aktuálny čas CPU (v čase dopytovania hodnoty z inštančného dátového bloku časovača) - ST.
- RU - Indikácia časovania. V stave TRUE bol časovač spustený.



Obr. 3.195. Inštančný dátový blok inštrukcie *TON*

3.6.1 Časovač s oneskoreným zapnutím (TON)

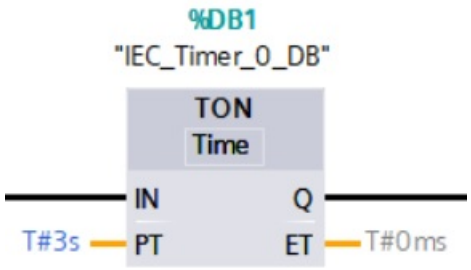
Časovač *TON* (Obr. 3.196) je neretečný časovač, ktorý pri zmene vstupnej podmienky na vstupe IN z FALSE na TRUE spustí časovanie. Pri platnej vstupnej podmienke (IN = TRUE) sa navyšuje čas ET od 0ms po prednastavený čas PT. Stavy v dátovom bloku časovača (Q a ET) sa aktualizujú len v prípade použitia stavov časovača v programe! V opačnom prípade nedôjde k „spusteniu“ časovača aj keď je vstup IN v stave TRUE (Obr. 3.197). Po spustení časovača a dosiahnutí prednastaveného času sa nastaví výstup Q na TRUE a časovač prestane počítať. Aktuálna hodnota ET bude rovná presne hodnote PT a výstup Q sa drží na hodnote TRUE. Časovač TON sa volá „Timer On Delay“ preto lebo oneskorí „zapnutie“ signálu Q o čas PT. Ak sa počas časovania alebo po ukončení časovania privedie na vstup IN časovača signál FALSE, tak sa časovač vynuluje (t. j. Q = FALSE, ET = 0ms). Časový diagram stavov časovača je zobrazený na Obr. 3.198. V prvej časti je zobrazený vstupný signál IN, ktorý sa aktivuje v rôznych časových úsekoch. V strednej časti vidíme priebeh ET. Ide o čas, ktorý uplynul od aktivácie vstupu. V spodnej časti je výstup časovača Q, ktorý sa nastaví na logickú 1 až vtedy, keď ET dosiahne prednastavený čas PT (v tomto prípade 2 sekundy).

Na začiatku sa IN aktivuje na 5 sekúnd (od 1 s do 6 s), čo je dlhšie ako PT, preto ET rastie až na 2 sekundy a potom sa aktivuje aj Q. Keď IN znovu klesne na 0 (v čase 6 s), výstup Q sa okamžite vynuluje.

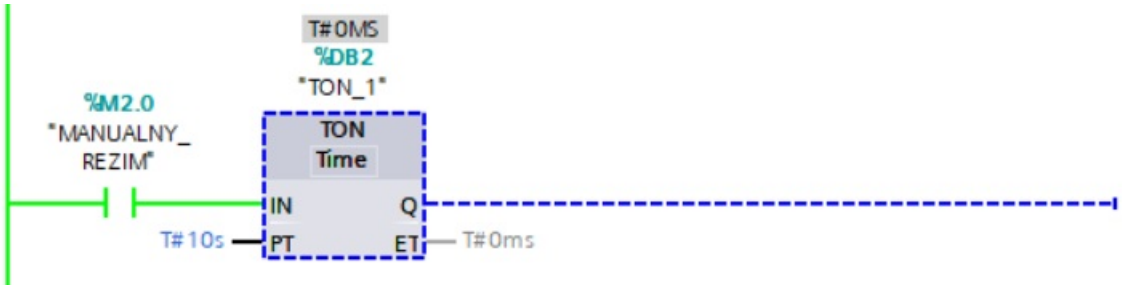
V druhej situácii sa IN aktivuje iba na 1 sekundu (od 10 s do 11 s), čo je menej než PT, takže ET nestihne dosiahnuť 2 sekundy a výstup Q sa neaktivuje.

V tretej situácii (od 14 s do 18 s) je IN opäť aktívny dostatočne dlho, ET narastie na PT a Q sa aktivuje po 2 sekundách. Keď IN klesne na 0, Q sa okamžite vypne.

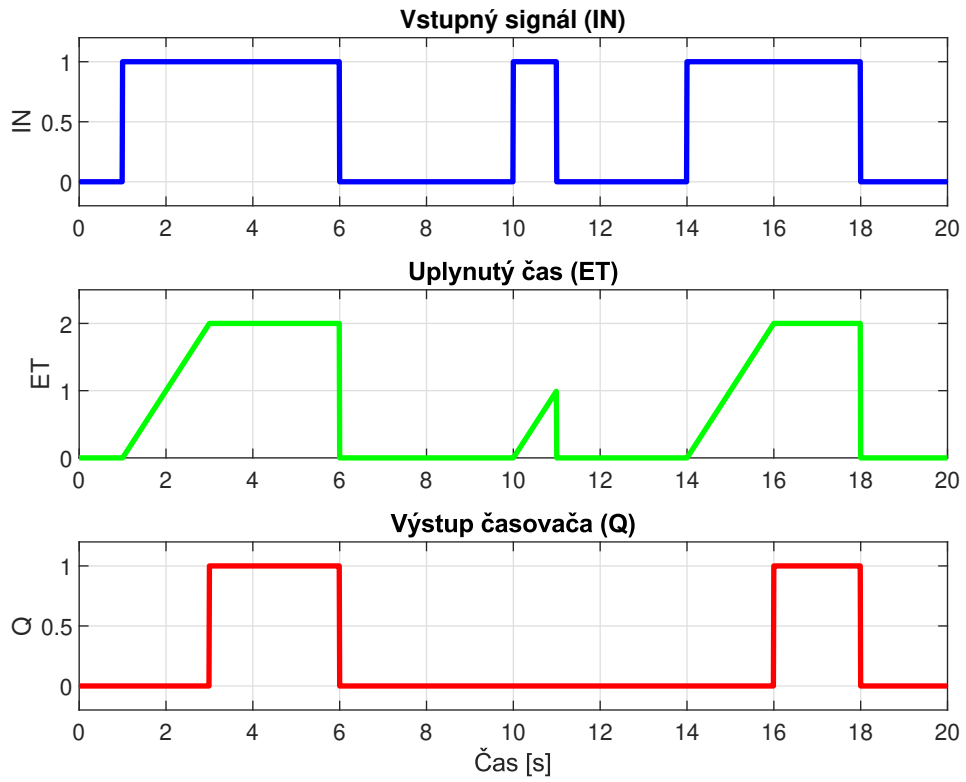
Tento graf demonštruje typické správanie TON časovača. Jeho výstup sa aktivuje až po oneskorení a len v prípade, že vstup IN zostane aktívny po celý tento čas.



Obr. 3.196. Symbol inštrukcie *TON*



Obr. 3.197. Nesprávne použitie inštrukcie *TON*



Obr. 3.198. Časový diagram stavov časovača *TON*

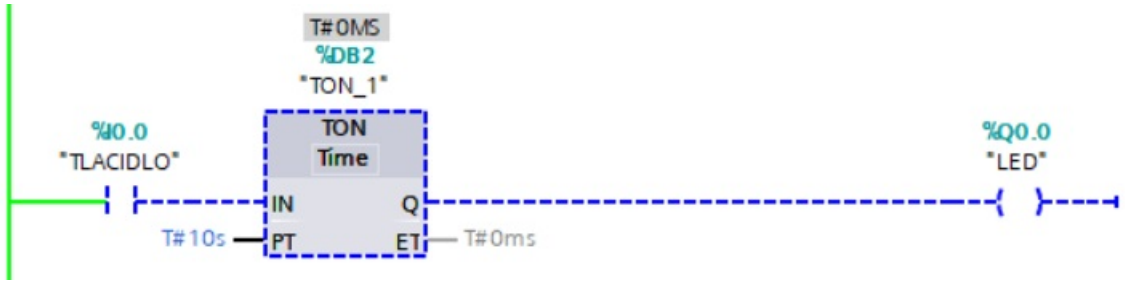
Na Obr. 3.199) až 3.201) sú zaznamenané fázy monitorovaných stavov jednoduchého príkladu. Signál TLACIDLO má oneskorene zapnúť výstup LED.

Na Obr. 3.199 je na vstup IN privedený stav FALSE. Časovač je resetnutý (Q = FALSE, ET = T#0s), preto hodnota LED je FALSE. Aktuálny čas ET = T#0ms je animovaný nad symbolom inštrukcie.

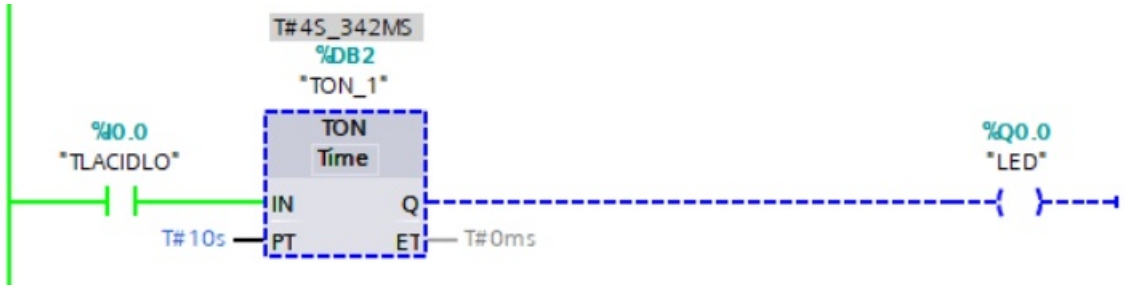
Na Obr. 3.200 je časovač spustený (na vstupe IN je stav TRUE). Uplynutý čas od spustenia časovača sú 4 s a 324 ms. Časovač ešte nedosiahol prednastavenú hodnotu 10 s. Výstup Q je naďalej FALSE, preto sa do LED zapisuje hodnota FALSE.

Na poslednom Obr. 3.201 časovač ukončil časovanie lebo bol dosiahnutý čas PT. Výstup Q je TRUE, preto do LED sa zapisuje hodnota TRUE. Tento stav je nezmenený pokiaľ nezmeníme stav tlačidla na FALSE. Príklad by bol zhodný so stavom na Obr. 3.199.

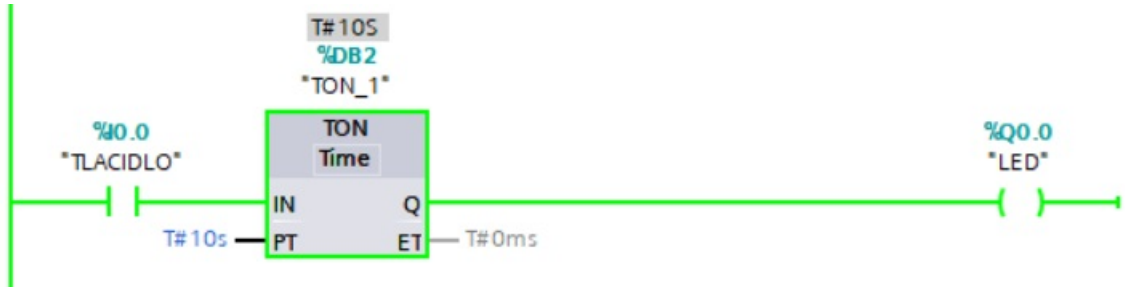
Oneskorené príkazy pripájame za výstup Q symbolu časovača alebo v iných častiach programu testovaním hodnoty výstupu časovača bitovými inštrukciami. Na Obr. 3.202 je v 2. vetve operandom inštrukcie *spínací kontakt* premenná TON_1.Q. Ide o premennú z inštančného dátového bloku časovača TON_1.



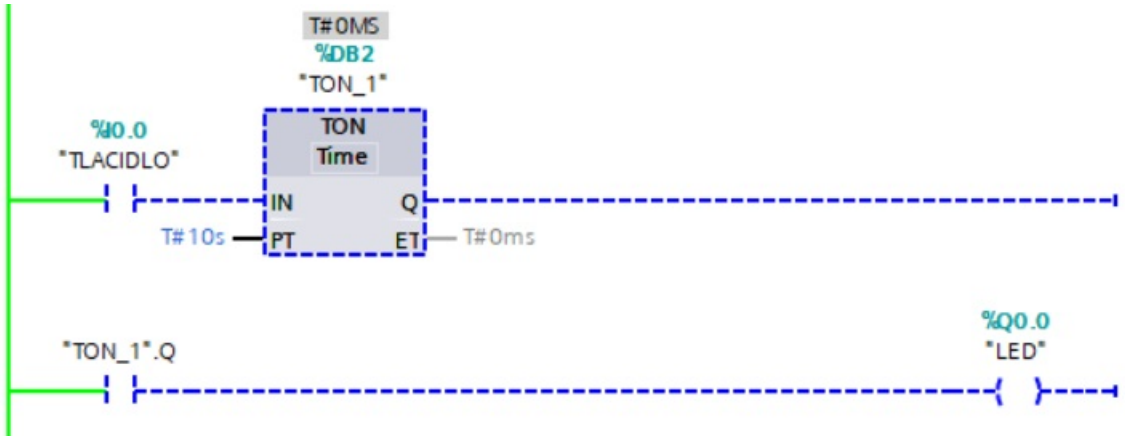
Obr. 3.199. 1. fáza monitorovania program s časovačom *TON*



Obr. 3.200. 2. fáza monitorovania program s časovačom *TON*



Obr. 3.201. 3. fáza monitorovania program s časovačom *TON*



Obr. 3.202. Príklad využitia výstupu Q bitovou inštrukciou

K premenným v dátových blokoch môžeme pristupovať aj viackrát. Príklad využitia uplynutého času časovača je na Obr. 3.203 až 3.205.

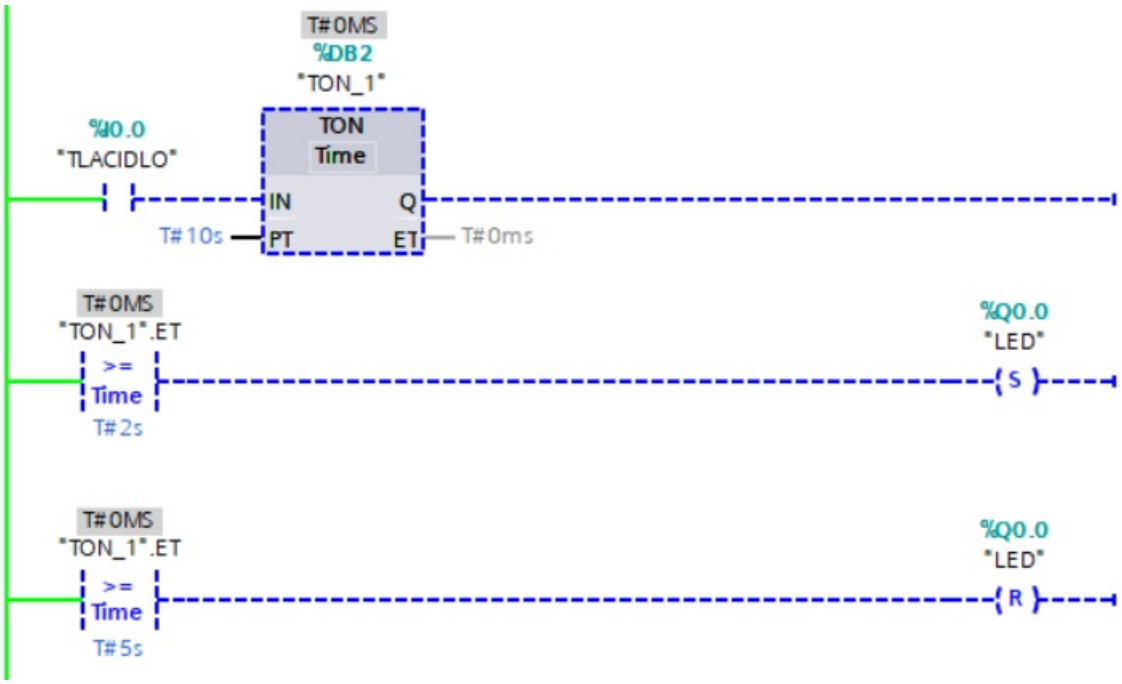
Na Obr. 3.203 je časovač vynulovaný. Aktuálna hodnota ET je 0 ms. Vo vetvách č. 2 a 3 sa aktuálny čas časovača porovnáva s konštantnými hodnotami. Ani jedna z podmienok nie je platná. Príkazy *SET* a *RESET* sa nevykonávajú.

Na Obr. 3.204 je uplynutý čas 2 s 951 ms. Výsledok prvého porovnávacieho bloku je platný. Inštrukcia *SET* nastavila hodnotu LED na TRUE.

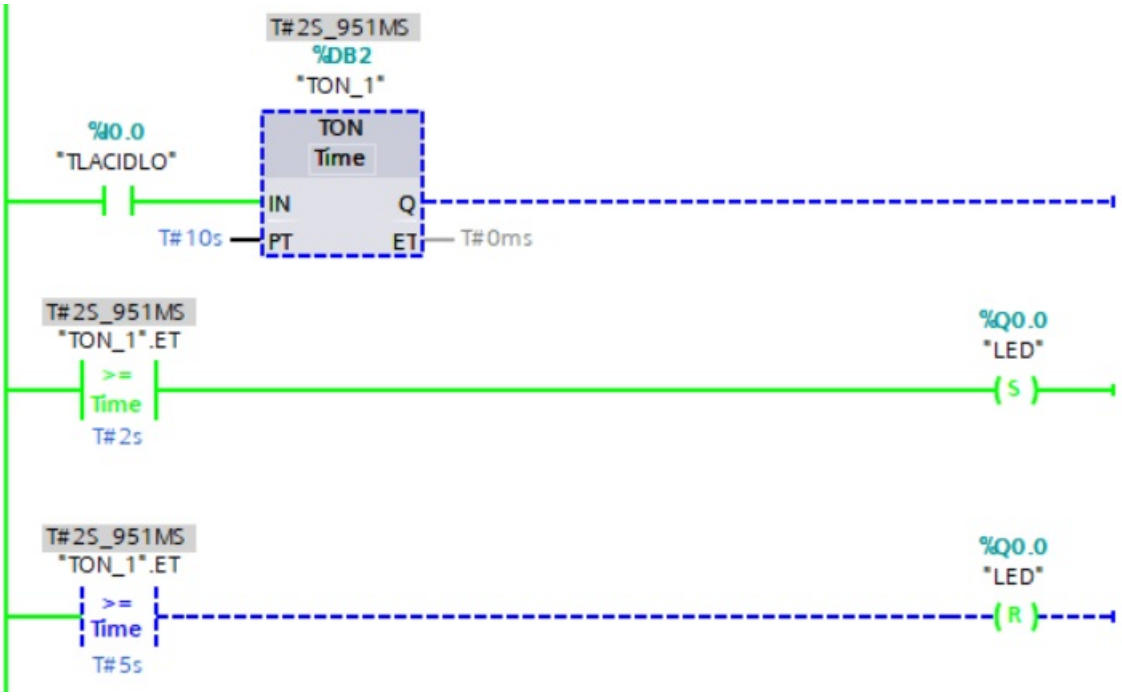
Na Obr. 3.205 je uplynutý čas 5 s 365 ms. Výsledok prvého porovnávacieho bloku je platný. Inštrukcia *SET* nastavila hodnotu LED na TRUE. Vzápätí od 5 s platí aj výsledok druhého porovnávacieho bloku. Inštrukcia *RESET* prepisuje hodnotu LED na FALSE.

Funkcionalitu programu môžeme zhrnúť takto: Ak tlačidlo nie je zatlačené, výstup nie je zapnutý. Ak tlačidlo stlačíme dlhodobo (napr. 10 s), LED sa zapne po uplynutí 2 s na 3 s. Akonáhle uvoľníme tlačidlo, LED ostáva v poslednom stave, podľa toho ktorá z inštrukcií *SET* alebo *RESET* sa vykonávala:

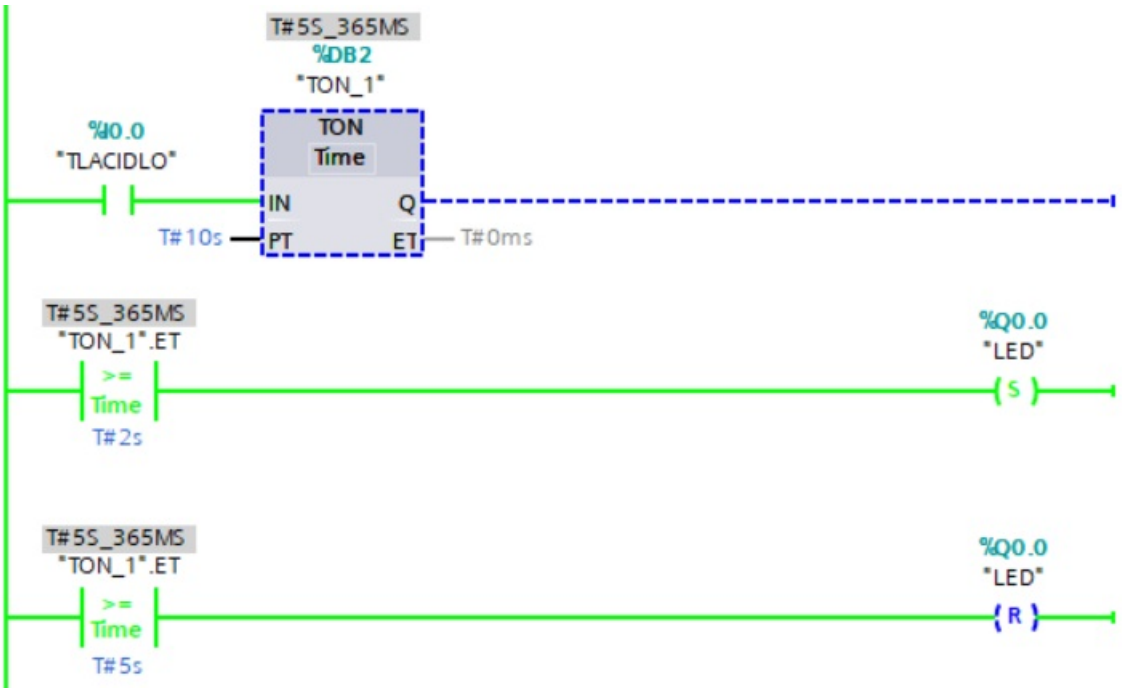
- Ak sa uvoľní v čase do 2 s od stlačenia, LED je zhasnutá (inštrukcie *SET* a *RESET* ešte neboli vykonané).
- Ak sa uvoľní v intervale 2 až 5 s od stlačenia, LED bude svietiť lebo sa inštrukcia *SET* vykonávala a *RESET* ešte nie.
- Ak sa uvoľní po 5 s od stlačenia, LED bude zhasnutá lebo sa vykonali *SET* aj *RESET*, ale *RESET* prepísal stav LED na FALSE.



Obr. 3.203. 1. fáza monitorovania programu s časovačom *TON*



Obr. 3.204. 2. fáza monitorovania programu s časovačom *TON*



Obr. 3.205. 3. fáza monitorovania programu s časovačom *TON*

3.6.2 Oneskorené vypnutie (TOF)

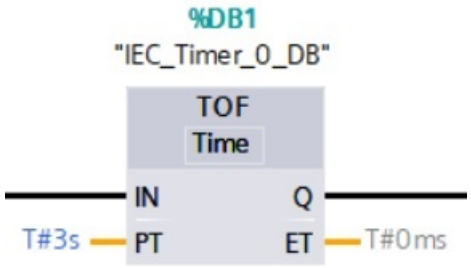
Symbol inštrukcie *TOF* je na Obr. 3.206. Inštrukcia *TOF* je neretenčný časovač, ktorý sa spustí pri zmene vstupného signálu z TRUE na FALSE, ktorý ostáva v stave FALSE. Časovač TOF sa preto volá „Timer Off Delay“, lebo po uplynutí prednastaveného času sa

bit Q, ktorý mal dovtedy hodnotu TRUE zmení na FALSE, t. j. výstup Q sa „vypne“. Platí to isté ako v prípade časovača TON, že je potrebné použiť dátový blok časovača, ktorý sa aktualizuje pri jeho prístupe k premenným (Q alebo ET). Časový diagram časovača TOF je na Obr. 3.207. Všimnite si, že ak je na vstupe IN hodnota TRUE, výstup Q má hodnotu TRUE. Ide o resetnutý stav časovača (Q = TRUE, ET = 0 ms). Neretenčné premenné resp. neretenčné dátové bloky sa pri prepnutí CPU z režimu STOP do režimu RUN vynulujú.

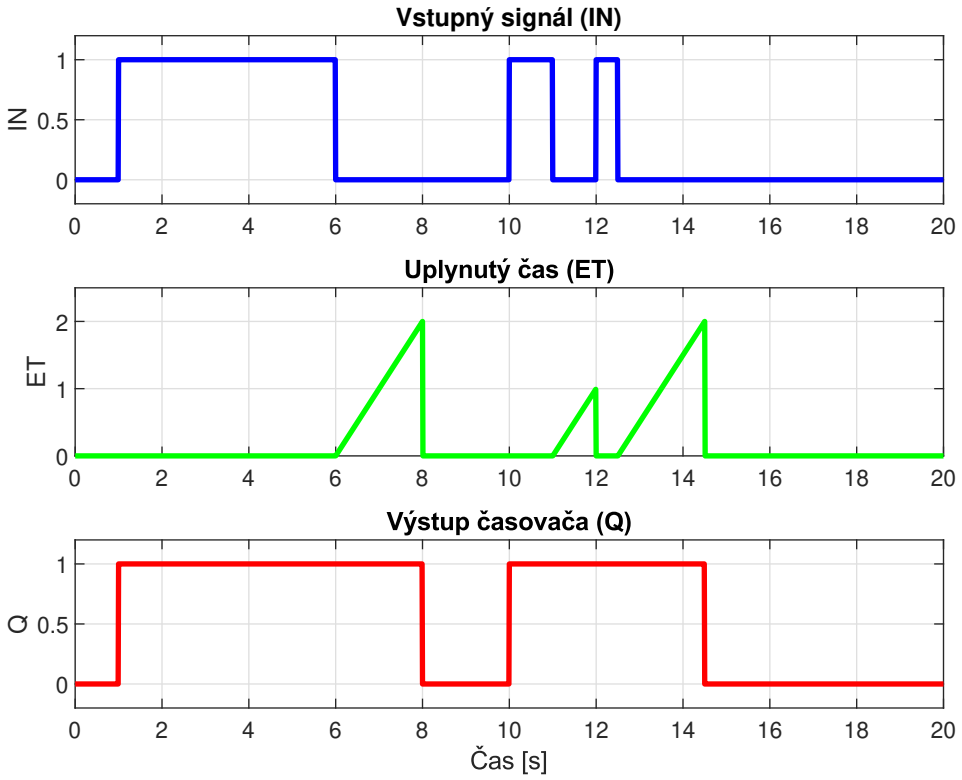
V hornej časti Obr. 3.207 je vstupný signál IN, ktorý sa počas simulácie niekoľkokrát aktivuje a deaktivuje. V strede vidíme priebeh ET, ktorý zobrazuje trvanie oneskorenia po vypnutí vstupu. V spodnej časti je výstupný signál Q, ktorý reaguje s oneskorením po zmene IN z 1 na 0.

V prvej situácii (čas 1 s – 6 s) je IN aktívny počas 5 sekúnd, počas ktorých je Q tiež aktívny. Keď sa IN zmení na 0, časovač začne počítať čas po prednastavený čas PT = 2 s. Počas týchto 2 sekúnd zostáva výstup Q ešte aktívny. Po uplynutí oneskorenia sa Q vypne.

V druhej situácii (10 s – 11 s) sa IN aktivuje krátko, potom sa deaktivuje a TOF začne odpočítavať. Ale ešte predtým, než uplynie PT, sa IN opäť aktivuje (v čase 12 s). Tým sa časovanie preruší a Q ostáva 1. Výstup sa nevypne, pretože časovač bol resetovaný v dôsledku novej aktivácie vstupu.



Obr. 3.206. Symbol inštrukcie *TOF*



Obr. 3.207. Časový diagram časovača *TOF*

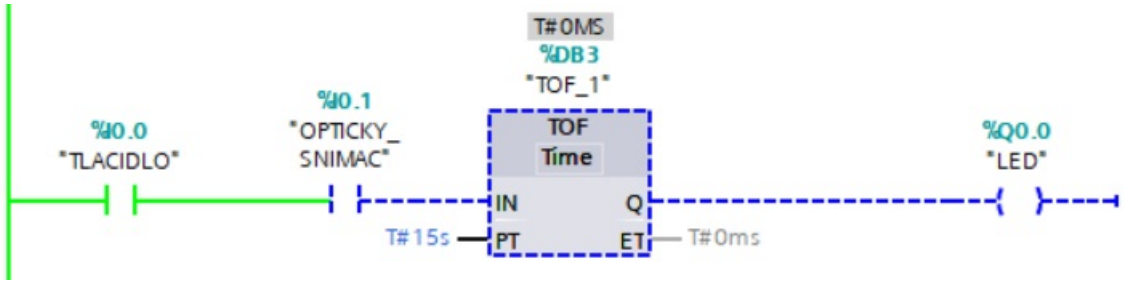
Na Obr. 3.208 až 3.211 sú zobrazené rôzne fázy monitorovania hodnôt premenných programu.

Na Obr. 3.208 je stav logického spojenia AND na vstupe IN časovača FALSE. Časovač nie je spustený, lebo doposiaľ vstup IN nemal hodnotu TRUE. Výstup Q je FALSE.

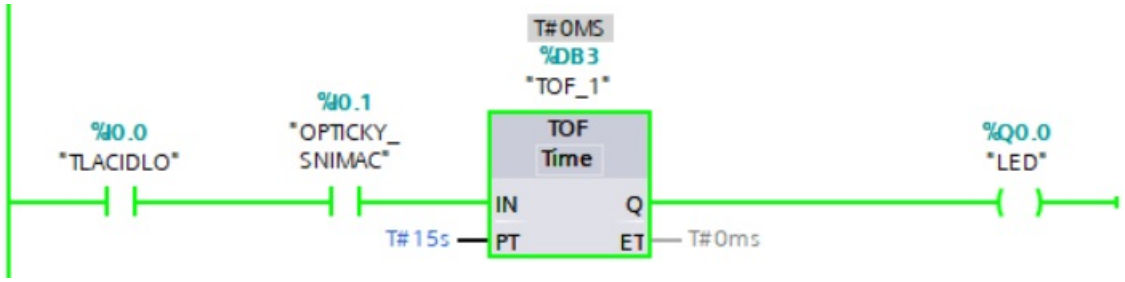
Na druhom Obr. 3.209 je na vstupe IN hodnota TRUE. Výstup Q tiež TRUE. Časovač stále nie je spustený. Čaká sa na zmenu stavu z TRUE na FALSE na vstupe IN.

Na ďalšom Obr. 3.210 sa vstupný signál na vstupe IN zmenil z TRUE na FALSE. Časovač je spustený. Aktuálny uplynutý čas je 3 s 204 ms. Výstup Q je naďalej v stave TRUE.

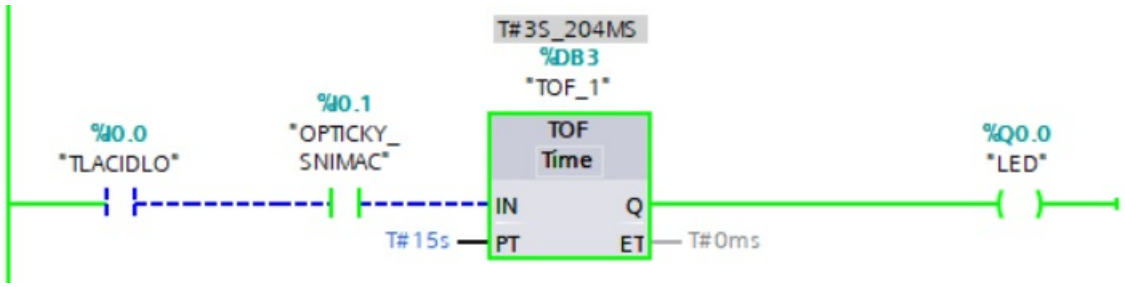
Na poslednom Obr. 3.211 uplynul prednastavený čas 15 s. Výstup Q sa zmenil na FALSE. Došlo k tzv. oneskorenému „vypnutiu“. Hodnota ET naďalej ostáva 15 s. Všimnite si rozdiel medzi Obr. 3.208 a 3.211. V oboch prípadoch je na vstupe IN stav FALSE a výstup Q je tiež FALSE. Rozdiel je v hodnote ET.



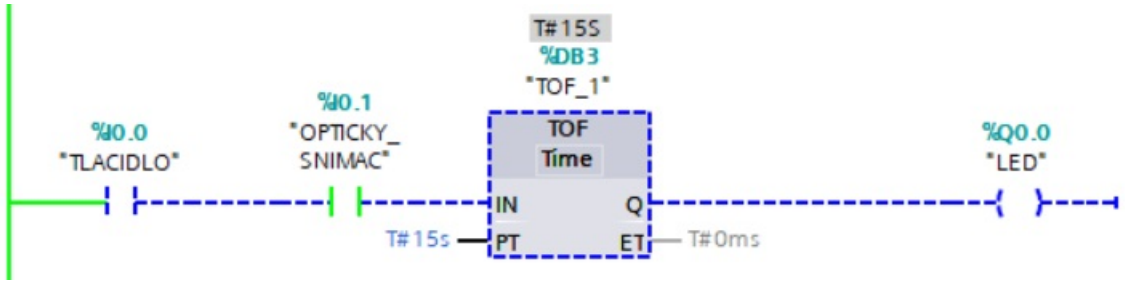
Obr. 3.208. 1. fáza monitorovania programu s časovačom *TOF*



Obr. 3.209. 2. fáza monitorovania programu s časovačom *TOF*



Obr. 3.210. 3. fáza monitorovania programu s časovačom *TOF*

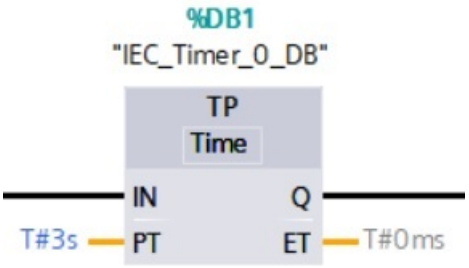


Obr. 3.211. 4. fáza monitorovania programu s časovačom *TOF*

3.6.3 Pulzný časovač (TP)

Symbol inštrukcie *TP* je na Obr. 3.212. Inštrukcia *TP* je neretenčný časovač, ktorý sa spustí pri zmene vstupného signálu z FALSE na TRUE za predpokladu, že časovač nie je

spustený. Svoju činnosť dokončí aj pri zmene vstupného signálu na FALSE. Počas spusteného časovača je na výstupe Q = TRUE, inak Q = FALSE. Zjednodušene povedané pri nábežnej hrane na vstupe IN časovač vygeneruje pulz dĺžky PT na výstupe Q. Na Obr. 3.213 sa nachádza časový diagram časovača PT.



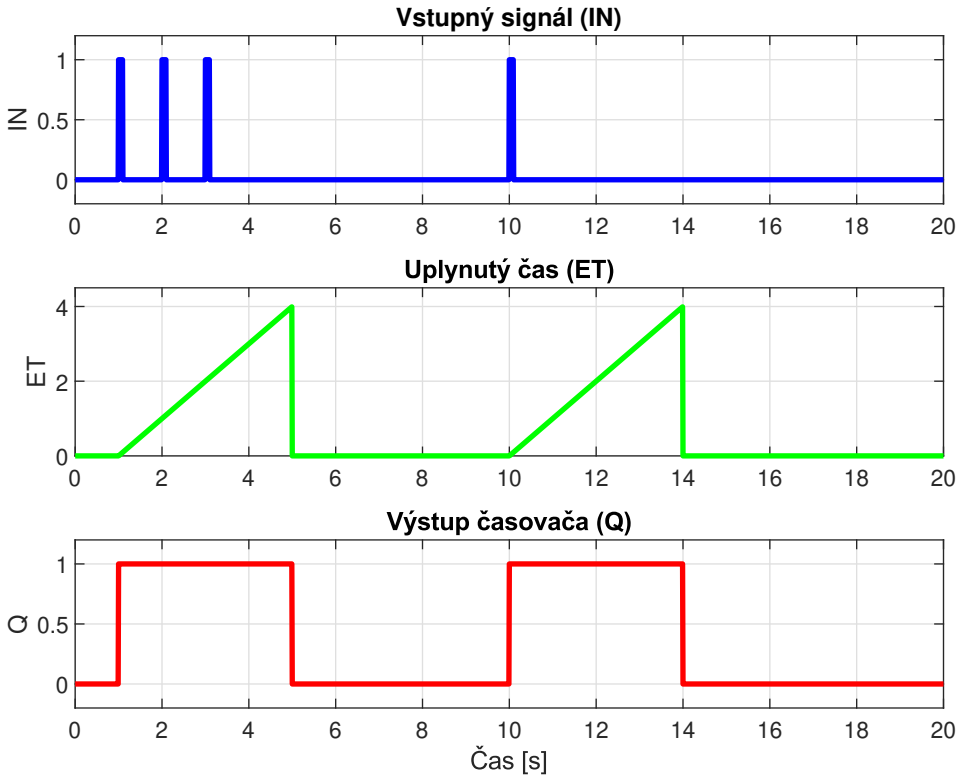
Obr. 3.212. Symbol inštrukcie *TP*

V hornej časti Obr. 3.213 je zobrazený vstupný signál IN, v strede vnútorný časovač ET a v spodnej časti výstup Q. TP časovač reaguje na nábežnú hranu vstupného signálu, t. j. zmenu z 0 na 1. Pri nábehovej hrane sa výstup Q nastaví na 1 a zostáva aktívny počas vopred definovaného času PT, bez ohľadu na to, ako dlho zostáva IN aktívny.

V prvej časti simulácie sa IN aktivuje krátkym impulzom v čase 1 sekunda. Na túto nábehovú hranu TP zareaguje – výstup Q sa nastaví na logickú 1 a časovač začína odpočítavať. Počas 4 sekúnd rastie hodnota ET až do dosiahnutia hodnoty PT, potom sa výstup automaticky vypne.

Počas tohto aktívneho časovania (od 1 s do 5 s) prichádzajú ďalšie dva impulzy na vstupe (v čase 2 s a 3 s). TP časovač však tieto impulzy ignoruje, pretože počas aktívneho výstupu nereaguje na ďalšie nábehové hrany. Výstup Q tak zostáva nezmenený a dobehne pôvodné časovanie.

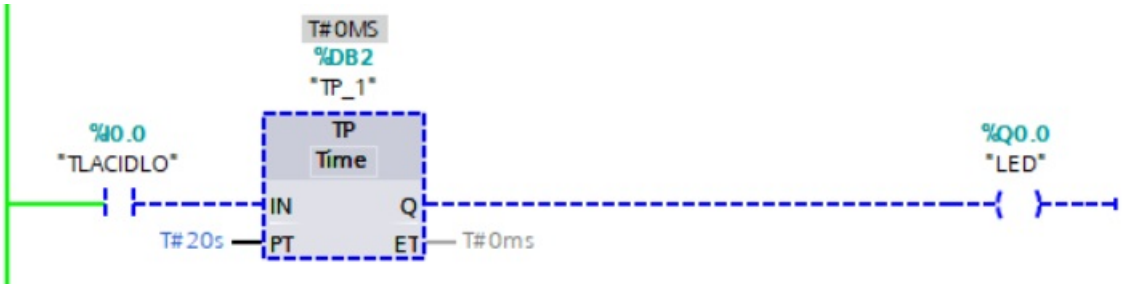
V druhej časti simulácie (čas 10 s) prichádza ďalší impulz IN, keď už časovač nie je aktívny. TP opäť reaguje – výstup Q sa nastaví na 1 a prebehne rovnaký časovací cyklus, ako v prvej časti.



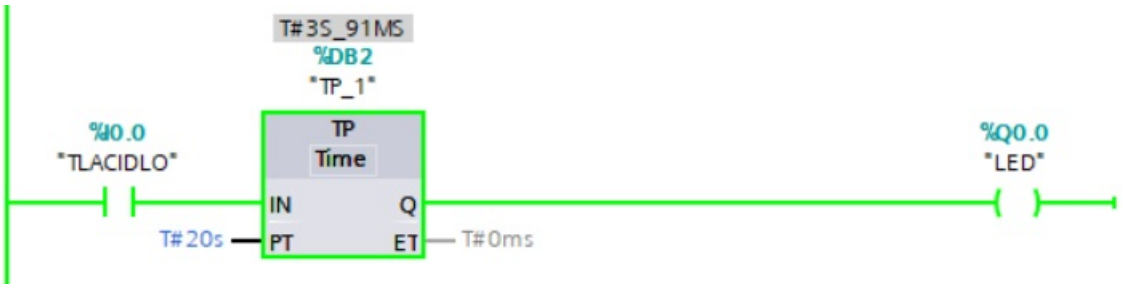
Obr. 3.213. Časový diagram časovača *TP*

Na Obr. 3.214 až 3.218 sú rôzne fázy monitorovania programu:

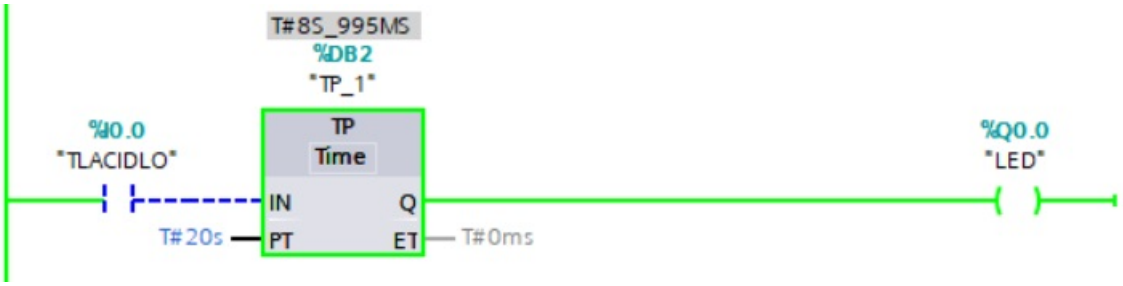
- Na Obr. 3.214 je na vstupe IN stav FALSE. Časovač ešte nebol spustený (pozri hodnotu ET nad symbolom inštrukcie). Výstup časovača Q je v stave FALSE.
- Zmenou vstupu IN z FALSE na TRUE bola na vstupe IN detegovaná nábežná hrana, ktorá spustila časovač (Obr. 3.215). Výstup Q je v stave TRUE počas doby PT bez ohľadu na stav vstupu IN.
- Na Obr. 3.216 sa na vstupe IN zmenila hodnota na FALSE. Časovač naďalej počíta a výstup je v stave TRUE.
- Na ďalšom obrázku (Obr. 3.217) sa zmenil vstup opäť na TRUE. Nová nábežná hrana na vstupe IN nespôsobila žiadnu zmenu časovania.
- Na poslednom Obr. 3.218 časovač dokončil časovanie (výstup Q je v stave FALSE a ET časovača sa rovná hodnote PT). Na opätovné spustenie časovača musíme priviesť na vstup IN novú nábežnú hranu, t. j. vstup IN sa musí aspoň na jeden cyklus zmeniť na FALSE a aspoň na jeden cyklus zmeniť na TRUE.



Obr. 3.214. 1. fáza monitorovania stavov inštrukcie *TP*

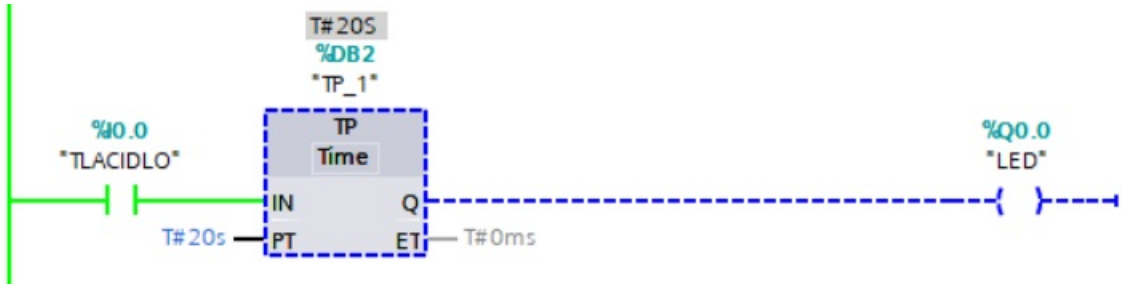


Obr. 3.215. 2. fáza monitorovania stavov inštrukcie *TP*



Obr. 3.216. 3. fáza monitorovania stavov inštrukcie *TP*

Obr. 3.217. 4. fáza monitorovania stavov inštrukcie *TP*

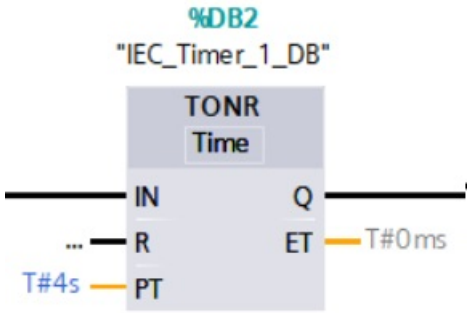


Obr. 3.218. 5. fáza monitorovania stavov inštrukcie *TP*

3.6.4 Retenčné oneskorené zapnutie (TONR)

Inštrukcia *TONR* je retenčný časovač (Obr. 3.219), ktorý sa spustí pri zmene vstupného signálu z FALSE na TRUE. Časovač *TONR* sa líši od časovača *TON* tým, že po zmene vstupného signálu z TRUE na FALSE sa časovač nevynuluje. Časovač si pamätá aktuálnu hodnotu a pri jeho opätovnom spustení pokračuje v navyšovaní ET od aktuálnej hodnoty po prednastavenú. Vstup R (RESET) slúži na vynulovanie stavov časovača (ET = 0 ms, Q = FALSE). Ak je na vstupoch IN a R stav TRUE súčasne, tak má prioritu operácia RESET. Ak oba vstupy sú v stave TRUE a R sa zmení na FALSE, tak sa časovač nespustí lebo na spustenie časovača je potrebná zmena na vstupe IN z FALSE na TRUE. Časový diagram časovača *TONR* je na Obr. 3.220. Retenčné premenné, resp. retenčné dátové bloky si uchovávajú svoje hodnoty aj po prepnutí CPU z režimu STOP do RUN.

Pravdepodobne najčastejšie nasadeným časovačom v praxi je *TON*. Ak sa zmení stav podmienky na vstupe IN na FALSE (napr. snímač krátkodobo zmení svoj stav, zareaguje motorová ochrana, vznikne alarm, prestane platiť bezpečný stav zariadenia a pod.) časovač *TON* sa vynuluje. Pri obnovení spúšťacej podmienky časovača začne počítanie času od začiatku. Inštrukcia *TONR* je vhodnou náhradou v takýchto prípadoch.



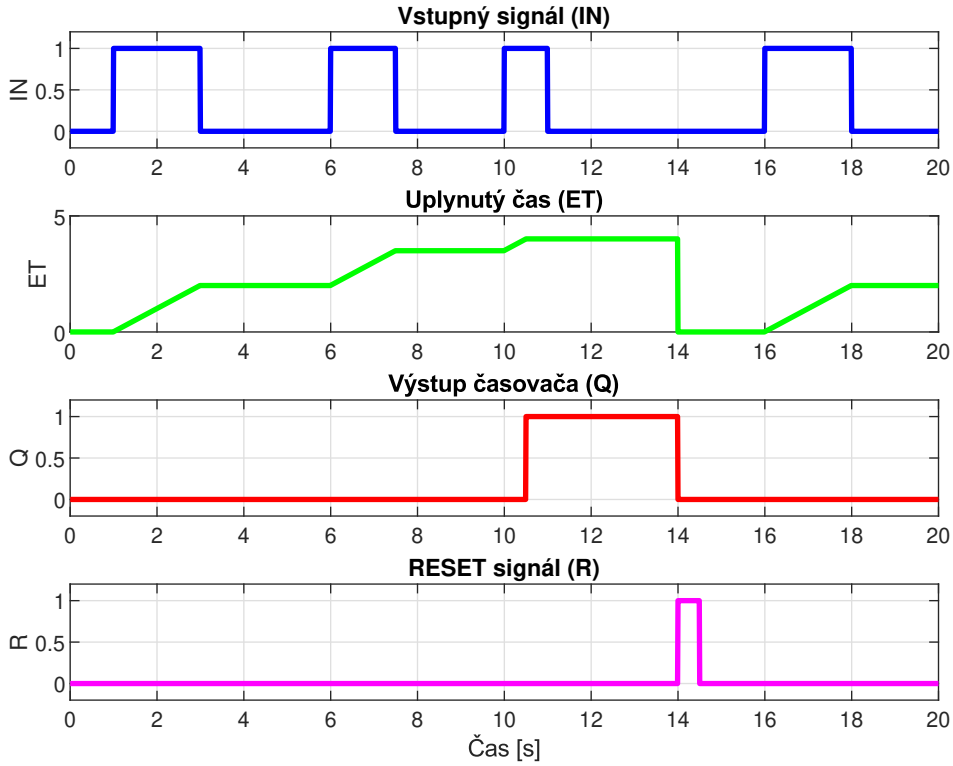
Obr. 3.219. Symbol inštrukcie *TONR*

Graf nižšie znázorňuje správanie TONR časovača. V prvej časti je zobrazený vstupný signál IN, ktorý sa aktivuje v niekoľkých krátkych intervaloch. V druhej časti vidíme priebeh uplynutého času ET. Na rozdiel od klasického TON, ET sa tu nevracia na nulu, keď sa IN zmení na FALSE, ale si hodnotu pamätá. V tretej časti je výstup časovača Q, ktorý sa aktivuje až po tom, čo ET dosiahne nastavený prednastavený čas PT (tu 4 sekundy). V štvrtej časti je zobrazený resetovací signál R.

V prvej fáze je IN aktívny v intervaloch 1–3 s, 6–7,5 s a 10–11 s. Každý z týchto intervalov je kratší ako PT, takže výstup Q sa v týchto chvíľach neaktivuje. Avšak ET si pamätá celkový čas, ktorý IN bol aktívny – postupne narastá z 2 s na 3,5 s a nakoniec prekročí 4 s, čím sa aktivuje Q.

V čase 14–14.5 s sa aktivuje resetovací signál R, ktorý okamžite vynuluje ET aj Q. To znamená, že časovač sa úplne resetuje (Q=FALSE, ET=0 ms).

Po resete je IN opäť aktívny v intervale 16–18 s a časovač začne odpočítavať od nuly. ET sa opäť zvyšuje počas aktivácie IN a výstup Q by sa aktivoval po dosiahnutí PT.

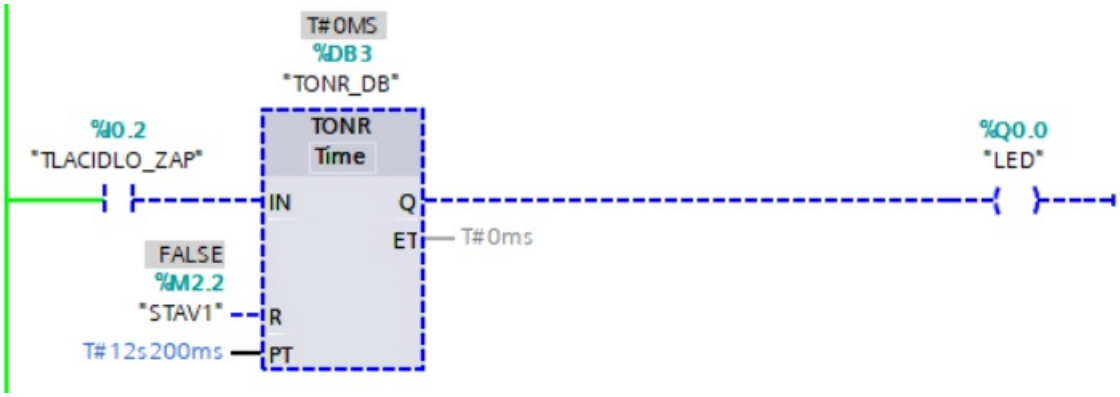


Obr. 3.220. Časový diagram časovača *TONR*

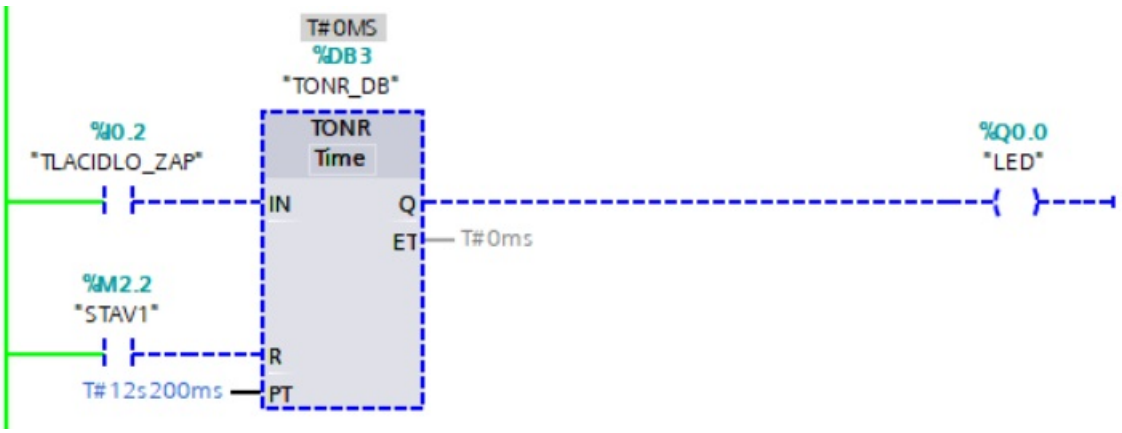
Na Obr. 3.221 je príklad použitia inštrukcie *TONR*. Na vstup R sa priradila premenná STAV1 bez nutnosti vkladania bitových inštrukcií. Ekvivalentný program je na Obr. 3.222. V tomto príklade sa na spracovanie stavu premennej STAV1 použila inštrukcia *normálne otvorený kontakt*. Bitové inštrukcie sa používajú v prípade potreby návrhu zložitejších podmienok ako je stav jednej premennej. Rôzne fázy monitorovania programu sú na Obr. 3.222 až 3.227:

- Na Obr. 3.222 je počiatočný stav. Vstupy a výstupy časovača sú FALSE.
- Na Obr. 3.223 bol na vstup IN privedený signál TRUE. Časovač je spustený podobne ako časovač *TON*. Aktuálny čas časovača je 2 s 808 ms.
- Na Obr. 3.224 bol na vstup IN privedený signál FALSE. Časovač *TONR* na rozdiel od *TON* si zachoval svoje stavy, t. j. ET sa nevynuloval (Q si tiež zachoval svoju hodnotu, doposiaľ je FALSE).

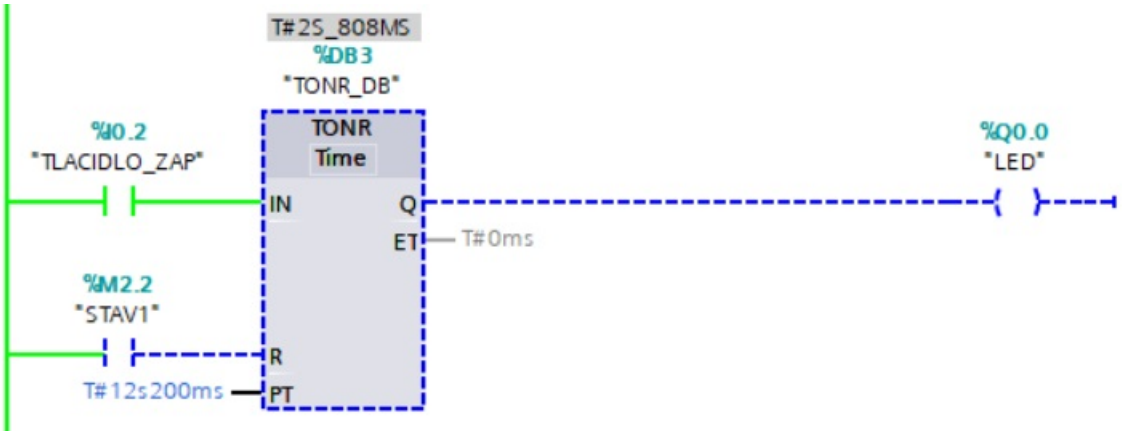
- Na Obr. 3.225 bol časovač opätovne spustený. Pokračoval vo svojej činnosti až dosiahol čas 12 s 200 ms. Výstup Q sa nastavil na TRUE ako pri časovači *TON*.
- Na demonštráciu zachovania stavov časovača bol na Obr. 3.226 zmenený vstup IN na hodnotu FALSE. Hodnoty ET a Q sú nezmenené.
- Na poslednom Obr. 3.227 sa na vstup R priviedol logický stav TRUE. Časovač sa vynuloval (ET = 0 ms, Q = FALSE).



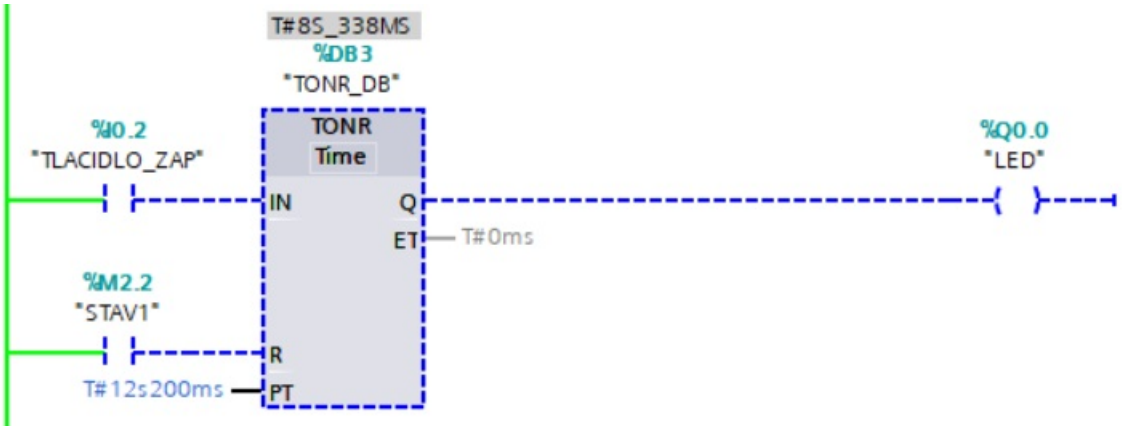
Obr. 3.221. Príklad inštrukcie *TONR*



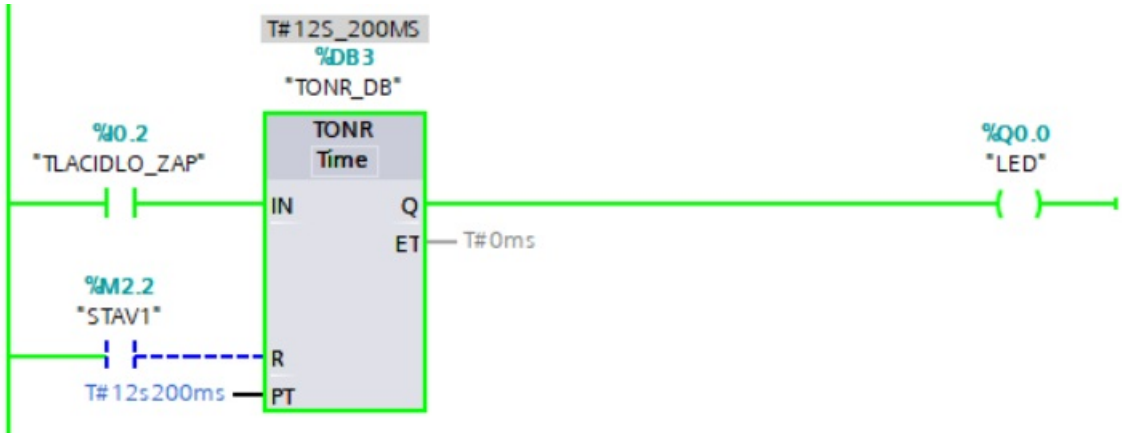
Obr. 3.222. 1. fáza monitorovania programu s *TONR*



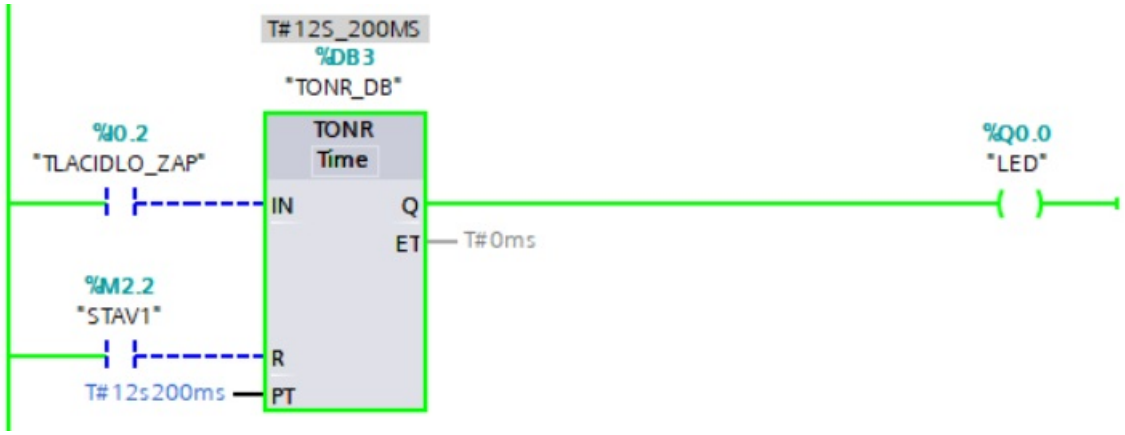
Obr. 3.223. 2. fáza monitorovania programu s *TONR*



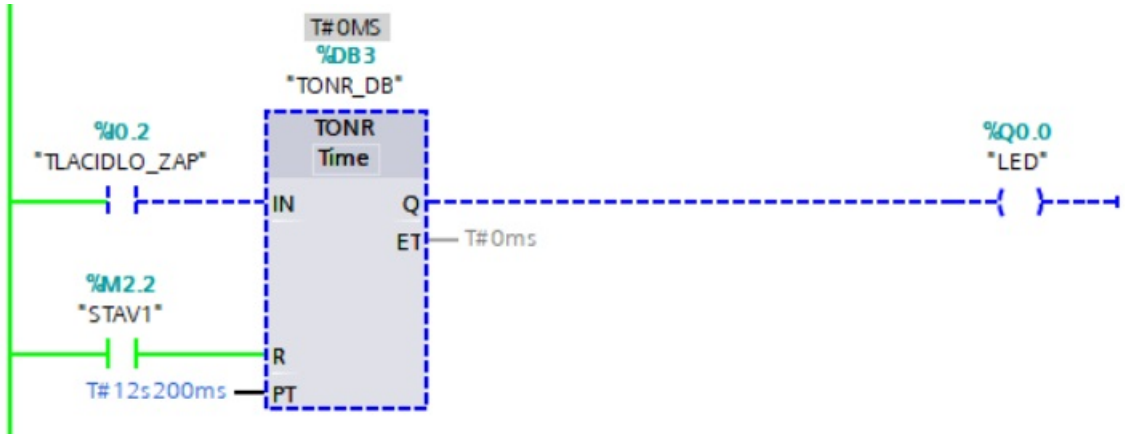
Obr. 3.224. 3. fáza monitorovania programu s *TONR*



Obr. 3.225. 4. fáza monitorovania programu s *TONR*



Obr. 3.226. 5. fáza monitorovania programu s *TONR*



Obr. 3.227. 6. fáza monitorovania program s *TONR*

3.6.5 Reset časovača (RT)

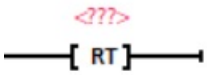
Inštrukcia *RT* (Obr. 3.228) sa používa na vynulovanie (reset) časovača. Nad symbol inštrukcie sa vkladná symbolický názov dátového bloku (alebo fyzická adresa dátového bloku). Inštrukcia sa vykoná len vtedy, ak je výsledok logickej operácie (RLO) na vstupe inštrukcie TRUE. Inštrukcia nemá vplyv na RLO. RLO na vstupe sa posiela priamo na jej výstup.

Na Obr. 3.229 sú za výstup časovača *TON* pripojené dve inštrukcie. Prvou je inštrukcia vynulovania časovača *RT* a druhá *ADD*. Časovač nie je spustený, preto je výstup Q stále FALSE.

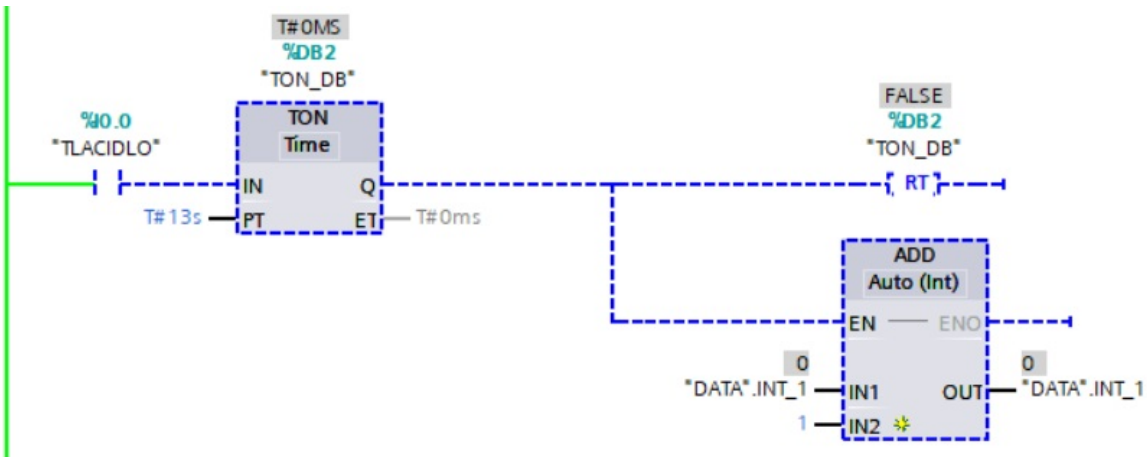
Na Obr. 3.230 je zaznamenaný stav spusteného časovača, ktorý doposiaľ nedosiahol prednastavený čas. Inštrukcie pripojené k výstupu Q neboli vykonané.

Program funguje takto: Ak je tlačidlo stlačené aspoň na 13 s, časovač zmení svoj výstup Q na TRUE a vykonajú sa inštrukcie *RT* a *ADD*. Prvá inštrukcia vynuluje časovač. Z dôvodu vynulovania sa časovač opätovne spustí a začne časovať od 0 ms. Samozrejme za predpokladu, že vstup IN je v stave TRUE. Inštrukciou ADD navyšujeme hodnotu DATA.INT_1. Hodnota v tomto programe znamená počet ukončených časovaní časovačom. Na Obr. 3.231 je hodnota premennej už 2, t. j. práve prebieha 3. cyklus časovania

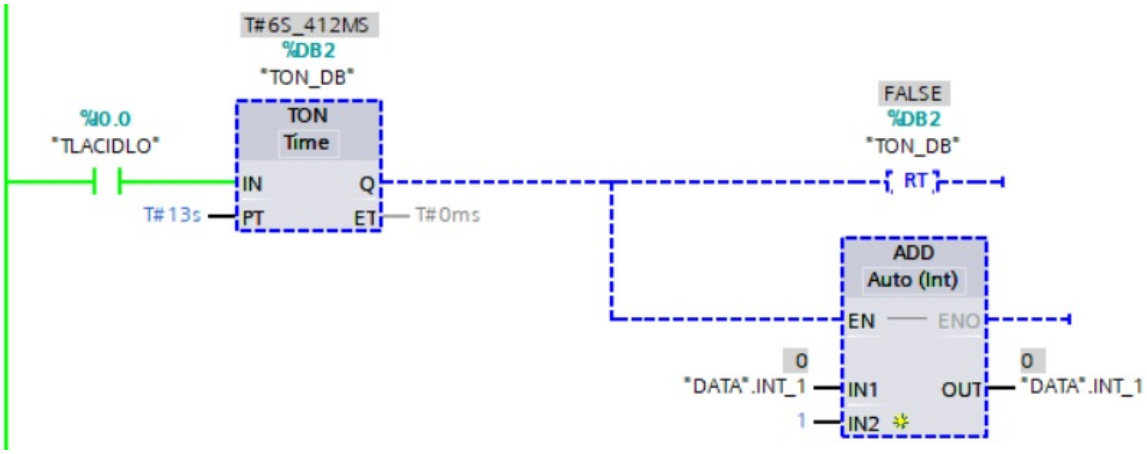
časovačom. Poradie pripojených inštrukcií *RT* a *ADD* by mohlo byť aj v opačnom poradí. To, že sa časovač vynuluje, t. j. hodnota Q sa mení na FALSE sa v RLO zmení až v nasledujúcom cykle (pozri poradie vykonávania inštrukcií v LAD).



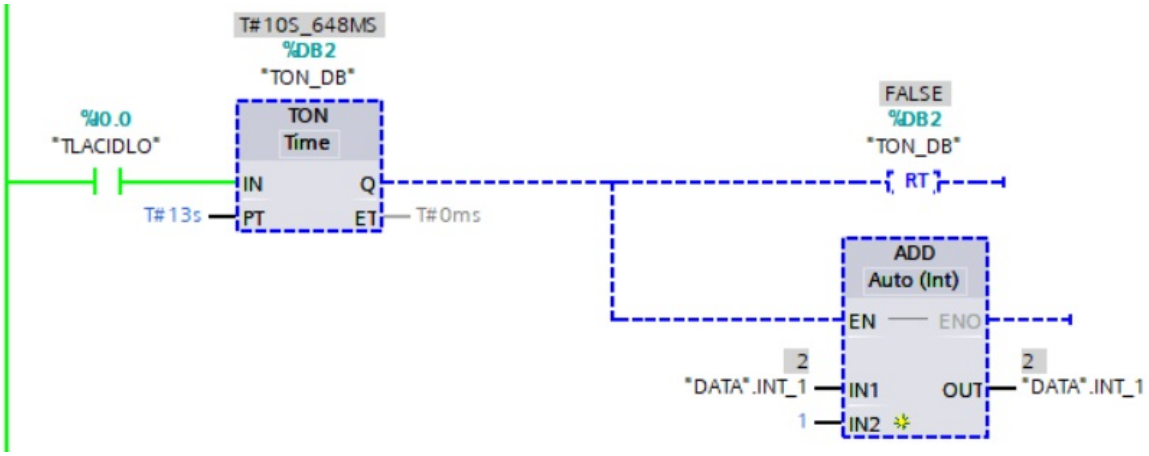
Obr. 3.228. Symbol inštrukcie *RT*



Obr. 3.229. 1. fáza monitorovania programu s inštrukciou *RT*



Obr. 3.230. 2. fáza monitorovania programu s inštrukciou *RT*



Obr. 3.231. 3. fáza monitorovania programu s inštrukciou *RT*

3.7 Počítadlá (Counter operations)

Počítadlá, rovnako ako časovače patria do skupiny štandardných funkčných blokov. Slúžia na počítanie udalostí alebo počtov v procese ako napr. počet výrobkov, porúch, zmien nastavení a iné. Nahrádzajú základné matematické inštrukcie ako *ADD* a *SUB* kvôli ich dobre prepracovanej dátovej štruktúre. Cieľom kapitoly je oboznámiť čitateľa s vybranými počítadlami softvéru TIA Portal v jazyku rebríkových schém. Dátový blok počítadiel je zobrazený na Obr. 3.233. Všimnite si, že premenné v dátovom bloku sú nastavené na retenčné. Význam premenných je takýto:

- CU - vstup (count up), ktorým sa navyšuje hodnota počítadla.
- CD - vstup (count down), ktorým sa dekrementuje hodnota počítadla.
- R - vstup (reset) nulujúci aktuálnu hodnotu počítadla.
- LD - vstup (load), ktorým sa priradí prednastavená hodnota PV do aktuálnej hodnoty CV.
- QU - výstup (q up) indikujúci dosiahnutie prednastavenej hodnoty a vyššej.
- QD - výstup (q down) indikujúci dosiahnutie hodnoty 0 a menšej.
- PV - prednastavená hodnota (preset value), ktorú chceme dosiahnuť počítaním smerom nahor alebo odpočítavanie do 0 začne z tejto hodnoty.
- CV - aktuálna hodnota (curent value) počítadla.

Zoznam dostupných počítadiel v TIA Portal pre CPU S7-1200 je na Obr. 3.232.

Obr. 3.232. Zoznam inštrukcií počítadiel

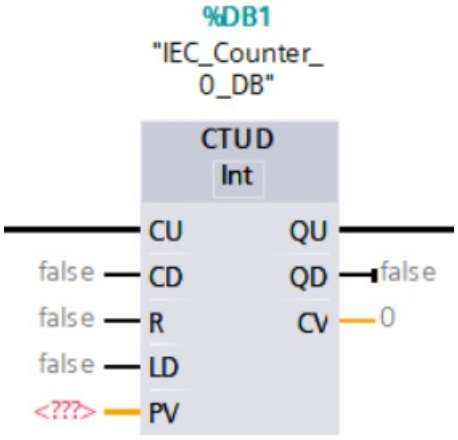
IEC_Counter_0_DB

	Name	Data type	Start value	Retain
1	Static			
2	CU	Bool	false	☑
3	CD	Bool	false	☑
4	R	Bool	false	☑
5	LD	Bool	false	☑
6	QU	Bool	false	☑
7	QD	Bool	false	☑
8	PV	Int	0	☑
9	CV	Int	0	☑

Obr. 3.233. Inštančný dátový blok inštrukcie *počítanie hore a dole*

3.7.1 Počítanie hore a dole (CTUD)

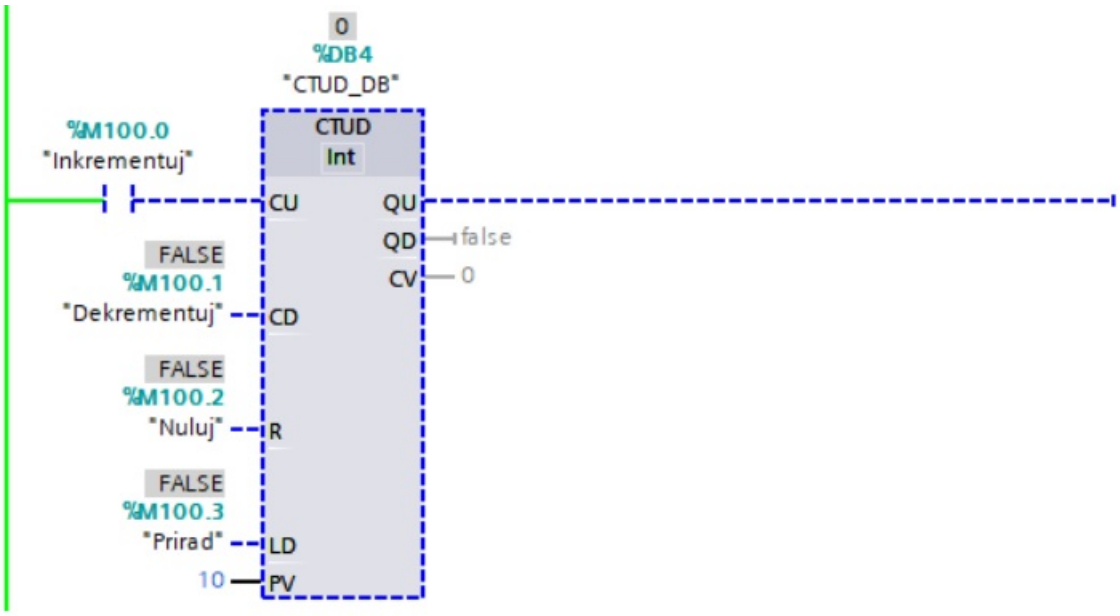
Symbol inštrukcie *CTUD* je na Obr. 3.234.



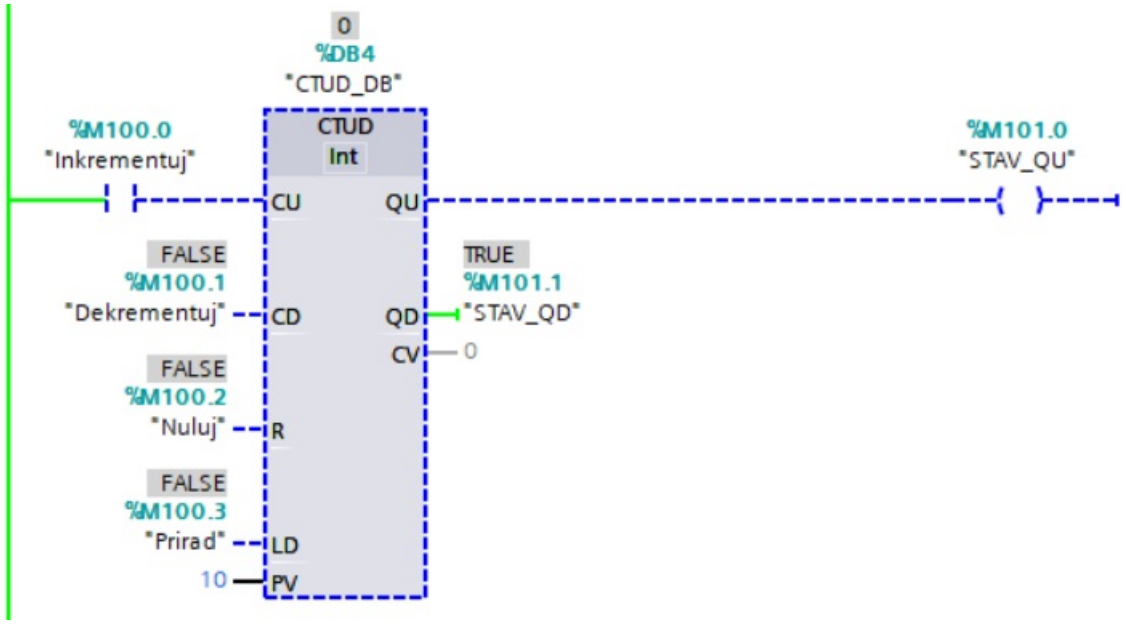
Obr. 3.234. Symbol inštrukcie *CTUD*

Ide o počítadlo, ktoré dokáže inkrementovať a dekrementovať svoju aktuálnu hodnotu. Počítadlá *CTU* a *CTD* prezentované v neskorších kapitolách majú len časť funkcionality *CTUD*. Funkcionalitu si podrobne opíšeme na Obr. 3.235 až 3.239.

Na Obr. 3.235 sú všetky vstupy a výstupy počítadla nulové okrem výstupu QD lebo hodnota CV = 0. Stav TRUE nie je animovaný zelenou farbou, lebo k výstupu QD nie je priradená žiadna premenná. Modifikovaný program je na Obr. 3.236.



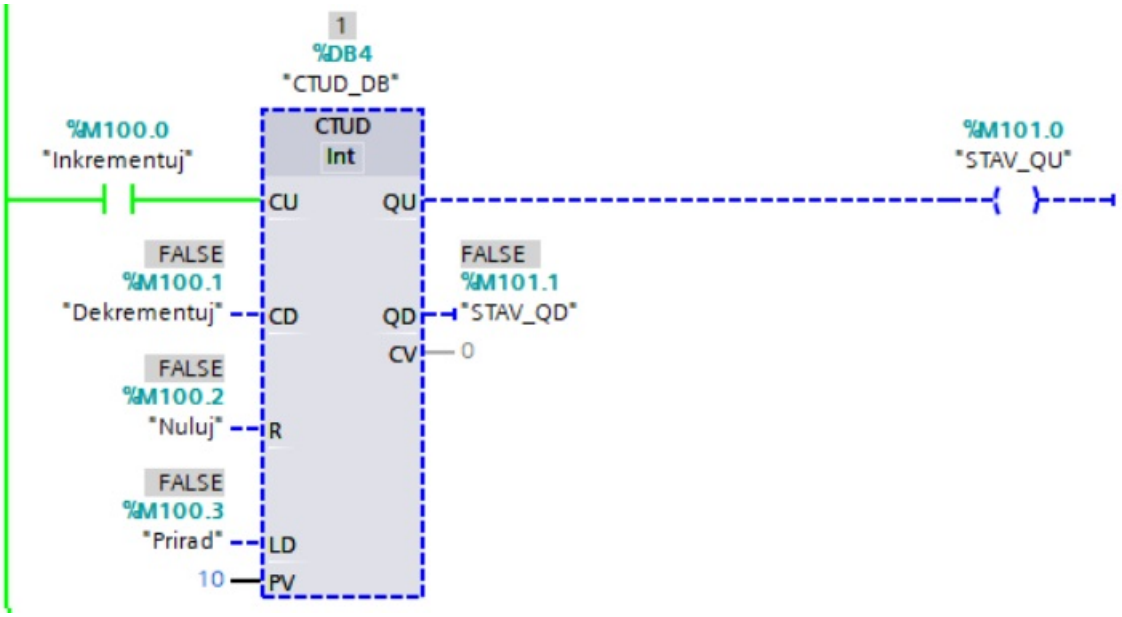
Obr. 3.235. 1. fáza monitorovania programu s inštrukciou *CTUD*



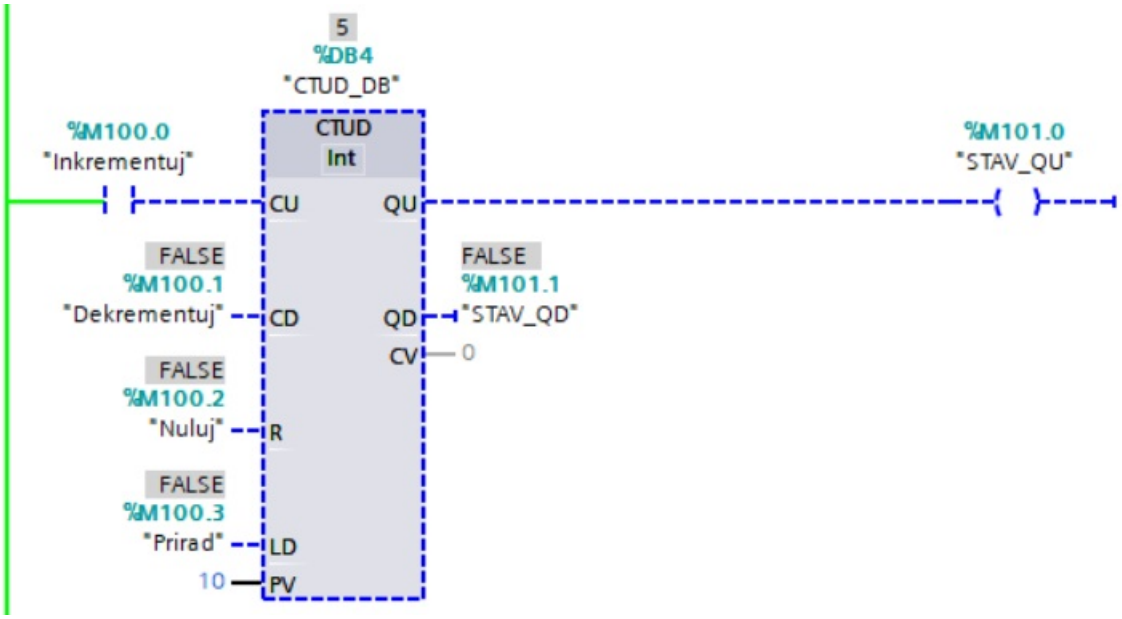
Obr. 3.236. 1. fáza monitorovania programu s inštrukciou *CTUD* – alternatíva

Privedením hodnoty TRUE na vstup CU je inštrukciou detegovaná nábežná hrana. Hodnota počítadla CV bola navýšená na hodnotu 1 (Obr. 3.237). Opätovné navýšenie

hodnoty je možné zmenou signálu na vstupe CU (generovanie nábežnej hrany). Postupná inkrementácia hodnoty je na Obr. 3.238.

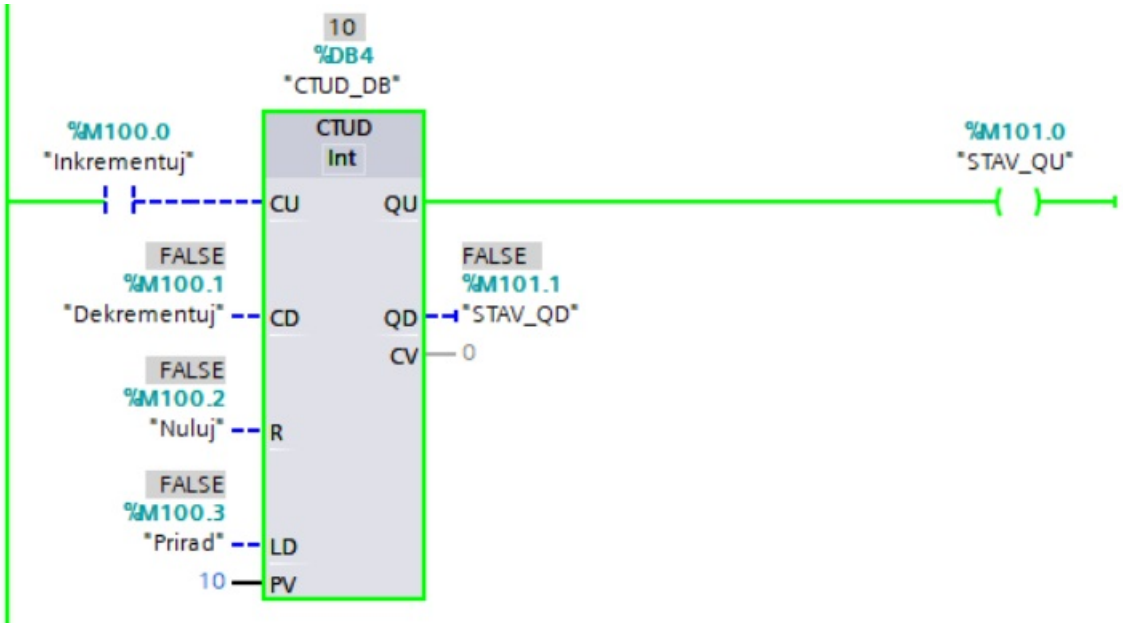


Obr. 3.237. 2. fáza monitorovania programu s inštrukciou *CTUD*

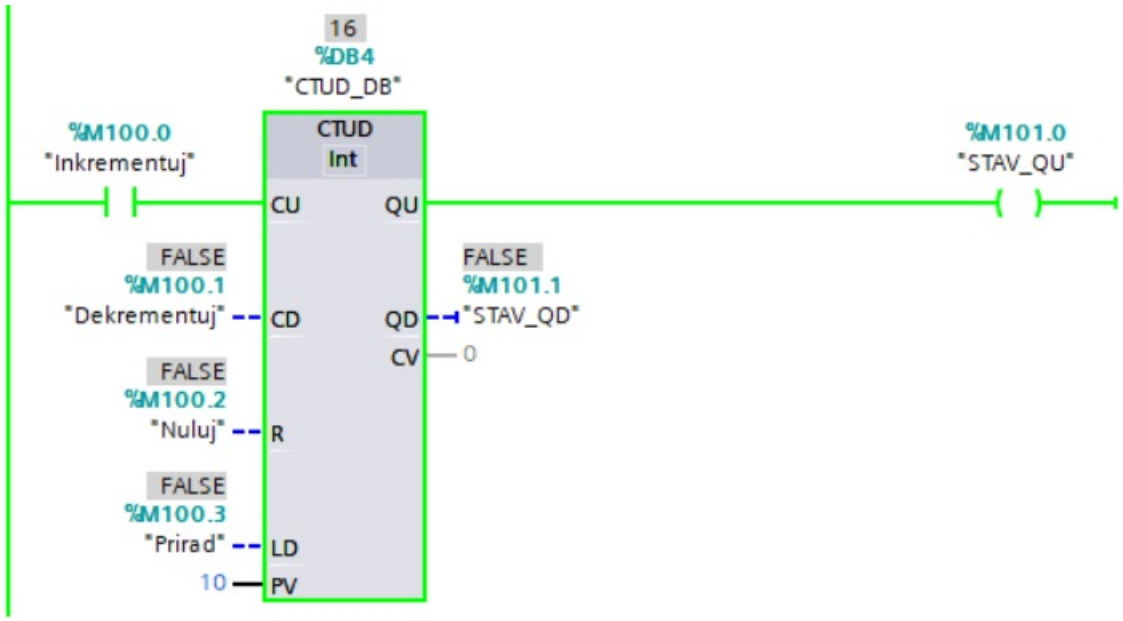


Obr. 3.238. 3. fáza monitorovania programu s inštrukciou *CTUD*

Na Obr. 3.239 bola dosiahnutá prednastavená hodnota. Výstup počítadla QU má stav TRUE. QU je v stave TRUE, keď platí CV>=PV. Príklad takéhoto stavu je na Obr. 3.240. Postupným navyšovaním hodnoty CU by sme mohli dosiahnuť maximálnu hodnotu dátového typu (v príklade INT). Ďalším navýšením by nastalo pretečenie. Hodnota CV by nadobudla najmenšiu hodnotu dátového typu.

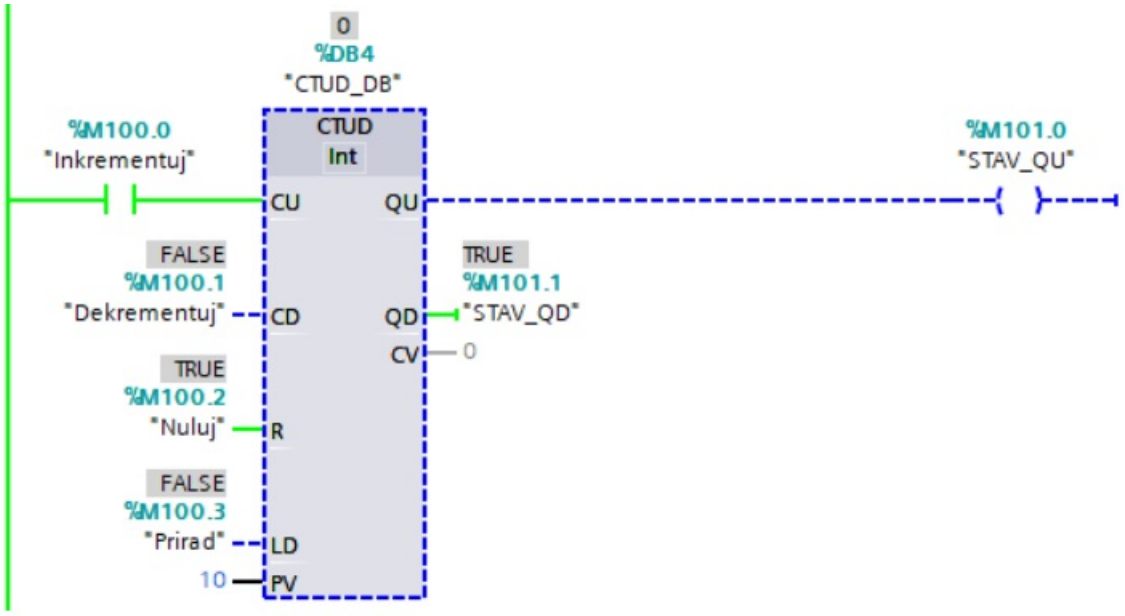


Obr. 3.239. 4. fáza monitorovania programu s inštrukciou *CTUD*



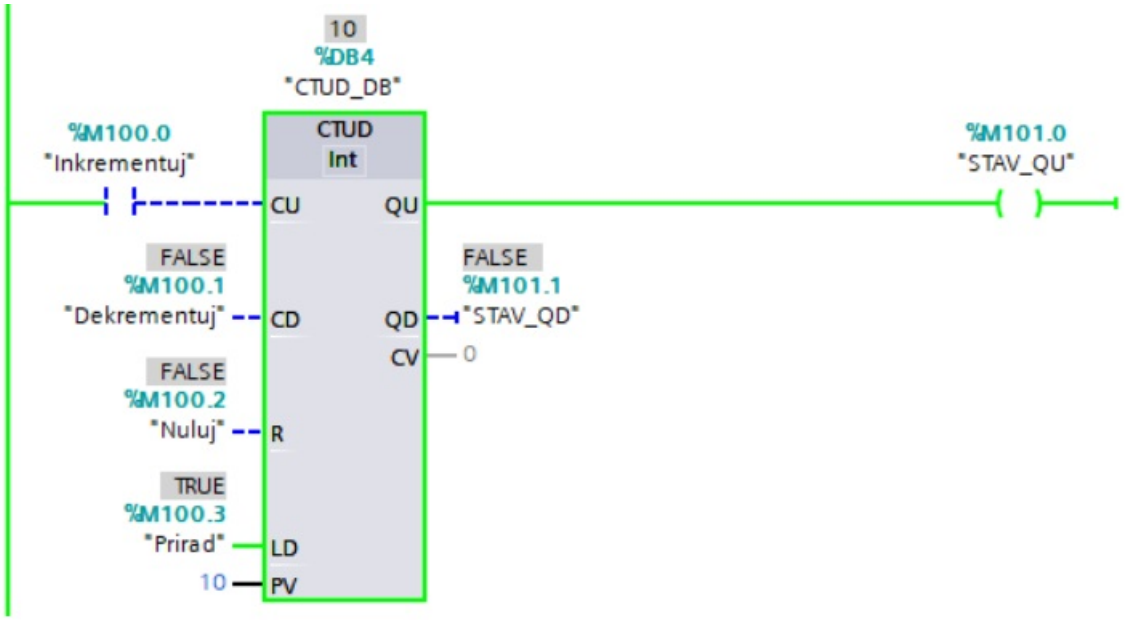
Obr. 3.240. 5. fáza monitorovania programu s inštrukciou *CTUD*

Príklad vynulovanie hodnoty časovača je na Obr. 3.241. Na vstup R sa priviedol logický signál TRUE.



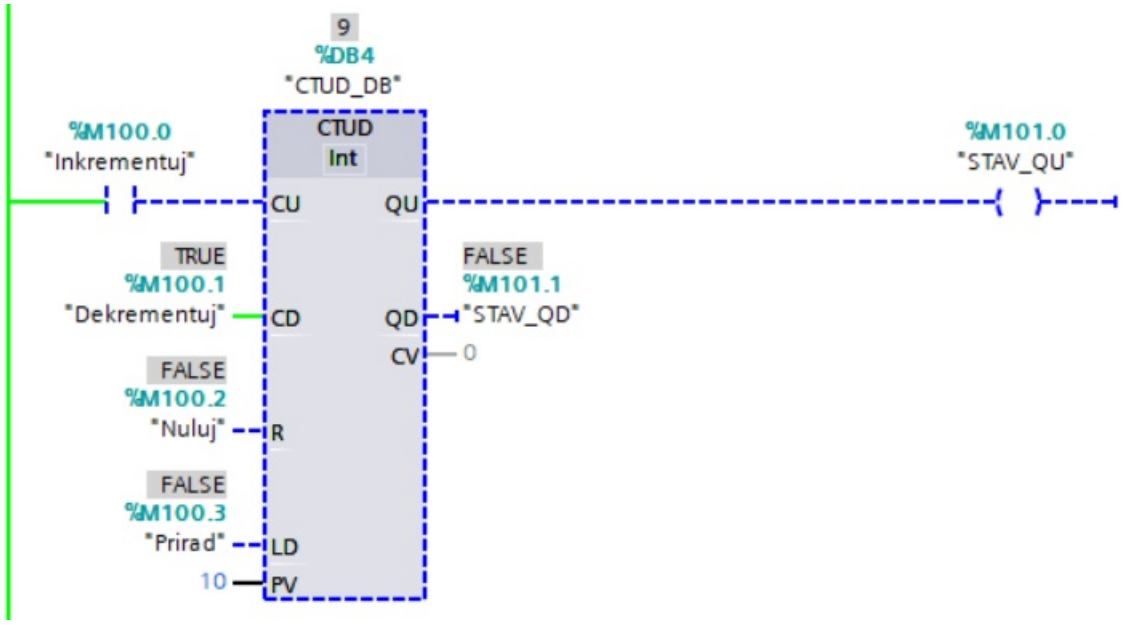
Obr. 3.241. 6. fáza monitorovania programu s inštrukciou *CTUD*

Monitorovanie programu keď došlo k priradeniu hodnoty PV do CV je na Obr. 3.242. Priradenie sa realizovalo priradením hodnoty TRUE k vstupu LD. Keďže hodnota CV a PV sa rovnajú, výstup QU je v stave TRUE.



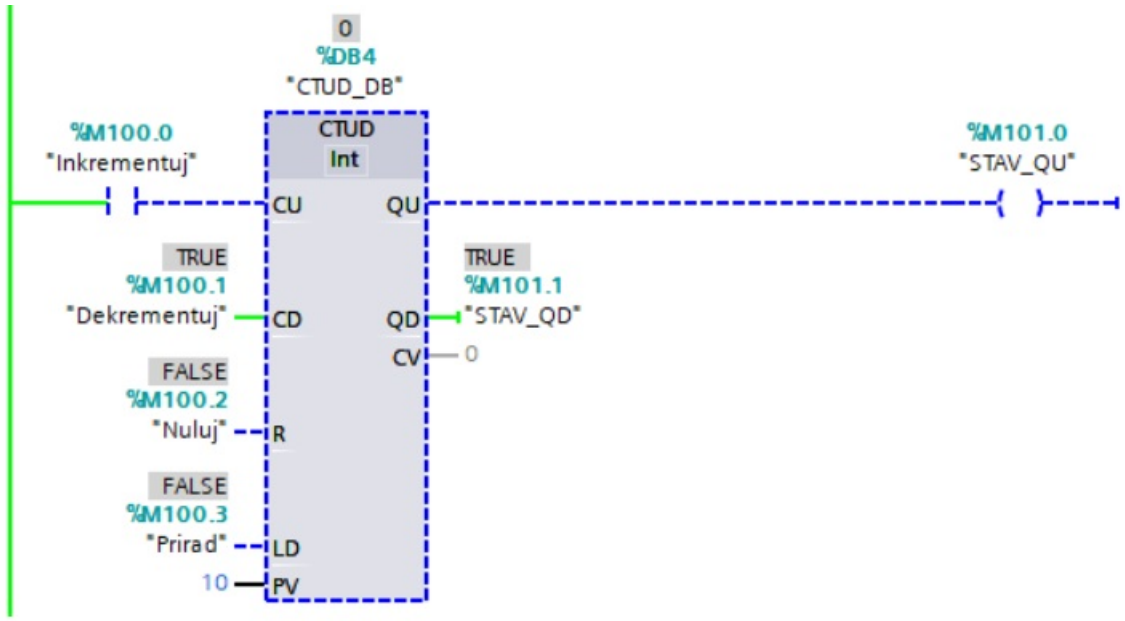
Obr. 3.242. 7. fáza monitorovania programu s inštrukciou *CTUD*

Na Obr. 3.243 došlo k dekrementácii aktuálnej hodnoty počítadla z 10 na 9 privedením signálu TRUE na vstup CD. Inštrukcia reaguje na nábežnú hranu, preto sa dekrementácia vykoná len raz.

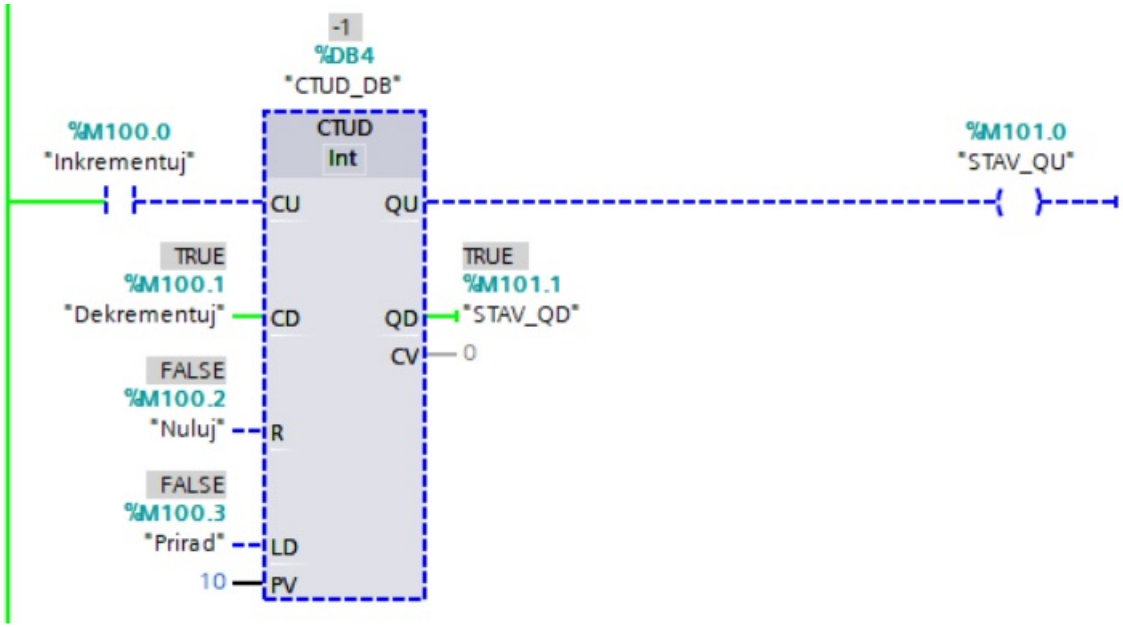


Obr. 3.243. 8. fázaa monitorovania programu s inštrukciou *CTUD*

Postupnou dekrementáciou hodnoty počítadla pomocou vstupu CD hodnota CV dosiahne 0. Výstup QD bude aktivovaný (Obr. 3.244). Ďalšou dekrementáciou sa hodnota CV zmení z 0 na -1 (Obr. 3.245) lebo počítadlo má nastavený dátový typ INT. Hodnota je nižšia ako 0, preto je výstup QD aktivovaný. Hodnota CV by klesala postupnou dekrementáciou až po najnižšiu hodnotu dátového typu, nastalo by pretečenie dátového typu čím by počítadlo nadobudlo maximálnu hodnotu rozsahu dátového typu.



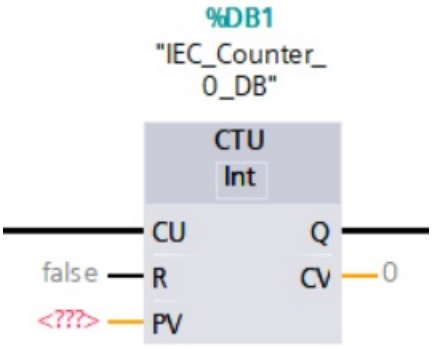
Obr. 3.244. 9. fáza monitorovania programu s inštrukciou *CTUD*



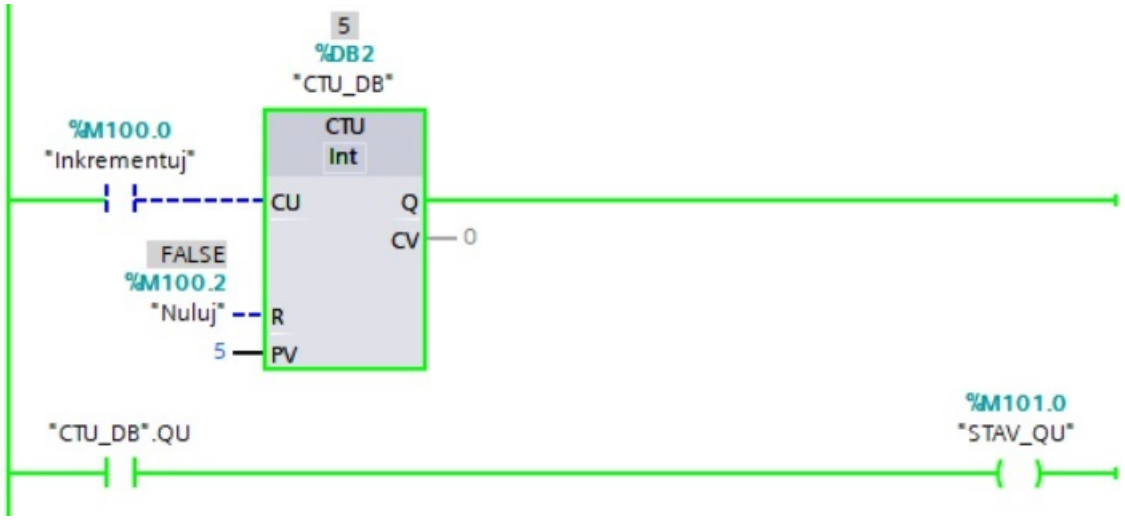
Obr. 3.245. 10. fáza monitorovania programu s inštrukciou *CTUD*

3.7.2 Počítanie hore (CTU)

Symbol inštrukcie *CTU* je na Obr. 3.246. Ide o zjednodušený symbol inštrukcie *CTUD*, ktorý sa používa na inkrementáciu hodnoty CV z 0 na hodnotu PV. Výstup Q je totožný s výstupom QU. Príklad dosiahnutia prednastavenej hodnoty je na Obr. 3.247. Inštrukciou *normálne otvorený kontakt* sa v 2. vetve načíta stav dosiahnutia hodnoty PV. Všimnite si, že namiesto Q je použitá premenná QU opísaná v skorších kapitolách.



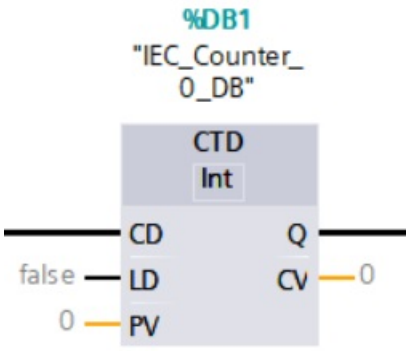
Obr. 3.246. Symbol inštrukcie *CTU*



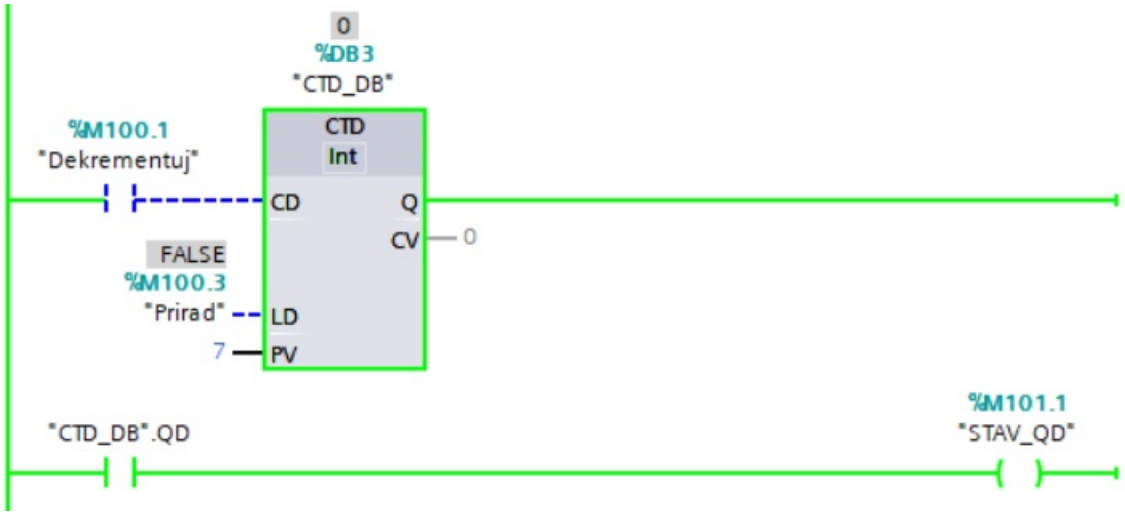
Obr. 3.247. Príklad inštrukcie *CTU*

3.7.3 Počítanie dole (CTD)

Symbol inštrukcie *CTD* je na Obr. 3.248. Ide o zjednodušený symbol inštrukcie *CTUD*, ktorý sa používa na dekrementáciu hodnoty CV z hodnoty PV na 0. Výstup Q je totožný s výstupom QD. Príklad dosiahnutia hodnoty 0 je na Obr. 3.249. Inštrukciou *normálne otvorený kontakt* v 2. vetve načíta stav dosiahnutia hodnoty 0.



Obr. 3.248. Symbol inštrukcie *CTD*



Obr. 3.249. Príklad inštrukcie *CTD*

3.8 Inštrukcie presunu (Move operations)

Inštrukcie presunu presúvajú hodnoty z jedného pamäťového miesta na iné pamäťové miesto. V tejto kapitole sa zameriame len na inštrukciu *MOVE* z celkového zoznamu (Obr. 3.250).

Move operations	
MOVE	Move value
Deserialize	Deserialize
Serialize	Serialize
MOVE_BLK	Move block
MOVE_BLK_VARIANT	Move block
UMOVE_BLK	Move block uninterruptible
FILL_BLK	Fill block
UFILL_BLK	Fill block uninterruptible
SCATTER	Parse the bit sequence into individual bits
SCATTER_BLK	Parse elements of an ARRAY of bit sequence into individual bits
GATHER	Merge individual bits into a bit sequence
GATHER_BLK	Merge individual bits into multiple elements of an ARRAY of bit sequence
SWAP	Swap
Variant	
Array[*]	
Legacy	

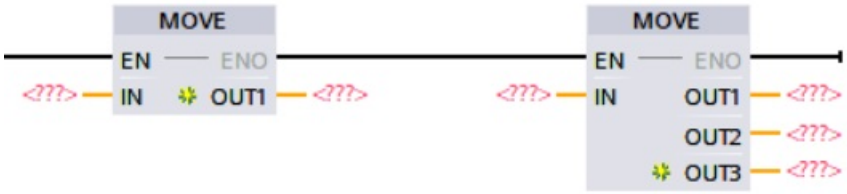
Obr. 3.250. Zoznam inštrukcií presunu pre CPU S7-1200

3.8.1 Presun hodnoty premennej (MOVE)

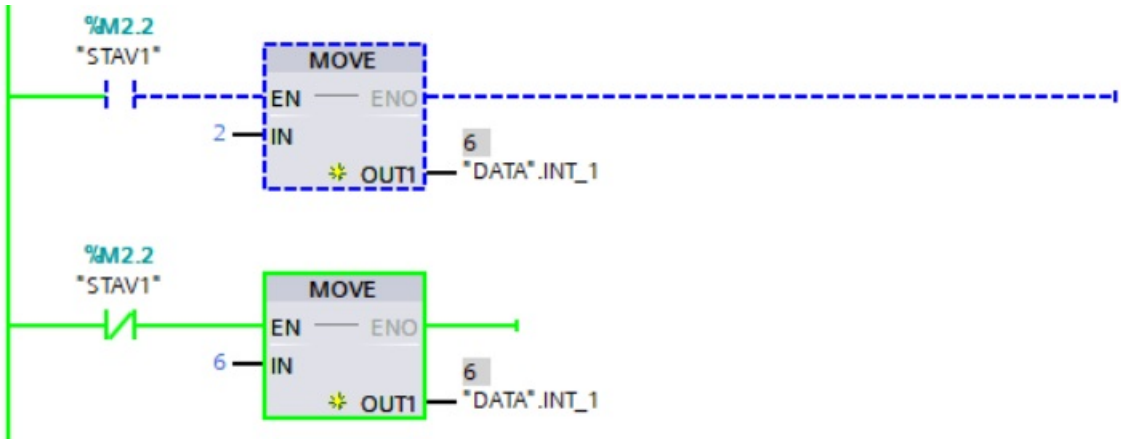
Základný symbol inštrukcie *MOVE* je na Obr. 3.251 vľavo. Inštrukcia presúva hodnotu premennej pripojenej na vstup IN na výstup OUT1. Programátor môže zadefinovať viacej

výstupov (pozri 3.251 vpravo). Hodnota IN sa zapíše na všetky výstupy OUT1 až OUTn. Dátové typy premenných na vstupe a výstupoch sa musia zhodovať. Okrem základných (elementárnych dátových typov) možno pripojiť používateľom definovaný dátový typ. V takomto prípade sa všetky hodnoty štrukturovanej premennej priradia na výstup. Zjednodušuje to programovanie, lebo namiesto N inštrukcií *MOVE* (kde N je počet prvkov UDT) stačí jedna inštrukcia *MOVE*.

Na Obr. 3.252 je jednoduchý príklad použitia inštrukcie. V závislosti od stavu premennej STAV1 sa do premennej DATA.INT_1 priradí hodnota 2 alebo 6. Hodnota premennej STAV1 je FALSE, preto sa vykonáva priradenie v 2. vetve. Táto premenná by mohla byť parametrom počítadla, ktorý sa priradí k vstupu PV. Počítadlo by rátalo rôzny počet udalostí v závislosti od stavu premennej STAV1.



Obr. 3.251. Symbol inštrukcie *MOVE*



Obr. 3.252. Príklad použitia inštrukcie *MOVE*

Na Obr. 3.253 bol navrhnutý demonštračný príklad UDT. Na Obr. 3.254 bol v dátovom bloku DATA vytvorený array pomocou UDT. Presun celej štruktúry (viacerých premenných) je na Obr. 3.255. Pri tomto presune sa hodnota DATA.vektor[1].INT_1 priradí do DATA.vektor[2].INT_1, DATA.vektor[1].INT_2 do DATA.vektor[2].INT_2 atď. Všimnite si chýbajúci symbol žltej hviezdičky rozšírenia počtu výstupov. Pri priradení UDT možno mať len jeden výstup.

UDT_Priklad

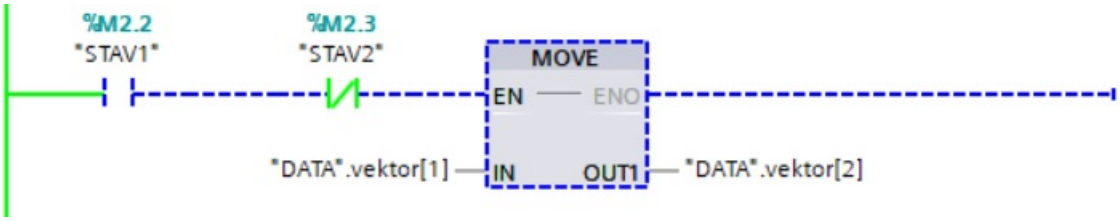
	Name	Data type
1	INT_1	Int
2	INT_2	Int
3	REAL_1	Real
4	BIT_1	Bool
5	ARRAY_1	Array[0..2] of Int
6	BIT_2	Array[0..2] of Int

Obr. 3.253. Príklad UDT

DATA

	Name	Data type
1	Static	
2	vektor	Array[0..3] of "UDT_Priklad"
3	vektor[0]	"UDT_Priklad"
4	INT_1	Int
5	INT_2	Int
6	REAL_1	Real
7	BIT_1	Bool
8	ARRAY_1	Array[0..2] of Int
9	BIT_2	Array[0..2] of Int
10	vektor[1]	"UDT_Priklad"
11	vektor[2]	"UDT_Priklad"
12	vektor[3]	"UDT_Priklad"

Obr. 3.254. Príklad dátového bloku s UDT



Obr. 3.255. Príklad inštrukcie *MOVE* s UDT

3.9 Inštrukcie riadenia programu (Program control operations)

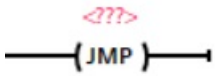
Inštrukcie riadenia programu dokážu vynechať časti programového kódu (napr. skok na návestie), predčasne ukončiť volanie funkcie, spracovať chybové hlásenia a pod. V tejto kapitole sa zameriame len na inštrukcie *JMP*, *JMPN* a *LABEL* z celkového zoznamu na Obr. 3.256.

Program control opera...	
–(JMP)	Jump if RLO = 1
–(JMPN)	Jump if RLO = 0
LABEL	Jump label
JMP_LIST	Define jump list
SWITCH	Jump distributor
–(RET)	Return
Runtime control	
ENDIS_PW	Limit and enable password legitimation
RE_TRIGR	Restart cycle monitoring time
STP	Exit program
GET_ERROR	Get error locally
GET_ERR_ID	Get error ID locally
RUNTIME	Measure program runtime

Obr. 3.256. Zoznam inštrukcií riadenia programu pre CPU S7-1200

3.9.1 Skok JMP

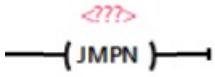
Symbol inštrukcie *JMP* je na Obr. 3.257. Inštrukcia pri platnej podmienke RLO (RLO = TRUE) na vstupe vykoná skok na návestie, ktoré je parametrom inštrukcie. Návestie musí existovať. Návestie sa definuje inštrukciou *LABEL*. Príklad je uvedený v kap. 3.9.3 Návestie (LABEL).



Obr. 3.257. Symbol inštrukcie *JMP*

3.9.2 Skok JMPN

Symbol inštrukcie *JMPN* je na Obr. 3.258. Inštrukcia pri neplatnej podmienke RLO (RLO = FALSE) na vstupe vykoná skok na návestie, ktoré je parametrom inštrukcie. Návestie musí existovať. Návestie sa definuje inštrukciou *LABEL*.



Obr. 3.258. Symbol inštrukcie *JMPN*

3.9.3 Návestie (LABEL)

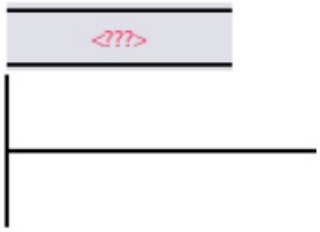
Symbol inštrukcie *LABEL* je na Obr. 3.259. Symbol sa zobrazuje v ľavej hornej časti vybraného rebríka (network). Do symbolu sa vkladá názov návestia. Názov návestia skoku môže byť v bloku priradený iba raz. Inštrukcia skoku a návestie musia byť v tom istom

programovom bloku (OB, FC alebo FB). Pri použití CPU S7-1200 možno deklarovať až 32 návestí skokov a pri použití CPU S7-1500 maximálne 256 návestí skokov.

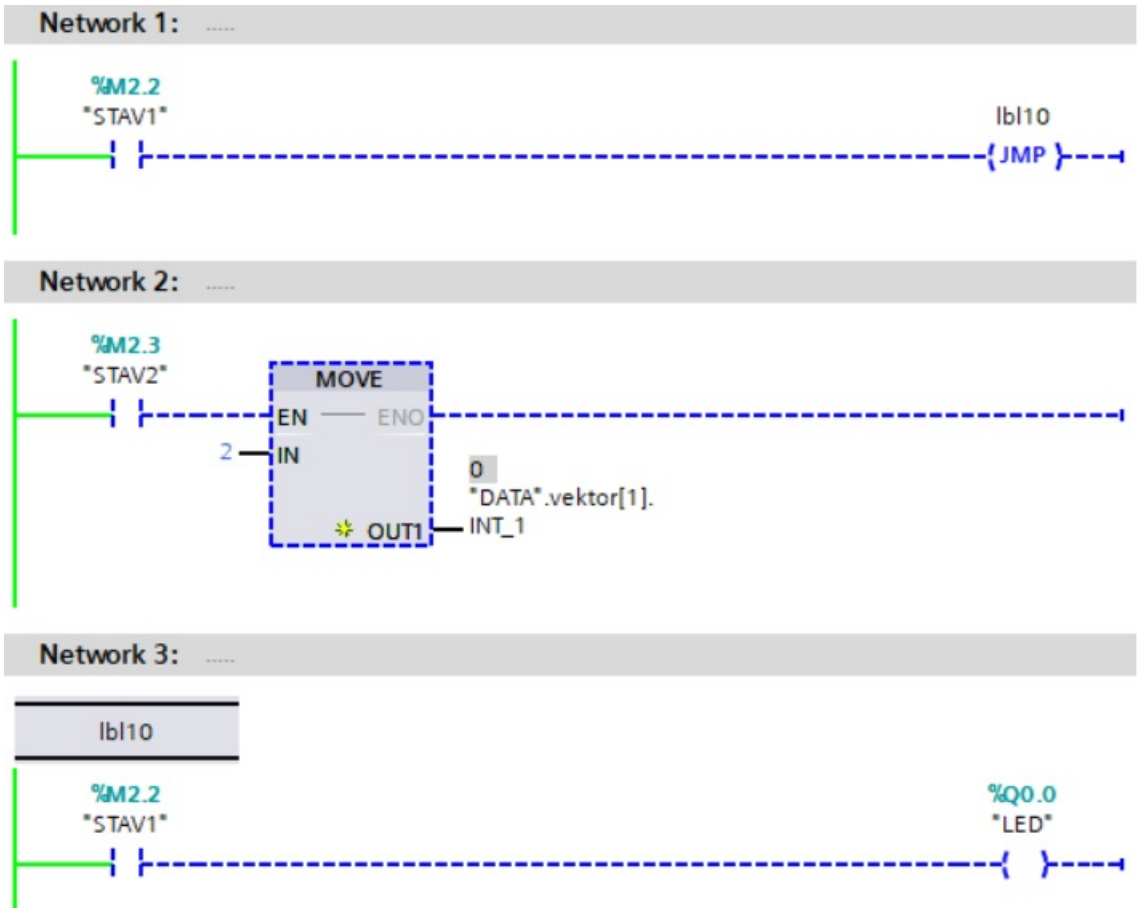
Na Obr. 3.260 v Network 1 nie je platná podmienka. Skok na návestie lbl10 sa nevykoná, preto sa pokračuje vykonaním programu na Network 2 a 3.

Na Obr. 3.261 v Network 1 je podmienka platná. Skok na návestie lbl10 sa vykoná, preto program na Network 2 sa nevykoná (pozri bledozelenú farbu v Network 2). Program pokračuje na Network 3.

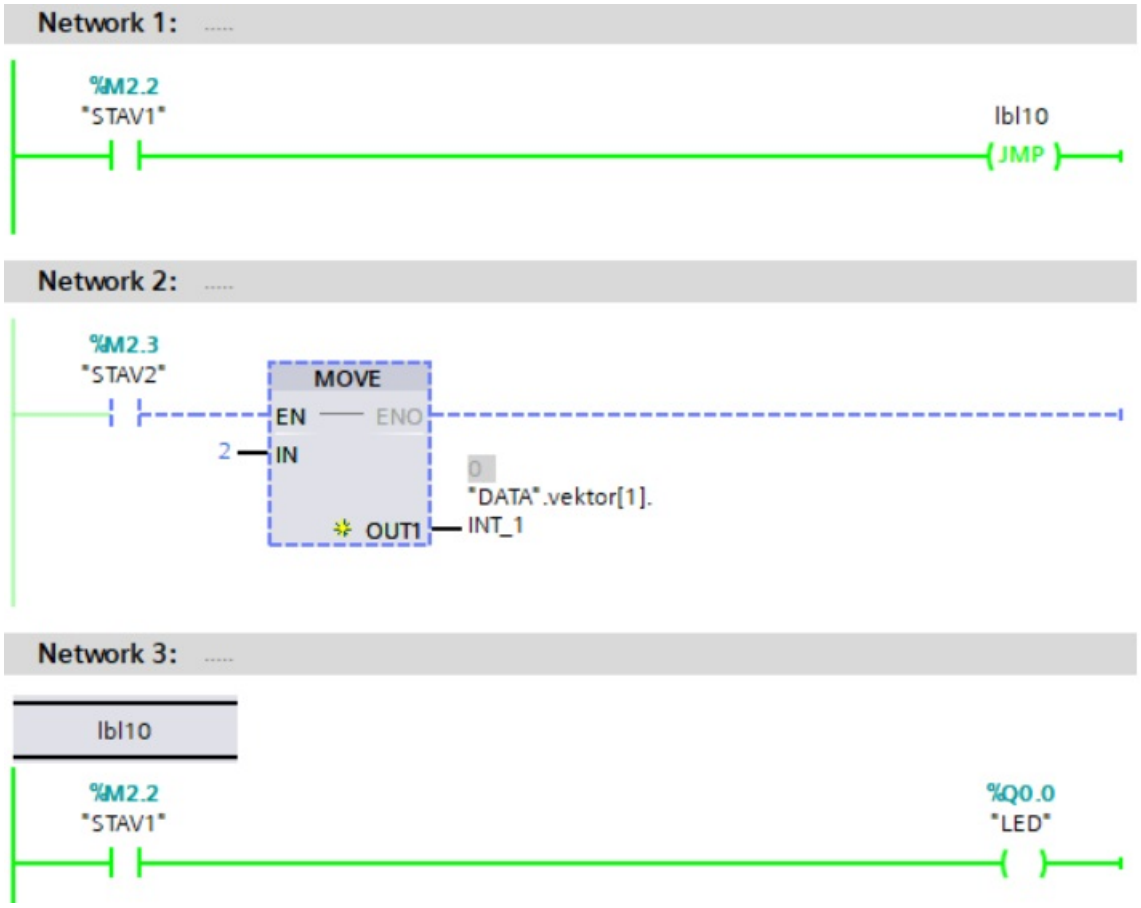
Na Obr. 3.262 je príklad skoku na Network 1, ktorý sa nachádza pred inštrukciou skoku na návestie. Ak by stav premennej STAV1 bol dlhodobo TRUE, program by neskočil do ochranného času watchdog a CPU by sa preplo do režimu STOP.



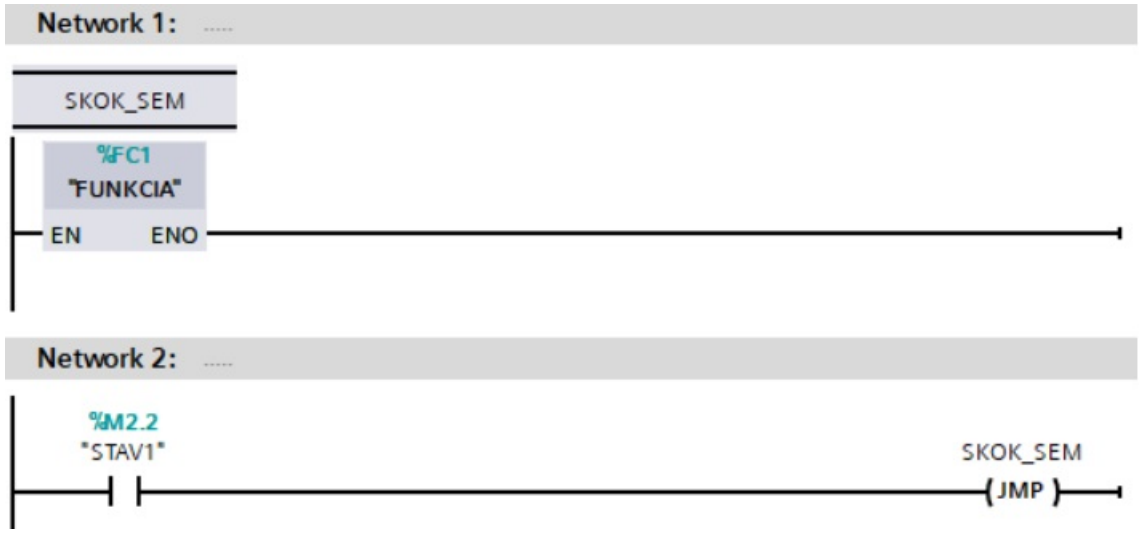
Obr. 3.259. Symbol inštrukcie *LABEL*



Obr. 3.260. 1. fáza monitorovania



Obr. 3.261. 2. fáza monitorovania



Obr. 3.262. Príklad inštrukcie *LABEL*

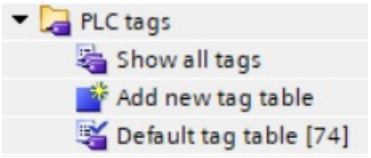
Kapitola 4

Príklady

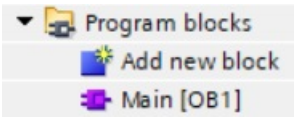
V tejto kapitole uvádzame zbierku príkladov aplikácie vybraných inštrukcií opísaných v predchádzajúcej kapitole. Prvé príklady budú analyzované detailnejšie s cieľom uviesť čitateľa do problematiky. Prvý blok príkladov je zameraný na pochopenie samotných príkazov. Druhá časť sa zaoberá aplikáciu príkazov pre jednoduché príklady a virtuálne modely. Pred samotným návrhom a analýzou programov sú uvedené príklady premenných.

4.1 Premenné

Cieľom nižšie uvedených príkladov je ozrejmenie adresácie premenných, ktoré sú parametrami inštrukcií. Ich správne definovanie a použitie minimalizuje chyby, ktoré by mohli vznikať pri návrhu algoritmu riadenia v riadiacom systéme. Premenné skupín %I, %Q a %M vytvárame v PLC tags (Obr. 4.1) alebo priamo v programe. Každá vytvorená premenná sa ukladá do tzv. tag table (tabuľky premenných), ktorých môže byť viacej. Dátové bloky sa pridávajú v menu Add new block (Obr. 4.2) podobne ako organizačné bloky, funkcie a funkčné bloky.



Obr. 4.1. Časť stromovej štruktúry v TIA Portal - Miesto uloženia zoznamu premenných

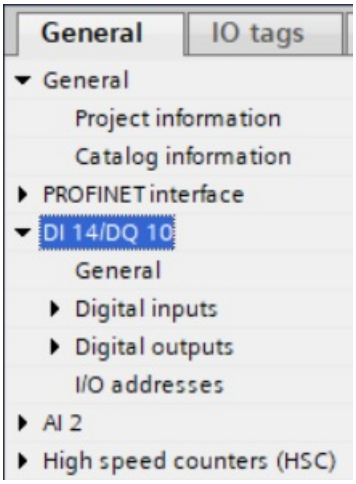


Obr. 4.2. Časť stromovej štruktúry v TIA Portal - Miesto pridávania dátových blokov

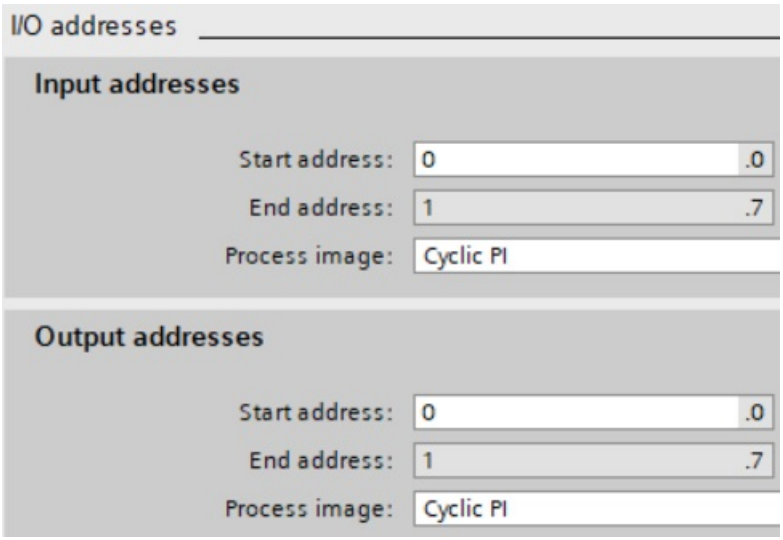
4.1.1 Príklad 1 - Vstupy a výstupy S7-1200

V tejto kapitole si uvedieme príklad adresácie vstupov a výstupov. Majme riadiaci systém S7-1214 DC/DC/DC s katalógovým číslom 6ES7 214-1AE30-0XB0. Otvorením hardvérovej

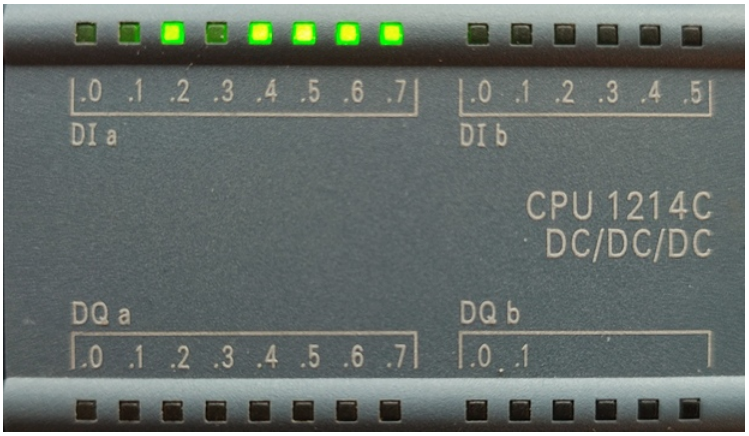
konfigurácie vo vlastnostiach procesora nájdeme nastavenie digitálnych vstupov a výstupov (Obr. 4.3). Z textu DI 14/DQ 10 vyplýva počet vstupov a výstupov, t. j. 14 digitálnych vstupov a 10 digitálnych výstupov. Kliknutím na I/O addresses (Obr. 4.4) sa otvorí okno konfigurácie počiatočných adries (bajtov) integrovaných digitálnych vstupov a výstupov na CPU. Tu vidíme počiatočné adresy (Start address) 0 a posledné alokované adresy (End address) 1. Vstupy CPU majú preto adresy %I0.0 až %I0.7 a %I1.0 až %I1.5 (spolu 14 adries). Analogicky pre výstupy %Q0.0 až %Q0. 7a %Q1.0 až %Q1.1 (spolu 10 adries). Pre vstupy a výstupy sa alokujú celé bajty, ale podľa konkrétneho CPU sa využije iný počet bitov pre vstupy a výstupy. Tieto adresy sú zahrnuté v programovom cykle PLC (viď nastavenie adries vstupov a výstupov na Obr. 4.4 a kap. 1.2.2 Cyklus PLC). Príklad aktuálnych stavov vstupov a výstupov je na Obr. 4.5. DIa a DQa sú adresy počiatočných bajtov (v našom príklade 0) a DIb a DQb sú adresy posledných alokovaných bajtov (v našom príklade 1). LED pri vstupoch %I0.2,%I0.4, %I0.5, %I0.6 a %I0.7 svietia na zeleno indikujúc pripojenie 24V k digitálnym vstupom. Stav uvedených vstupov je TRUE. Pre zvyšné vstupy sú stavy FALSE. V príklade na obrázku nie sú zopnuté žiadne digitálne výstupy. Indikovali by sa rozsvietením príslušnej zelenej LED. Zoznam adries je dostupný vo vlastnostiach CPU v záložke IO Tags ako aj na CPU po priblížení rozhrania vstupov a výstupov (Obr. 4.6). Animácie stavov vstupov a výstupov v TIA Portal nájdete v kap. 4.3 Návrh programov.



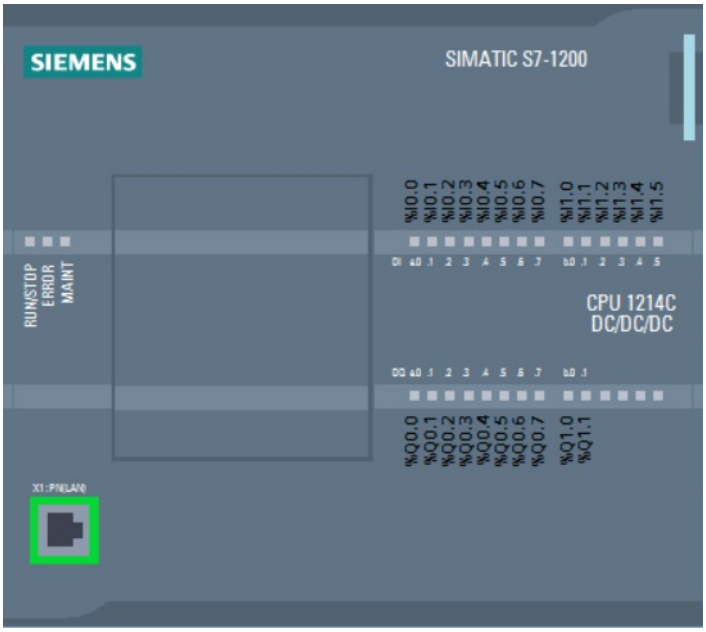
Obr. 4.3. Nastavenie konfigurácie digitálnych vstupov a výstupov S7-1200



Obr. 4.4. Nastavenie počiatočných adries vstupov a výstupov



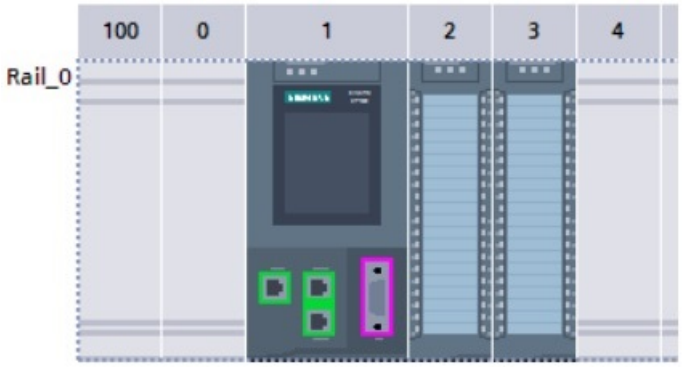
Obr. 4.5. Príklad stavov vstupov a výstupov na CPU



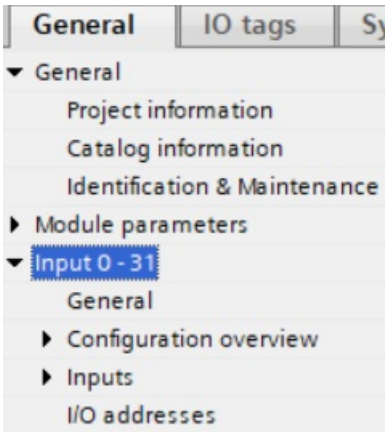
Obr. 4.6. Priblíženie hardvérovej konfigurácie S7-1200 v TIA Portal

4.1.2 Príklad 2 - Vstupy a výstupy S7-1500

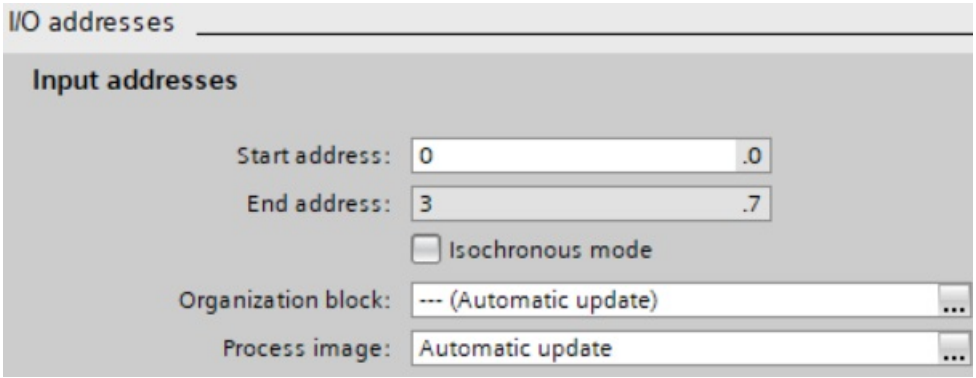
V druhom príklade si uvedieme príklad adresácie digitálnych vstupov a výstupov S7-1500. Majme riadiaci systém S7-1516-3 PN/DP s katalógovým číslom 6ES7 516-3AN01-0AB0. Po vložení procesora je nutné pridať rozširujúce moduly nakoľko S7-1500 procesory nemajú na sebe integrované vstupno-výstupné rozhrania. Ide o tzv. modulárne riadiace systémy. V pravej časti TIA Portal pridajme z hardvérového katalógu digitálny vstupný modul DI 32x24VDC HF (katalógové číslo 6ES7 521-1BL00-0AB0) a digitálny výstupný modul DQ 32x24VDC/0.5A ST (katalógové číslo 6ES7 522-1BL00-0AB0). Ide o 32 kanálové vstupné a výstupné moduly. Finálna hardvérová konfigurácia je na Obr. 4.7. Kliknutím na vstupný modul nájdeme v nastaveniach podobné nastavenia ako v prípade S7-1200 (Obr. 4.8). Nastavenia sú závislé od typu modulu, ale aj firmvéru (Obr. 4.9). Na obrázku vidno počiatočnú adresu (bajt) 0 a koncovú adresu 3. Adresy vstupov sú %I0.0 až %I0.7, %I1.0 až %I1.7, %I2.0 až %I2.7 a nakoniec %I3.0 až %I3.7, t. j. spolu 32 vstupov. Príklad aktívnych a neaktívnych digitálnych vstupov je na Obr. 4.10. Na obrázku znak a vľavo hore znamená počiatočný bajt adresy, pod ním je bajt b, vľavo hore bajt c a pod ním vľavo dole posledný bajt d. Na obrázku sú aktívne vstupy %I0.0, %I0.2, %I0.3, %I0.5, %I0.6, %I0.7, %I1.0, %I1.1 a %I0.2. Konfigurácia počiatočnej adresy výstupného modulu je podobná. Na Obr. 4.11 je priblíženie vstupov a výstupov hardvérovej konfigurácie v TIA Portal.



Obr. 4.7. Hardvérová konfigurácia S7-1500

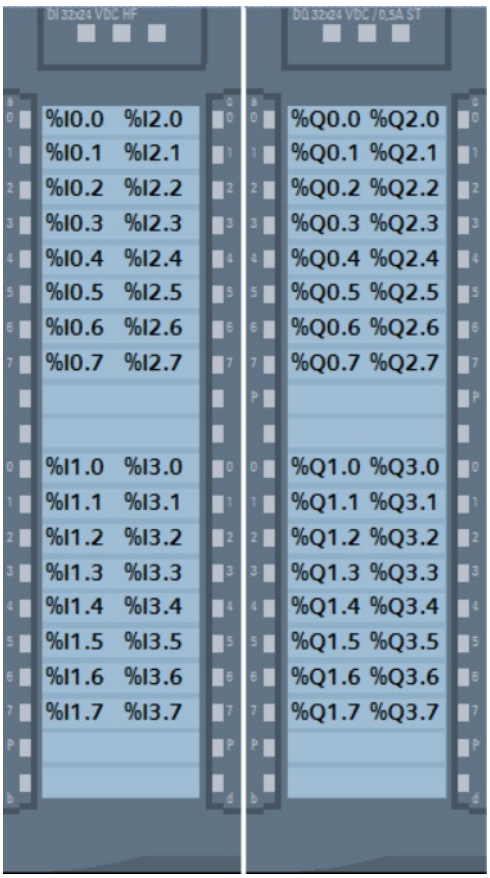


Obr. 4.8. Menu Input 0-31



Obr. 4.9. Nastavenia počiatočnej adresy vstupného modulu

Obr. 4.10. Príklady stavov digitálnych vstupov



Obr. 4.11. Priblíženie vstupov a výstupov v hardvérovej konfigurácii

4.1.3 Príklad 3 - Pamäťový priestor %M

V praxi na riadenie alebo monitorovanie procesu nestačia len vstupy (%I) a výstupy (%Q). Pamäťová oblasť %M (ako aj %DB) sa v PLC používa na ukladanie hodnôt, ktoré sa menia počas činnosti programu. Nezískavajú sa zo snímačov a neovládajú priamo akčné členy. Táto oblasť slúži na vytvorenie pomocných stavov zariadenia. Majme tri dopravníky, ktoré sa ovládajú digitálnymi výstupmi D1 %Q0.0, D2 %Q0.1 a D3 %Q0.2 smerom vpred. Majme na každom dopravníku optický senzor detegujúci prepravovaný objekt OS1 %I0.0, OS2 %I0.1 a OS3 %I0.2. A na záver informáciu z motorovej ochrany každého dopravníka D1OK %I0.3, D2OK %I0.4 a D3OK %I0.5. Nech sú všetky vstupy zapojené ako normálne otvorené kontakty, t. j. ak je OS1 v stave TRUE, deteguje objekt a ak vstup D1OK je v stave TRUE, motorová ochrana dopravníka D1 je v poriadku a možno ho ovládať.

Cieľom je vytvoriť pomocné stavy objektov, ktoré ovládame a monitorujeme. V našom príklade je objektom dopravník, ktorý má tri inštancie. Vytvorme ako prvé stavy (pomocné premenné) obsadenosti dopravníkov. Vytvorme premenné D1OBS s adresou %M0.0, D2OBS s adresou %M0.1 a D3OBS s adresou %M0.2. Premenné sme adresovali od bajtu 0 a bitu 0 lebo sme doteraz nemali vytvorené žiadne pamäťové premenné. Naložením objektu na D1 a detegovaním objektu snímačom OS1 sa môže nastaviť inštrukciou *SET* premenná D1OBS na TRUE. Po presune objektu z D1 na D2 a detekciou objektu snímačom OS2 sa inštrukciou *RESET* vynuluje D1OBS a nastaví sa D2OBS na TRUE. Takto zabezpečíme to, že ak sa objekt pohybuje medzi snímačmi OS1 a OS2 v premennej D1OBS máme informáciu, že sa medzi snímačmi nachádza objekt aj keď nie je detegovaný snímačmi.

Vytvorme alarmové stavy prepravy objektov D1_ALARM1 s adresou %M0.3, D2_A LARM1 s adresou %M0.4 s D3_ALARM1 s adresou %M0.5. Mohlo by sa stať, že sa prepravovaný objekt z D1 na D2 nedostane. Dôvodom môže byť fyzická porucha dopravníka (prešmykovanie pásu dopravníka) napriek funkčnej motorovej ochrane alebo odobratie objektu obsluhou. Riešením je časovač TON s vhodnou podmienkou. Podmienkou môže byť D1OBS AND D1 AND NOT OS2, t. j. dopravník D1 je obsadený, je v pohybe a snímač na D2 nedeteguje objekt. V tom prípade sa spúšťa časovač s dostatočnou rezervou a ak skončí k výstupu Q časovača pripojíme inštrukciu SET, ktorá nastaví alarm D1_ALARM1 na hodnotu TRUE. Po vzniku alarmu môže byť objekt oneskorene dopravený na D2 čím sa vynuluje časovač, ale pomocný stav (D1_ALARM) ostáva v stave TRUE, pokiaľ ho nezrušíme.

Vytvorme stavy pripravenosti dopravníkov odoslať objekt D1_PRIPRAVENY _ODOSLAT s adresou %M0.6, D2_PRIPRAVENY_ODOSLAT s adresou %M0.7 a D3_PRIPRA VENY_ODOSLAT s adresou %M1.0. Posledná premenná má adresu %M1.0 lebo sme využili všetky bity bajtu 0. Pripravenosť bude znamenať existenciu objektu na dopravníku (obsadenosť) a funkčnú motorovú ochranu dopravníka. Do premennej D1_PRIPRAVE NY_ODOSLAT by sme v každom cykle zapisovali inštrukciou *Priradenie* výsledok podmienky D1OBS AND D1OK. Ak je výsledok podmienky pravdivý, dopravník je pripravený odoslať objekt. V opačnom prípade má stav FALSE.

V praxi jeden objekt generuje viacej alarmov. Pridajme alarmy od motorových ochrán D1_ALARM2 s adresou %M1.1, D2_ALARM2 s adresou %M1.2 s D3_ALARM2 s adresou %M1.3. Ak je vypnutá motorová ochrana, t. j. D1OK má hodnotu FALSE, potom sa má vykonať inštrukcia SET, ktorá nastaví hodnotu D1_ALARM2 na TRUE. Po zmene vstupu na TRUE je alarm naďalej aktívny. Alarmy sa potvrdzujú fyzickým tlačidlom a/alebo softvérovým tlačidlom z vizualizácie.

V našom príklade vidno, že postupným pridávaním premenných za sebou v pamäti môže vzniknúť ich horšie dohľadanie. Napríklad alarmy nie sú za sebou. Otázkou je aj rozšíriteľnosť. Ak by sme chceli rozšíriť úlohu o ďalší dopravník D4 a jeho pomocné premenné, potom by premenné D3OBS a D4OBS neboli v pamäti za sebou. Z toho dôvodu je dobré určité skupiny premenných alokovať do rovnakého bajtu. Napríklad obsadenosti do bajtu 0, pripravenosti do bajtu 1 atď. Takto sú premenné prehľadnejšie a sú rozšíriteľné. V prípade alarmov sa alarmy musia ukladať do 16 alebo 32 bitových slov pre ich efektívnu komunikáciu do HMI.

V uvedenom príklade sme uviedli príklady postupného pridávania premenných. Pridaním nových premenných je potrebné myslieť na existenciu iných premenných, aby nedošlo k prepisovaniu spoločných pamäťových oblastí. Vytvorme premenné na uchovanie počtu spustení motorov dopravníkov D1_POCET_SPUSTENI, D2_POCET_SPUSTENI a D3_POCET_SPUSTENI napr. dátových typov INT. Bajty 0 a 1 sa používajú na uchovanie binárnych stavov, preto fyzická adresa %MW0 nie je vhodná lebo sa skladá z bajtov 0 a 1. Rovnako nie je vhodná ani fyzická adresa %MW1 lebo sa skladá z bajtov 1 a 2. Mohli by sme zadefinovať %MW2, ale čo by sa stalo prípadným pridaním nových binárnych stavov. Riešením je začať s posunutou adresou v pamäti, napr. %MW10 pre D1_POCET_SPUSTENI, %MW12 pre D2_POCET_SPUSTENI a %MW14 pre poslednú premennú D3_POCET_SPUSTENI. Príklad lepšieho rozdelenia premenných je na Obr. 4.12.

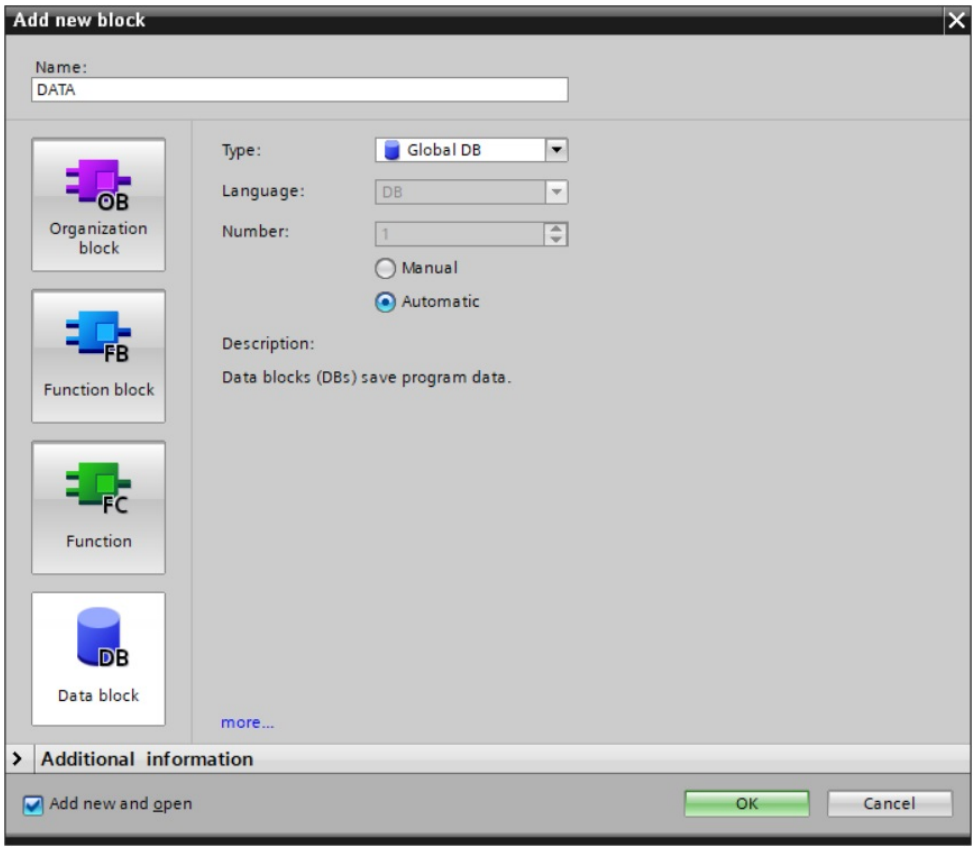
PLC tags

	Name	Tag table	Data type	Address
1	D1	Default tag table	Bool	%Q0.0
2	D2	Default tag table	Bool	%Q0.1
3	D3	Default tag table	Bool	%Q0.2
4	OS1	Default tag table	Bool	%I0.0
5	OS2	Default tag table	Bool	%I0.1
6	OS3	Default tag table	Bool	%I0.2
7	D1OK	Default tag table	Bool	%I0.3
8	D2OK	Default tag table	Bool	%I0.4
9	D3OK	Default tag table	Bool	%I0.5
10	D1OBS	Default tag table	Bool	%M0.0
11	D2OBS	Default tag table	Bool	%M0.1
12	D3OBS	Default tag table	Bool	%M0.2
13	D1_PRIPRAVENY_ODOSLAT	Default tag table	Bool	%M1.0
14	D2_PRIPRAVENY_ODOSLAT	Default tag table	Bool	%M1.1
15	D3_PRIPRAVENY_ODOSLAT	Default tag table	Bool	%M1.2
16	D1_ALARM1	Default tag table	Bool	%M2.0
17	D2_ALARM1	Default tag table	Bool	%M2.1
18	D3_ALARM1	Default tag table	Bool	%M2.2
19	D1_ALARM2	Default tag table	Bool	%M3.0
20	D2_ALARM2	Default tag table	Bool	%M3.1
21	D3_ALARM2	Default tag table	Bool	%M3.2
22	D1_POCET_SPUSTENI	Default tag table	Int	%MW10
23	D2_POCET_SPUSTENI	Default tag table	Int	%MW12
24	D3_POCET_SPUSTENI	Default tag table	Int	%MW14

Obr. 4.12. Príklad zoznamu vytvorených premenných

4.1.4 Príklad 4 - Pamäťový priestor %DB

Dátové bloky odstraňujú nevyhnutnosť sledovania fyzických adries premenných pri ich vytváraní a to bez ohľadu na to, či sú optimalizované alebo neoptimalizované. Kliknutím na Add new block v stromovej štruktúre sa zobrazí menu pridania nových objektov (Obr. 4.13). Zvolíme si Data block v ľavom dolnom rohu. V hornej časti zadefinujeme symbolický názov dátového bloku (v príklade DATA). V type je prednastavená voľba Global DB, t. j. dátový blok v ktorom možno ľubovoľne vytvárať premenné. Inštančné dátové bloky sú naviazané na funkčné bloky a preto ich štruktúra je pevne daná. Všimnite si číslo za Number (v príklade 1). Ide o číslo dátového bloku (v príklade %DB1). V automatickom režime TIA Portal nájde nepoužité číslo dátového bloku, ktorý mu priradí, aby nedošlo k duplicitným fyzickým adresám DB. V manuálnom režime používateľ zadáva číslo DB. Vytvorme obsadenosti, pripravenosti odoslať objekt a počty spustení dopravníkov podobne ako v predchádzajúcom príklade. Príklad vytvorených premenných je na Obr. 4.14. Pridaním nových premenných medzi skupiny obsadenosti a pripravenosti je zachované logické členenie premenných (4.15).



Obr. 4.13. Vyskakovacie (popup) okno konfigurácie nového dátového bloku

DATA

		Name	Data type	Start value	Retain
1		Static			
2		D1OBS	Bool	false	
3		D2OBS	Bool	false	
4		D3OBS	Bool	false	
5		D1_PRIPRAVENY_ODOSLAT	Bool	false	
6		D2_PRIPRAVENY_ODOSLAT	Bool	false	
7		D3_PRIPRAVENY_ODOSLAT	Bool	false	
8		D1_POCET_SPUSTENI	Int	0	
9		D2_POCET_SPUSTENI	Int	0	
10		D3_POCET_SPUSTENI	Int	0	

Obr. 4.14. Príklad binárnych (BOOL) a celočíselných (INT) premenných v DB

DATA

	Name	Data type	Start value	Retain
1	Static			
2	D1OBS	Bool	false	
3	D2OBS	Bool	false	
4	D3OBS	Bool	false	
5	D4OBS	Bool	false	
6	D1_PRIPRAVENY_ODOSLAT	Bool	false	
7	D2_PRIPRAVENY_ODOSLAT	Bool	false	
8	D3_PRIPRAVENY_ODOSLAT	Bool	false	
9	D4_PRIPRAVENY_ODOSLAT	Bool	false	
10	D1_POCET_SPUSTENI	Int	0	
11	D2_POCET_SPUSTENI	Int	0	
12	D3_POCET_SPUSTENI	Int	0	
13	D4_POCET_SPUSTENI	Int	0	

Obr. 4.15. Príklad rozšírenia zoznamu premenných v DB

4.1.5 Príklad 5 - Pamäťový priestor %DB

V predchádzajúcom príklade sme v dátovom bloku vytvorili 3 skupiny premenných pre 4 dopravníky. Ak je v technológii viacero objektov, je vhodné vytvoriť pole (array) štruktúr (Struct). Jedna štruktúra reprezentuje práve jeden objekt. V našom príklade štruktúra zodpovedá dopravníku. Vytvorme v DB premennú D (skratka pre dopravník) dátového typu Array[0..3] of Struct (Obr. 4.16). Nové premenné do štruktúry pridávame po rozbalení premennej D pod D[0] kliknutím na Add new. Pridajme premenné OBS, PRIPRAVENY_ODOSLAT a POCET_SPUSTENI príslušných dátových typov (Obr. 4.17). Uvedeným postupom sme vytvorili premenné, ktorých celý názov je DATA.D[0].OBS, DATA.D[0].PRIPRAVENY_ODOSLAT, DATA.D[0].POCET_SPUSTENI, DATA.D[1].OBS atď. Ide o efektívnu metódu tvorby štruktúrovaných premenných. Podobne ako v predchádzajúcom príklade môžeme rozšíriť premenné (štruktúru) ako aj navýšiť počet objektov (veľkosť array).

DATA

	Name	Data type	Start value	Retain
1	Static			
2	D	Array[0..3] of Struct		
3	D[0]	Struct		
4	<Add new>			
5	D[1]	Struct		
6	D[2]	Struct		
7	D[3]	Struct		

Obr. 4.16. Príklad pola (array) v DB

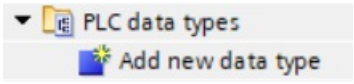
DATA

	Name	Data type	Start value	Retain
1	Static			
2	D	Array[0..3] of Struct		
3	D[0]	Struct		
4	OBS	Bool	false	
5	PRIPRAVENY_ODOSLAT	Bool	false	
6	POCET_SPUSTENI	Int	0	
7	D[1]	Struct		
8	OBS	Bool	false	
9	PRIPRAVENY_ODOSLAT	Bool	false	
10	POCET_SPUSTENI	Int	0	
11	D[2]	Struct		
12	D[3]	Struct		

Obr. 4.17. Príklad priradených premenných v Struct

4.1.6 Príklad 6 - Pamäťový priestor %DB

Posledným príkladom je aplikácia používateľom vytvoreného dátového typu namiesto dátového typu Struct. Nový dátový typ vytvoríme kliknutím na Add new data type v menu PLC data types (Obr. 4.18). Pridaním a premenovaním UDT na NASA_STRUKTURA môžeme pristúpiť k editácii premenných. Vytvorme premenné OBS, PRIPRAVENY_ODOSLAT a POCET_SPUSTENI príslušných dátových typov (Obr. 4.19). V dátovom bloku modifikujme premennú D na dátový typ Array[0..3] of "NASA_STRUKTURA". Výsledok je na Obr. 4.20. Výhodou uvedeného návrhu je možnosť presunu UDT do knižnice. Umožňuje to jeho opätovné využitie v iných projektoch ako aj nasadenie v HMI.



Obr. 4.18. Časť stromovej štruktúry TIA Portal - Tvorba UDT

NASA_STRUKTURA

	Name	Data type	Default value
1	OBS	Bool	false
2	PRIPRAVENY_ODOSLAT	Bool	false
3	POCET_SPUSTENI	Int	0

Obr. 4.19. Premenné vo vytvorenom dátovom type

DATA

	Name	Data type	Start value	Retain
1	Static			
2	D	Array[0..3] of "NASA_STRUKTURA"		
3	D[0]	"NASA_STRUKTURA"		
4	OBS	Bool	false	
5	PRIPRAVENY_ODOSLAT	Bool	false	
6	POCET_SPUSTENI	Int	0	
7	D[1]	"NASA_STRUKTURA"		
8	OBS	Bool	false	
9	PRIPRAVENY_ODOSLAT	Bool	false	
10	POCET_SPUSTENI	Int	0	
11	D[2]	"NASA_STRUKTURA"		
12	D[3]	"NASA_STRUKTURA"		

Obr. 4.20. Pole používateľom definovaného dátového typu

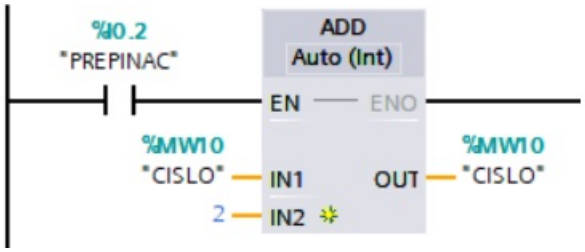
4.2 Analýza programov

V nasledujúcich príkladoch uvedieme analýzy jednoduchých programov. Bude daný program a počiatočné hodnoty premenných, na základe ktorých bude cieľom vysvetliť ako sa bude program vykonávať v CPU riadiaceho systému. Ak nebude uvedené inak, všetky vstupy budú uvažované ako normálne otvorené kontakty a počiatočné hodnoty premenných budú nulové (FALSE, 0 resp. 0.0).

4.2.1 Príklad 1

Analyzujme program na Obr. 4.21. Aký je najmenší počet prepnutí prepínača zo stavu FALSE do stavu TRUE, aby v premennej CISLO bola hodnota 10?

Riešenie: Ak má vstup PREPINAC stav FALSE, inštrukcia *ADD* sa nevykoná. Ak má vstup PREPINAC stav TRUE, privedie sa logická hodnota TRUE na vstup EN inštrukcie *ADD*, ktorá vykoná operáciu CISLO := CISLO +2. Inštrukcia *ADD* sa vykoná v každom cykle keď má na vstupe EN stav TRUE, preto stačí priviesť prepínačom hodnotu TRUE raz a po 5. cykle premenná CISLO nadobudne hodnotu 10. Samozrejme ak naďalej platí PREPINAC = TRUE, premenná v ďalších cykloch nadobúda hodnoty 12, 14 atď. **Výsledok, ako hodnota premennej, nezávisí od počtu prepnutí prepínača, ale od počtu cyklov, počas ktorých zostáva prepínač v stave TRUE a inštrukcia *ADD* sa vykonáva.**

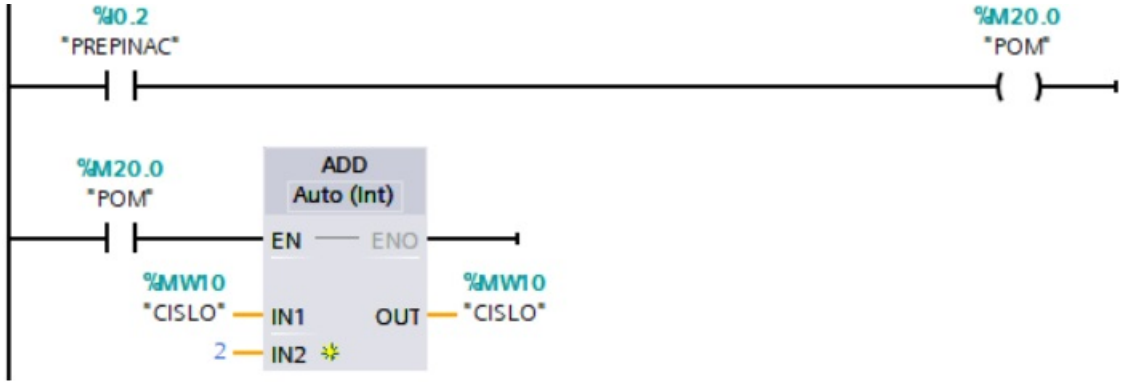


Obr. 4.21. Príklad analýzy programu v LD

4.2.2 Príklad 2

Analyzujme program na Obr. 4.22. Aký je najmenší počet prepnutí prepínača zo stavu FALSE do stavu TRUE, aby v premennej CISLO bola hodnota 10?

Riešenie: Do pomocnej premennej POM sa v každom cykle zapisuje stav vstupu PREPINAC. V spodnej vetve sa načíta hodnota pomocnej premennej POM, od ktorej závisí vykonanie inštrukcie *ADD*. Ak má premenná POM hodnotu TRUE (tzn. aj vstup PREPINAC má hodnotu TRUE) vykoná sa operácia CISLO := CISLO +2. Aj v tomto príklade stačí prepnúť prepínač jeden raz do stavu TRUE a po 5. cykle premenná CISLO nadobudne hodnotu 10. Z pohľadu funkcionality ide o podobný prístup ako v predošlom príklade. **Výsledok, ako hodnota premennej, nezávisí od počtu prepnutí prepínača, ale od počtu cyklov, počas ktorých zostáva prepínač v stave TRUE a inštrukcia *ADD* sa vykonáva.**

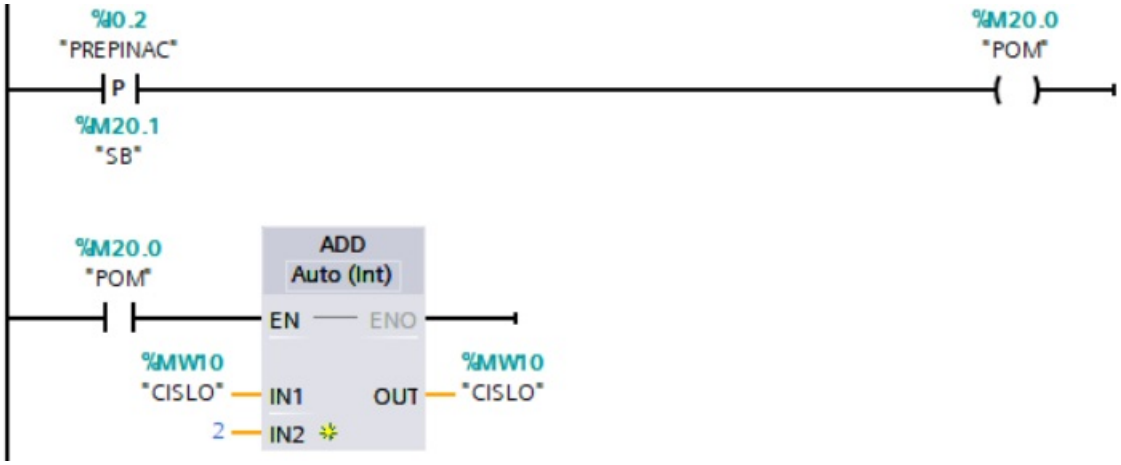


Obr. 4.22. Príklad analýzy programu v LD

4.2.3 Príklad 3

Analyzujme program na Obr. 4.23. Aký je najmenší počet prepnutí prepínača zo stavu FALSE do stavu TRUE, aby v premennej CISLO bola hodnota 10?

Riešenie: Do pomocnej premennej POM sa zapisuje výsledok detekcie nábežnej hrany nad hodnotou vstupu PREPINAC. Premenná POM nadobudne hodnotu TRUE len v prípade, že sa stav vstupu PREPINAC zmenil z FALSE na TRUE. V druhej vetve sa načíta stav premennej POM, ktorý ak má hodnotu TRUE, vykoná výpočet CISLO := CISLO +2. Na to, aby sme mali hodnotu 10 v premennej CISLO, musíme 5-krát priviesť hodnotu TRUE, t. j. 5 krát prepnúť prepínač zo stavu FALSE na TRUE.

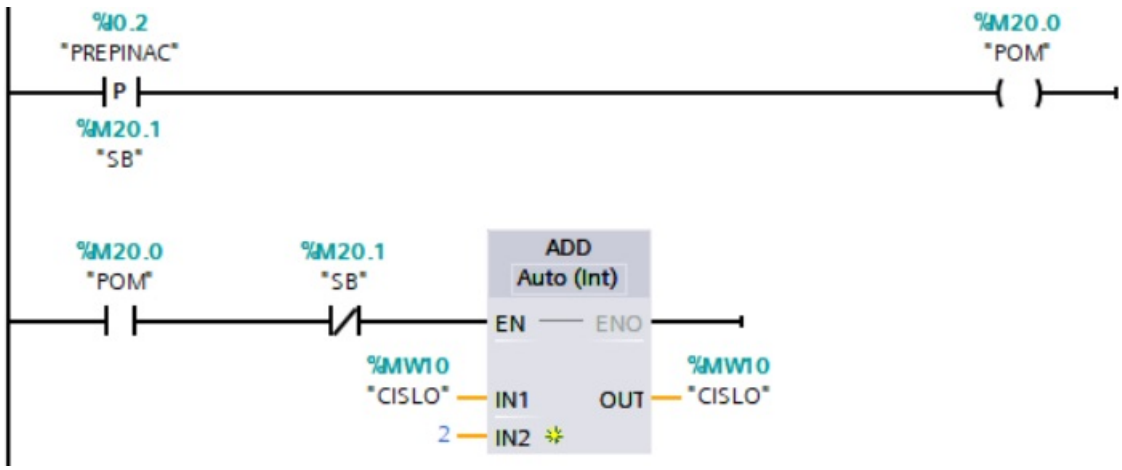


Obr. 4.23. Príklad analýzy programu v LD

4.2.4 Príklad 4

Analyzujme program na Obr. 4.24. Aký je najmenší počet prepnutí prepínača zo stavu FALSE do stavu TRUE, aby v premennej CISLO bola hodnota 10?

Riešenie: Na rozdiel od predchádzajúceho príkladu je na Obr. 4.24 pred inštrukciou *ADD* AND podmienka: POM AND NOT(SB). Ak platí podmienka, vykoná sa inštrukcia *ADD*. Hodnota POM je TRUE ak sa deteguje nábežná hrana nad vstupom PREPINAC. Nábežná hrana sa generuje vtedy, ak PREPINAC = TRUE a SB = FALSE (pomocný bit detekcie hrany). Premenná SB je inštrukciou čítaná, ale aj zapisovaná. V každom cykle sa do SB zapisuje hodnota PREPINAC. Preto podmienka pred 4.24 nemôže nikdy platiť, lebo ak je nábežná hrana, hodnota SB nadobudne hodnotu TRUE.

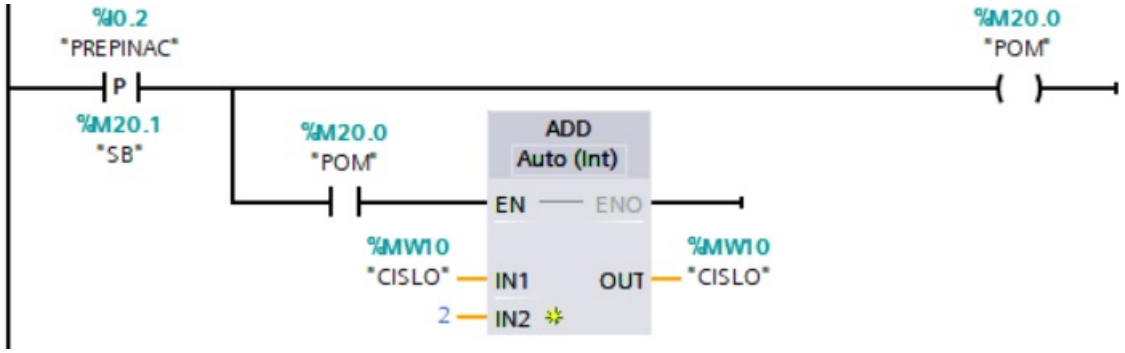


Obr. 4.24. Príklad analýzy programu v LD

4.2.5 Príklad 5

Analyzujme program na Obr. 4.25. Aký je najmenší počet prepnutí prepínača zo stavu FALSE do stavu TRUE, aby v premennej CISLO bola hodnota 10?

Riešenie: Príklad je podobný príkladu 3 v kap. 4.2.5. Do premennej POM sa zapisuje výsledok detekcie nábežnej hrany. V spodnej vetve program pokračuje načítaním hodnoty POM. Musí platiť AND detekcie nábežnej hrany a POM = TRUE, ktorý bude platiť vždy ak je nábežná hrana, t. j. vstup PREPINAC zmení stav z FALSE na TRUE. Na to, aby sme mali hodnotu 10 v premennej CISLO, musíme 5-krát priviesť hodnotu TRUE, t. j. 5-krát prepnúť prepínač zo stavu FALSE na TRUE.

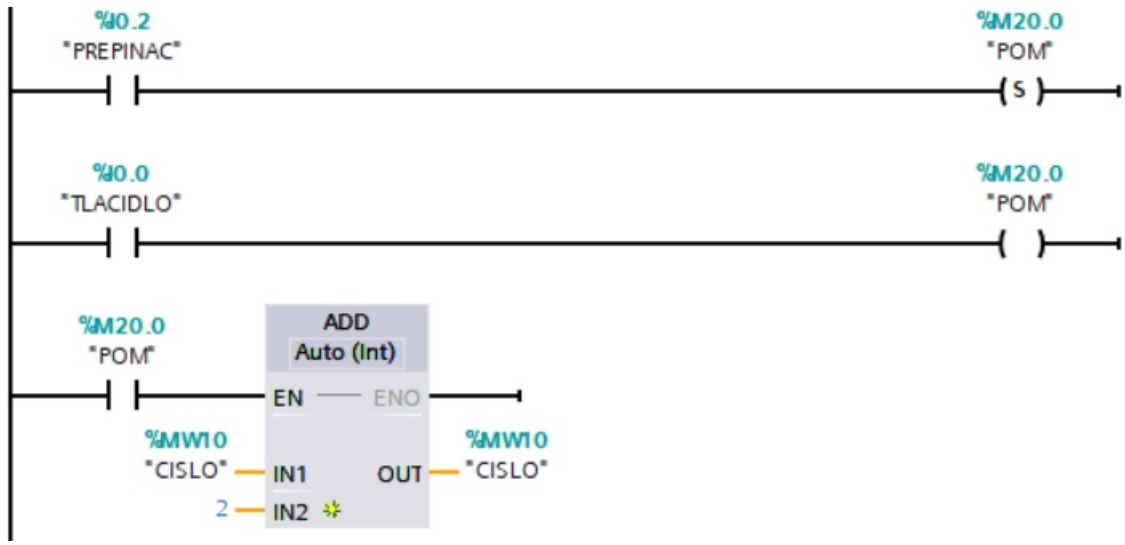


Obr. 4.25. Príklad analýzy programu v LD

4.2.6 Príklad 6

Analyzujme program na Obr. 4.26. Aký je najmenší počet prepnutí prepínača zo stavu FALSE do stavu TRUE, aby v premennej CISLO bola hodnota 10?

Riešenie: Ak je prepínač zopnutý (PREPINAC = TRUE), do pomocnej premennej POM sa inštrukciou *SET* priradí hodnota TRUE. Podľa stavu vstupu TLACIDLO sa do POM zapíše FALSE alebo TRUE, t. j. ak je tlačidlo zatlačené, POM bude TRUE, inak FALSE. Ak je POM = TRUE, vykoná sa inštrukcia *ADD*. Čiže hodnotu premennej CISLO neovplyvňuje vstup PREPINAC, ale len TLACIDLO. Ak zatlačíme a podržíme dlhšie tlačidlo, potom po 5. cykle bude v premennej CISLO hodnota 10. **Inštrukcia *ADD* sa nevykonáva v závislosti od stavu prepínača, ale len od stavu tlačidla, ktorý v každom cykle prepisuje hodnotu premennej POM.**

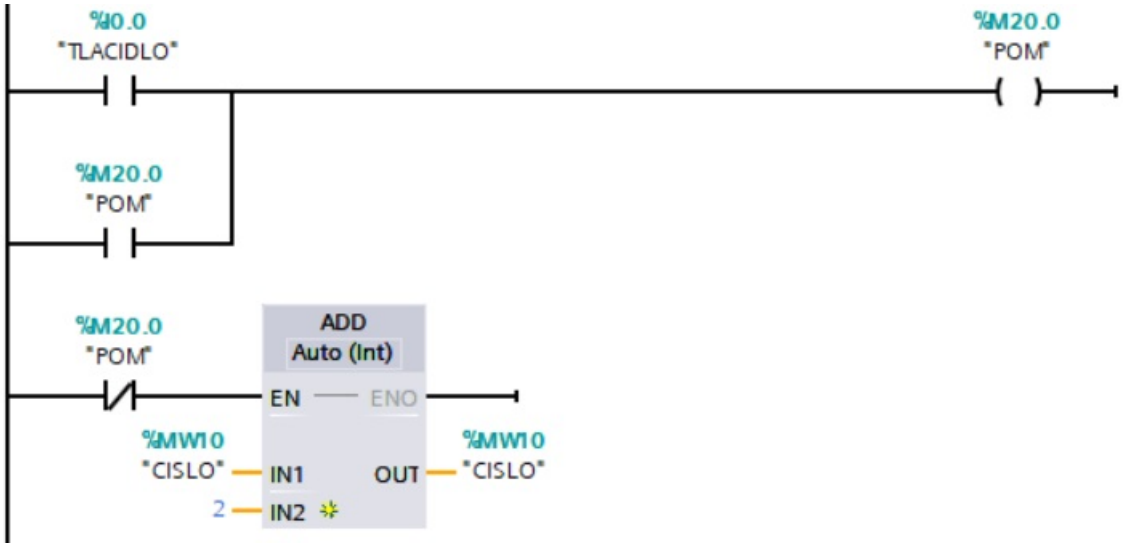


Obr. 4.26. Príklad analýzy programu v LD

4.2.7 Príklad 7

Analyzujme program na Obr. 4.27. Ako závisí zmena hodnoty premennej CISLO od vstupu TLACIDLO?

Riešenie: Ak sú všetky premenné FALSE, potom neplatí podmienka TLACIDLO OR POM, t. j. do POM sa v každom cykle zapisuje hodnota FALSE. Ak POM = FALSE, výstupom inštrukcie *Normálne zatvorený kontakt* je TRUE, preto sa v každom cykle vykoná inštrukcia *ADD*. Inštrukcia realizuje výpočet CISLO := CISLO +2. Premenná CISLO je dátového typu INT. Po dosiahnutí maximálnej hodnoty 32 768 by došlo k pretečeniu a hodnota by nadobudla zápornú hodnotu, ktorá by sa v každom cykle navyšovala od -32 768 do 32 768. Zatlačením tlačidla TLACIDLO v danom cykle (označme ho n) platí horná časť OR vetvy, t. j. do POM sa zapíše TRUE. Nakoľko hodnota POM = TRUE, výstupom inštrukcie *Normálne zatvorený kontakt* je FALSE, preto sa ADD nevykoná. V ďalšom cykle ($n+1$) ak by sme stihli zmeniť stav tlačidla na FALSE, bude platiť spodná časť OR vetvy, lebo do POM sa v predchádzajúcom cykle zapísala hodnota TRUE. V praxi pri zatlačení a uvoľnení tlačidla má vstup TLACIDLO hodnotu TRUE počas viacerých cyklov, t. j. platí aj horná aj dolná časť OR vetvy. Vyššie uvedenú analýzu môžeme zhrnúť takto. Pokiaľ sa tlačidlo nezatlačí, v každom cykle sa vykoná inštrukcia *ADD*, až kým sa tlačidlo nezatlačí.



Obr. 4.27. Príklad analýzy programu v LD

4.2.8 Príklad 8

Analyzujme program na Obr. 4.28. Ako závisí stav výstupu LED od vstupu TLACIDLO?

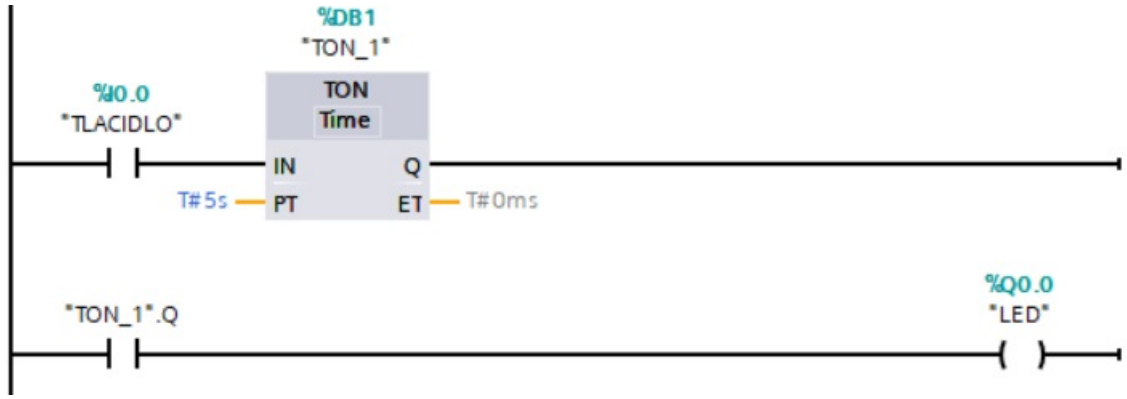
Riešenie: Ak je tlačilo nezatlačené, časovač *TON* je vynulovaný, t. j. výstup Q = FALSE sa v každom cykle zapisuje do premennej LED. LED nesvieti. Aby sa LED rozsvietila musí výstup časovača Q nadobudnúť hodnotu TRUE, t. j. musí sa dokončiť časovanie. Zatlačením a podržaním tlačidla TLACIDLO aspoň na 5 sekúnd sa aktivuje výstup časovača a zapne sa LED. Uvoľnením tlačidla do 5 s od stlačenia sa časovanie nedokončí, lebo sa časovač *TON* vynuluje. Uvoľnením tlačidla po dokončení časovania sa časovač vynuluje a výstup LED zhasne (LED = FALSE).

Obr. 4.28. Príklad analýzy programu v LD

4.2.9 Príklad 9

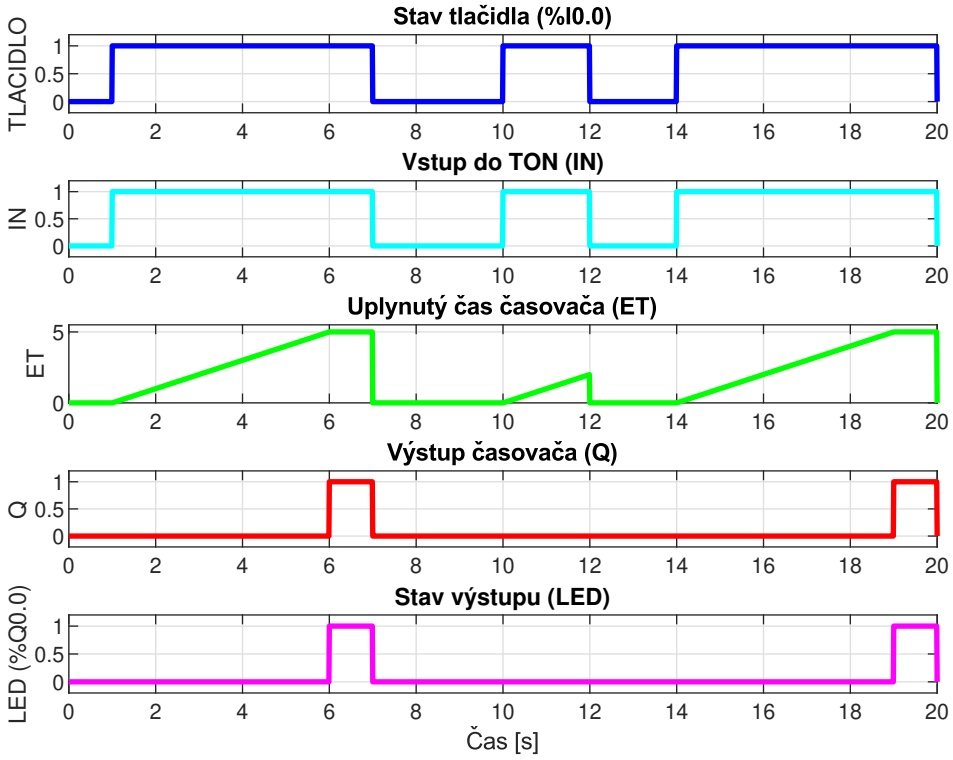
Analyzujme program na Obr. 4.29. Ako závisí stav výstupu LED od vstupu TLACIDLO?

Riešenie: Z pohľadu funkcionality je príklad totožný s funkcionalitou z predošlého príkladu 4.2.8. Hlavným rozdielom je rozdelenie programu na dve vetvy. V spodnej vetve sa inštrukciou *Normálne otvorený kontakt* zisťuje stav výstupu časovača TON_1.Q, ktorý je priradený do premennej LED.



Obr. 4.29. Príklad analýzy programu v LD

Príklad časového priebehu signálov je na Obr. 4.30.

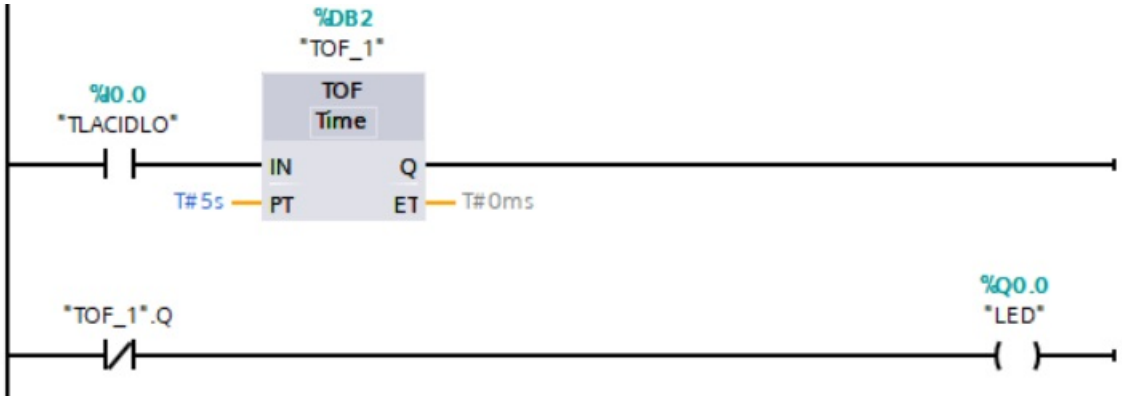


Obr. 4.30. Príklad časového priebehu signálov

4.2.10 Príklad 10

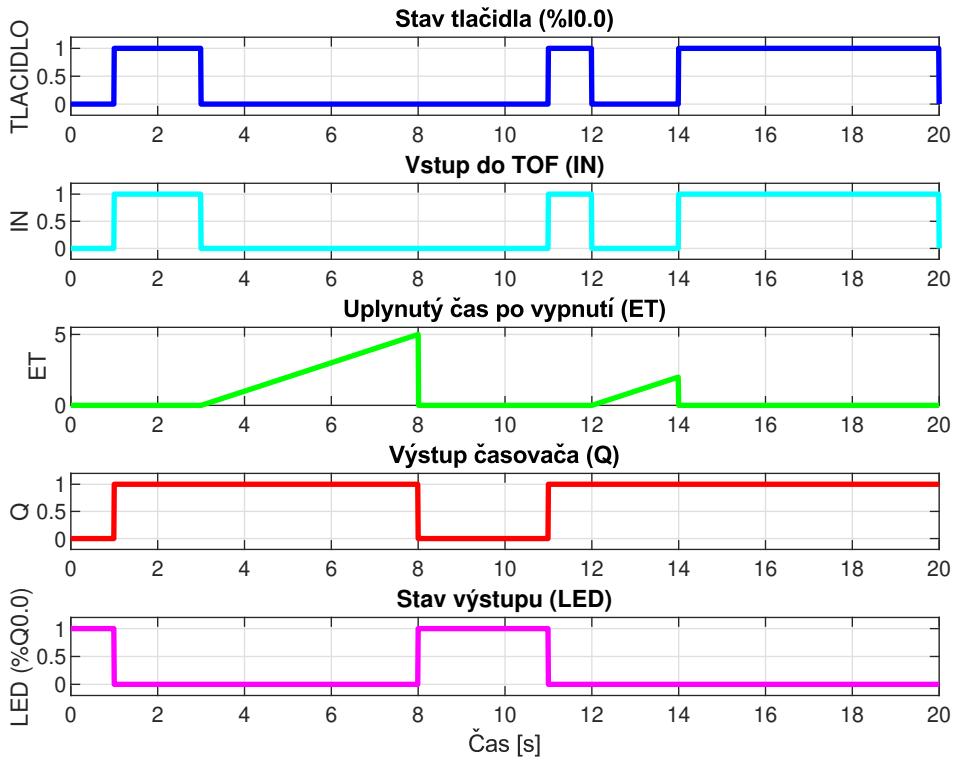
Analyzujme program na Obr. 4.31. Ako závisí stav výstupu LED od vstupu TLACIDLO?

Riešenie: Ak je na začiatku tlačidlo nezatlačené, na vstupe časovača *TOF* je hodnota FALSE, preto na jeho výstupe je hodnota FALSE. V spodnej vetve je výstupom inštrukcie *Normálne zatvorený kontakt* hodnota TRUE, t. j. LED svieti. Zatlačením a držaním tlačidla bude na vstupe časovača *TOF* hodnota TRUE. Výstup časovača Q bude tiež TRUE. V spodnej vetve je výstupom inštrukcie *Normálne zatvorený kontakt* hodnota FALSE, t. j. LED zhasne. Uvoľnením tlačidla začne časovač navyšovať čas od 0 s do 5 s. Ak počas uvedeného intervalu zatlačíme tlačidlo, časovač sa vynuluje a naďalej drží na výstupe hodnotu TRUE, t. j. LED je zhasnutá. Ak uvoľníme tlačidlo na aspoň 5 sekúnd, výstup časovača sa zmení na FALSE. V spodnej vetve bude výstupom inštrukcie *Normálne zatvorený kontakt* hodnota TRUE, t. j. LED sa zapne.



Obr. 4.31. Príklad analýzy programu v LD

Príklad časového priebehu signálov je na Obr. 4.32.



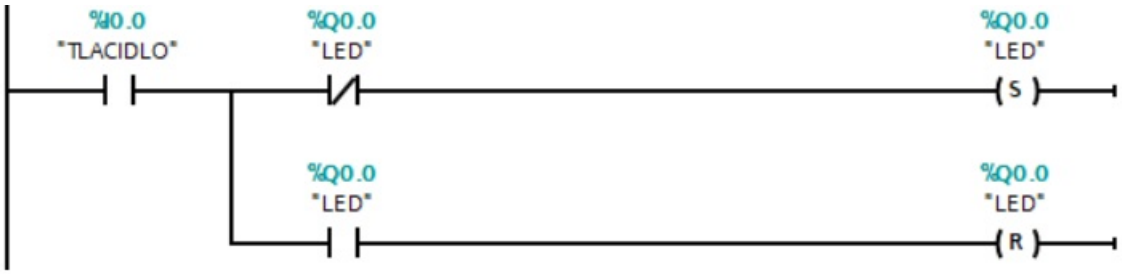
Obr. 4.32. Príklad časového priebehu signálov

4.2.11 Príklad 11

Analyzujme program na Obr. 4.33. Ako závisí stav výstupu LED od vstupu TLACIDLO?

Riešenie: Ak je tlačilo nezatlačené a LED nesvieti, potom neplatí v hornej ani dolnej vetve podmienka AND, t. j. inštrukcie *SET* a *RESET* sa nevykonávajú. Ak zatlačíme tlačidlo a LED nesvieti, vykoná sa inštrukcia *SET*, čím sa LED nastaví na TRUE. V spodnej vetve sa načíta hodnota LED, ktorá je TRUE. Platí podmienka TLACIDLO AND LED,

preto sa vykoná aj inštrukcia *RESET*. Naposledy zapísaná hodnota (FALSE) sa zapíše na výstup, preto je LED zhasnutá. Táto sekvencia sa opakuje v každom cykle kedy je tlačidlo zatlačené. Ak by sa manuálne nastavila hodnota LED na TRUE (napr. pomocou watch table), potom by stále svietila až kým by sme nezatlačili tlačidlo, lebo sa vykoná inštrukcia *RESET* (pozn. inštrukcia *SET* v hornej vetve sa nevykoná lebo sme LED manuálne nastavili na TRUE a neplatí podmienka pre jej vykonanie). **Pri zatlačenom tlačidle sa v každom cykle vykoná *SET* aj *RESET*, preto sa LED nikdy nezapne.**

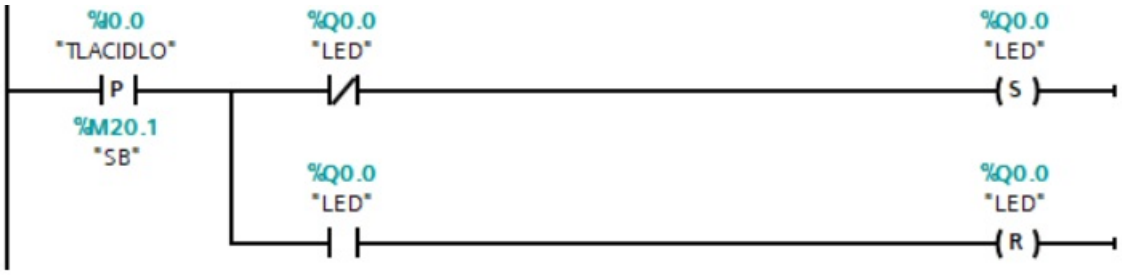


Obr. 4.33. Príklad analýzy programu v LD

4.2.12 Príklad 12

Analyzujme program na Obr. 4.34. Ako závisí stav výstupu LED od vstupu TLACIDLO?

Riešenie: V príklade 11 (kap. 4.2.11) sa pri držaní tlačidla vykonali inštrukcie *SET* a *RESET* v každom cykle. V tomto príklade sa inštrukcie vykonajú len raz a to pri nábežnej hrane vstupu TLACIDLO, t. j. ak TLACIDLO zmení stav z FALSE na TRUE a LED nesvieti, vykoná sa inštrukcia *SET*. Program pokračuje v spodnej vetve. Načíta sa hodnota LED (TRUE) a vykoná sa inštrukcia *RESET* (pozn. lebo plati nábežná hrana AND LED). Ak držíme tlačidlo, v ďalších cykloch je výstupom inštrukcie *Detekcia nábežnej hrany* stav FALSE, nemôže platiť AND v hornej a dolej vetve, preto sa inštrukcie *SET* a *RESET* nevykonávajú. **V momente zatlačenia tlačidla sa vykoná *SET* aj *RESET*, preto sa LED nikdy nezapne.**



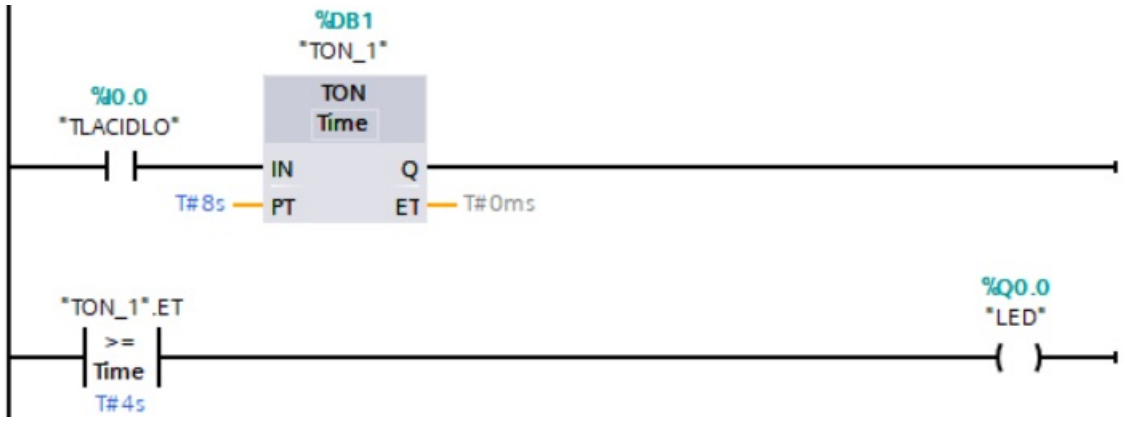
Obr. 4.34. Príklad analýzy programu v LD

4.2.13 Príklad 13

Analyzujme program na Obr. 4.35. Ako závisí stav výstupu LED od vstupu TLACIDLO?

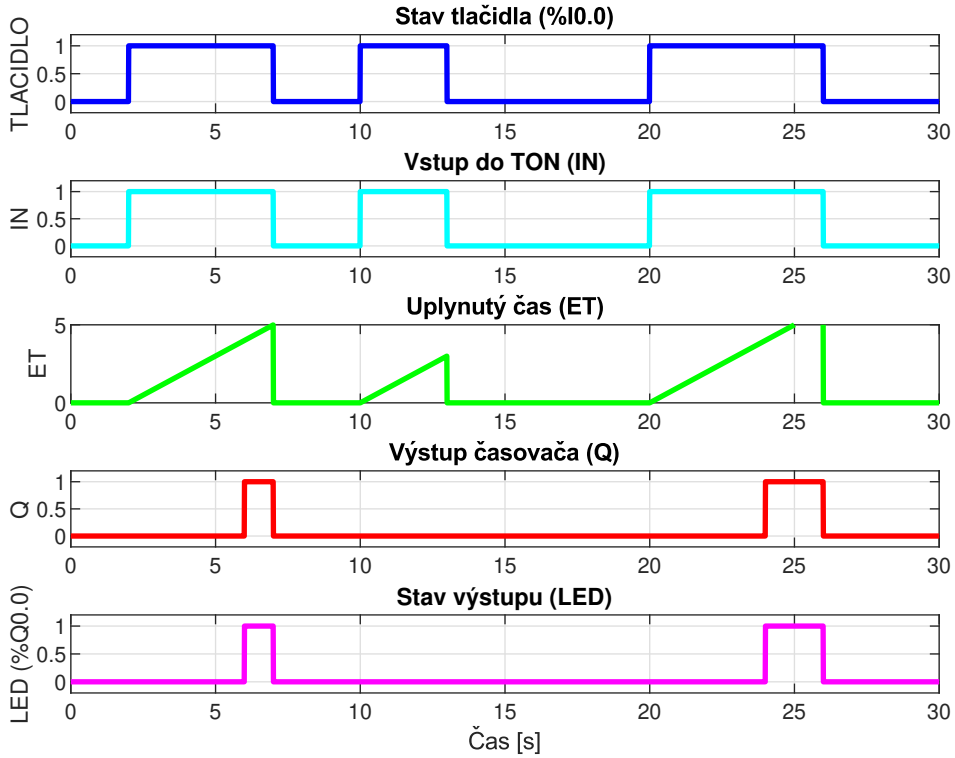
Riešenie: Ak je tlačidlo nezatlačené, časovač *TON* je vynulovaný (ET=0 s, Q=FALSE). V spodnej vetve neplatí porovnanie, lebo aktuálny čas časovača (0 s) nie je väčší alebo rovný ako 4 s, preto je výstupom porovnávacieho bloku FALSE. Do premennej LED sa

zapisuje v každom cykle hodnota FALSE. Ak zatlačíme tlačidlo, spustí sa časovač. Ak držíme tlačidlo aspoň 4 s, časovač naakumuluje v ET hodnotu 4 s a viac, t. j. po 4 s bude výstupom porovnávacieho bloku TRUE. Do LED sa zapíše TRUE, t. j. LED zasvieti. LED bude svietiť pokiaľ tlačidlo neuvoľníme, lebo po vynulovaní časovača prestane platiť výsledok porovnania. Stručné zhrnutie: Zatlačením (a držaním) tlačidla aspoň na 4 s sa LED rozsvieti a potom svieti, pokiaľ tlačidlo neuvoľníme.



Obr. 4.35. Príklad analýzy programu v LD

Príklad časového priebehu signálov je na Obr. 4.36.

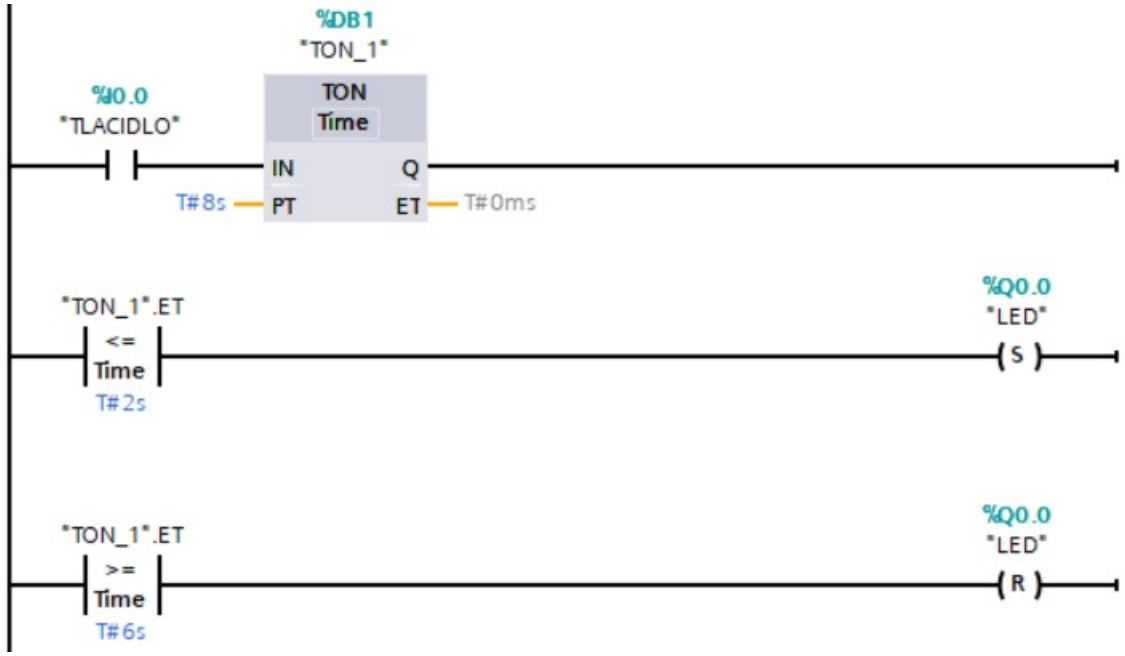


Obr. 4.36. Príklad časového priebehu signálov

4.2.14 Príklad 14

Analyzujme program na Obr. 4.37. Ako závisí stav výstupu LED od vstupu TLACIDLO?

Riešenie: Ak je tlačidlo nezatlačené, časovať *TON* nie je spustený, t. j. hodnota ET = 0 ms. Z nižších podmienok platí horná podmienka porovnania, preto sa v každom cykle vykonáva inštrukcia *SET* (LED svieti). Zatlačením a držaním tlačidla začne časovač navyšovať čas od 0 ms po 8 s. Do 2 s od zatlačenia tlačidla sa pre hornú podmienku neustále vykonáva inštrukcia *SET* (LED svieti). Po 2 s neplatia podmienky v porovnávacích blokoch. Prestala sa vykonávať inštrukcia *SET*, ale LED naďalej svieti, lebo hodnota LED ostáva TRUE. Po 6 s od zatlačenia tlačidla platí spodná podmienka a v každom cykle sa vykonáva inštrukcia *RESET*. LED zhasne. Ak tlačidlo uvoľníme, časovač TON sa vynuluje čím sa LED rozsvieti. Stručné zhrnutie: Pri nezatlačení tlačidla LED svieti. Zatlačením a držaním tlačidla do 6s LED svieti a potom zhasne. Kedykoľvek sa tlačidlo uvoľní, LED sa rozsvieti.

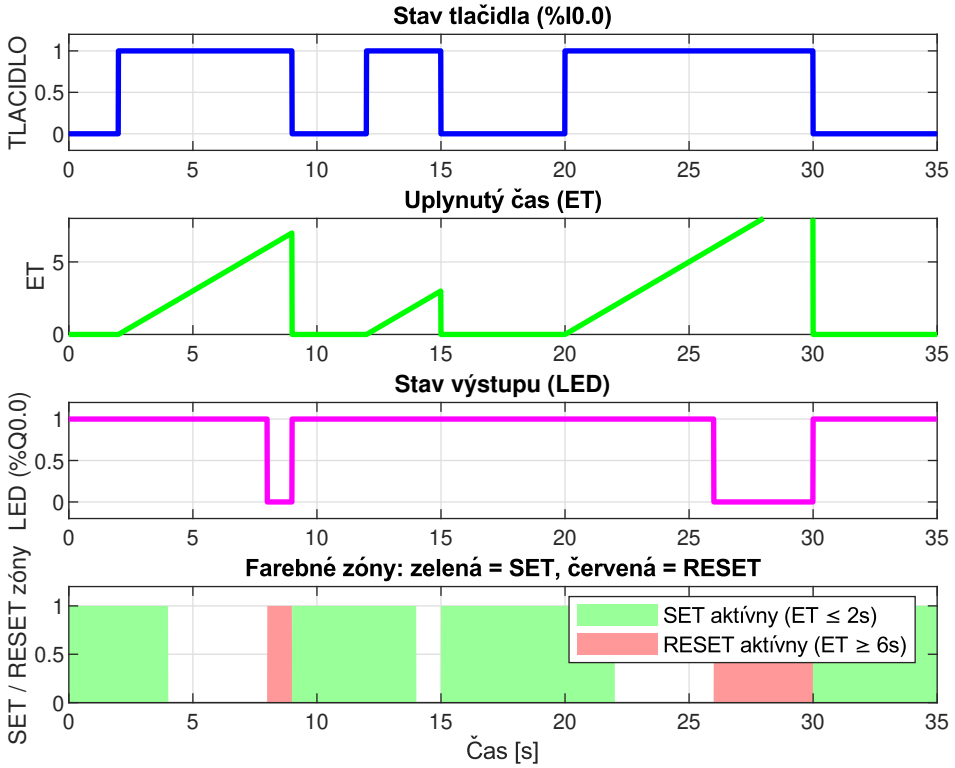


Obr. 4.37. Príklad analýzy programu v LD

Príklad časového priebehu signálov je na Obr. 4.38. Na začiatku, keď je tlačidlo neuvoľnené (TLACIDLO = 0), časovač je vynulovaný ($ET = 0\ s$) a vďaka platnosti podmienky $ET \leq 2\ s$ sa vykonáva inštrukcia *SET* – LED svieti.

- V prvom scenári (t = 2 s až 9 s) je tlačidlo stlačené na 7 sekúnd. Časovač začne počítať:
  - Do 6 s od stlačenia ($ET \leq 6\ s$), sa ešte nevykonáva žiadna RESET inštrukcia – LED zostáva rozsvietená.
  - Po 6 s (t = 8 s), začne platiť podmienka $ET \geq 6\ s$, a vykonáva sa RESET – LED zhasne.
  - Po uvoľnení tlačidla (t = 9 s), sa časovač vynuluje (ET = 0 s) a opäť platí podmienka *SET* – LED sa rozsvieti.

- V druhom scenári (t = 12 s až 15 s), je tlačidlo stlačené len na 3 s:
  - Počas celého stlačenia platí podmienka $ET \leq 2\ s$, takže LED ostáva svietiť vďaka opakovanému vykonávaniu *SET*.
  - *RESET* podmienka nikdy neplatí, takže LED nezhasne.
- V treťom scenári (t = 20 s až 30 s), je tlačidlo držané 10 s:
  - Po 6 s stlačenia (t = 26 s), začne platiť podmienka $ET \leq 6\ s$, vykonáva sa *RESET* a LED zhasne.
  - Po uvoľnení tlačidla (t = 30 s), sa opäť vykoná *SET* vďaka vynulovanému časovaču a LED sa rozsvieti.



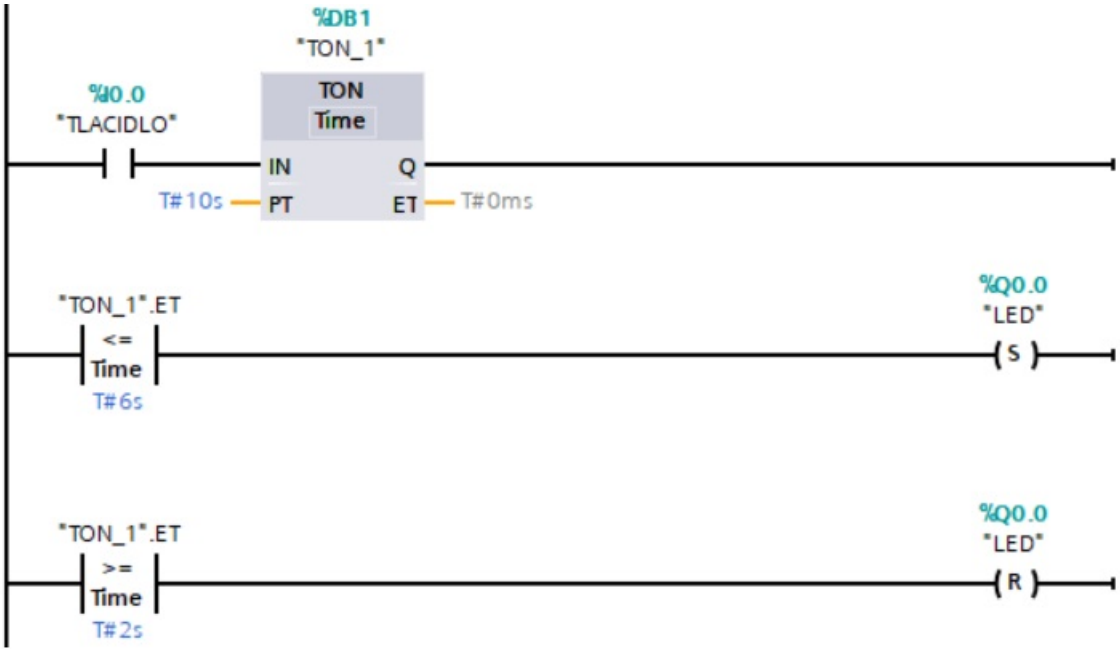
Obr. 4.38. Príklad časového priebehu signálov

4.2.15 Príklad 15

Analyzujme program na Obr. 4.39. Ako závisí stav výstupu LED od vstupu TLACIDLO?

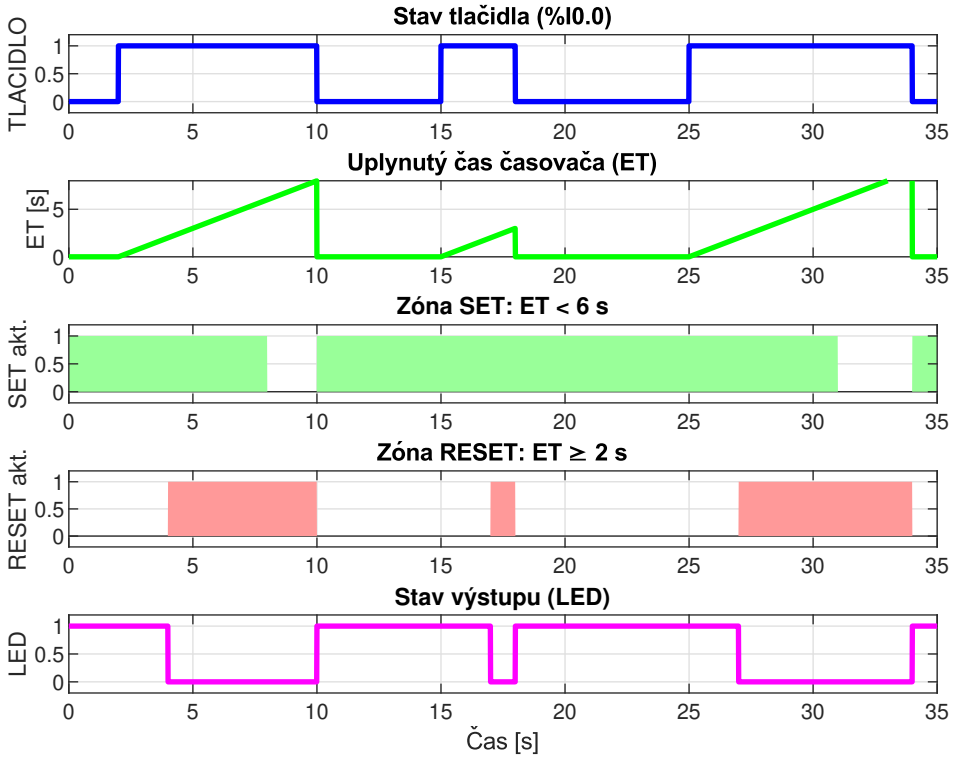
Riešenie: Ak je tlačidlo nezatlačené, časovač je vynulovaný. Z podmienok platí horná podmienka lebo aktuálny čas časovača (0 ms) je menej ako 6 s. V každom cykle sa vykonáva inštrukcia *SET*, t. j. LED svieti. Zatlačením a držaním tlačidla sa spustí časovač a začne sa navyšovať čas časovača. Od 2 s sa neustále vykonáva inštrukcia *SET*. Po 2 s (vrátane) spustenia časovača platia obe podmienky. Tu je dôležité poradie vykonania inštrukcií. Nakoľko platí horná podmienka vykoná sa inštrukcia *SET*. Keďže platí aj spodná

podmienka, vykoná sa inštrukcia *RESET*, ktorá zmení hodnotu LED na FALSE. Táto hodnota sa reálne na konci programového cyklu zapíše na digitálny výstup, preto bude LED zhasnutá. Po 6 s prestáva platiť horná podmienka, ale naďalej platí spodná, preto sa v každom cykle vykonáva *RESET*, t. j. LED je naďalej zhasnutá. Ak uvoľníme tlačidlo, časovač sa vynuluje a začne platiť horná podmienka, t. j. LED sa rozsvieti. Stručné zhrnutie: Pri nezatlačení tlačidla LED svieti. Zatlačením a držaním tlačidla LED svieti 2 s a potom zhasne. Uvoľnením tlačidla sa LED rozsvieti.



Obr. 4.39. Príklad analýzy programu v LD

Príklad časového priebehu signálov je na Obr. 4.40. Na základe časového priebehu možno pozorovať, že LED svieti, keď je tlačidlo neuvoľnené alebo bolo krátko stlačené. Pri dlhšom stlačení (viac ako 2 sekundy) dôjde k zhasnutiu LED, lebo sa uplatní RESET aj napriek predchádzajúcemu SET. Po každom uvoľnení tlačidla sa časovač vynuluje a LED sa automaticky opäť rozsvieti.



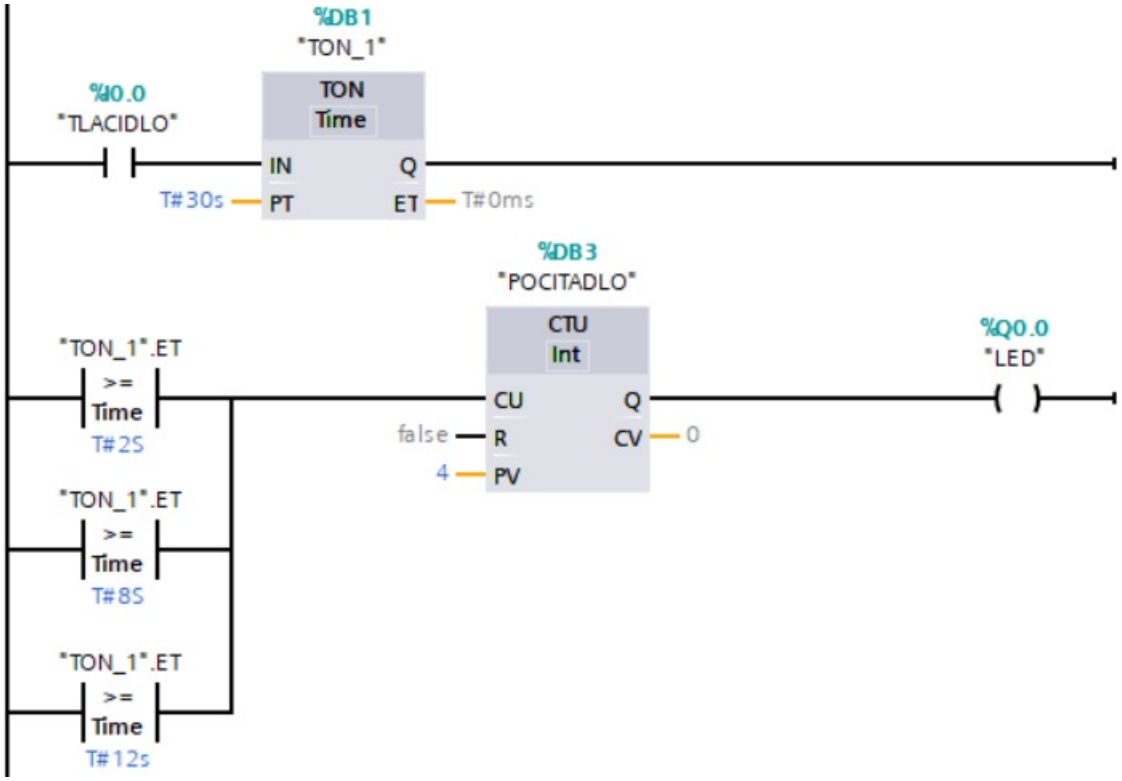
Obr. 4.40. Príklad časového priebehu signálov

4.2.16 Príklad 16

Analyzujme program na Obr. 4.41. Ako závisí stav výstupu LED od vstupu TLACIDLO?

Riešenie: Ak je tlačidlo nezatlačené, časovač je vynulovaný. Z troch porovnávacích blokov neplatí ani jedno z porovnaní, preto ich výstupom je FALSE. Tento signál postupuje na vstup CU počítadla. Výstup Q nie je aktivovaný, preto LED má hodnotu FALSE. Zatlačením a držaním tlačidla sa pustí časovač. Po 2 s bude platiť prvá podmienka. V prvej vetve je výstupom inštrukcie TRUE v nižších FALSE. Výsledkom logického spojenia OR (TRUE OR FALSE OR FALSE) je TRUE, ktoré je privedené na vstup CU. Počítadlo na vstupe CU deteguje nábežnú hranu, preto vykoná jeden raz navýšenie svojej aktuálnej hodnoty z 0 na 1, t. j. CV = 1. Počítadlo nedosiahlo hodnotu 4 (PV=4), preto výstup Q ostáva naďalej deaktivovaný. V ďalšom cykle (nech je to napr. 2,01 s) platia rovnaké stavy. V prvej vetve je výstupom porovnávacej inštrukcie TRUE a nižšie FALSE a preto na vstupe CU je stále TRUE. Po 8 s bude okrem prvej podmienky platiť aj stredná. Výsledkom logického spojenia OR (TRUE OR TRUE OR FALSE) je naďalej TRUE, ale počítadlo nedeteguje nábežnú hranu. Analogický stav platí aj po 12 s. Aby sme dokázali navýšiť hodnotu počítadla, musíme tlačidlo uvoľniť a zatlačiť aspoň na 2 s. Po 2 s sa opäť navýši aktuálna hodnota počítadla. Po 4 stlačeniach tlačidla aspoň na 2 s sa LED rozsvieti. Ak by sme neustále navyšovali hodnotu počítadla na maximálnu hodnotu dátového typu INT, došlo by k pretečeniu a výstup Q by sa zmenil na FALSE. LED by zhasla. Postupným navyšovaním hodnoty počítadla od najmenšej zápornej hodnoty INT až po hodnotu 3 by LED bola zhasnutá. Stručné zhrnutie: Ak neuvažujeme pretečenie dátového typu, tak po

4 stlačeniach tlačidla na aspoň 2 s sa LED rozsvieti a bude stále svietiť.



Obr. 4.41. Príklad analýzy programu v LD

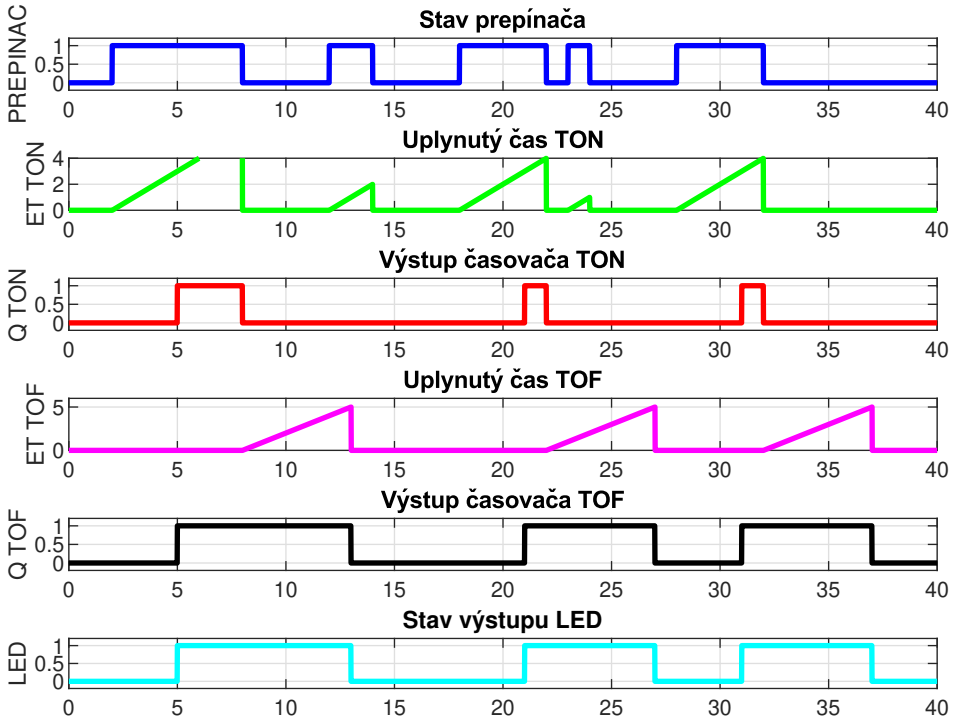
Poznámka: Zvyšné dve podmienky (pre časy 8 s a 12 s) sa pri navyšovaní počítadla neuplatnia, pretože logický signál na vstupe CU je už od 2 s trvale v stave TRUE a počítadlo reaguje len na nábežnú hranu.

4.2.17 Príklad 17

Analyzujme program na Obr. 4.42. Ako závisí stav výstupu LED od vstupu TLACIDLO?

Riešenie: Ak má prepínač stav FALSE, časovač *TON* je vynulovaný, časovač *TOF* je nespustený a preto LED je zhasnutá. V momente prepnutia prepínača do stavu TRUE je na vstupe IN časovača *TON* hodnota TRUE. Časovač navyšuje uplynutý čas od 0 s po 3 s. Po 3 s sa aktivuje výstup Q časovača *TON*, ktorý je privedený na vstup časovača *TOF*. Na jeho výstupe Q bude zapísaná hodnota TRUE, t. j. LED sa rozsvieti. Prepnutím prepínača do stavu FALSE sa privedie FALSE na vstup časovača *TON*, čím sa vynuluje. Jeho výstup Q bude hneď FALSE. To spôsobí na vstupe IN časovača *TOF* zmenu z TRUE na FALSE. Časovač sa spustí, ale nasledujúcich 5 s drží výstup v stave TRUE. Po uplynutí 5 s sa výstup Q zmení na FALSE, t. j. LED zhasne. Stručné zhrnutie: Prepnutím prepínača do stavu TRUE na aspoň 3 s sa po 3 s zapne LED a prepnutím prepínača do stavu FALSE na aspoň 5 s sa po 5 s LED vypne. Príklad časového priebehu signálov je na Obr. 4.43.

Obr. 4.42. Príklad analýzy programu v LD



Obr. 4.43. Príklad časového priebehu signálov

Na začiatku je prepínač v stave `FALSE`, časovač *TON* je vynulovaný ($ET = 0\ s$), časovač *TOF* nespustený a LED zhasnutá.

- V prvom scenári (t = 1 s až 9 s) je prepínač zopnutý na 8 sekúnd:
  - Po 3 sekundách (t = 4 s) sa aktivuje výstup časovača *TON*, čím sa aktivuje aj TOF a LED sa rozsvieti.
  - Po vypnutí prepínača (t = 9 s) sa okamžite vynuluje *TON*, ale *TOF* ešte drží výstup TRUE počas 5 sekúnd.
  - Po uplynutí 5 sekúnd (t = 14 s) zhasne LED, keďže Q z *TOF* sa prepne na FALSE.
- V druhom scenári (t = 17 s až 18 s) je prepínač zopnutý iba na 1 sekundu:

  - Keďže nedôjde k naplneniu prednastaveného času *TON* (3 s), výstup *TON_1.Q* sa nikdy neaktivuje.
  - TOF teda nie je aktivovaný a LED ostáva zhasnutá.

- V treťom scenári (t = 22 s až 29 s) je prepínač zopnutý 7 sekúnd:
  - Po 3 sekundách zopnutia (t = 25 s) sa aktivuje *TON.Q* a potom aj výstup *TOF_1.Q*, čo spôsobí rozsvietenie LED.
  - Po vypnutí prepínača (t = 29 s) sa *TON* vynuluje, ale *TOF* drží výstup aktívny do t = 34 s, kedy LED zhasne.

4.3 Návrh programov

Cieľom tejto kapitoly je tvorba algoritmov na základe slovných zadaní. Prvé príklady sú opísané detailnejšie pre lepšie pochopenie základných princípov. Vybrané príklady sú doplnené o alternatívne riešenia na demonštráciu širokého spektra možných riešení, ktoré by mali byť inšpiráciou pri počiatočnom návrhu nových algoritmov. Príklady sú doplnené aj o online monitorované stavy programu, pre lepšie pochopenie súvislostí.

4.3.1 Príklad 1 - Ovládanie LED

Majme digitálny vstup, ktorým by sme chceli ovládať zapnutie digitálneho výstupu. Nech je vstup TLACIDLO pripojený k digitálnemu vstupnému modulu ako **normálne otvorený kontakt**. Výstup LED má ovládať zopnutie svetla.

Úloha

Ak je tlačidlo zatlačené, LED svieti. Ak je tlačidlo uvoľnené, LED zhasne.

1. riešenie

Na Obr. 4.44 je príklad riešenia pomocou inštrukcií *normálne otvorený kontakt* a *priradenie*. Na obrázku vidno, že stav vstupu TLACIDLO je FALSE, preto výstupom inštrukcie *normálne otvorený kontakt* je tiež FALSE. Táto hodnota FALSE je potom zapísaná inštrukciou *priradenie* do premennej LED.

Obr. 4.44. Príklad ovládania výstupu LED - výstup nie je aktivovaný

Na Obr. 4.45 je zobrazený stav zatlačeného tlačidla a zopnutého výstupu. Stav vstupu TLACIDLO je TRUE, preto výstupom inštrukcie *normálne otvorený kontakt* je tiež TRUE. Hodnota TRUE je potom zapísaná pomocou inštrukcie *priradenie*.

Obr. 4.45. Príklad ovládania výstupu LED - výstup je aktivovaný

2. riešenie

Na Obr. 4.46 je príklad riešenia pomocou inštrukcií *normálne zatvorený kontakt* a *negované priradenie*. Toto riešenie je komplikovanejšie a menej prehľadné ako predchádzajúce riešenie. Je uvedené len na demonštráciu možnosti implementácie algoritmov. Na Obr. 4.46 má vstup TLACIDLO stav FALSE, preto výstupom inštrukcie *normálne zatvorený kontakt* je TRUE. Táto hodnota je negovane zapísaná inštrukciou *negované priradenie* do premennej LED, preto má LED hodnotu FALSE. Hodnota FALSE je animovaná modrým sfarbením inštrukcie.

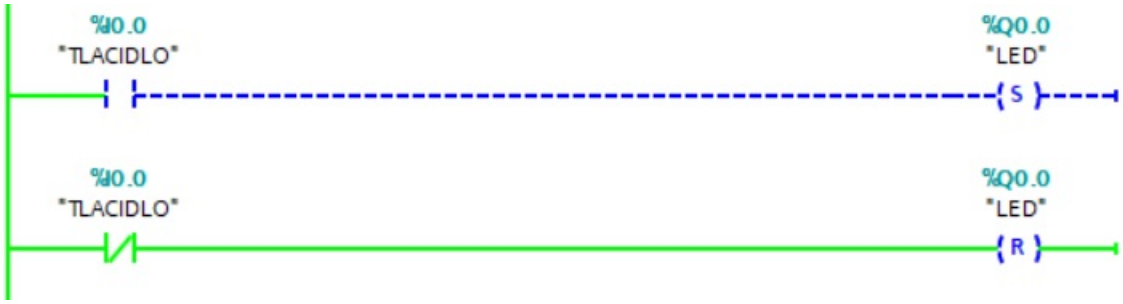
Obr. 4.46. Príklad ovládania výstupu LED - výstup nie je aktivovaný

Animácia zatlačenia tlačidla je na Obr. 4.47. Tlačidlo má stav TRUE, preto výstupom inštrukcie *normálne zatvorený kontakt* je FALSE. Táto hodnota sa negovane zapisuje inštrukciou *negované priradenie* do premennej LED, preto má LED hodnotu TRUE.

Obr. 4.47. Príklad ovládania výstupu LED - výstup je aktivovaný

3. riešenie

Posledné riešenie aplikuje inštrukcie *SET* a *RESET* na zmenu stavu LED. Na Obr. 4.48 je príklad s nezatlačeným tlačidlom. V hornej vetve inštrukcia *normálne otvorený kontakt* načíta stav operandu, preto výstupom inštrukcie je FALSE, lebo tlačidlo nie je zatlačené. Inštrukcia *SET* sa nevykoná, lebo neplatí podmienka pred inštrukciou. V spodnej vetve inštrukcia *normálne zatvorený kontakt* má na výstupe hodnotu TRUE (opačný stav vstupu TLACIDLO). Nakoľko podmienka pre inštrukciou *RESET* je platná, *RESET* sa vykoná. Zjednodušene povedané, pokiaľ je tlačidlo nezatlačené v každom cykle sa vykonáva inštrukcia *RESET*, a preto LED je zhasnutá.



Obr. 4.48. Príklad ovládania výstupu LED - výstup nie je aktivovaný

Opačný stav je na Obr. 4.49. Tlačidlo je zatlačené, preto inštrukcia *normálne otvorený kontakt* má na výstupe stav TRUE. Tento výstup umožní vykonanie inštrukcie *SET*. V spodnej vetve inštrukcia *normálne zatvorený kontakt* má na výstupe stav FALSE, preto sa *RESET* nevykoná. Pri zatlačenom tlačidle sa v každom cykle vykonáva inštrukcia *SET*, preto LED svieti.

Obr. 4.49. Príklad ovládania výstupu LED - výstup je aktivovaný

4.3.2 Príklad 2 - Ovládanie LED

Majme digitálny vstup, ktorým by sme chceli ovládať zapnutie digitálneho výstupu. Nech je vstup TLACIDLO na rozdiel od príkladu uvedeného v kap. 4.3.1 Príklad 1 - Ovládanie LED pripojený k digitálnemu vstupnému modulu ako **normálne zatvorený kontakt**. Výstup LED má ovládať zopnutie svetla.

Úloha

Ak je tlačidlo zatlačené, LED svieti. Ak je tlačidlo uvoľnené, LED zhasne.

Všetky riešenia by sme mohli opísať tak, že v riešeniach Príkladu 4.3.1 Príklad 1 - Ovládanie LED namiesto *normálne zatvorený kontakt* použijeme *normálne otvorený kontakt* a opačne. Uvedieme len 2 riešenia.

1. riešenie

Prvý príklad riešenia je Obr. 4.50. V aktuálnom cykle má vstup TLACIDLO hodnotu TRUE (pozn. tlačidlo nie je zatlačené, ide o NC vstup). Výstupom inštrukcie *normálne zatvorený kontakt* je opačná hodnota operandu, t. j. FALSE. Hodnota FALSE je zapísaná inštrukciou *priradenie* do premennej LED. V aktuálnom stave LED nesvieti.

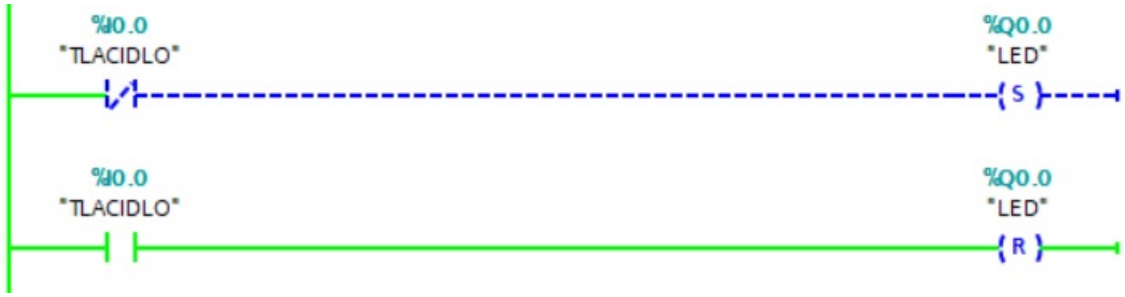
Obr. 4.50. Príklad ovládania výstupu LED - výstup nie je aktivovaný

Na Obr. 4.51 je tlačidlo zatlačené. Hodnota vstupu TLACIDLO je FALSE, preto výstupom inštrukcie *normálne zatvorený kontakt* je TRUE. Táto hodnota je zapísaná do premennej LED. LED svieti.

Obr. 4.51. Príklad ovládania výstupu LED - výstup je aktivovaný

2. riešenie

Na Obr. 4.52 tlačidlo pripojené k digitálnemu vstupnému modulu ako normálne zatvorený kontakt nie je aktivované. Jeho stav je TRUE. Výstupom inštrukcie v druhej vetve je stave TRUE, preto sa inštrukcia *RESET* vykonáva.



Obr. 4.52. Príklad ovládania výstupu LED - výstup nie je aktivovaný

Na Obr. 4.52 je tlačidlo zatlačené. Jeho stav sa zmenil z TRUE na FALSE. Inštrukcia *normálne zatvorený kontakt* má na výstupe stav TRUE, preto sa vykonáva inštrukcia *SET*.

Ak je teda tlačidlo nezatlačené, v každom cykle sa vykonáva inštrukcia *RESET*. Výstup je vypnutý. Ak je tlačidlo zatlačené, v každom cykle sa vykonáva inštrukcia *SET*. Výstup je aktivovaný, LED svieti.

Obr. 4.53. Príklad ovládania výstupu LED - výstup je aktivovaný

4.3.3 Príklad 3 - Zapnutie LED bez *SET*

Majme digitálny vstup, ktorým by sme chceli ovládať zapnutie digitálneho výstupu. Nech je vstup TLACIDLO pripojený k digitálnemu vstupnému modulu ako **normálne otvorený kontakt**. Výstup LED má ovládať zopnutie svetla.

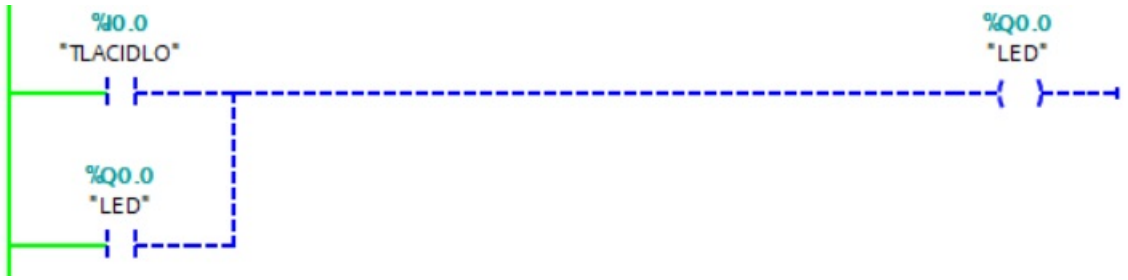
Úloha

Zatlačením tlačidla zapneme LED. Po uvoľnení tlačidla LED naďalej svieti. Program realizujte bez inštrukcie *SET*.

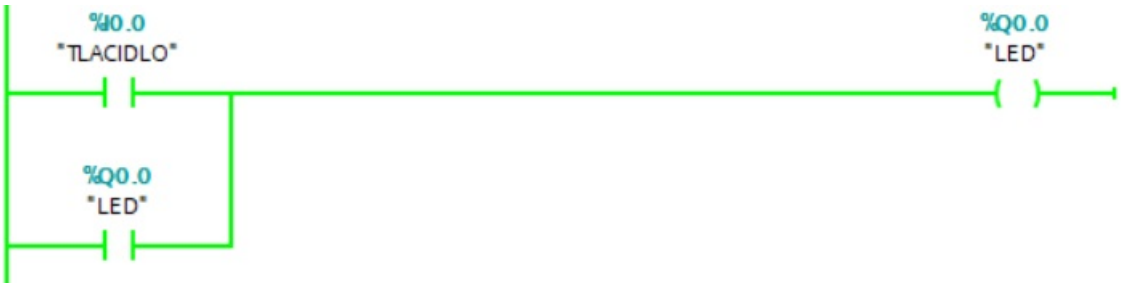
Riešenie pomocou tzv. samodŕžneho kontaktu je na Obr. 4.54. Vstup a výstup majú stav FALSE. LED nesvieti.

Na Obr. 4.55 je tlačidlo zatlačené. V prvom cykle, keď bolo tlačidlo zatlačené platí horná vetva OR podmienky. Dolná, kde je operandom LED, je zatiaľ FALSE. Výstup OR vetvy zapíše TRUE na výstup LED. V ďalšom cykle je tlačidlo stále zatlačené, preto naďalej platí horná vetva. V predošlom cykle sa do LED zapísala hodnota TRUE, preto platí aj spodná časť OR podmienky. LED naďalej svieti, lebo výsledkom OR je TRUE. Tento stav je zaznamenaný na Obr. 4.55. Nakoľko jeden cyklus trvá rádovo 1 ms, bolo by náročné ba nemožné zachytiť moment, keď platí len horná časť OR vetvy (pozn. klasickým monitorovaním programu).

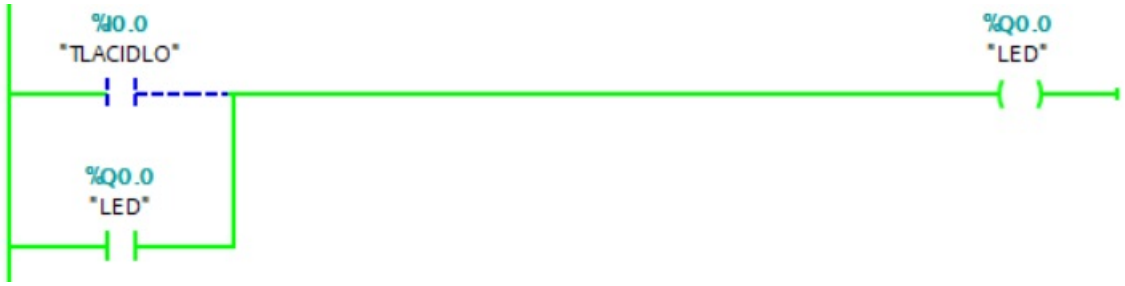
Na Obr. 4.56 je tlačidlo uvoľnené. Horná časť OR podmienky prestáva platiť, len naďalej platí spodná časť. Výsledkom OR je stav TRUE, ktorý sa zapisuje na výstup LED. Signál LED si „drží“ svoju hodnotu.



Obr. 4.54. 1. fáza monitorovania programu



Obr. 4.55. 2. fáza monitorovania programu



Obr. 4.56. 3. fáza monitorovania programu

4.3.4 Príklad 4 - Ovládanie LED bez *SET* a *RESET*

Majme digitálne vstupy TLACIDLO_ZAP a TLACIDLO_VYP, ktorými by sme chceli ovládať zapnutie digitálneho výstupu. Nech sú vstupy pripojené k digitálnemu vstupnému modulu ako **normálne otvorené kontakty**. Výstup LED má ovládať zopnutie svetla.

Úloha

Zatlačením tlačidla TLACIDLO_ZAP zapneme LED. Po uvoľnení tlačidla TLACIDLO_ZAP, stav LED nie je ovlyvnený (LED naďalej svieti). Zatlačením tlačidla TLACIDLO_VYP LED vypneme. Po uvoľnení tlačidla TLACIDLO_VYP stav LED nie je ovlyvnený. Pri zatlačení oboch tlačidiel je výstup vypnutý. Program realizujte bez inštrukcií *SET* a *RESET*.

Využijeme postup z Príkladu 4.3.3 Príklad 3 - Zapnutie LED bez *SET*. K OR vetve pridáme stav tlačidla TLACIDLO_VYP pomocou inštrukcie *normálne zatvorený kontakt*.

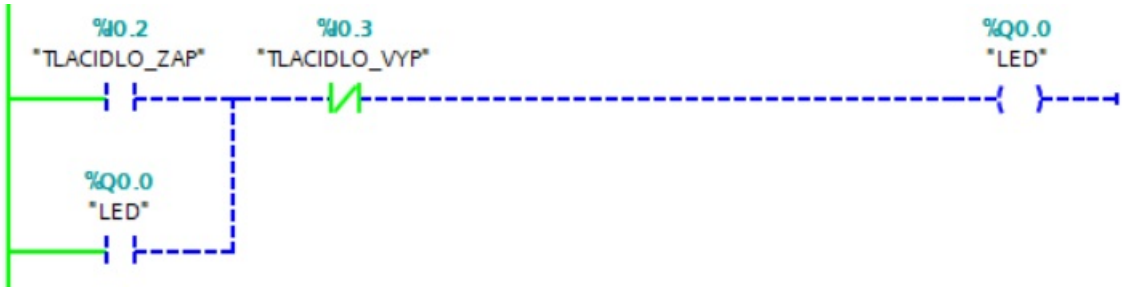
Na Obr. 4.57 sú tlačidlá nezatlačené a LED bolo vypnuté, LED nesvieti.

Zatlačenie tlačidla TLACIDLO_ZAP je na Obr. 4.58. V momente zatlačenia platí horná časť OR vetvy. Kedže platí aj výstup inštrukcie *normálne zatvorený kontakt*, lebo stav tlačidla TLACIDLO_VYP je FALSE, do LED sa zapíše TRUE. Od ďalšieho cyklu platia obe časti OR podmienky pokiaľ tlačidlo TLACIDLO_ZAP držíme.

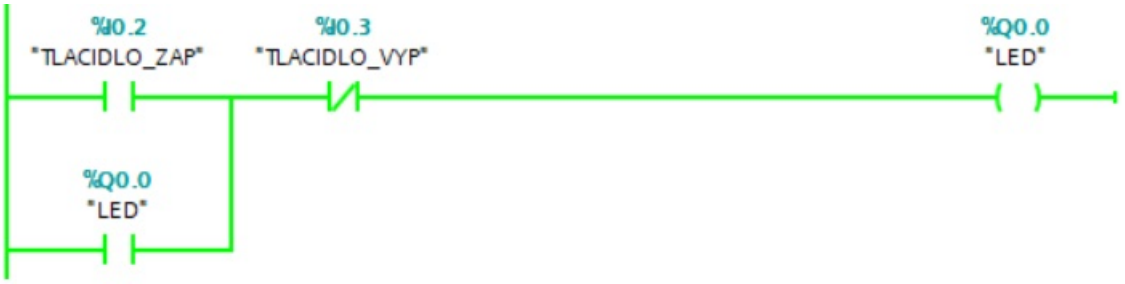
Uvoľnenie tlačidla TLACIDLO_ZAP je na Obr. 4.59. LED naďalej svieti.

Na Obr. 4.60 je zaznamenaná fáza vypnutia LED zatlačením tlačidla TLACIDLO_VYP. V momente zatlačenia má LED stav TRUE, výstupom OR vetvy je TRUE. Stav tlačidla TLACIDLO_VYP je TRUE, výstupom inštrukcie *normálne zatvorený kontakt* je FALSE. Podmienka AND neplatí, preto sa v danom cykle do LED zapíše FALSE. Od ďalšieho cyklu už neplatí spodná časť OR vetvy a ani druhá časť AND podmienky. LED ostáva zhasnutá.

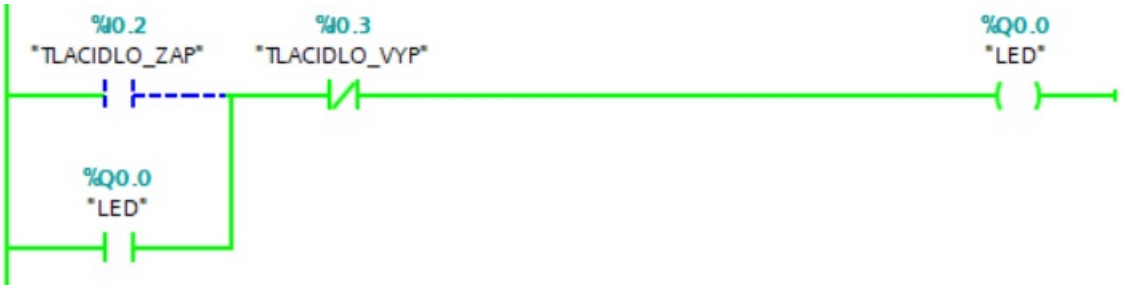
Na poslednom Obr. 4.61 sú zatlačené obe tlačidlá, LED nesvieti, lebo aj keď horná časť OR podmienky je platná, druhá časť AND podmienky neplatí.



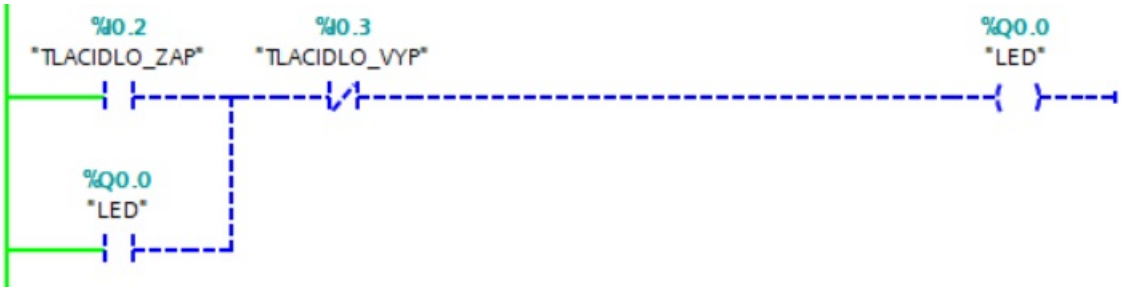
Obr. 4.57. 1. fáza monitorovania programu



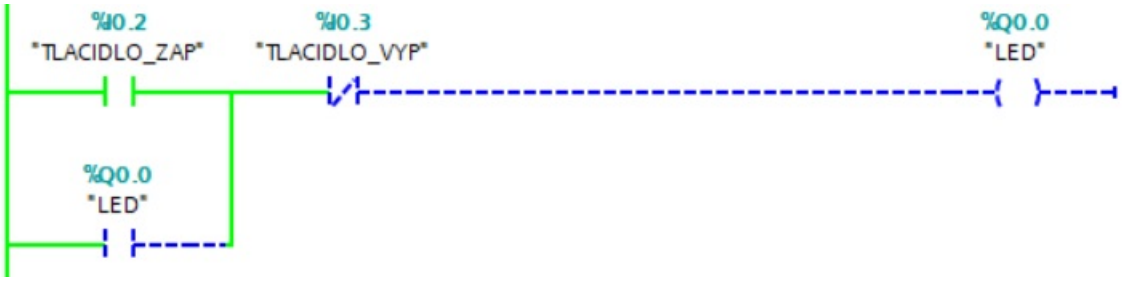
Obr. 4.58. 2. fáza monitorovania programu



Obr. 4.59. 3. fáza monitorovania programu



Obr. 4.60. 4. fáza monitorovania programu



Obr. 4.61. 5. fáza monitorovania programu

4.3.5 Príklad 5 - Ovládanie dopravníka 1

Majme dopravník na Obr. 4.62. Na začiatku dopravníka je snímač S1 a na konci snímač S2. Oba snímače sú pripojené ako spínacie kontakty k digitálnemu vstupnému modulu. Snímače detegujú prepravované objekty dopravníkom. Dopravník sa ovláda len smerom vpred (od S1 k S2) digitálnym výstupom D1.

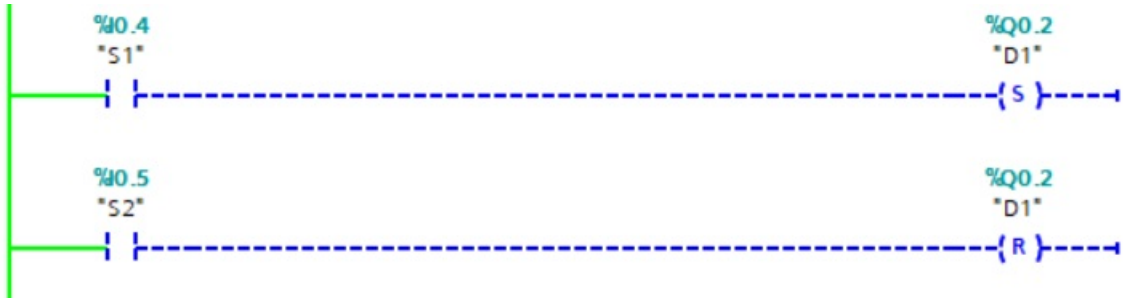
Obr. 4.62. Model dopravníka

Úloha

Naložením objektu na začiatku dopravníka bude objekt detegovaný snímačom S1. Objekt sa má prepraviť na koniec dopravníka, kde bude detegovaný snímačom S2. Po odobratí je umožnené opätovné nakladanie (a preprava objektu na koniec dopravníka). Uvažujeme len 1 ks objektu na dopravníku. Ide teda o ovládanie chodu dopravníka s cieľom prepraviť nakladané objekty na začiatku dopravníka na koniec dopravníka.

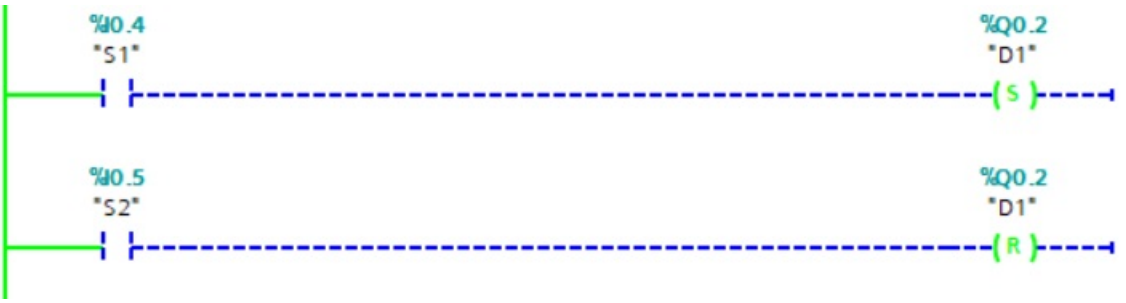
Riešenie je veľmi jednoduché. Aktivovaný snímač S1 vykoná inštrukciu *SET* na spustenie dopravníka a aktivovaný snímač S2 vykoná inštrukciu *RESET* na zastavenie dopravníka.

Na Obr. 4.63 je stav, keď snímače S1 a S2 nedetegujú objekty a dopravník nie je v pohybe. Detekciou objektu na začiatku dopravníka (Obr. 4.64) sa spúšťa chod dopravníka. Po krátkom čase transportu objektu snímač S1 prestáva detegovať objekt, ale dopravník je naďalej v pohybe (Obr. 4.65). Príchodom na koniec dopravníka je snímačom S2 detegovaný objekt čím sa dopravník vypína. Odobratie objektu by zodpovedalo stavom na Obr. 4.63.

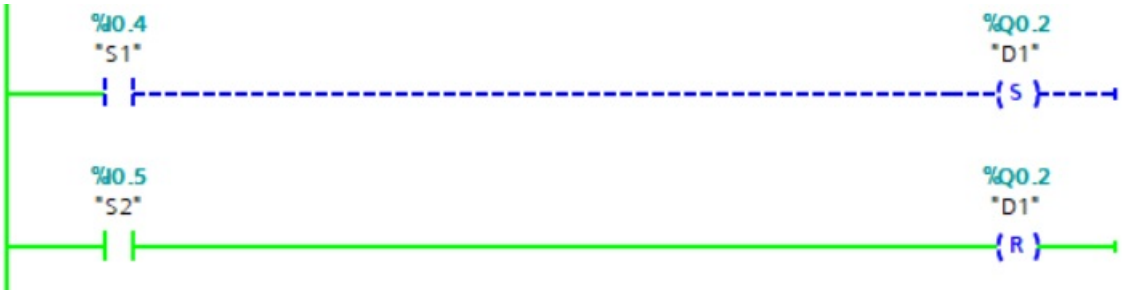


Obr. 4.63. 1. fáza monitorovania programu

Obr. 4.64. 2. fáza monitorovania programu



Obr. 4.65. 3. fáza monitorovania programu



Obr. 4.66. 4. fáza monitorovania programu

4.3.6 Príklad 6 - Ovládanie dopravníka 2

Majme dopravník z Príkladu 4.3.5 Príklad 5 - Ovládanie dopravníka 1. Modifikujme zadanie takto:

Úloha

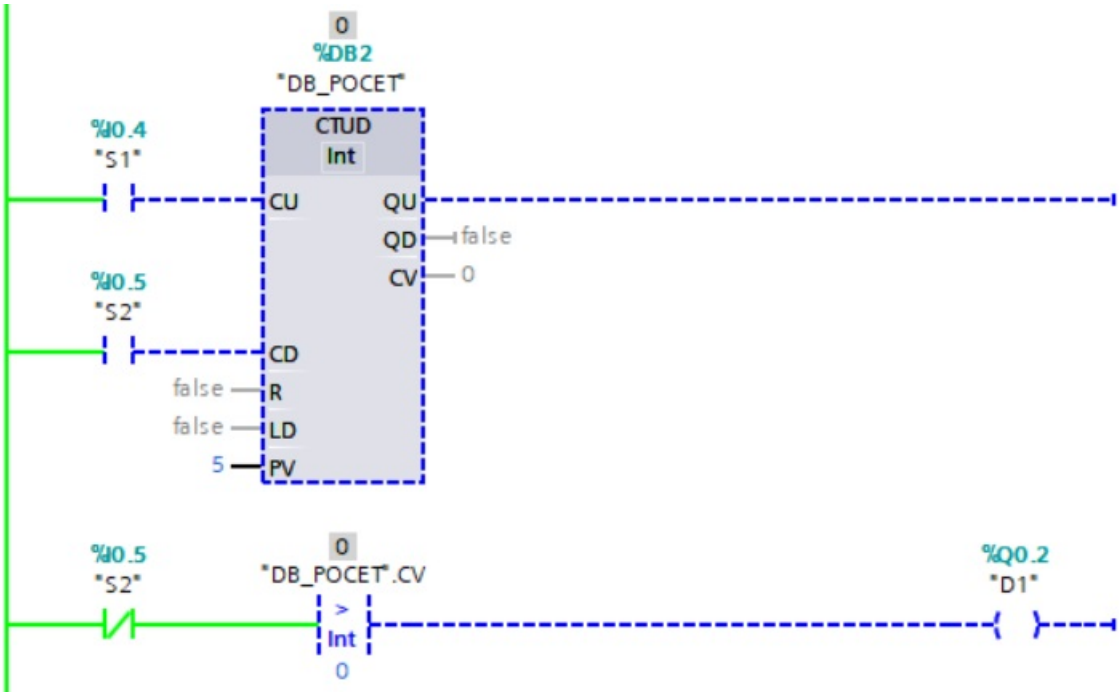
Objekty nakladané (a detegované snímačom S1) na začiatku dopravníka sa majú prepraviť na koniec dopravníka. Snímač S2 vypína dopravník. Na dopravníku môže byť ľubovoľný počet objektov.

Na implementáciu pre túto úlohu nestačia len snímače a výstup, ale je nutné vytvoriť stavy dopravníka. Mohla by nastať situácia, že S2 deteguje objekt a dopravník je zastavený. Druhý objekt je v strede dopravníka. Po obratí objektu na konci dopravníka musíme vedieť či existuje aspoň jeden objekt medzi snímačmi S1 a S2. Na to využijeme počítanie počtu objektov medzi S1 a S2.

1. riešenie

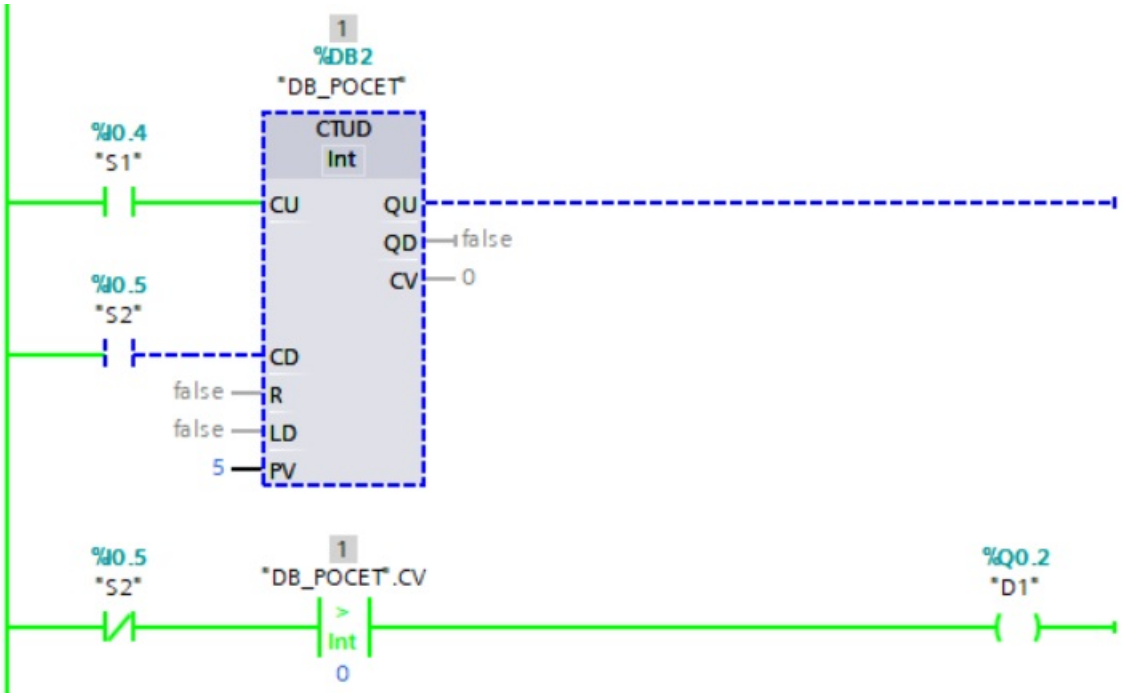
Riešenie je na Obr. 4.67. Na vstup CU počítadla sa pripojil signál zo snímača S1. Pri zmene signálu z FALSE na TRUE počítadlo navýši svoju hodnotu. Detekciou objektu snímačom S2 sa na vstup CD privedie zmena z FALSE na TRUE čím sa hodnota počítadla dekrementuje. Dolná časť programu ovláda samotný chod dopravníka. Dopravník je v chode ak snímač S2 nie je obsadený a na dopravníku je nenulový počet objektov. Postupnosť monitorovania je na Obr. 4.67 až 4.74.

Na Obr. 4.67 snímače S1 a S2 nedetegujú objekty, počítadlo má hodnotu 0 a výstup dopravníka nie je aktívny. V našom príklade ide o počiatočný stav.



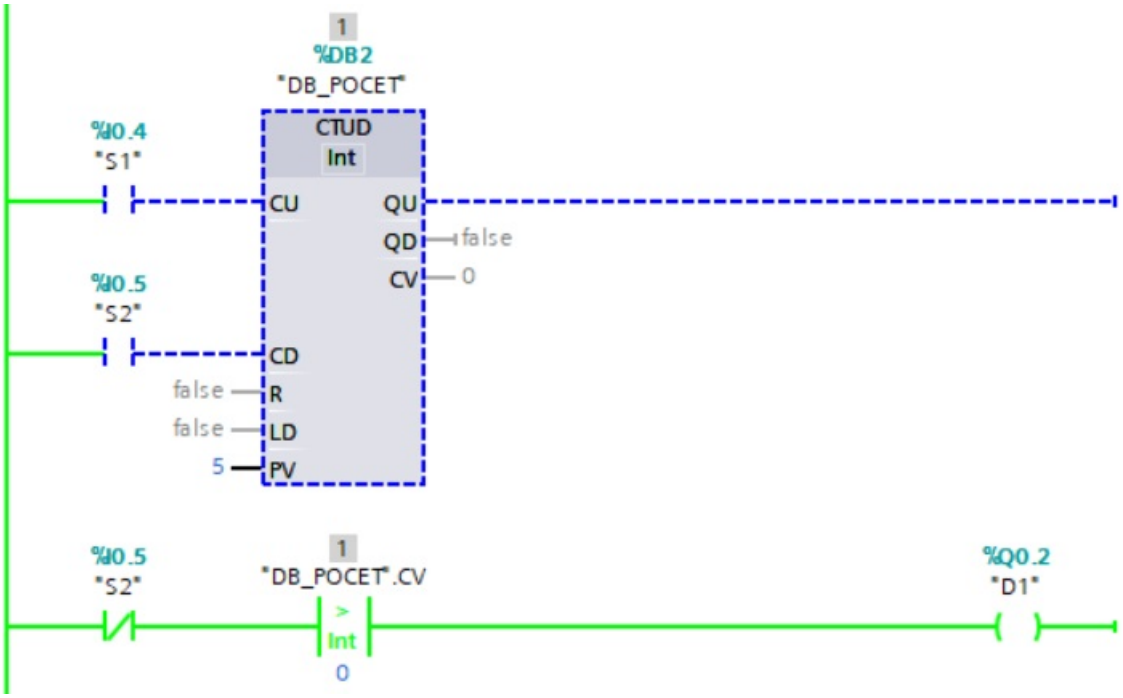
Obr. 4.67. 1. fáza monitorovania programu

Na Obr. 4.68 je snímačom S1 detegovaný objekt. Počítadlo navýšilo svoju hodnotu. Dopravník je v chode, lebo je platná podmienka AND.



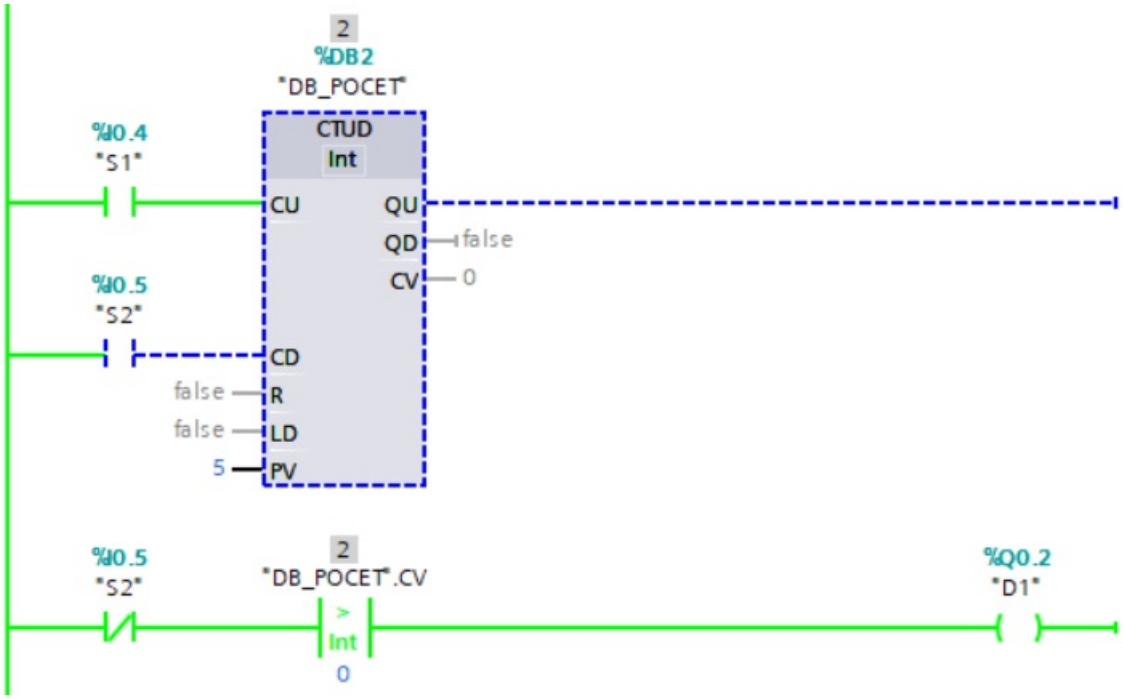
Obr. 4.68. 2. fáza monitorovania programu

Na Obr. 4.69 uplynul krátky čas transportu objektu a snímačom už nie je detegovaný objekt. Dopravník je stále v chode.



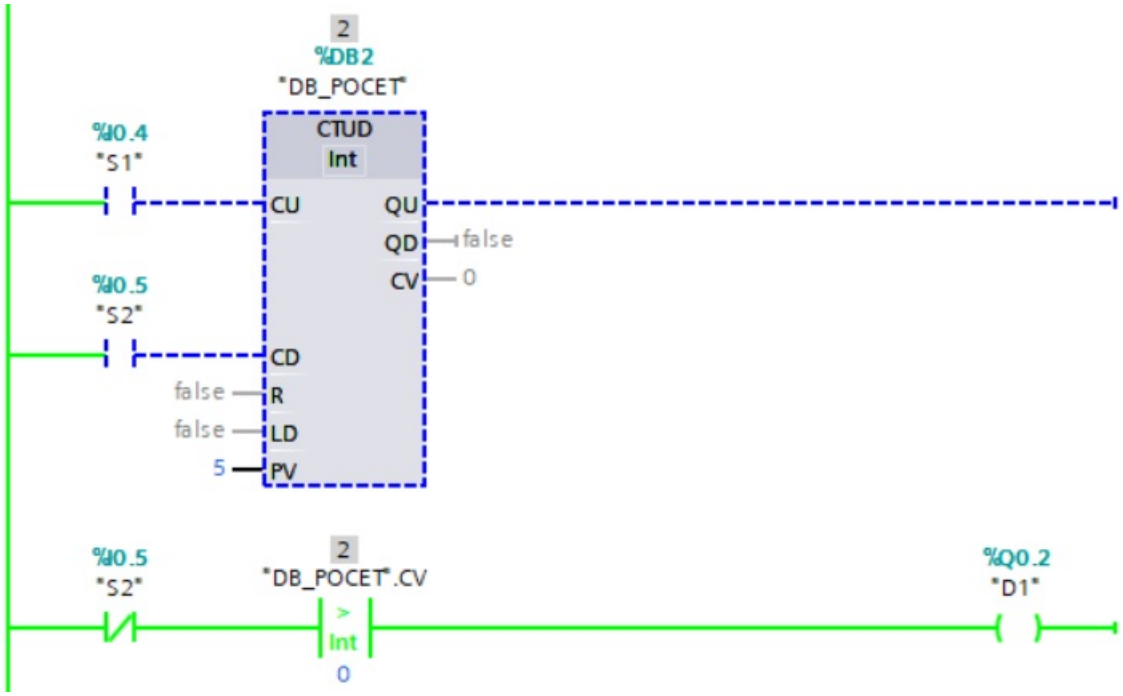
Obr. 4.69. 3. fáza monitorovania programu

Na Obr. 4.70 bol naložený nový objekt, ktorý je detegovaný snímačom S1. Hodnota počítadla sa navýšila na 2. Dopravník je stále v chode, lebo prvý objekt zatiaľ nedorazil na koniec dopravníka.



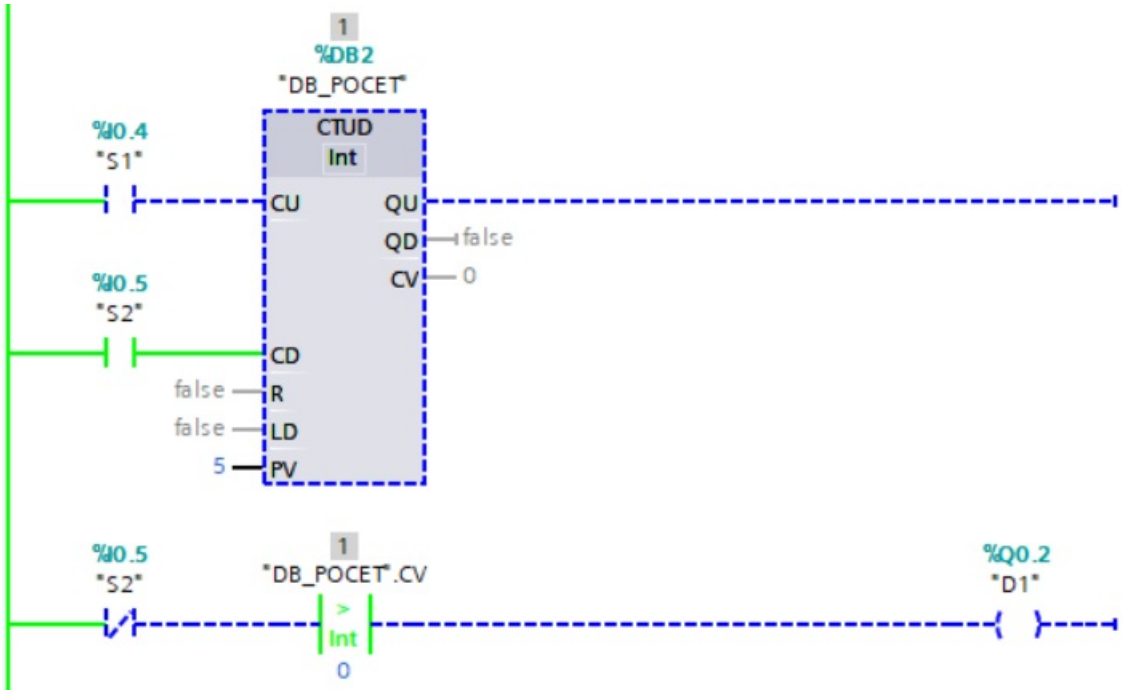
Obr. 4.70. 4. fáza monitorovania programu

Transport dvoch objektov dopravníkom keď druhý objekt nie je detegovaný snímačom S1 je na Obr. 4.71.



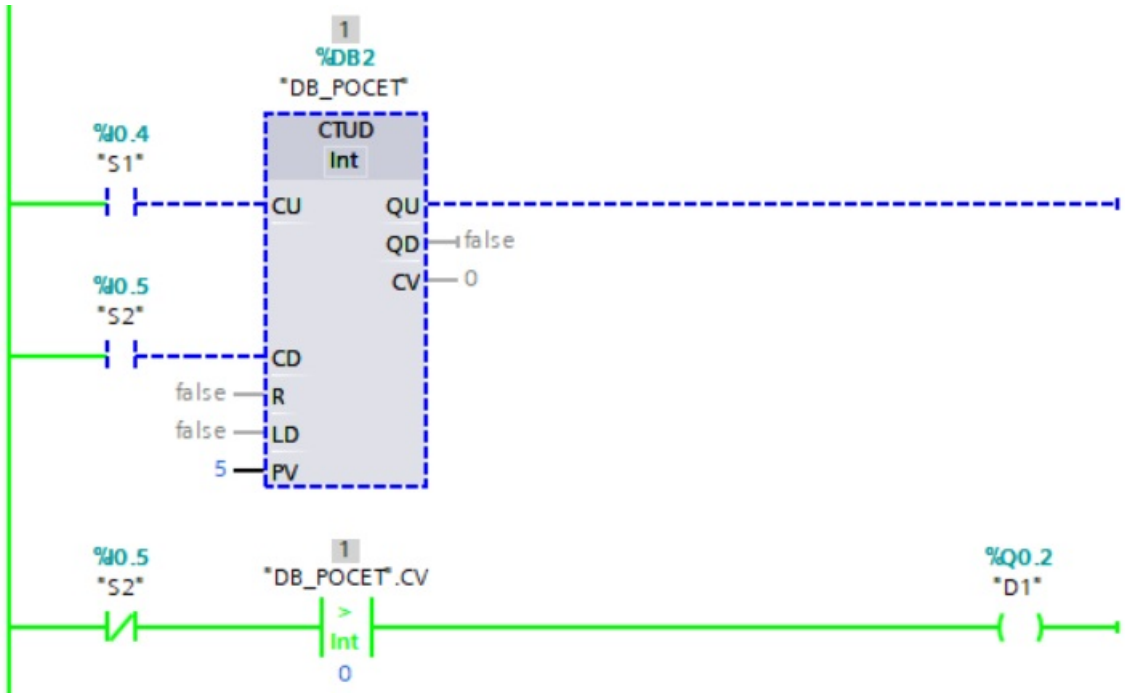
Obr. 4.71. 5. fáza monitorovania programu

Detekcia prvého objektu snímačom S2 na konci dopravníka je na Obr. 4.72. Snímač S2 privedený na vstup CD počítadla dekrementoval počet objetov z 2 na 1. Dopravník zastal, lebo prestala platiť prvá časť podmienky AND.



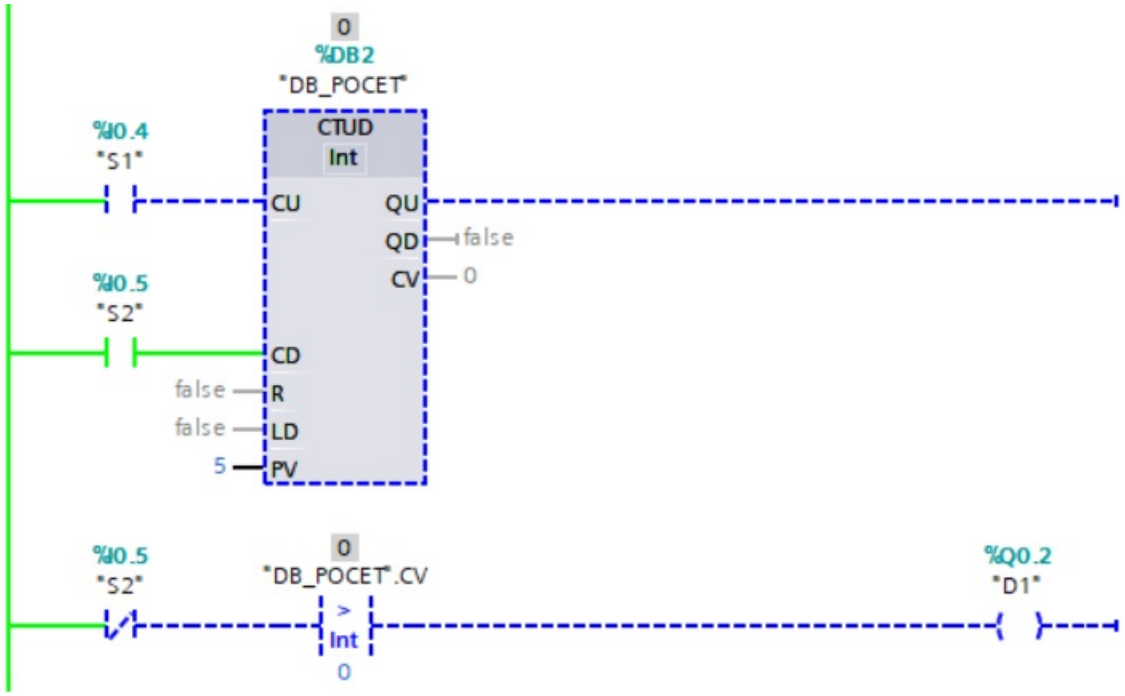
Obr. 4.72. 6. fáza monitorovania programu

Odobratie objektu a opätovné spustenie dopravníka je na Obr. 4.73.



Obr. 4.73. 7. fáza monitorovania programu

Príchod posledného objektu na koniec dopravníka je na Obr. 4.74. Počet objektov sa zmenil na 0.



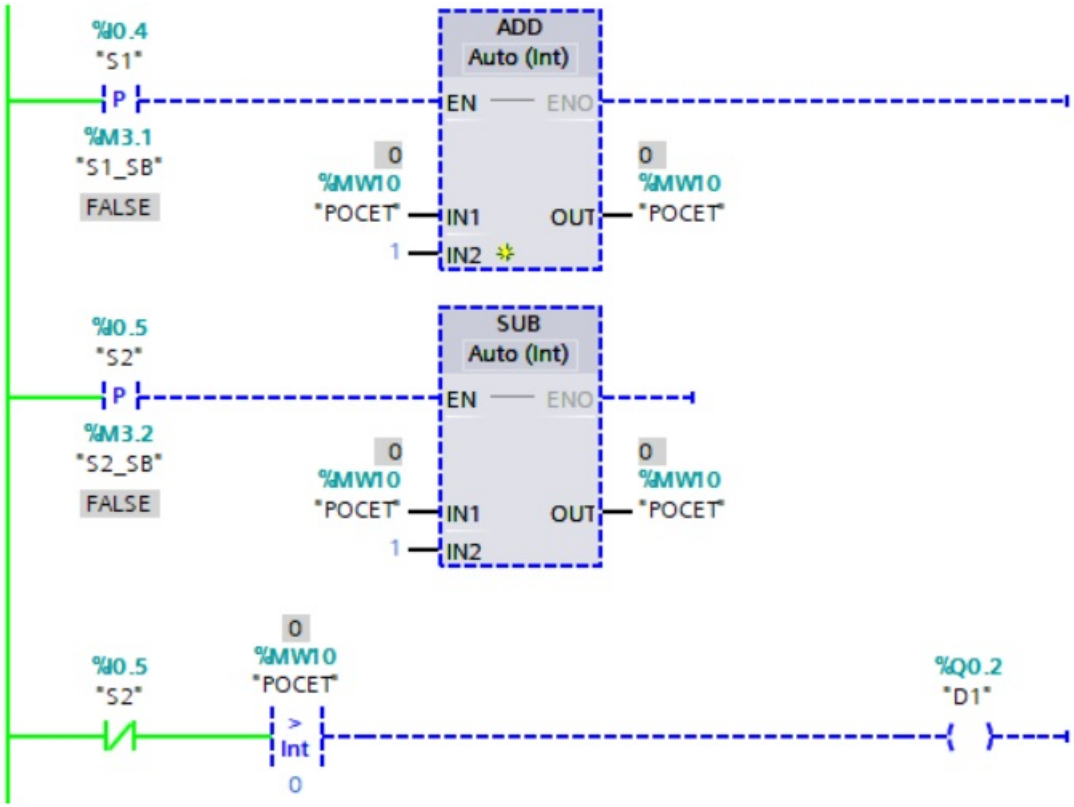
Obr. 4.74. 8. fáza monitorovania programu

Odobratím posledného objektu sa dopravník nespustí, lebo bude platiť stav na Obr. 4.67, t. j. druhá časť AND podmienky (počet objektov je väčší ako 0) nie je platný.

V praxi je potrebné myslieť na filtráciu digitálnych vstupov. Snímač S1 by mohol byť zašumený, alebo v dôsledku rôznorodého materiálu objektu by S1 zmenil svoj stav, čím by nesprávne detegoval počet kusov. To by viedlo k stálemu chodu dopravníka, lebo snímačom S2 by sa hodnota počítadla nemohla dekrementovať na 0. Podobná situácia by nastala aj na konci dopravníka. Počas odoberania by S2 mohol viackrát zmeniť svoj stav, a tým napr. pri poslednom objekte napočítať zápornú hodnotu. Naložením nového objektu na začiatku dopravníka by bola hodnota nulová alebo záporná a dopravník by sa nespustil.

2. riešenie

V druhom riešení (Obr. 4.75) namiesto inštrukcie *CTUD* využijeme základné matematické inštrukcie *ADD* a *SUB*. Aby program fungoval správne, musíme k vstupom EN inštrukcií *ADD* a *SUB* pripojiť inštrukcie detekcie nábežných hrán. V opačnom prípade by sa v každom cykle, kým je detegovaný objekt snímačom navyšovala alebo znižovala hodnota premennej POCET. V inštrukciách *nábežná hrana operandu* boli použité pomocné bity S1_SB a S2_SB ako 2. operand. Ide o pamäťové bity (%M). Tieto bity sa nikde nesmú zapisovať (používať). Ak je snímačom S1 detegovaný objekt, inštrukcia *ADD* sa vykoná len raz. Dôjde k navýšenie počtu objektov výpočtom POCET = POCET + 1. Ak je objekt detegovaný snímačom S2, inštrukcia *SUB* sa vykoná len raz. Dôjde k zníženiu počtu objektov výpočtom POCET = POCET - 1. Ak posledný snímač nie je obsadený a na dopravníku je nenulový počet objektov, dopravník je v chode, inak je zastavený. Ďalším riešením by mohla byť náhrada inštrukcií *ADD* a *SUB* inštrukciami *INC* a *DEC*.



Obr. 4.75. Alternatívne riešenie

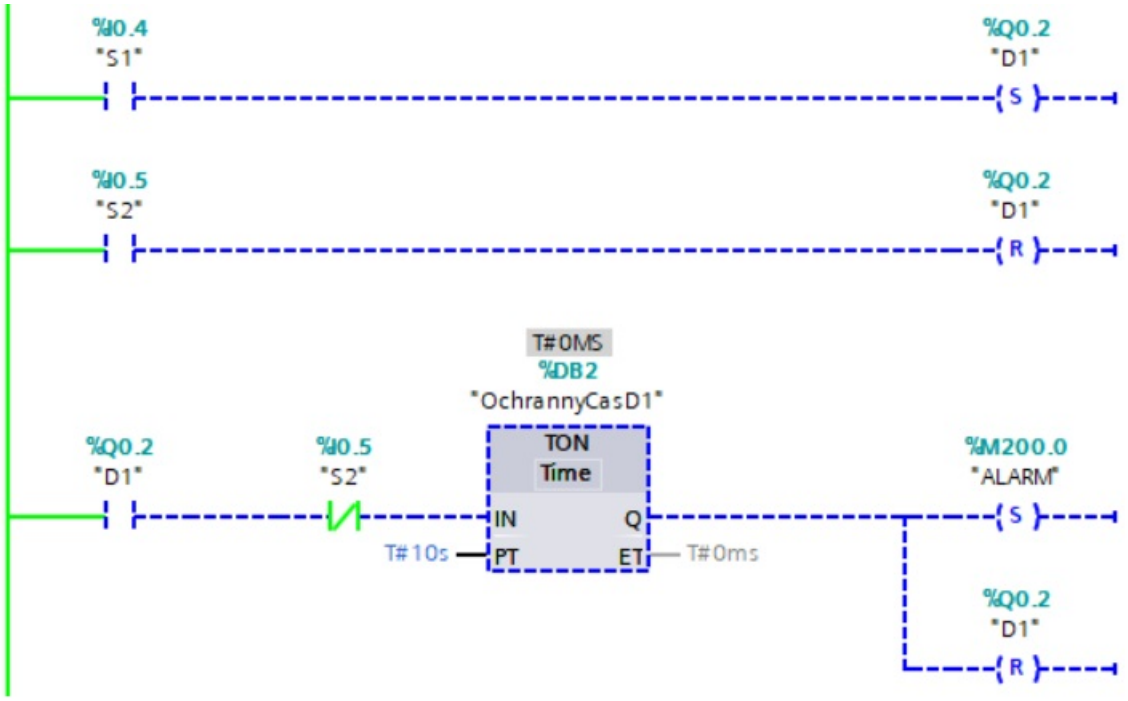
4.3.7 Príklad 7 - Alarm dopravníka

Majme dopravník z príkladu 4.3.5 Príklad 5 - Ovládanie dopravníka 1. V praxi sa často základné ovládanie programu dopĺňa o množstvo ďalších stavov. Sú to napr. varovania, alarmy a pod. Ak uvažujeme riešenie z Príkladu 4.3.5 Príklad 5 - Ovládanie dopravníka 1 mohla by nastať situácia, že ak objekt spadne alebo sa zasekne na dopravníku, tak by bol dopravník stále v chode. Modifikujme pôvodné zadanie takto:

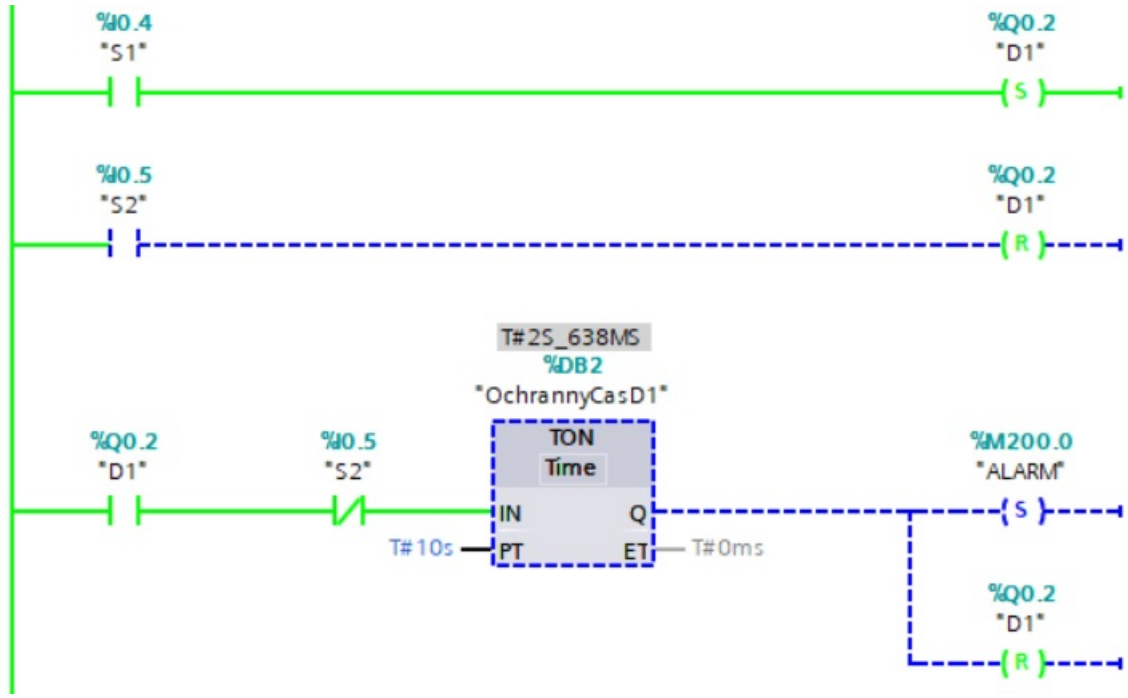
Úloha

Naložením objektu na začiatku dopravníka bude objekt detegovaný snímačom S1. Objekt sa má prepraviť na koniec dopravníka, kde bude detegovaný snímačom S2. Po odobratí je umožnené opätovné nakladanie. Uvažujeme len 1 ks objektu na dopravníku. Ak transportovaný objekt nebude detegovaný snímačom S2 po uplynutí ochranného času 10 s, dopravník zastane a vznikne alarm.

Riešenie je na Obr. 4.76. Prvé dve vetvy sú totožné s Príkladom 4.3.5 Príklad 5 - Ovládanie dopravníka 1. Ak je dopravník v chode a objekt nedorazil na snímač S2, je platná podmienka AND pripojená na vstup IN časovača (Obr. 4.77).

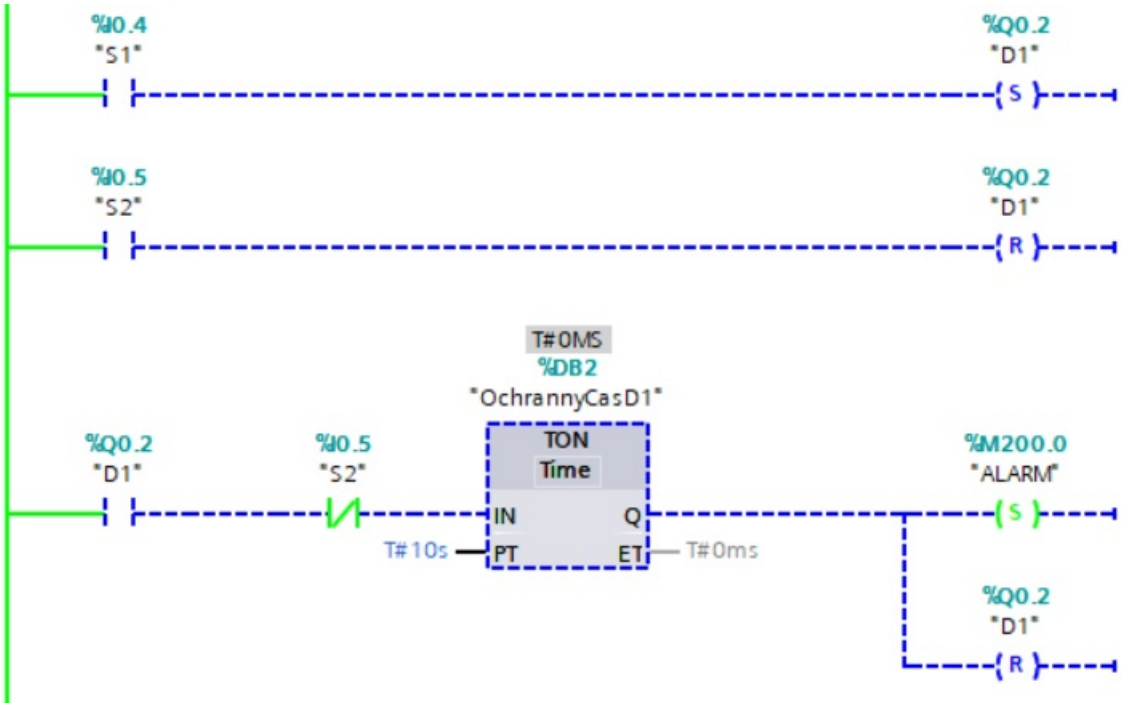


Obr. 4.76. 1. fáza monitorovania programu

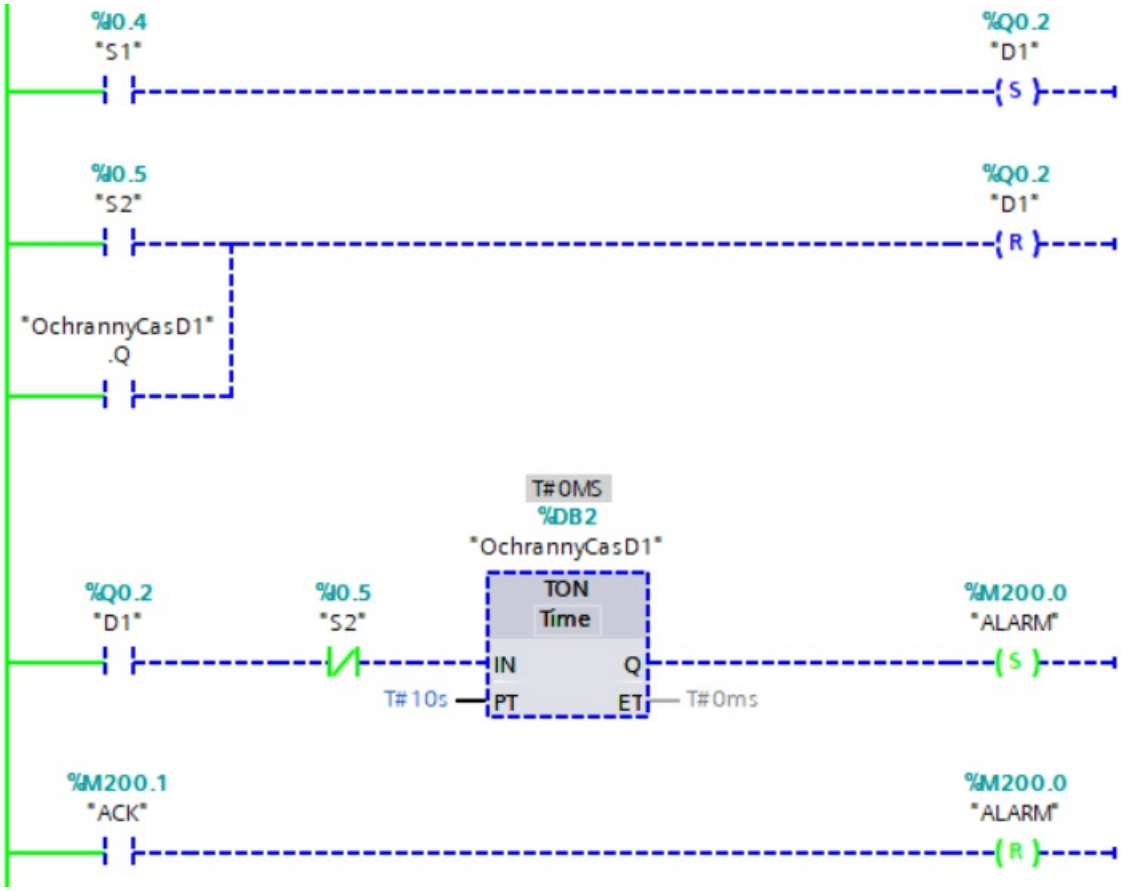


Obr. 4.77. 2. fáza monitorovania programu

Ak objekt nebude po uplynutí času 10 s detegovaný snímačom S2, zopne sa výstup Q časovača, ktorý vykoná priradenie TRUE do premennej ALARM a FALSE do D1, čím sa dopravník vypne. Tento stav platí len jeden cyklus, lebo v ďalšom cykle dopravník nie je v chode, t. j. neplatí prvá časť AND podmienky (Obr. 4.78).

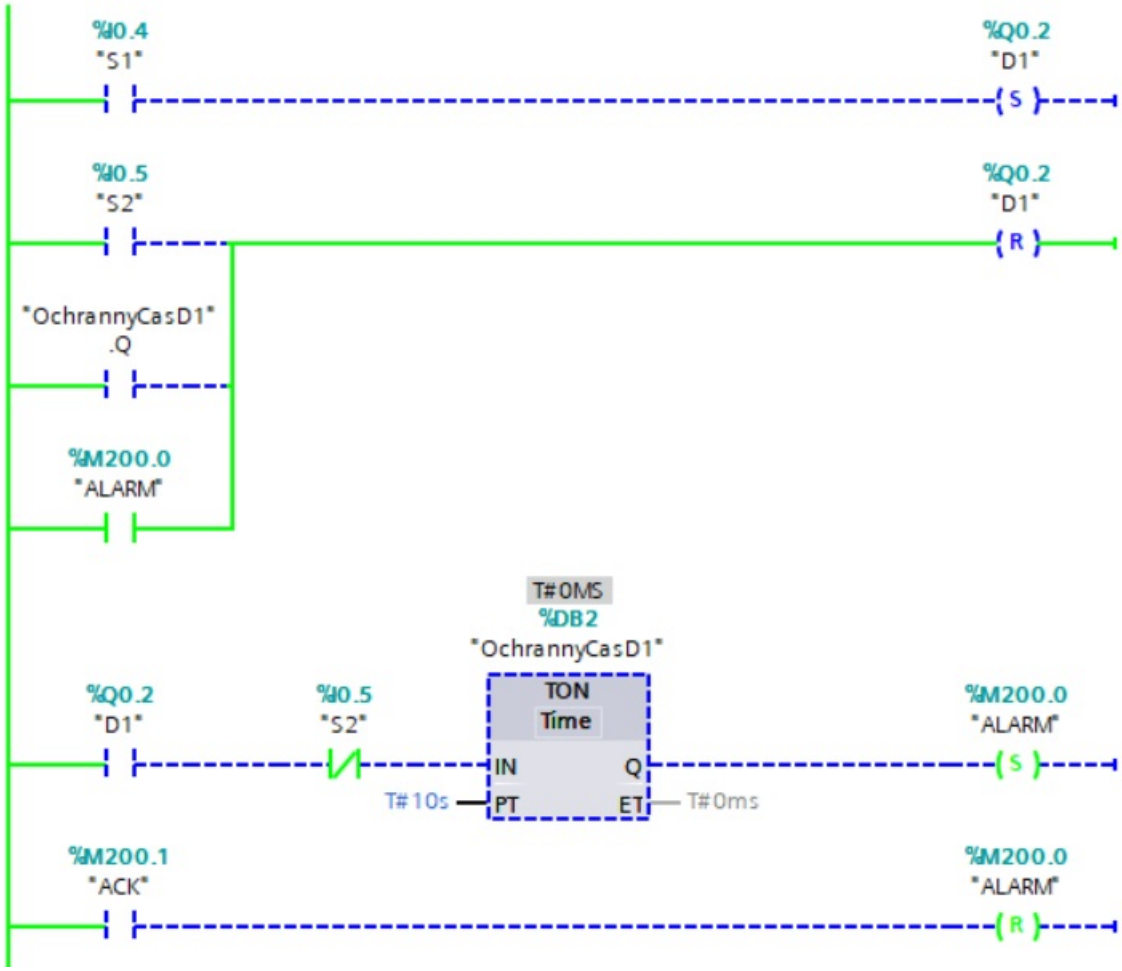


Obr. 4.78. 3. fáza monitorovania programu



Obr. 4.79. Modifikácia programu

Modifikovaný program s dvoma úpravami je na Obr. 4.79. Výstup časovača bol pridaný v druhej vetve do OR podmienky vynulovania hodnoty D1. S tým súvisí aj zrušenie inštrukcie *RESET* za časovačom. Druhou modifikáciou je rozšírenie programu o vynulovanie alarmu v poslednej vetve. Použili sme bit ACK, ktorý by mohol byť zapisovaný z vizualizácie pomocou grafického objektu tlačidla. Alarm v tejto verzii programu len indikuje alarm a nevplýva na chod dopravníka. Opätovným naložením objektu na začiatok dopravníka by sa dopravník spustil. Zapracovaná blokácia chodu dopravníka od alarmu je na Obr. 4.80. V druhej vetve bola doplnená podmienka OR. Ako vidno, v každom cykle existencie alarmu sa vykonáva inštrukcia *RESET*, preto dopravník je blokovaný.



Obr. 4.80. Modifikácia programu

4.3.8 Príklad 8 - Signalizácia alarmu

Majme dopravník z Príkladu 4.3.5 Príklad 5 - Ovládanie dopravníka 1 s riadením z príkladu 4.3.7 Príklad 7 - Alarm dopravníka. Alarm, ktorý vznikol, je vnútorným pamäťovým bitom, ktorý by mohol byť signalizovaný vo vizualizácii. Ak by sme chceli zobraziť alarm napr. pomocou signalizačného svetla, sirénou a pod. stavom alarmu musíme ovládať výstup.

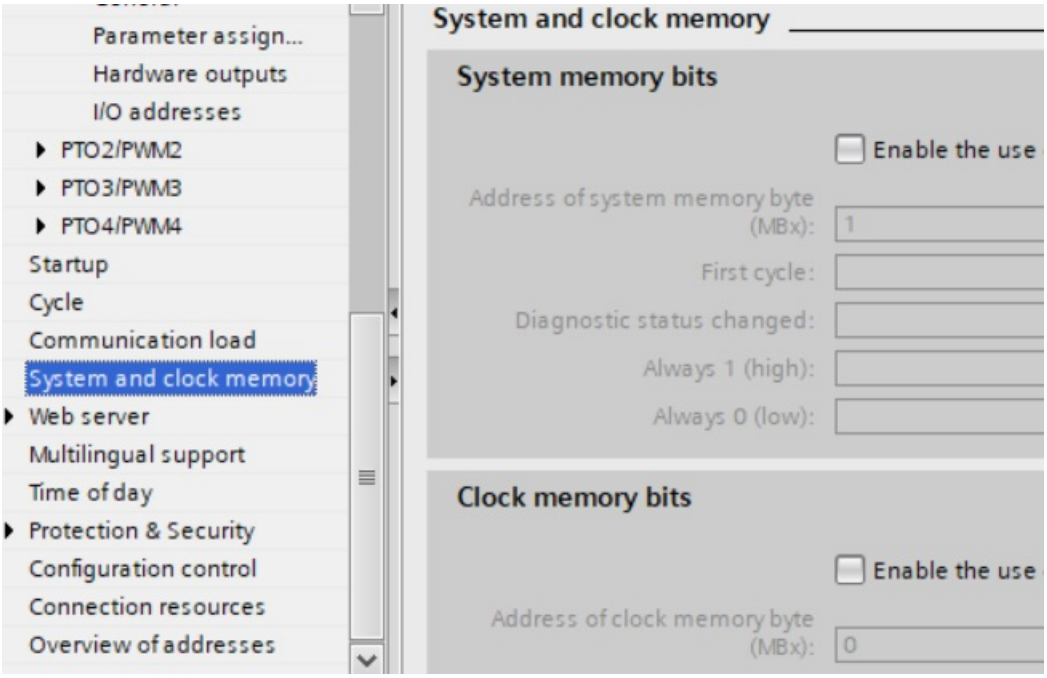
Úloha

Signalizujte vzniknutý alarm z Príkladu 4.3.7 Príklad 7 - Alarm dopravníka na výstupe LED. LED má blikať s periódou 0,5 s (t. j. 0,5 s bude TRUE, 0,5 s bude FALSE).

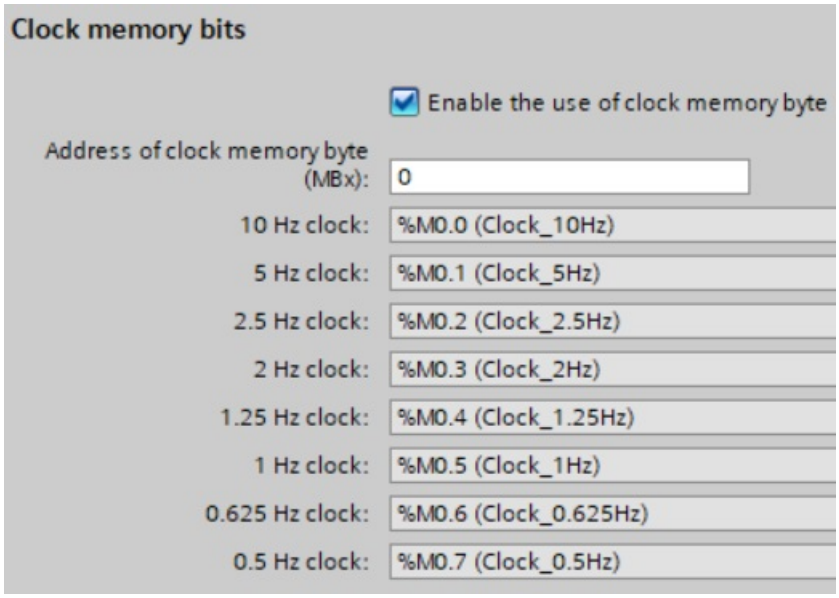
1. riešenie

Vytvoríme procesorom generované bity. V hardvérovej konfigurácii CPU nájdeme položku System and clock memory (Obr. 4.82). V časti Clock memory bits zapneme mapovanie systémových bitov do pamäťovej oblasti (Pozn. je potrebné naškrtnúť voľbu "Enable the use of clock memory byte"). Ponecháme prednastavené Bajt číslo 0 (Obr. 4.83). Praktické využitie majú aj tzv. System memory bits. Napríklad premenné, ktoré majú stále hodnotu TRUE (AlwaysTRUE) alebo FALSE (AlwaysFALSE). Ponecháme prednastavené Bajt číslo 0 (Obr. 4.84). Uvedené zmeny nastavení sú dostupné len v offline režime. Po zmene je nutné nahrať do CPU aj hardvérovú konfiguráciu. V opačnom prípade vytvorené premenné nebudú zapisované procesorom. Z vytvorených premenných využijeme Clock_1Hz, ktorý mení svoj stav každých 0.5s. Ak je alarm aktívny, tak ako mení svoju hodnotu premenná Clock_1Hz bude blikať výstup LED (Obr. 4.81).

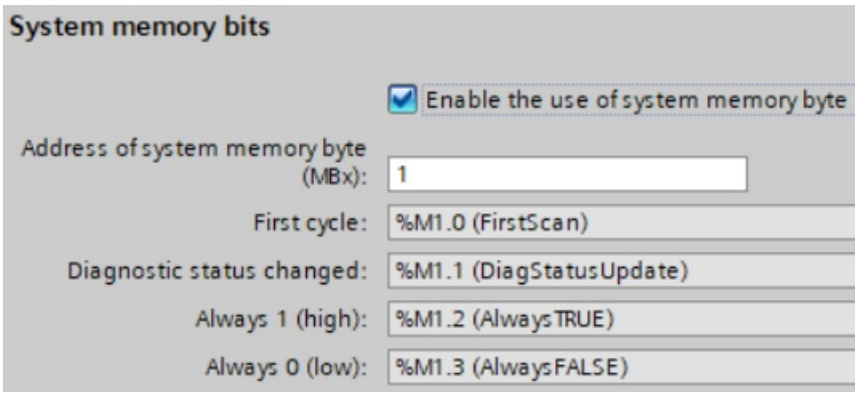
Obr. 4.81. Riešenie úlohy



Obr. 4.82. Nastavenia System and clock memory



Obr. 4.83. Nastavenia Clock memory bits

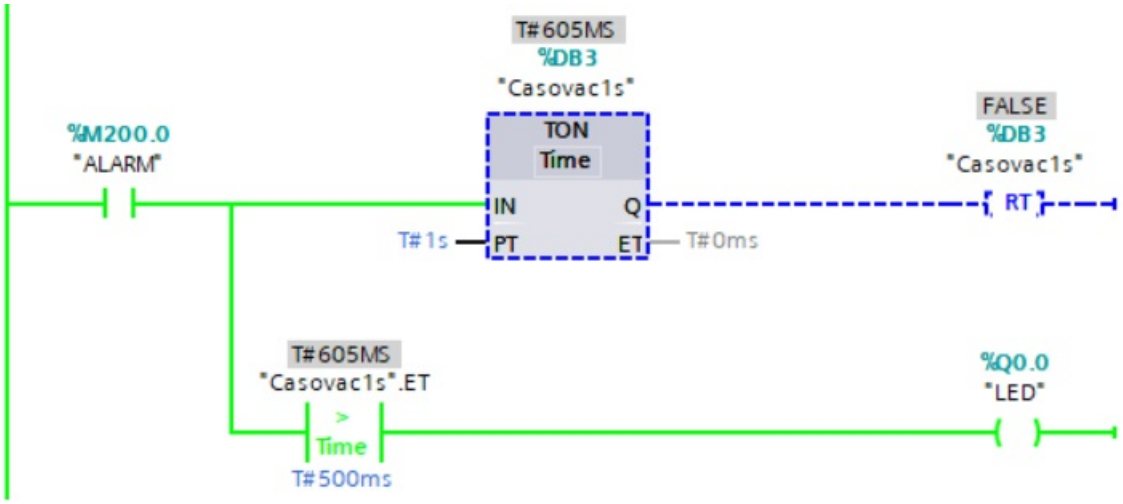


Obr. 4.84. Nastavenia System memory bits

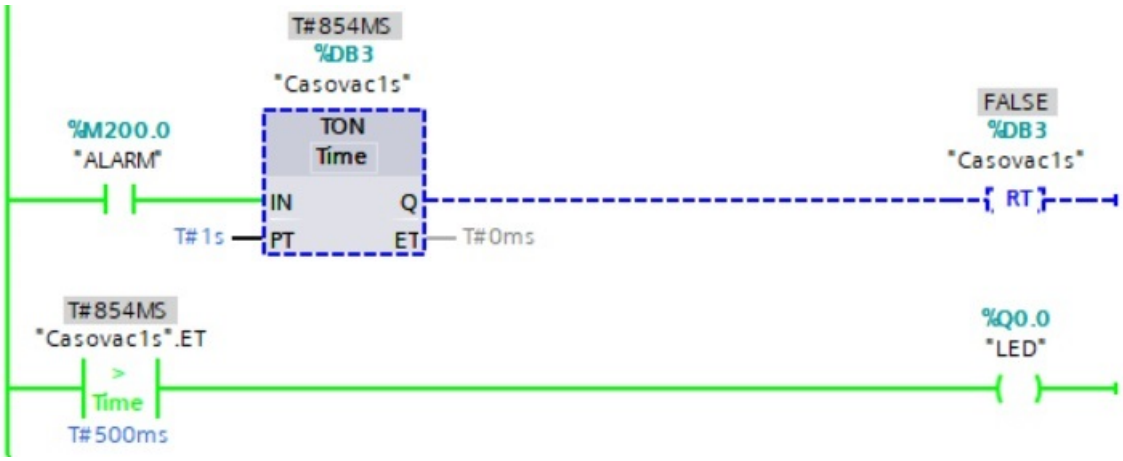
2. riešenie

Namiesto systémového bitu použijeme časovač. Časovač potrebuje podmienku na vstupe, t. j. v TIA Portal nie je možné priviesť signál TRUE z rebríka bez podmienkovej inštrukcie. V príklade sa hodí stav ALARM. Ak je ALARM v stave TRUE, spustí sa časovač. Po 1 s sa zopne výstup Q, ktorý spôsobí vynulovanie časovača. Ak je alarm a uplynutý čas je menší alebo rovný ako 500 ms, LED je FALSE. Ak je alarm a uplynutý čas je viac ako 500 ms, LED je TRUE. (Obr. 4.85).

Alternatívou je program na Obr. 4.86. Ak je alarm, časovač navyšuje hodnotu ET a po 500 ms sa LED zapne. Po resete časovača je ET menej ako 500 ms, LED zhasne. V prípade, že by sa stav alarmu vynuloval, vynuluje sa aj časovač, t. j. ET = 0 ms. Porovnávací blok v druhej vetve bude mať na výstupe FALSE. LED zhasne, t. j. prestane blikať.



Obr. 4.85. Riešenie úlohy



Obr. 4.86. Modifikácia riešenia úlohy

4.3.9 Príklad 9 - Ovládanie dopravníkov 1

V praxi jeden riadiaci systém monitoruje a ovláda viacero zariadení ako sú dopravníky, piesty, robotické ramená atď. V predchádzajúcich príkladoch sme uviedli ovládanie dopravníka priamo od vstupov. Majme tri dopravníky D1, D2 a D3. Nech sa dopravníky ovládajú výstupmi D1_VPRED, D2_VPRED a D3_VPRED. Tieto výstupy ovládajú chod dopravníka vpred. Majme na každom dopravníku optický snímač indikujúci prítomnosť prepravovaných objektov D1_OS, D2_OS a D3_OS. Všetky vstupy sú zapojené ako spínacie kontakty.
Poznámka: Presná poloha snímačov na dopravníkoch (či sú umiestnené na začiatku, v strede alebo na konci) nehrá v tomto príklade zásadnú úlohu. Podstatné je, že každý snímač slúži na detekciu objektu, ktorý vstupuje na daný dopravník z predchádzajúcej pozície. Ich úlohou je vyvolať zmenu obsadenosti, ktorá potom zostáva aktívna aj po opustení oblasti detekcie snímača pomocou pomocných premenných.

Úloha

Vytvorte nové stavy (premenné) dopravníkov, tzv. obsadenosti, ktoré budú indikovať obsadenosť aj keď objekt nebude detegovaný optickým snímačom. Pomocou nových stavov vytvorte algoritmus prepravy objektov z D1 na D3 tak, aby sa objekty presúvali medzi dopravníkmi len ak je aktuálny dopravník obsadený a nasledujúci dopravník je voľný. Na poslednej pozícii (dopravníku D3) manuálnym odobratím objektu vynulujte obsadenosť pozície.

Na Obr. 4.87 je zoznam vytvorených premenných. Okrem vstupov, výstupov a premenných indikujúcich obsadenosti dopravníkov (D1_OBS až D3_OBS) boli pripravené aj pomocné premenné (SB1 až SB3) na detekciu hrán snímačov.

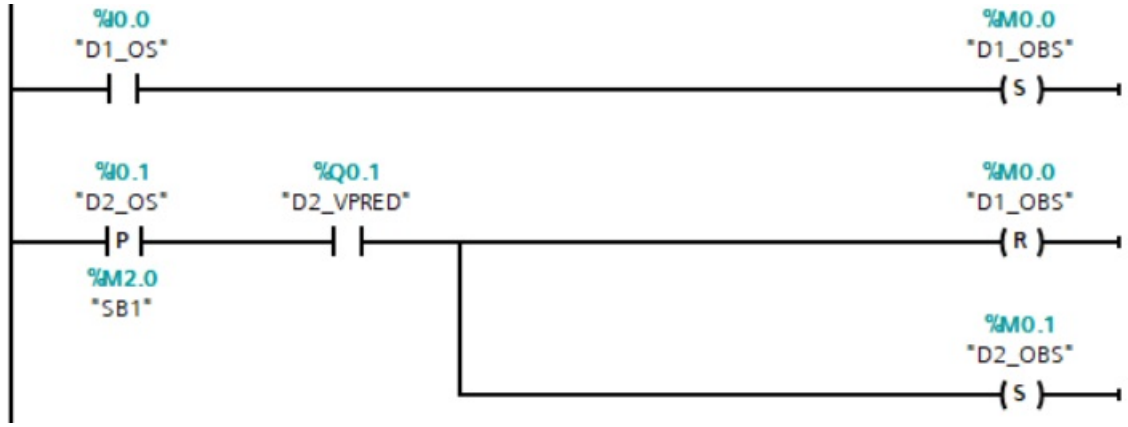
Default tag table

	Name	Data type	Address
1	D1_VPRED	Bool	%Q0.0
2	D2_VPRED	Bool	%Q0.1
3	D3_VPRED	Bool	%Q0.2
4	D1_OS	Bool	%I0.0
5	D2_OS	Bool	%I0.1
6	D3_OS	Bool	%I0.2
7	D1_OBS	Bool	%M0.0
8	D2_OBS	Bool	%M0.1
9	D3_OBS	Bool	%M0.2
10	SB1	Bool	%M2.0
11	SB2	Bool	%M2.1
12	SB3	Bool	%M2.2

Obr. 4.87. Zoznam premenných

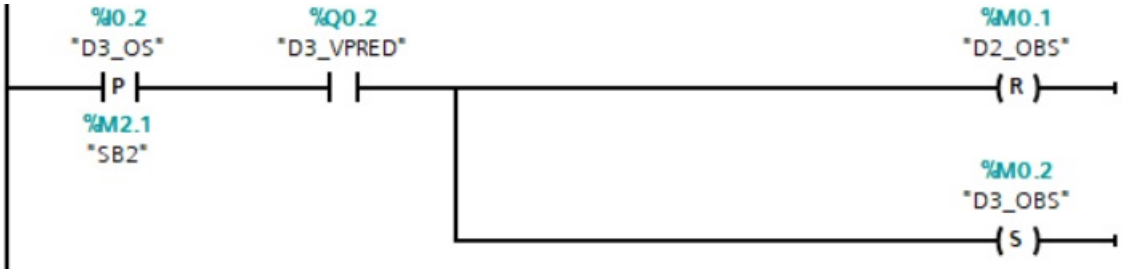
Na Obr. 4.88 sa aktiváciou optického snímača D1_OS nastaví D1_OBS na stav TRUE. Ak bude objekt v pohybe z D1 na D2 a už nebude detegovaný snímačom D1_OS, bude táto premenná niesť informáciu o obsadenosti pozície. Ak by dopravník zastal z nejakého dôvodu, na základe uvedeného pomocného stavu by sa preprava objektu mohla obnoviť. Ak by sa program spoliehal len na snímače, objekt by nebol detekovateľný a bolo by ho potrebné manuálne presunúť k snímaču.

V spodnej časti sa nábežnou hranou deteguje príchod objektu na dopravník D2. Podmienka je rozšírená o informáciu o chode dopravníka D2. Ak by podmienka absentovala a snímač by sa zatienil rukou, pozícia na D2 by sa obsadila aj pri prázdnych a vypnutých dopravníkoch. V praxi väčšinou máme spätnú väzbu o chode dopravníka. Nakoľko ju v príklade nemáme, doplnili sme aspoň výstup. Príchodom objektu na D2 sa nastaví obsadenosť D2_OBS a vynuluje obsadenosť predchádzajúcej pozície (D1_OBS). Bez nábežnej hrany a AND podmienky by sa mohlo stať, že D1 a D2 majú objekty, ktoré sú detegované snímačmi, ale snímač na D2 v každom cykle nuluje pozíciu D1.



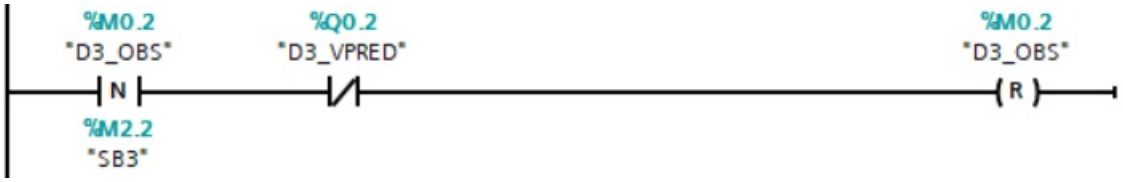
Obr. 4.88. Stavy pri prechode objektu z D1 na D2

Analogicky je prechod objektu z D2 na D3 (Obr. 4.89). Nábežná hrana D3_OS indikuje príchod objektu na D3. Ak bol dopravník D3 v pohybe, vynuluje sa pozícia D2 a aktivuje sa obsadenosť D3.



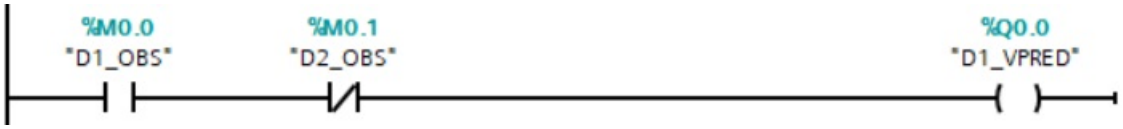
Obr. 4.89. Stavy pri prechode objektu z D2 na D3

Manuálnym odobratím objektu z D3_OS sa vynuluje obsadenosť D3. Odobratie objektu znamená detegovanie dobežnej hrany D3_OS (prvá časť podmienky AND) pri vypnutom dopravníku D3 (druhá časť podmienky AND). Rebríková schéma je Obr. 4.90.



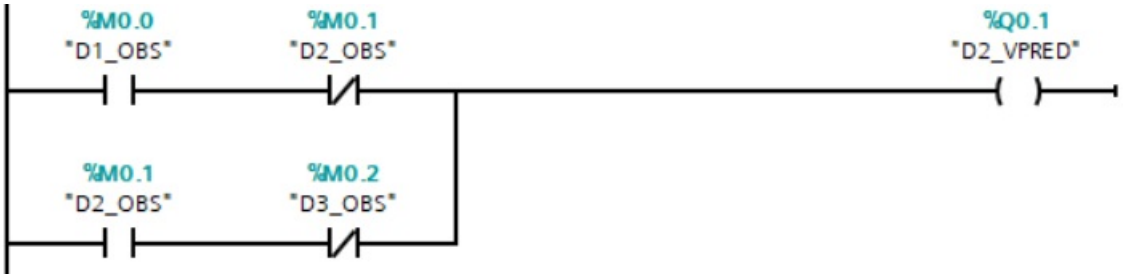
Obr. 4.90. Odobratie objektu z D3

Na záver pridáme riadenie dopravníkov na základe vytvorených stavov. Riadenie môžeme rozdeliť na odoslanie (na nasledovnú pozíciu) a prijatie (z predchádzajúcej pozície). Na Obr. 4.91 dopravník D1 nemá predchádzajúcu pozíciu, bude len odosielať v prípade, že D2 nie je obsadená a D1 je obsadená.



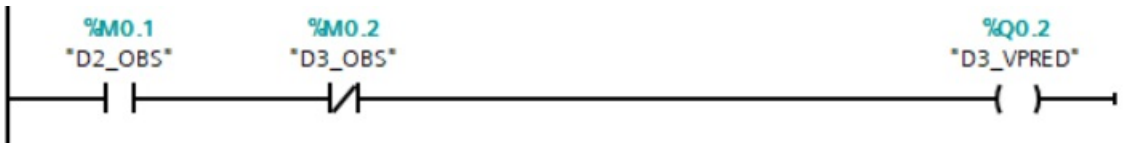
Obr. 4.91. Riadenie D1

Dopravník D2 prijíma objekt z D1, ale aj odosiela na D3 (4.92). V hornej vetve je realizácia chodu (spustenia) D2 počas prijatia objektu z D1 a v spodnej vetve spustenie D2 pri odoslaní objektu na D3.



Obr. 4.92. Riadenie D2

Posledný dopravník D3 objekt len prijíma z D2 (4.93).



Obr. 4.93. Riadenie D3

4.3.10 Príklad 10 - Ovládanie dopravníkov 2

Majme dva dopravníky D1 a D2. Nech sa dopravníky ovládajú výstupmi D1_VPRED a D2_VPRED (Obr. 4.94). Tieto výstupy ovládajú chod dopravníkov vpred. Majme na každom dopravníku dva optické snímače indikujúce prítomnosť prepravovaných objektov. Na začiatku D1 je umiestnený snímač D1_OS1 a na konci dopravníka snímač D1_OS2. Podobne na začiatku D2 je umiestnený snímač D2_OS1 a na konci dopravníka snímač D2_OS2 (Obr. 4.94). Všetky vstupy sú zapojené ako rozpínacie kontakty.

Úloha

Vytvorte nové stavy (premenné) dopravníkov, tzv. obsadenosti, ktoré budú indikovať obsadenosť aj keď objekt nebude detegovaný optickým snímačom. Pomocou nových stavov vytvorte algoritmus prepravy objektov zo začiatku D1 na koniec D2 (pozn. samozrejme ak je D2 voľný). Objekt naložený na D1 a detegovaný snímačom D1_OS1 spustí prepravu objektu na koniec dopravníka D1, ak je pozícia voľná. Objekt prepravený na koniec dopravníka D1 a detegovaný D1_OS2 je prepravený na D2. V momente detekcie objektu snímačom D2_OS1 sa vypína dopravník D1 a objekt putuje ďalej na koniec dopravníka D2, kde objekt zastane a bude sa čakať na jeho odobratie. Na poslednej pozícii (dopravníka D2) manuálnym odobratím objektu vynulujte obsadenosť pozície. Uvažujme vždy len jeden objekt na jednom dopravníku.

Na Obr. 4.94 je zoznam vstupov a výstupov. Vytvorme v prvom kroku premenné reprezentujúce obsadenosti pozícií na dopravníku. Pre každý snímač jednu premennú v globálnom dátovom bloku DATA (Obr. 4.95). V uvedenom riešení sa budeme vyhýbať detekcii hrán.

Default tag table

		Name	Data type	Address
1		D1_OS1	Bool	%I0.0
2		D1_OS2	Bool	%I0.1
3		D2_OS1	Bool	%I0.2
4		D2_OS2	Bool	%I0.3
5		D1_VPRED	Bool	%Q0.0
6		D2_VPRED	Bool	%Q0.1

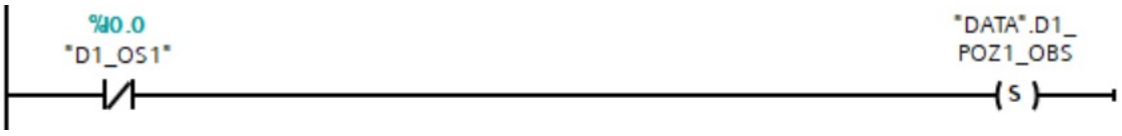
Obr. 4.94. Zoznam vstupov a výstupov

DATA

		Name	Data type	Start value
1		Static		
2		D1_POZ1_OBS	Bool	false
3		D1_POZ2_OBS	Bool	false
4		D2_POZ1_OBS	Bool	false
5		D2_POZ2_OBS	Bool	false

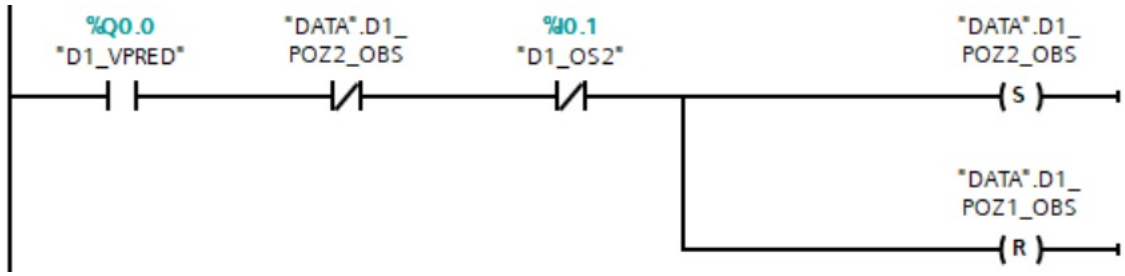
Obr. 4.95. Zoznam vytvorených premenných

Naložením objektu pred prvý snímač na D1 sa nastaví obsadenosť prvej pozície DATA. D1_POZ1_OBS na TRUE (Obr. 4.96).



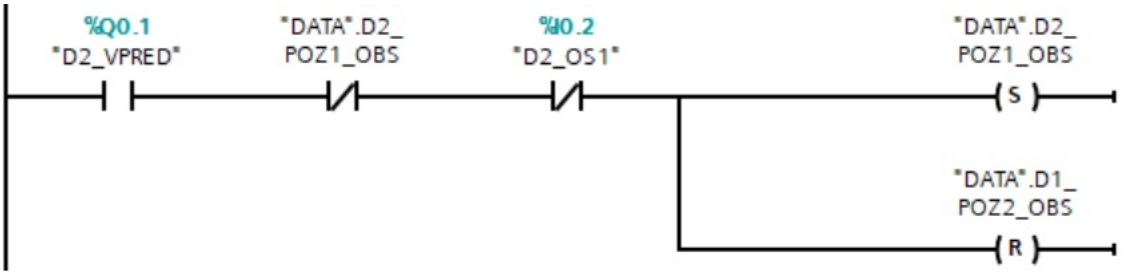
Obr. 4.96. Obsadenosť prvej pozície na D1

Počas chodu dopravníka D1 (D1_VPRED je TRUE), pri voľnej druhej pozícii (DATA .D1_POZ2_OBS je FALSE) a detekcii objektu optickým snímačom D1_OS2 sa nastaví druhá pozícia D1 na TRUE a predchádzajúca pozícia na FALSE (Obr. 4.97). V prípade prázdneho modelu naložením objektu priamo pred druhý snímač sa neobsadí pozícia lebo dopravník nebol v chode. Ide o ošetrenie, ktorým sa dajú predísť hazardné stavy.



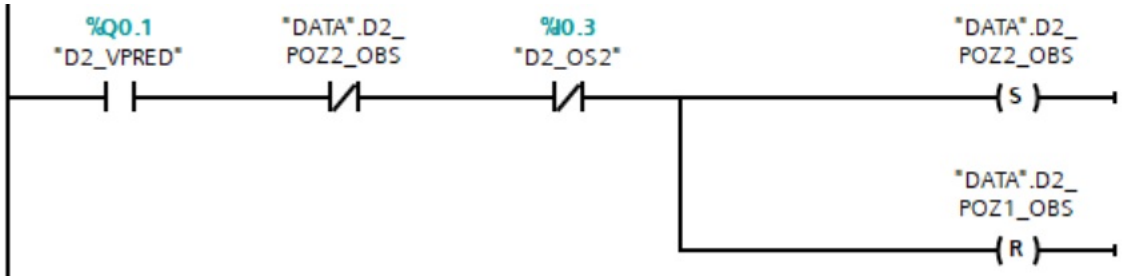
Obr. 4.97. Obsadenosť druhej pozície na D1

Na Obr. 4.98 je analogické riešenie výmeny stavov medzi druhou pozíciou D1 a prvou pozíciou D2.



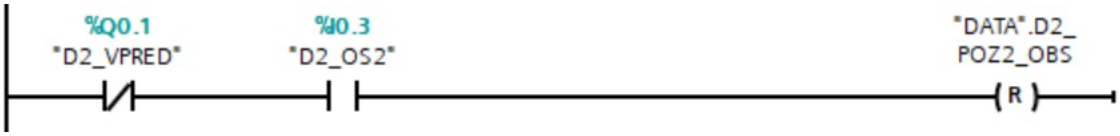
Obr. 4.98. Obsadenosť prvej pozície na D2

Rovnako postupujeme pri prechode objektu z prvej pozície D2 na druhú pozíciu (Obr. 4.99).



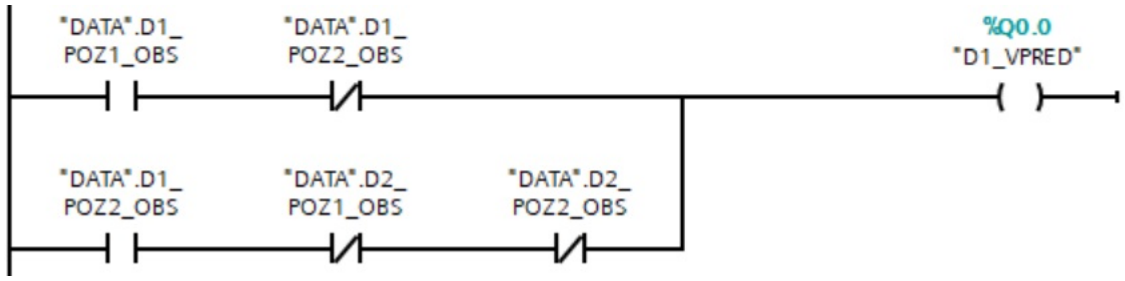
Obr. 4.99. Obsadenosť druhej pozície na D2

Na záver na Obr. 4.100 detegujeme odobratie pri vypnutom dopravníku D2. Ak platí podmienka (dopravník nie je v pohybe a druhý snímač nedeteguje objekt), potom sa vynuluje pozícia.



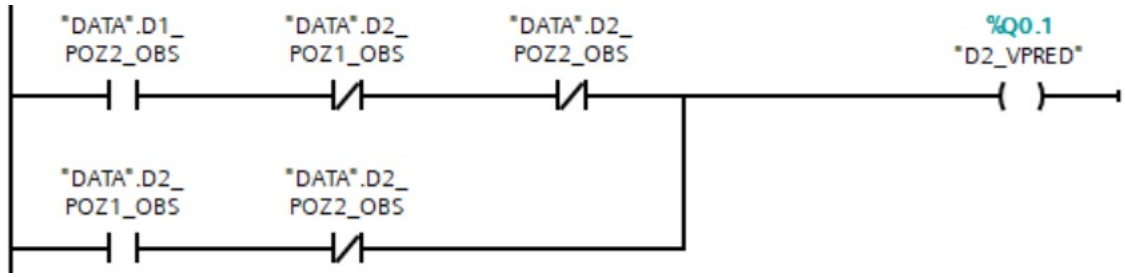
Obr. 4.100. Vynulovanie druhej pozície na D2

Ostáva navrhnúť riadenie dopravníkov pomocou vytvorených stavov. Dopravník D1 má byť v chode počas prepravy objektu z prvej pozície na druhú pozíciu a počas prepravy objektu z druhej pozície na druhý dopravník (na prvú pozíciu). Logicky by sme nemali objekt prepravovať z D1 na D2, ak je druhá pozícia D2 obsadená, lebo by objekt spadol a nespĺňal by požiadavku čakania na manuálne odobratie. Riešenie chodu dopravníka D1 je na Obr. 4.101.

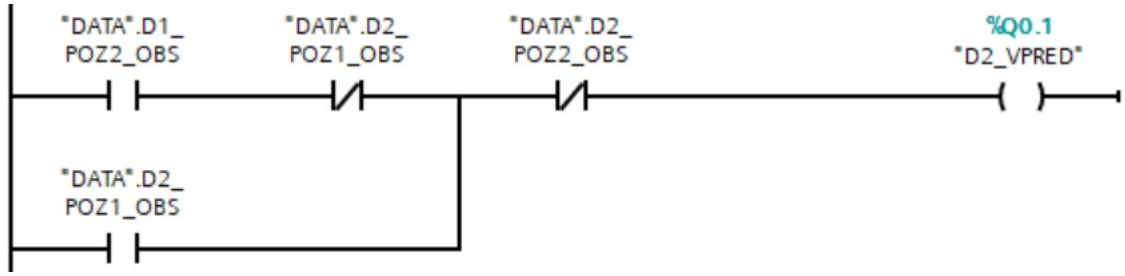


Obr. 4.101. Ovládanie výstupu D1

Riešenie chodu dopravníka D2 je na Obr. 4.102. Horná časť OR vetvy reprezentuje odoslanie objektu z obsadenej druhej pozície D1 na prvú pozíciu D2 (druhá musí byť tiež voľná). Dolná vetva OR podmienky realizuje chod dopravníka počas prepravy z prvej pozície D2 na druhú pozíciu. Alternatívne riešenie, ktoré zjednodušuje kód je na Obr. 4.103.



Obr. 4.102. Ovládanie výstupu D2



Obr. 4.103. Ovládanie výstupu D2

4.3.11 Príklad 11 - Takt 1s

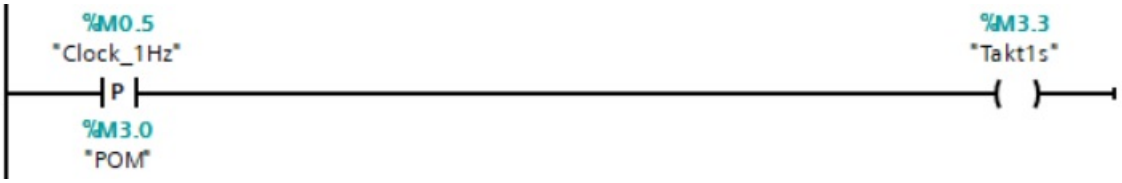
V predchádzajúcom Príklade 4.3.8 Príklad 8 - Signalizácia alarmu sme uviedli dva postupy ako vytvoriť signál meniaci svoju logickú hodnotu so stanovenou frekvenciou. V praxi sa často využíva nábežná hrana takýchto signálov. V stanovenú frekvenciu premenná nadobudne hodnotu TRUE, inak je FALSE. Tento signál možno použiť na inkrementáciu premennej.

Úloha

Vytvorte premennú, ktorá nadobudne hodnotu TRUE každú sekundu, vo zvyšnom čase má hodnotu FALSE.

1. riešenie

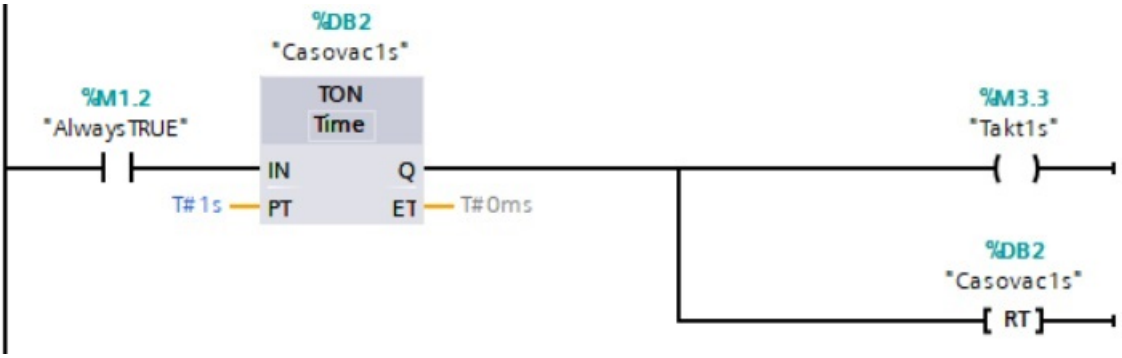
Na Obr. 4.104 je prvé (najjednoduchšie) riešenie úlohy. Stav nábežnej hrany zapisujeme do pamäťového bitu, ktorý každý sekundu nadobudne hodnotu TRUE. Vo zvyšnom čase má hodnotu FALSE.



Obr. 4.104. Prvé riešenie úlohy

2. riešenie

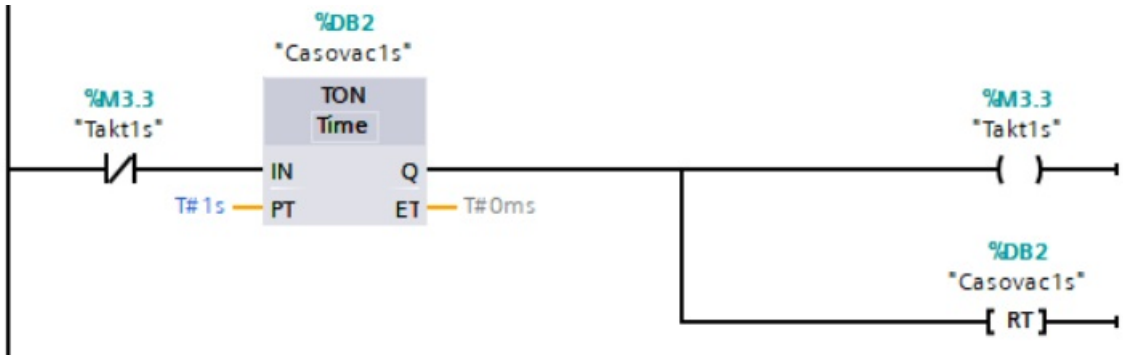
V druhom riešení je alternatívou časovač (Obr. 4.105). Pre spustenie časovača je privedený signál z premennej AlwaysTRUE, ktorá nadobúda stále hodnotu TRUE. Pokiaľ časovač nedosiahol prednastavený čas, jeho výstup Q je FALSE, preto premenná Takt1s má tiež stav FALSE. Po dosiahnutí prednastaveného času sa do premennej Takt1s zapíše hodnota TRUE a vynuluje sa časovač. V tomto cykle má premenná Takt1s stav TRUE. V ďalšom cykle sa časovač opätovne spúšťa, lebo na vstup IN prichádza hodnota TRUE a v predchádzajúcim cykle bol vynulovaný. Výstup časovača v tomto cykle je FALSE, preto sa hodnota Takt1s zmení na FALSE.



Obr. 4.105. Druhé riešenie úlohy

3. riešenie (nie úplne správne riešenie)

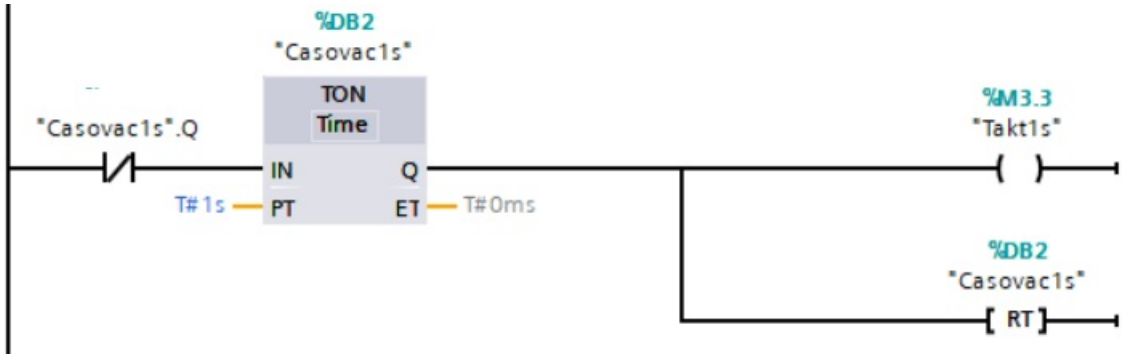
Na Obr. 4.106 sa oproti 2. riešeniu zmenil vstupný signál spúšťajúci časovač. Po dosiahnutie času PT sa program chová identicky. Uvažujme, že v aktuálnom cykle časovač dosiahol prednastavený čas a stav Takt1s je TRUE. V ďalšom cykle sa negovaný signál premennej Takt1s privedie na vstup, ktorý bude FALSE. Časovač sa opäť vynuluje, čím sa do premennej Takt1s zapíše stav FALSE z časovača. Až v ďalšom cykle sa časovač reálne spustí. Ak je čas cyklu zanedbateľný a presnosť časovania nie je kritická, aj toto riešenie môže postačovať. V prípade nutnosti presného časovania a dlhšieho cyklu, napr. 100 ms je rozdiel medzi očakávaným taktom a reálnym až 10 %.



Obr. 4.106. Tretie (nie úplne správne) riešenie úlohy

4. riešenie (zlé riešenie)

Na Obr. 4.107 je zlé riešenie zadania. V teoretickej časti sme v časti venovanej časovačom spomenuli, že ak sa odkazujeme na Q a ET časovača, CPU automaticky aktualizuje premenné v inštančnom dátovom bloku. Práve preto, ak je na začiatku Q v stave FALSE, časovač sa spúšťa. Časovač navyšuje uplynutý čas. Nech je cyklus, v ktorom časovač dosiahne hodnotu PT. Prvá inštrukcia programu zisťuje stav Q. Predtým sa aktualizujú premenné v inštančnom DB. ET nadobudne hodnotu PT a Q hodnotu TRUE. Inštrukcia preto načíta stav TRUE ešte pred evaluáciou samotnej inštrukcie *TON*. Privedený stav FALSE na vstup IN časovača resetne časovač. Výstup Q sa nastaví na FALSE. Kedže je Q = FALSE, na výstupe časovača sa do Takt1s nikdy nepriradí hodnota TRUE a nikdy sa nevykoná inštrukcia *RT*.



Obr. 4.107. Štvrté (zlé) riešenie úlohy

4.3.12 Príklad 12 - Ovládanie piestu

Majme objekt pohybujúci sa na dopravníku, ktorý detegujeme optickým snímačom. Tieto objekty sa často prekladajú z dopravníka robotom na iné miesto (do iného zariadenia, na dopravník, tvorí sa stoh na výstupnom mieste a pod.). Pred odobratím z dopravníka musí byť objekt zarovnaný, aby nedošlo ku kolízii robota s objektom. Majme jeden výstup ovládajúci chod piestu vpred (PIEST_CHOD_VPRED), jeho koncové polohy detegované snímačmi (PIEST_SNIMAC_VPRED a PIEST_SNIMAC_VZAD) a optický snímač (OS) detegujúci objekt.

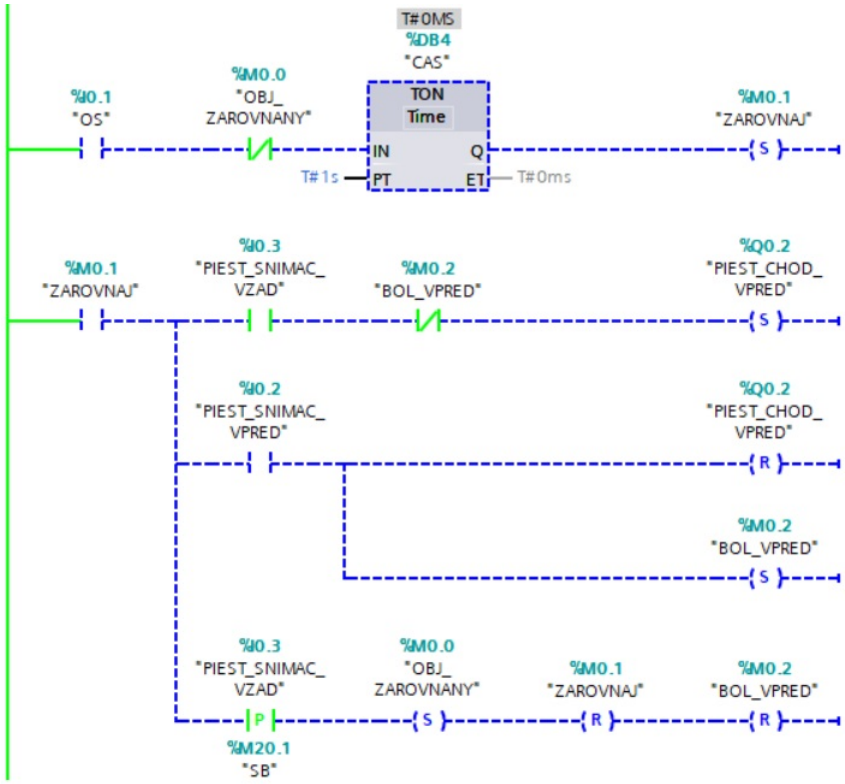
Úloha

Ak je objekt detegovaný optickým snímačom, má sa po 1 s zarovnať pohybom piestu vpred a vzad.

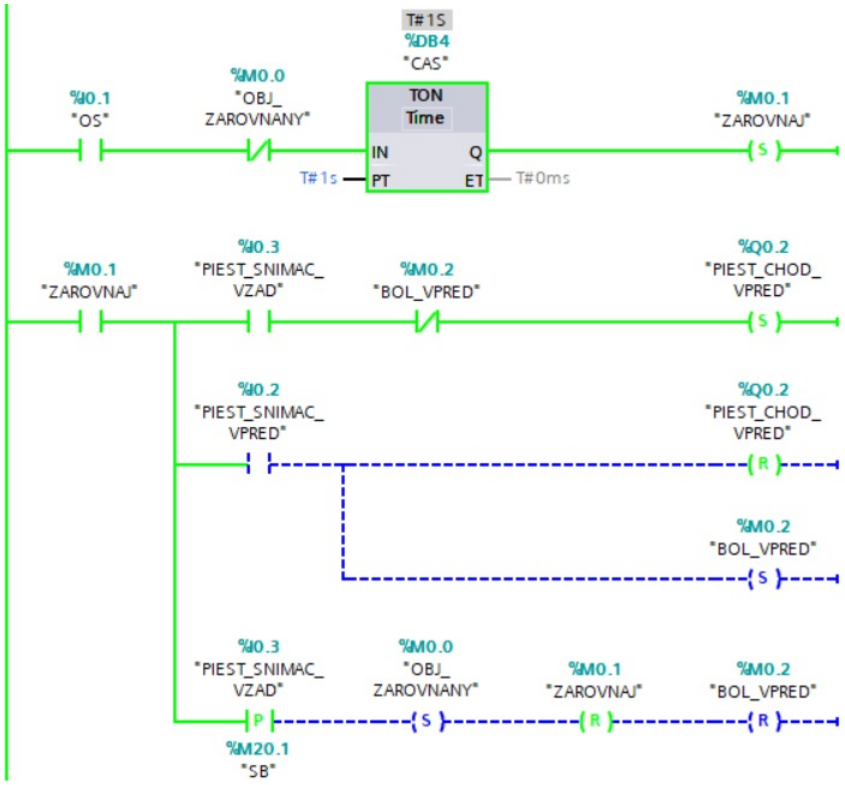
1. riešenie

Príklad riešenia je na Obr. 4.108. Program obsahuje 3 pomocné stavy. Prvým je bit (príkaz) ZAROVNAJ, ktorý má hodnotu TRUE počas procesu zarovnávania. Druhou premennou je OBJ_ZAROVNANY, signalizujúci stav zarovnanosti detegovaného objektu. Tretím je pomocný stav, ktorý bude blokovať prípadné opakovanie pohybu vpred po vykonaní zarovnania. Objekt sa má zarovnať ak je detegovaný optickým snímačom a doposiaľ nebol zarovnaný. Ak platí podmienka AND, spustí sa časovač a po 1 s nastaví pomocný bit ZAROVNAJ na TRUE. V spodnej vetve bude platiť prvá podmienka AND podmienok ZAROVNAJ. Ak je piest v polohe vzad, vykoná sa inštrukcia SET, ktorá vysunie piest vpred. Ak bude vpred, zruší sa pohyb vpred v 2. paralelnej vetve, čím sa priest vráti do východzej polohy (vzad). Zároveň nábežnou hranou polohy vzad sa nastaví pomocný bit OBJ_ZAROVNANY na TRUE a ZAROVNAJ na FALSE.

Na Obr. 4.108 je fáza očakávania objektu. Piest je v polohe vzad. Pomocné premenné majú hodnotu FALSE. Na Obr. 4.109 je detekcia objektu optickým snímačom. Po uplynutí 1 s sa aktivuje stav ZAROVNAJ a v spodnej vetve sa zopol výstup PIEST_CHOD_VPRED.

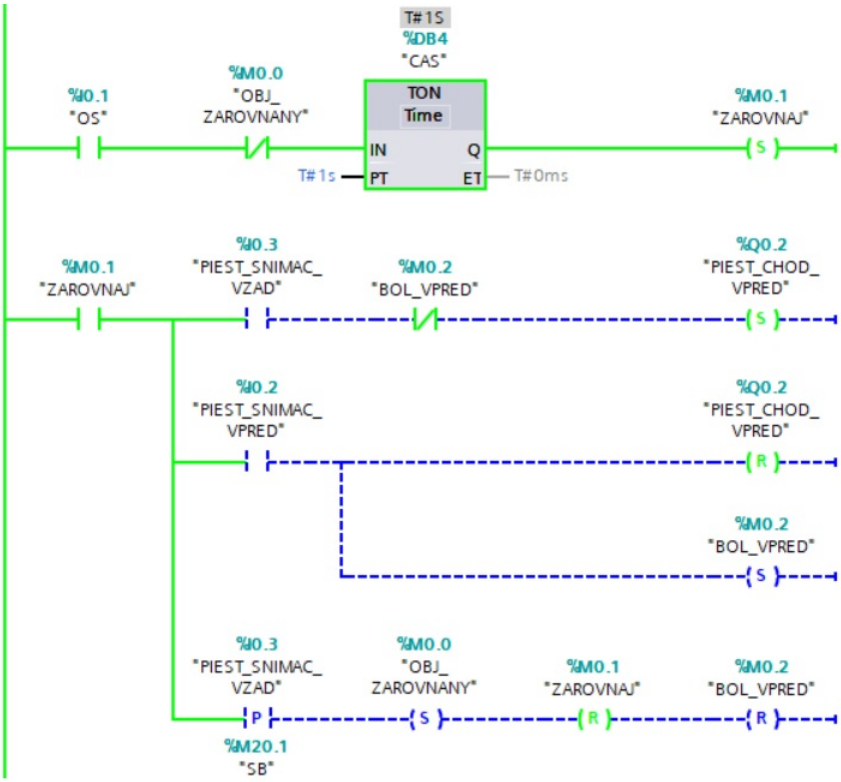


Obr. 4.108. 1. fáza monitorovania programu



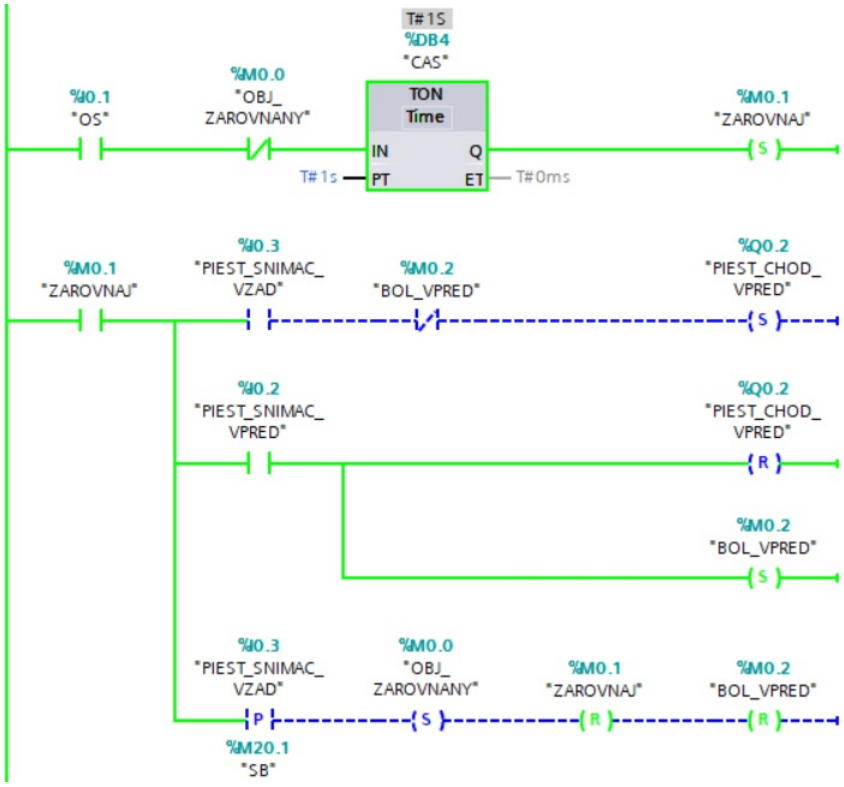
Obr. 4.109. 2. fáza monitorovania programu

Na Obr. 4.110 je fáza pohybu piestu vpred. Piest je v medzipolohe, lebo jeho poloha nie je detegovaná koncovými snímačmi.

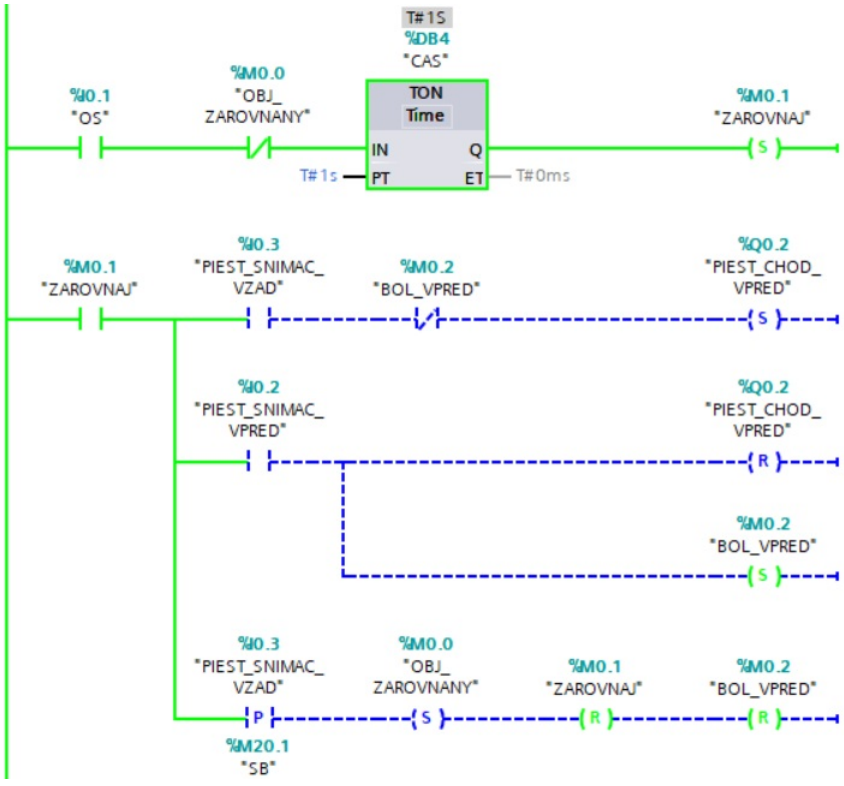


Obr. 4.110. 3. fáza monitorovania programu

Na Obr. 4.111 piest dosiahol koncovú polohu vpred. Pokiaľ je aktivovaný snímač PIEST_SNIMAC_VPRED, vykonáva sa RESET pohybu vpred, t. j. aktivuje sa pohyb vzad. Zopnutím snímača vpred sa nastavila premenná BOL_VPRED na TRUE. Na Obr. 4.112 piest vykonáva pohyb vzad a je v medzipolohe.

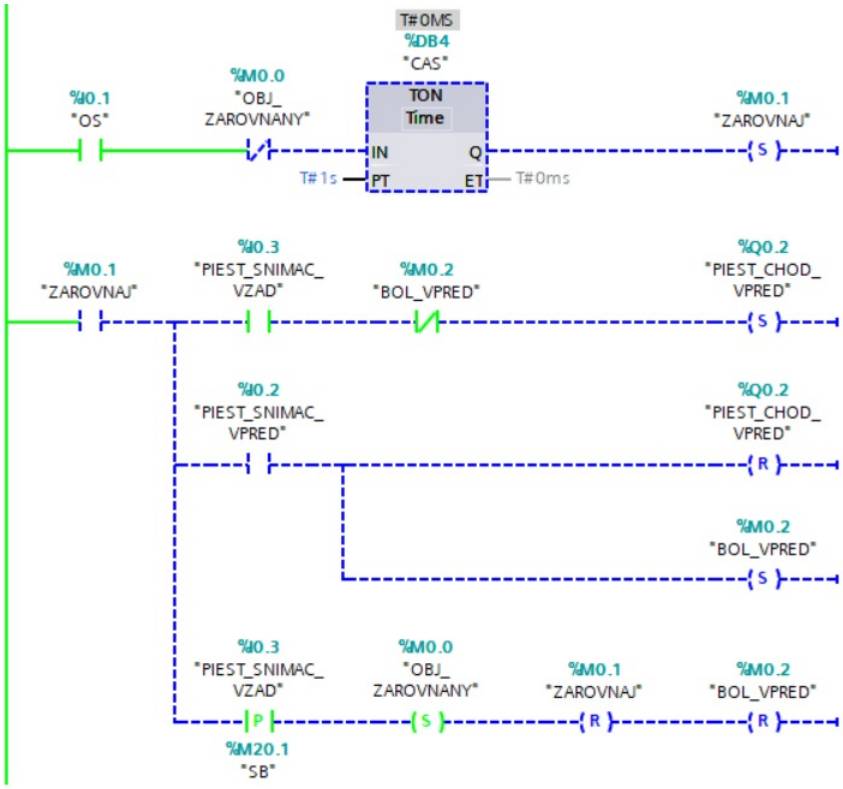


Obr. 4.111. 4. fáza monitorovania programu



Obr. 4.112. 5. fáza monitorovania programu

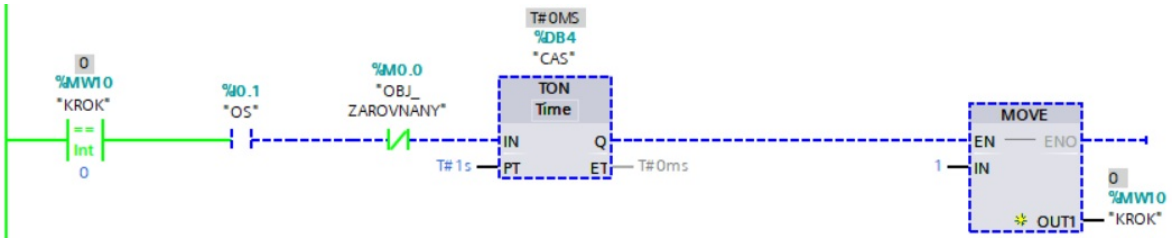
Na Obr. 4.113 sa priest vrátil do polohy vzad. Detegovala sa nábežná hrana v spodnej vetve, čím sa vynulovali dve pomocné premenné a premenná OBJ_ZAROVNANY nastavila na TRUE. Tá blokuje opätovné spustenie časovača a vykonanie sekvencie.



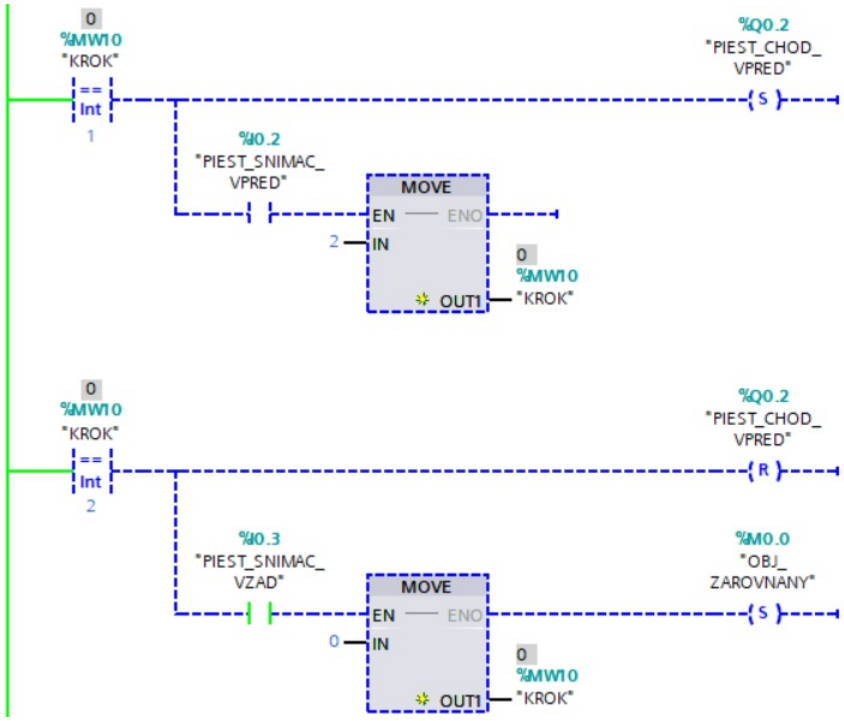
Obr. 4.113. 6. fáza monitorovania programu

2. riešenie

V určitých situáciách sa hodí tzv. krokové riadenie. V závislosti od celočíselnej hodnoty premennej sa vykonávajú sady príkazov. Ak sú splnené podmienky, prechádza sa do ďalšieho kroku, v ktorom sa realizujú iné príkazy a opäť sa čaká na splnenie podmienok na prechod do nového kroku. Príklad sme rozdelili na rebríky (Networky), ktoré sú na Obr. 4.114 a 4.115. Prvá časť očakáva objekt. Súčasťou známej podmienky z 1. riešenia je porovnávací blok. Ak je hodnota premennej KROK = 0 a platia zvyšné podmienky, spustí sa časovač. Po ukončení časovania sa do premennej KROK uloží hodnota 1. To zároveň preruší podmienku AND na vstupe, lebo v opačnom prípade by sa v každom cykle do premennej KROK zapisovala hodnota 1. Túto časť, kedy KROK = 0, by sme mohli nazvať čakanie na príchod objektu. V nižšom monitorovanom programe (Obr. 4.114 a 4.115) neplatia podmienky porovnania.

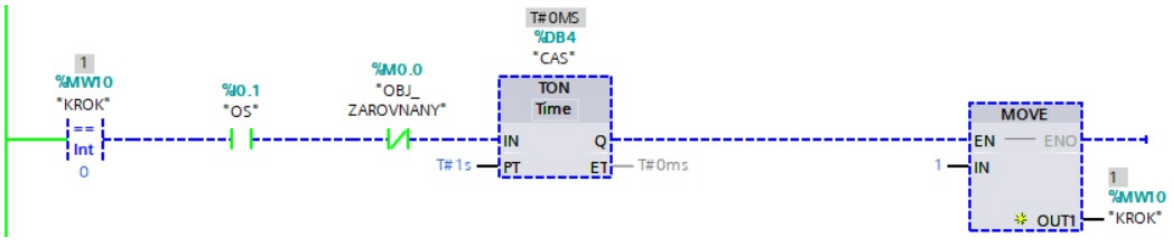


Obr. 4.114. 1. fáza monitorovania programu - Network 1

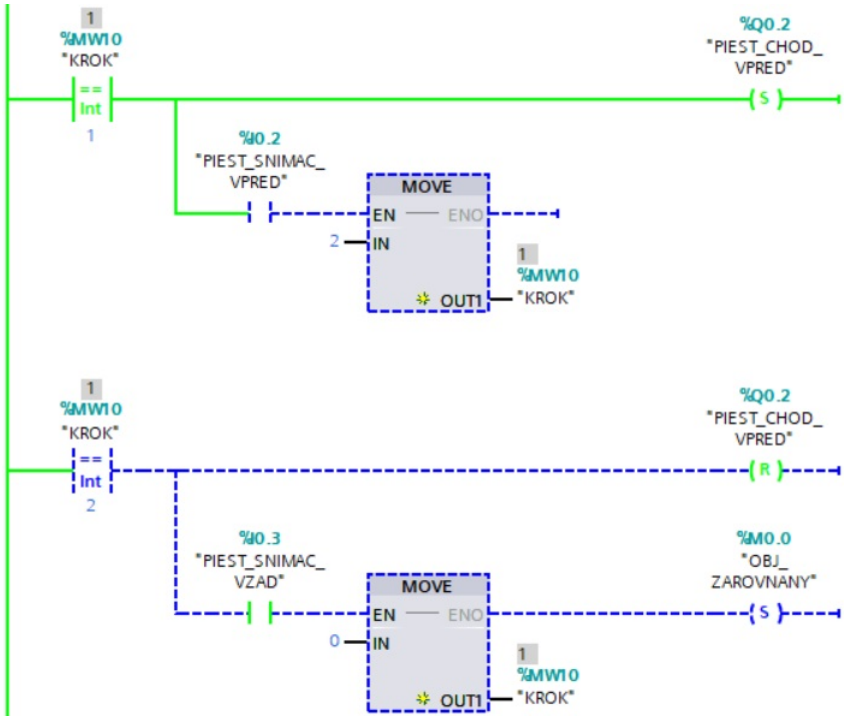


Obr. 4.115. 1. fáza monitorovania programu - Network 2

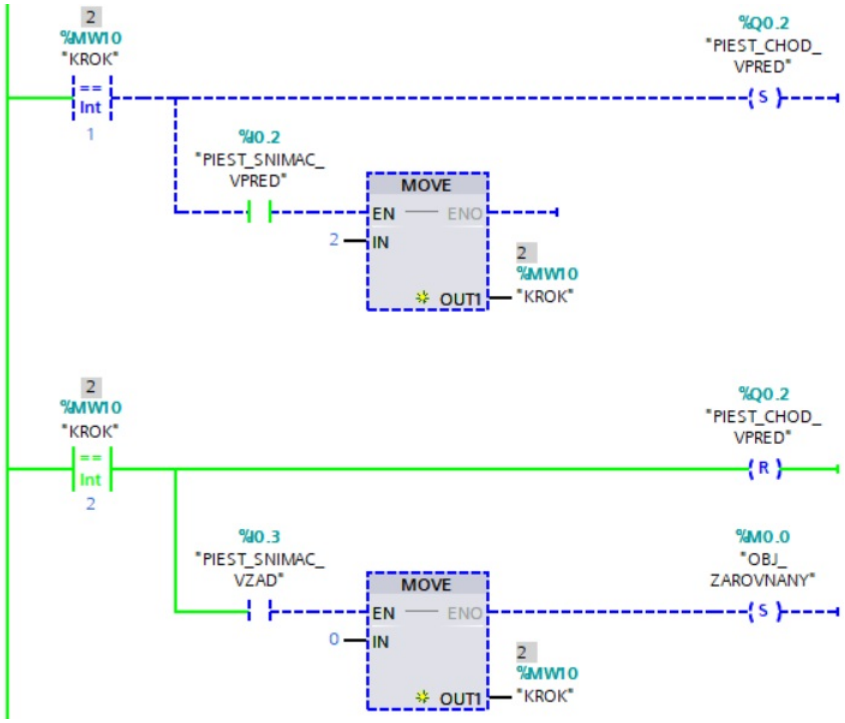
Na Obr. 4.116 bol detegovaný objekt a do premennej KROK sa zapísala hodnota 1. Na Obr. 4.117 sa aktivovala podmienka KROK = 1. Vykonáva sa pohyb vpred. Po dosiahnutí polohy vpred sa do premennej KROK zapíše hodnota 2 (Obr. 4.118). V kroku 2 sa aktivuje pohyb vzad a čaká sa na koncovú polohu vzad. Po dosiahnutí polohy vzad sa nastaví hodnota premennej KROK na 0 (inicializačný krok očakávajúci nový objekt) a nastaví pomocná premenná OBJ_ZAROVNANY na TRUE.



Obr. 4.116. 2. fáza monitorovania programu - Network 1

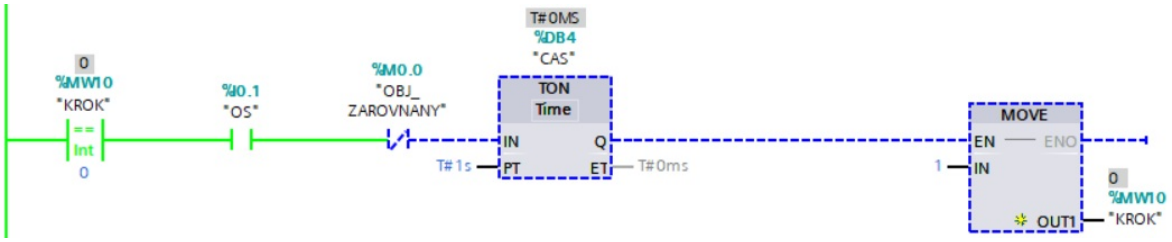


Obr. 4.117. 2. fáza monitorovania programu - Network 2

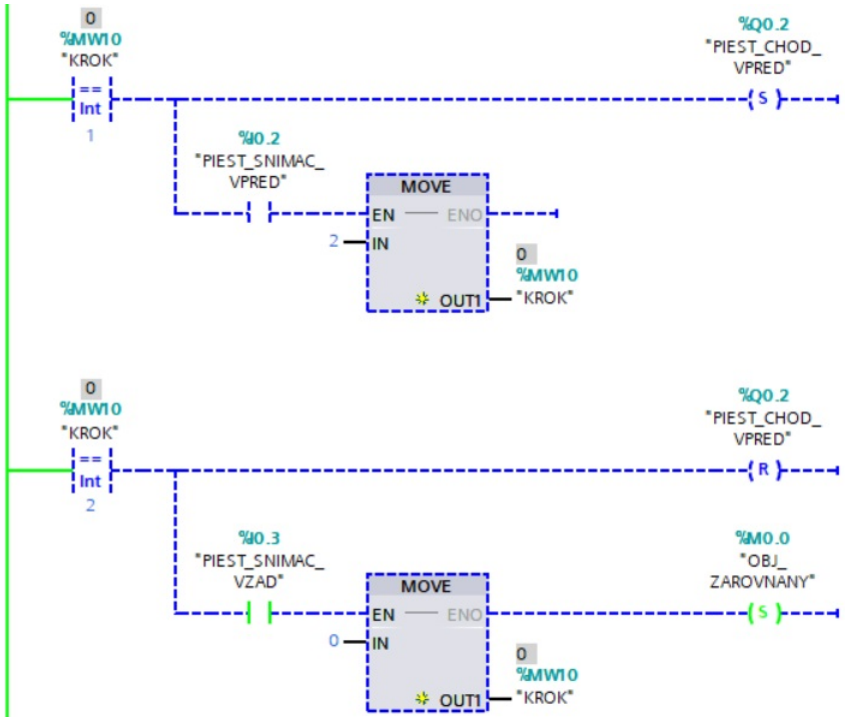


Obr. 4.118. 3. fáza monitorovania programu - Network 2

Na Obr. 4.119 sa piest vrátil do polohy vzad. Je aktívny inicializačný krok a objekt bol zarovnaný. Na Obr. 4.120 sa nevykonávajú žiadne príkazy ovládajúce technológiu.



Obr. 4.119. 4. fáza monitorovania programu - Network 1



Obr. 4.120. 4. fáza monitorovania programu - Network 2

4.3.13 Príklad 13 - Ovládanie stroja

Majme stroj bez spätnej väzby o chode, ktorý simuluje nejaký úkon (napr. vŕtanie, leštenie, nanášanie farby ať.). Ovláda sa jedným digitálnym výstupom STROJ_ZAP. Neskorším cieľom bude spojiť riadenie dopravníka z predošlých úloh s riadením stroja.

Úloha

Stroj má vykonať trikrát sekvenciu zapnutia a vypnutia výstupu stroja. Zapnutie trvá 1 s a vypnutie 2 s.

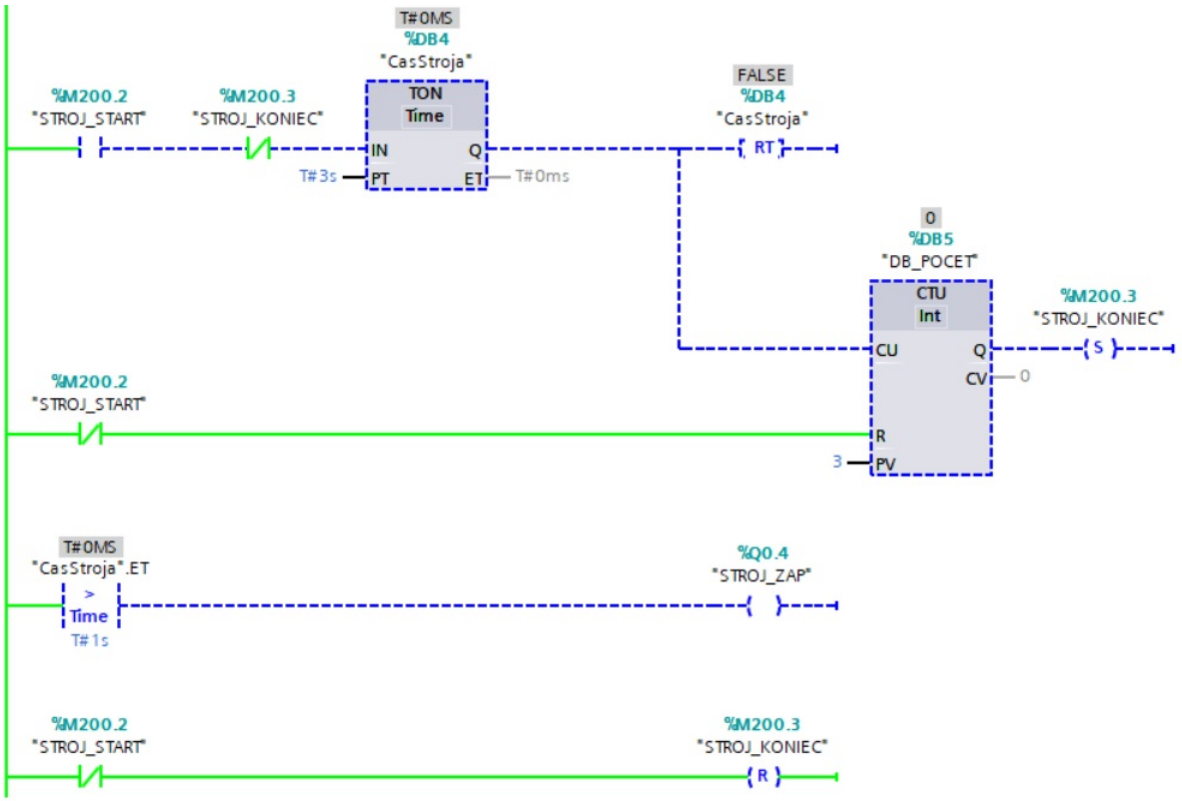
1. riešenie

Na časové ovládanie stroja použijeme časovač. Spustenie časovača vyžaduje podmienku. Vytvoríme pomocný signál STROJ_START, ktorý povolí štart stroja. Ten bude neskôr

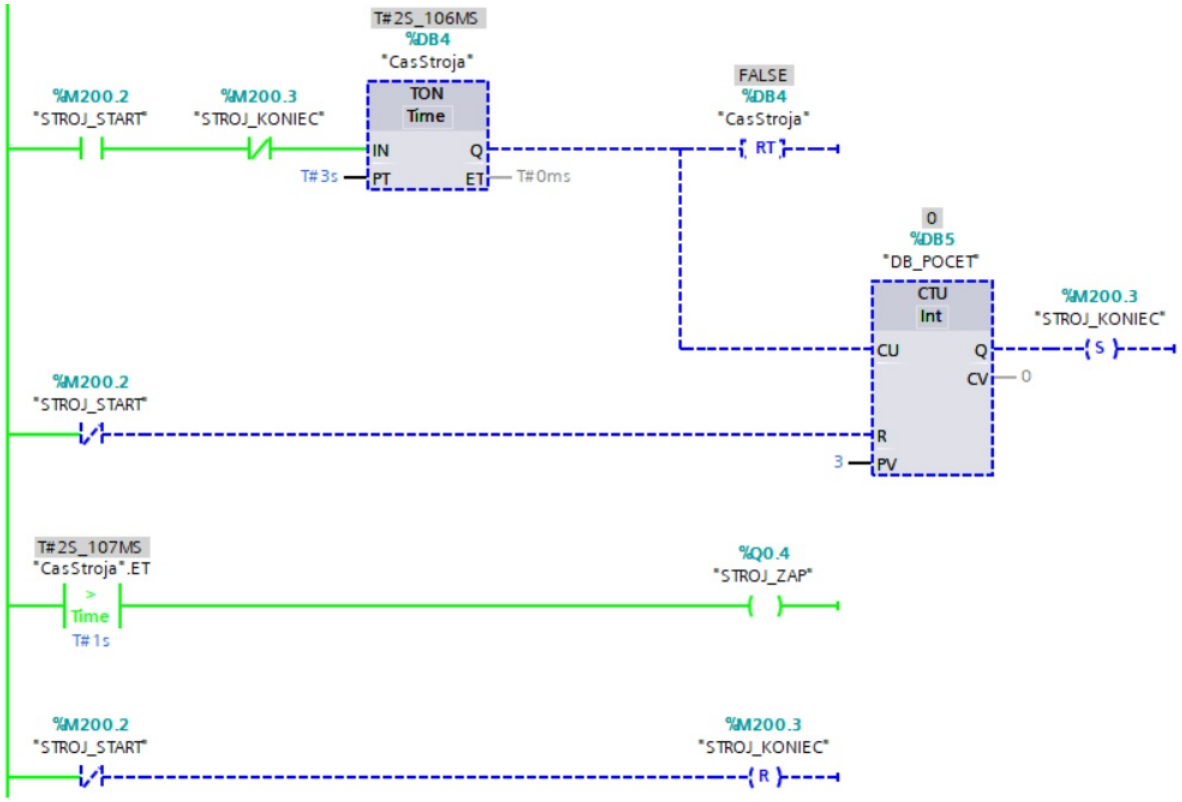
ovládaný dopravníkom. V tomto príklade použijeme počítadlo. To, že je ukončený požadovaný počet opakovaní (zapnutia a vypnutia stroja) môže byť signalizované výstupom QU počítadla, zistené porovnávacím blokom alebo priradením spomínaných stavov do novej pamäťovej premennej. V praxi sa vytvárajú stavy technológie, preto vytvoríme novú premennú STROJ_KONIEC.

Ak sa manuálne nastaví hodnota premennej STROJ_START a nie je koniec sekvencie (STROJ_KONIEC je FALSE), spustí sa časovač. Po 3 s sa navýši hodnota počítadla a vynuluje časovač, ktorý v ďalšom cykle začne časovanie pri platnej vstupnej podmienke. Ak program dosiahol 3 opakovania, hodnota STROJ_KONIEC bude TRUE. Ten v ďalšom cykle zabráni opätovnému privedeniu stavu TRUE na vstup IN časovača (druhá časť podmienky AND nebude platiť). Počas navyšovania času v intervale 0 ms až 1 s je výstup STROJ_START FALSE a v intervale 1 s až 3 s je TRUE, t. j. 1 s je stroj vypnutý a 2 s je zapnutý. Stav dosiahnutia počtu opakovaní (hodnota počítadla a aj premenná STROJ_KONIEC) je potrebné nulovať od vhodnej podmienky. Po zmene podmienky povolenia práce stroja na FALSE sa príslušné stavy vynulujú.

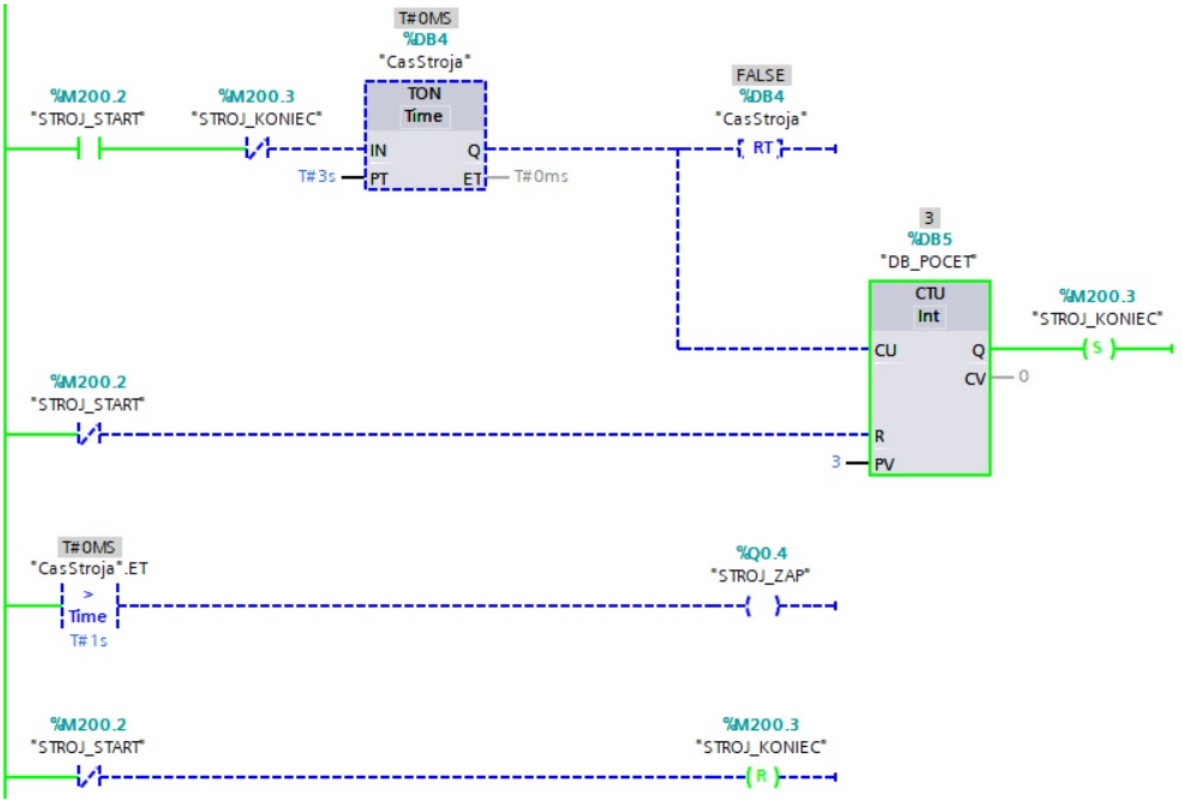
Animácia počiatočného stavu je na Obr. 4.121. Na Obr. 4.122 je proces sekvencie stroja. Na poslednom Obrázku 4.123 je stav ukončenej sekvencie.



Obr. 4.121. 1. fáza monitorovania programu



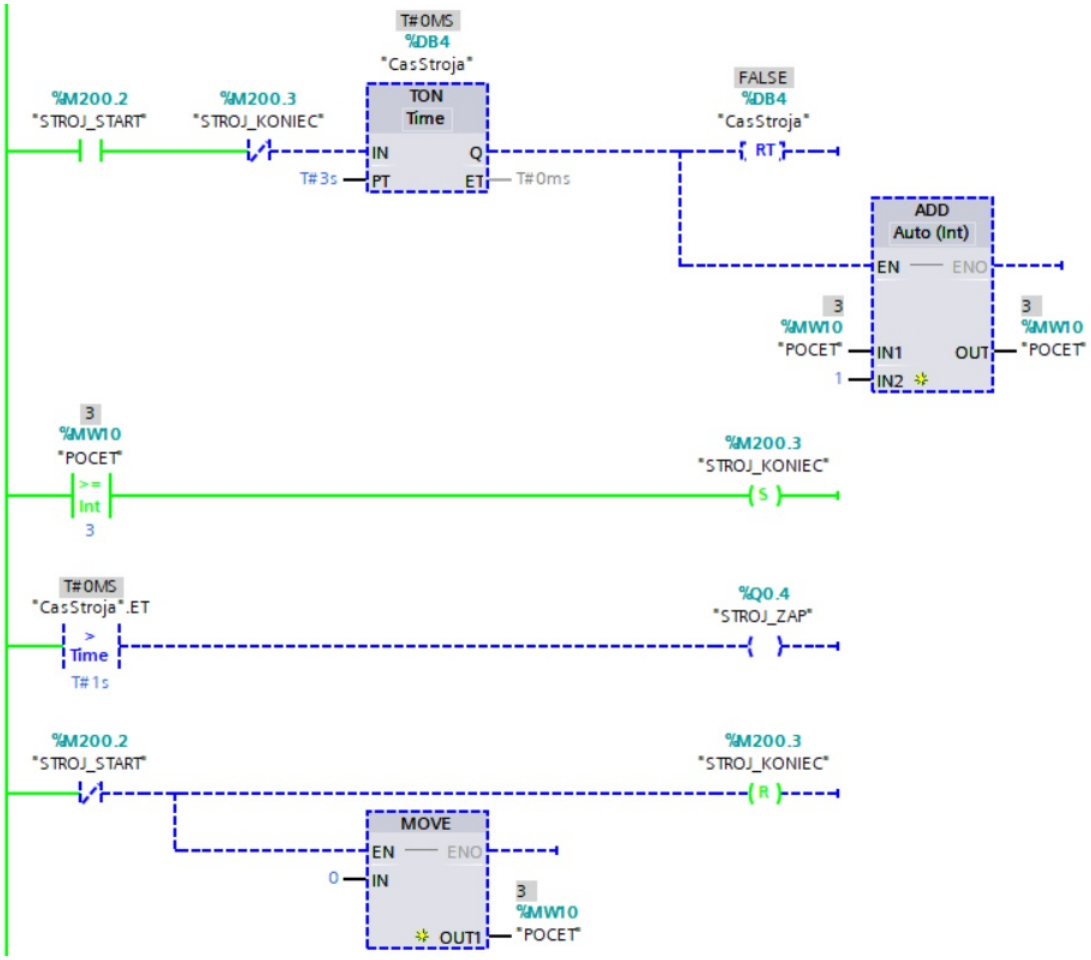
Obr. 4.122. 2. fáza monitorovania programu



Obr. 4.123. 3. fáza monitorovania programu

2. riešenie

Počítadlo nahradíme inštrukciami *ADD* a *MOVE*. Inštrukcia *ADD* navyšuje počet opakovaní sekvencie stroja. Inštrukcia *MOVE* bude nulovať počet pri zmene stavu STROJ_START na FALSE. Zvyšné časti programu sú totožné s 1. riešením. Výsledný program s monitorovaním stavu ukončenej sekvencie je na Obr. 4.124.



Obr. 4.124. Nové riešenia úlohy

4.3.14 Príklad 14 - Motohodiny 1

V predchádzajúcich dvoch príkladoch boli naznačené postupy vytvárania pomocných stavov (premenných). Majme dopravník z kap. 4.3.5 Príklad 5 - Ovládanie dopravníka 1. Dopravník bol ovládaný výstupom D1 s fyzickou adresou %Q0.2. V tomto príklade nebude cieľom návrh rôznych algoritmov riadenia dopravníka, ale vytvorenie nových premenných. Tieto premenné sa využívajú napr. pri monitorovaní stavu riadenia objektov a pod. Na úvod si vytvoríme tzv. motohodiny, ktoré reprezentujú pracovný čas motora.

Úloha

Vytvorte dátový blok MOTOR a v ňom tieto premenné:

- SEKUNDY (DINT) - reprezentuje čas chodu motora v sekundách,
- DATUM_CAS (DTL) - v premennej sa bude uchovávať dátum a čas posledného nulovania času chodu motora,
- NULUJ (BOOL) - bit, ktorý vyvolá nulovanie motohodín a uloženie dátumu a času nulovania.

Vytvorte program, ktorý bude navyšovať hodnotu premennej SEKUNDY každú sekundu ak je motor zapnutý. V prípade, že premenná NULUJ nadobudne hodnotu TRUE, premenná SEKUNDY sa vynuluje, do premennej DATUM_CAS sa zapíše aktuálny dátum a čas nulovania motohodnín a premenná NULUJ sa tiež vynuluje.

Štruktúra dátového bloku je na Obr. 4.125. Premenné SEKUNDY a DATUM_CAS boli nastavené ako retenčné, aby sa ich hodnota zachovala. Premenná DATUM_CAS je štruktúrovaná premenná, ktorej štruktúra je na Obr. 4.125. Obsahuje premenné ako rok, mesiac, deň a pod.

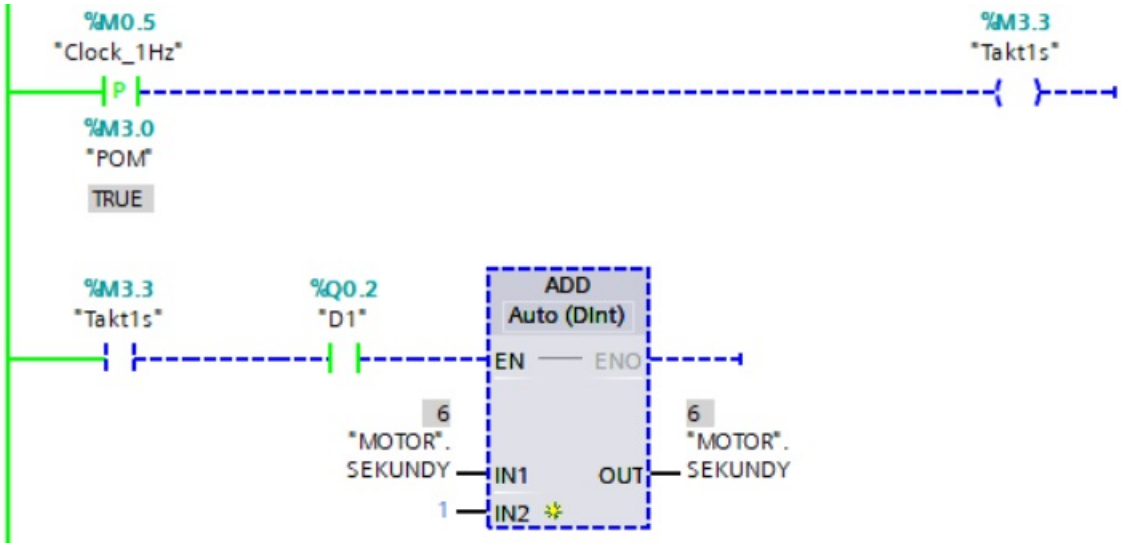
MOTOR

	Name	Data type	Start value	Retain
1	Static			
2	SEKUNDY	DInt	0	☑
3	DATUM_CAS	DTL	DTL#1970-01-01-00:00:00	☑
4	YEAR	UInt	1970	☑
5	MONTH	USInt	1	☑
6	DAY	USInt	1	☑
7	WEEKDAY	USInt	5	☑
8	HOUR	USInt	0	☑
9	MINUTE	USInt	0	☑
10	SECOND	USInt	0	☑
11	NANOSECOND	UDInt	0	☑
12	NULUJ	Bool	false	☐

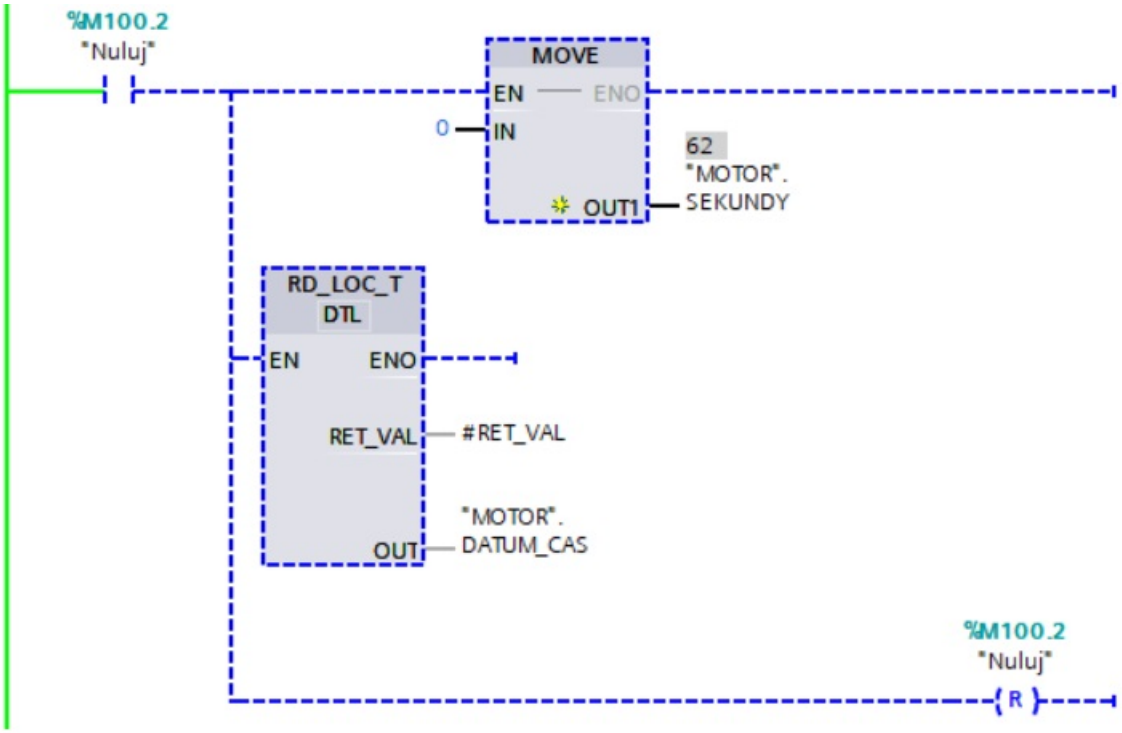
Obr. 4.125. Štruktúra dátového bloku MOTOR

Riešenie programu je na Obr. 4.126 a 4.127. Na prvom sa generuje nábežná hrana každú sekundu. Ak je motor zapnutý (D1 je TRUE), každú sekundu sa navyšuje hodnota premennej SEKUNDY. V praxi by sme využili spätnú väzbu (digitálny vstup) o chode motora, ktorý môže mať význam zopnutého stýkača, kontaktu meniča a pod. Na druhom obrázku je časť programu zodpovedná za nulovanie. Ak má NULUJ stav TRUE, do premennej SEKUNDY sa uloží hodnota 0, inštrukciou RD_LOC_T (Read Local Time) sa vyčíta aktuálny dátum a čas CPU a vynuluje sa hodnota NULUJ. Výstupom inštrukcie je vrátený dátum a čas v štruktúre DTL (OUT) a RET_VAL dátového typu Int indikujúca prípadné chyby. V príklade bola použitá dočasná (temp) premenná.

Príklad monitorovania dátového bloku po vymazaní času v premennej SEKUNDY je na Obr. 4.128. Na obrázku vidno nenulovú hodnotu premennej DATUM_CAS, vynulovanú hodnotu NULUJ a postupne navyšovanú hodnotu SEKUNDY po vynulovaní.



Obr. 4.126. 1. časť riešenia



Obr. 4.127. 2. časť riešenia

MOTOR

	Name	Data type	Start value	Monitor value
1	Static			
2	SEKUNDY	DInt	0	15
3	DATUM_CAS	DTL	DTL#1970-01-01-00:00:00	DTL#2022-01-25-14:49:17.016180
4	YEAR	UInt	1970	2022
5	MONTH	USInt	1	1
6	DAY	USInt	1	25
7	WEEKDAY	USInt	5	3
8	HOUR	USInt	0	14
9	MINUTE	USInt	0	49
10	SECOND	USInt	0	17
11	NANOSECOND	UDInt	0	16180000
12	NULUJ	Bool	false	FALSE

Obr. 4.128. Monitorované hodnoty v dátovom bloku

4.3.15 Príklad 15 - Motohodiny 2

V predošlom príklade sme vytvorili jednoduchý program rátania motohodín v sekundách. V praxi nie je rozumné zobraziť operátorom motohodiny v podobe sekúnd, ale zobraziť napr. v podobe sekúnd, minút a hodín.

Úloha

Upravme dátový blok MOTOR:

- SEKUNDY (USINT) - reprezentuje čas chodu motora v sekundách v rozsahu 0 - 59 s,
- MINUTY (USINT) - reprezentuje čas chodu motora v minútach v rozsahu 0 - 59 min,
- HODINY (UINT) - reprezentuje čas chodu motora v hodinách,
- DATUM_CAS (DTL) - v premennej sa bude uchovávať dátum a čas posledného nulovania času chodu motora,
- NULUJ (BOOL) - bit, ktorý vyvolá nulovanie motohodín a uloženie dátumu a času nulovania.

Vytvorte program, ktorý bude navyšovať hodnotu premennej SEKUNDY každú sekundu ak je motor zapnutý. V prípade dosiahnutia 60s sa navýši hodnota premennej MINUTY a vynuluje sa hodnota premennej SEKUNDY. Ak premenná MINUTY nadobudne hodnotu 60, navýši sa hodnota premennej HODINY a vynuluje sa premenná MINUTY.

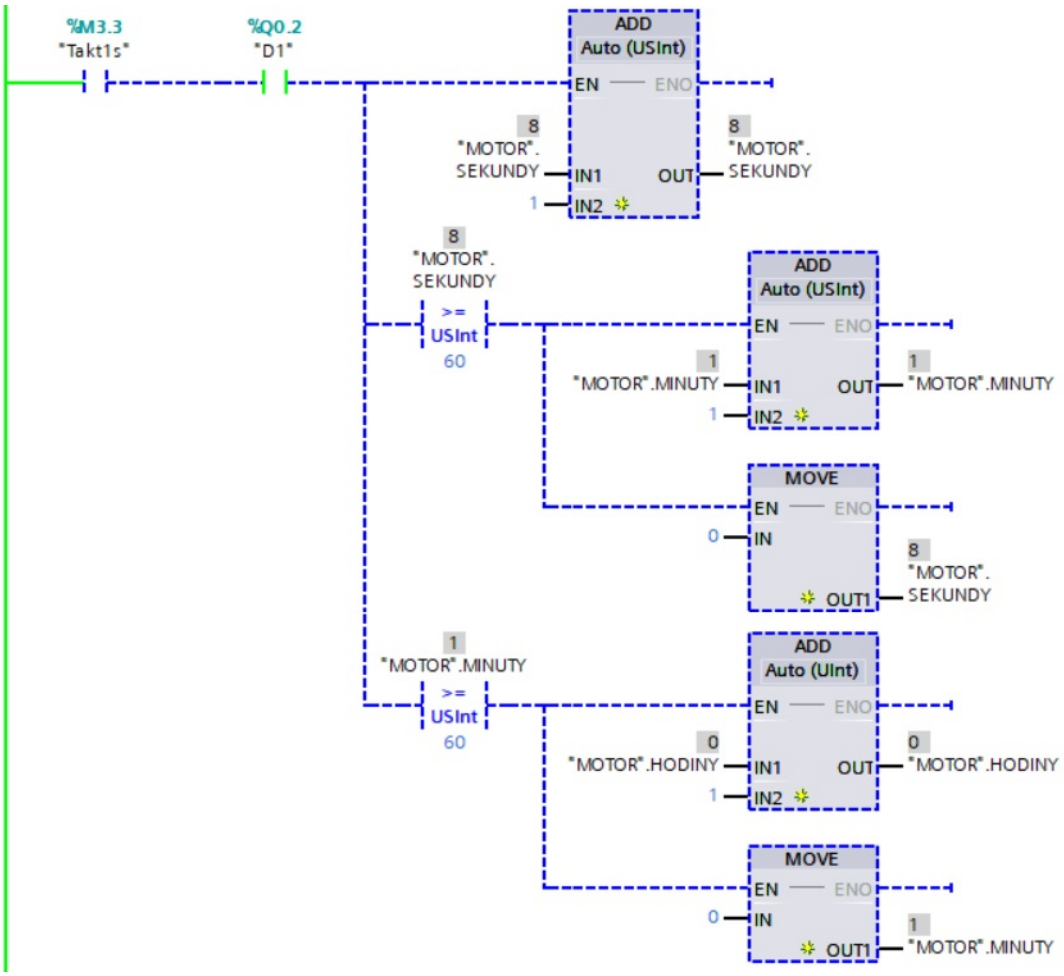
Upravený dátový blok MOTOR podľa zadania je na Obr. 4.129. Pre premenné SEKUNDY a MINUTY postačujú dátové typy menších (kladných) rozsahov.

MOTOR

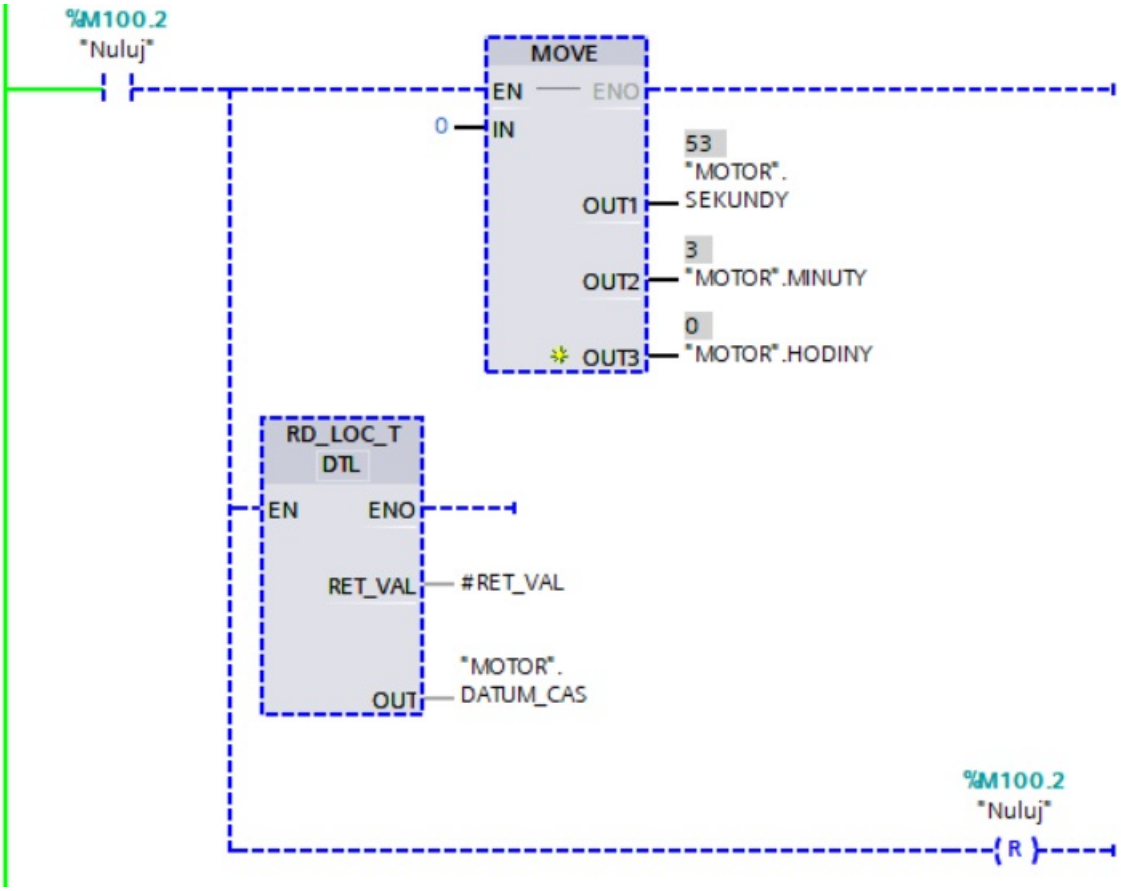
	Name	Data type	Start value	Retain
1	Static			
2	SEKUNDY	USInt	0	☑
3	MINUTY	USInt	0	☑
4	HODINY	UInt	0	☑
5	DATUM_CAS	DTL	DTL#1970-01-01-00:00:00	☑
6	NULUJ	Bool	false	☐

Obr. 4.129. Upravený dátový blok MOTOR

Program, ktorý implementuje motohodiny, je na Obr. 4.130 a 4.131. Na prvom obrázku je vynechané generovanie signálu Takt1s. Postupne sa navyšuje hodnota premennej SEKUNDY. Ak hodnota premennej bude väčšia alebo rovná 60, navýši sa hodnota premennej MINUTY a vynuluje sa hodnota premennej SEKUNDY. Rovnakým spôsobom ak dosiahneme hodnotou premennej MINUTY väčšiu alebo rovnú 60, navýši sa hodnota premennej HODINY a vynuluje sa hodnota premennej MINUTY. Na druhom obrázku bol počet výstupov inštrukcie *MOVE* rozšírený na 3. Inštrukcia vynuluje všetky premenné motohodín.



Obr. 4.130. 1. časť riešenia



Obr. 4.131. 2. časť riešenia

4.3.16 Príklad 16 - Riadenie skipu

V tomto príklade bude snahou ukázať tvorbu komplexnejšieho programu na príklade riadenia skipu. Skip je v podstate vozík, do ktorého sa v betonárkach dávkujú rôzne frakcie štrku. Nadávkované množstvo kameniva sa vynáša do miešačky, ktorá je situovaná vyššie ako miesto váženia a dávkovania kameniva. Chod skipu hore a dolu je ovládaný asynchrónnym motorom riadeným frekvenčným meničom. Pohyb je ovládaný navíjaním alebo odvíjaním oceľového lana v závislosti od smeru otáčania motora. Skip má brzdu, aby vedel zastať v ľubovoľnej polohe. Brzdu môže riadiť frekvenčný menič alebo PLC. V našom príklade ju bude riadiť frekvenčný menič, preto nepotrebujeme digitálny výstup na jej ovládanie. Zoznam vstupov a výstupov je Obr. 4.132. Všetky vstupy sú zapojené ako spínacie kontakty. Výstupy START_HORE a START_DOLE ovládajú smer pohybu. Výstupy RYCHLOST_POMALY a RYCHLOST_RYHCHLO definujú rýchlosť pohybu. Pri pohybe zo spodnej polohy (SNIMAC_DOLNY) po snímač SNIMAC_SPOMAL sa skip pohybuje rýchlo a potom spomalí od snímača SNIMAC_SPOMAL po snímač SNIMAC_HORNY. Smerom dolu sa bude pohybovať pomalou rýchlosťou.

Default tag table

		Name	Data type	Address
1		START_HORE	Bool	%Q0.0
2		START_DOLE	Bool	%Q0.1
3		RYCHLOST_POMALY	Bool	%Q0.2
4		RYCHLOST_RYCHLO	Bool	%Q0.3
5		SNIMAC_DOLNY	Bool	%I0.0
6		SNIMAC_HORNY	Bool	%I0.1
7		SNIMAC_SPOMAL	Bool	%I0.2

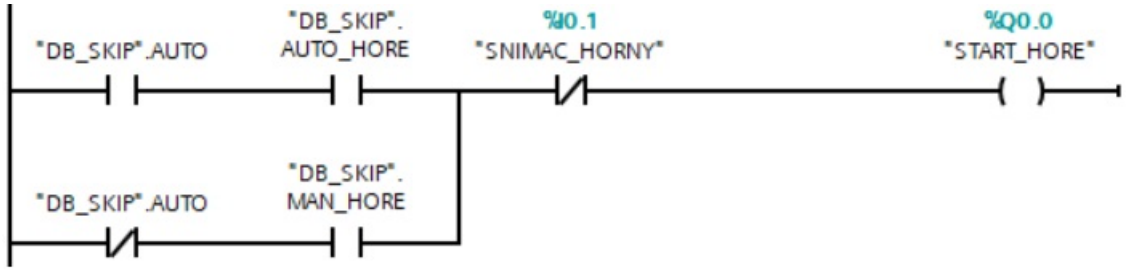
Obr. 4.132. Zoznam vstupov a výstupov

Úloha

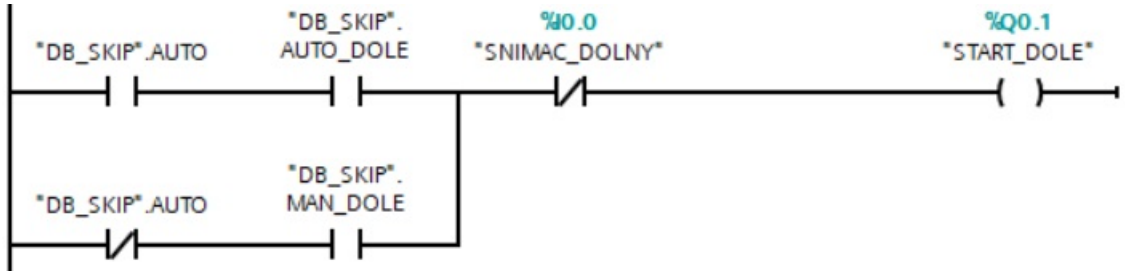
Vytvorte program riadenia skipu, ktorý má zahŕňať:

1. riadenie v automatickom a manuálnom režime,
2. riadenie chodu hore v automatickom režime, ak je skip plný a miešačka očakáva novú dávku,
3. riadenie chodu dole v automatickom režime, ak je skip prázdny,
4. riadenie rýchlostí na základe všeobecného opisu skipu,
5. generovanie alarmov (nedošiel do dolnej polohy, nedošiel do hornej polohy, neaktivoval sa spomaľovací snímač atď.),
6. generovanie pomocných stavov pre vizualizáciu (skip je medzi dolným a spomaľovacím snínačom, skip je medzi spomaľovacím snímačom a horným snímačom).

Pomocné stavy skipu vytvoríme v dátovom bloku DB_SKIP. Postupne budeme premenné pridávať a na konci uvedieme zoznam všetkých premenných. Pre splnenie prvej úlohy vytvorme premennú AUTO dátového typu bool. Ak bude táto premenná v stave TRUE, skip je v automatickom režime. Ak bude táto premenná v stave FALSE, skip je v manuálnom režime. Vytvorme premenné na automatický a manuálny chod skipu hore a dole, t. j. 4 premenné (AUTO_HORE, AUTO_DOLE, MAN_HORE, MAN_DOLE). Riešenie prvej úlohy je na Obr. 4.133 a 4.134.



Obr. 4.133. Všeobecná časť riadenia skipu hore v automatickom a manuálnom režime

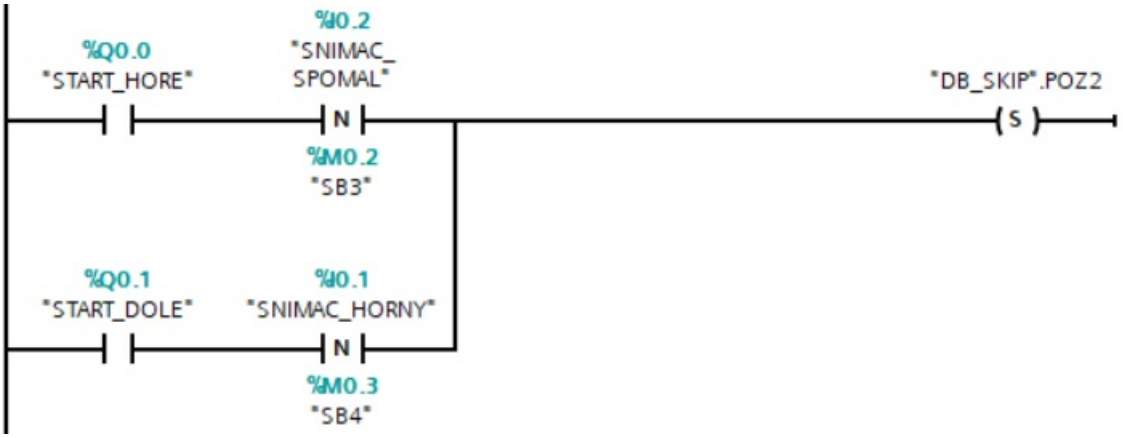


Obr. 4.134. Všeobecná časť riadenia skipu dole v automatickom a manuálnom režime

V uvedených programoch horná vetva bude platná len v automatickom a dolná vetva len v manuálnom režime. Prepínanie režimu AUTO sa často realizuje softvérovým tlačidlom vo vizualizácii. Tlačidlo s textom AUTO vo vizualizácii realizuje nastavenie premennej AUTO na TRUE. Tlačidlo s nápisom MAN vo vizualizácii realizuje nastavenie premennej AUTO na FALSE. Premenné AUTO_HORE a AUTO_DOLE budú ovládané v neskorších úlohách navrhnutým algoritmom riadenia. Premenné MAN_HORE a MAN_DOLE sa analogicky ako premenná AUTO ovládajú z vizualizácie (prípadne to môžu byť fyzické tlačidlá). Výstup START_HORE je aktivovaný pokiaľ skip nebude detegovaný horným snímačom (SNIMAC_HORNY) a opačne, výstup START_DOLE je aktivovaný pokiaľ skip nebude detegovaný dolným snímačom (SNIMAC_DOLNY).

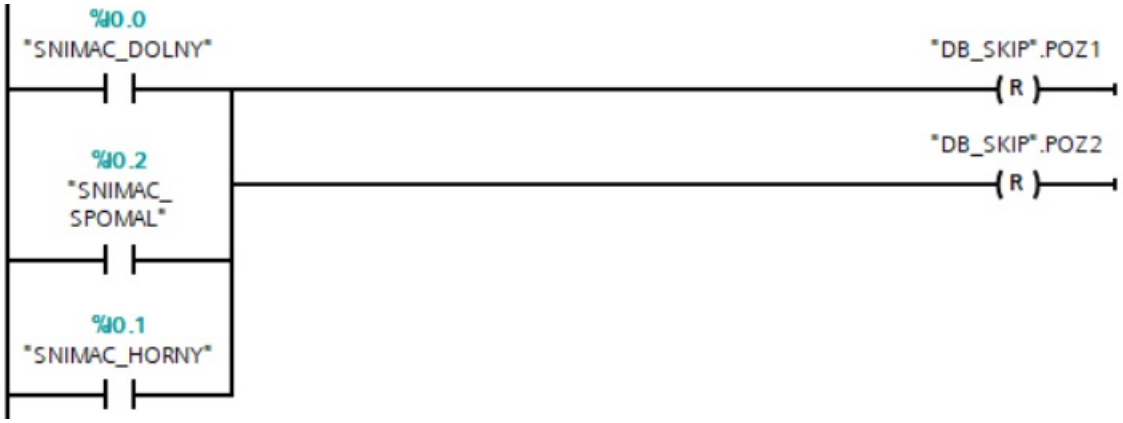
V druhom kroku vyriešme 6. úlohu. Pomocné stavy sa dajú vhodne integrovať do algoritmov riadenia skipu. Definujme premenné POZ1 a POZ2. Premenná POZ1 bude v stave TRUE, ak bude skip v spodnej časti dráhy medzi snímačmi SNIMAC_DOLNY a SNIMAC_SPOMAL, inak bude v stave FALSE. Premenná POZ2 bude v stave TRUE, ak bude skip v hornej časti dráhy medzi snímačmi SNIMAC_HORNY a SNIMAC_SPOMAL, inak bude v stave FALSE. Riešenie vyžaduje detekcie hrán (Obr. 4.135 a 4.136). Pomocné bity hrán vytvoríme v pamäťovej oblasti %M, aby sme oddelili užitočné stavy od pomocných. Ak spúšťame skip smerom hore a detegujeme dobežnú hranu na dolnom snímači, znamená to odchod skipu zo spodnej pozície, preto sa nastaví premenná POZ1 na TRUE. Druhá verva realizuje nastavenie rovnakého stavu na TRUE pri chode skipu dole a prechodom cez spomaľovací snímač (kedy klesne pod snímač a už nie je detekovaný). Analogický program je pre POZ2. Všimnite si duplicitu detekcie dobežnej hrany spomaľovacieho snímača. Na zjednodušenie programu by sme mohli vytvoriť pomocnú premennú, do ktorej by sme zapisovali výsledok detekcie hrany a ten by sme vložili namiesto aktuálnych podmienok.

Obr. 4.135. Zapnutie pomocného stavu POZ1



Obr. 4.136. Zapnutie pomocného stavu POZ2

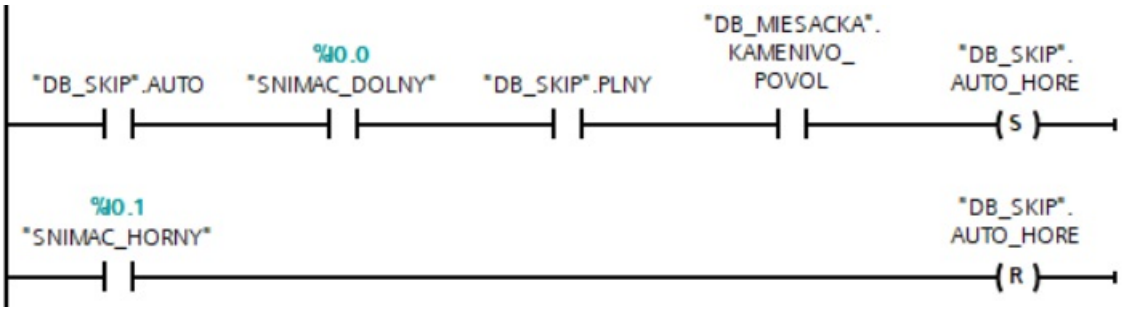
Nastavené pomocné premenné je nutné nulovať. Riešenie je na Obr. 4.137. Aktiváciou jedného zo snímačov sa deaktivujú pomocné medzistavy.



Obr. 4.137. Vypnutie pomocných stavov POZ1 a POZ2

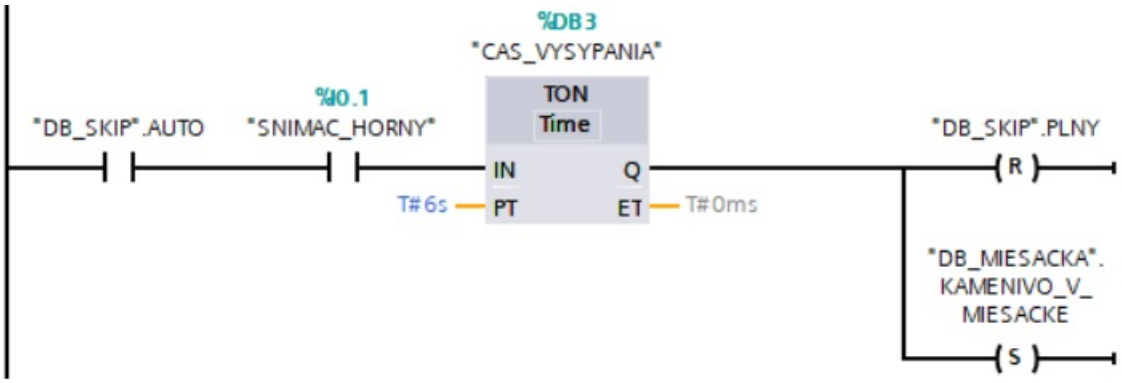
Vytvorme program, ktorý splní úlohu č. 2. Signál skip je plný doplníme do dátového bloku skipu a signál od miešačky do nového dátového bloku DB_MIESACKA. Príklad

riešenia úlohy je na Obr. 4.138. Ak je skip v automatickom režime, je v dolnej polohe, je plný a miešačka očakáva kamenivo, nastaví sa premenná AUTO_HORE na TRUE. Signál skip plný by sa nastavil po navážení kameniva. Na zjednodušenie neuvádzame riadenie okolitých zariadení. Podobne, premenná KAMENIVO_POVOL môže zahŕňať stav miešačka je prázdna, je spustená, segment miešačky (dvierka) sú zatvorené a kamenivo nebolo nadávkované. Pohyb hore sa musí vypnúť dosiahnutím horného snímača (pozri spodná vetva rebríkovej schémy).



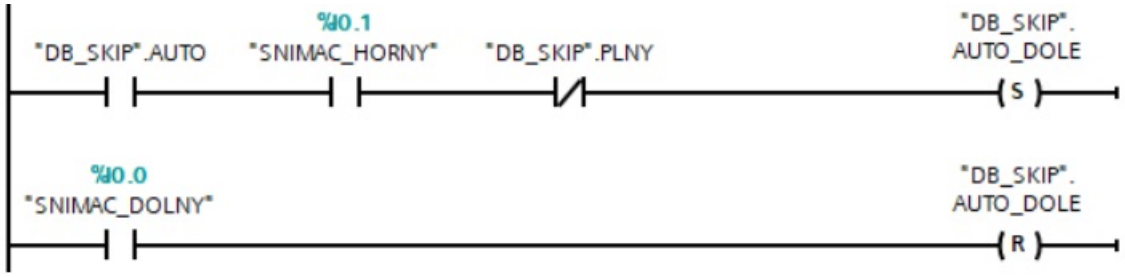
Obr. 4.138. Riadenie chodu hore v automatickom režime

Ďalšia časť programu je súčasťou úlohy 3. V hornej polohe sa dávkuje kamenivo do miešačky. Dávkovanie je riadené časom (Obr. 4.139). Ak je skip v automatickom režime, je v hornej polohe a uplynie 6 sekúnd, vynulujeme plnosť skipu a nastavíme pomocný stav miešačky na TRUE.



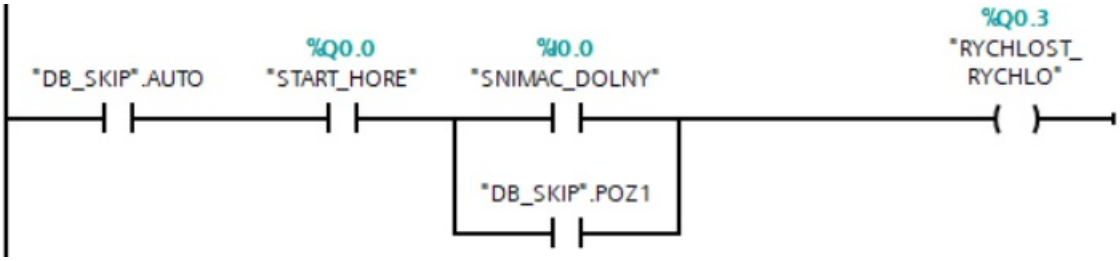
Obr. 4.139. Vysypanie kameniva

Využijeme vyššie uvedenú zmenu plnosti skipu na riešenie úlohy 3 - automatický chod dole (Obr. 4.140). Ak je skip v automatickom režime, je v hornej polohe a už nie je plný, zapne sa pohyb dole. Po dosiahnutí dolnej polohy sa pohyb v automatickom režime vypne.

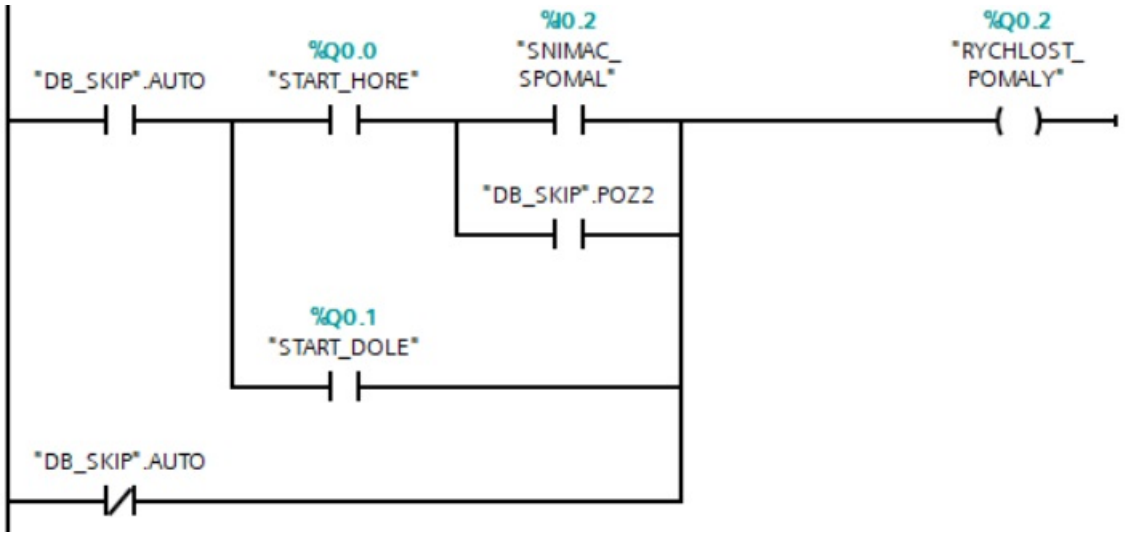


Obr. 4.140. Riadenie chodu dole v automatickom režime

V úlohe 4 je cieľom meniť rýchlosť skipu v závislosti od polohy. Ak sa skip rozbieha smerom hore v automatickom režime z dolnej polohy a nedosiahne spomaľovaciu polohu, má sa pohybovať vyššou rýchlosťou ((Obr. 4.141). Vo zvyšných stavoch sa má pohybovať pomaly (Obr. 4.142). V automatickom režime sa skip pohybuje pomaly smerom hore po dosiahnutí spomaľovacej pozície a v medzipolohe POZ2. Rovnako pri pohybe dole v automatickom režime sa pohybuje pomaly. Nakoniec v manuálnom režime nezávisle od smeru pohybu sa pohybuje vždy pomaly (posledná OR vetva podmienky).



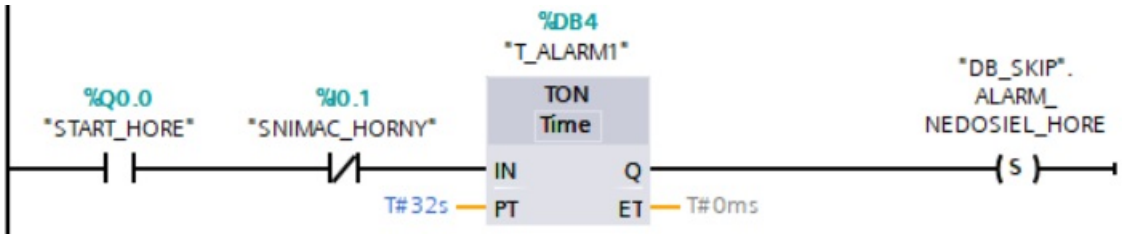
Obr. 4.141. Riadenie rýchlosti - rýchlo



Obr. 4.142. Riadenie rýchlosti - pomaly

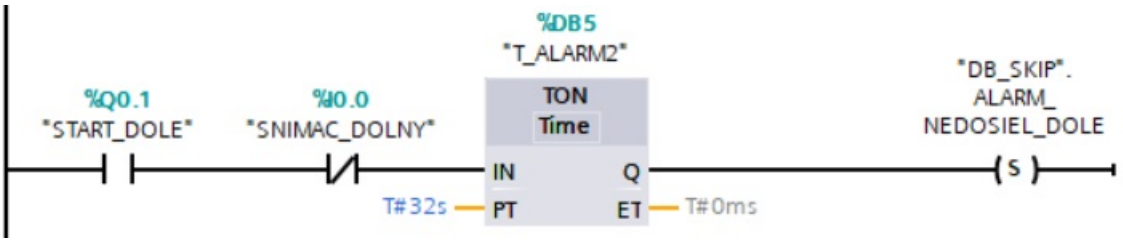
Poslednou neriešenou úlohou sú alarmy. Vytvorme alarm, ktorý sa aktivuje pri pohybe hore z dolnej polohy po hornú. Pri reálnom zariadení je potrebné čas pohybu odmerať

napr. pri plnom zaťažení skipu a pridať malú rezervu. Nech je tento čas v našom príklade 32 sekúnd. Ak je aktívny pohyb hore a nie je dosiahnutý (aktivovaný) horný snímač, počíta sa čas a po uplynutí času 32 s sa nastaví alarm na TRUE (Obr. 4.143). Využívame inštrukciu *SET*, aby bolo nutné daný alarm potvrdiť (z vizualizácie a/alebo fyzickým tlačidlom). Vytvorený alarm by mohol blokovať pohyb skipu.



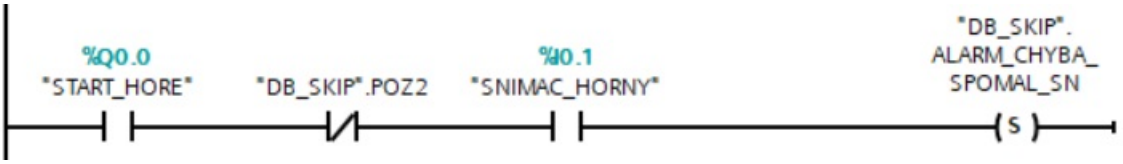
Obr. 4.143. Prvý alarm

Analogicky pre pohyb dole (Obr. 4.144).



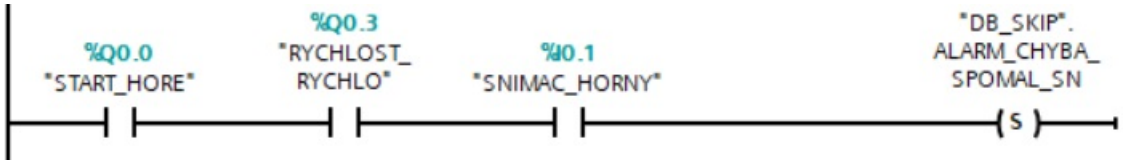
Obr. 4.144. Druhý alarm

Posledným príkladom alarmu je chyba spomaľovacieho snímača (Obr. 4.145). Ak sa skip pohybuje smerom hore, dosiahol hornú polohu, ale nebol zopnutý stav POZ2, znamená to, že spomaľovací snímač nedetegoval skip. Uvedený fragment kódu je nutné umiestniť pred reset POZ1 a POZ2 od snímačov, vrátane horného. Mohlo by sa stať, že POZ2 mala stav TRUE, ale dosiahnutím hornej polohy sa premenná vynulovala a generovali by sme falošný signál.



Obr. 4.145. Tretí alarm - prvá verzia

Alternatívne riešenie je na Obr. 4.146. Namiesto POZ2 využijeme rýchlosť pohybu (s inštrukciou priameho kontaktu). Predložený fragment kódu sa nemusí umiestňovať pred nulovanie pozícií. Na záver uvádzame zoznam premenných v dátovom bloku DB_SKIP (Obr. 4.147).



Obr. 4.146. Tretí alarm - druhá verzia

DB_SKIP

		Name	Data type	Start value
1		Static		
2		AUTO	Bool	false
3		AUTO_HORE	Bool	false
4		AUTO_DOLE	Bool	false
5		MAN_HORE	Bool	false
6		MAN_DOLE	Bool	false
7		POZ1	Bool	false
8		POZ2	Bool	false
9		PLNY	Bool	false
10		ALARM_NEDOSIEL_HORE	Bool	false
11		ALARM_NEDOSIEL_DOLE	Bool	false
12		ALARM_CHYBA_SPOMAL_SN	Bool	false

Obr. 4.147. Zoznam premenných v dátovom bloku DB_SKIP

4.3.17 Príklad 17 - Triedenie objektov podľa farby

Majme jeden dopravník D1, ktorý sa pohybuje len vpred. Dopravník je ovládaný výstupom D1_VPRED. Na začiatku dopravníka sa nachádza optický snímač OS1_Vstup. Detegovaním objektu sa spustí dopravník a objekt bude prepravený k optickému snímaču OS2_Detekcia. Na tomto mieste sa nachádza aj analógový snímač pre detekciu farby prepravovaných objektov. Neskôr pri dopravníku sú umiestnené 3 optické snímače OS3_Farba1, OS4_Farba2 a OS5_Farba3. Za každým snímačom sú umiestnené piesty P1 až P3 ovládané výstupmi Piest1_Farba1 až Piest3_Farba3. Piesty slúžia na presun objektov z dopravníka do zásobníka, t. j. ich úlohou bude triediť objekty podľa detegovanej farby. Piesty sa ovládajú len vpred, späť do pôvodnej polohy sa dostanú nastavením výstupu na FALSE. Pneumaticky ovládané piesty potrebujú v prípade požiadavky realizácie pohybu zapnúť kompresor. Všetky optické snímače sú zapojené ako rozpínacie kontakty. Na dopravníku môže byť viacej objektov (s vhodným rozostupom).

Úloha

Navrhnite stavy a podľa stavov ovládanie dopravníka, piestov a kompresora tak, aby sa naložený objekt prepravil pred príslušný piest a tam bol odstránený z dopravníka. Pri poslednom kuse dopravník zastane, aby nešiel naprázdno. Analógové hodnoty pre 3 farby si určte v intervaloch.

Na Obr. 4.148 je zoznam vstupov a výstupov.

PLC tags

	Name	Tag table	Data type	Address
1	OS1_Vstup	Default tag table	Bool	%I0.0
2	OS2_Farba1	Default tag table	Bool	%I0.1
3	OS2_Farba2	Default tag table	Bool	%I0.2
4	OS2_Farba3	Default tag table	Bool	%I0.3
5	Dopravnik	Default tag table	Bool	%Q0.0
6	Piest1_Farba1	Default tag table	Bool	%Q0.1
7	Piest2_Farba2	Default tag table	Bool	%Q0.2
8	Piest3_Farba3	Default tag table	Bool	%Q0.3
9	Kompresor	Default tag table	Bool	%Q0.4

Obr. 4.148. Zoznam vstupov a výstupov

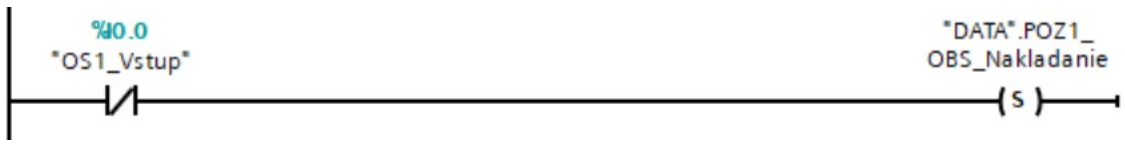
Začneme návrhom stavov. Pre každý snímač vytvoríme obsadenosť a od pozície, na ktorej sa deteguje farba aj premenné dátového typu INT, do ktorých uložíme typ (farbu) objektu. Opäť sa budeme snažiť nepoužívať detekciu hrán. Zoznam vytvorených premenných je na Obr. 4.149.

DATA

	Name	Data type	Start value
1	Static		
2	POZ1_OBS_Nakladanie	Bool	false
3	POZ2_OBS_Detekcia	Bool	false
4	POZ3_OBS_Farba1	Bool	false
5	POZ4_OBS_Farba2	Bool	false
6	POZ5_OBS_Farba3	Bool	false
7	POZ2_DETEKCIA	Int	0
8	POZ3_FARBA1	Int	0
9	POZ4_FARBA2	Int	0
10	POZ5_FARBA3	Int	0

Obr. 4.149. Stavy riadenia

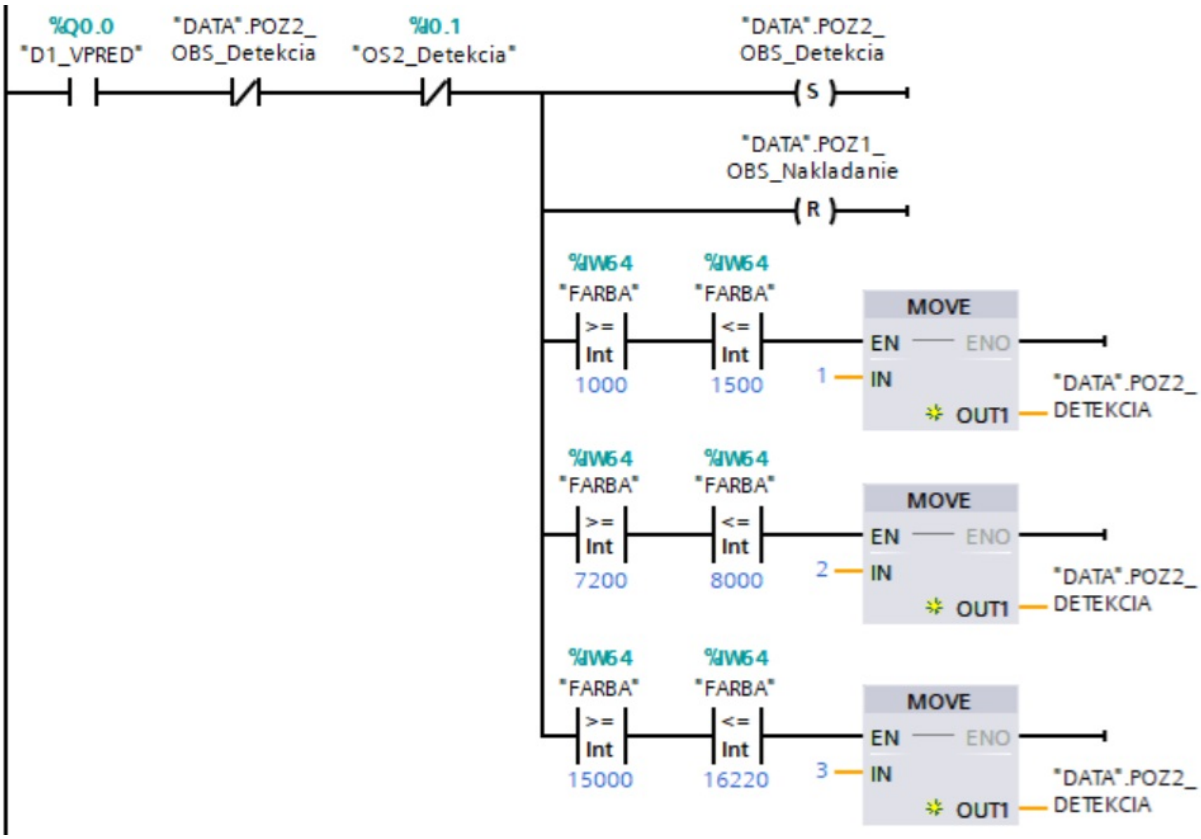
Obsadenosť prvej pozície je na Obr. 4.150.



Obr. 4.150. Implementácia obsadenosti prvej pozície

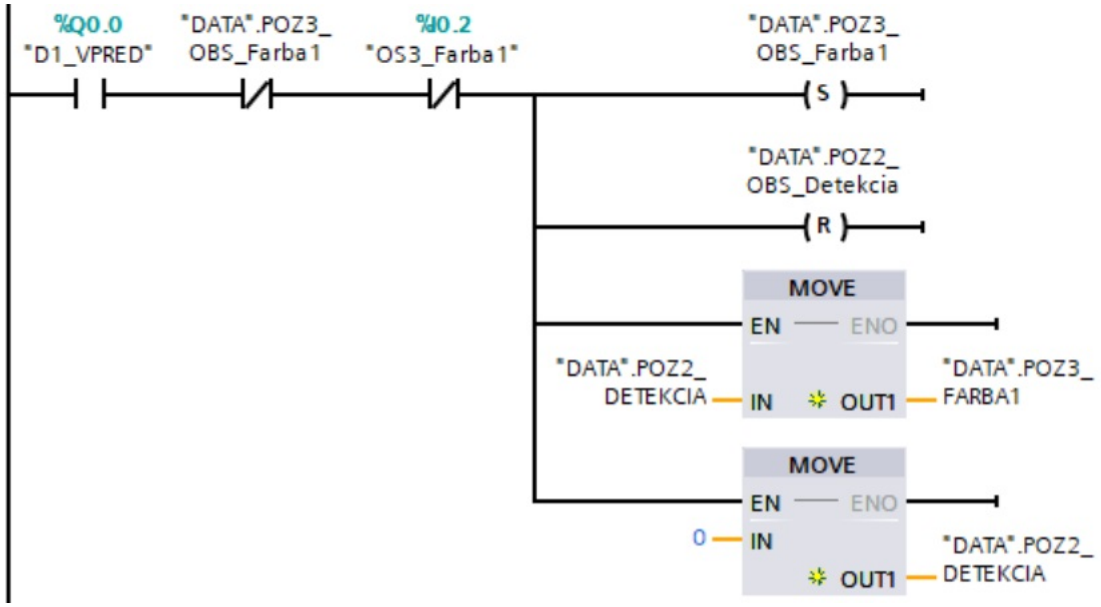
Obsadenosť druhej pozície v mieste detekcie farby je na Obr. 4.151. Ak je dopravník v pohybe, druhá pozícia nie je obsadená a objekt je detegovaný snímačom na druhej pozícii, nastaví sa obsadenosť druhej pozície a vynuluje sa obsadenosť prvej pozície. Zároveň sa podľa hodnoty z analógového vstupu FARBA priradí hodnota 1, 2 alebo 3 do premennej, ktorá reprezentuje typ (farbu) prepravovaného objektu. Interval je vymyslený, ide skôr

o príklad intervalu, ktorý zohľadňuje svetelné podmienky. Hodnota analógového vstupu (farby) je v rozsahu 0 až 27648. Príkazy *SET*, *RESET* a *MOVE* sa vykonajú raz.



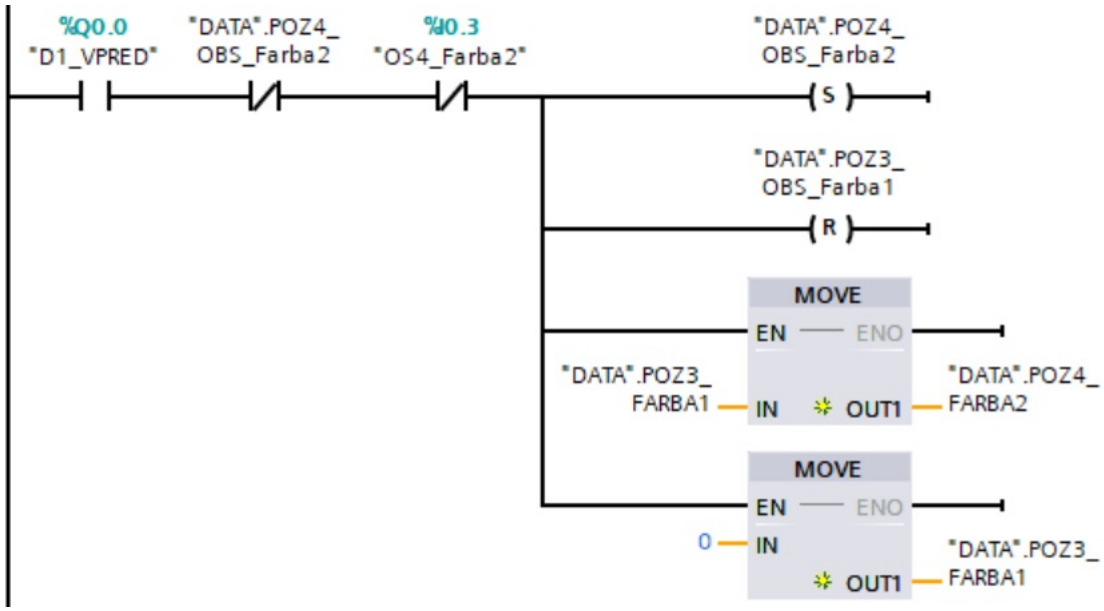
Obr. 4.151. Implementácia obsadenosti druhej pozície

Na Obr. 4.152 je analogické riešenie medzi pozíciami 2 a 3. Inštrukcie *MOVE* presúvajú typ objektu z pozície 2 na pozíciu 3. Prvé volanie inštrukcie *MOVE* skopíruje hodnotu a druhé volanie pod ním vynuluje predchádzajúcu pozíciu.

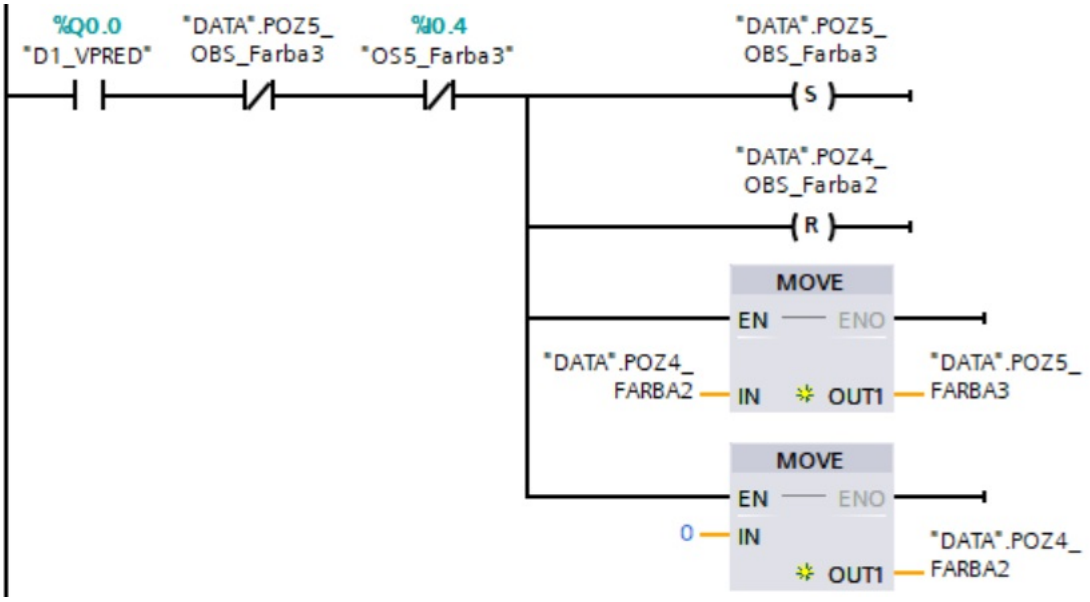


Obr. 4.152. Implementácia obsadenosti tretej pozície

Programy pre zvyšné pozície sú na Obr. 4.153 a 4.154.



Obr. 4.153. Implementácia obsadenosti štvrtej pozície



Obr. 4.154. Implementácia obsadenosti piatej pozície

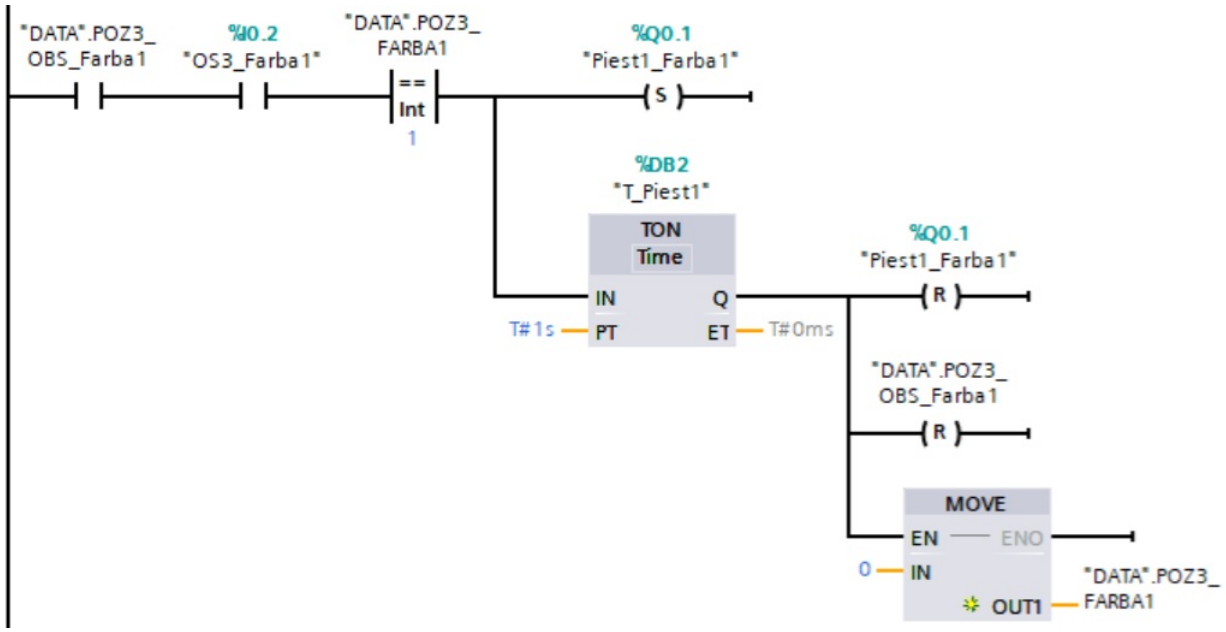
Dopravník sa spustí ak je obsadená aspoň jedna z pozícií (Obr. 4.155).

Obr. 4.155. Implementácia ovládania dopravníka

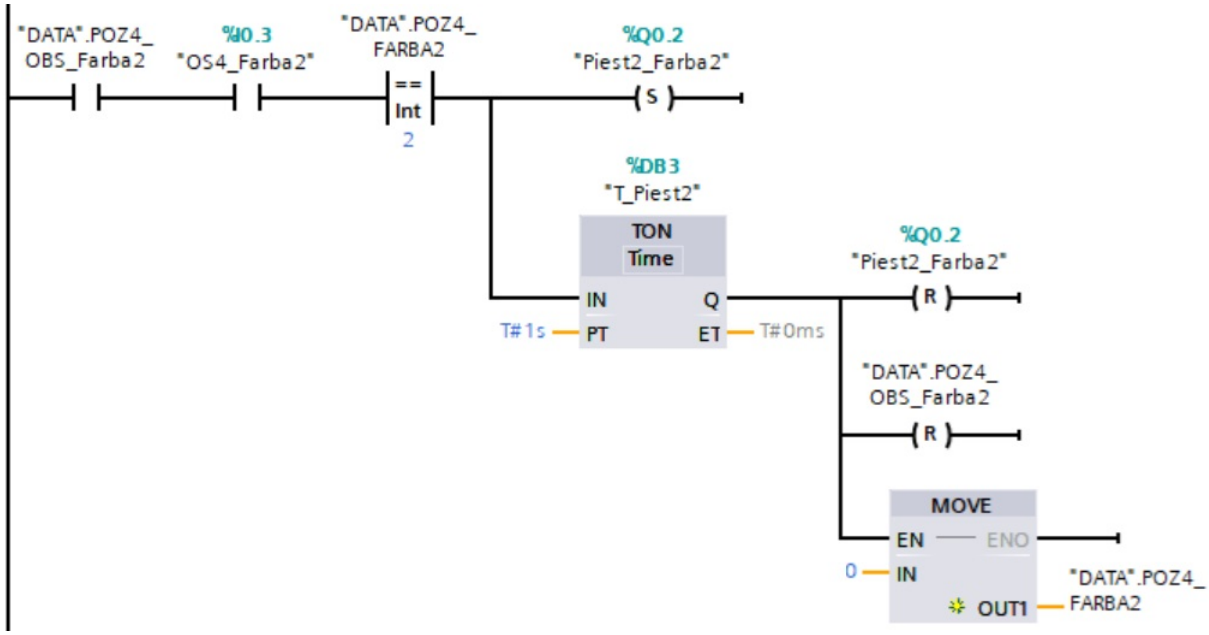
Kompressor sa zapne, ak je požiadavka pohybu aspoň jedného piestu (Obr. 4.156).

Obr. 4.156. Implementácia ovládania dopravníka

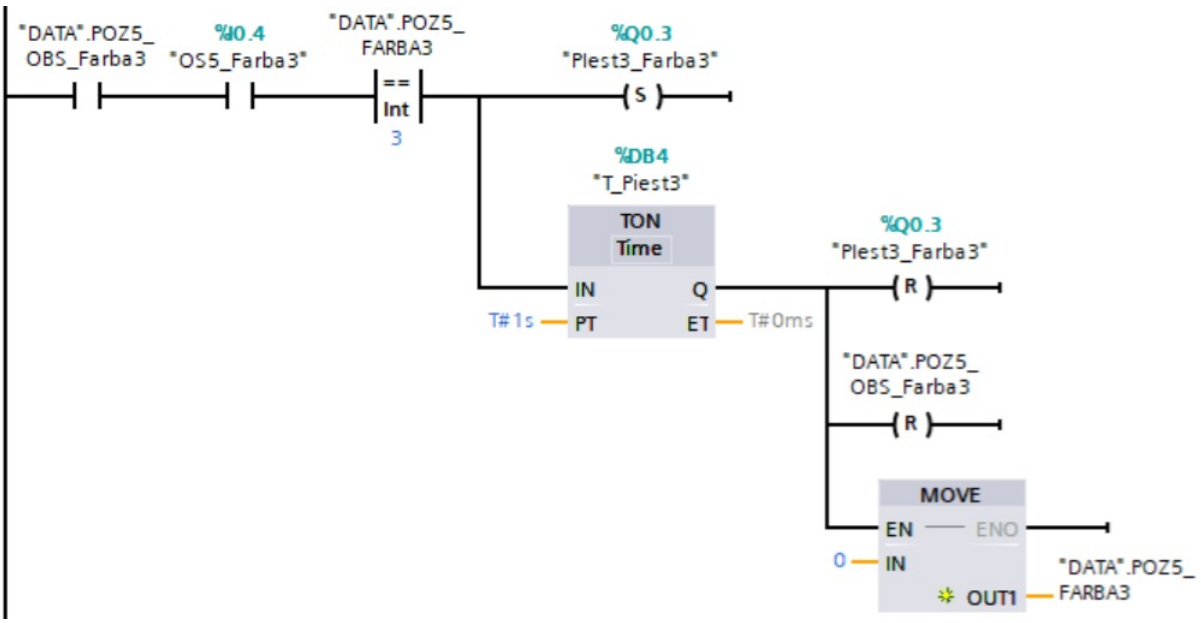
Ovládania piestov sú na Obr. 4.157 až 4.159. Ak je obsadená príslušná pozícia a snímač je voľný, znamená to, že objekt je prepravovaný na dopravníku a už nie je detegovaný optickým snímačom. V prípade správneho typu sa môže odstrániť z dopravníka. Typ objektov, ktoré majú pridelenú hodnotu, 1 sa vysúvajú z dopravníka pri optickom snímači farby 1. Typ objektov, ktoré majú pridelenú hodnotu 2, sa vysúvajú z dopravníka pri optickom snímači farby 2. A nakoniec typ objektov, ktoré majú pridelenú hodnotu 3, sa vysúvajú z dopravníka pri optickom snímači farby 3. Pri platnej podmienke sa nastaví výstup príslušného piestu na TRUE, čím sa zapne kompresor (pozri Obr. 4.156). Zároveň sa spustí časovanie na 1s. Po uplynutí času sa vypne pohyb piestu vpred, zruší sa obsadenosť a vynuluje údaj o type objektu.



Obr. 4.157. Implementácia ovládania piestu 1



Obr. 4.158. Implementácia ovládania piestu 2



Obr. 4.159. Implementácia ovládania piestu 3

4.3.18 Príklad 18 - Regálový zakladač

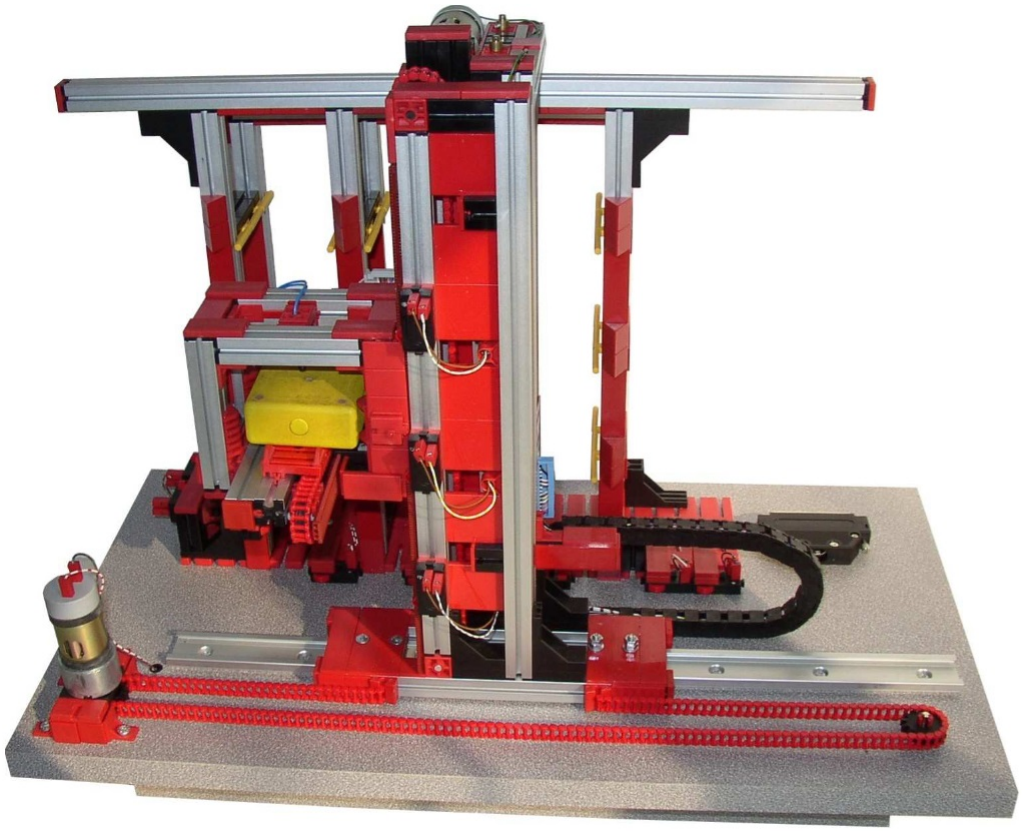
Majme model regálového zakladača na Obr. 4.160, ktorého vstupy a výstupy sú zobrazené na Obr. 4.161. Os x sa pohybuje vpravo a vľavo. Jeho poloha pred regálom je určená pomocou troch kontaktných snímačov. Poloha určená snímačom x1 je vľavo, snímačom x2 je v strede a snímačom x3 je vpravo, t. j. regál obsahuje 3 stĺpce. Os z sa pohybuje vpred (smer k regálu) a vzad. Snímač z3 definuje polohu vpred (v regále), snímač z2 stred - prepravná pozícia (práve vidno na Obr. 4.160) a z3 vzad - nakladacia. Os y sa

ovláda dvoma výstupmi smerom hore a dole. Polohy zakladača určujú snímače y1 až y6. Regál obsahuje 3 riadky (3 výšky) na ktoré môžeme ukladať objekty. Snímače y1 a y2 definujú polohu nad a pod prvým (spodným) miestom. Snímače y3 a y4 definujú polohu nad a pod druhým (stredným) miestom. Snímače y5 a y6 definujú polohu nad a pod tretím (najvyšším) miestom. Na každú polohu potrebujeme 2 snímače z dôvodu nakladania a vyberania objektov. Ak je cieľom vložiť objekt na v spodnom rade, žiadanou pozíciu je y2. Po dosiahnutí pozície sa vysúva os z vpred. Po dosiahnutí polohy vpred sa os y posunie na nižšiu polohu y1, čím sa objekt položil do zakladača. Na záver sa os z posunie do strednej polohy. Výber objektov je opačný. Najprv sa os y nastaví do nižšej polohy (y1, y3 alebo y5) podľa požadovanej výšky. Vysunie sa os z vpred. Po dosiahnutí polohy vpred sa os y posunie do vyššej polohy (y2, y4 alebo y6) čím sa nadvihne objekt z ukladacej pozície v regáli. Na záver sa os z presunie do strednej polohy.

Úloha

Vytvorte program pre vkladanie a vyberanie objektov pomocou zakladača, ktorý bude zahŕňať:

- Ovládanie osí na základe žiadaných hodnôt.
- Vkladanie a vyberanie objektov pomocou krokového riadenia, ktoré bude využívať nastavovanie žiadaných hodnôt pre jednotlivé osi.



Obr. 4.160. Obrázok modelu

Default tag table

	Name	Data type	Address
1	x1	Bool	%I0.0
2	x2	Bool	%I0.1
3	x3	Bool	%I0.2
4	y1	Bool	%I0.3
5	y2	Bool	%I0.4
6	y3	Bool	%I0.5
7	y4	Bool	%I0.6
8	y5	Bool	%I0.7
9	y6	Bool	%I1.0
10	z1	Bool	%I1.1
11	z2	Bool	%I1.2
12	z3	Bool	%I1.3
13	x_vpravo	Bool	%Q0.0
14	x_vlavo	Bool	%Q0.1
15	y_hore	Bool	%Q0.2
16	y_dole	Bool	%Q0.3
17	z_vpred	Bool	%Q0.4
18	z_vzad	Bool	%Q0.5

Obr. 4.161. Zoznam vstupov a výstupov modelu

Ako prvé si vytvorme premenné, ktoré budeme potrebovať pri riadení modelu zakladača. Potrebujeme aktuálne polohy osí, ktoré sa budú meniť v závislosti od snímačov, žiadané polohy, krok riadenia nakladania alebo vykladania dátového typu INT a bitové premenné, ktoré spustia nakladanie, resp. vykladanie. Premenné týkajúce sa jednotlivých osí sme umiestnili do dátových blokov OS_X, OS_Y a OS_Z. Uvedené premenné sú na Obr. 4.162 až 4.164. Žiadané polohy a aktuálne polohy sme zvolili ako dátové typy Real, aby sme mohli implementovať aj medzipolohy. Všimnite si počiatočné (inicializačné) hodnoty žiadaných polôh. Ak by sme ponechali nulové, zakladač by mal snahu dosiahnuť tieto polohy, ktoré sú mimo regál. Premenná na_snimaci bude v stave TRUE, ak je aktivovaný jeden z príslušných snímačov. Slúži na zjednodušenie detekcie hrany ak os opustí jeden zo snímačov. Ako pomocnú premennú na detekciu hrany sme vytvorili na_snimaci_sb. V praxi často môže dôjsť k pnutiu lán, reťazí a pod., preto sme navrhli dve premenné, ktoré budú indikovať posledný smer pohybu. V dátovom bloku DATA sú umiestnené riadiace premenné ako krok pre krokové riadenie, žiadané polohy x a y a požiadavky začatia sekvencie. Žiadané polohy budú v rozsahu 1 až 3, t. j. uvažujeme s mriežkou 3 x 3, kde ľavá dolná poloha je poloha [1,1].

OS_X

		Name	Data type	Start value
1		Static		
2		ziadana	Real	1.0
3		aktualna	Real	0.0
4		na_snimaci	Bool	false
5		na_snimaci_sb	Bool	false
6		bol_pohyb_vpravo	Bool	false
7		bol_pohyb_vlavo	Bool	false

Obr. 4.162. Zoznam vytvorených premenných pre os X

OS_Y

		Name	Data type	Start value
1		Static		
2		ziadana	Real	1.0
3		aktualna	Real	0.0
4		na_snimaci	Bool	false
5		na_snimaci_sb	Bool	false
6		bol_pohyb_hore	Bool	false
7		bol_pohyb_dole	Bool	false

Obr. 4.163. Zoznam vytvorených premenných pre os Y

OS_Z

		Name	Data type	Start value
1		Static		
2		ziadana	Real	1.0
3		aktualna	Real	0.0
4		na_snimaci	Bool	false
5		na_snimaci_sb	Bool	false
6		bol_pohyb_vpred	Bool	false
7		bol_pohyb_vzad	Bool	false

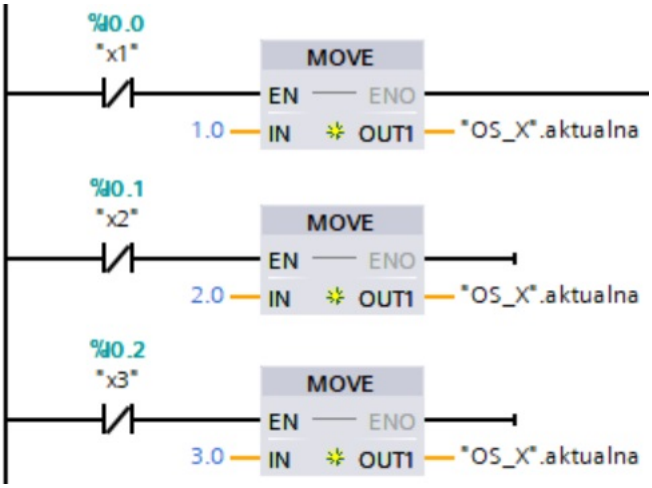
Obr. 4.164. Zoznam vytvorených premenných pre os Z

DATA

		Name	Data type	Start value
1		Static		
2		KROK	Int	0
3		x	Real	1.0
4		y	Real	1.0
5		Poziadavka_naloz	Bool	false
6		Poziadavka_vyloz	Bool	false
7		Poziadavka_naloz_sb	Bool	false
8		Poziadavka_vyloz_sb	Bool	false

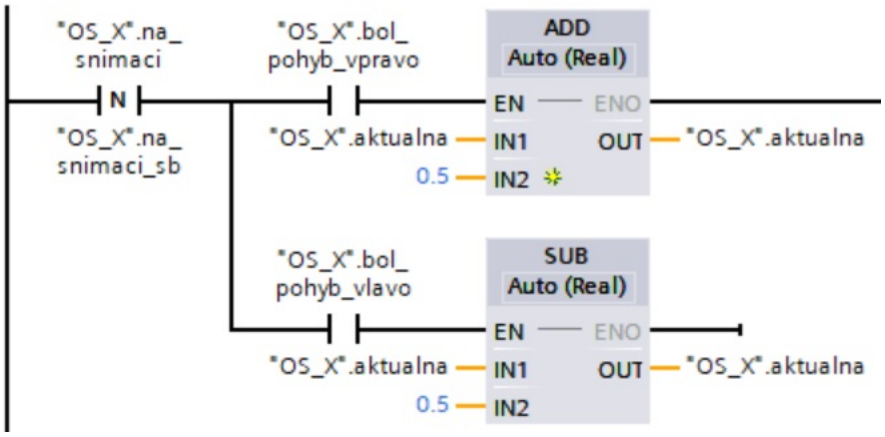
Obr. 4.165. Riadiace premenné

Na Obr. 4.166 je implementácia polohy v číselnej podobe v závislosti od snímačov. Ak je aktivovaný prvý snímač x1, do premennej OS_X.aktualna sa uloží hodnota 1,0 atď. Na Obr. 4.167 je navrhnutý program, ktorý zapisuje do premennej OS_X.na_snimaci stav TRUE pri aktivácii jedného zo snímačov. Tento signál využijeme v programe na Obr. 4.168. Ak os opustí jeden zo snímačov (je detegovaná dobežná hrana) a bol pohyb v kladnom alebo zápornom smere (vpravo alebo vľavo), potom upravíme aktuálnu polohu o 0,5. Ak bol zopnutý napr. snímač x1, mali sme polohu 1,0. Pohybom vpravo a už nedetekovaním polohy snímačom x1 sa poloha zmení na 1,5 pridaním 0,5. Podobne ak by sme mali napr. polohu 2,0 určenú snímačom x2, potom pohybom vľavo a nedetekovaním polohy x2 sa od 2,0 odčíta hodnota 0,5, čím dostávame hodnotu 1,5. Čiže poloha 0,5 je vľavo do x1, poloha 1,5 je medzi x1 a x2, poloha 2,5 je medzi x2 a x3 a nakoniec poloha 3,5 je vpravo od x3.



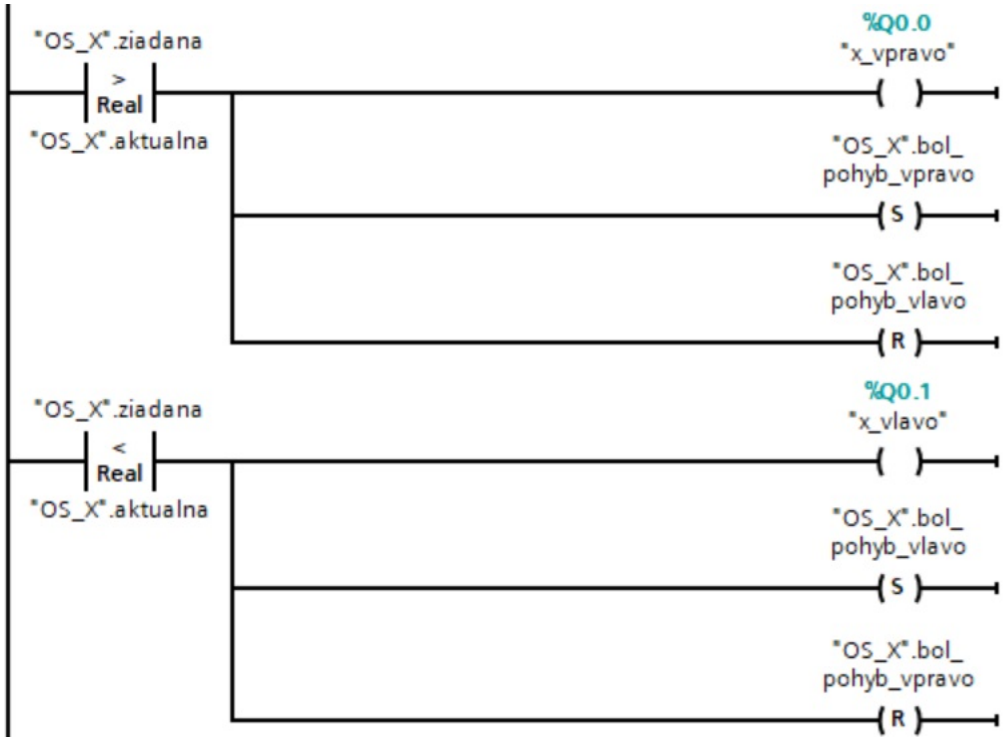
Obr. 4.166. Vytvorenie spätnej väzby

Obr. 4.167. Vytvorenie signálu na snímači



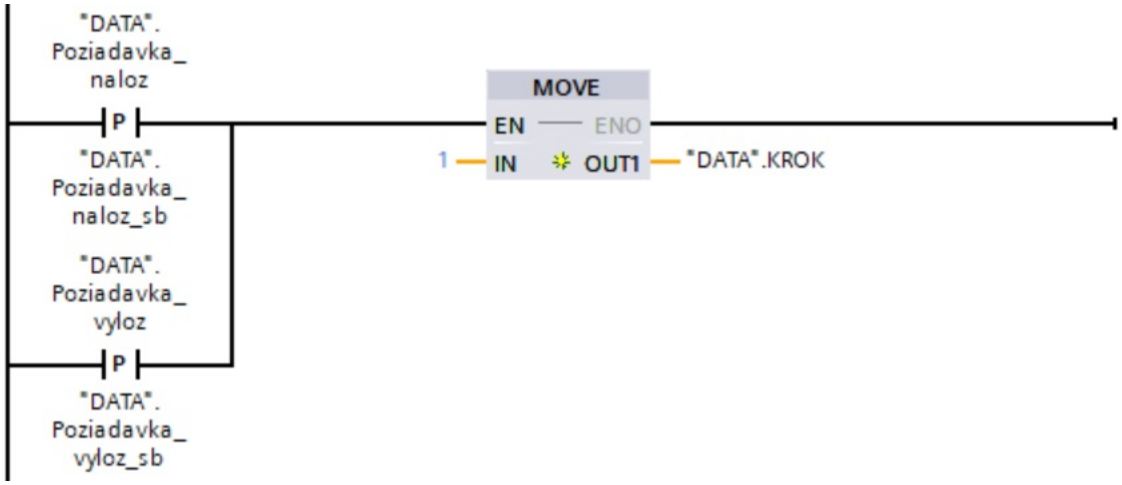
Obr. 4.168. Implementácia medzipolohy

V závere na Obr. 4.169 je príklad ovládania osi X v závislosti od žiadanej a aktuálnej polohy. Ak je je žiadaná poloha väčšia ak aktuálna, os x sa pohne vpravo. Ak je je žiadaná poloha menšia ak aktuálna, os x sa pohne vľavo. Program obsahuje nastavovanie pomocných bitov pre naposledy vykonaný pohyb, ktoré sa využívajú pri určení medzipolôh.



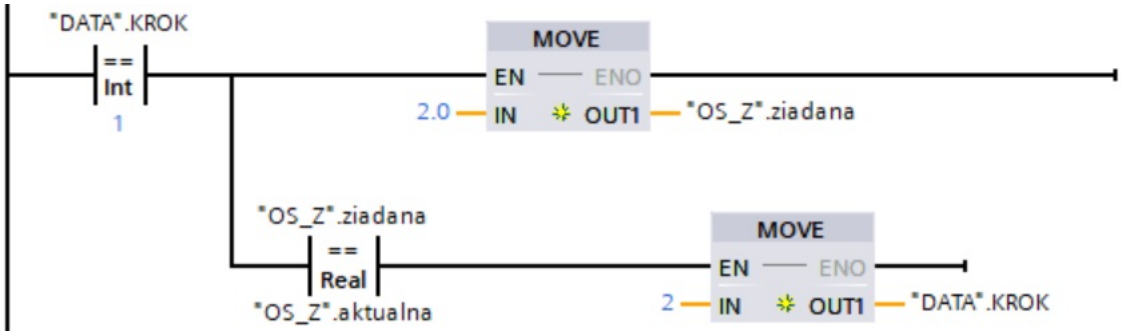
Obr. 4.169. Riadenie osi X

Analogicky je vytvorený program pre osi x a y s tým, že pre os Y je viac snímačov, ktoré určujú polohu 1,0 až 6,0. Program pre krokové riadenie je na Obr. 4.170 až 4.171. Na Obr. 4.170 sa aktiváciou jednou z požiadaviek dostávame do kroku jedna.



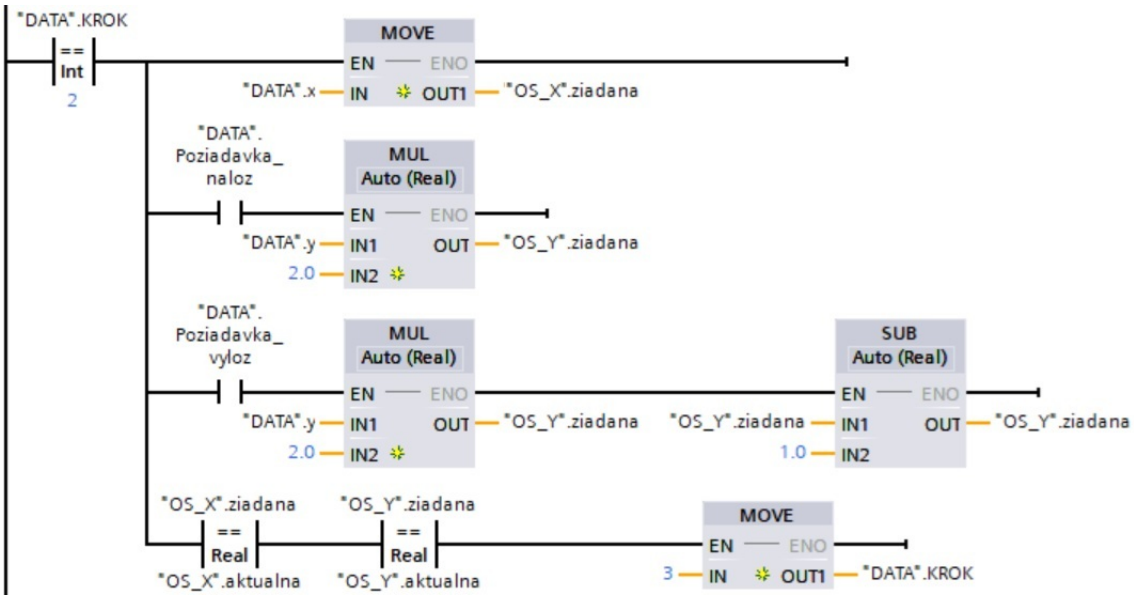
Obr. 4.170. Prechod do krokového riadenia

V kroku 1 (Obr. 4.171) presúvame os z do strednej polohy nastavením žiadanej polohy na 2,0. Po dosiahnutí želanej polohy sa presúvame do kroku 2.



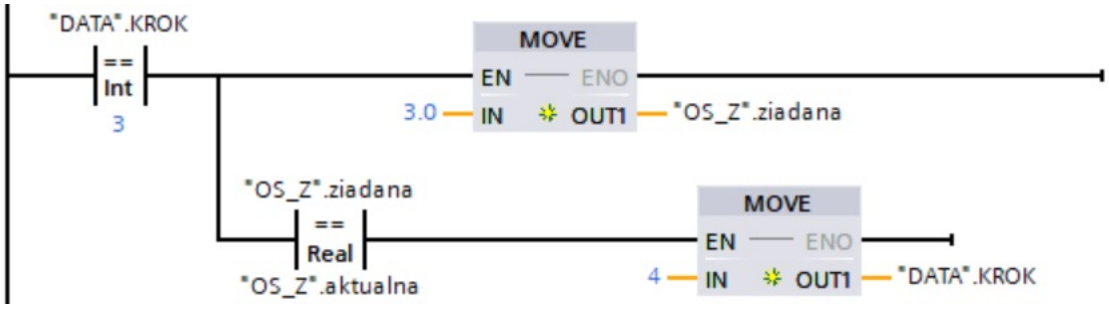
Obr. 4.171. Krok 1 - Os Z do strednej polohy

V kroku 2 (Obr. 4.172) sa os x posúva do želanej polohy a os y podľa nakladania alebo vykladania na vyššiu alebo nižšiu polohu. Ak je požiadavka vložiť na polohy 1, 2 alebo 3, želané polohy pre riadenie osi majú byť 2, 4 a 6. Pre vyloženie sú to želané polohy 1, 3 a 5. Po dosiahnutí želaných polôh osí sa presúvame do kroku 3.



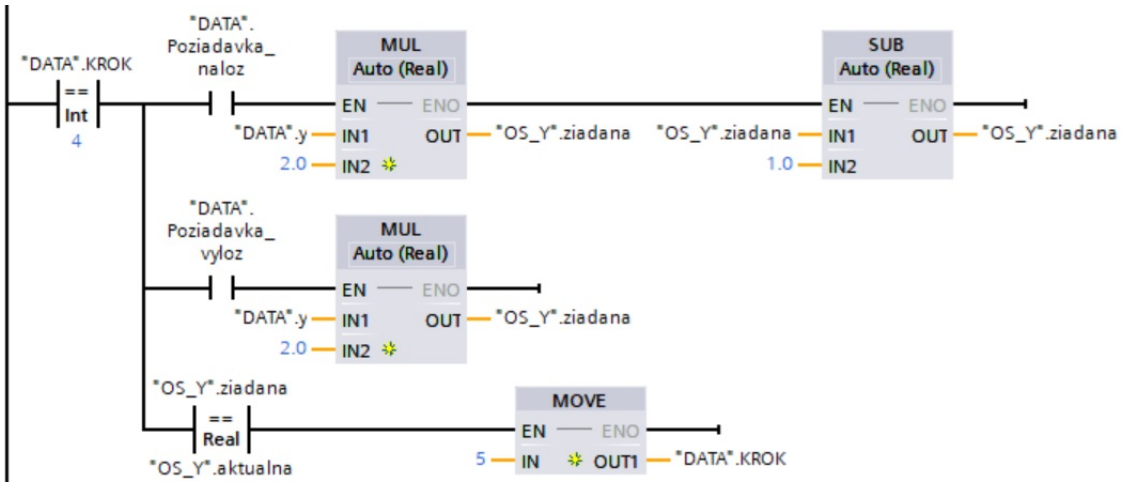
Obr. 4.172. Krok 2 - Os X a Y do želaných polôh

V kroku 3 (Obr. 4.173) presúvame os z do polohy vpred nastavením žiadanej polohy na 3.0. Po dosiahnutí želanej polohy sa presúvame do kroku 4.



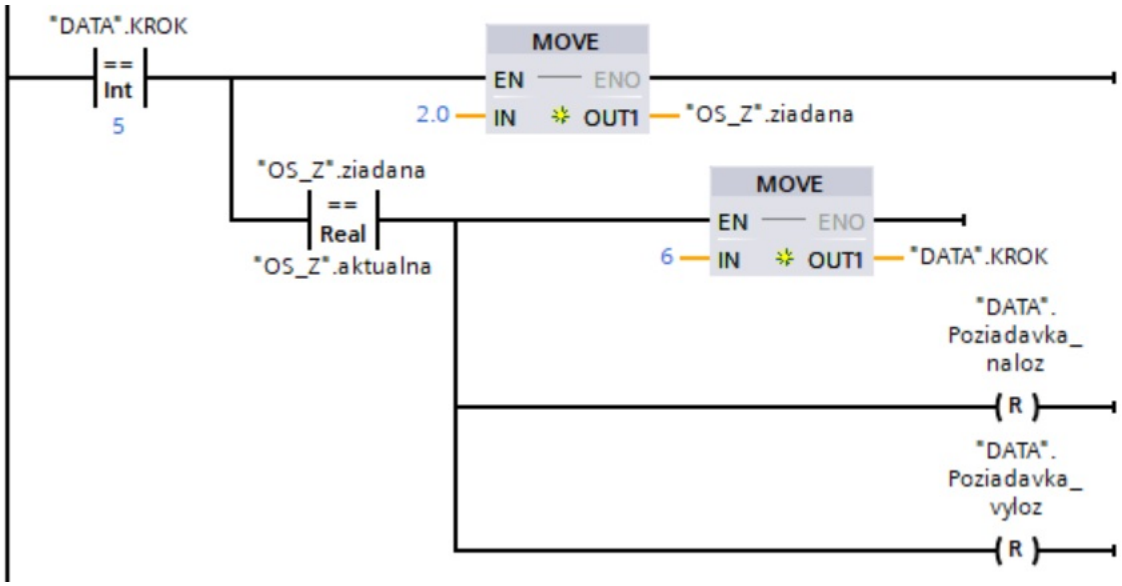
Obr. 4.173. Krok 3 - Os Z do polohy vpred

V kroku 4 (Obr. 4.174) sa os y posúva do opačne polohy ako v kroku 2, t. j. ak bol na vyššej polohe lebo sa objekt má vložiť, teraz sa presunie nižšie a opačne. V danom kroku sme vymenili výpočty pre želané polohy osi y. Po dosiahnutí želanej polohy osi Y sa presúvame do kroku 5.



Obr. 4.174. Krok 4 - Os Y na opačnú polohu

Na záver v kroku 5 (Obr. 4.175) presúvame os z do strednej polohy nastavením žiadanej polohy na 2.0. Po dosiahnutí želanej polohy sa presúvame do kroku 6, ktorý nerealizuje žiadny pohyb. Zároveň sa nulujú požiadavky pohybu. Takto navrhnutý program neošetruje možnosť nastavenia nakladania a vyberania objektu, resp. modifikáciu požiadavky počas vykonávania sekvencie. Uvedený program slúžil ako demonštračný príklad implementácie krokového riadenia, ako aj návrh riadenia len pomocou stavov (žiadaných a aktuálnych polôh).



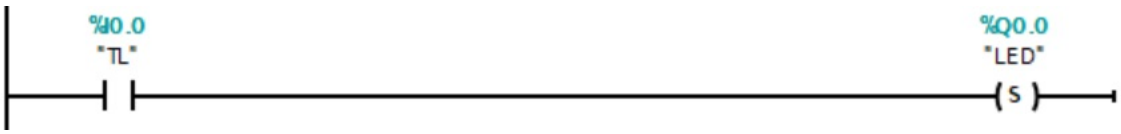
Obr. 4.175. Krok 5 - Os Z do strednej polohy

4.3.19 Príklad 19 - Prvý program v OB30

V nasledujúcich príkladoch uvedieme jednoduché príklady logického riadenia implementovaného v periodickej úlohe OB30 s cieľom ozrejmiť nevýhody takto navrhnutých programov. Tlačidlo TL bude zapojené ako spínací kontakt.

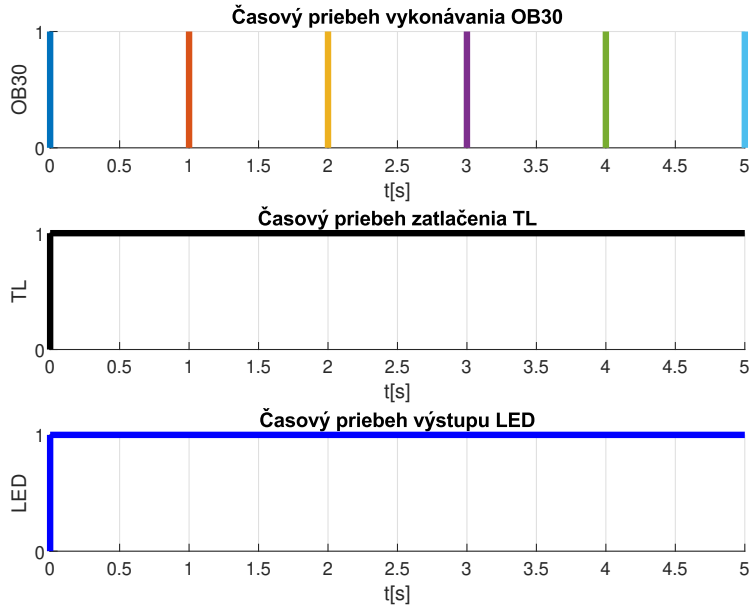
Úloha

Majme program na Obr. 4.176, ktorý sa vykonáva v OB30. Perióda OB30 je 1 sekunda. Nech vykonanie programu trvá veľmi krátko (napr. 1 ms). Analyzujte ako sa bude meniť digitálny výstup LED v závislosti od digitálneho vstupu TL. Vstup TL reprezentuje tlačidlo.

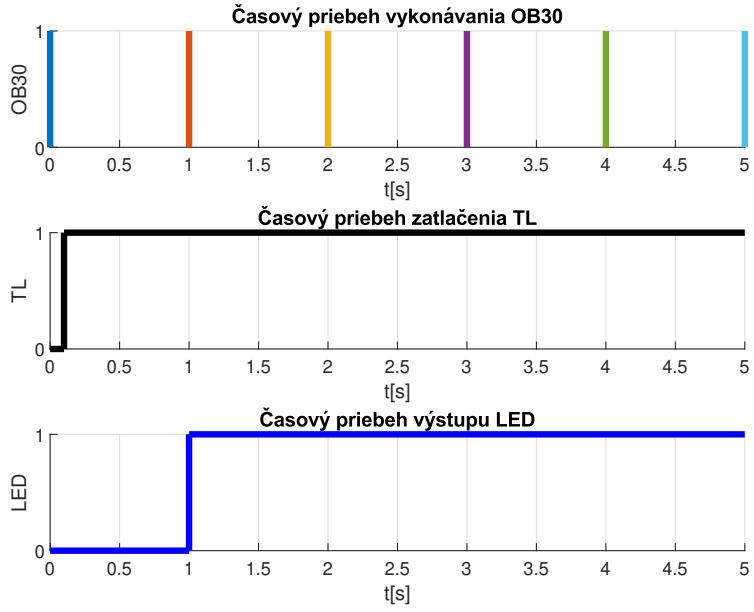


Obr. 4.176. Program v OB30

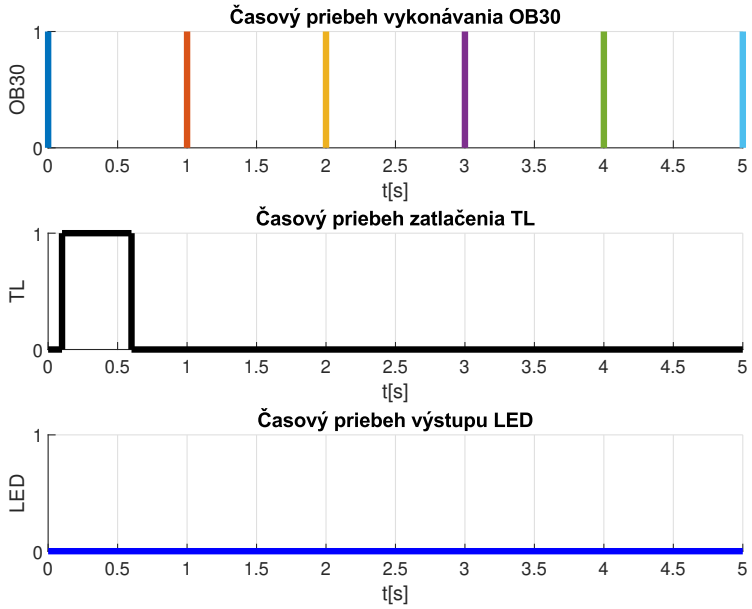
Program má jednoduchú funkcionalitu. Ak sa vykoná a tlačidlo je zatlačené, potom sa LED zapne a ostáva v stave TRUE. Na Obr. 4.177 a 4.178 sú vyznačené volania OB30 v 0 s (v momente prepnutia PLC z režimu STOP do režimu RUN), 1 s atď. Na prvom obrázku je tlačidlo TL zatlačené od 0 s. Program sa prvý raz vykoná v nultej sekunde, t. j. LED sa nastaví na TRUE, preto svieti. Na druhom obrázku je príklad zatlačenia tlačidla v 0,1 sekunde. Program, ktorý sa vykonal v 0. sekunde mal na vstupe TL signál FALSE, preto sa inštrukcia *SET* nevykonala. Program sa druhýkrát vykoná v prvej sekunde, keď je tlačidlo TL zatlačené, preto sa LED nastaví na TRUE. Z uvedeného vyplýva, že na zapnutie LED potrebujeme zatlačiť a držať tlačidlo aspoň 1s, aby sa vykonalo jedno volanie OB30. Príklad krátkeho zatlačenia tlačidla je na Obr. 4.179. Z časových priebehov je zrejmé, že sa počas krátkeho zatlačenia tlačidla TL program nevykonal, a preto sa výstup LED nezapol.



Obr. 4.177. Časové priebehy vykonávania OB30, zatlačenia tlačidla TL a zapnutie výstupu LED



Obr. 4.178. Časové priebehy vykonávania OB30, zatlačenia tlačidla TL a zapnutie výstupu LED

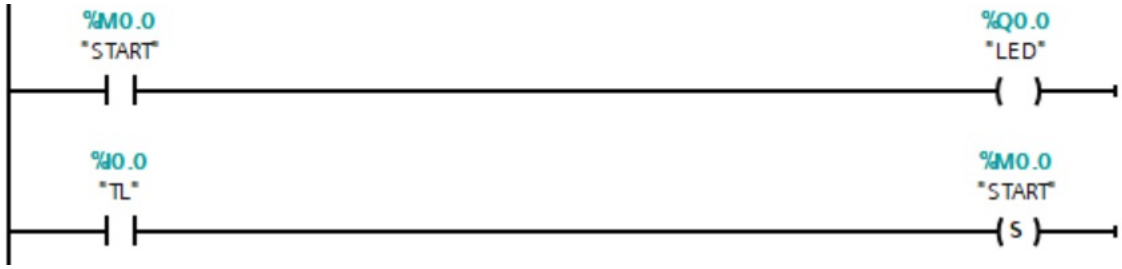


Obr. 4.179. Časové priebehy vykonávania OB30, zatlačenia tlačidla TL a zapnutie výstupu LED

4.3.20 Príklad 20 - Druhý program v OB30

Úloha

Majme podobný program v OB30 ako v predchádzajúcom príklade (Obr. 4.180). Perióda OB30 je 1 sekunda. Nech vykonanie programu trvá veľmi krátko (napr. 1 ms). Analyzujte ako sa bude meniť digitálny výstup LED v závislosti od digitálneho vstupu TL. Vstup TL reprezentuje tlačidlo. Nech má premenná START na začiatku hodnotu FALSE.



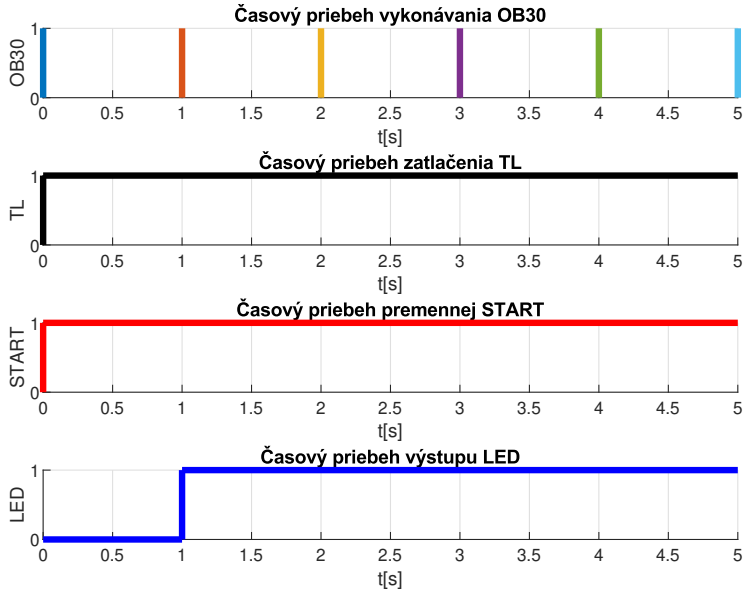
Obr. 4.180. Program v OB30

Ak sa program vykoná a tlačidlo je zatlačené, START sa nastaví na TRUE. V ďalšom cykle (ďalšom vykonaní programu), sa hodnota START = TRUE priradí do LED, t. j. bude svietiť. Ak inštrukcia *SET* nebola vykonaná, v každom cykle sa do LED zapisuje stav FALSE. Na Obr. 4.181 a 4.182 sú vyznačené volania OB30 v 0 s (v momente prepnutia PLC z režimu STOP do režimu RUN), 1 s atď.

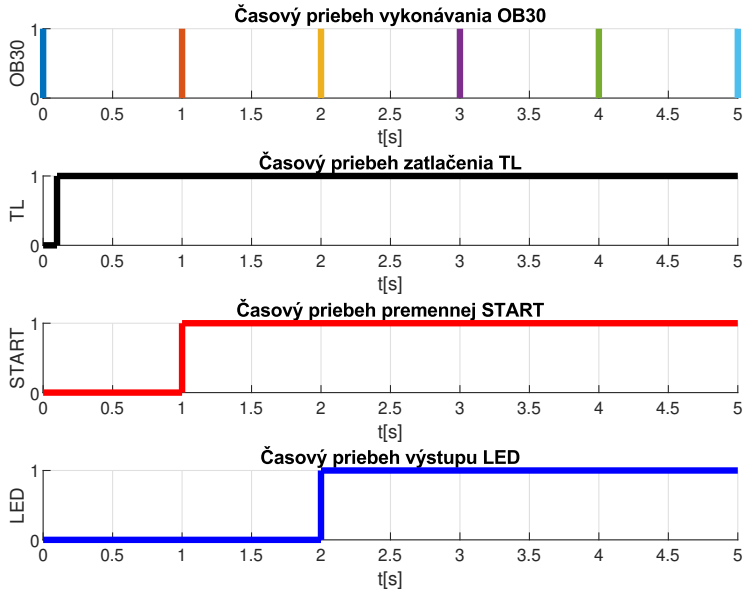
Na prvom obrázku je tlačidlo TL zatlačené od 0s. V prvom volaní OB30 v nultej sekunde sa vykoná inštrukcia *SET*, t. j. START = TRUE. Ďalší cyklus sa vykoná až v 1. sekunde. Program sa vykoná zhora nadol a po rebríkoch zľava doprava, t. j. hodnota START = TRUE sa priradí do LED, preto sa LED zapne v 1. sekunde.

Na druhom obrázku bolo tlačidlo zatlačené v 0,1 sekunde. Program sa druhýkrát vykoná až v 1. sekunde, keď sa START nastaví na TRUE. Až v ďalšom cykle v 2. sekunde sa zapne LED. V oboch príkladoch je potrebné tlačidlo držať aspoň sekundu.

Uvedeným príkadom sme ukázali nevýhodu implementácie diskrétneho riadenia v periodických úlohách (pozn. s veľkou periódou). V danom príklade sa LED zapne od 1 s do 2 s od zatlačenia tlačidla.



Obr. 4.181. Časové priebehy vykonávania OB30, zatlačenia tlačidla TL a zapnutie výstupu LED



Obr. 4.182. Časové priebehy vykonávania OB30, zatlačenia tlačidla TL a zapnutie výstupu LED

4.3.21 Príklad 21 - Tretí program v OB30

Úloha

Majme príklad s časovačom (Obr. 4.183), ktorý sa vykonáva v OB30. Perióda OB30 je 1 sekunda. Nech vykonanie programu trvá veľmi krátko (napr. 1 ms). Analyzujte ako sa bude meniť digitálny výstup LED v závislosti od digitálneho vstupu TL. Vstup TL reprezentuje tlačidlo.

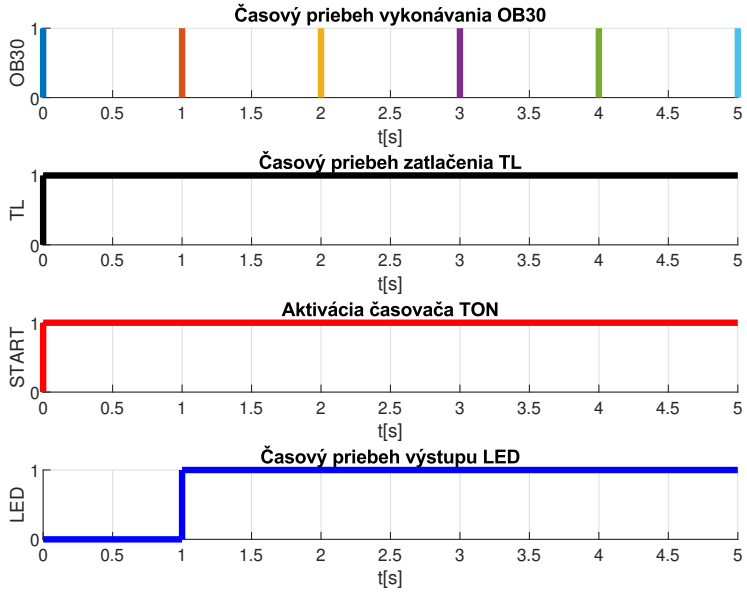
Obr. 4.183. Program v OB30 pre zapnutie LED

Ak by sme mali uvedený program v OB1, t. j. vykonáva sa veľmi rýchlo (napr. každú milisekundu alebo rýchlejšie), potom zatlačením (a držaním) tlačidla sa sa spustí časovač. O 100 ms sa vykoná inštrukcia *SET*.

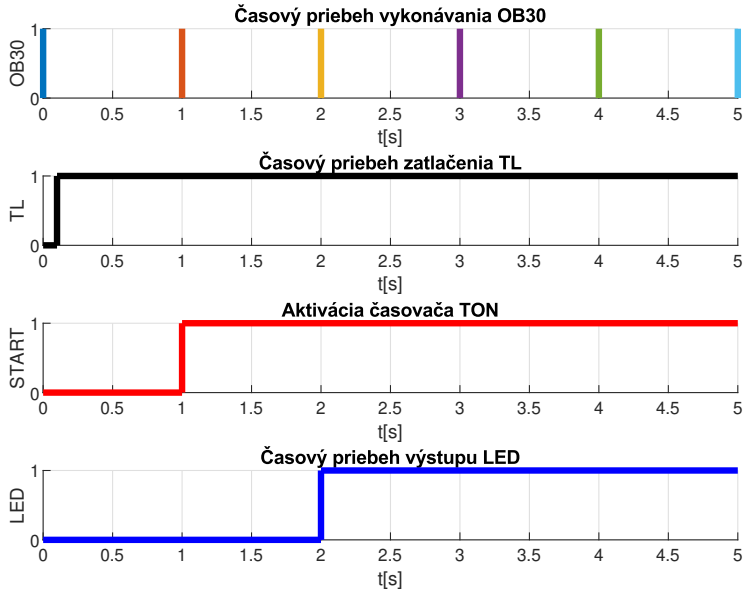
Na Obr. 4.184 a 4.185 sú zaznačené volania OB30 v celých sekundách. V prvom príklade je od začiatku zatlačené tlačidlo, t. j. v cykle, ktoré sa vykonalo v 0. sekunde sa spúšťa časovač *TON*. Druhé vykonanie programu je v prvej sekunde, keď uplynul čas časovača 100 ms, t. j. vykoná sa inštrukcia *SET*.

V druhom príklade bolo tlačidlo zatlačené v 0,1 s. Prvý cyklus, keď je TL zatlačené je v 1. sekunde. V tomto cykle sa spúšťa časovač *TON*. V poradí tretie vykonanie programu je v 2. sekunde, keď skončil časovač (uplynul čas 1 s, ktorý je väčší ako 100 ms), a preto sa výstup LED zapol.

Výsledkom vyššej analýzy je potreba zatlačenia tlačidla na 2 s, aby sa LED s určitosťou nastavil na TRUE. V ideálnom prípade je to 1 s a pri zatlačení tlačidla po vykonaní OB30 sú to 2 s. Z uvedeného vyplýva, že použitie časovačov v periodických úlohách (pozn. s veľkou periódou) nie je vyhovujúce pre precízne úlohy časovania.



Obr. 4.184. Časové priebehy vykonávania OB30, zatlačenia tlačidla TL a zapnutie výstupu LED

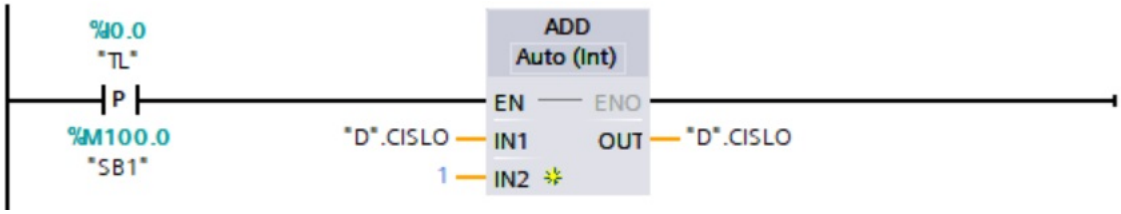


Obr. 4.185. Časové priebehy vykonávania OB30, zatlačenia tlačidla TL a zapnutie výstupu LED

4.3.22 Príklad 22 - Štvrtý program v OB30

Úloha

Majme program na detekciu nábežnej hrany na Obr. 4.186, ktorý sa vykonáva v OB30. Perióda OB30 je 1 sekunda. Nech vykonanie programu trvá veľmi krátko (napr. 1ms). Analyzujte ako sa bude vykonávať inštrukcia *ADD* v závislosti od digitálneho vstupu TL. Vstup TL reprezentuje tlačidlo. Časové priebehy zatlačenia tlačidla sú na Obr. 4.187 a 4.188.

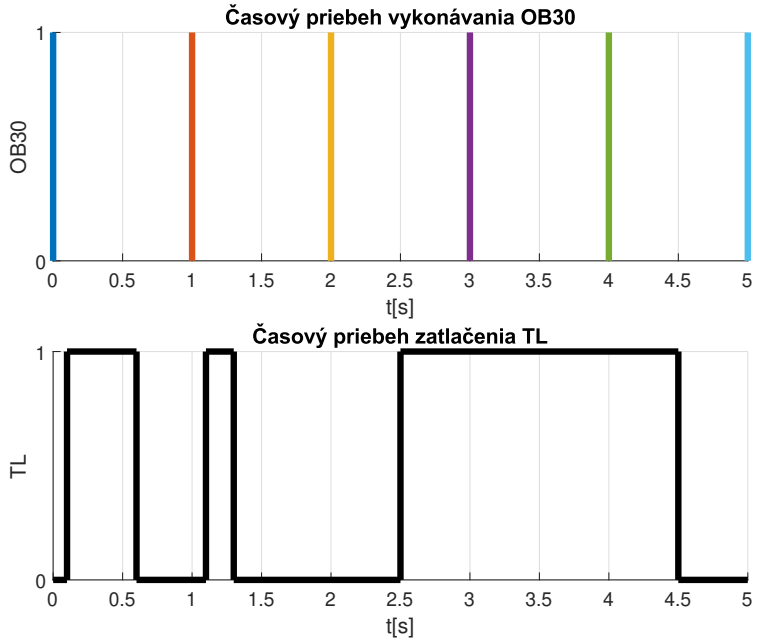


Obr. 4.186. Program v OB30

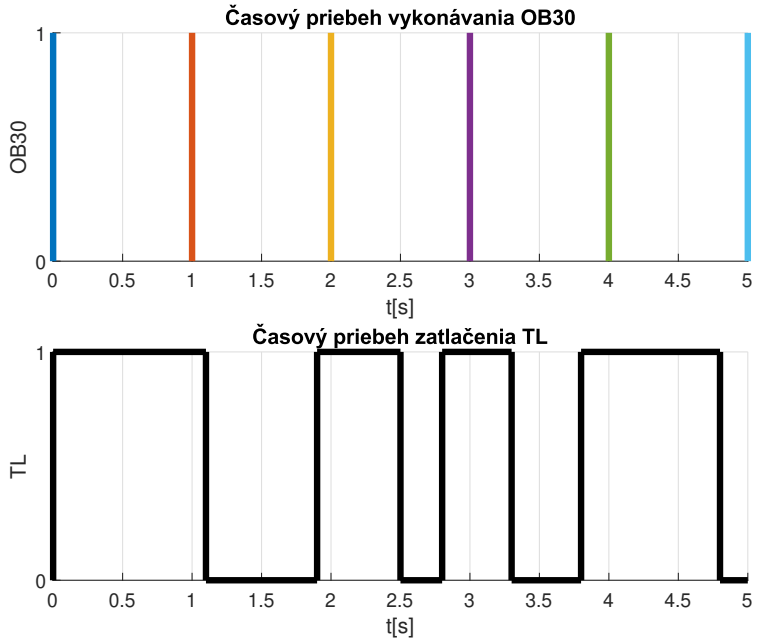
Na Obr. 4.187 a 4.188 sú zaznačené periódy vykonania OB30. V uvedených periódach odčítame stavy TL:

	0s	1s	2s	3s	4s	5s
Príklad a	0	0	0	1	1	0
Príklad b	1	1	1	1	1	0

Napr. na Obr. 4.187 (príklad a) je v 0 s hodnota TL FALSE. Tlačidlo bolo zatlačené v intervale od 0,1 s do 0,6 s, ale počas zatlačenia sa OB30 nevykonala, t. j. nedošlo k aktualizácii obrazu vstupov a výstupov. V čase 1s je tlačidlo uvoľnené, preto má stav FALSE. Uvedené hodnoty boli takto zapísané do vyššie uvedenej tabuľky. V príklade a) prvá zmena z FALSE na TRUE je v 3 s, keď sa vykoná inštrukcia *ADD*. Z analýzy prvého programu vyplýva, že sa hodnota premennej D.CISLO inkrementuje len raz. V príklade b) má TL stav TRUE vo všetkých cykloch okrem posledného v čase 5 s. Nábežná hrana sa deteguje porovnaním starej hodnoty s aktuálnou. Stará hodnota sa ukladá do pomocnej premennej SB1. Ak uvažujeme, že v čase 0 s sme prepli CPU z režimu STOP do RUN, mohla byť pamäťová oblasť %M vymazaná, preto v 0 s sa deteguje nábežná hrana. Aj v tomto príklade len raz. Cieľom úlohy bolo uvedomiť si ďalšie nevýhody, ktoré súvisia napr. s detekciou hrán v periodických úlohách.



Obr. 4.187. Časové priebehy vykonávania OB30 a zatlačenia tlačidla TL - príklad a



Obr. 4.188. Časové priebehy vykonávania OB30 a zatlačenia tlačidla TL - príklad b

Kapitola 5

Záver

Cieľom predložených skrípt bolo uvedenie čitateľa do problematiky riadenia diskrétnych udalostných systémov pomocou programovateľných logických automatov v jazyku rebríkových schém. Absolvent predmetu Riadiace systémy študijného programu Robotika a kybernetika na FEI STU navrhuje, implementuje a overuje algoritmy riadenia diskrétnych udalostných systémov pomocou základných inštrukcií a pomocou funkčných blokov. Ku koncu semestra vytvára uzavretú regulačnú slučku na báze PID regulátorov. Skriptá sa zameriavali na prvú časť, a to na základné inštrukcie. Zvládnutie základov je dobrým predpokladom správneho návrhu objektovo orientovaných algoritmov, ako aj zvládnutie spracovania analógových signálov a PID regulácie. Pevne veríme, že skriptá pomôžu študentom, ktorí získajú okrem teoretických znalostí aj praktické znalosti využiteľné v praxi.

Kapitola 6

Prílohy - Zbierka príkladov

V tejto kapitole sa nachádza zbierka príkladov rozdelená do niekoľkých kategórií. Prvá kategória obsahuje príklady, v ktorých je cieľom určiť poradie vykonania inštrukcií. V ďalšej podkapitole sú prezentované jednoduché príklady, ktoré majú slúžiť na krátku analýzu. Tretia kapitola obsahuje krátke slovné zadania. V poslednej kapitole nájdete riešenia príkladov.

6.1 Poradie vykonania príkazov

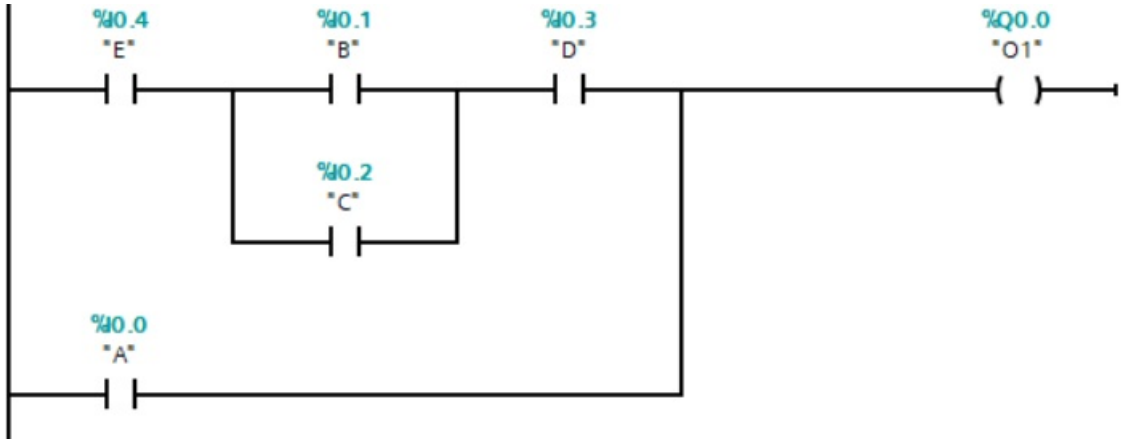
6.1.1 Zadanie 1

Majme príklad na Obr. 6.1. V akom poradí sa vykonajú inštrukcie?

Obr. 6.1. Zadanie 1 - poradie vykonania inštrukcií

6.1.2 Zadanie 2

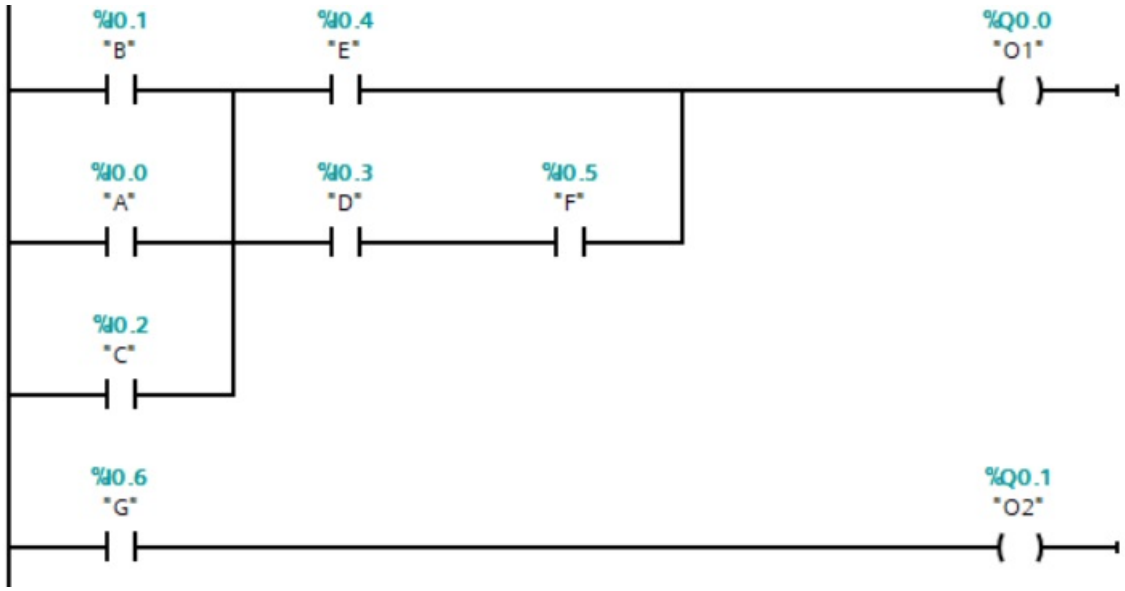
Majme príklad na Obr. 6.2. V akom poradí sa vykonajú inštrukcie?



Obr. 6.2. Zadanie 2 - poradie vykonania inštrukcií

6.1.3 Zadanie 3

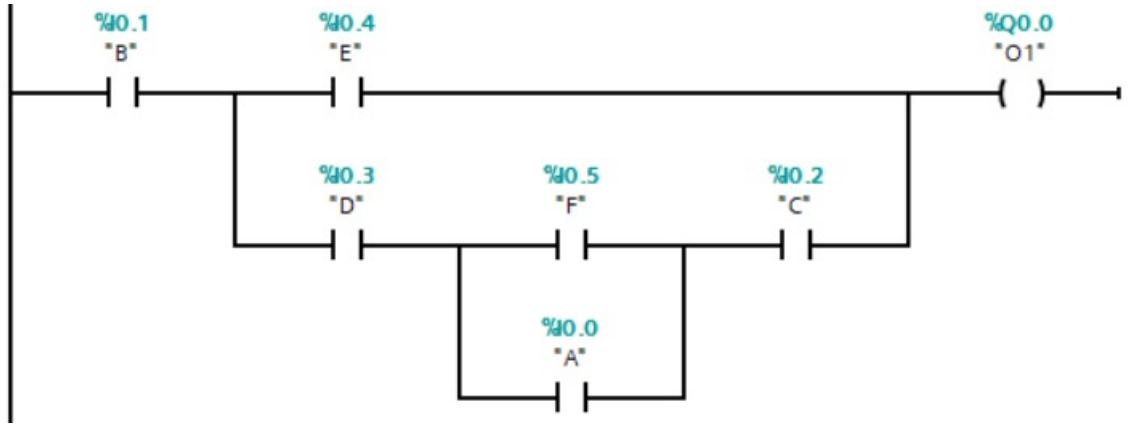
Majme príklad na Obr. 6.3. V akom poradí sa vykonajú inštrukcie?



Obr. 6.3. Zadanie 3 - poradie vykonania inštrukcií

6.1.4 Zadanie 4

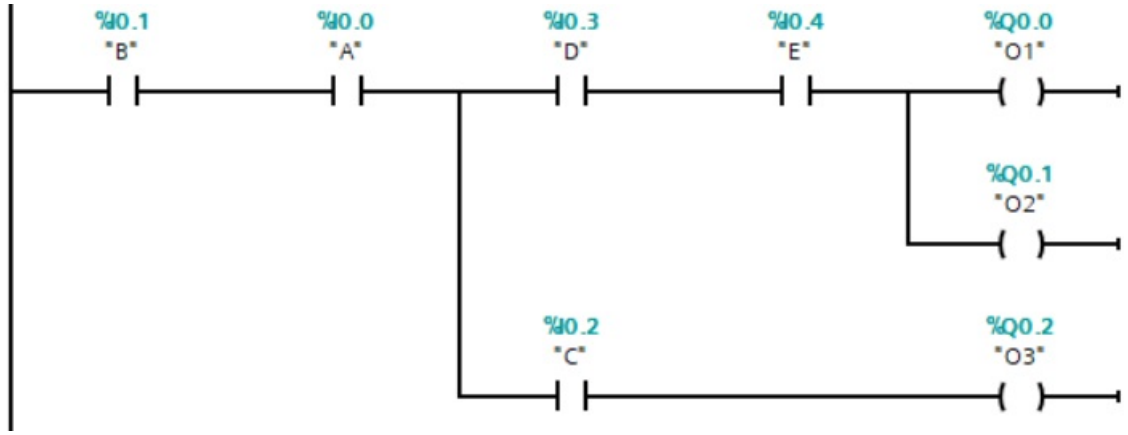
Majme príklad na Obr. 6.4. V akom poradí sa vykonajú inštrukcie?



Obr. 6.4. Zadanie 4 - poradie vykonania inštrukcií

6.1.5 Zadanie 5

Majme príklad na Obr. 6.5. V akom poradí sa vykonajú inštrukcie?

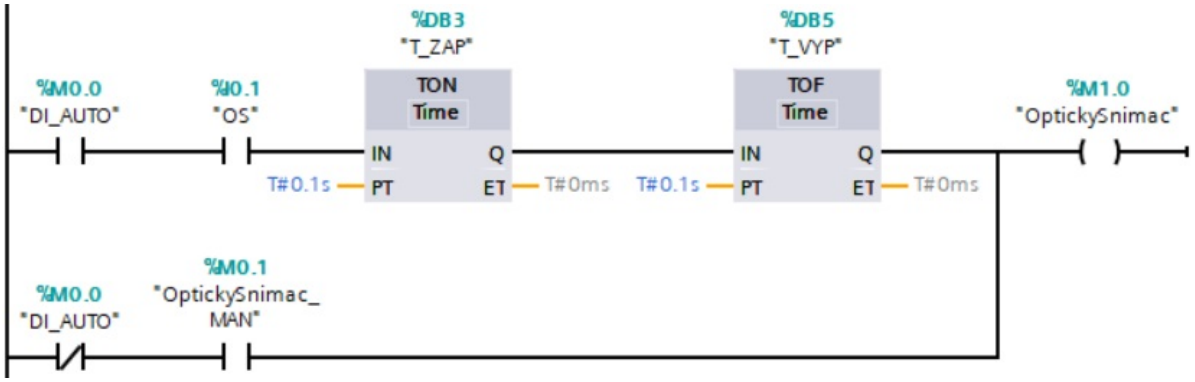


Obr. 6.5. Zadanie 5 - poradie vykonania inštrukcií

6.2 Analýza programov

6.2.1 Zadanie 1

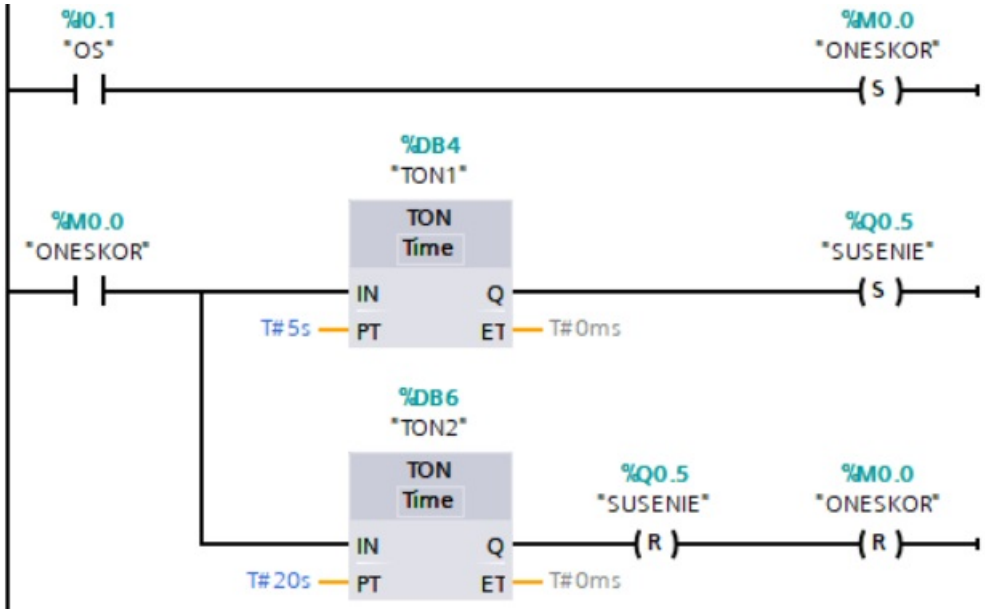
Majme príklad na Obr. 6.6. Opíšte funkcionalitu programu.



Obr. 6.6. Zadanie 1 - analýza programu

6.2.2 Zadanie 2

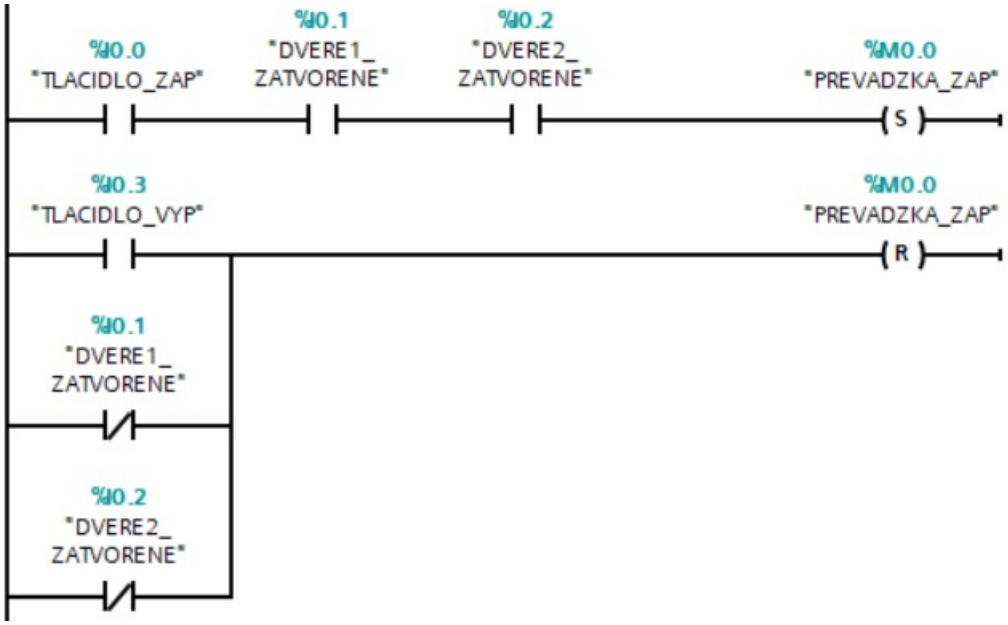
Majme príklad na Obr. 6.7. Opíšte funkcionalitu programu.



Obr. 6.7. Zadanie 2 - analýza programu

6.2.3 Zadanie 3

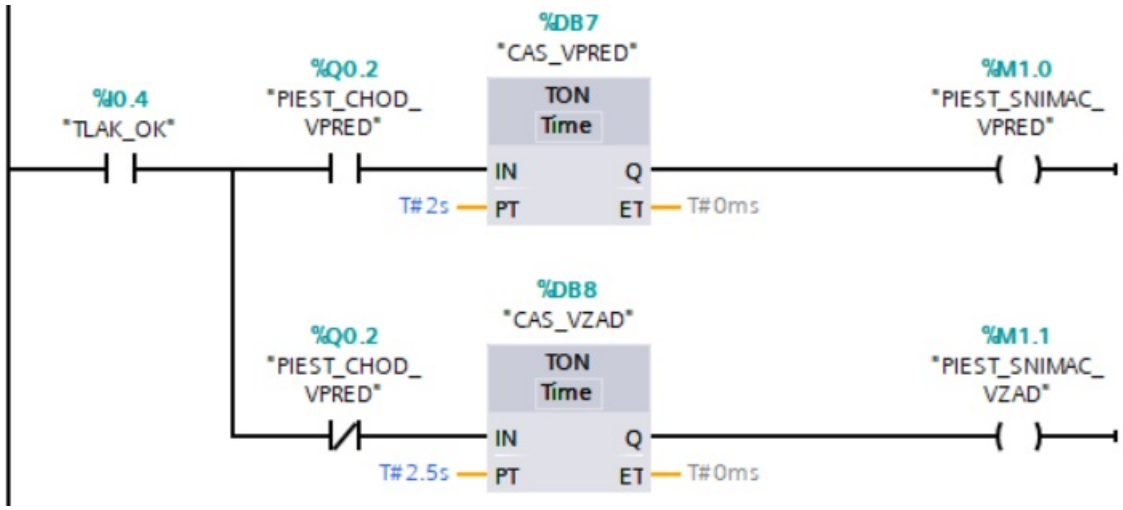
Majme príklad na Obr. 6.8. Opíšte funkcionalitu programu.



Obr. 6.8. Zadanie 3 - analýza programu

6.2.4 Zadanie 4

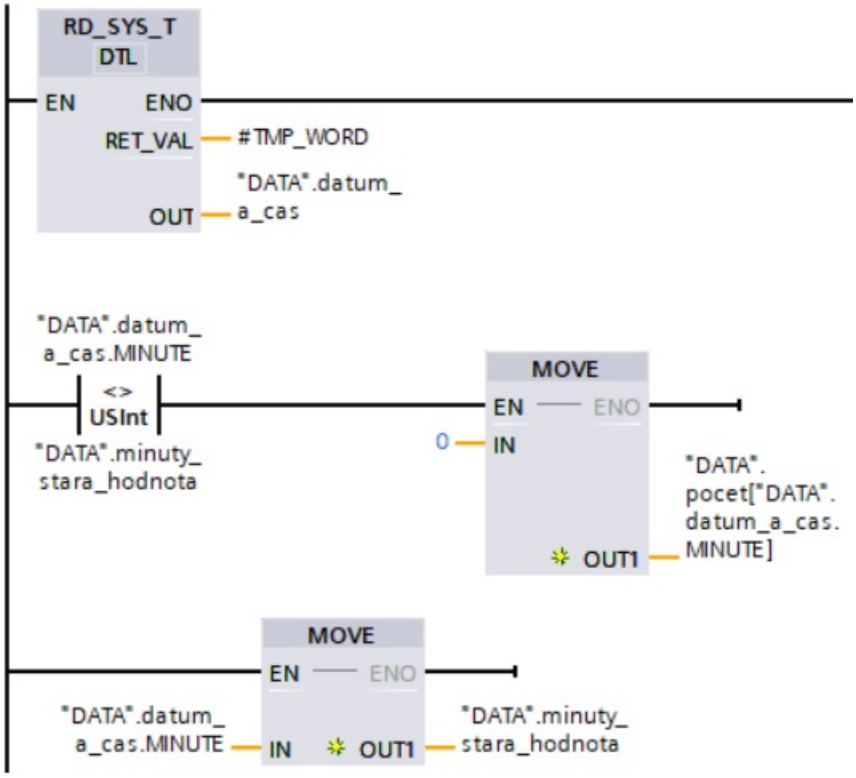
Majme príklad na Obr. 6.9. Opíšte funkcionalitu programu.



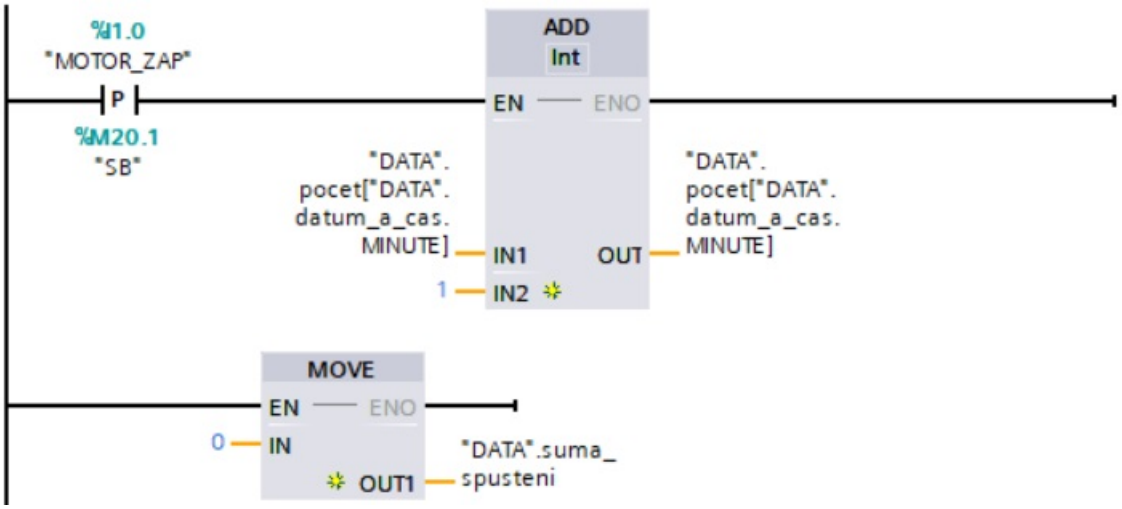
Obr. 6.9. Zadanie 4 - analýza programu

6.2.5 Zadanie 5

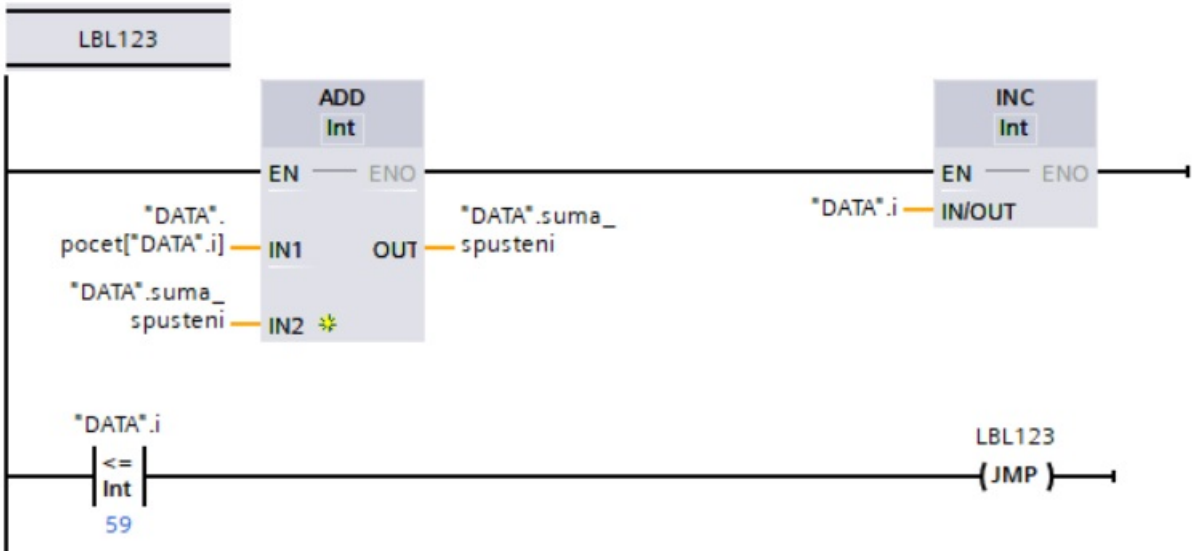
Majme program rozdelený na viacero rebríkov (Obr. 6.10 až 6.13). Opíšte funkcionalitu programu.



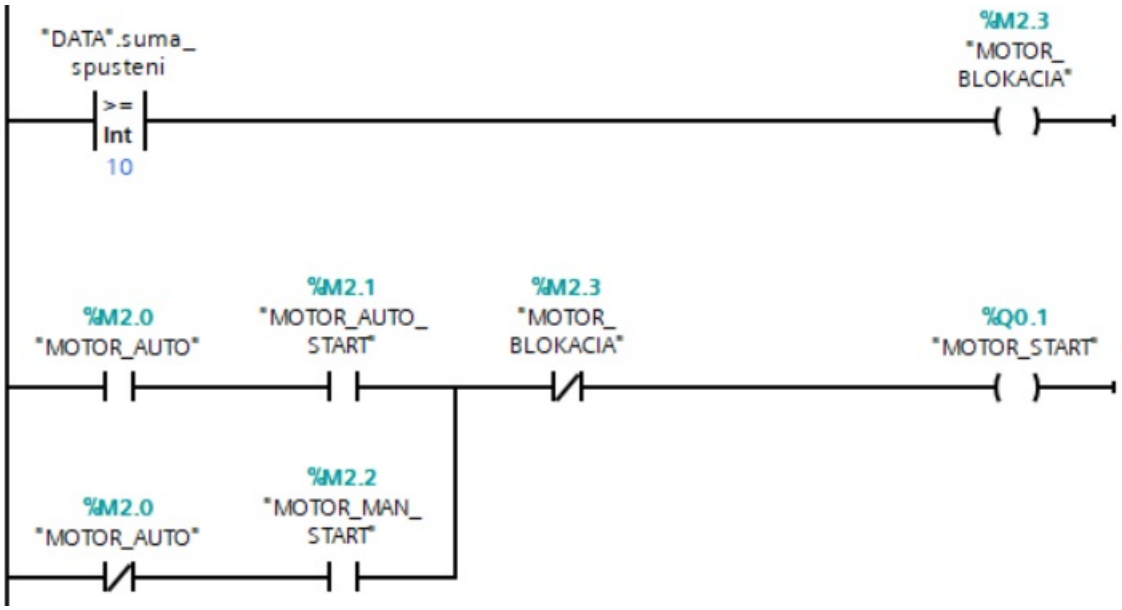
Obr. 6.10. Zadanie 5 - analýza programu (Network 1)



Obr. 6.11. Zadanie 5 - analýza programu (Network 2)



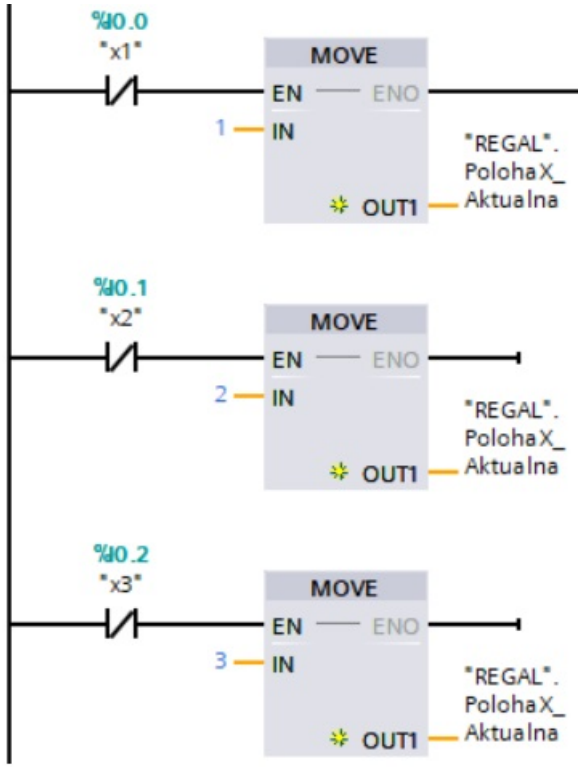
Obr. 6.12. Zadanie 5 - analýza programu (Network 3)



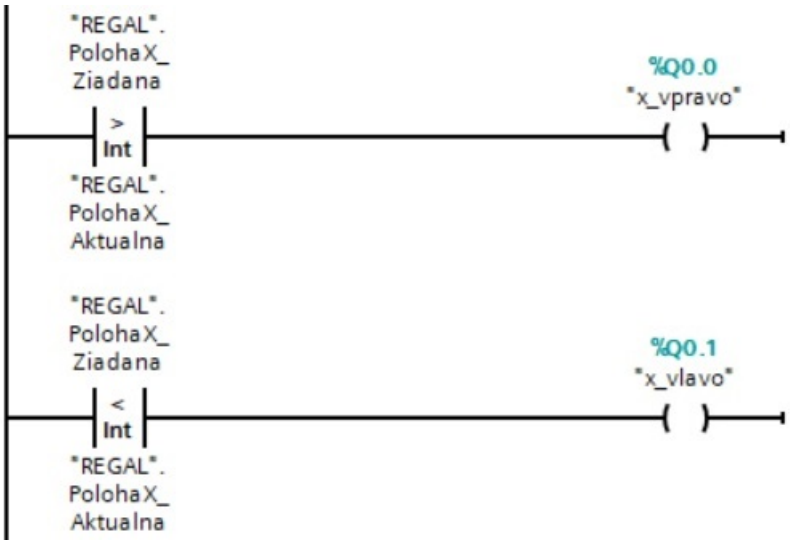
Obr. 6.13. Zadanie 5 - analýza programu (Network 4)

6.2.6 Zadanie 6

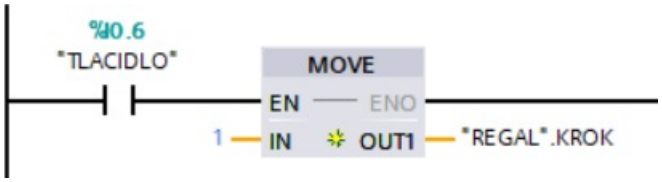
Majme program rozdelený na viacero rebríkov (Obr. 6.14 až 6.20). Opíšte funkcionalitu programu.



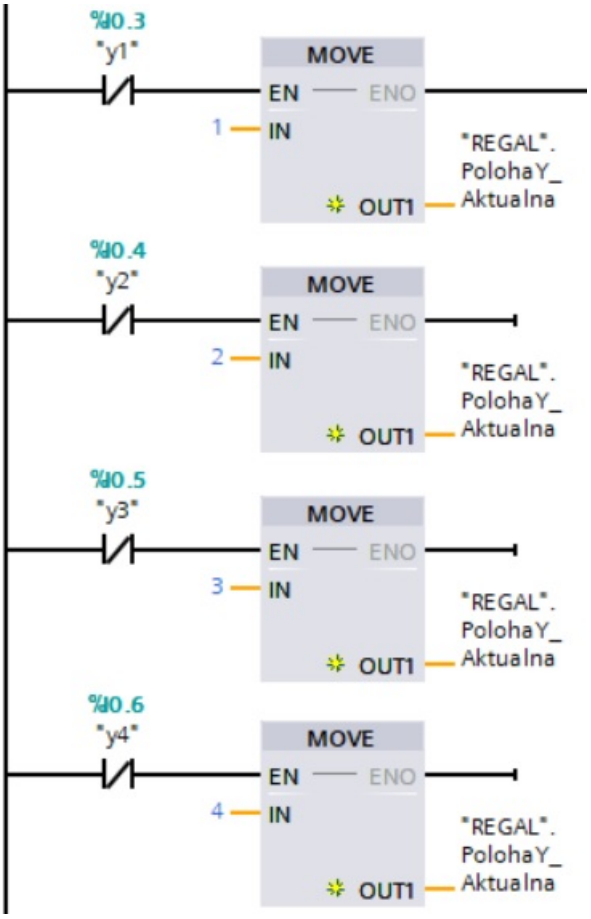
Obr. 6.14. Zadanie 6 - analýza programu (Network 1)



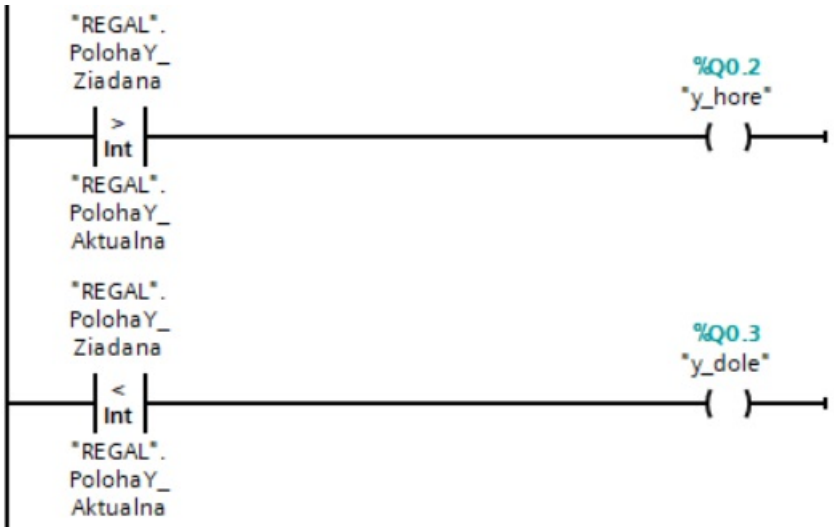
Obr. 6.15. Zadanie 6 - analýza programu (Network 2)



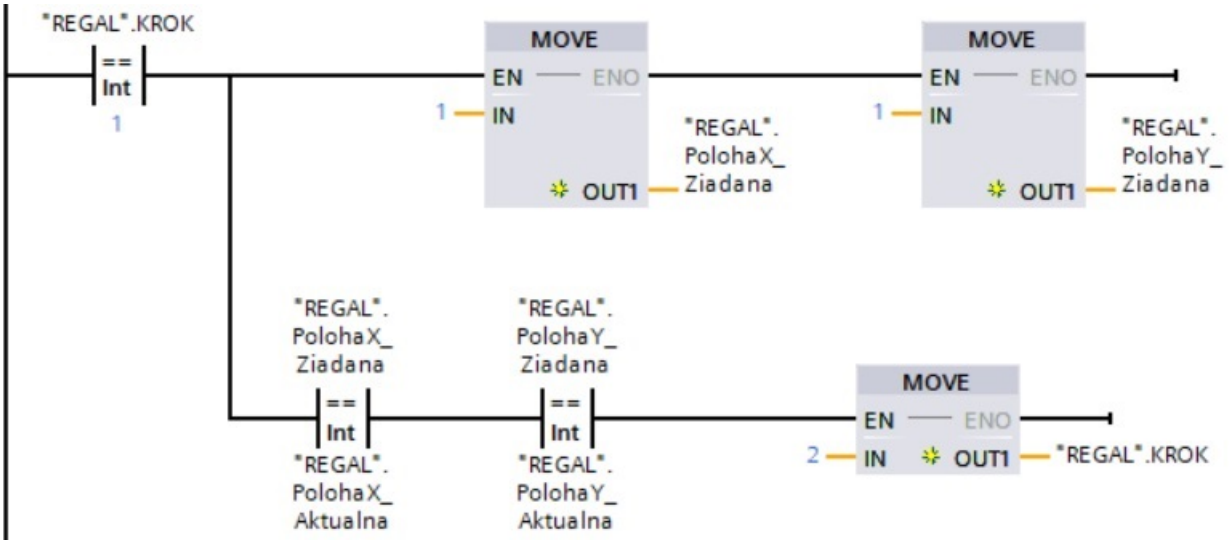
Obr. 6.16. Zadanie 6 - analýza programu (Network 3)



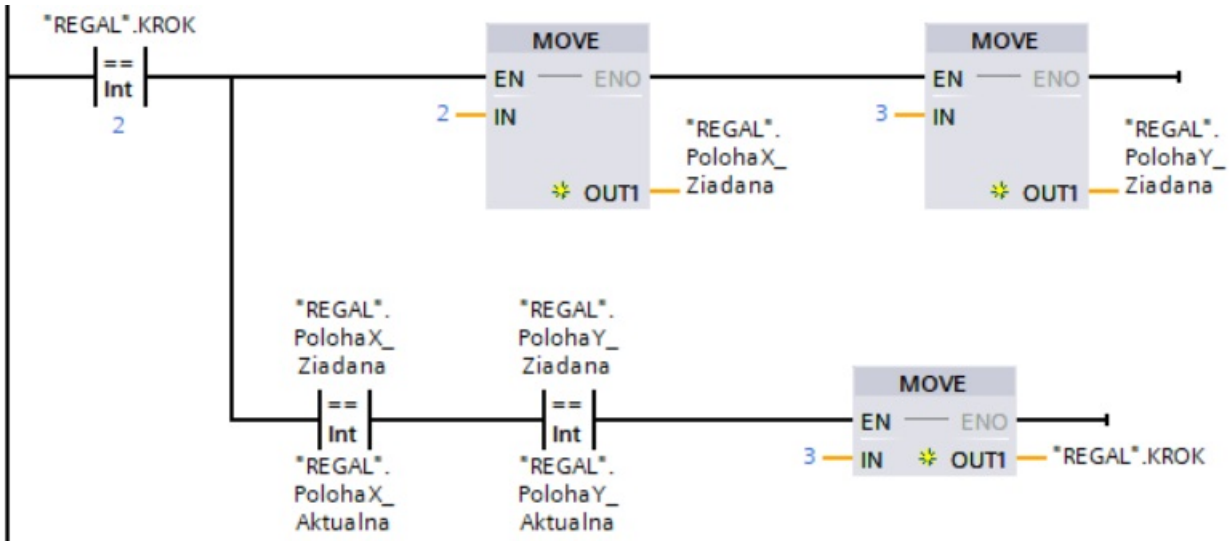
Obr. 6.17. Zadanie 6 - analýza programu (Network 4)



Obr. 6.18. Zadanie 6 - analýza programu (Network 5)



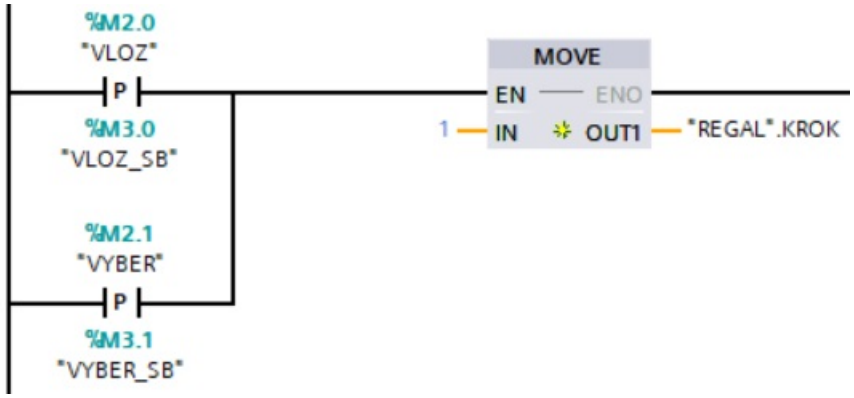
Obr. 6.19. Zadanie 6 - analýza programu (Network 6)



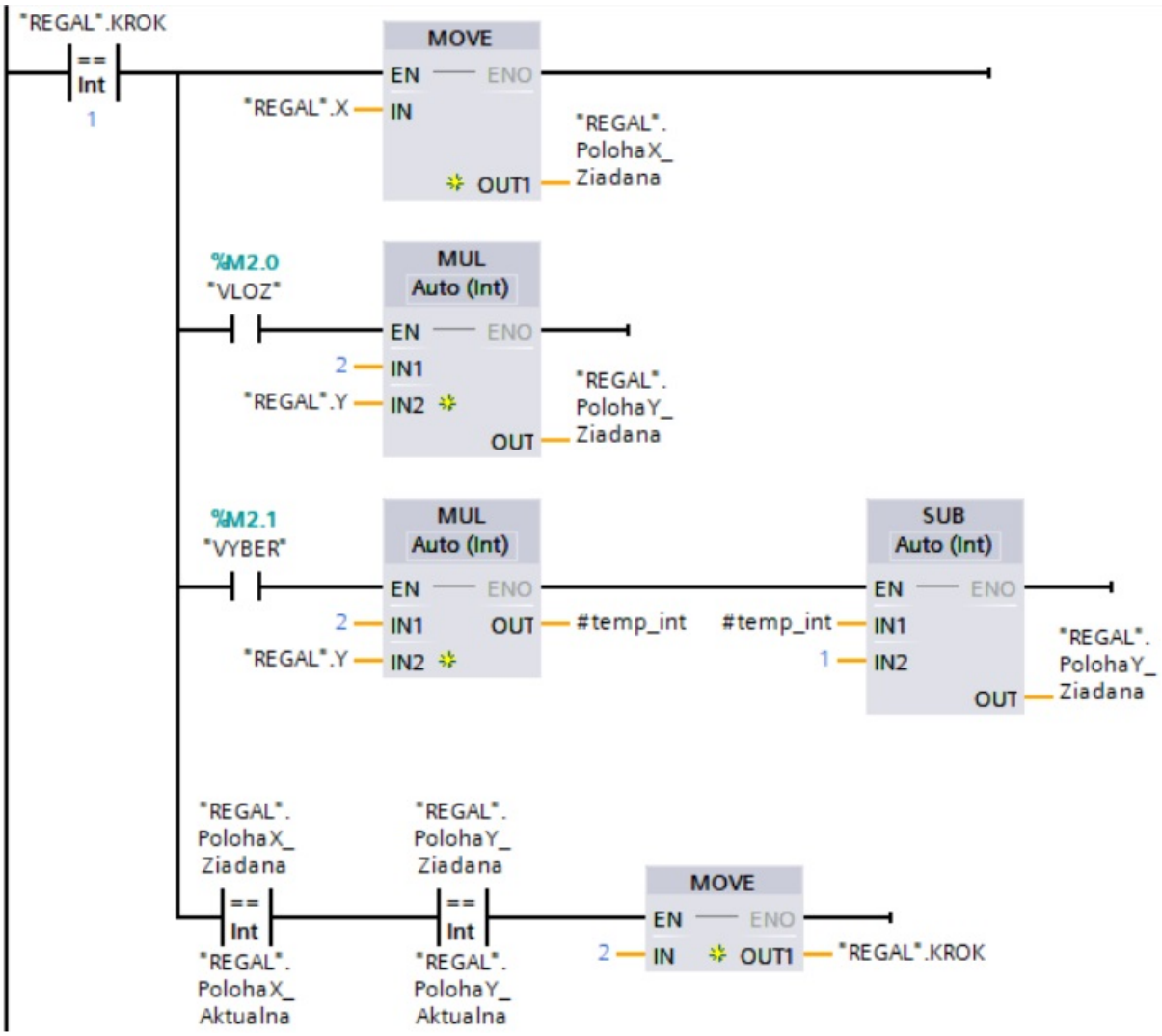
Obr. 6.20. Zadanie 6 - analýza programu (Network 7)

6.2.7 Zadanie 7

Majme modifikovaný program z predchádzajúceho zadania. Modifikácie sú v rebríkoch č. 3 a 6 (Obr. 6.21 až 6.22). Opíšte funkcionalitu programu.



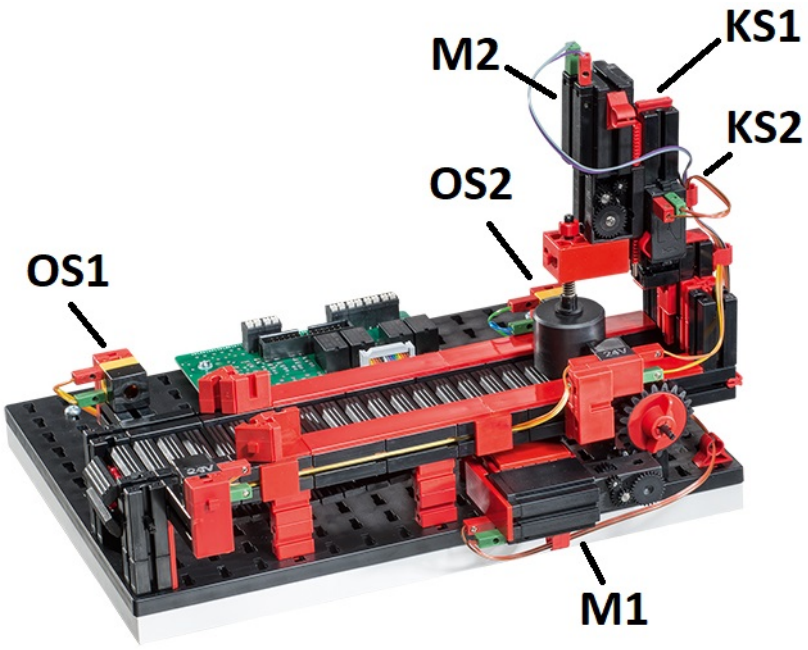
Obr. 6.21. Zadanie 7 - analýza programu (Network 3)



Obr. 6.22. Zadanie 7 - analýza programu (Network 6)

6.3 Slovné zadania

Majme model na Obr. 6.23. Model sa skladá z dopravníka, ktorý je poháňaný motorom M1 vpred a vzad. Na dopravníku sú umiestnené rozpínacie optické snímače OS1 a OS2 detegujúce prepravované objekty dopravníkom. Na konci dopravníka je umiestnené lisovacie zariadenie, ktoré sa ovláda motorom M2 smerom hore a dole. Jeho dolná a horná poloha sú určené spínacími kontaktnými snímačmi KS1 a KS2. Zoznam vstupov a výstupov je na Obr. 6.24.



Obr. 6.23. Model „Punching Machine with Conveyor Belt 24V - Education“

	Name	Tag table	Data type	Address
1	OS1	Default tag table	Bool	%I0.0
2	OS2	Default tag table	Bool	%I0.1
3	KS1	Default tag table	Bool	%I0.2
4	KS2	Default tag table	Bool	%I0.3
5	M1_VPRED	Default tag table	Bool	%Q0.0
6	M1_VZAD	Default tag table	Bool	%Q0.1
7	M2_DOLE	Default tag table	Bool	%Q0.2
8	M2_HORE	Default tag table	Bool	%Q0.3

Obr. 6.24. Adresy vstupov a výstupov modelu

6.3.1 Zadanie 1

Vytvorte program v jazyku LD, ktorý objekt naložený pred OS1 prenesie na koniec dopravníka k OS2 a potom ho vráti späť na nakladaciu pozíciu k OS1. Na tomto mieste dopravník zastane a bude sa čakať na odobratie objektu. Po odobratí možno opäť naložiť nový objekt. Uvažujme len jeden objekt na dopravníku.

6.3.2 Zadanie 2

Doplňte program z predchádzajúceho zadania o parameter M1_ZIADANY_POCET, ktorý predstavuje žiadaný počet presunutí objektu k OS2. Upravte program tak, aby objekt vykonal žiadaný počet presunutí (v predchádzajúceho úlohe to bolo 1) a na konci čakal objekt na odobratie na OS1. Ak napr. M1_ZIADANY_POCET bude 2, znamená to presun objektu z OS1 na OS2, späť k OS1, opäť k OS2 a návrat k OS1, kde bude čakať na odobratie. Uvažujme len jeden objekt na dopravníku. Nie je potrebné ošetriť nulový žiadaný počet.

6.3.3 Zadanie 3

Vytvorte program, ktorý realizuje pohyb lisovacieho zariadenia dole a potom hore, ak je objekt detegovaný snímačom OS2. Opätovné spustenie sekvencie má byť po odobratí objektu a naložení nového. Pozn.: úloha je podobná 1. úlohe s tým rozdielom, že sa objekt nepohybuje, ale pohybuje sa lisovacia časť modelu.

6.3.4 Zadanie 4

Integrujte vytvorený program zo zadania 3 do zadania 2.

6.3.5 Zadanie 5

Modifikujte zadanie 4 tak, aby pre lis bolo možné zadať žiadaný počet lisovaní cez parameter dátového typu INT. Nie je potrebné ošetriť nulový žiadaný počet.

6.4 Riešenia

Riešenia zadaní z kap. 6.1:

- Zadanie 1 - Poradie je A, C, D, B, O1.
- Zadanie 2 - Poradie je E, B, C, D, A, O1.
- Zadanie 3 - Poradie je B, A, C, E, D, F, O1, G, O2.
- Zadanie 4 - Poradie je B, E, D, F, A, C, O1.
- Zadanie 5 - Poradie je B, A, D, E, O1, O2, C, O3.

Riešenia zadaní z kap. 6.2:

- Zadanie 1 - Program slúži na spracovanie vstupu z optického snímača OS. V automatickom režime sa zapína oneskorenie po 0,1 sekunde a vypína po 0,1 sekunde. V praxi je často požiadavka na filtrovanie digitálnych signálov alebo ich podržanie na dlhší čas. V dolnej vetve programu možno v manuálnom režime (realizácia: nie v automatickom) nastaviť (simulovať) hodnotu snímača bez nutnosti vynucovať túto hodnotu.

- Zadanie 2 - Po detekcii objektu optickým snímačom sa premenná ONESKOR nastaví na hodnotu TRUE. Po uplynutí 5 sekúnd sa spustí sušenie, ktoré sa vypne 20 sekúnd po oneskorení. Súčasne sa pomocný stav ONESKOR nastaví na FALSE. Ak objekt spôsobí oneskorenie, spustí sa sušenie po 5 sekundách a bude trvať 15 sekúnd.

- Zadanie 3 - Ak je tlačidlo TLACIDLO_ZAP stlačené a dvere 1 aj 2 sú zatvorené, spustí sa prevádzka. Ak sa stlačí tlačidlo TLACIDLO_VYP alebo sa otvoria dvere 1 alebo dvere 2, prevádzka sa preruší.

- Zadanie 4 - Ide o simuláciu koncových polôh piestu, pretože piest nemá koncové snímače. Vstup TLAK_OK kontroluje, či je dostatočný tlak vzduchu na ovládanie pohybu piestu. Ak je požadovaný pohyb vpred, po 2 sekundách sa zapne simulovaný snímač vpred. Ak nie je požadovaný pohyb vpred (t. j. je požadovaný pohyb vzad), po 2.5 sekundách sa aktivuje simulovaný snímač vzad. V príklade sú použité dva rôzne časy, pretože pohyb vpred a vzad nemusia trvať rovnako dlho. Ak tlak vzduchu nie je dostatočný, piest sa nepohne a simulované snímače sa nezapnú.

- Zadanie 5 - Program v prvom rebríku získa aktuálny dátum a čas. Aktuálne minúty sa porovnajú s hodnotou z predchádzajúceho cyklu. Ak sa tieto hodnoty líšia, došlo k zmene hodnoty minút a položka s indexom minúty v poli pozícií sa vymaže. Napríklad, ak sa minúta zmenila z 2 na 3, hodnota v poli pocet[3] sa vymaže. Pole DATA.pocet slúži na uchovanie počtu zapnutí motora v aktuálnej minúte. V dolnej časti sa uchováva aktuálna hodnota minút pre potreby porovnania v ďalšom cykle. V druhom rebríku sa inkrementuje počet zapnutí motora. Táto hodnota nie je riešená jednou premennou, ale poľom, pretože je stanovený maximálny dovolený počet spustení motora za hodinu (ktorý bude riešený nižšie). Ak sa pokúšame v 4. minúte spustiť motor viackrát, počet spustení sa inkrementuje v premennej počet[4]. V spodnej časti sa vymaže celkový počet spustení, ktorý sa v ďalšom rebríku počíta ako celková suma spustení počas 60 minút. For cyklus je realizovaný skokom na návestie. Najprv sa k sume pripočíta počet spustení v premennej počet[0]. Potom sa inštrukciou INC zvýši hodnota indexu. V dolnej vetve sa testuje, či sa sčítali všetky spustenia za všetky minúty. Ak nie, skočí sa na návestie LBL123. Po dokončení cyklu máme v premennej suma_spusteni celkový počet spustení motora za poslednú hodinu. V poslednom rebríku sa motor ovláda v manuálnom a automatickom režime, ak nie je motor blokovaný. Jedinou blokáciou je počet spustení, ktorý ak je väčší alebo rovný ako 10, motor nie je možné spustiť. Na opätovné spustenie musí klesnúť počet spustení za poslednú hodinu. Nulovanie starých záznamov sa rieši prvým rebríkom programu.

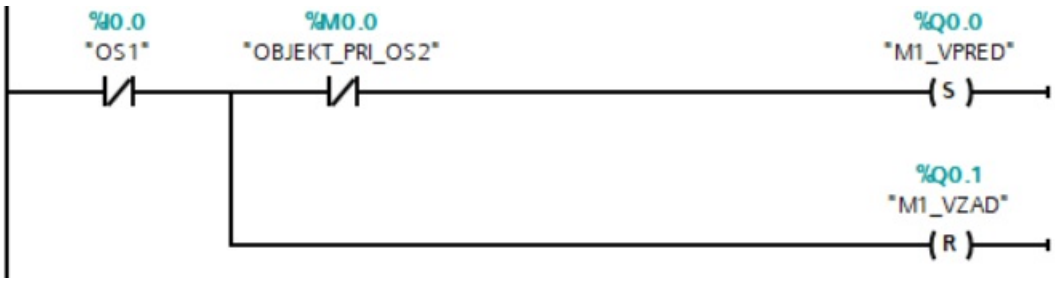
- Zadanie 6 - Program v prvom rebríku sníma polohu regálového zakladača na základe signálov z NC snímačov a zapisuje ju do premennej PolohaX_Aktualna dátového typu INT. V druhom rebríku sa porovnáva žiadaná poloha s aktuálnou. Ak je žiadaná poloha väčšia ako aktuálna, os X zakladača sa spustí vpravo. Ak je žiadaná poloha menšia ako aktuálna, os X zakladača sa spustí vľavo. Samozrejme, ak sú polohy rovnaké, tak nie je potrebné vykonávať žiadny pohyb. Podobne je realizovaný pohyb osi Y smerom hore a dole v rebríkoch 4 (Obr. 6.17) a 5 (Obr. 6.18). V treťom rebríku (Obr. 6.16) sa po stlačení tlačidla do premennej KROK zapíše hodnota 1 a začne sa vykonávať krokové riadenie zakladača. V šiestom rebríku, ak sa KROK

rovná 1, nastavia sa pevné preddefinované polohy, kam sa má zakladač premiestniť. Týmto spôsobom sa zakladač polohuje na pozíciu 1, 1 (poloha vľavo dole). Po dosiahnutí želanej polohy sa do premennej KROK vloží hodnota 2. Analogicky v poslednom rebríku sa zakladač polohuje na pozíciu 2, 3 a po dosiahnutí želanej polohy sa prechádza na krok 3.

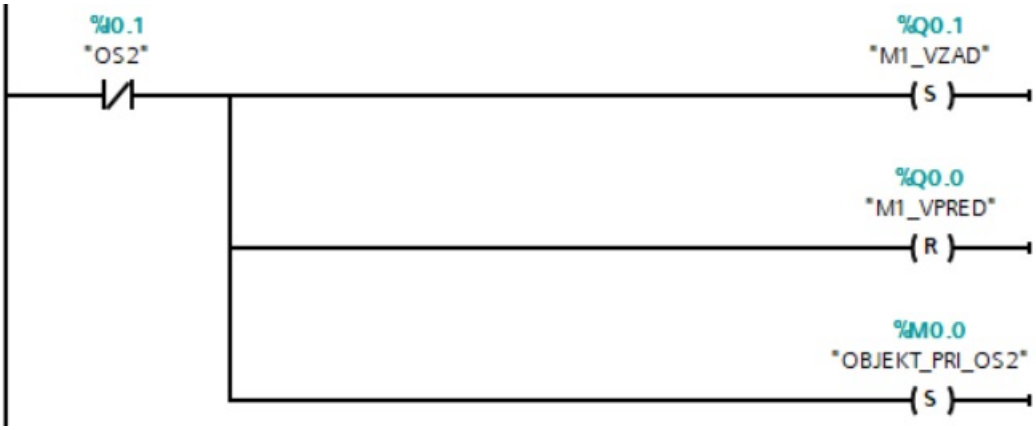
- Zadanie 7 - Program, ako aj v predošlom zadaní, ovláda regálový zakladač, ktorý vkladá a vyberá objekty v úložnom priestore. Pre každú výšku (polohu y) máme dvojicu snímačov, jeden definuje polohu pod a druhý polohu nad danou výškou. Napríklad, snímač y1 je umiestnený pod polohu 1 a snímač y2 nad polohou 1. Podobne snímač y3 je umiestnený pod polohu 2 a snímač y4 nad polohou 2. Pri vkladaní je cieľom navigovať zakladač na vyššiu polohu, vojsť do regála, položiť objekt (t. j. polohovať na nižší snímač) a vyjsť vzad (von) z regála. Opačný postup sa používa pri vyberaní. Najprv sa polohuje na nižšiu polohu, nasleduje pohyb vpred, pohyb hore (nadvihnutie objektu) a poloha vzad (výber objektu zo regálu). Program v 3. rebríku realizuje vkladanie aj vyberanie od nábežných hrán. V 6. rebríku je uvedená časť programu na polohovanie osí X a Y, pričom reálna poloha osi Y závisí od toho, či sa vkladá alebo vyberá. Ak sa vkladá, polohujeme na pozíciu 2, 4, 6 a tak ďalej a ak vyberáme, polohujeme na pozície 1, 3, 5 a tak ďalej. Matematicky to zabezpečujú inštrukcie MUL alebo MUL a SUB.

Riešenia zadaní z kap. 6.3:

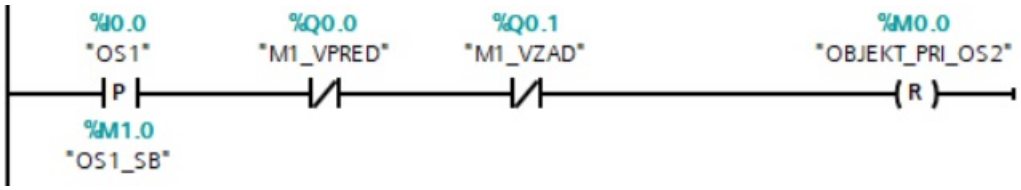
- Zadanie 1. Príklad riešenia je na Obr. 6.25 až 6.27. V prvom rebríku snímač OS1 deteguje objekt a spúšťa pohyb dopravníka vpred (a zároveň nie vzad). Pomocný stav OBJEKT_PRI_OS2 bol zavedený na blokovanie opätovného spustenia pohybu po návrate objektu z konca dopravníka. Pomocný bit je len v hornej vetve, lebo detekciou objektu sa v spodnej vetve musí pohyb zastaviť. Druhý rebrík je podobný prvému. Ak bol objekt prepravený na koniec dopravníka pred optický snímač OS2, otočí sa smer pohybu a nastaví sa pomocný stav, že objekt bol na konci dopravníka. Posledný rebrík realizuje odobratie objektu. Presnejšie ide o vynulovanie pomocného stavu. Odobratím objektu vzniká zmena z FALSE na TRUE, t. j. nábežná hrana. V AND podmienke sú doplnené výstupy, aby sa zabezpečilo, že ide o odobratie, keď dopravník nie je v pohybe.



Obr. 6.25. Riešenie zadania 1 - Rebrík 1

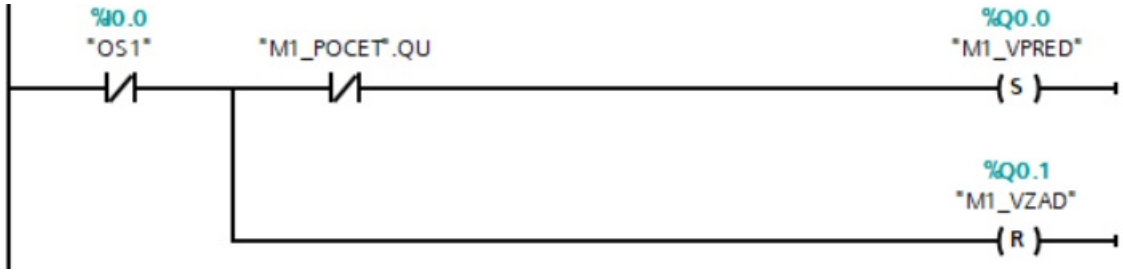


Obr. 6.26. Riešenie zadania 1 - Rebrík 2

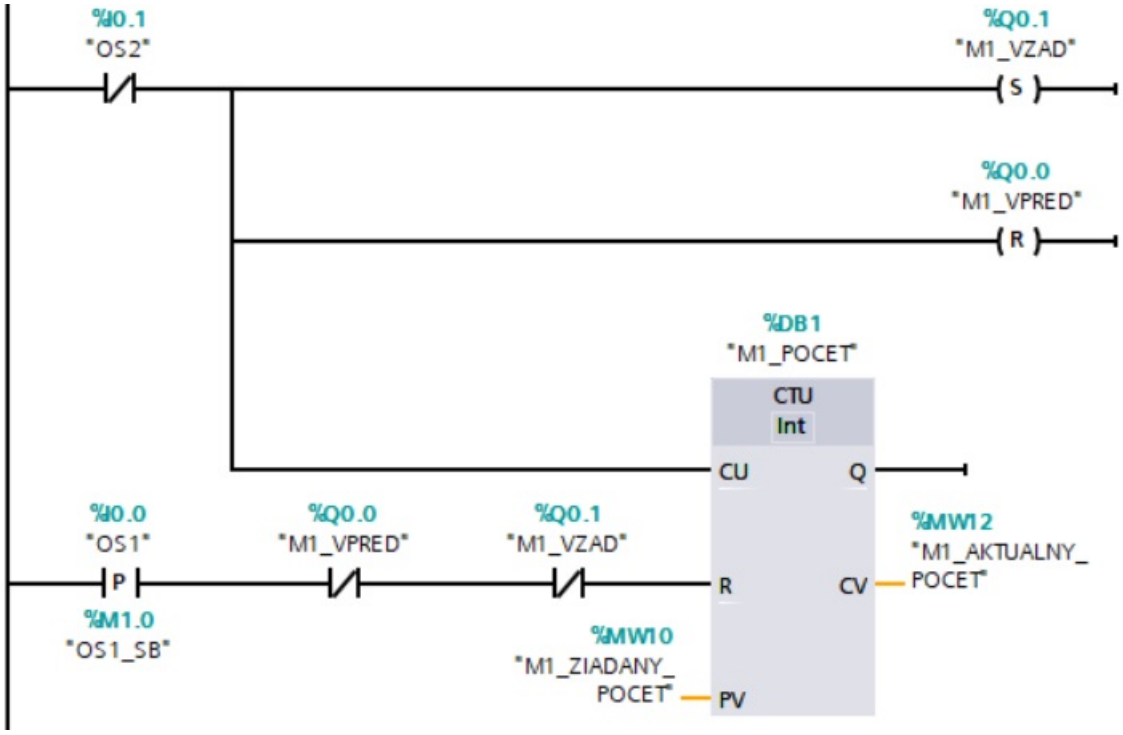


Obr. 6.27. Riešenie zadania 1 - Rebrík 3

- Zadanie 2 - Príklad prvého riešenia je na Obr. 6.28 až 6.29. Riešenie je založené na využití počítadla (viď 2. rebrík). Zakaždým keď je detegovaný objekt optickým snímačom OS2, navýši sa jeho hodnota CV. V demonštračnom príklade sa táto hodnota ukladá do premennej M1_AKTUALNY_POCET. Podmienka vynulovania počítadla je rovnaká ako podmienka vynulovania pomocného bitu v predchádzajúcej úlohe. Ak sa dosiahne žiadaný počet opakovaní, výstup počítadla Q (presnejšie QU) sa nastaví na TRUE. Ten blokuje opätovné spustenie dopravníka analogicky ako v 1. úlohe. Nulový žiadaný počet by sme mohli ošetriť napr. pridaním porovnávacieho bloku za výstup Q počítadla, kde by sa testovala nenulová žiadaná hodnota.

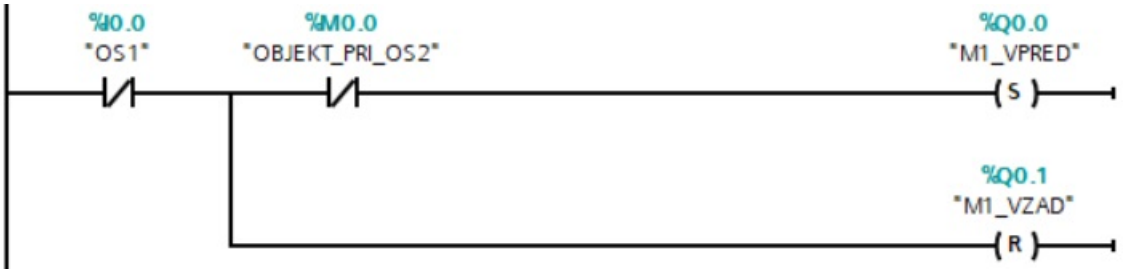


Obr. 6.28. 1. riešenie zadania 2 - Rebrík 1

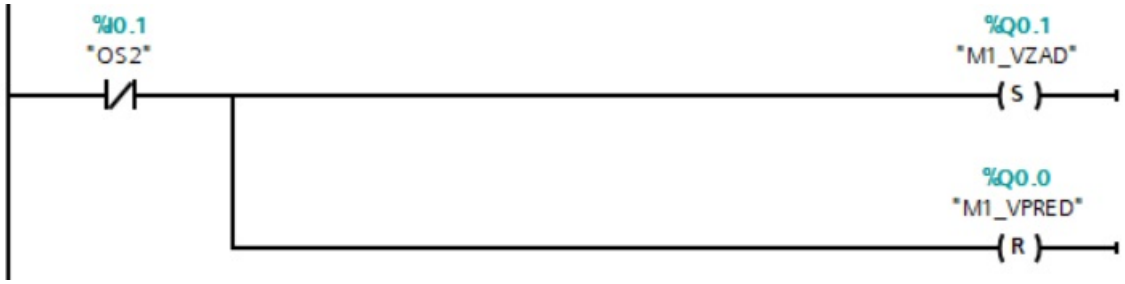


Obr. 6.29. 1. riešenie zadania 2 - Rebrík 2

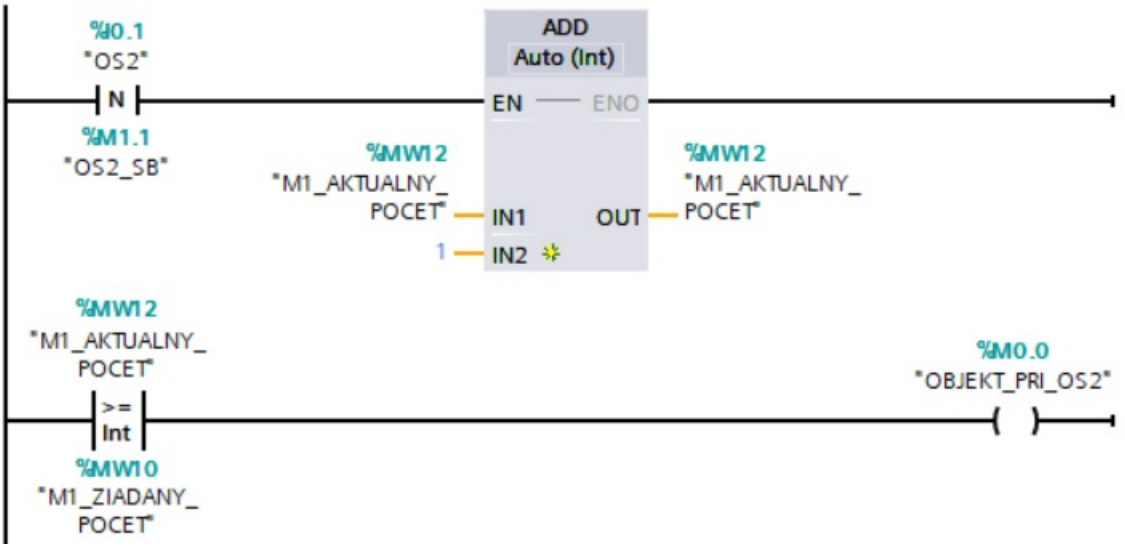
Príklad druhého riešenia je na Obr. 6.30 až 6.33. Namiesto počítadla je použitá inštrukcia ADD, ktorá vyžaduje jednorazové vykonanie. Využijeme na to dobežnú hranu snímača OS2 (pozri 3. rebrík). Ak je dosiahnutý požadovaný počet operácií, nastaví sa pomocný bit OBJEKT_PRI_OS2 na TRUE (pozri spodnú časť 3. rebríka). V tejto implementácii má význam dosiahnutia požadovaného počtu opakovaní. Spustenie dopravníka vpred a vzad je podobné ako v predošlých príkladoch. Nulovanie na poslednom rebríku č. 4 je realizované priradením hodnoty 0 do premennej M1_AKTUALNY_POCET. Ten spôsobí vynulovanie pomocného bitu OBJEKT_PRI_OS2 a umožnenie spustenia dopravníka vpred v 1. rebríku.



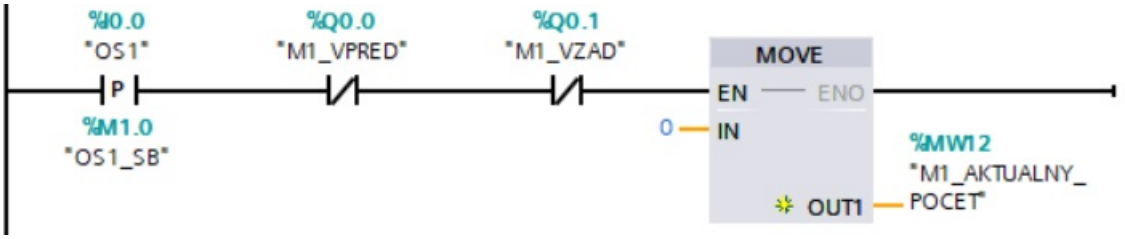
Obr. 6.30. 2. riešenie zadania 2 - Rebrík 1



Obr. 6.31. 2. riešenie zadania 2 - Rebrík 2

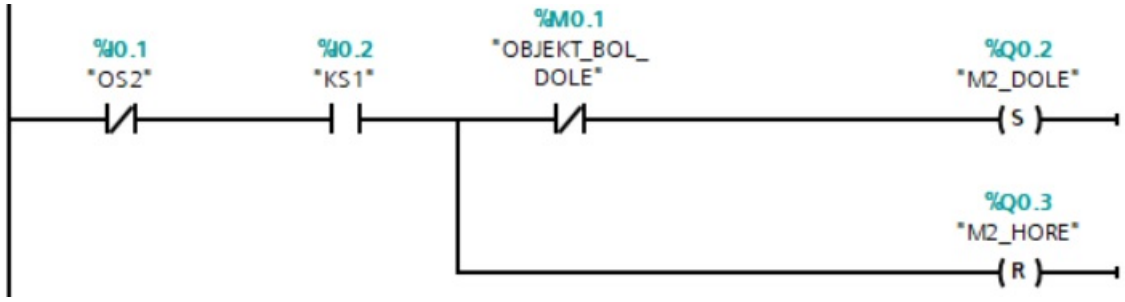


Obr. 6.32. 2. riešenie zadania 2 - Rebrík 3

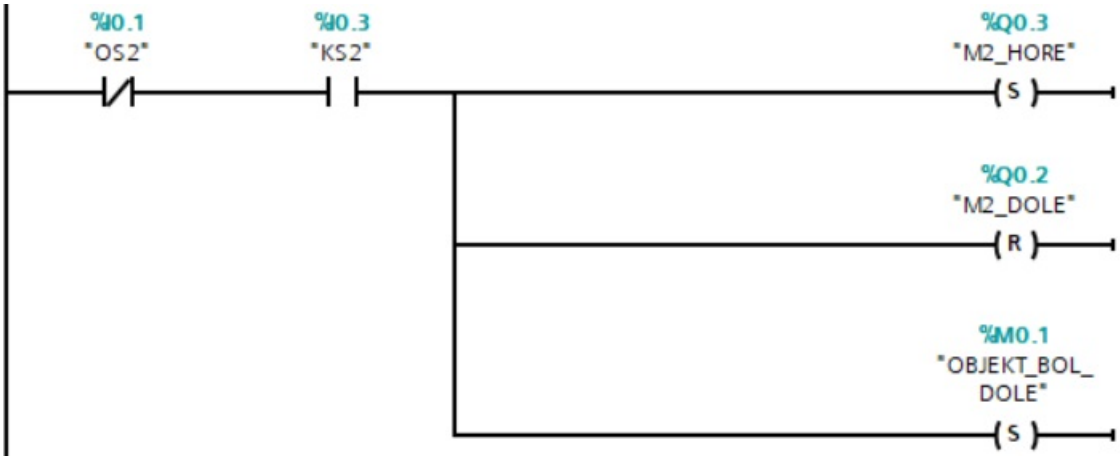


Obr. 6.33. 2. riešenie zadania 2 - Rebrík 4

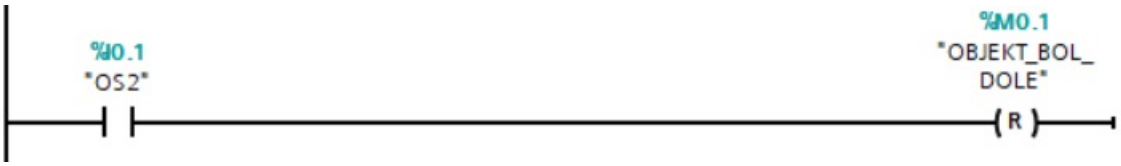
- Zadanie 3 - Príklad riešenia je na Obr. 6.34 až 6.36. V prvom rebríku sa deteguje prítomnosť objektu optickým snímačom OS2. Ak je zároveň zopnutý horný kontaktný snímač KS1, lis sa začne pohybovať dole. Pomocný bit OBJEKT_BOL_DOLE blokuje opätovné vykonanie pohybu dole po návrate lisu do hornej polohy. V druhom rebríku dochádza k zmene smeru pohybu a nastaveniu pomocného bitu OBJEKT_BOL_DOLE na TRUE. Posledný rebrík vynuluje pomocný bit po odobratí objektu od optického snímača OS2.



Obr. 6.34. Riešenie zadania 3 - Rebrík 1

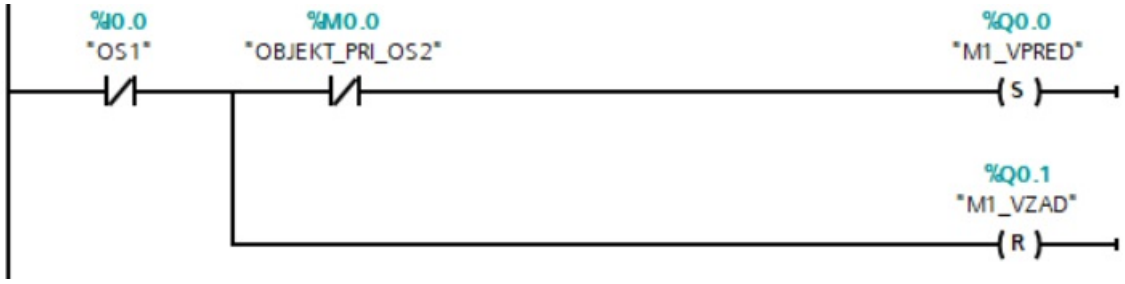


Obr. 6.35. Riešenie zadania 3 - Rebrík 2

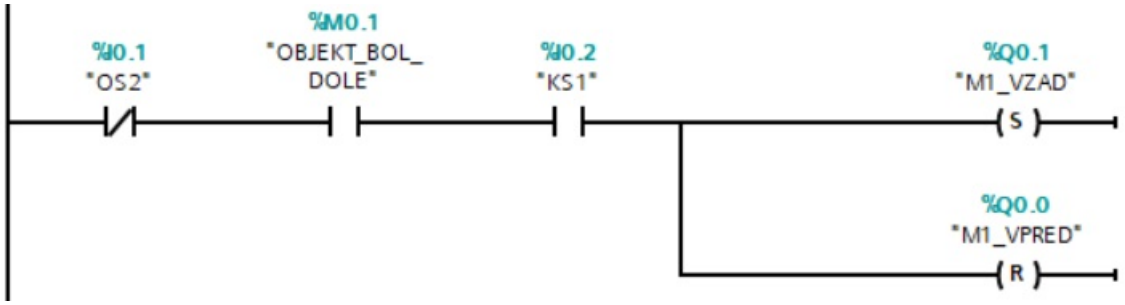


Obr. 6.36. Riešenie zadania 3 - Rebrík 3

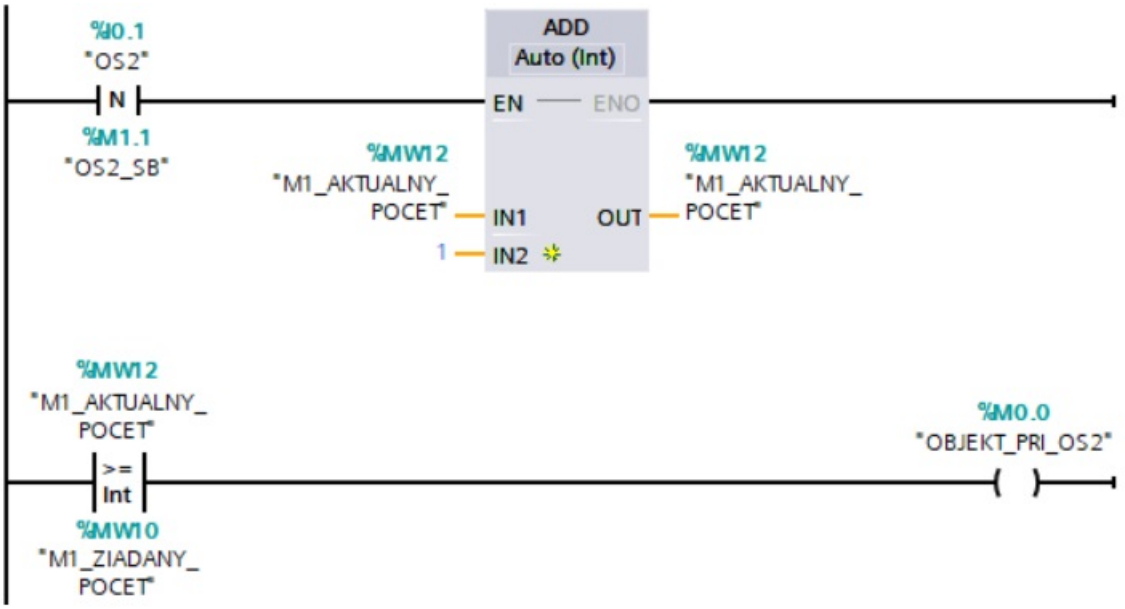
- Zadanie 4 - Príklad riešenia je na Obr. 6.37 až 6.43. Prepojenie predchádzajúcich riešení je v rebríku 2. Ak je objekt detegovaný optickým snímačom OS2, nemožno ho poslať na začiatok dopravníka bez toho, aby sa dokončilo lisovanie. Do AND podmienky sa pridala podmienka, že lis musel byť dole a zároveň sa vrátil do hornej polohy. Bez detekcie koncovej polohy snímačom KS1 by sa objekt začal pohybovať po dopravníku počas fázy lisovania, čo by v praxi viedlo k poškodeniu produktu, ale aj zariadenia. Zvyšné časti programu sú bez zmeny. Ošetrenie nenulového žiadaného počtu by bolo možné pridaním AND podmienky v spodnej vetve rebríka 3, kde by sa porovnávacím blokom testoval nenulový žiadaný počet opakovaní.



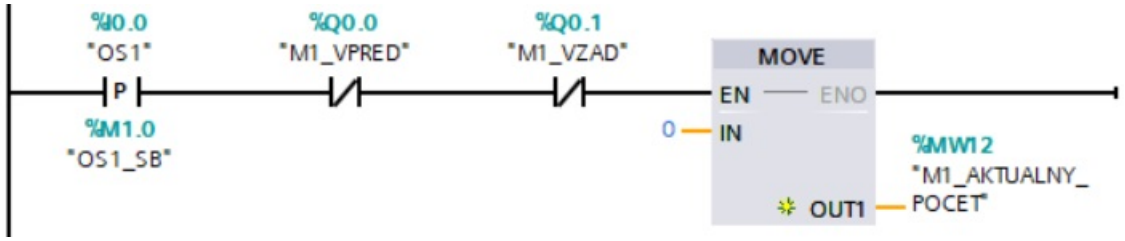
Obr. 6.37. Riešenie zadania 4 - Rebrík 1



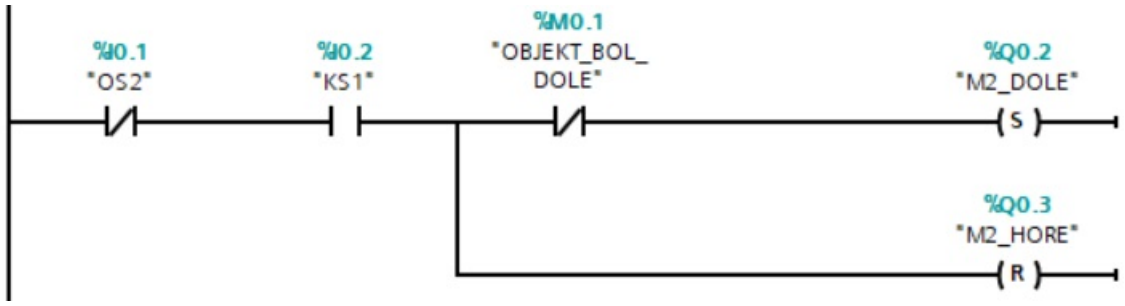
Obr. 6.38. Riešenie zadania 4 - Rebrík 2



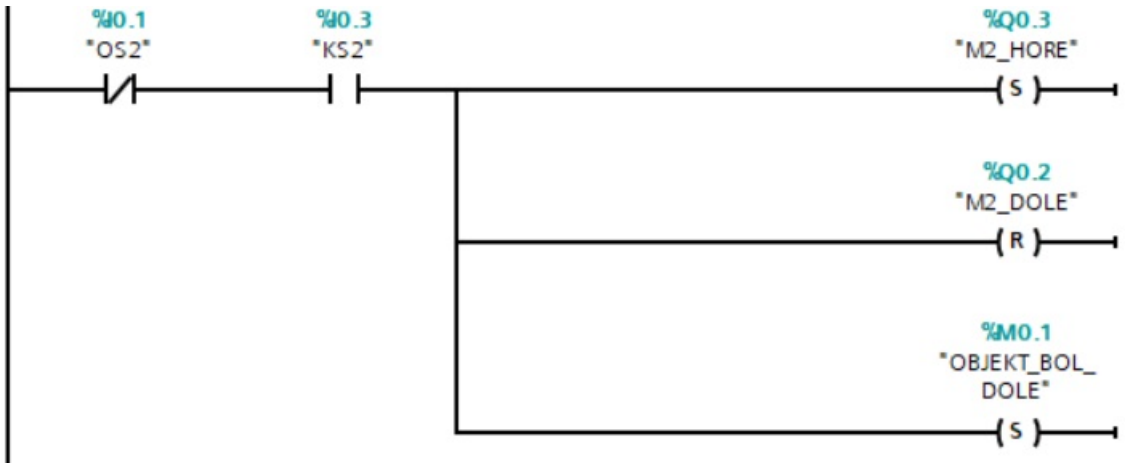
Obr. 6.39. Riešenie zadania 4 - Rebrík 3



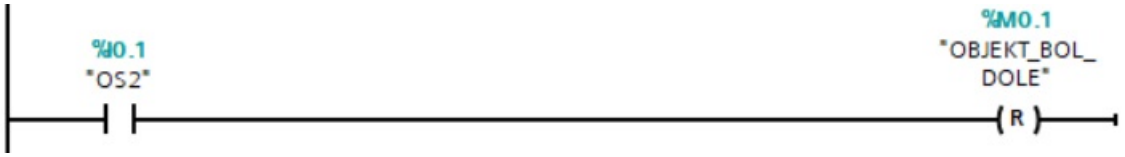
Obr. 6.40. Riešenie zadania 4 - Rebrík 4



Obr. 6.41. Riešenie zadania 4 - Rebrík 5



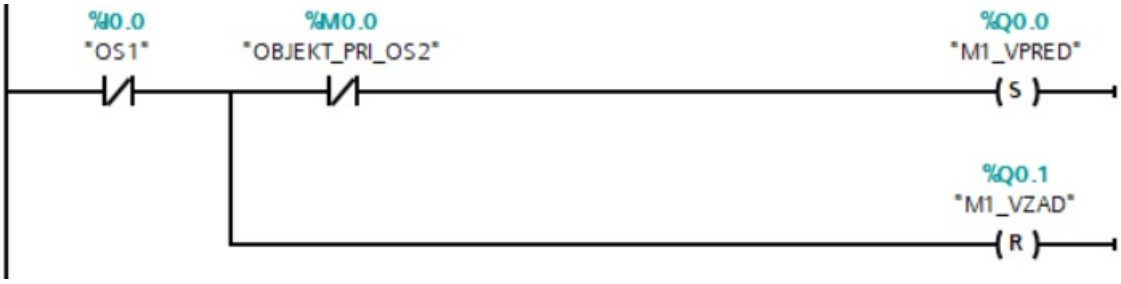
Obr. 6.42. Riešenie zadania 4 - Rebrík 6



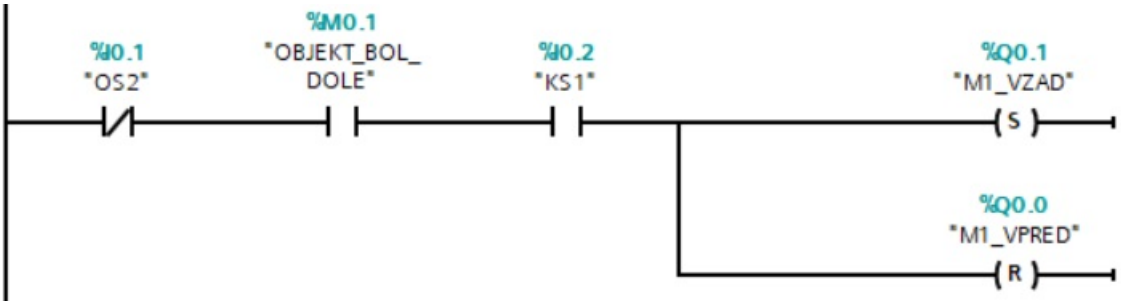
Obr. 6.43. Riešenie zadania 4 - Rebrík 7

- Zadanie 5 - Príklad riešenia je na Obr. 6.44 až 6.50. Hlavná modifikácia spočíva v zavedení nábežnej hrany KS2 spodného snímača, ktorý využívame na inkrementáciu počtu lisovaní inštrukciou ADD v 6. rebríku. V rebríku 7 hodnota premennej

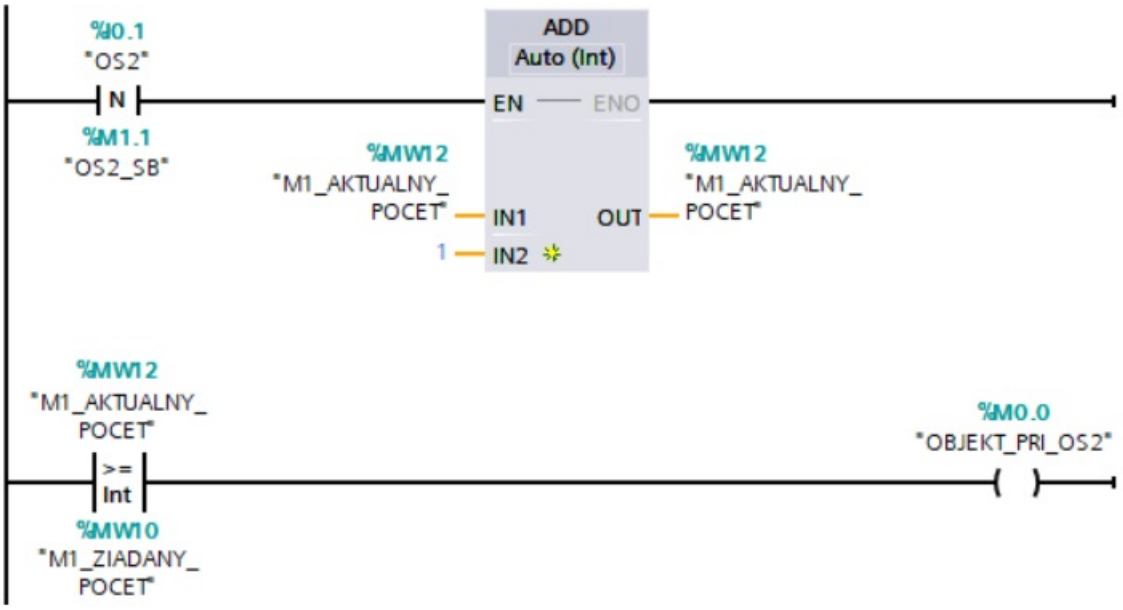
OBJEKT_BOL_DOLE závisí od dosiahnutia (resp. nedosiahnutia) požadovaného počtu lisovaní. Význam pomocnej premennej je teda vykonanie požadovaného počtu lisovaní. V 8. rebríku sa nuluje počet lisovaní po odchode objektu z OS2.



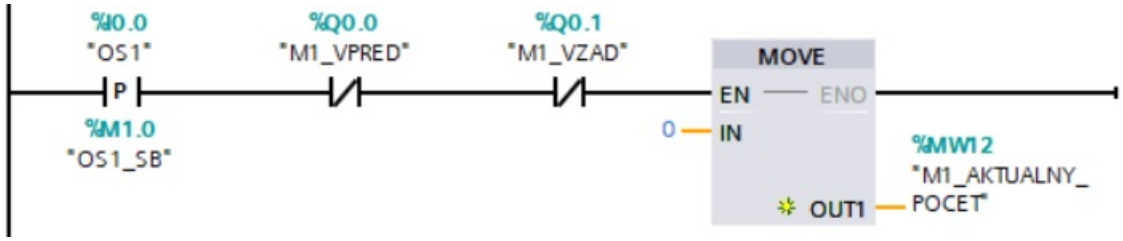
Obr. 6.44. Riešenie zadania 5 - Rebrík 1



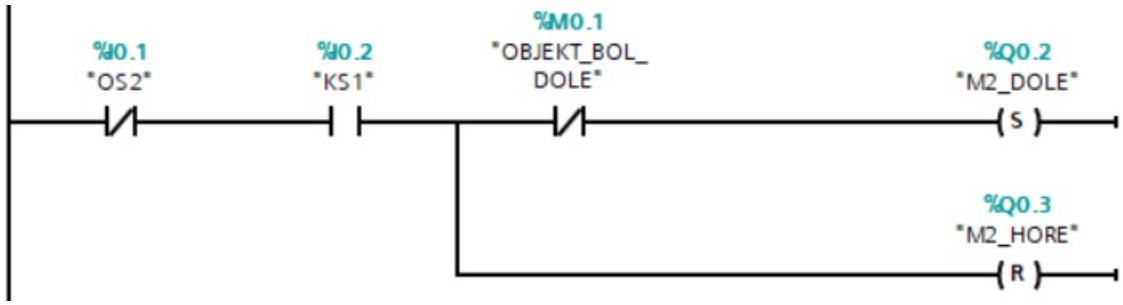
Obr. 6.45. Riešenie zadania 5 - Rebrík 2



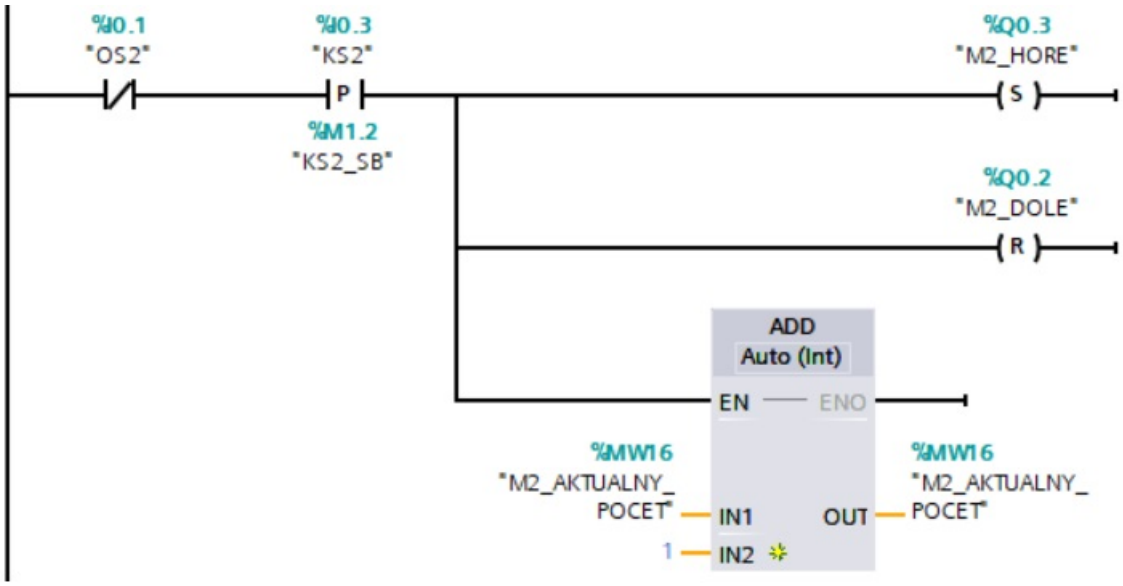
Obr. 6.46. Riešenie zadania 5 - Rebrík 3



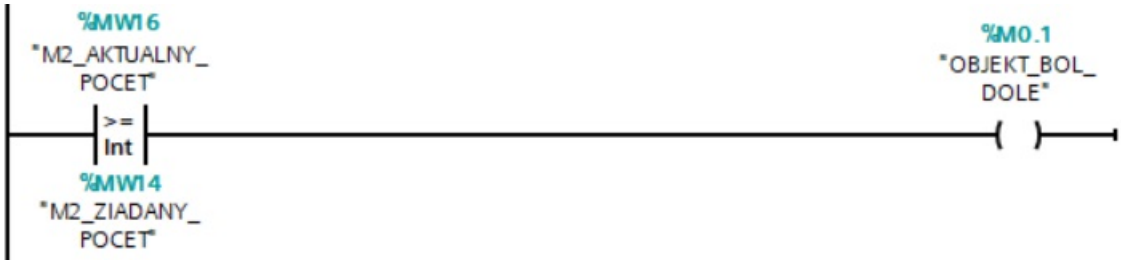
Obr. 6.47. Riešenie zadania 5 - Rebrík 4



Obr. 6.48. Riešenie zadania 5 - Rebrík 5



Obr. 6.49. Riešenie zadania 5 - Rebrík 6



Obr. 6.50. Riešenie zadania 5 - Rebrík 7

Obr. 6.51. Riešenie zadania 5 - Rebrík 8

Literatúra

[1] D. Bailey a E. Wright. *Practical SCADA for Industry*. 1st. Newnes, 2003, s. 304. ISBN: 0750658053.

[2] H. Berger. *Automating with SIMATIC S7-1500: Configuring, Programming and Testing with STEP 7 Professional*. 1st. Publicis, 2014, s. 831. ISBN: 3895784044.

[3] H. Berger. *Automating with SIMATIC S7-300 inside TIA Portal*. 1st. Publicis, 2012, s. 709. ISBN: 389578382X.

[4] H. Berger. *Automating with SIMATIC S7-400 inside TIA Portal*. 1st. Publicis, 2013, s. 746. ISBN: 3895783838.

[5] H. Berger. *Automating with SIMATIC: Controllers, Software, Programming, Data*. 5th. Publicis, 2013, s. 284. ISBN: 3895783870.

[6] W. Bolton. *Programmable Logic Controllers*. 5th. Newnes, 2009, s. 416. ISBN: 1856177513.

[7] L. A. Bryan a E. A. Bryan. *Programmable Controllers - Theory and Implementation*. 2nd. Amer Technical Pub, 2003, s. 1035. ISBN: 0826913008.

[8] P. Bubniak a L. Körösi. „Diagnostické funkcie PLC“. In: *Posterus [elektronický zdroj]* 7.2 (2014), online. ISSN: 1338-0087.

[9] J. R. Hackworth a F. D. Jr. Hackworth. *Programmable Logic Controllers - Programming Methods and Applications*. Prentice Hall, 2003, s. 320. ISBN: 0130607185.

[10] J. Hugh. *Automating Manufacturing Systems with PLCs*. 5th. Lulu.com, 2008, s. 352. ISBN: 0557344255.

[11] L. Körösi a L. Mrafko. „RsLogix5000 - Bitové inštrukcie“. In: *Posterus [elektronický zdroj]* 7.8 (2014), online. ISSN: 1338-0087.

[12] L. Körösi a L. Mrafko. „RsLogix5000 - Časovače a počítadlá“. In: *Posterus [elektronický zdroj]* 7.10 (2014), online. ISSN: 1338-0087.

[13] L. Körösi a L. Mrafko. „RsLogix5000 - Matematické inštrukcie“. In: *Posterus [elektronický zdroj]* 7.9 (2014), online. ISSN: 1338-0087.

[14] L. Körösi a L. Mrafko. „RsLogix5000 - Porovnávacie bloky“. In: *Posterus [elektronický zdroj]* 7.10 (2014), online. ISSN: 1338-0087.

[15] L. Körösi, L. Mrafko a M. Mrosko. „PLC and their Programming - 2. PLC, PAC, DCS - Who Whom?“ In: *Posterus [elektronický zdroj]* 4.10 (2011), online. ISSN: 1338-0087.

[16] L. Körösi a J. Paulusová. „Neural network for PLC“. In: *Technical Computing Bratislava 2014 [elektronický zdroj] : 22nd Annual Conference Proceedings. Bratislava* (CD-ROM, 2014).

[17] R. L. Krutz. *Securing SCADA Systems*. 1st. Wiley, 2005, s. 218. ISBN: 0764597876.

[18] B. G. Lipták. *Instrument Engineers' Handbook, Vol. 2: Process Control and Optimization*. 4th. CRC Press, 2005, s. 2464. ISBN: 9780849310812.

[19] Omron Asia Pacific Pte Ltd. *PLC Programming by OMRON*. Dostupné z `https://electrical-engineering-portal.com/download-center/books-and-guides/electrical-engineering/plc-programming-omron`. 1999.

[20] L. Mrafko, M. Mrosko a L. Körösi. „PLC a ich programovanie 1. Čo je to PLC ?“ In: *Posterus [elektronický zdroj]* 3.4 (2010), online. ISSN: 1338-0087.

[21] E. A. Parr. *Programmable Controllers - An engineer's guide*. 3th. Newnes, 2003, s. 448. ISBN: 075065757X.

[22] F. D. Petruzella. *Programmable Logic Controllers*. 5th. McGraw-Hill Education, 2016, s. 432. ISBN: 9780073373843.

[23] Schneider Electric. *EcoStruxure Control Expert - Program Languages and Structure Reference Manual*. Dostupné z `https://www.se.com/sk/sk/download/document/35006144K01000/`. 2025.

[24] K. Stouffer et al. *Guide to Industrial Control Systems (ICS) Security*. 2nd. Dostupné z `http://dx.doi.org/10.6028/NIST.SP.800-82r2`. National Institute of Standards and Technology, 2015, s. 247. ISBN: 0557344255.

doc. Ing. Ladislav Körösi, PhD., Ing. Leo Mrafko, PhD., doc. Ing. Eva Miklovičová, PhD.

PROGRAMOVATEĽNÉ LOGICKÉ AUTOMATY

Vydala Slovenská technická univerzita v Bratislave vo Vydavateľstve SPEKTRUM STU, Bratislava, Vazovova 5, v roku 2025.

Edícia skrípt

Rozsah 261 strán, 514 obrázkov, 4 tabuľky, 23,421 AH, 23,728 VH, 1. vydanie, edičné číslo 6236, vydané v elektronickej forme.

85 – 219 – 2025

ISBN 978-80-227-5509-2
DOI: 10.61544/UIBS9660