CSIKÓS SÁNDOR

PLC PROGRAMOZÁS

UNIVERSITY OF SZEGED
FACULTY OF ENGINEERING
SZEGED, 2014

PLC PROGRAMMOZÁS

Készült a

Instrument for Pre-accession Assistance (IPA)
HUSRB/1203/221/075 azonosítójú
JOINT DEVELOPMENT OF CURRICULA AND TEACHING MATERIALS OF MECHANICAL ENGINEER ON MSc LEVEL c. pályázat támogatásával

Írta:

Csikós Sándor

Szerkesztette:

Csikós Sándor

Lektorálta:

Dr. Gyeviki János PhD

Kiadta:
Szegedi Tudományegyetem, Mérnöki Kar – Szeged (MAGYARORSZÁG), 2014

ISBN xxx-xxx-xxx-xxx

TARTALOMJEGYZÉK

ELŐSZÓ 4

1. MI IS A PLC? 5

1.1. PLC-k akkor és most (A PLC-k rövid történelmi áttekintése) 5

1.2. Hogyan működik a PLC 5

1.3. A PLC-k felépítése 8
1.3.1. Diszkrét I/O modulok 9
1.3.2. Analóg I/O modulok 16

1.4. Programozási nyelvek 17

1.5. A PLC-k üzemmódjai 23

2. PROGRAMOZÁS RSLOGIX 5000 KÖRNYEZBEN..... 24

2.1. Bemenetek, kimenetek és alapműveletek 24
2.1.1. Kezdjünk programozni! 24
2.1.2. Rendszer tag-ek 34
2.1.3. I. Feladat: Garázsajtó 35
2.1.4. II. Feladat: Siló 38

2.2. Ideje PLC-zni! 41
2.2.1. Időzítő típusok és alkalmazásuk 41
2.2.2. III. Feladat: Forgalomirányítás 44

2.3. Kezdjünk összehasonlítani 50
2.3.1. Összehasonlító műveletek 50
2.3.2. IV. feladat: Forgalom irányítás 2. megoldás 53

2.4. Számoljunk egy kicsit 55
2.4.1. Számláló típusok és alkalmazásuk 55
2.4.2. V. feladat: Keverék elkészítése 58

2.5. Haladó szint 62
2.5.1. Matematikai utasítások 62
2.5.2. Logikai műveletek 66
2.5.3. Fájl műveletek 73

2.5.4. Program strukturálása **82**
2.5.5. VI. feladat: Palackozó gépsor **88**

ELŐSZÓ

A programozható logikai vezérlőket (PLC-k) már több mint 35 éve használják az iparban. Ma a legtöbb automatizált termelő rendszert PLC vezérel, és már felügyelő rendszerként is használják.

A PLC számos előnyös tulajdonságának (nagy megbízhatósága, stabilitása, felhasználóbarát programozó felülete kiegészítve érintőképernyőkkel) köszönhetően az ipari folyamatok vezérlése átláthatóbb, gazdaságosabb és megbízhatóbb.

Ez a könyv azért készült, hogy betekintést nyújtson a PLC-k belső felépítésébe, hogy képesek legyünk megtervezni és megírni egy jól működő programot **RSLogix 5000** környezetben.

1. MI IS A PLC?

A PLC-t magyarul Programozható Logikai Vezérlőt 1978-ban a „National Electrical Manufacturers' Association (NEMA)" a következőként definiálta 1978-ban:

"Egy digitálisan működő elektronikus eszköz, amely a memóriájába írt logikai- és aritmetikai utasítások, időzítők, számlálók segítségével, digitális vagy analóg bemenetei és kimenetei között teremt kapcsolatot, gépek és folyamatok irányítására."

1.1. PLC-k akkor és most (A PLC-k rövid történelmi áttekintése)

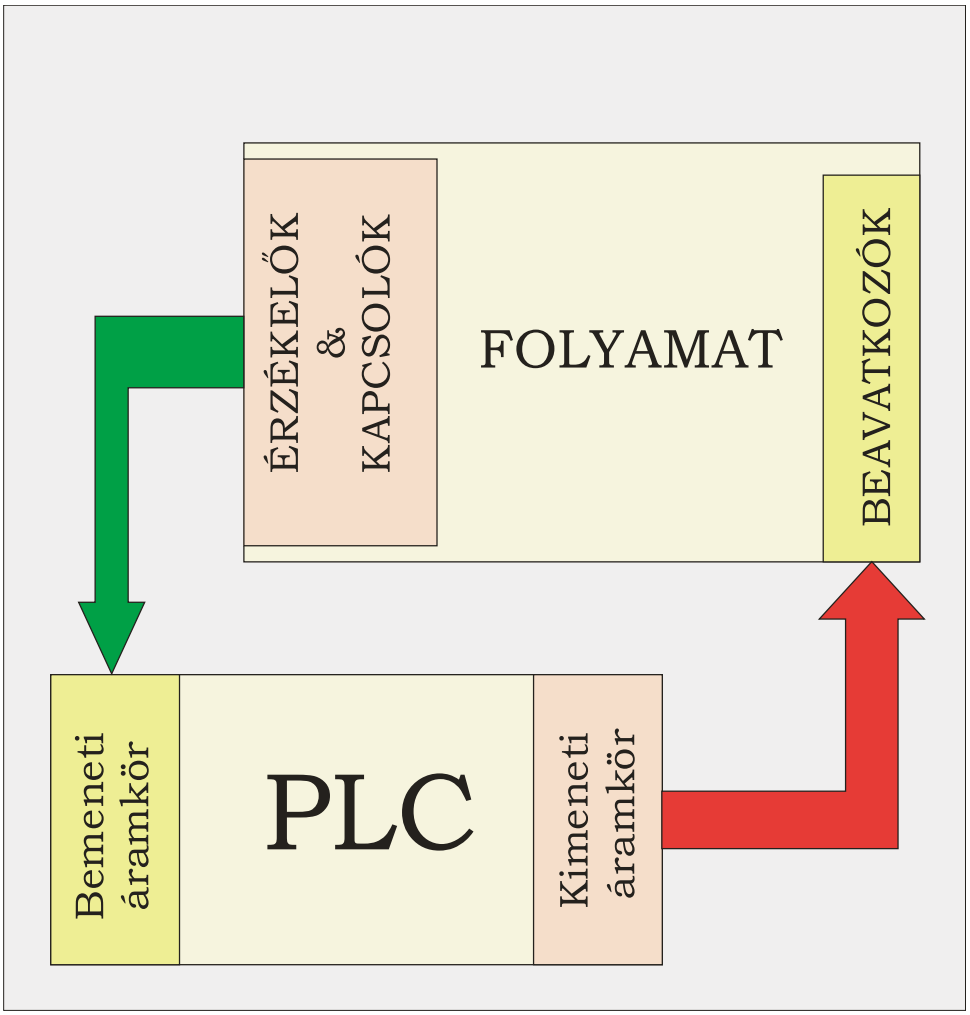
A PLC-k elterjedése előtt a gépek vezérlése huzalozott áramkörökkel történt. Ezekben a rendszerekben jelfogók, programdobok és egyedi szabályozók voltak. Sajnos nagy projektek esetén a jelfogók száma több ezret is elérte ezért a kapcsolószekrény nagyon nagy volt. A rendszer esetleges módosítása nagyon drága és időigényes volt.

Az amerikai autóipar volt az első, amely felvetette a jelfogós rendszerek lecserélésének az igényét. 1968-ban Bedford és társai Bedford Massachusettsben elkészítették az első PLC-t (a 084-et) a General Motors leányvállalatának a GM Hydramaticnak. A PLC kifejlesztésének célja a huzalozott rendszerek leváltása volt. Mivel a PLC-t az autóipar számára fejlesztették ki elengedhetetlen volt, hogy ipari körülmények között biztonságosan működjön (por, pára, szélsőséges hőmérsékleti viszonyok).

A PLC-k fejlesztésére, gyártására, forgalmazására és szervezésére Bedford és társai megalapították a MODICON (MOduláris DIgitális CONtroller) céget.

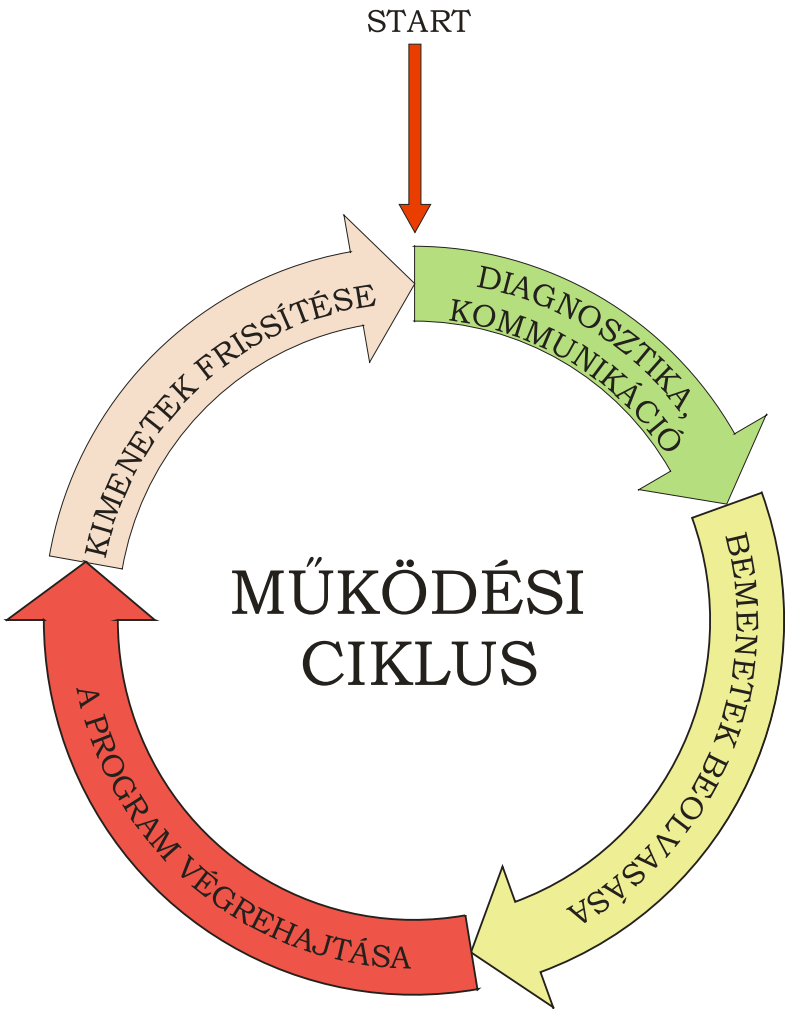
1.2. Hogyan működik a PLC

Kezdetben a PLC programozó felületét a létradiagram analógiájára alakították ki. Ezért a mérnökök és technikusok, programozói ismerek nélkül is elsajátíthatták a PLC programozást. Ez a módszer olyan sikeresnek bizonyult, hogy ma is az egyik legelterjedtebb PLC programozói nyelv. Amikor egy folyamatot PLC vezérel bemenőjelként használja az érzékelők jeleit, hogy döntést hozzon az aktuátorokat működtető kimenetek állapotáról (1. ábra). A folyamat egy valós folyamat, ami az idő függvényében változik. Az érzékelőkből érkező jelek határozzák meg a PLC működését. Az aktuátorok viszik a rendszert új állapotokba.



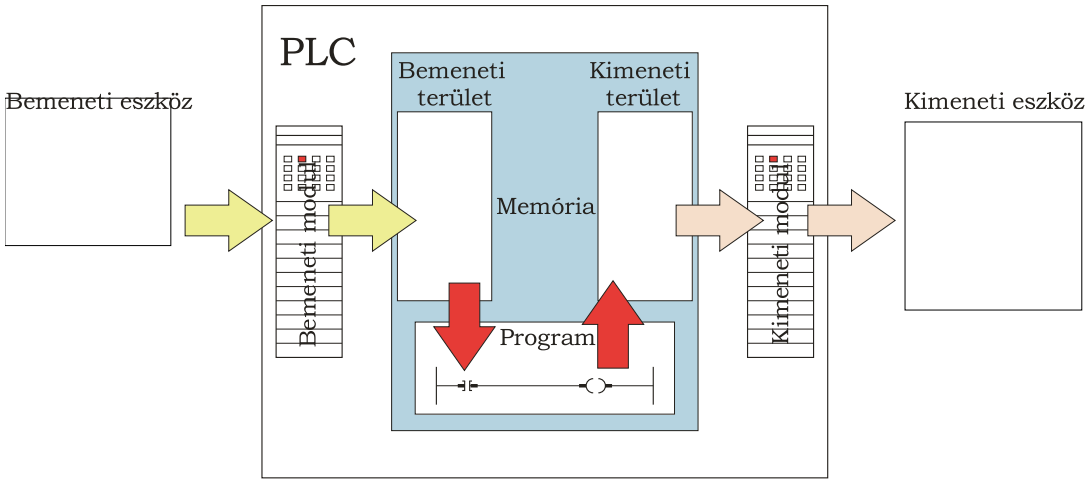
1. ábra A PLC kapcsolata az irányított folyamattal

A PLC-s vezérlés egy végtelen ciklus ahol a PLC beolvassa a bemeneteket, végrehajtja a létra logikát, majd az eredményt kiteszi a kimenetekre. Más számítógépekhez hasonlóan ez nagyon gyorsan játszódik le. A 2. ábrán a PLC munkaciklusát láthatjuk. Amikor a tápfeszültséget bekapcsoljuk egy inicializálási folyamat játszódik le, ahol a PLC megbizonyosodik arról, hogy minden részegysége helyesen működik-e. Hiba esetén a PLC leáll és jelzi a hibát. Például a tápfeszültség kimaradása is hibát eredményez.



2. ábra A PLC működési ciklusa

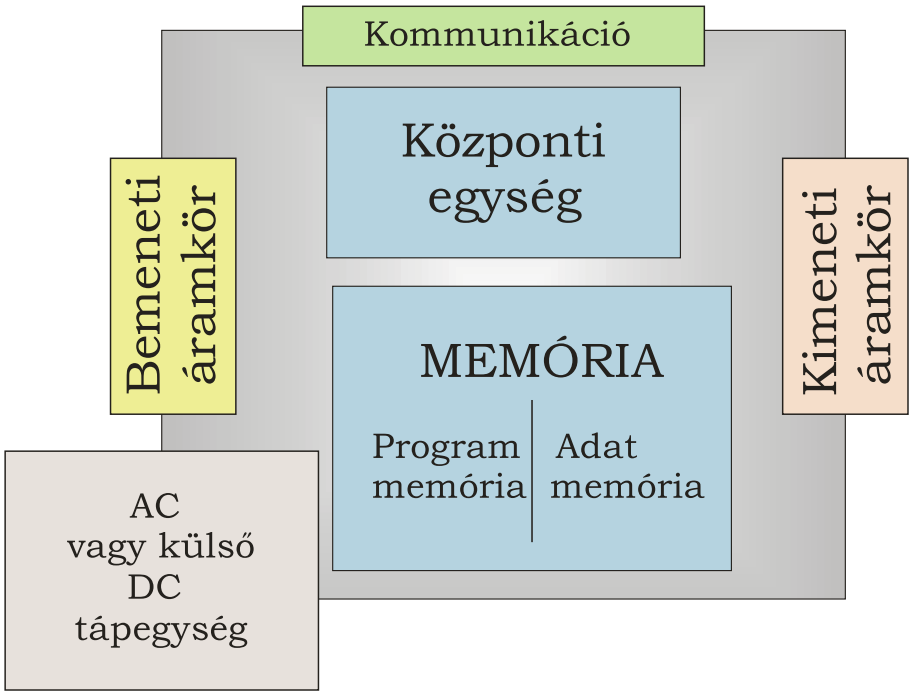
Amikor a PLC inicializálása befejeződik, megtörténik a bemenetek beolvasása. A bemenetek állapotát a PLC egy memóriaterületen (bemeneti térkép) tárolja le. A létradiagram végrehajtása közben a bemeneti értékeket erről a memóriaterületről veszi a PLC, és nem a bemenetek pillanatnyi értékét használja. Ez azért fontos, hogy a program futása közben megváltozott bemenet ne okozzon zavart. A program futása közben meghatározott kimeneti értékeket egy másik memóriaterületre (kimeneti térkép) tárolja, majd mikor a program lefutott egyszerre frissíti a kimeneteket (3. ábra). Ezek után a rendszer öndiagnosztikát hajt végre, majd a ciklus ismétlődik. A számítógépektől eltérően itt minden ciklusban a teljes program lefut. A ciklusidőt milliszekundumokban mérjük.



3. ábra A PLC működési folyamata

1.3. A PLC-k felépítése

A PLC öt fő részből áll (processzor, memória, tápegység, ki- és bemenetek és kommunikációs port), melyeket a 4. ábrán láthatunk.



4. ábra A PLC fő részei

A bemenetek és kimenetek állapotait RAM-ban a PLC utasításait pedig EPROMban tároljuk. A programot a PLC-re számítógéppel vagy kézi programozóval tölthetjük fel.

A memóriában tárolt programot a processzor dekódolja és végrehajtja.

A folyamatból érkező jeleket a PLC a bemenetein keresztül fogadja és értelmezi. A folyamatot befolyásoló aktuátorok a működtető jelet a kimenetekről kapják. Bemeneti eszközöknek nevezzük azokat az eszközöket, amelyek jelet vagy adatot szolgáltatnak a PLC-nek. Tipikus bemeneti eszközök: nyomógombok, kapcsolók, enkóderek és érzékelők. A kimeneti eszközök a PLC-től kapott jelek alapján végzik feladatukat. Tipikus kimeneti eszközök: mágneskapcsolók, jelzőlámpák, hangjelzők, motorok és szelepek. Ezeket gyűjtőnéven ki- és bemeneti, vagy I/O eszközöknek nevezzük. A kompakt PLC-k előre meghatározott I/O számmal rendelkeznek, míg a moduláris PLC-k I/O kártyákkal bővíthetők.

A PLC I/O pontjait, - melyek lehetnek diszkrétek vagy analógok - vásárláskor kell meghatározni. A diszkrét modulok két állapottal rendelkeznek (magas és alacsony) míg az analóg modulok véges számú értéket vehetnek fel, egy előre meghatározott tartományban. Ezen értékek számát az analóg modul felbontása határozza meg.

1.3.1. Diszkrét I/O modulok

Diszkrét bemeneti eszközöknek alaphelyzetben nyitott (N.O.) és/vagy alaphelyzetben zárt (N.C.) érintkezői lehetnek. Legegyszerűbb példa ezekre a nyomógomb, amit N.O. vagy, N.C. érintkezőkkel vásárolhatunk. Az alaphelyzet azt az állapotot jelenti, amikor az eszköz nincs működtetve. A diszkrét I/O modulokra magas/alacsony állapotokkal rendelkező eszközöket kapcsolunk, mint a végálláskapcsolók, nyomógombok, mágneskapcsolók, elektromechanikus jelfogók stb. Minden diszkrét I/O modulnak tápfeszültségre van szüksége. Az I/O modulokat ezen feszültségek alapján csoportosíthatjuk. A felosztást az 1. Táblában láthatjuk. Az I/O modulokon az egyes I/O pontok állapotát LED-ek jelzik.

1. **Tábla** Bemenetek, kimenetek feszültség-szintjei, a gyakoriság sorrendjében

Bemeneti modul	Kimeneti modul
12-24 V DC	120 V AC
100-120 V AC	24 V DC
10-60 V DC	12-48 V AC
12-24 V AC/DC	12-48 V DC
5 V DC (TTL)	5V DC (TTL)
200-240 V AC	230 V AC
48 V DC	
24 V AC	

Az egyes feszültségek alkalmazásának vannak előnyei és hátrányai is:

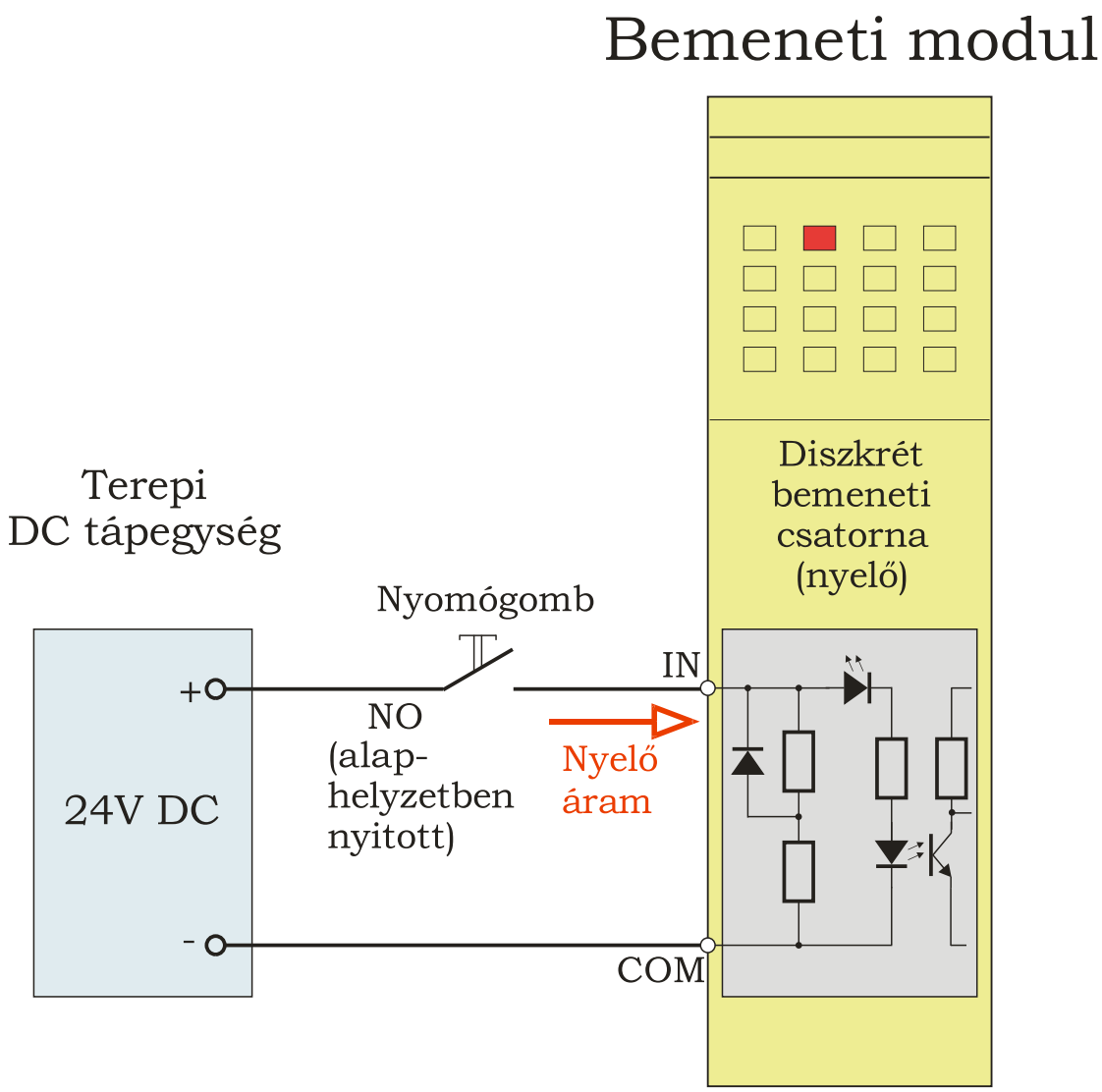
- Az alkalmazott egyenfeszültségek alacsonyabbak ezért biztonságosabbak (pl.: 12-24V).
- AC bemeneteknek több idő kell a jel észleléséhez, mint a DC bemeneteknek. Például az 50 Hz-es jel felismerése akár 1/50 s-ig is eltarthat.
- DC feszültségek alkalmazása elterjedtebb.
- AC jelek zajérzéketlenebbek, mint a DC jelek, ezért nagy távolságokon és zajos környezetekben jobban alkalmazhatóak.
- AC feszültségeket kisebb költséggel és könnyebben juttathatjuk el az alkalmazott berendezésekhez.
- Nagyobb teljesítményű eszközöknél az AC feszültség elterjedtebb.

Bemeneti modul:

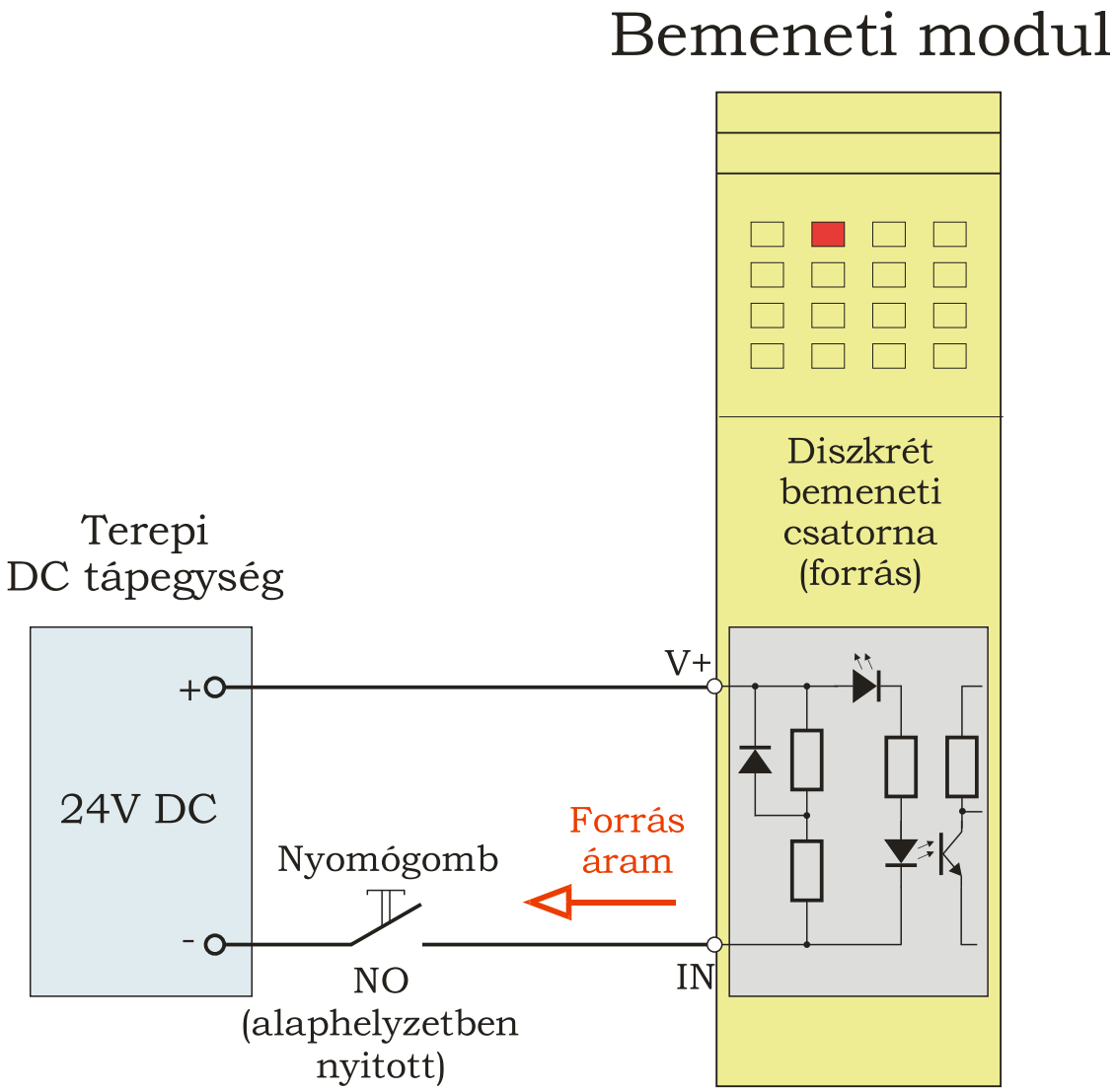
A bemeneti modul összeköti a bemeneti csatlakozó-pontokat a rendszer többi részével. A bemeneti pontokat a belső áramkörtől optocsatolók választják el galvanikusan. Az optocsatolóban a jelet egy LED fénye adja át a fototranzisztornak. Így védik az optocsatolók a PLC-k belső áramkörét a káros feszültségcsúcsoktól. A megengedettnél nagyobb bemeneti feszültség hatására a bemeneti modul megsérül, de az optocsatolónak köszönhetően a PLC belső áramköre nem károsodik. Az optocsatolók akár 10 kV feszültségig is védelmet nyújtanak.

Mivel a bementi modulok nem adnak tápfeszültséget, a bemeneti eszközöket külső tápról kell működtetni. Ha a rendszerben független tápfeszültségeket alkalmazunk, akkor kell, hogy legyen egy közös nullpontjuk (0 V-al jelölt csatlakozó). A diszkrét bemeneti moduloknak két típusa létezik nyelő (sinking) és forrás (sourcing). A nyelő és forrás kifejezés az áram irányára utal.

A nyelő és forrás elnevezés mellett a PLC bemeneti pontjainak jellemzésére gyakran használatos a bemenetre csatolt kontaktus nélküli közelítéskapcsoló típusa, ami lehet PNP vagy NPN. Nyelő (PNP) modulokra olyan bemeneti eszközt (közelítés kapcsolót) kell csatlakoztatni, amely áramot ad (5. ábra). Míg a forrás (NPN) típusúak olyan eszközt igényelnek, ami nyeli az áramot (6. ábra).



5. ábra Nyelő (PNP) bemeneti modul



6. ábra Forrás (NPN) bemeneti modul

Kimeneti modul:

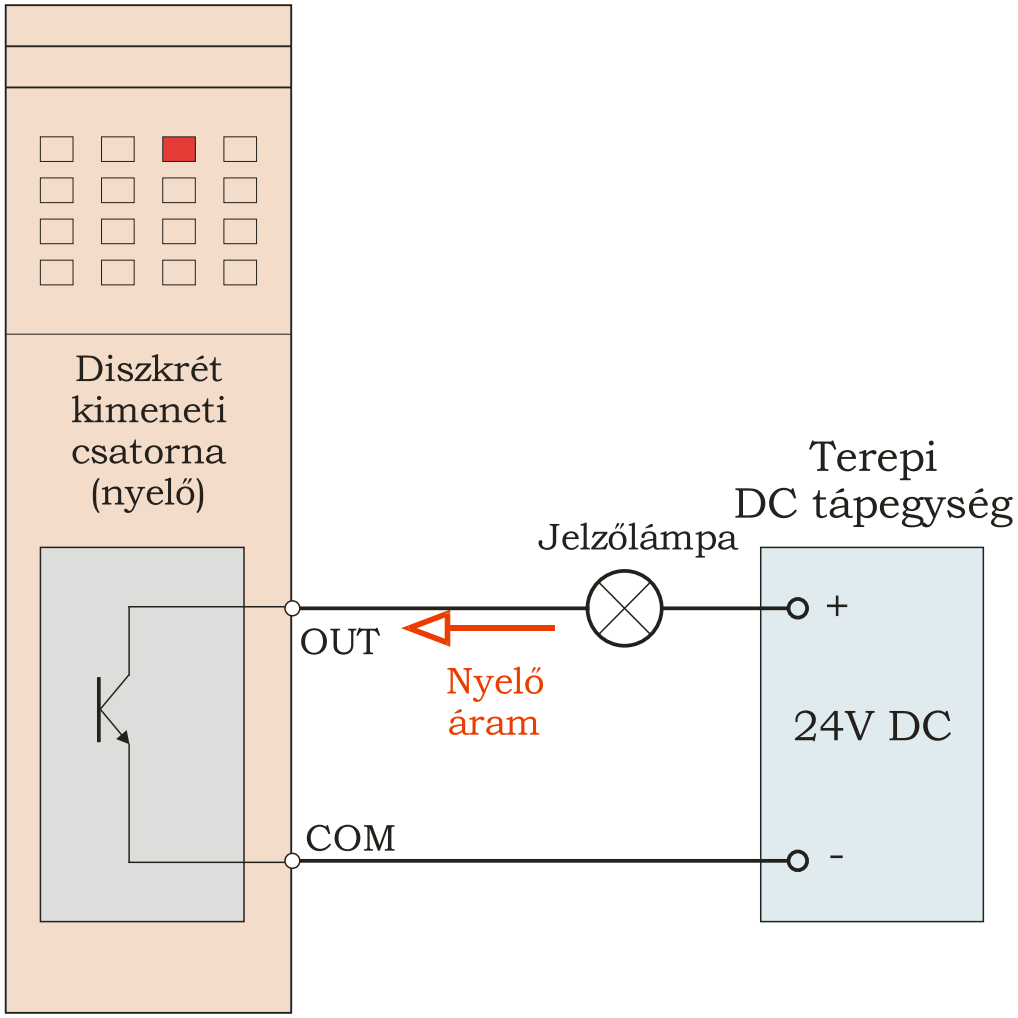
A kimeneti modul feszültséget kapcsol a hozzá csatlakozó kimeneti egységekhez. A kimeneti modul lehet érintkezős (relés), tanzisztoros vagy triakos. A tranzisztoros csak egyenfeszültséget tud kapcsolni, a triakos csak váltót, az érintkezős mindkettőt. A kimeneti modul túlterhelését el kell kerülni. A bemeneti modulokhoz hasonlóan a kimenetek száma is véges. A PLC és a kimenetéhez kapcsolt külső eszköz általában optocsatolóval galvanikusan el van választva, kivételt tesznek a jelfogós kimenetek ahol a jelfogó képezi a galvanikus leválasztást.

A bemeneti modulokhoz hasonlóan a kimeneti modulok sincsenek ellátva feszültségforrással, így egy külső feszültségforrás szükségeltetik. Független feszültségforrások esetén szükséges egy közös nullpont (0 V-al jelölt csatlakozó). A bemeneti modulokhoz hasonlóan diszkrét tranzisztoros kimeneti moduloknak is két típusát különböztetjük meg, forrás (PNP) és nyelő (NPN). Vegyük észre, hogy a

kimeneten a PNP forrás, míg a bemeneten nyelő. Az eltérés oka az, hogy a bemenetnél a rákapcsolt közelítéskapcsoló tranzisztorát tekintjük mérvadónak.

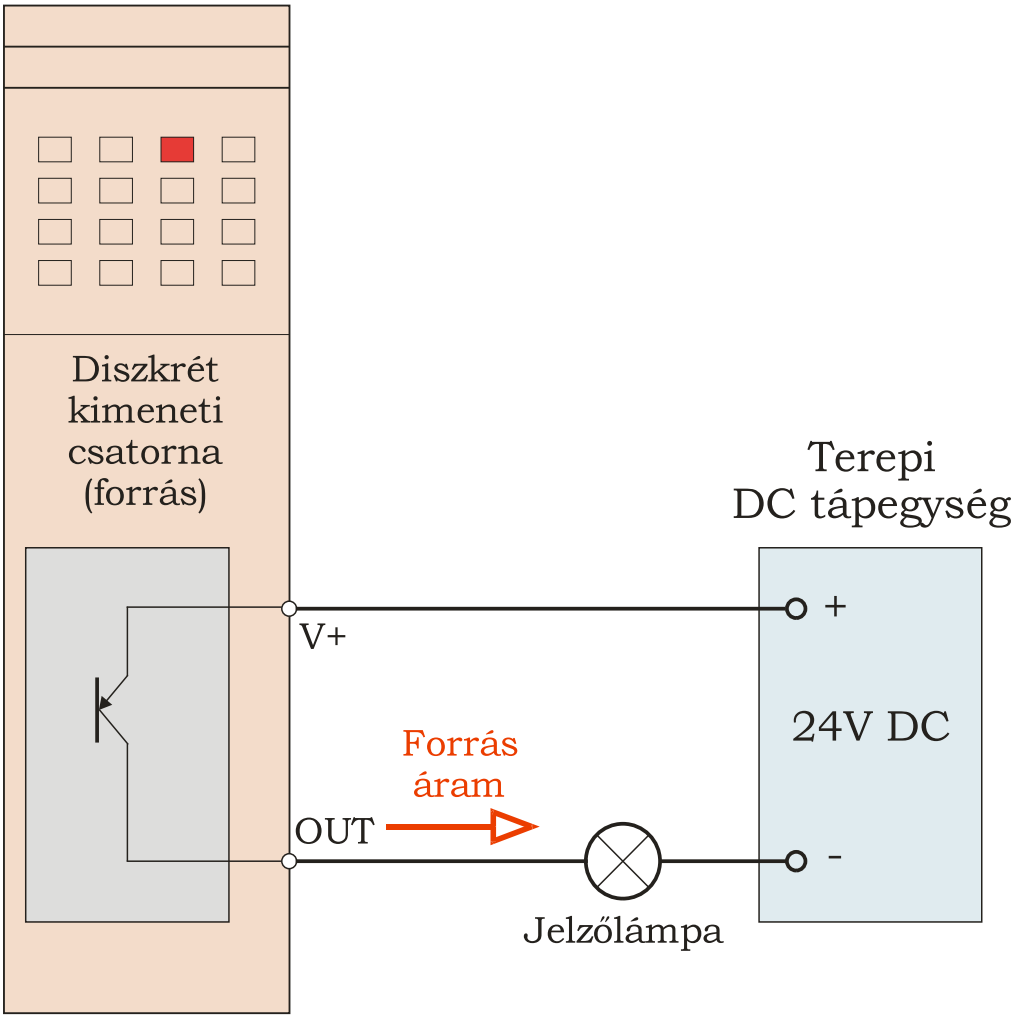
A nyelő kimenetnél (NPN) a terhelésen keresztül a kimenet felé folyik az áram (7. ábra). A forrás (PNP) kimenetből áram folyik a terhelés felé (8. ábra).

Kimeneti modul

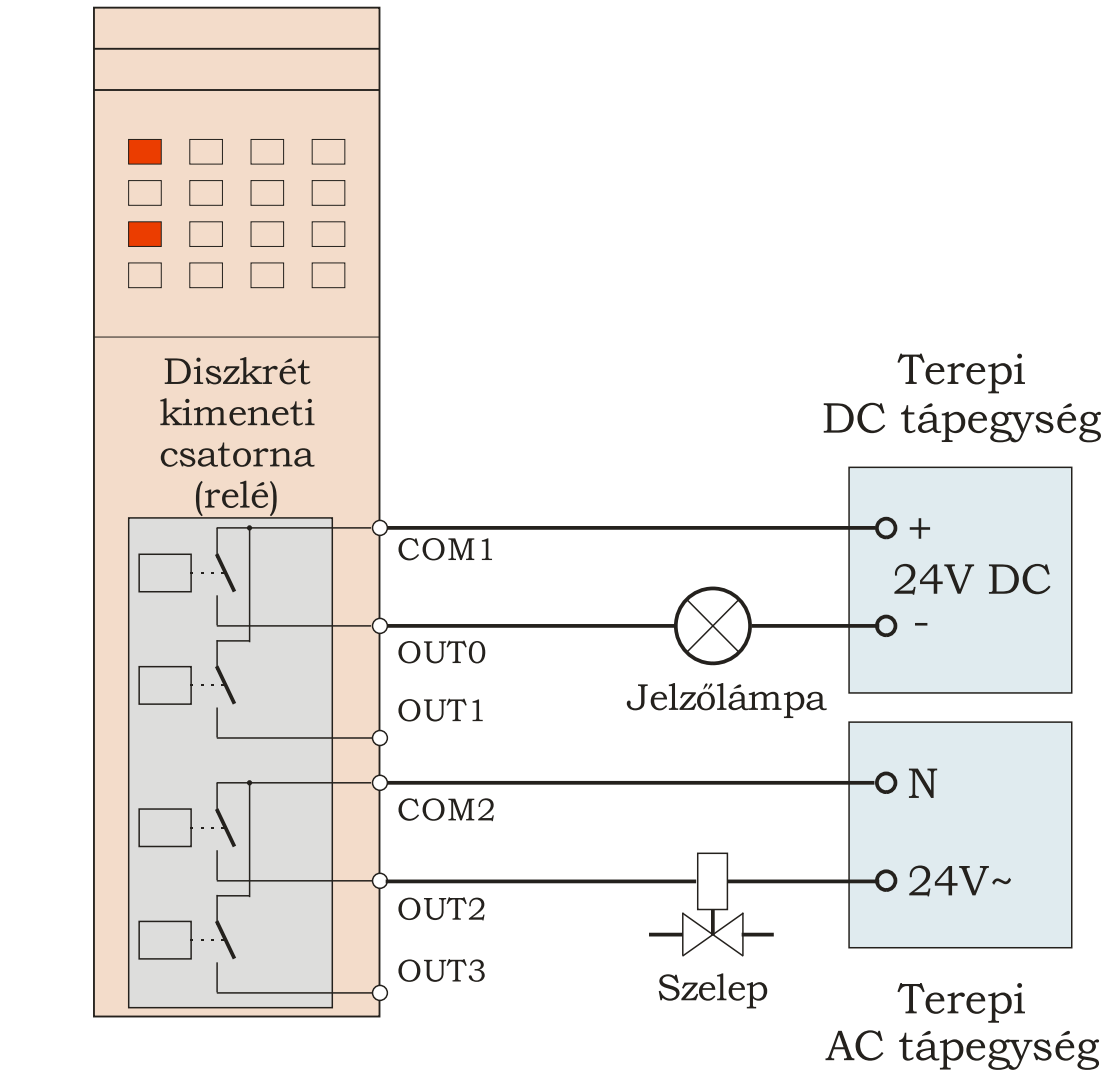


7. ábra Nyelő (NPN) kimeneti modul

Kimeneti modul



8. ábra Forrás (PNP) kimeneti modul

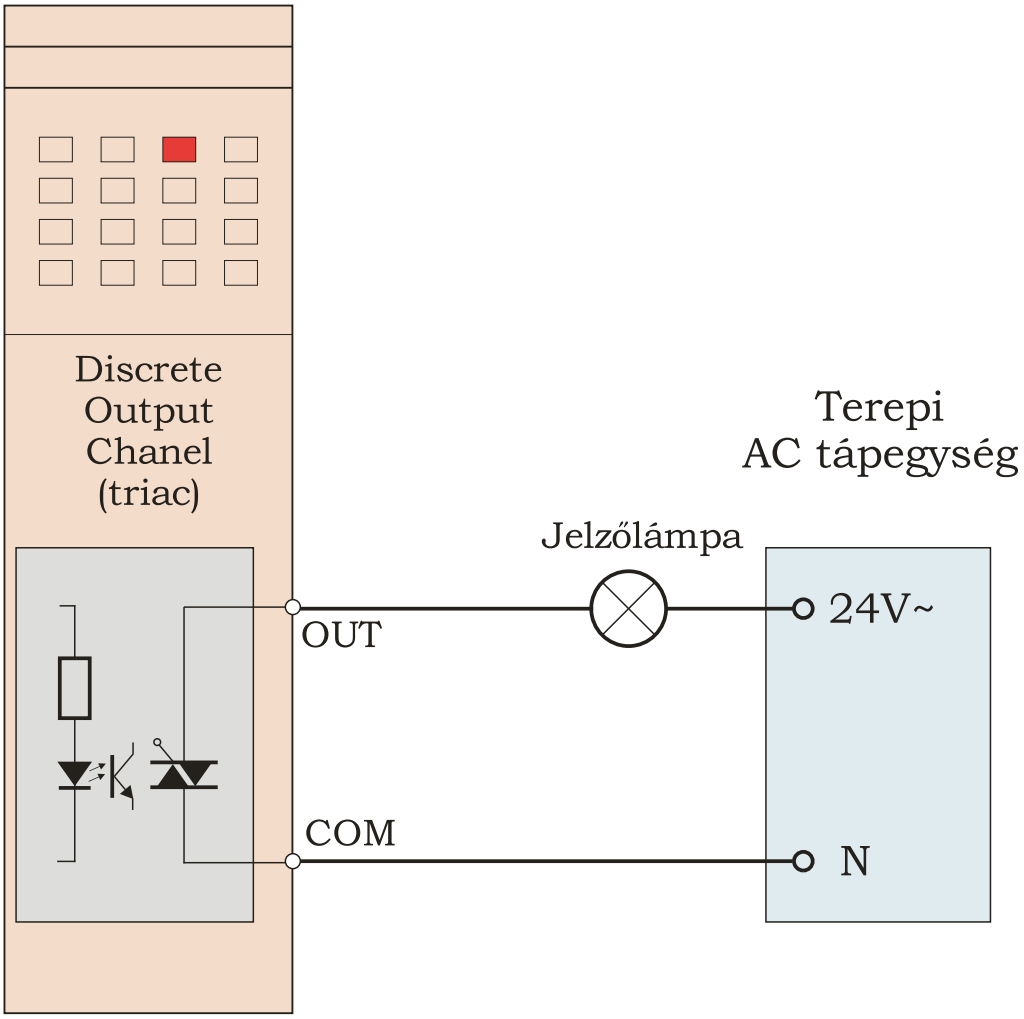


9. ábra Relés kimeneti modul

A relés kimeneti modulok AC és DC feszültségek kapcsolására is alkalmasak. A relés kimeneti modul érintkezőit csoportosítani szokták. Így a PLC különböző nagyságú és típusú feszültségek kapcsolására is alkalmas. Ezeket a modulokat könnyen fölismerhetjük rájuk jellemző kattogásukból. Ezek univerzálisak, de egyes alkalmazásoknál a relé lassúsága hátrányos (9. ábra).

Triakos modulokat AC feszültségek nagysebességű kapcsolására alkalmazzák (10. ábra).

Kimeneti modul



10. ábra Triakos kimeneti modul

1.3.2. Analóg I/O modulok

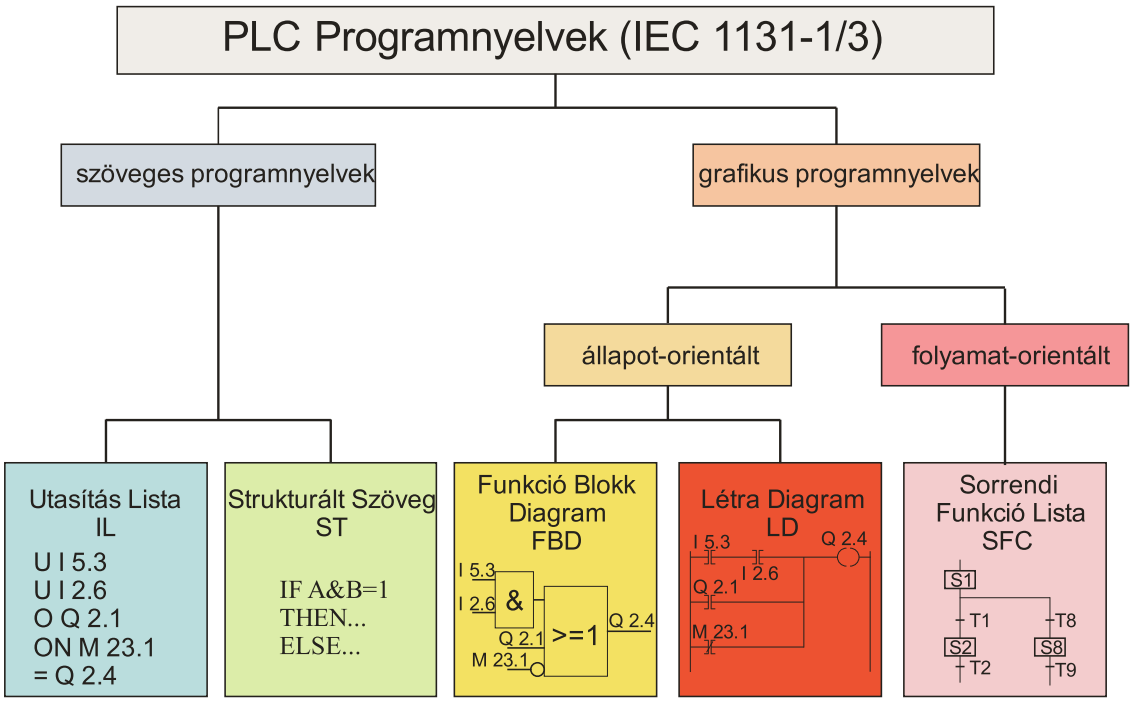
Mivel a PLC csak digitális értékekkel tud dolgozni a PLC bemenetére kötött analóg jeleket egy analóg-digitális átalakítóval (ADC) digitalizáljuk. A PLC kimenetére küldött digitális értéket digitál-analóg átalakítóval (DAC) alakítjuk át analóg értékre. Az analóg bemeneteket és kimeneteket a felbontásukkal jellemezzük. Ha a modul felbontása n bit, az állapotainak száma 2^n. A tartományt ezzel az értékkel kell elosztanunk, hogy megkapjuk 1 bit értékét. Tipikus analóg I/O tartományok 0-10 V, 0-20 mA, 4-20 mA. Abban az esetben, ha az analóg modulunk tartománya 0-10 V és felbontása 10 bit, 1024 állapottal rendelkezik. Tehát 1 bit értéke 0.009766 V.

1.4. Programozási nyelvek

PLC programozásra számos nyelv létezik mivel minden gyártó saját nyelvet fejlesztett ki. A nyelveket szabványos nyelvcsaládokba sorolhatjuk. Ezek:

- Structured text (**ST**)
- Instruction list (**IL**)
- Ladder diagram (**LD**)
- Function block diagram (**FBD**)
- Sequential function chart (**SFC**)

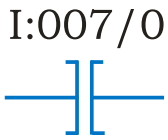
Leszámítva a "struktured text" és "sequential function chart" nyelveket a többi nyelv áramköri működés alapján értelmezhető. Ez a programozás stílusában tükröződik. Adott feladathoz kiválasztható az arra legalkalmasabb nyelv. A programozási nyelvek összefoglalását a 11. árán láthatjuk. A korszerű PLC-k több nyelvet is ismernek és ezek kombinálása és a közöttük való átjárás is megengedett.



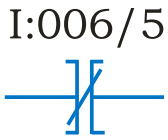
11. ábra Programnyelvek

A programozás alapjait létra logika (**LL**) nyelven mutatjuk be, ami az **LD** nyelv egyszerűsített változata. Bonyolultabb feladatok megoldásánál áttérünk az **LD** nyelv használatára. Az **LL** diagramot a jobb- és baloldali tápsínek közé rajzoljuk. A bemeneti utasítások a baloldalon helyezkednek el, a kimeneti utasítások a jobboldalon. Ha az **LL** diagramban létrejön a logikai folytonosság, a kimenet aktívvá válik. Logikai folytonosságnak azt nevezzük, ha minden bemeneti feltétel teljesül.

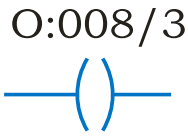
12.-14. ábrákon láthatjuk az **LL** nyelv utasításait. Minden utasításhoz egy memóriahely van hozzárendelve.



12. ábra Alaphelyzetben nyitott érintkező (bemeneti utasítás)



13. ábra Alaphelyzetben zárt érintkező (bemeneti utasítás)



14. ábra Tekercs (kimeneti utasítás)

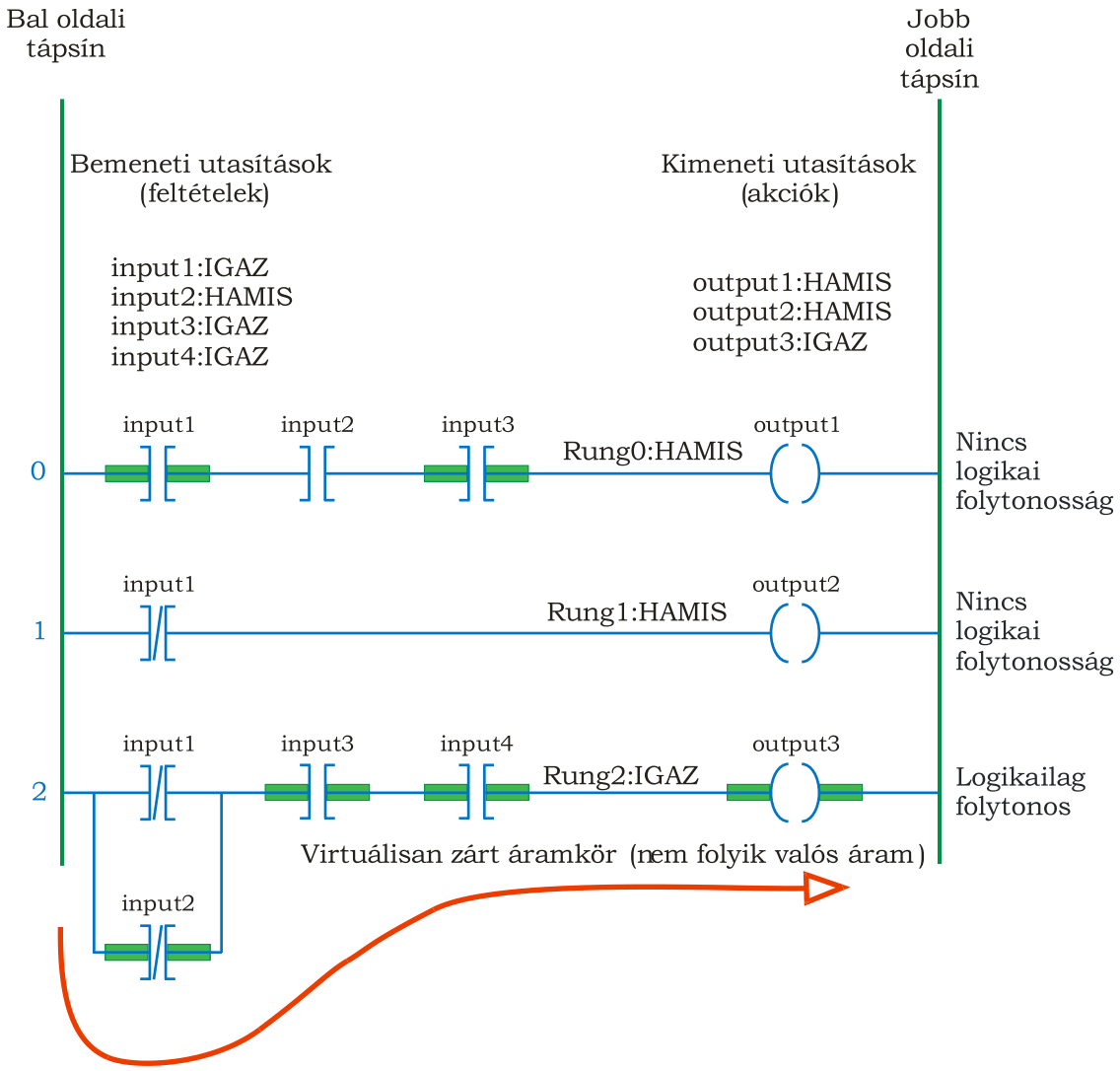
Az alaphelyzetben nyitott utasítás eredménye akkor **IGAZ,** ha a memóriából (bemeneti térkép) **1** értéket olvasott ki, egyébként **HAMIS**.

Az alaphelyzetben zárt utasítás megfordítja a memóriában talált értéket, azaz akkor **IGAZ,** ha a memóriából **0** értéket olvasott ki, egyébként **HAMIS**.

A tekercs a létraág eredményét a memóriába teszi (kimeneti térkép). Logikai folytonosság esetén ez **1**, egyébként **0**.

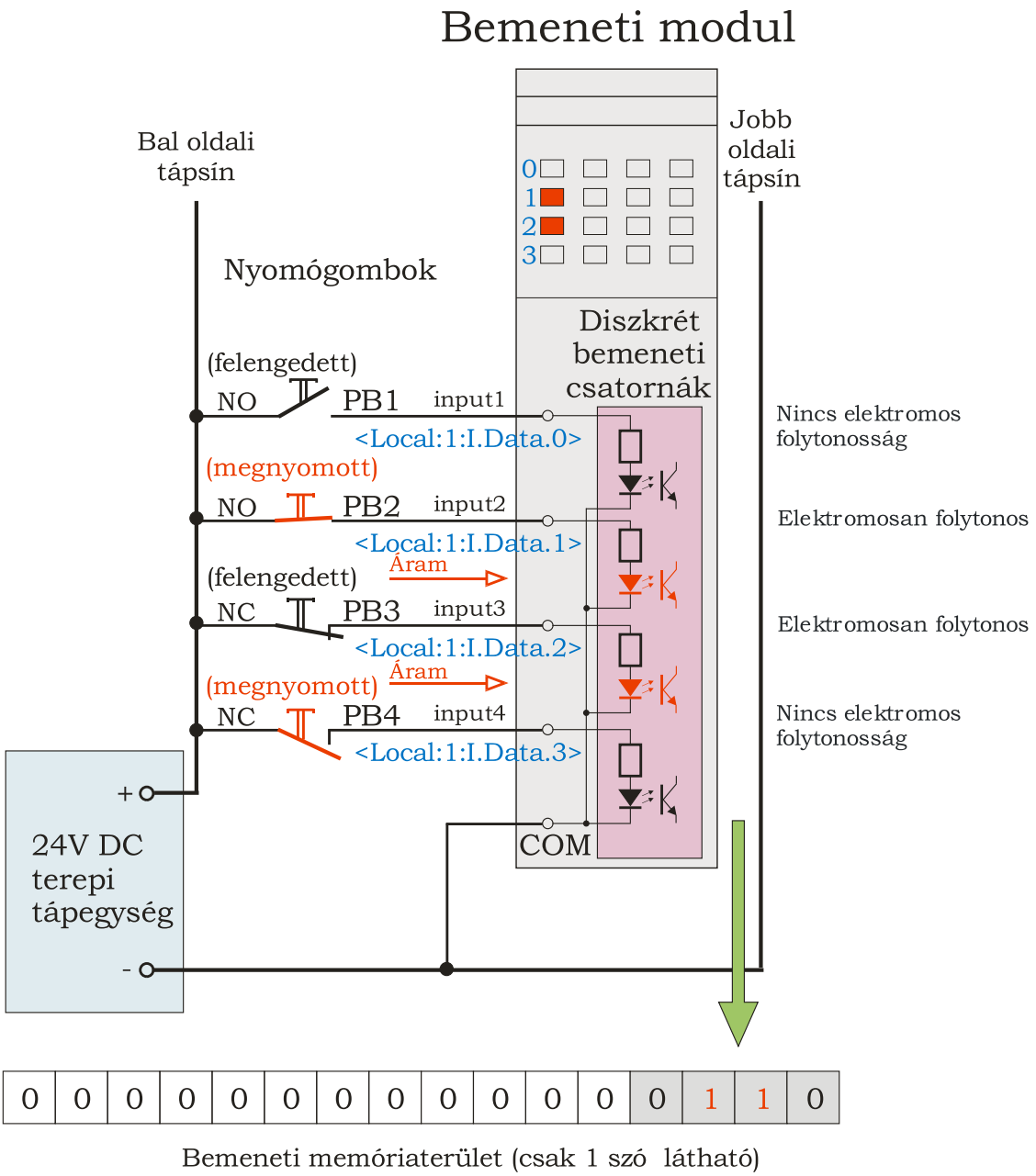
A program szemléletességét növeli, az hogy az **IGAZ** utasításokat zölden jelöli. A 15. ábrán az **LL** programra láthatunk egy példát. A baloldali áramsínt fázissínnek is szokták nevezni, a jobboldalit pedig 0-nak. A 15. ábrán különböző bemeneti kombinációk láthatók. A bemeneti értékek jeladókról jönnek. A kimenet valamilyen eszközt működtet.

A felső ágban nincs logikai folytonosság mivel **input2** hamis. A középső ágon egy alaphelyzetben zárt utasítást látunk, amin nincs logikai folytonosság. A legalsó ágon egy példát látunk a logikai folytonosságra.



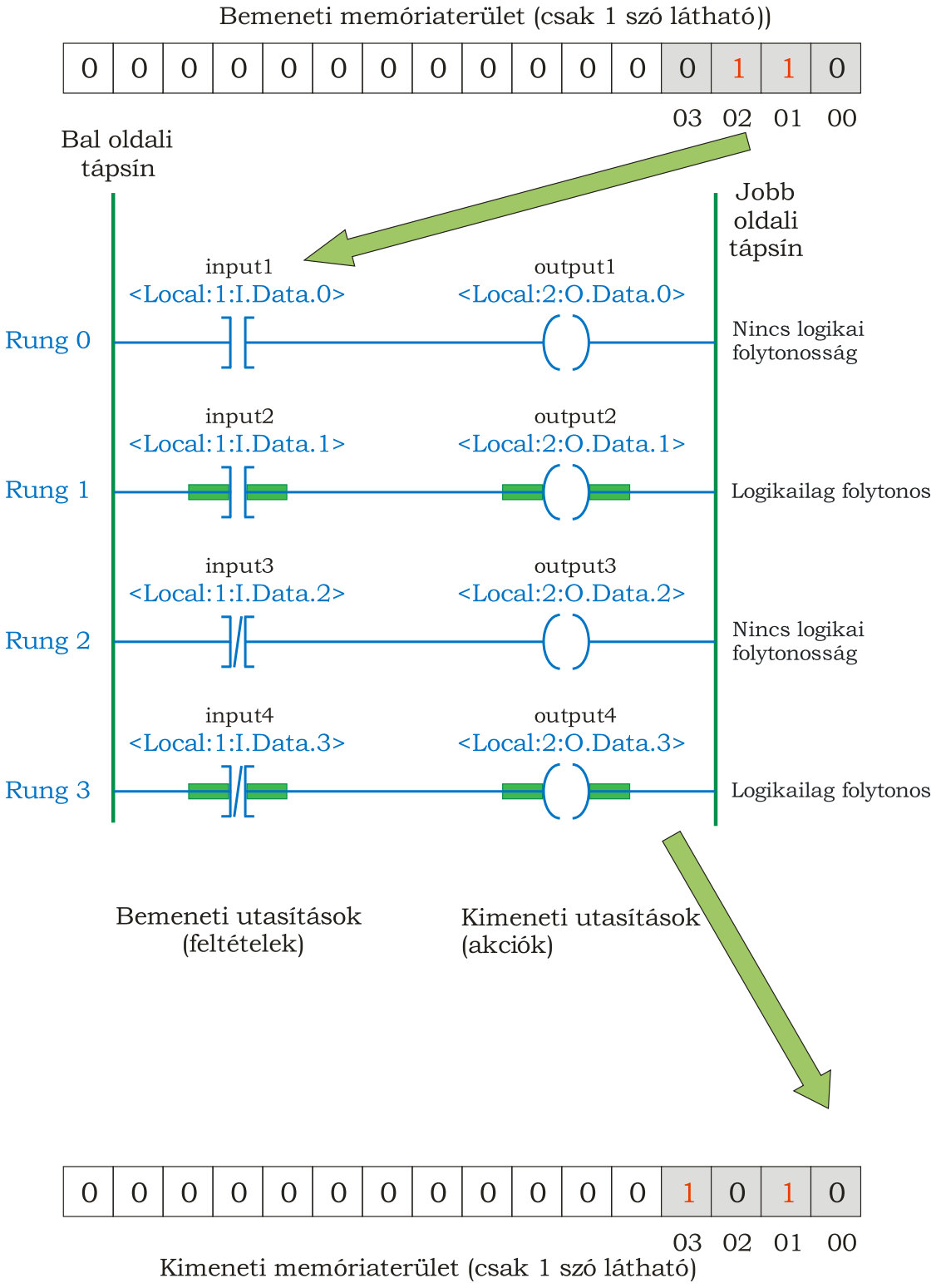
15. ábra Egy **LL** példaprogram

A 16. ábrán látható PLC bemenetre két darab **NO**, és két darab **NC** nyomógombot kötöttünk. Ahol a bemeneten elektromos folytonosságot tapasztalunk ott a bemeneti memóriaterületre írt érték **1**. Ahol nincs elektromos folytonosság, a memóriaterületre **0** íródik.



16. ábra Bemenetek beolvasása

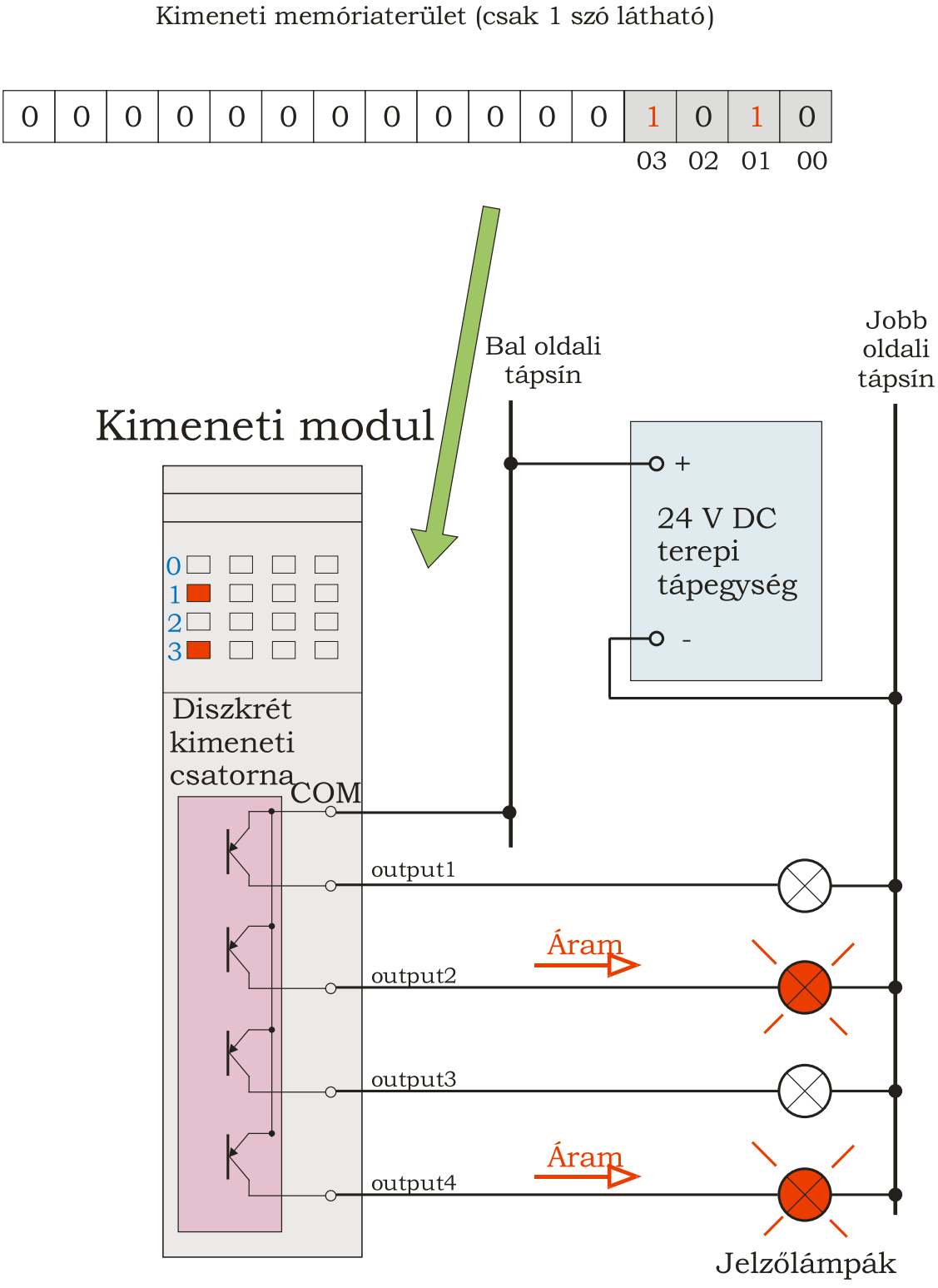
A PLC program logika kapcsolatot teremt a bemenetek és kimenetek között. A program futásakor a PLC a bemeneti értéktáblából kiolvasott értékek felhasználásával határozza meg a kimeneti értéktábla értékeit (17. ábra).



17. ábra A PLC program futása

Minden egyes diszkrét kimenethez tartozik egy hely (bit) a kimeneti memóriatáblában. Ahhoz, hogy egy kimenet aktívvá váljon, a hozzá rendelt bitet **1**-

be kell állítani. A kimenetek frissítése során a PLC a kimeneti tábla értékeit átmásolja a kimenetre (18. ábra).



18. ábra A kimenetek frissítése

1.5. A PLC-k üzemmódjai

A PLC-nek két üzemmódja létezik. Az üzemmódokat a PLC-n valamint a programozó környezetben lehet beállítani. Az üzemmódok a következők:

- **Program**: Ebben az üzemmódban a programot tölthetünk fel a PLC-re. A program futása nem indul el.
- **Futás**: Ebben az üzemmódban a feltöltött program fut és újabb programfeltöltés nem kezdeményezhető.

Attól függően, hogy mit szeretnénk elérni, a PLC-t mindig a megfelelő üzemmódba kell állítanunk. A PLC programozói környezetében is több üzemmód közül választhatunk. Ezek az üzemmódok:

- **Online:** A PLC és a programozó számítógép összekapcsolódik. Abban az esetben, ha a PLC **Program** üzemmódban van, program feltöltést, vagy letöltést kezdeményezhetünk, ha **Futás** üzemmódban van, monitorozhatjuk a PLC működését.
- **Offline:** Ebben az üzemmódban írhatunk programot és módosíthatjuk azt.

2. PROGRAMOZÁS RSLOGIX 5000 KÖRNYEZBEN

2.1. Bemenetek, kimenetek és alapműveletek

Ebben a fejezetben áttekintjük, az alapműveleteket, és azt, hogy hogyan írunk egy egyszerű programot. Feltételezzük, hogy a PLC és a számítógép közötti kommunikációt már az **RSLinx** program segítségével megteremtettük.

Az alapműveletek között 2 bemeneti és 3 kimeneti utasítás van. Ezek a következők:

Bemeneti utasítások:

- Examine If Open (**XIO**) (Bemenet negált beolvasása): A művelet értéke akkor lesz **1** ha a memória cellában **0** található.
- Examine If Closed (**XIC**) (Bemenet beolvasása): A művelet értéke akkor lesz 1 ha a memória cellában **1** található.

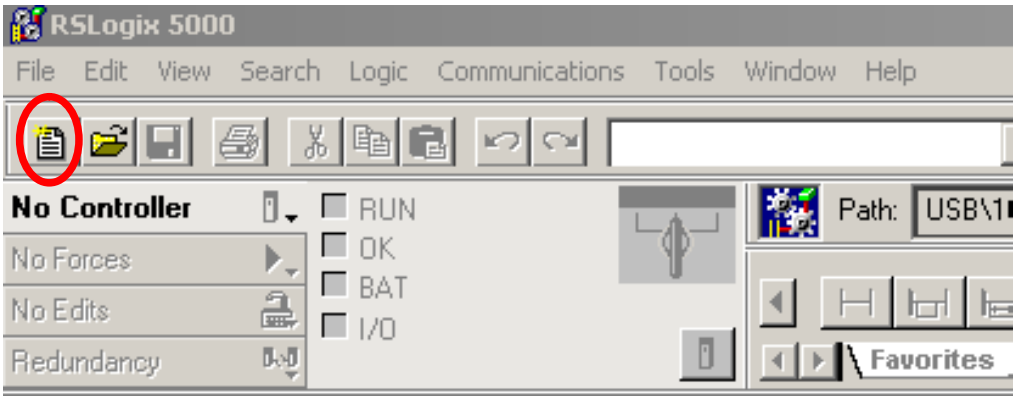
Kimeneti utasítások:

- Output Energize (**OTE**) (Kimenet írása): A memória cella értékét **1**-re állítja, ha a feltétel **IGAZ**, **0**-ra ha a feltétel **HAMIS**.
- Output Latch (**OTL**) (Kimenet **1**-be állítása): A memória cella értékét **1**-re állítja, ha a feltétel **IGAZ**, **HAMIS** feltétel esetén megtartja az előző értékét.
- Output Unlatch (**OTU**) (Kimenet **0**-ba állítása): A memória cella értékét **0**-ra állítja, ha a feltétel **IGAZ**, **HAMIS** feltétel esetén megtartja az előző értékét.

Ezzel az 5 utasítással már elég jól elboldogulhatunk. A következő fejezetben példákon keresztül szemléltetjük az utasítások használatát.

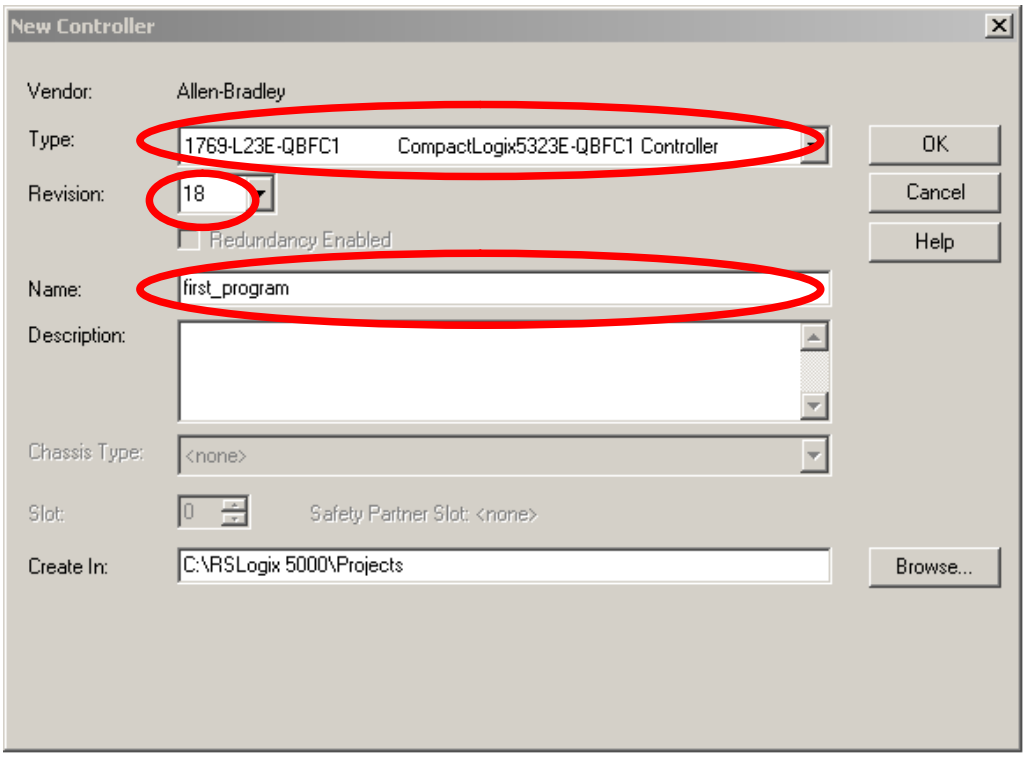
2.1.1. Kezdjünk programozni!

Először indítsuk el az **RSLogix 5000** programot. Amikor betöltődött, klikkeljünk a **New** gombra (19. ábra).

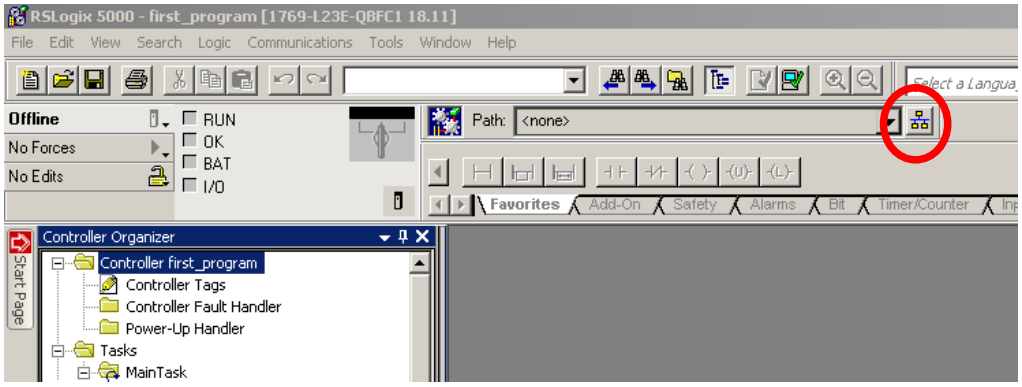


19. ábra Az **RSLogix 5000** program indítása

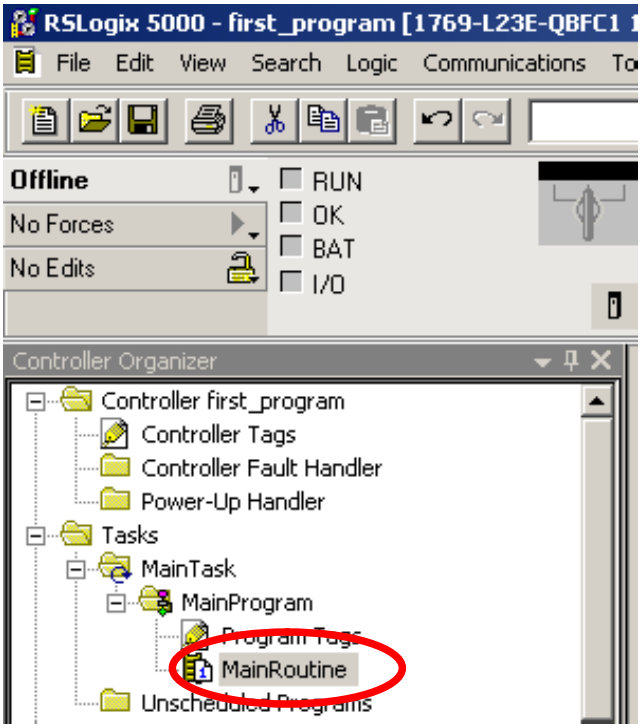
Egy új ablak fog felugrani (20. ábra), itt kiválaszthatjuk a PLC típusát, a firmware verzióját és megadhatjuk a program nevét. Amikor ez megtörtént, kattintsunk az **OK** gombra. Az **RSLogix 5000** elkészíti a program sablonját. Ez eltarthat egy ideig. Amikor végzett, válasszuk ki azt a PLC-t amire a programot tölteni szeretnénk. Ezt az **RSWho** menüpontban tehetjük meg (21. ábra). Amikor kiválasztottuk az elérési útvonalat, nyissuk meg a **MainRoutine**-t a **Main Task** alatt (22. ábra).



20. ábra A PLC típusának kiválasztása

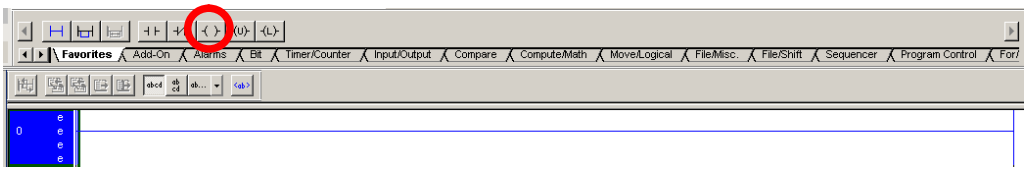


21. ábra A PLC elérési útvonalának kiválasztása

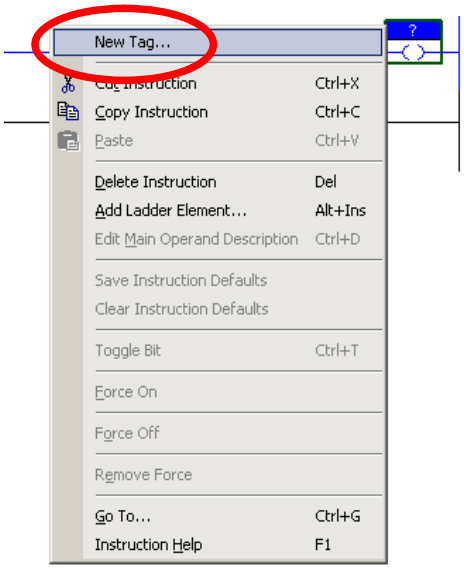


22. ábra A **MainRoutine** megnyitása

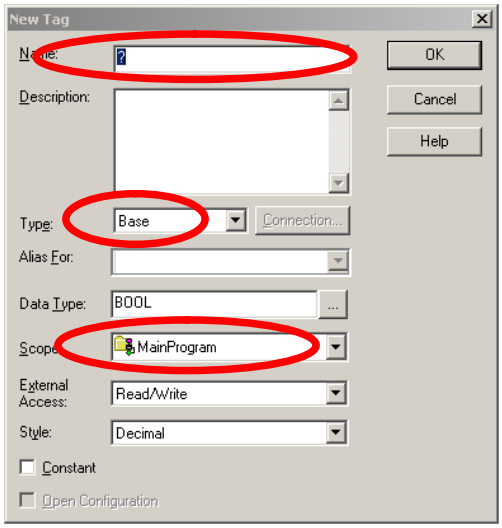
A PLC program írásához csak annyit kell tennünk, hogy az utasítást az eszköztárból behúzzuk az aktuális létraágba. Minden létraágnak legalább egy kimeneti utasítással kell rendelkeznie (23. ábra). A kimeneti utasítások a létraágak a jobb oldalán helyezkednek el. Amint elhelyezünk egy bemeneti-, vagy kimeneti utasítást, el kell látnunk egy **tag**-el. A **tag** megadásával az utasításhoz egy fizikai bemenetet vagy kimenetet rendelünk, ugyanakkor a **tag** az utasításhoz hozzárendel a memóriában egy bitet, vagy bitek csoportját. Helyezzünk el egy **OTE** utasítást a sor végére, majd lássuk el egy **tag**-el. A jobb egérgombbal klikkeljünk rá és válasszuk ki a **New Tag**-et (24. ábra) vagy nyomjuk le a **ctrl+w** billentyűket (25. ábra). Egy új ablak fog felugrani ahol megadhatjuk a **tag** nevét, típusát valamint azt, hogy **base** vagy **alias** legyen. **Base tag** egy bitet vagy bitek csoportját jelöli a memóriában, az **alias** pedig fizikai bemenetre vagy kimenetre vonatkozik. Hogy ehhez a kimeneti utasításhoz egy fizikai kimenetet rendeljünk, válasszuk ki az **alias**-t és azon belül is a fizikai kimenet csatornáját (26 ábra), majd nevezzük el "**output1**"-nek. Figyeljük meg, hogy mint a legtöbb programozási nyelvben az első elemet 0-val jelölik. Amint ezzel elkészültünk máris letölthetjük a programot a PLC-be. Ha programozás során szemantikai hibát követtünk el a letöltés megszakad és az **RSLogix 5000** program ezt jelzi is. Szemantikai ellenőrzés külön is kérhető (27. ábra).



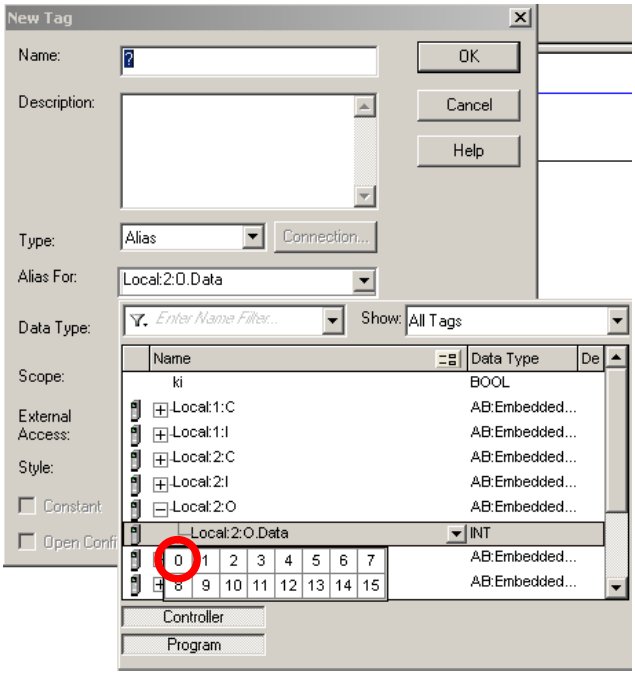
23. ábra Az aktuális létraág



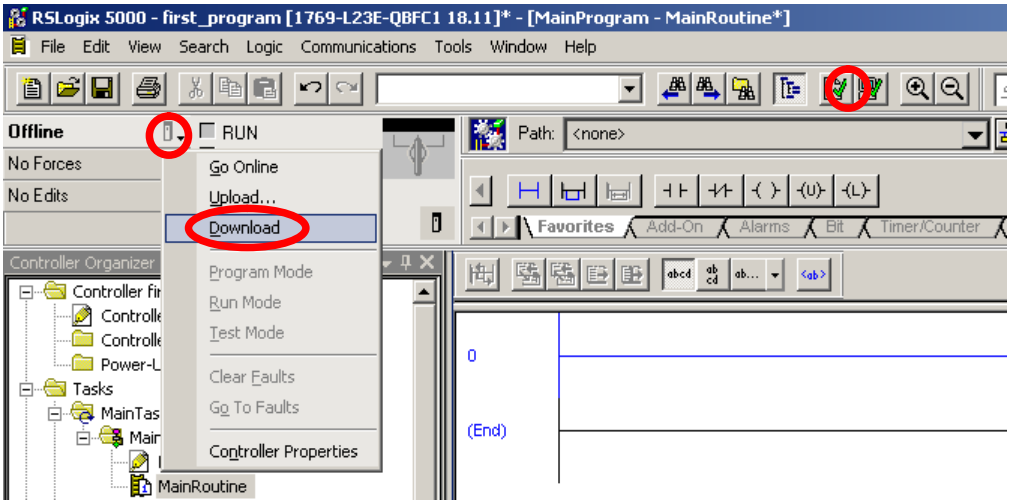
24. ábra A **New Tag** kiválasztása



25. ábra A **New Tag** megadása

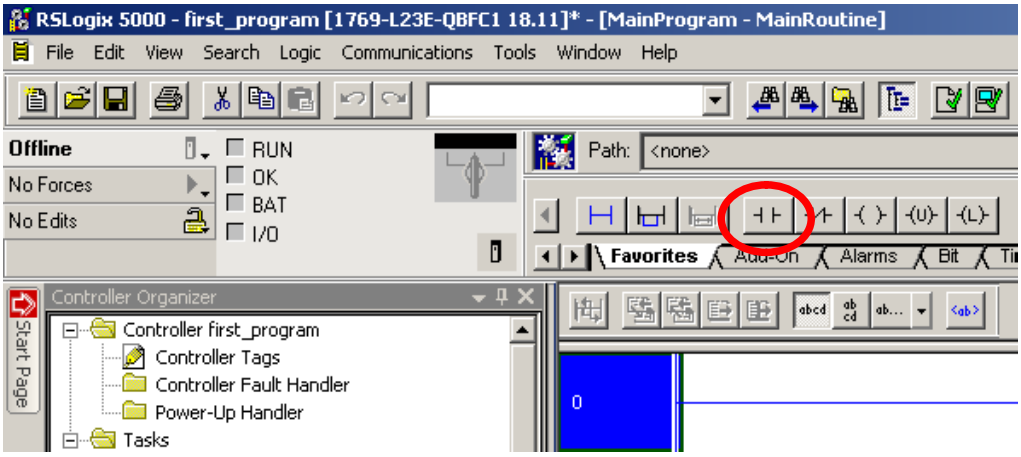


26. ábra Egy fizikai kimenet megadása

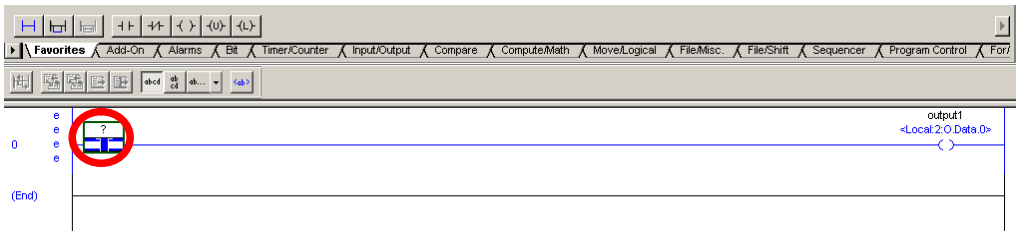


27. ábra A PLC program letöltése

A program rákérdez, hogy akarunk-e üzemmódot váltani. Válasszuk azt, hogy letöltés után a PLC kapcsoljon **RUN** üzemmódba és a program legyen **Online**. Az **Online** üzemmód a fő eszközünk a program működésének ellenőrzésére. A program módosítása csak **Offline** üzemmódban lehetséges. Tegyünk egy feltételt a programsor elejére. Ha nem teszünk feltételt, az megfelel egy mindig igaz utasításnak, ami mindig végrehajtódik. Az első feltétel, amit megvalósítunk, a **XIC** (28. ábra). Ennek az értéke akkor **IGAZ**, ha a hozzá rendelt **tag** éréke is **IGAZ** egyébként **HAMIS**. Helyezzük a szimbólumot a létraág elejére (29. ábra).

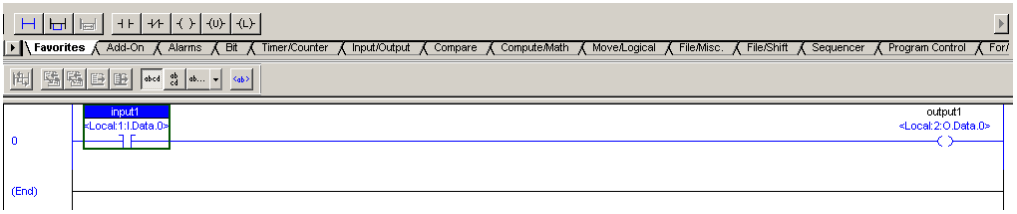


28. ábra Bemeneti feltétel elhelyezése



29. ábra Egy **XIC** utasítás elhelyezése

Végül rendeljünk hozzá egy **tag**-et. Legyen **alias**-a az első fizikai bemenetnek. Nevezzük el "**input1**"-nek (30. ábra).



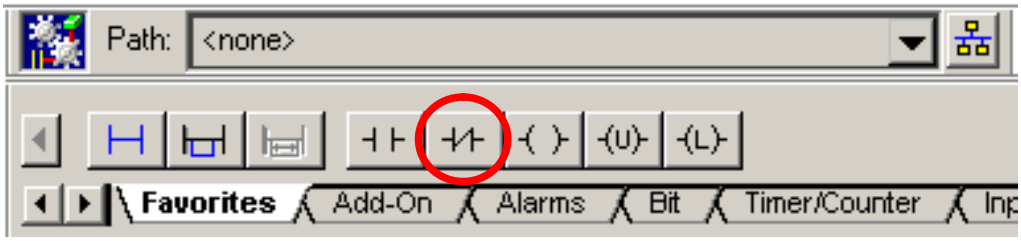
30. ábra A **XIC** utasítás elnevezése

Töltsük le a programot a PLC-be és helyezzük a PLC-t **RUN** üzemmódba. A vezérlő a 2. Táblában látható igazságtáblázat szerint fog működni.

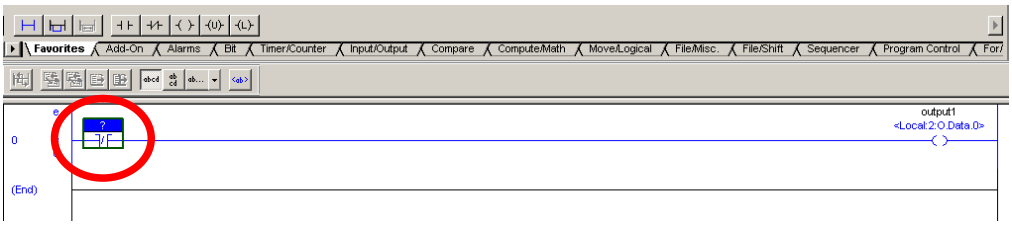
2. Tábla A program igazságtáblázata

input1	output1
0	0
1	1

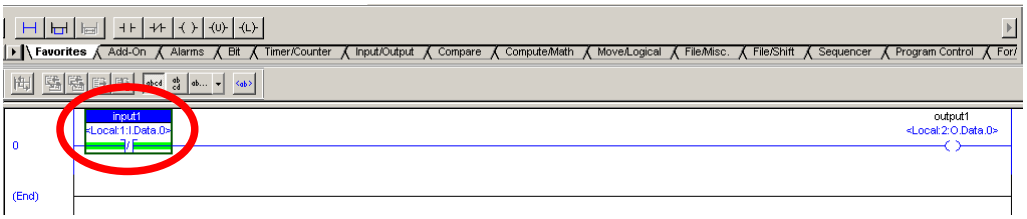
A következő feltétel, amit megtekintünk, a **XIO** (31. ábra). Ennek az értéke akkor **IGAZ**, ha a hozzá rendelt tag éréke **HAMIS**, egyébként **HAMIS**. Cseréljük le a **XIC** utasítást **XIO** utasításra (32. ábra). Rendeljük hozzá az "**input1**" tag-et (33. ábra).



31. ábra A **XIO** utasítás kiválasztása



32. ábra A **XIO** utasítás elhelyezése



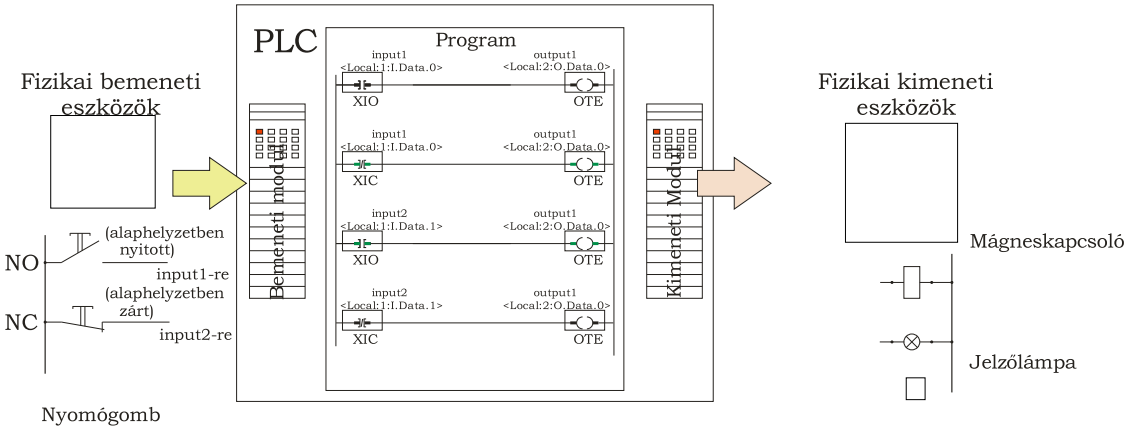
33. ábra Egy **tag** hozzárendelése

Töltsük le a programot a PLC-be és helyezzük a PLC-t **RUN** üzemmódba. A vezérlő a 3. Táblában látható igazságtáblázat szerint fog működni.

3. Tábla A program igazságtáblázata

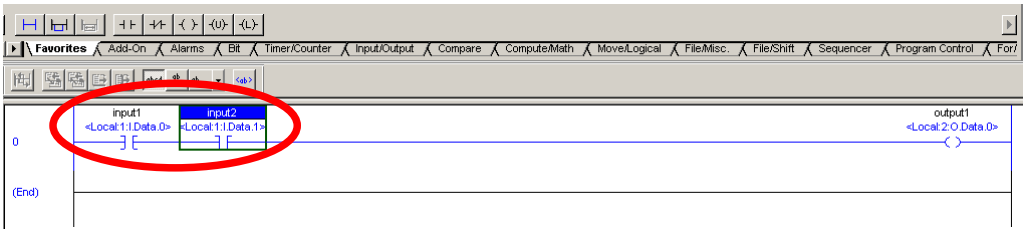
input1	output1
0	1
1	0

A 34. ábrán egy példát látunk a bemeneti (**XIO**, **XIC**) és kimeneti (**OTE**) utasítások működésére.



34. ábra A bemeneti (**XIO**, **XIC**) és kimeneti (**OTE**) utasítások működése

A következőkben a logikai **ÉS** kapcsolatot elemezzük. Logikai **ÉS** kapcsolat elkészítésére helyezzünk kettő vagy több feltételt soros elrendezésbe (34. ábra). A logikai **ÉS** kapcsolat akkor lesz igaz, ha minden benne szereplő feltétel igaznak van kiértékelve (mindegyik zöld). A 35. ábrán látható program működését a 4. Tábla alapján ellenőrizhetjük.

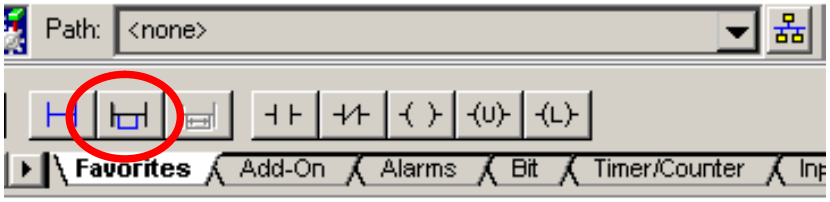


35. ábra Logikai **ÉS** kapcsolat programozása

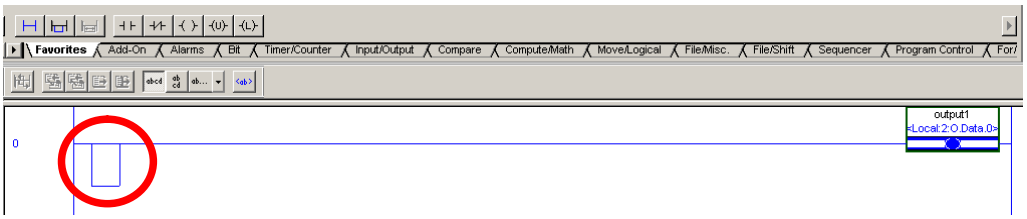
4. Tábla A program igazságtáblázata

input1	input2	output1
0	0	0
0	1	0
1	0	0
1	1	1

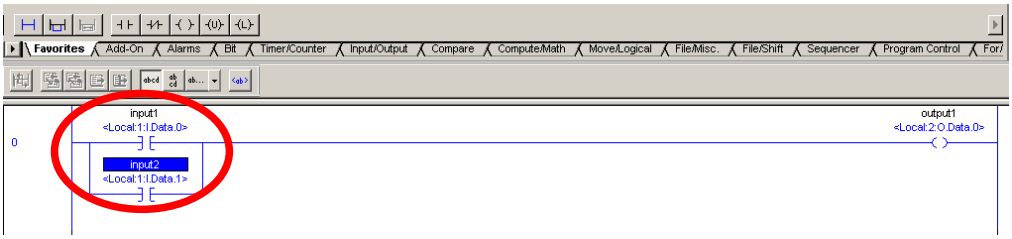
A következőkben a logikai **VAGY** kapcsolatot elemezzük. Logikai **VAGY** kapcsolat elkészítésére helyezzünk kettő vagy több feltételt párhuzamos elrendezésbe a **branch** utasítással (36. ábra). A logikai **VAGY** kapcsolat akkor lesz **IGAZ**, ha bármely benne szereplő feltétel **IGAZ**-nak van kiértékelve (valamelyik zöld). Helyezzük a **branch** utasítást a létraágra (37. ábra), majd helyezzük a feltételeket a párhuzamos ágakra (38. ábra). A 38. ábrán látható program működését az 5. Tábla alapján ellenőrizhetjük.



36. ábra A **branch** utasítás kiválasztása



37. ábra A **branch** utasítás elhelyezése

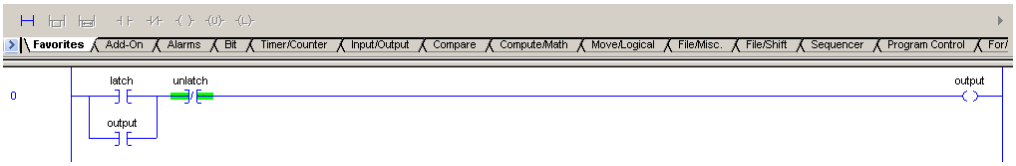


38. ábra Logikai **VAGY** programozása

5. Tábla A program igazságtáblázata

input1	input2	output1
0	0	0
0	1	1
1	0	1
1	1	1

A **branch** utasítás segítségével több kimeneti utasítás is hozzárendelhető egy létraághoz. És most kedves olvasó elérkeztünk az egyik legfontosabb áramkörhöz, az öntartó kapcsoláshoz (39. ábra).



39. ábra Egy öntartó kapcsolás

Az öntartó kapcsolás két bemenettel, visszacsatolással és egy kimenettel rendelkezik. Mivel a PLC ciklusokban működik a visszacsatolás a kimenet előző állapotát adja. Ez a példa azt is szemlélteti, hogy a kimeneti **tag**-et tetszőleges számban felhasználhatjuk bemeneti feltételeknél. Ha a "**latch**" bemenet **IGAZ**, az „**output”** is **IGAZ** és ez bemeneti feltételként áthidalja a "**latch**" bemenetet. Ha a "**latch**" közben visszavált **0**-ra az „**output”** mindaddig **1** marad, amíg az "**unlatch**" meg nem változtatja azt. A 39. ábrán látható program működését leellenőrizhetjük a 6. Tábla alapján.

6. Tábla A program igazságtáblázata

Latch	Unlatch	A kimenet előző állapota	A kimenet állapota
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Ezt a kapcsolást olyan gyakran alkalmazzuk, hogy külön utasítást is készítettek hozzá. Ezek az **OTL** és **OTU**.
Ezekkel az alaputasítások a feladattól függően bonyolultabb programok alapelemeiként használhatóak.

2.1.2. Rendszer tag-ek

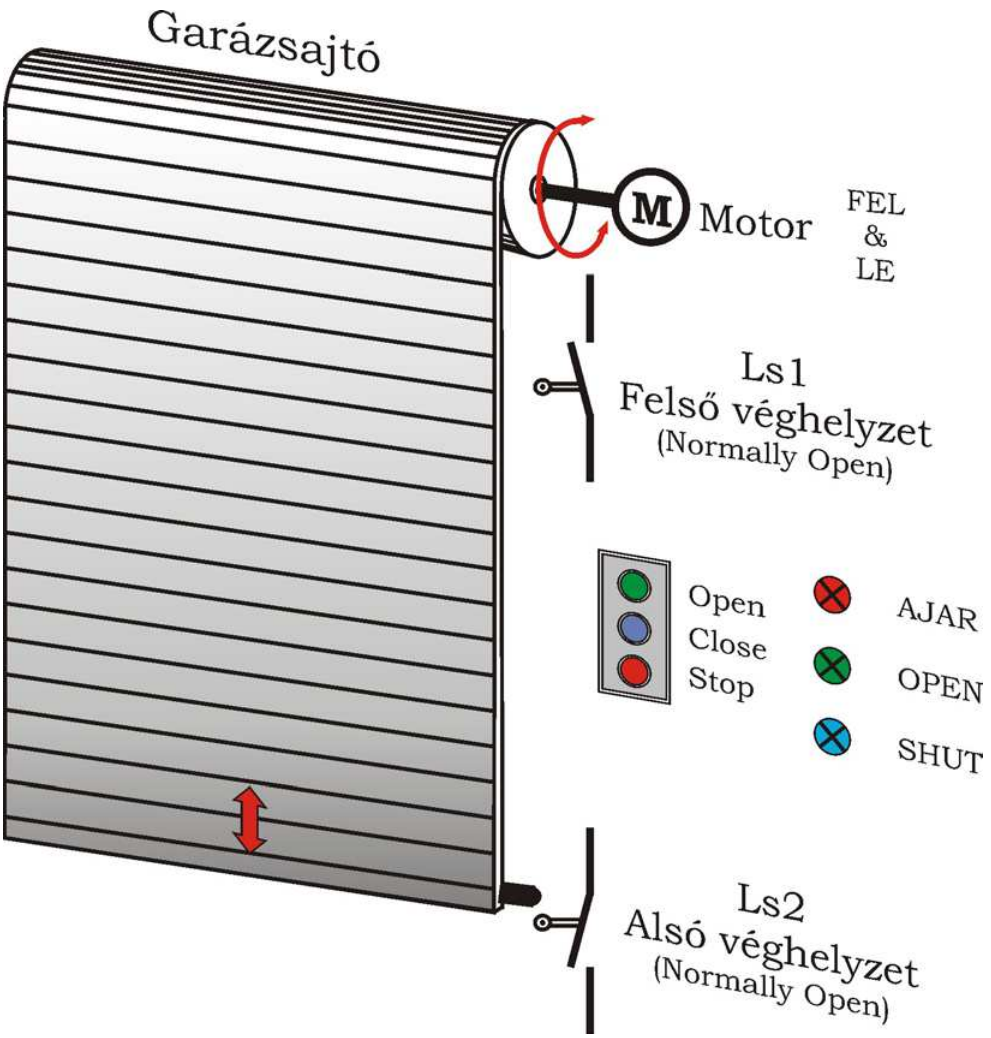
A legtöbb PLC rendelkezik olyan bitekkel, melyek különleges szerepet töltenek be a rendszer működésében. Ezek nem váltanak szint monitorozás közben. A legfontosabb az első ciklust jelző bit, ami csak a PLC bekapcsolása utáni első ciklusban aktív.

A legfontosabb **rendszer tag**-ek és funkcióik:

- **S:FS** a program **első ciklusa**
- **S:MINOR** kisebb hibát jelző **tag**
- **S:V** akkor aktív, ha a tárolni kívánt érték meghaladja a tag kapacitását. Minden aktiválásával **S:MINOR** is jelez.
- **S:Z** jelzi, ha az utasítás célértéke **0**.
- **S:N** jelzi, ha az utasítás célértéke **negatív**.
- **S:C** az aritmetikai utasítás **carry** bitje.

2.1.3. I. Feladat: Garázsajtó

Vizsgáljunk meg egy gyakran használt rendszert, a garázsajtót. A rendszer vázlatát a 40. ábrán láthatjuk. A feladat megoldását a 41. ábra vázolja.



40. ábra Garázsajtó

A bemenetek a 7. táblában láthatók.

7. Tábla Bemeneti változók

Bemenetek (kapcsolók)	Név	Típus	Azonosító
Nyomógomb	Open	NO	Local:1:I.Data.0
Nyomógomb	Close	NO	Local:1:I.Data.1
Nyomógomb	Stop	NC	Local:1:I.Data.2
Szintkapcsoló	LS1	NC	Local:1:I.Data.3
Szintkapcsoló	LS2	NC	Local:1:I.Data.4

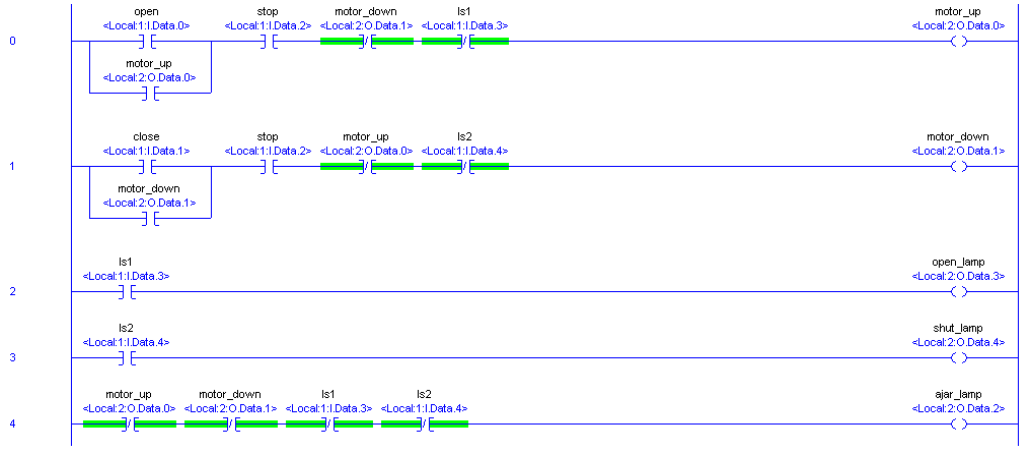
A kimenetek a 8. táblában láthatók.

8. Tábla Kimeneti változók

Kimeneti eszközök	Név	Azonosító
Mágneskapcsoló	Motor UP	Local:2:O.Data.0
Mágneskapcsoló	Motor DOWN	Local:2:O.Data.1
Jelzőlámpa	AJAR	Local:2:O.Data.2
Jelzőlámpa	OPEN	Local:2:O.Data.3
Jelzőlámpa	SHUT	Local:2:O.Data.4

A feladat a következő:

- Az **Open** jelű nyomógomb lenyomásával az ajtó felfelé indul amíg **LS1** nem jelzi a végállapot elérését.
- A **Close** jelű nyomógomb lenyomásával az ajtó lefelé indul míg **LS2** nem jelzi a végállapot elérését.
- A gombokat nem kell nyomva tartani a mozgás fenntartásához.
- A **Stop** jelű nyomógomb leállítja az aktuális mozgást.
- A **Motor UP** és **Motor DOWN** mágneskapcsolók nem lehetnek egyszerre aktívak.
- Az aktuális állapotot lámpák jelzik. **OPEN** a teljesen nyitott ajtót, **SHUT** a teljesen zárt ajtót és **AJAR**, ha az ajtó nem végpontban állt meg.



41. ábra A garázsajtó vezérlésének PLC programja

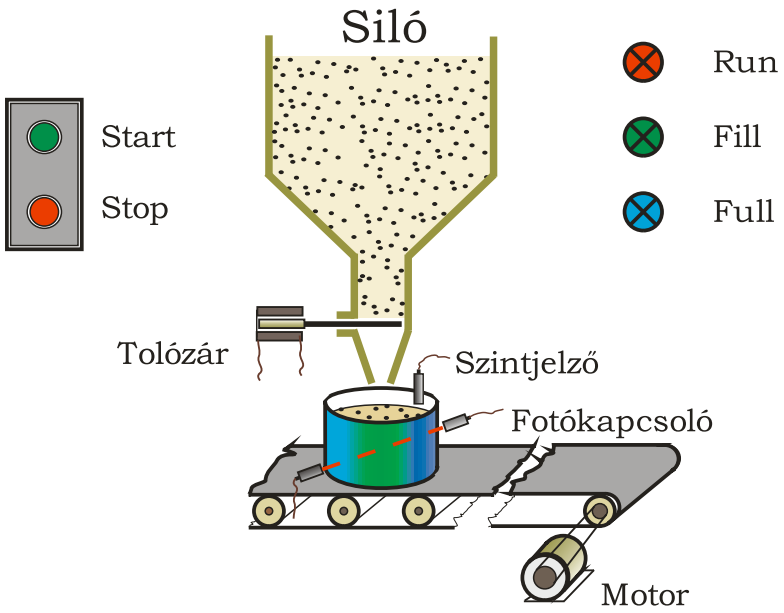
PLC programok írásakor érdemes néhány egyszerű szabályt követnünk. Minden sor csak egy fizikai kimenetre hivatkozó kimeneti utasítást tartalmazzon. Az **OTE** utasítás használatakor kerüljük a **tag**-ek ismétlését, mivel az utolsó mindig felülírja az előtte levőket. Az áttekinthetőség és könnyű karbantarthatóság érdekében minden feltétel, aminek függvényében változik a kimenet, egy ágban legyen alkalmazva.

A 0. sor az ajtó nyitásáért felelős. Mivel ez egy öntartó kapcsolás, az **Open** jelű nyomógombot nem kell folyamatosan nyomva tartani. Legalább egy leállási feltételt is be kell építeni. Itt három van, ezek a **Stop** jelű nyomógomb, az **LS1** jelű végálláskapcsoló és a **Motor DOWN**. Mivel a **Stop** jelű nyomógomb **NC** ezért mindig aktív kivéve mikor megnyomják, ezt felhasználva állíthatjuk le az ajtó nyitását (zárását). Ugyanez vonatkozik **LS1**-re. A **Motor DOWN** feltétel beépítésének célja az, hogy biztosítsa azt, hogy ha a **Motor DOWN** már aktív, akkor a **Motor UP** nem lehet az. Ezt keresztreteszelésnek nevezzük.

Az 1. sor hasonlít a 0. sorhoz. A 2. sor működteti az **OPEN** jelű jelzőlámpát **LS1**-től függően. A 3. sor működteti a **SHUT** jelű jelzőlámpát **LS2**-től függően. A 4. sor működteti az **AJAR** jelű jelzőlámpát akkor, ha a garázsajtó nincs véghelyzetben (**LS1**, **LS2**) és a mágneskapcsolók sem működnek.

2.1.4. II. Feladat: Siló

A 42. ábrán láthatjuk a rendszer vázlatát. A 43. ábra a feladat megoldását mutatja.



42. ábra Siló rendszer

A bemenetek a 9. táblában találhatóak.

9. Tábla Bemeneti változók

Bemenetek (kapcsolók)	Név	Típus	Azonosító
Nyomógomb	Start	NO	Local:1:I.Data.0
Nyomógomb	Stop	NC	Local:1:I.Data.1
Fotókapcsoló	Proxy	NO	Local:1:I.Data.2
Szintjelző	LS	NO	Local:1:I.Data.3

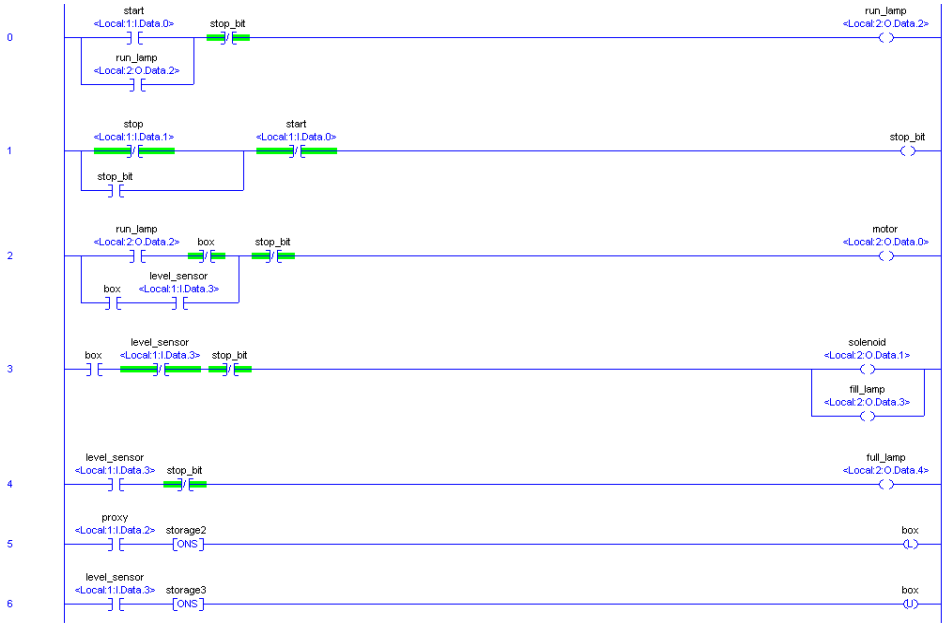
A kimenetek a 10. táblában láthatóak.

10. Tábla Kimeneti változók

Kimeneti eszközök	Név	Azonosító
Mágneskapcsoló	Motor	Local:2:O.Data.0
Mágneskapcsoló	Solenoid	Local:2:O.Data.1
Jelzőlámpa	Run	Local:2:O.Data.2
Jelzőlámpa	Fill	Local:2:O.Data.3
Jelzőlámpa	Full	Local:2:O.Data.4

A feladat a következő:

- **Start** gomb lenyomására a futószalag elindul és a **Run** lámpa jelez.
- Amikor a doboz eléri a **Proxy** érzékelőt a futószalag leáll.
- Amikor a doboz leáll a **Solenoid** hatására elkezd töltődni a doboz.
- Töltés közben a **Fill** lámpa jelez.
- Amikor a doboz megöltődött **Level sensor** jelez, ekkor a futószalag elszállítja a teli dobozt és egy üreset hoz helyére, majd a folyamat ismétlődik.
- **Stop** gomb lenyomására a rendszer leáll. **Start** gomb lenyomására a rendszer visszatér az előző állapotába.



43. ábra A Siló rendszer PLC programja

Ebben a programban egy új utasítást ismerünk meg. Az 5 - 6 sorban látható a **One Shot** (**ONS**) utasítás. Ez az utasítás figyeli az előtte levő feltétel(eke)t, amikor felfutó élt érzékel egy ciklusig **1** re váltja az állapotát. Lefutó él detektálásához csak a negálnunk kell.

Elemezzük a programot:

A 0. sor a **Run** lámpa működéséért felelős. A **Start** gomb lenyomására a **Run** lámpa jelez. A **stop_bit** aktiválására a jelzés megszűnik.

Az 1. sor a **stop_bit**-et működteti. A **Stop** gomb lenyomására a **stop_bit** aktiválódik. A **Start** gomb deaktiválja a **stop_bit**-et. A **stop_bit**-et a 0. 2. és 3. sorok használják a kimeneteik letiltására.

A 2. sor a futószalag motorját vezérli. A motornak abban az esetben kell működnie, amikor nincs doboz az érzékelőnél és jelez a **Run** lámpa (üzemel a gép) valamint akkor, amikor a doboz tele van. A **stop_bit** letiltja a motort.

A 3. sor vezérli a doboz töltését és működteti a **Fill** jelű lámpát. A kettőnek egyszerre kell működnie. A **stop_bit** letiltja a kimeneteket.

A 4. sor a **Full** lámpát vezérli akkor, amikor a **Level sensor** jelez. A **stop_bit** deaktiválja a kimenetet.

Az 5. sor a **Proxy** felfutó élére **box_bitet 1**-re állítja.

A 6. sor a **Level sensor** lefutó élére a **box_bitet 0**-ra állítja.

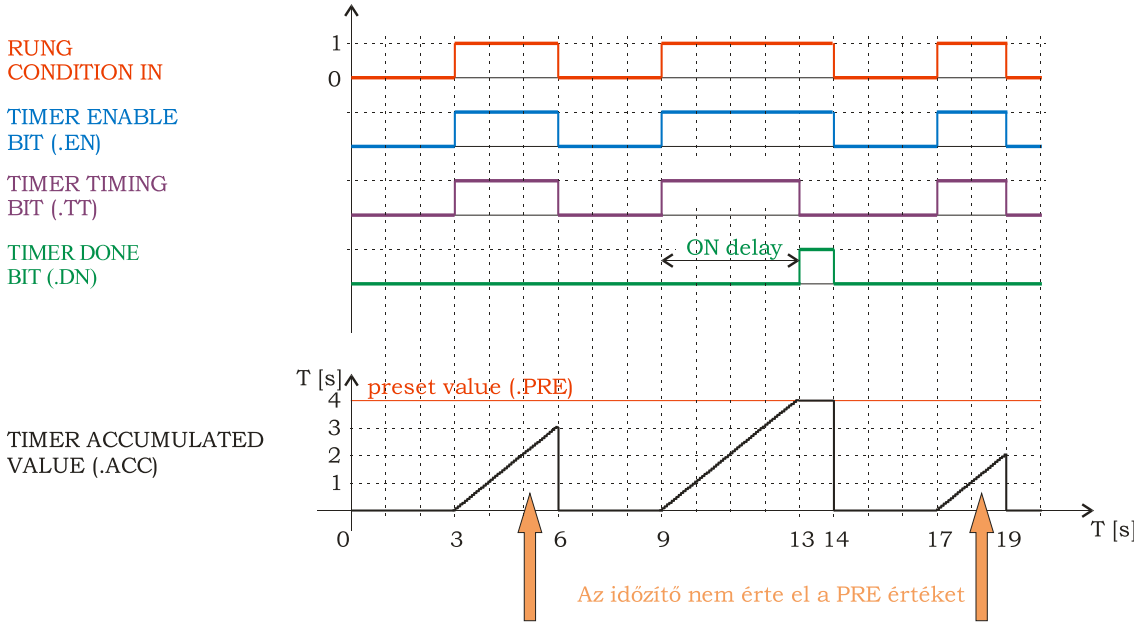
2.2. Ideje PLC-zni!

Az eddigi utasításokkal csak az azonnali eseményeket vagyunk képesek lekezelni, viszont sokszor van szükségünk olyan utasítások használatára, melyek bizonyos idő elteltével kell, hogy végrehajtódjanak. Ilyen feladatokra időzítőket használunk. Az időzítők időalappal rendelkeznek, amit általában milliszekundumban mérünk. A legkisebb mérhető időegység 1 ms. Egyes PLC-k 10, 100, 1000 ms-os időalappal is rendelkeznek. Következzenek a különböző időzítők és alkalmazásuk.

2.2.1. Időzítő típusok és alkalmazásuk

Az **RSLogix 5000** három típusú időzítőt ismer, ezek **Timer On Delay** (**TON**), **Timer Off Delay** (**TOF**), **Retentive Timer On Delay** (**RTO**). Mindegyik rendelkezik egy **tag**-el, **preset** értékkel (**PRE**) és **akkumulátorral** (**ACC**). A **preset** ms-ban tárolja azt az értéket, ameddig az időzítő vár. Az aktuálisan eltelt idő az **ACC**-ben van tárolva. Az időzítők rendelkeznek állapotjelző bitekkel is, ezek az **enable** (**EN**), a **done** (**DN**) és a **timing** (**TT**) bit.

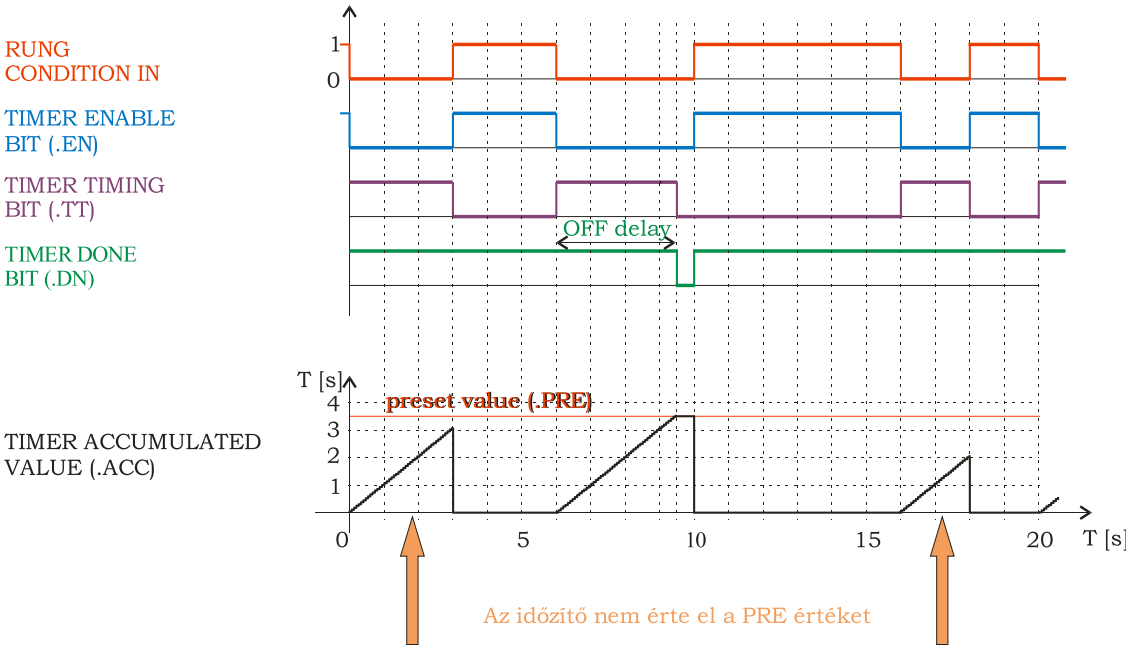
A **TON** működését a 44. ábrán láthatjuk. A **TON** a bemeneti feltétel felfutó élét késlelteti a **preset** értékkel, ha a bemenő jel legalább addig aktív. Az **EN** aktív mindaddig, amíg a bemeneti feltétel is aktív. A **TT** addig aktív, amíg az időzítő számol és az **ACC** értéke nem érte el a **PRE** értéket. A **DN** mindaddig aktív, amíg a bemeneti jel aktív és az időzítő elérte a **PRE** értéket. Amint a bemeneti feltétel megszűnik az időzítő kinullázódik.



44. ábra A **TON** időzítő működése

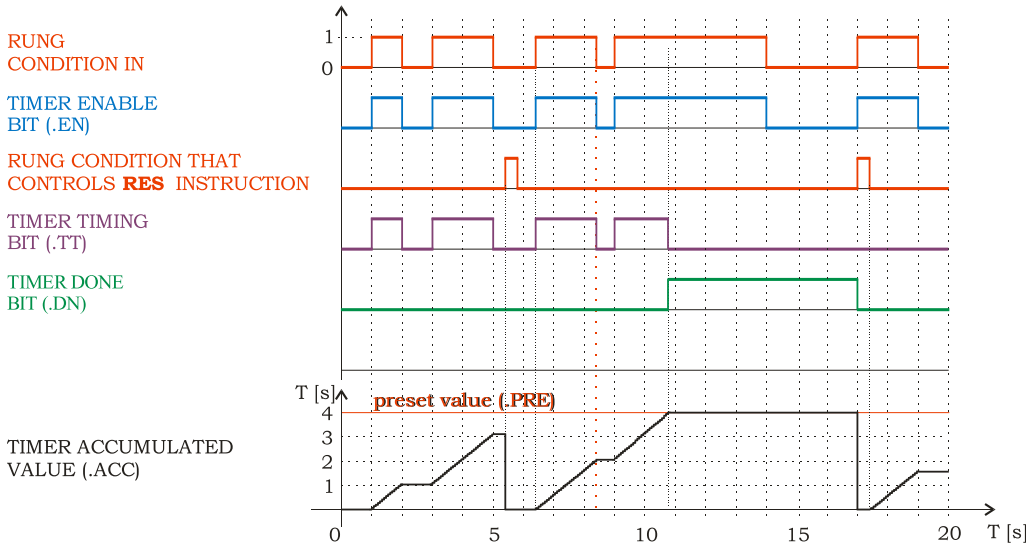
A **TOF** működését a 45. ábrán láthatjuk. A **TOF** a bemeneti feltétel lelfutó élét késlelteti a **preset** értékkel, ha a bemenő jel legalább addig inaktív. Az **EN** aktív mindaddig, amíg a bemeneti feltétel inaktív. A **TT** addig aktív, amíg az időzítő számol és az **ACC** értéke nem érte el a **PRE** értéket. A **DN** mindaddig aktív, amíg az

időzítő el nem érte a **PRE** értéket. Amint a bemeneti feltétel visszatér az időzítő kinullázódik.



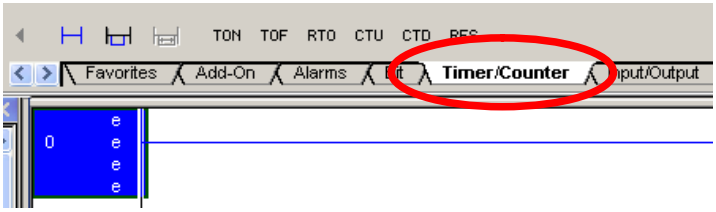
45. ábra A **TOF** időzítő működése

Az **RTO** működését a 46. ábrán láthatjuk. Az **RTO** a bemeneti feltétel felfutó élét késlelteti a **preset** értékkel, ha a bemenő jel legalább addig aktív. Az **EN** aktív mindaddig, amíg a bemeneti feltétel is aktív. A **TT** addig aktív, amíg az időzítő számol és az **ACC** értéke nem érte el a **PRE** értéket. A **DN** mindaddig aktív, amíg a bemeneti jel aktív és az időzítő elérte a **PRE** értéket. Miután a bemeneti feltétel megszűnik, az időzítő nem nullázódik ki.

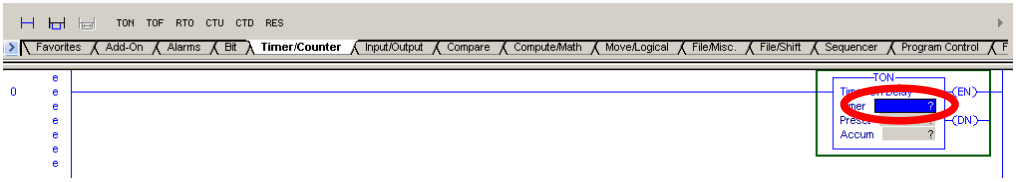


46. ábra A **RTO** időzítő működése

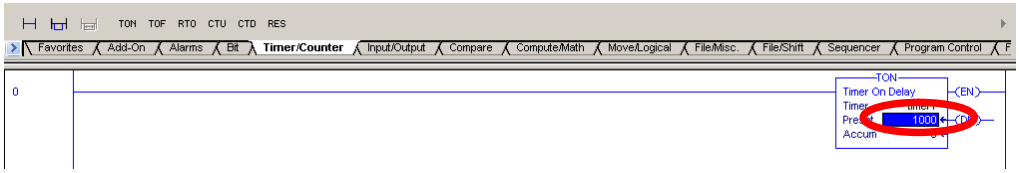
Lássunk egy példát az időzítők használatára. Az időzítők az eszköztárban a **Timer/Counter** bejegyzésnél találhatóak (47. ábra). Tegyünk be egy **TON** utasítást a létraágba. A 48. ábrán láthatjuk a **tag** helyét. Rendeljünk egy **tag**-et az időzítőhöz, majd állítsuk be a **preset** időt (49. ábra).



47. ábra A **Timer/Counter** kiválasztása

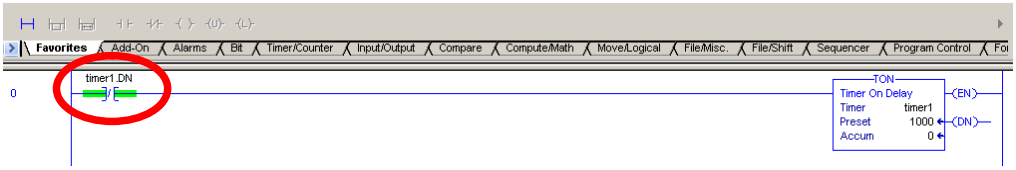


48. ábra Az időzítő címzése



49. ábra Az időzítő beállítása

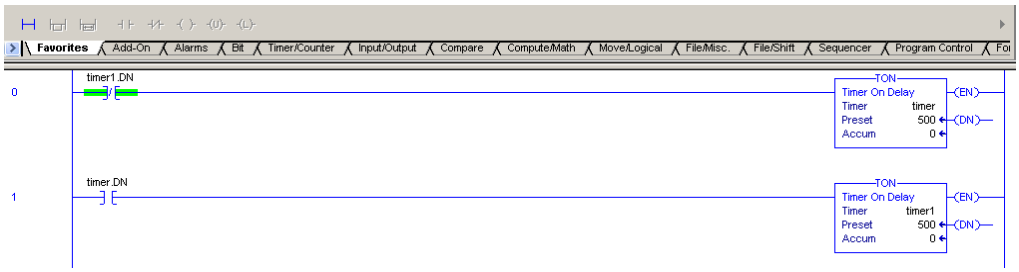
Tegyünk be egy feltételt, ami újraindítja az időzítőt amint a **preset** idő eltelt (50. ábra).



50. ábra Egy **Reset** feltétel beiktatása

Egy **tag** állapot bitjeit és értékeit a "." operátorral lehet elérni (pl.: **timer1.DN**). Bizonyosodjunk meg róla, hogy az utasítás képes fogadni az így megkapott értéket (pl.: **XIC** nem fogadhatja **timer1.ACC** értékét mivel nem bit). A négyszögjel előállítása egy tipikus feladat, amit időzítőkkel szoktunk megoldani. A 51. ábrán láthatunk egy programot, ami 50%-os kitöltési tényezővel rendelkező négyszögjelet állít elő, melynek periódusideje 1 s. A program működése a következő. A **timer** elkezd időzíteni mivel **timer1.dn** értéke **0**. Amint **timer ACC**-je eléri a **preset** értéket **timer.DN 1** lesz. Timer.DN elindítja **timer1**-et. Amint **timer1** eléri a **PRE**

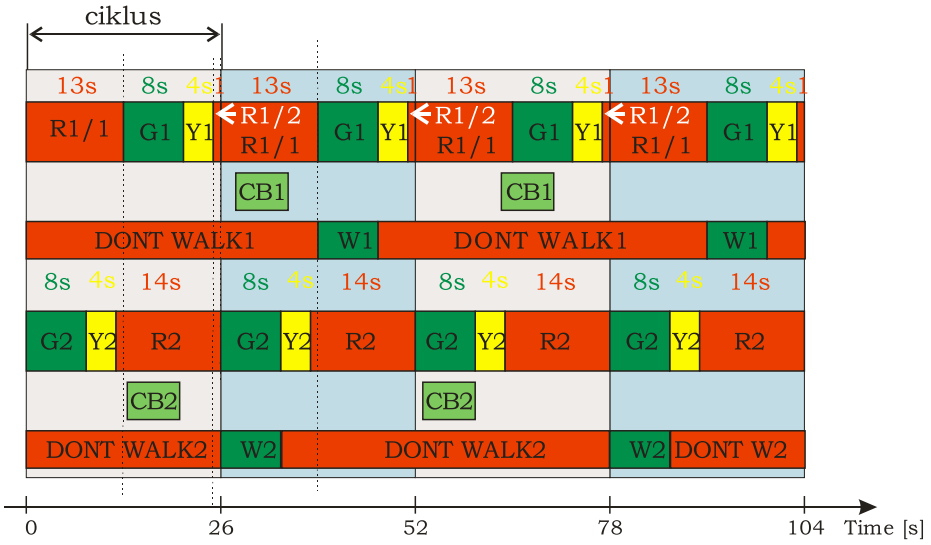
értékét **timer1.DN** értéke **1** lesz. Ez újraindítja mindkét időzítőt és a folyamat a végtelenségig ismétlődik.



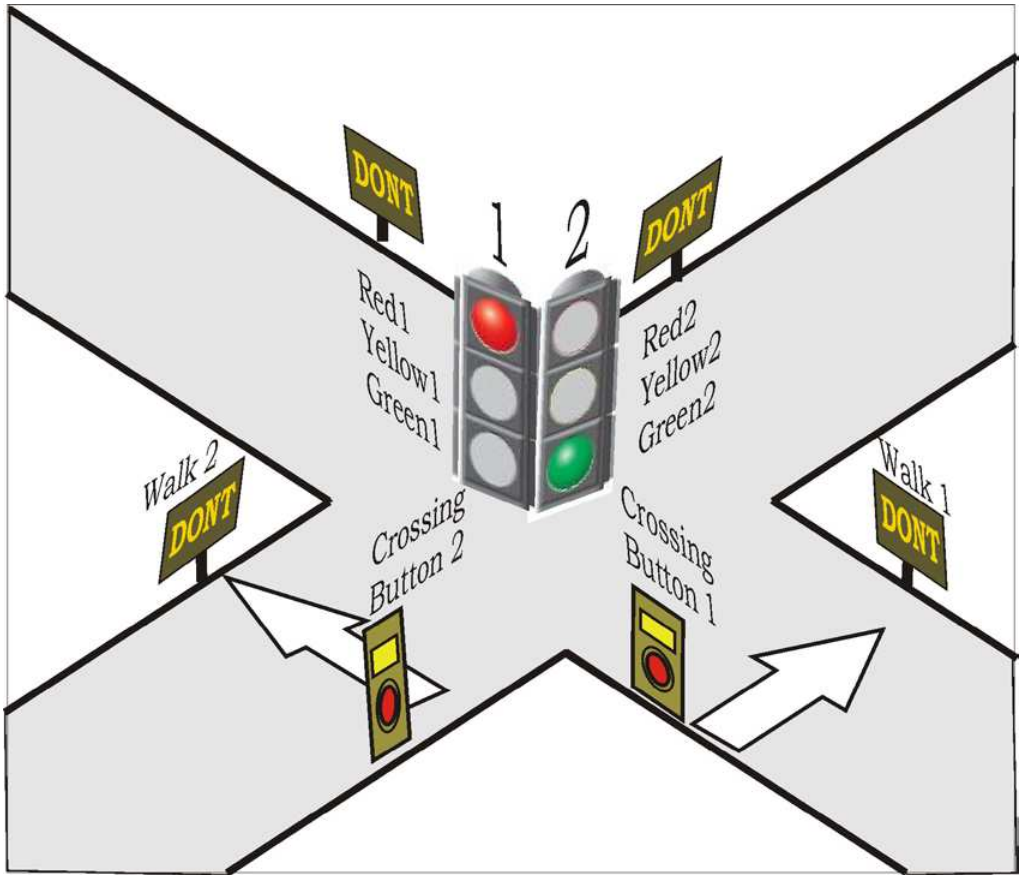
51. ábra Négyszögjel előállítása

2.2.2. III. Feladat: Forgalomirányítás

Ebben a feladatban forgalomirányító lámpát kell vezérelni. Az 52. ábrán láthatjuk a lámpa működését leíró idődiagramot. A gyalogosoknak szóló jelzés gombnyomásra változik, amint a megfelelő forgalomirányító lámpa zöldre vált. Az 53. ábrán láthatjuk a forgalomirányító rendszer felépítését. Az 54. és 55. ábrán láthatjuk a feladat megoldását.



52. ábra Idődiagram



53. ábra A forgalomirányító rendszer felépítése

A bemenetek a 11. táblában láthatók.

11. Tábla Bemeneti változók

Bemenetek (kapcsolók)	Név	Típus	Azonosító
Nyomógomb	Crossing Button1	NO	Local:1:I.Data.0
Nyomógomb	Crossing Button2	NO	Local:1:I.Data.1

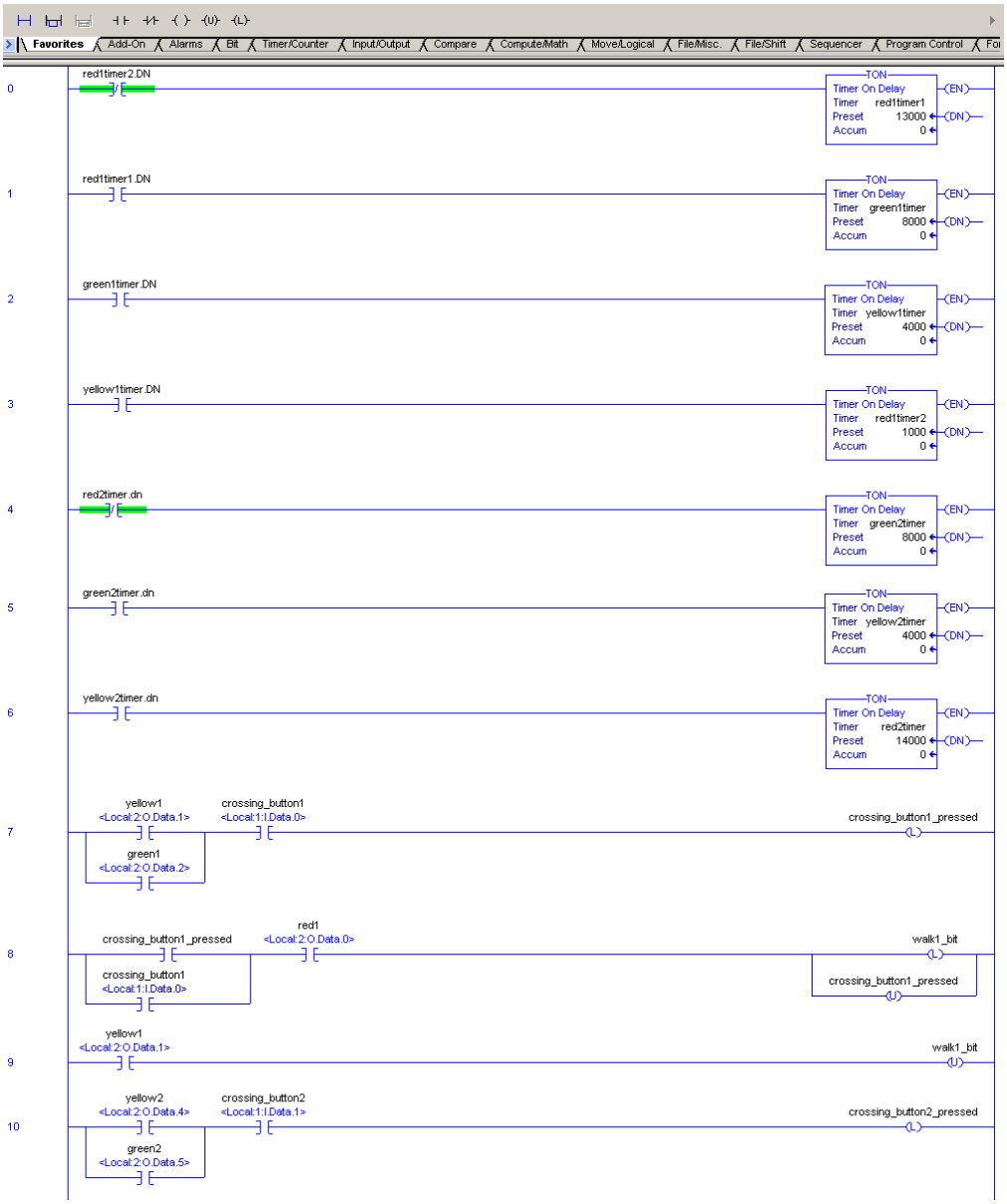
A kimenetek a 12. táblában láthatók.

12. Tábla Kimeneti változók

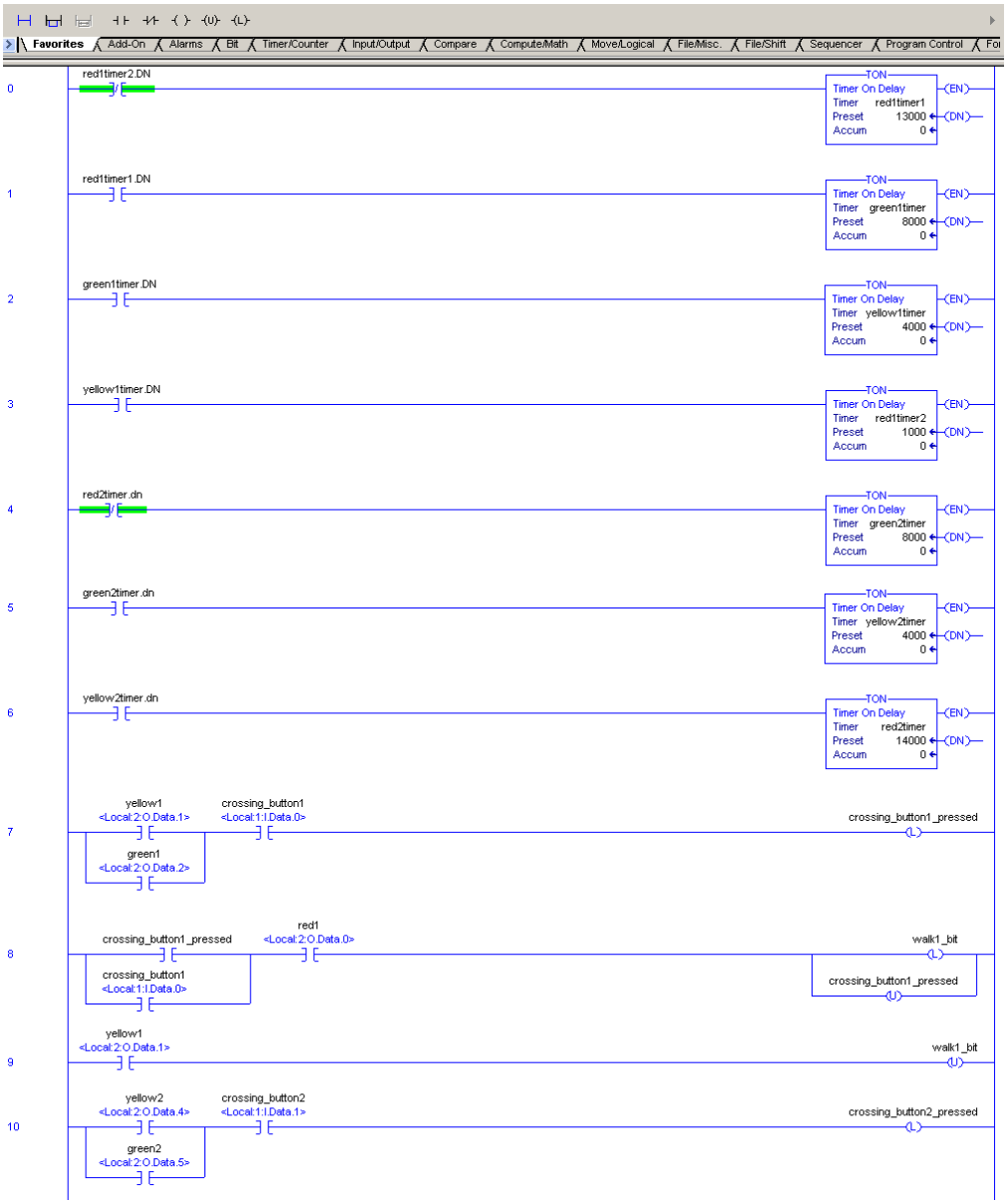
Kimeneti eszközök	Név	Azonosító
Forgalmi lámpa	Red1	Local:2:O.Data.0
Forgalmi lámpa	Yellow1	Local:2:O.Data.1
Forgalmi lámpa	Green1	Local:2:O.Data.2
Forgalmi lámpa	Red2	Local:2:O.Data.3
Forgalmi lámpa	Yellow2	Local:2:O.Data.4
Forgalmi lámpa	Green2	Local:2:O.Data.5
Forgalmi lámpa	Walk1	Local:2:O.Data.6
Forgalmi lámpa	Walk2	Local:2:O.Data.7

A feladat a következő:

- A **Red1**, **Red2**, **Yellow1**, **Yellow2**, **Green1**, **Green2** lámpák az 52. ábrán felvázoltak alapján kell, hogy működjenek.
- A **Walk1** csak a **Crossing Button1** lenyomása után legyen aktív.
- A **Walk1** csak a **Green1** következő bekapcsolásakor, annak teljes működése alatt legyen aktív.
- A **Walk2** csak **Crossing Button2** lenyomása után legyen aktív.
- A **Walk2** csak a **Green2** következő bekapcsolásakor, annak teljes működése alatt legyen aktív.



54. ábra A forgalomirányítás PLC programja



55. ábra A forgalomirányítás PLC programja

Elemezzük a programot. 0.-6.-ig a sorok időzítőket tartalmaznak, melyek két láncba vannak fűzve. Az első lánchoz rendeljük a **Red1**, **Yellow1** és **Green1** lámpa működését, a másodikhoz pedig a **Red2**, **Yellow2**, és **Green2**-t. A két lánc független egymástól.

7.-9. sorok a **Crossing Button1** logikai működését tartalmazzák. A gyalogátkelő gombok lenyomása közben a lámpa lehet piros, sárga vagy zöld. Sárga vagy zöld esetén beállítjuk a **crossing_button1_pressed** bitet **1**-re. Ez a bit fogja tárolni az információt, hogy a **Crossing Button1** le lett nyomva. Ha akkor nyomjuk le a gombot, amikor a lámpa piros, vagy már le lett nyomva egyszer a gomb, a **walk1_bit** bitet állítjuk **1**-re és **crossing_button1_pressed**-et **0**-ra. A 9. sorban a

walk1_bit-et visszaállítjuk **0**-ra. Ezzel azt érjük el, hogy egy gombnyomás a lámpák egy körforgásáig maradjon a rendszerben.

10.-12. sorok hasonlítanak a 7.-9. sorokra, az egyetlen különbség az, hogy a **Walk2** lámpa működését írják le.

13.-20. sorok a kimeneteket vezérelik. Ahogy az elején megbeszéltük a programunk sokkal átláthatóbb, ha soronként egy kimenetet teszünk és mindet a program végére rakjuk.

A 13. sor a **Red1** lámpát vezéreli. Mivel az idődiagram szerint a ciklus elején és végén is aktív kell, hogy legyen logikai **VAGY** feltétel köti össze a **red1timer1** és **red1timer2**-t.

A 14.-18. sorok a **Yellow1**, **Green1**, **Red2**, **Yellow2**, **Green2** lámpákat vezérlik. Ezek a lámpák az időzítőik TT bitjére vannak kötve.

A 19. sor a **Green1** és **walk1_bit** hatására a **Walk1** lámpát vezérli.

A 20. sor a **Green2** és **walk2_bit** hatására **Walk2** lámpát vezérli.

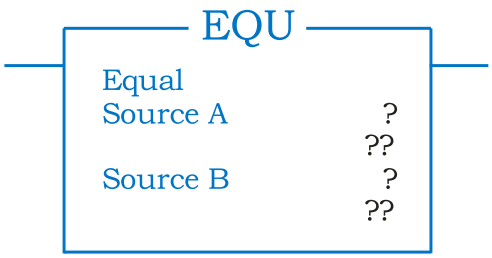
2.3. Kezdjünk összehasonlítani

Láthatjuk, hogy az időzítők lényegében számokat tartalmaznak. Eddig csak az **EN**, **DN**, **TT** bitjeit használhattuk ki egy időzítőnek, mivel csak **0** és **1** értékekkel tudtunk dolgozni. Ahhoz, hogy az **ACC** és **PRE** értékeit felhasználhassuk, először át kell őket logikai értékekké alakítani. Ezt összehasonlító műveletekkel végezhetjük.

2.3.1. Összehasonlító műveletek

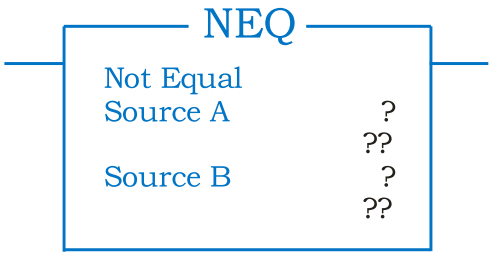
Az összehasonlító műveletek feltételek és hasonlóan kell őket használni. Az eszköztárban a **Compare** fül alatt találhatóak. A műveletek a következők:

- **EQU** - **EGYENLŐ** (56. ábra) **1**, ha az **A** és **B** értéke megegyezik, egyébként **0**.



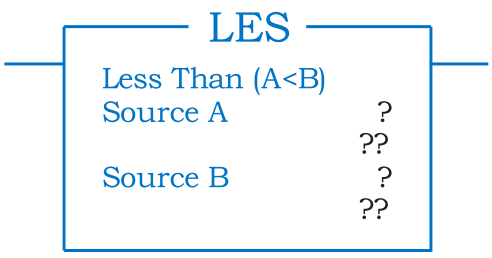
56. ábra Az **EGYENLŐ** (**EQU**) utasítás

- **NEQ** - **NEM EGYENLŐ** (57. ábra) **1**, ha az **A** és **B** értéke nem egyezik meg, egyébként **0**.



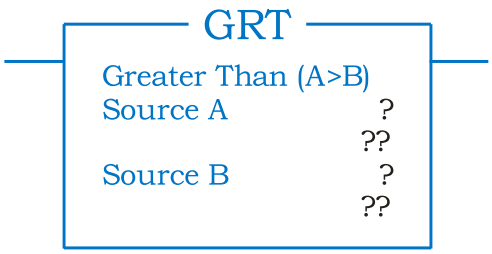
57. ábra A **NEM EGYENLŐ** (**NEQ**) utasítás

- **LES** - **KISEBB** (58. ábra) **1**, ha az **A** értéke kisebb mint **B** értéke, egyébként **0**.



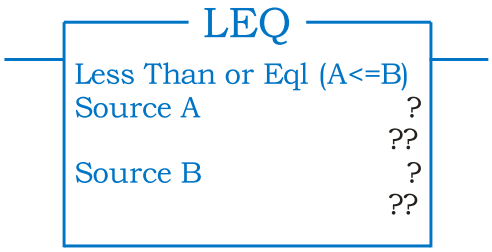
58. ábra A **KISEBB** (**LES**) utasítás

- **GRT** - nagyobb (59. ábra) **1**, ha az **A** értéke nagyobb, mint **B** értéke, egyébként **0**.



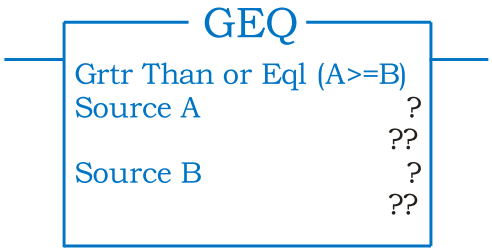
59. ábra A **NAGYOBB** (**GRT**) utasítás

- **LEQ** - **KISEBB EGYENLŐ** (60. ábra) **1**, ha az **A** értéke kisebb vagy egyenlő **B** értékével, egyébként **0**.



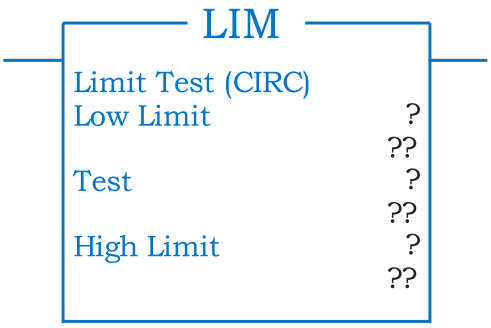
60. ábra A **KISEBB EGYENLŐ** (**LEQ**) utasítás

- **GEQ** - **NAGYOBB EGYENLŐ** (61. ábra) **1**, ha az **A** értéke nagyobb vagy egyenlő **B** értékével, egyébként **0**.



61. ábra A **NAGYOBB EGYENLŐ** (**GEQ**) utasítás

- **LIM** - **HATÁRÉRTÉK-VIZSGÁLAT** (62. ábra) **1**, ha a **Test** értéke nagyobb vagy egyenlő a **Low Limit** értékével és kisebb vagy egyenlő **High Limit** értékével, egyébként **0**.

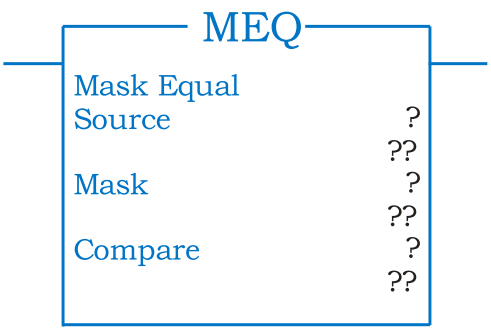


62. ábra A **HATÁRÉRTÉK VIZSGÁLAT** (**LIM**) utasítása

- **CMP** - **ÖSSZEHASONLÍTÁS** (63. ábra) **1**, ha a beírt kifejezés kiértékelése **1**, egyébként **0**.

63. ábra ÖSSZEHASONLÍTÁS (**CMP**) utasítás

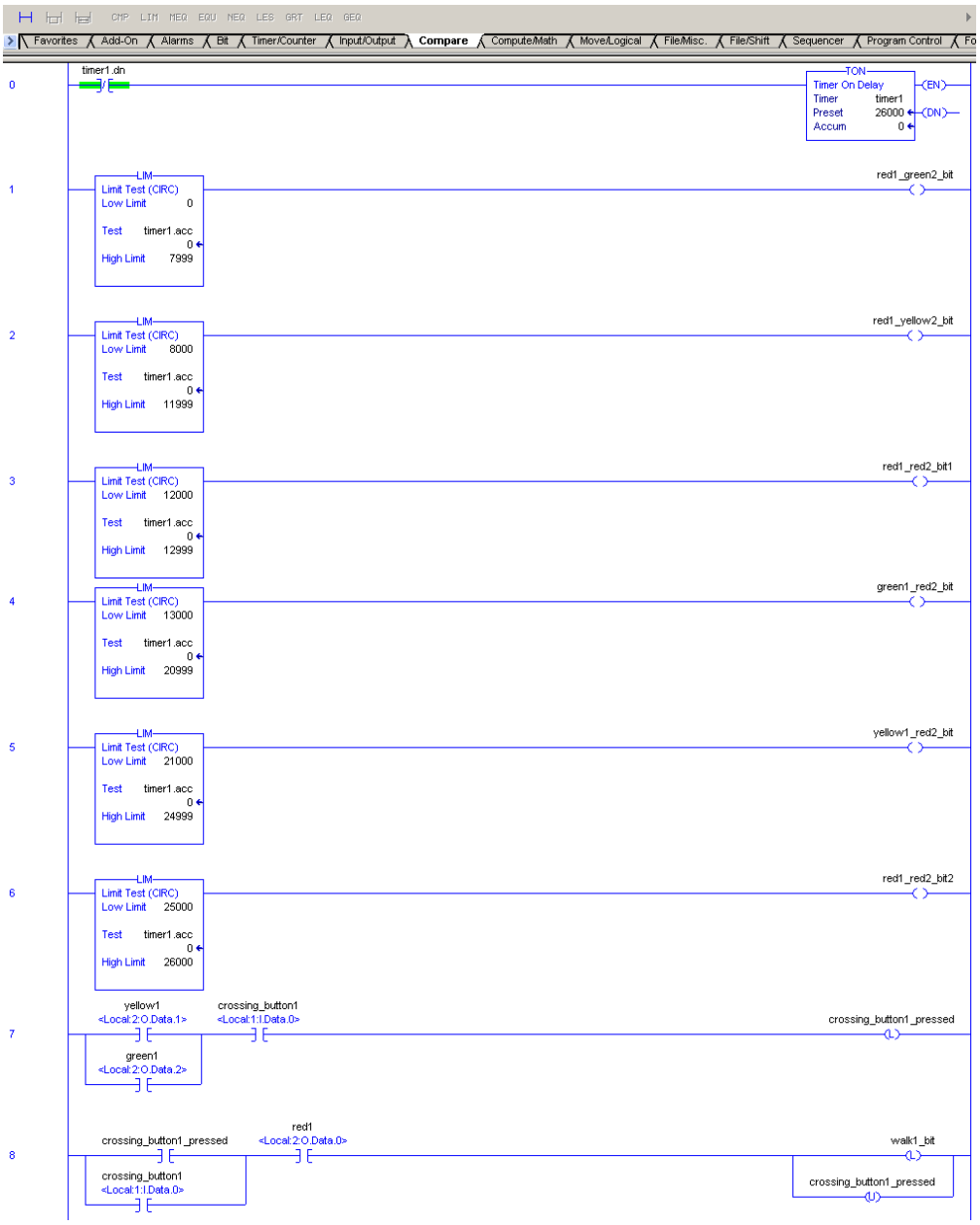
- **MEQ** - **MASZKOLT EGYENLŐ** (64. ábra) ha a **Source** értékének bináris reprezentációját maszkoljuk (**ÉS** művelet) a **Mask**-al. **1** ha az eredmény megegyezik a **Compare** értékével, egyébként **0**.



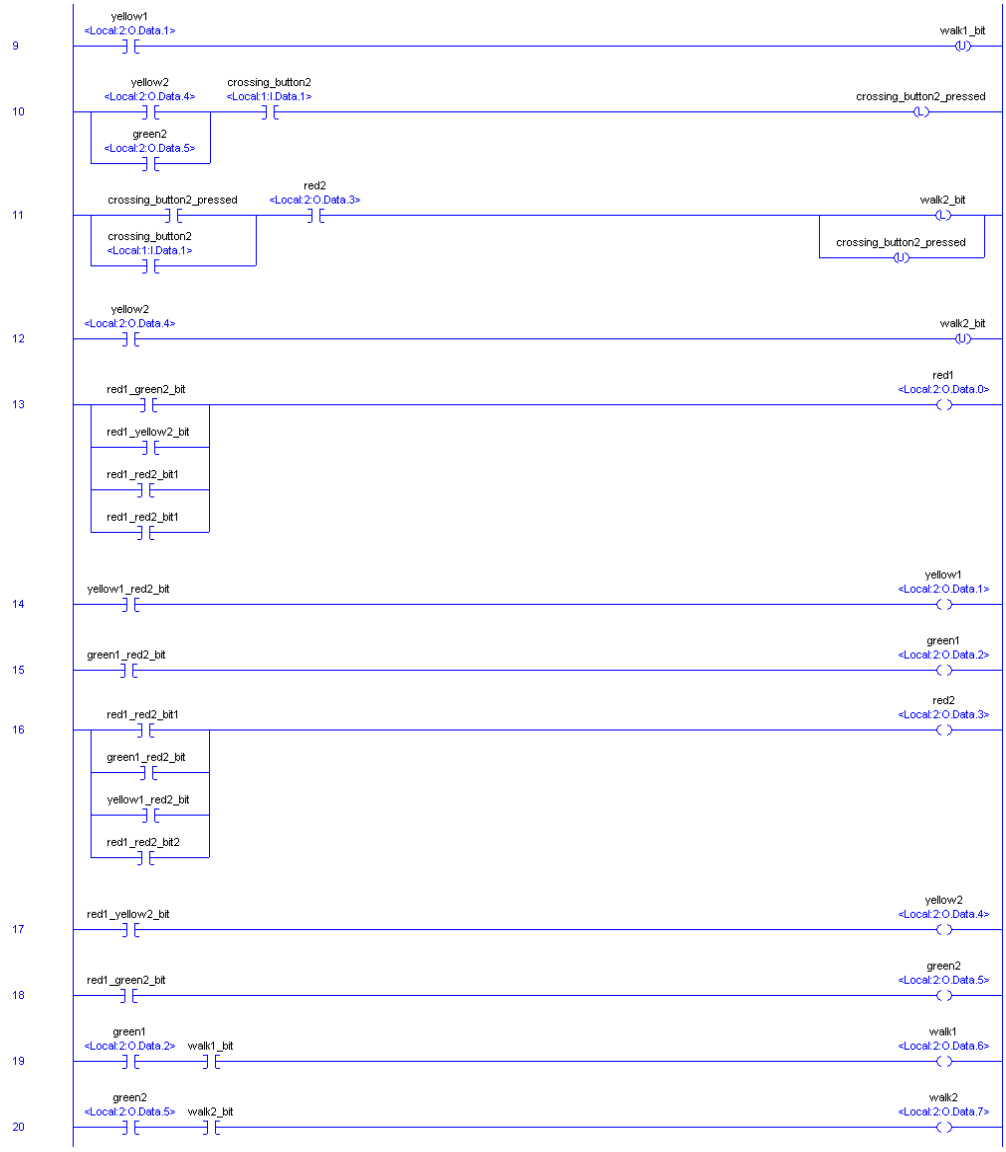
64. ábra A **MASZKOLT EGYENLŐ** (**MEQ**) utasítás

2.3.2. IV. feladat: Forgalom irányítás 2. megoldás

Oldjuk meg újra az előző feladatot egyetlen időzítő felhasználásával. A 65. és 66. ábrákon láthatjuk a feladat megoldását.



65. ábra A forgalomirányítás PLC programja



66. ábra A forgalomirányítás PLC programja

A 0. sor egy időzítőt tartalmaz, ami minden 26. s-ban újraindítja önmagát. Az 1- 6. sorok megszabják, hogy az időzítő futása (26 s) közben melyik lámpa gyulladjon fel. Ezt legkönnyebben a **LIM** utasítással lehet megoldani. Természetesen a kérdés máshogy is megoldható (pl.: **GEQ** és **LES** utasítások felhasználásával).

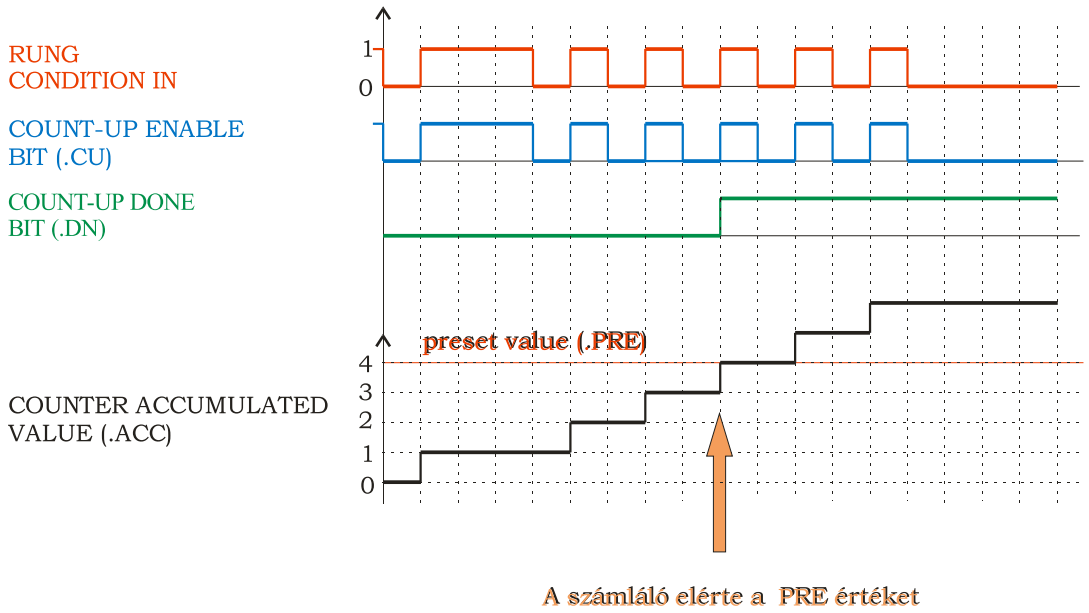
2.4. Számoljunk egy kicsit

Hasonlóan az időzítőkhöz, melyek időt számolnak, a számlálók események bekövetkezését számolják.

2.4.1. Számláló típusok és alkalmazásuk

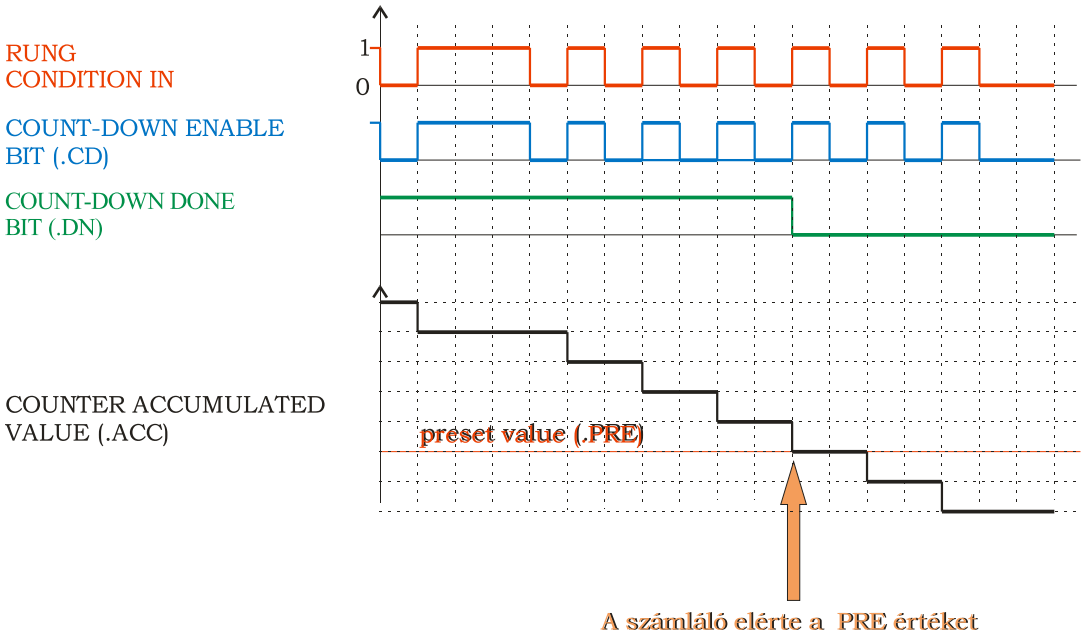
Az **RSLogix 5000** a számlálók három típusát ismeri, felfelé számláló (**CTU**), lefelé számláló (**CTD**) és fel-le számláló (**CTUD**). Ezek közül a **CTU** és **CTD** alkalmazható az **LD** nyelvben, míg a **CTUD**-t az **FBD** nyelvben lehet alkalmazni. Az **LD** nyelvben nem alkalmazható, mivel **CTU** és **CTD** együttes használatával ugyanaz az eredmény élhető el. Hasonlóan az időzítőkhöz a számlálók is rendelkeznek **tag**-el, Preset értékkel (**PRE**) és akkumulátorral (**ACC**). Állapotbitjeik a következők: túlcsordulás (**OV**), alulcsordulás (**UN**), done (**DN**) valamit számlálás (**CU**, CTU esetén és **CD** CTD esetén).

A **CTU** működését láthatjuk a 67. ábrán.



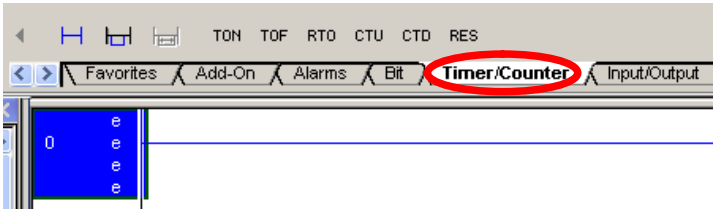
67. ábra A **CTU** működése

A **CTD** működését láthatjuk a 68. ábrán.

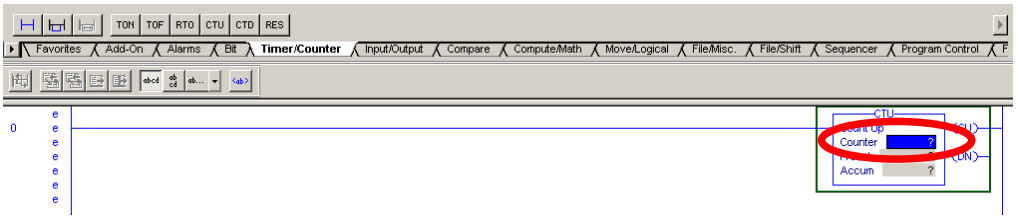


68. ábra A **CTD** működése

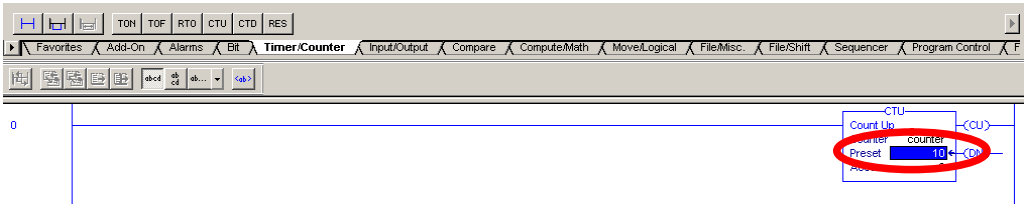
Lássunk egy példát számlálók alkalmazására. Keressük meg az eszköztárban a **Timer/Counter** fület (69. ábra). Helyezzük el a **CTU** utasítást a létraágon és rendeljünk hozzá egy **tag**-et (70. ábra). Adjuk meg a számláló **Preset** értékét (71. ábra). Végül helyezzünk el egy feltételt, aminek bekövetkezését számoljuk (72. ábra).



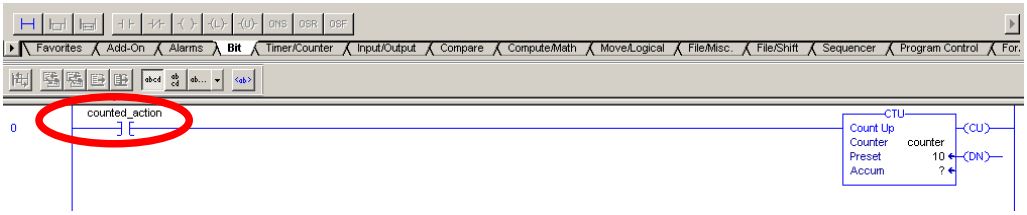
69. ábra A **Timer/Counter** kiválasztása



70. ábra A számláló címzése



71. ábra A számláló beállítása

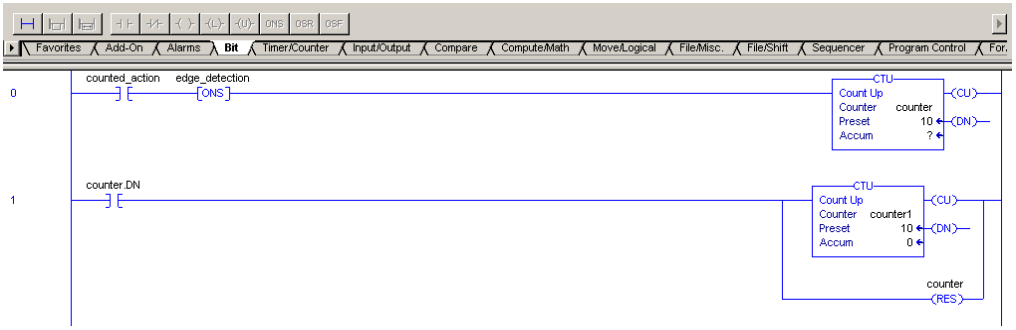


72. ábra Egy **Reset** feltétel beiktatása

Ez a rövid program azt számolja, hogy a **counted_action** feltétel hányszor vette fel az **1** értéket. A feltétel minden felfutó élére a számláló eggyel növeli az **ACC** értékét. Amint az **ACC** eléri a **PRE** értéket a **DN** bit **1** lesz. A 67. ábrán azt láthattuk, hogy miután **ACC**, elérte a **PRE** értéket az **ACC** továbbra is eggyel nő minden felfutó él alkalmával, amíg el nem éri a legnagyobb ábrázolható értéket (2147483647). Ha ez után is növelni próbáljuk, túlcsordul az értéke, ami az **OV** bit aktiválásával jár és az **ACC** értéke -2147483648 lesz.

Hasonlóan a **CTD** minden felfutó él alkalmával, eggyel csökkenti az **ACC** értékét. Amikor eléri a -2147483648 értéket és tovább csökkentjük alulcsordulás következik be. Ennek hatására az **UN** bit aktiválódik és az **ACC** értéke 2147483647 lesz.

Ahhoz, hogy az **ACC** értékét kinullázzuk, a számlálóknál és időzítőknél egyaránt a **Reset** utasítást kell alkalmazni. Ennek az utasításnak a számláló/időzítő **tag**-jét kell megadni és majd akkor állítja **0**-ra amikor a feltétele **1** lesz.
Azokban az esetekben, ha a fent említett számokat meg kell haladnunk lépcsős számlálókat alkalmazunk (73. ábra).

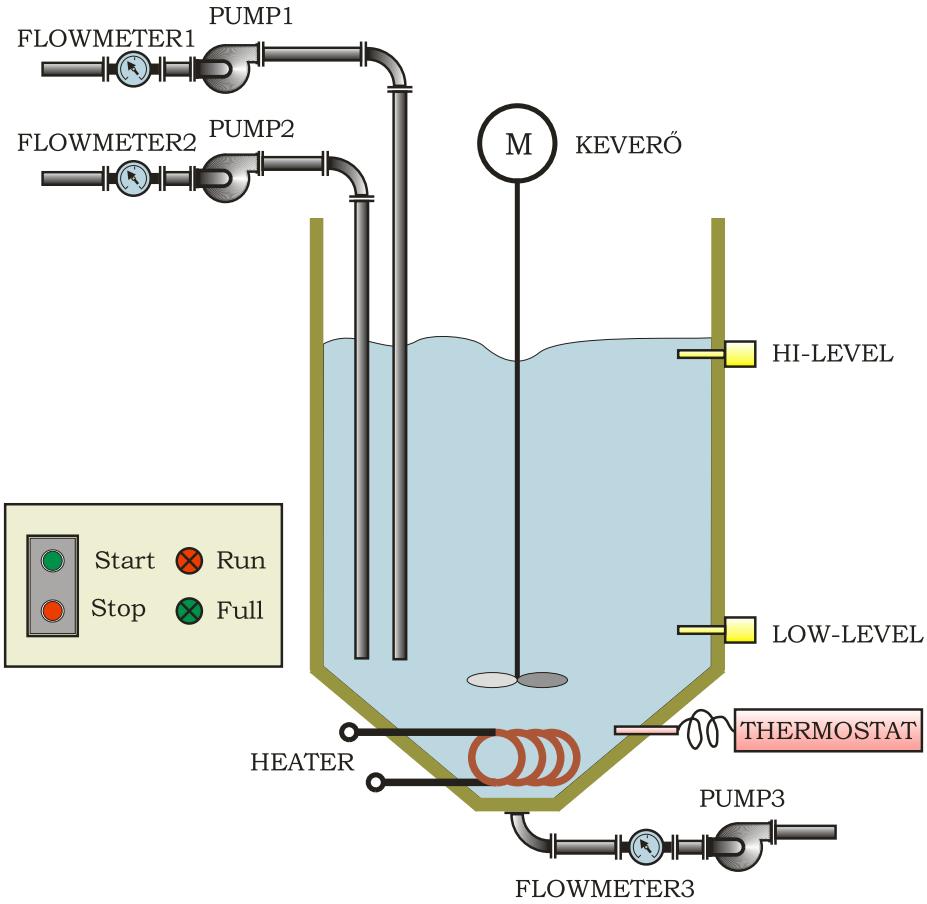


73. ábra Lépcsős számlálók

Általános esetben, amikor felfutó éleket számolunk, nem kell az **ONS** utasítást használni mivel a számlálóban már benne van. Viszont amikor a számláló **DN** bitjét használjuk a resetelésére mint ahogy a 73. ábrán tettük egy hamis felfutó élet fog észlelni és az **ACC 1**-ről fog indulni. Ennek az orvoslására **ONS**-t alkalmazunk.

2.4.2. V. feladat: Keverék elkészítése

A feladatunk, hogy egy keverőtartály vezérlő programját megírjuk. A keverőtartály a 74. ábrán látható.



74. ábra Keverőtartály

A bemenetek a 13. táblában láthatók.

13. Tábla Bemeneti változók

Bemenetek (kapcsolók)	Név	Típus	Azonosító
Nyomógomb	Start	NO	Local:1:I.Data.0
Nyomógomb	Stop	NC	Local:1:I.Data.1
Áramlásmérő	FLOWMETER1	NC	Local:1:I.Data.2
Áramlásmérő	FLOWMETER2	NC	Local:1:I.Data.3
Áramlásmérő	FLOWMETER3	NC	Local:1:I.Data.4
Szintjelző	HI-LEVEL	NO	Local:1:I.Data.5
Szintjelző	LOW-LEVEL	NO	Local:1:I.Data.6
Hőmérő	THERMOSTAT	NO	Local:1:I.Data.7

A kimenetek a 14. táblában láthatók.

14. Tábla Kimeneti változók

Kimeneti eszközök	Név	Azonosító
Szivattyú	PUMP1	Local:2:O.Data.0
Szivattyú	PUMP2	Local:2:O.Data.1
Szivattyú	PUMP3	Local:2:O.Data.2
Mágneskapcsoló	MIXER	Local:2:O.Data.3
Mágneskapcsoló	HEATER	Local:2:O.Data.4
Jelzőlámpa	Run	Local:2:O.Data.5
Jelzőlámpa	Full	Local:2:O.Data.6

A feladat a következő:

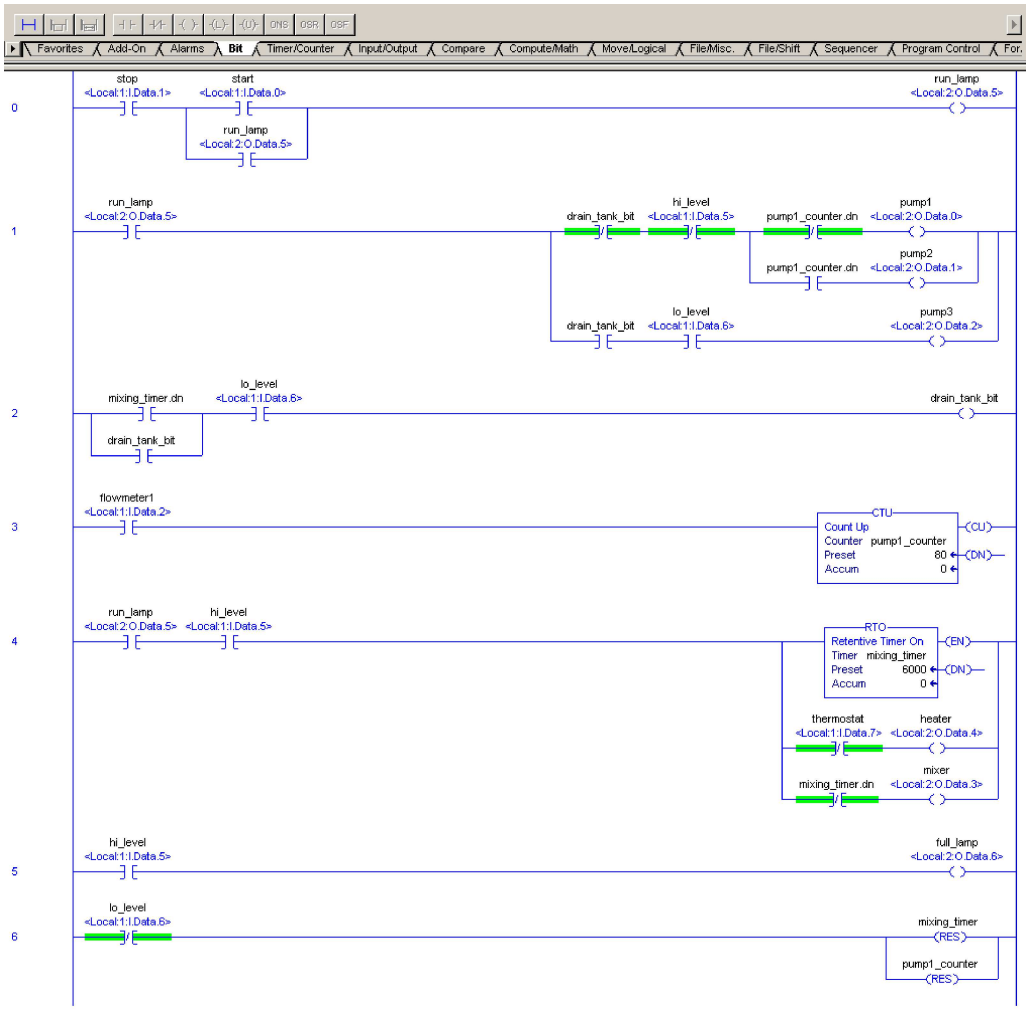
- A **Start** jelű nyomógomb megnyomására a folyamat elindul.
- Töltsük meg a tartályt a **PUMP1** és **PUMP2** segítségével.
- A **PUMP1** és **PUMP2** működtetésekor **FLOWMETER1** és **FLOWMETER2** jelzést ad minden egyes liter folyadék áthaladásakor. 80 liter után állítsuk le **PUMP1**-et.
- Amikor megtelt a tartály kezdjük el fűteni és keverni a tartályban levő keveréket.
- A beállított hőmérséklet elérését a termosztát fogja jelezni. Tartsuk a hőmérsékletet a beállított értéken.
- A keverés 60 sec után álljon le.
- Amint a keverés befejeződik, állítsuk le a fűtést és ürítsük ki a tartályt **PUMP3**-al.
- A folyamat ismétlődjön mindaddig, míg a **Stop** jelű nyomógombot meg nem nyomtuk.
- A **Run** jelű lámpával jelezzük a gép működését.
- Mindaddig, amíg a tartály tele van (a **HI-LEVEL** jelű szenzor jelet ad), jelezzük a **Full** lámpával.
- **Stop** gomb megnyomása esetén a folyamat leáll, majd **Start** gomb újbóli megnyomására újra indul.

A 75. ábrán láthatjuk a feladat megoldását.

A 0. sor egy öntartó kapcsolás, ami a **Run** lámpa működéséért felel. Mivel a **Stop** gomb **NC** ezért a **XIC** utasítással bontjuk az áramkört.

Az első sor felelős a tartály feltöltéséért és ürítéséért. A **run_lamp** tag-et felhasználjuk, hogy ellenőrizzük a **Start** gomb megnyomását. A program ezután két ágra szakad. Amikor a **drain_tank_bit 0** a tartály nincs tele így elkezdi feltölteni a tartályt **PUMP1**-el. Miután **PUMP1** 80 liter anyagot betöltött kikapcsoljuk és a maradékot **PUMP2** tölti fel mindaddig még a **HI-LEVEL** érzékelő nem jelez. Amikor a **drain_tank_bit 1**, a tartályt **PUMP3** leereszti mindaddig amíg **LO-LEVEL** érzékelő nem jelez.

A 2. sor egy öntartó kapcsolás, ami a **drain_tank_bit**-ért felel. Ez a bit azt jelzi, hogy üríteni kell a tartályt. A bit akkor válik aktívvá, amikor a keverési időt letelt. Azt, hogy a tartály kiürült a **LOW-LEVEL** érzékelő jelzi, és a **drain_tank_bit 0** lesz.



75. ábra A feladat megoldása

A 3. sor egy számlálót tartalmaz, ami a **PUMP1** által betöltött folyadék mennyiségét számolja. A **Preset** értéke 80, mivel ez a kért folyadék mennyisége. Ezt az értéket kell megváltoztatnunk, ha más arányt kívánunk beállítani.

A 4. sor felelős a fűtésért és a keverésért. Itt egy **RTO**-t használtunk az idő mérésére. Így ha a közben megnyomják a **Stop** gombot, az értéke nem fog kinullázódni. A fűtés feltétele csak annyi, hogy ha hőmérséklet pillanatnyi értéke kisebb mint a kívánt érték, kapcsoljuk be a fűtést, majd ha elértük kapcsoljuk ki. Ez a legegyszerűbb módszere a fűtésszabályozásnak, általában ettől fejlettebb módszereket használunk.

Az 5. sor felelős a **Full** lámpáért.

Amikor a tartály kiürült 6. sor kinullázza a számlálót és az időzítőt.

2.5. Haladó szint

Eddig olyan utasításokat ismertünk meg melyek a legtöbb PLC-ben jelen vannak. Most vizsgáljuk meg az **RSLogix5000** teljes kínálatát. Következzenek a matematikai utasítások.

2.5.1. Matematikai utasítások

A memóriában tárolt számokat **tag**-ekkel tudjuk elérni. Egy szám eltárolására egy bizonyos méretű memóriát kell félre tenni, más néven allokálni. A félretett memória területe határozza meg az ott tárolható szám értéktartományát. A 15. táblában láthatjuk az adat típusokat és a hozzájuk tartozó értékek terjedelmét.

15. Tábla Adattípusok

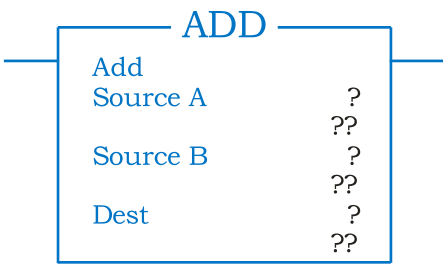
Adat-típus	Negatív	Pozitív
SINT	-127	127
INT	-32,768	32,767
DINT	-2,147,483,648	2,147,483,647
REAL	$-3.402823*10^{38}$	$1.1754944*10^{-38}$
	$-1.1754944*10^{-38}$	$3.402823*10^{38}$

Matematikai műveleteket az eszköztár **Compute/Math** füle alatt találhatjuk (76. ábra).

76. ábra Matematikai műveletek (**Compute/Math**)

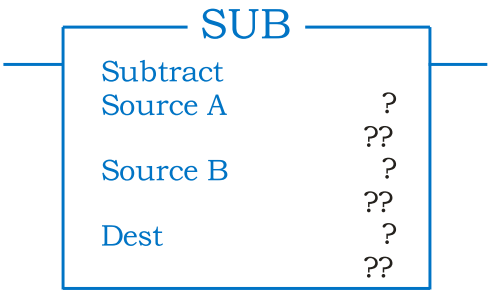
A műveletek a következők:

- **ADD** - **ÖSSZEADÁS** (77. ábra) a **Source A** és **Source B** értékét összeadja és eltárolja **Dest**-ben.



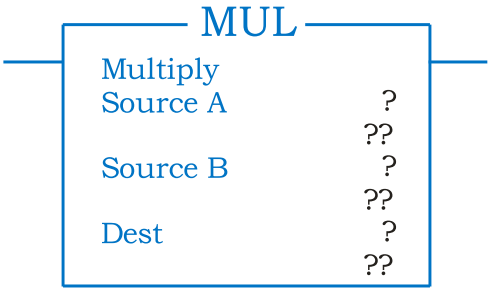
77. ábra ÖSSZEADÁS (ADD)

- **SUB** - **KIVONÁS** (78. ábra) A **Source B** értékét kivonja **Source A**-ból és eltárolja a **Dest**-ben.



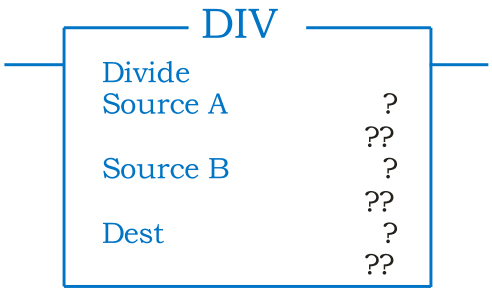
78. ábra KIVONÁS (SUB)

- **MUL** - **SZORZÁS** (79. ábra) A **Source A** és **Source B** értékét összeszorozza és eltárolja **Dest**-ben.



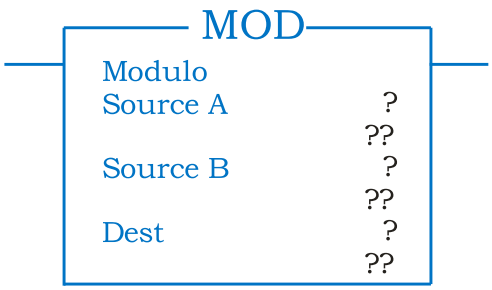
79. ábra SZORZÁS (MUL)

- **DIV** - **OSZTÁS** (80. ábra) A **Source A** értékét elosztja **Source B** értékével és az eredményt eltárolja **Dest**-ben.



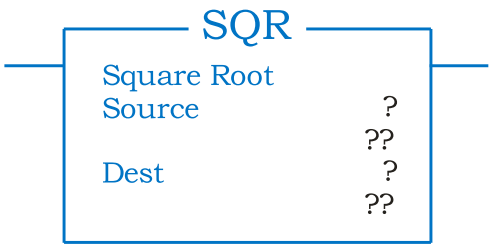
80. ábra OSZTÁS (**DIV**)

- **MOD** - **MODULUS** (81. ábra) A **Source A** értékét elosztja a **Source B** értékével és a maradékot eltárolja **Dest**-ben.



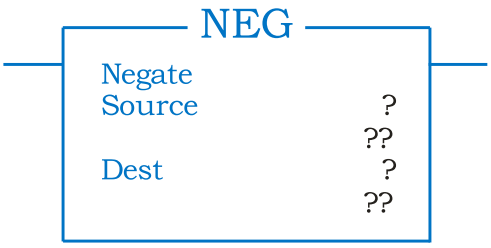
81. ábra Modulus (**MOD**)

- **SQR** - **GYÖKVONÁS** (82. ábra) **Source** értékének négyzetgyökét eltárolja **Dest**-ben.



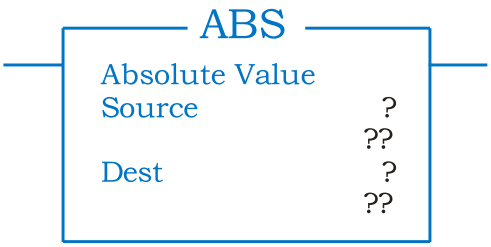
82. ábra GYÖKVONÁS (**SQR**)

- **NEG** - **ELŐJEL MEGFORDÍTÁS** (83. ábra) A **Source** értékének előjelét megfordítja, majd az új értéket eltárolja **Dest**-ben.



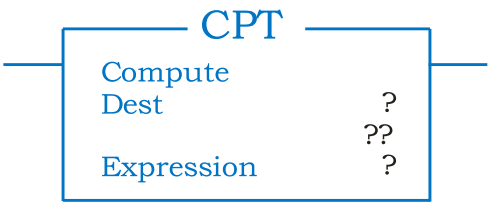
83. ábra ELŐJEL MEGFORDÍTÁS (NEG)

- **ABS** - **ABSZOLÚTÉRTÉK** (84. ábra) A **Source** abszolút értékét eltárolja **Dest**-ben.



84. ábra ABSZOLÚT ÉRTÉK (ABS)

- **CPT** - **KIÉRTÉKELÉS** (85. ábra) Kiszámítja az **Expression**-ben beírt kifejezés értékét, majd eltárolja **Dest**-ben.



85. ábra KIÉRTÉKELÉS (CPT)

Mivel négy adattípus létezik, ezért előfordulhat, hogy egy adattípusból szeretnénk másik adattípusba konvertálni. A konvertálás a következőképpen történik:

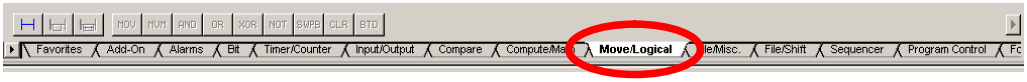
- Mielőtt az utasítás végrehajtódik a következők hajtódnak végre:
  - Ha az egyik bemenő operandus **REAL** típusú minden **SINT**, **INT** és **DINT** értéket **REAL** típusra konvertál.
  - Ha egyik bemenő operandus sem **REAL** típusú, minden **SINT** és **INT** értéket, **DINT** típusra konvertál.
- A művelet végrehajtása után az eredményül kapott (**REAL** vagy **DINT** típust) átkonvertálja az eredmény **tag** típusára.

A művelet végrehajtása után az aritmetikai bitek bővebb információval szolgálhatnak az eredményről. Az aritmetikai bitek a következők:

- **S:N** előjel. Értéke **1** ha az eredmény negatív.
- **S:C** carry. A carry jelzi, hogy az eredmény nem fér el az adattípusban. Nem része az adattípusnak.
- **S:Z** nulla. Amikor a **tag** értéke **0** a bit értéke **1**.
- **S:V** túlcsordulás. A túlcsordulás bit minden túlcsordulás bekövetkeztével **1** lesz. Amikor ez bekövetkezik minor hibát is generál. Ugyanez történik nullával való osztás esetén.

2.5.2. Logikai műveletek

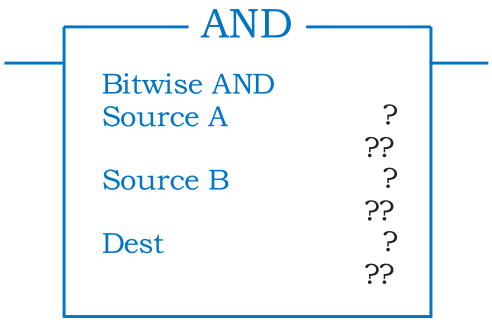
Logikai műveleteket hajtunk végre, amikor két vagy több feltételt sorosan vagy párhuzamosan kötünk. Viszont így csak biteken hajtunk végre műveleteket. A mostani logikai műveletek byte-ok csoportján vannak értelmezve. A logikai műveleteket az eszköztár **Move/Logical** fülénél találjuk (86. ábra).



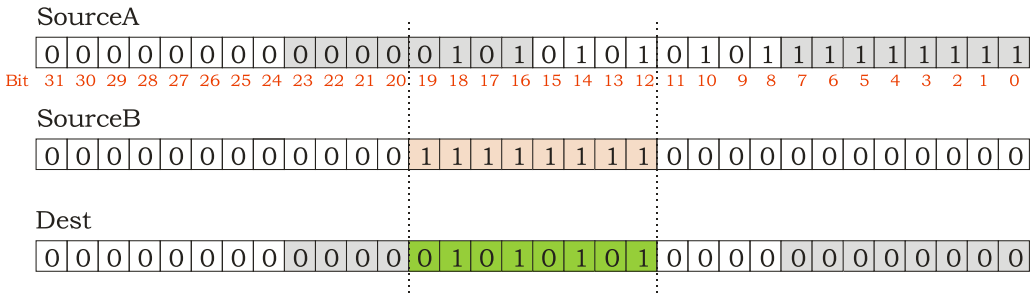
86. ábra Logikai műveletek (**Move/Logical**)

A műveletek a következők:

- **AND** - **ÉS** (87. ábra) A **Source A** és **Source B** bináris értékeit bitenként **ÉS** kapcsolatba hozza, majd az eredményt a **Dest**-ben tárolja. A 88. ábrán egy példát láthatunk erre.

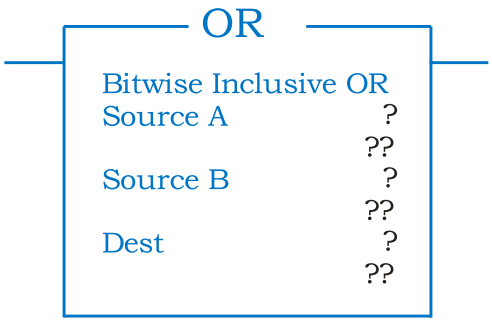


87. ábra Logikai **ÉS** művelet (**AND**)

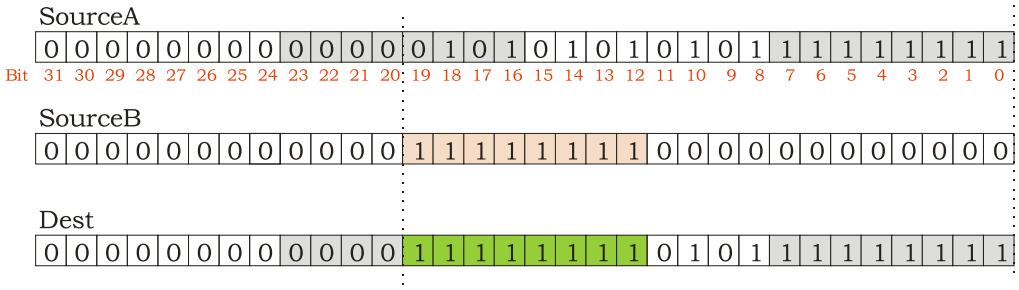


88. ábra Példa a logikai **ÉS** műveletre

- **OR** - **VAGY** (89. ábra) A **Source A** és **Source B** bináris értékeit bitenként VAGY kapcsolatba hozza, majd az eredményt a **Dest**-ben tárolja. A 90. ábrán egy példát láthatunk erre.

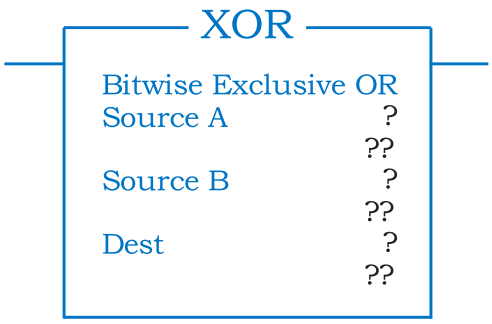


89. ábra Logikai **VAGY** művelet (**OR**)

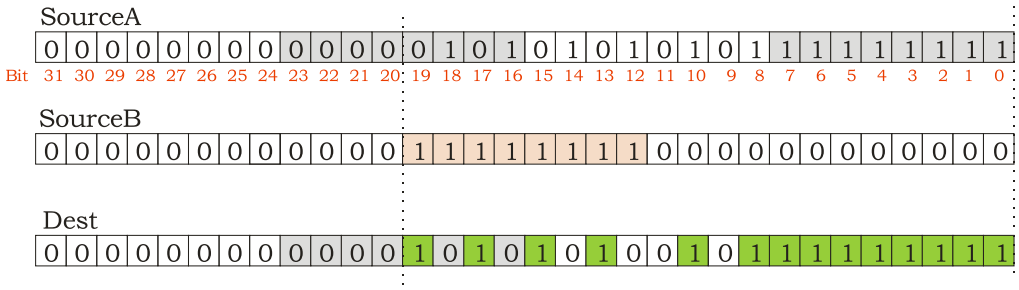


90. ábra Példa a logikai **VAGY** műveletre

- **XOR** - **KIZÁRÓ VAGY** (91. ábra) A **Source A** és **Source B** bináris értékeit bitenként KIZÁRÓ VAGY kapcsolatba hozza, majd az eredményt a **Dest**-ben tárolja. A 92. ábrán egy példát láthatunk erre.

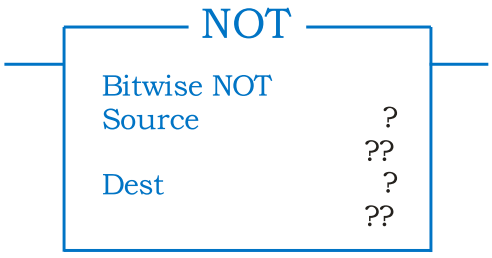


91. ábra KIZÁRÓ VAGY művelet (**XOR**)

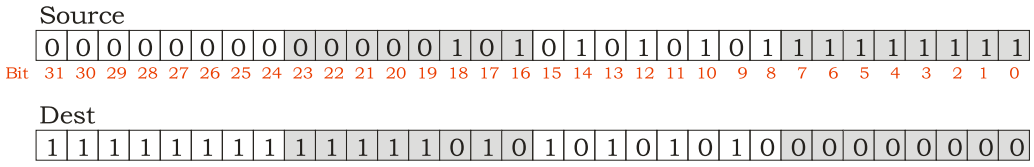


92. ábra Példa a **KIZÁRÓ VAGY** műveletre

- **NOT** - **NEGÁLÁS** (93. ábra) Bitenként negálja **Source A** bináris értékét, majd az eredményt **Dest**-ben tárolja. A 94. ábrán egy példát láthatunk erre.

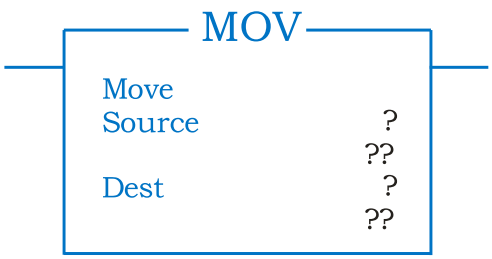


93. ábra A **NEGÁLÁS** (**NOT**) művelete



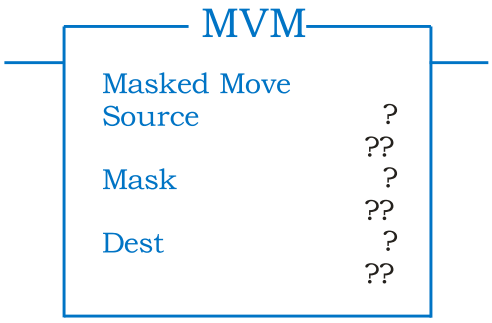
94. ábra Példa a **NEGÁLÁS** műveletre

- **MOV** - **MÁSOLÁS** (95. ábra) Bemásolja a **Source** értékét a **Dest**-be. **Source** értéke változatlan marad.

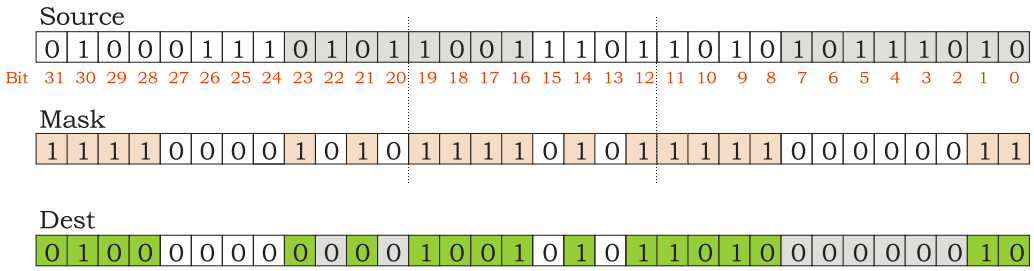


95. ábra MÁSOLÁS (**MOVE**)

- **MVM** - **MASZKON KERESZTÜLI MÁSOLÁS** (96. ábra) Bemásolja a **Source** értékét a **Dest**-be, de közben egy részét kimaszkolja. **Source** értéke változatlan marad. A 97. ábrán egy példát láthatunk erre.



96. ábra MASZKON KERESZTÜLI MÁSOLÁS (**MVM**)

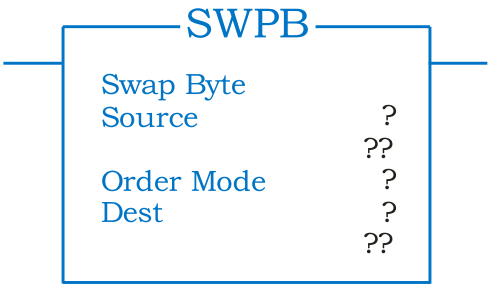


97. ábra Példa a **MASZKON KERESZTÜLI MÁSOLÁS**-ra

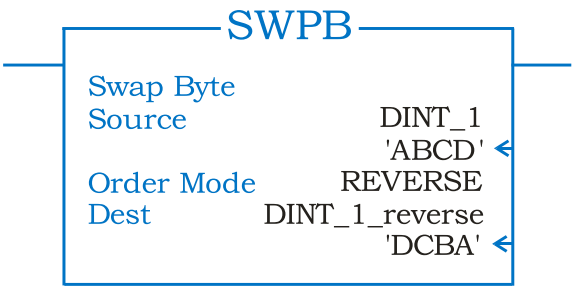
- **CLR** - **TÖRLÉS** (98. ábra) törli **Dest** értékét (**0** lesz).

98. ábra A **TÖRLÉS** művelete (**CLR**)

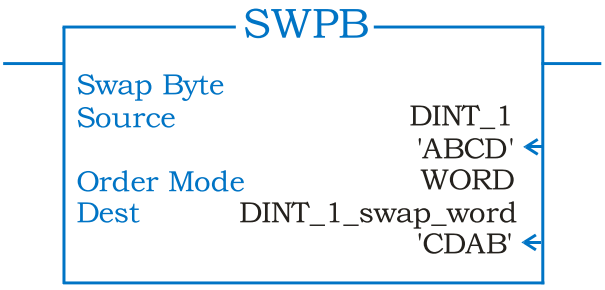
- **SWPB** - **BYTE SORREND FELCSERÉLÉS** (99. ábra) A **Source** bájtjainak sorrendjét felcseréli. A csere jellege lehet fordított, szó szintű, magas/alacsony. A 100-102. ábrákon példákat láthatunk az utasítás alkalmazására.



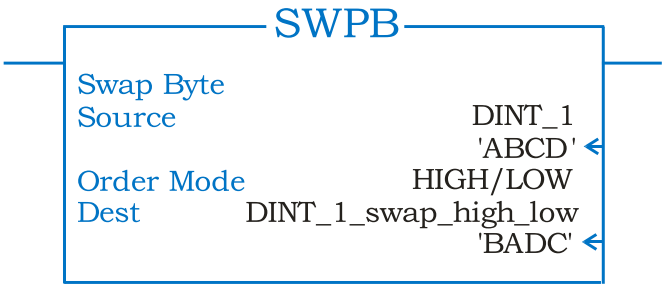
99. ábra BYTE SORREND FELCSERÉLÉS (SWPB)



100. ábra Példa a **BYTE SORREND FELCSERÉLÉS**-re

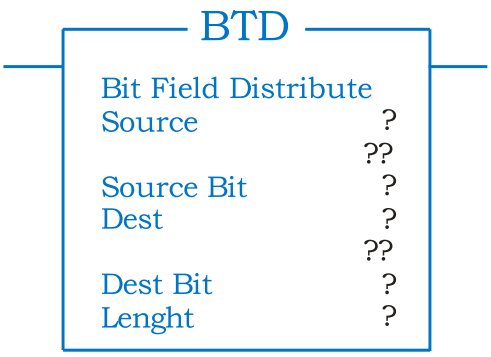


101. ábra Példa a **BYTE SORREND FELCSERÉLÉS**-re

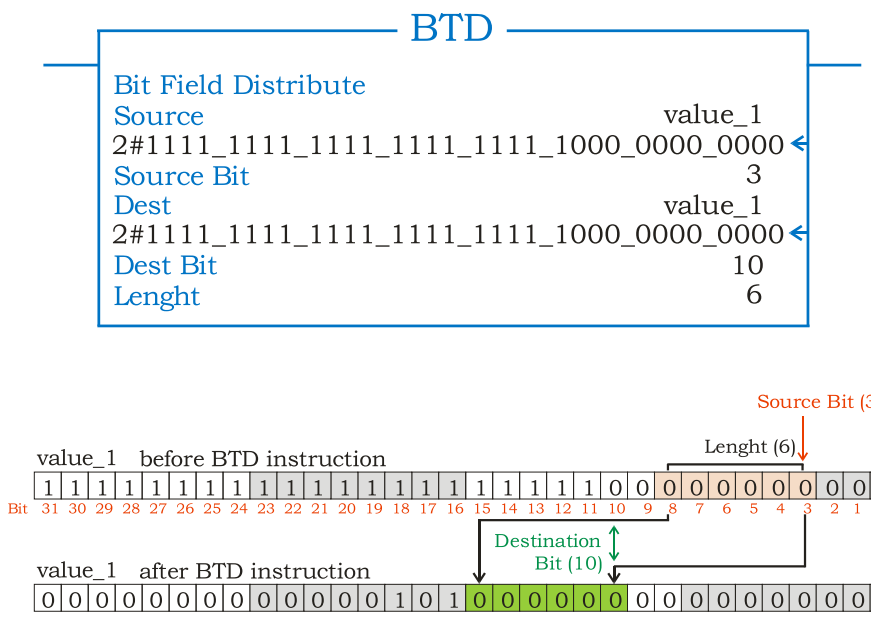


102. ábra Példa a **BYTE SORREND FELCSERÉLÉS**-re

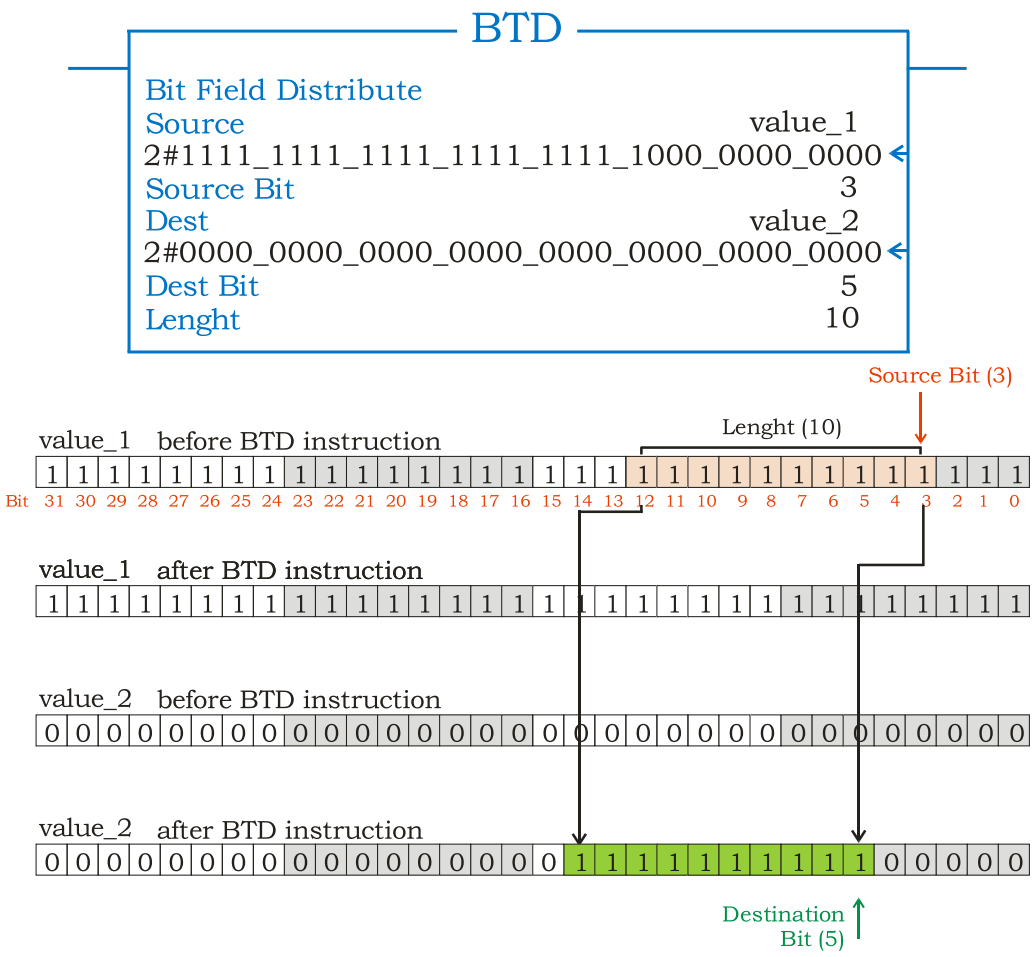
- **BTD** - **BIT SZÉTOSZTÁS** (103. ábra) Bemásolja a **Source** kiválasztott bitjeit a **Dest**-be, és közben eltolja. A 104-105. ábrákon egy-egy példát láthatunk erre.



103. ábra **BIT SZÉTOSZTÁS** (**BTD**)



104. ábra Példa a **BIT SZÉTOSZTÁS** (**BTD**)-ra



105. ábra Példa a **BIT SZÉTOSZTÁS** (**BTD**)-ra

2.5.3. Fájl műveletek

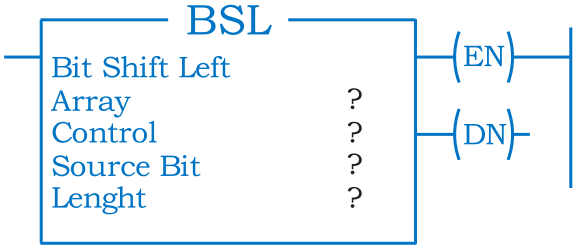
Idáig bitekkel és számokkal foglalkoztunk. A számok egy értékként vagy bájtokként voltak jelen. Ha nagyobb számhalmazokkal vagy több bájttal szeretnénk foglalkozni, fájlokat használunk. A fájlok a memóriában helyezkednek el, rendelkeznek kezdettel, tartalommal és véggel. PLC programozáskor fájlokban csoportosítunk össze számokat, ilyen például a tömb. A fájlokat használjuk még adatpufferként. A leggyakrabban alkalmazott adatpufferek a **FIFO** és a **LIFO**. A fájl műveleteket két fül alatt lehet megtalálni ezek a **File/Misc**. (106. ábra) és **File/Shift** (107. ábra). A fájl műveletek általában rendelkeznek egy **Control** mezővel. Helyezzünk egy **tag**-et ebbe a mezőbe amivel hivatkozhatunk az utasítás ezen példányára.

106. ábra Fájl műveletek (**File/Misc**)

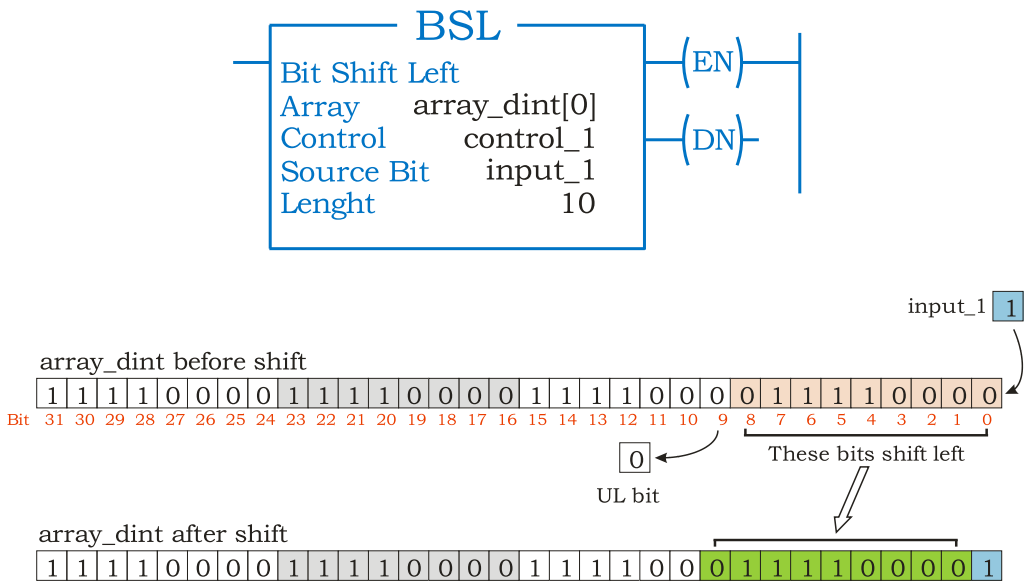
107. ábra Fájl műveletek (**File/Shift**)

A file műveletek a következők:

- **BSL** - **BALRA LÉPTETÉS** (siftelés) (108. ábra) A balra léptetés a tömbben levő kiválasztott biteket balra lépteti egy pozícióval. A 109. ábrán szemléltetjük a **BSL** működését. A **BSL** utasításnak 5 jellemzője van:
  - **EN** - Jelzi, hogy a **BSL** utasítás aktív.
  - **DN** - Jelzi, hogy a **BSL** utasítás végrehajtódott.
  - **UL** - Unload a **BSL** utasítás kimenete. Tartama az a bit, ami ki lett tolva a tömbből.
  - **ER** - Error értéke **1** ha **LEN<0**.
  - **LEN** - Meghatározza, hogy hány bitet sifteljünk.

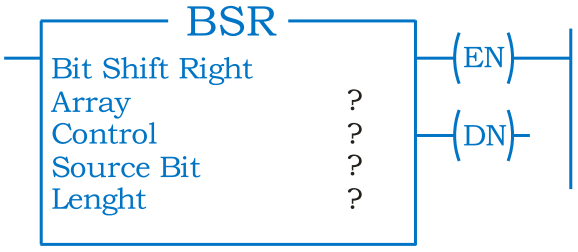


108. ábra BALRA LÉPTETÉS (**BSL**)

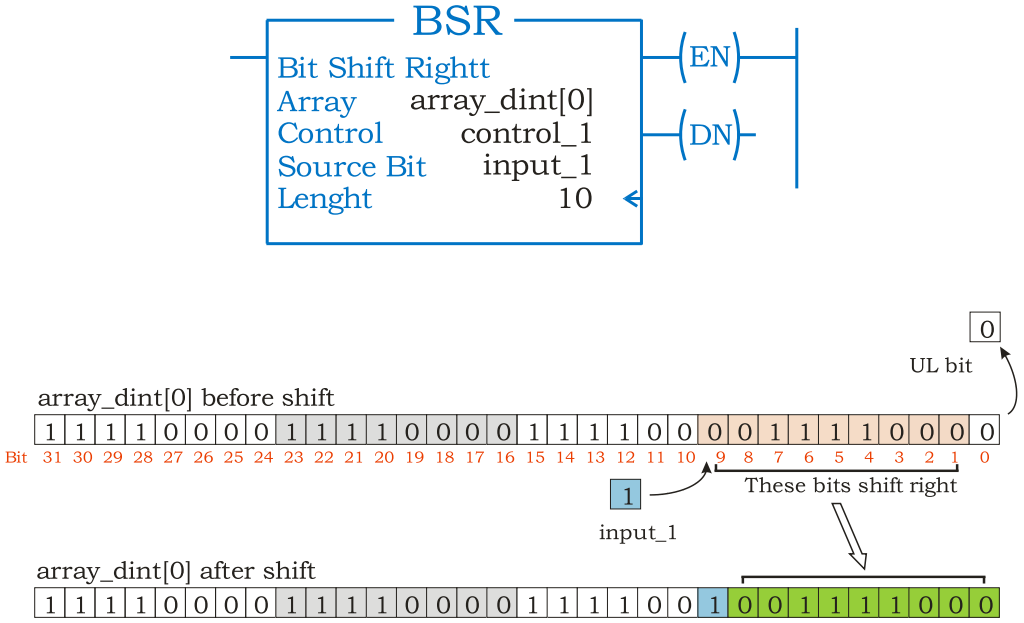


109. ábra A BALRA LÉPTETÉS (**BSL**) működése

- **BSR** - **JOBBRA LÉPTETÉS** (siftelés) (110. ábra) A jobbra léptetés a tömbben levő kiválasztott biteket jobbra lépteti egy pozícióval. A 111. ábrán szemléltetjük a **BSR** működését. A **BSR** utasításnak 5 jellemzője van:
  - **EN** - Jelzi, hogy a **BSR** utasítás aktív.
  - **DN** - Jelzi, hogy a **BSR** utasítás végrehajtódott.
  - **UL** - **Unload** a **BSL** utasítás kimenete. Tartama az a bit, ami ki lett tolva a tömbből.
  - **ER** - **Error** értéke **1** ha **LEN<0**.
  - **LEN** - Meghatározza, hogy hány bitet shifteljünk.

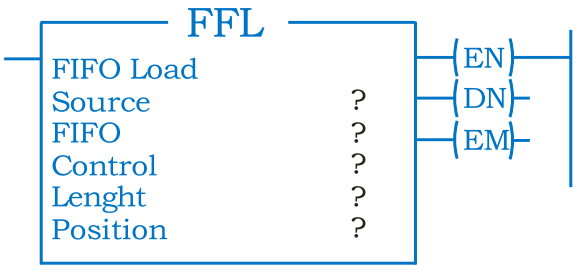


110. ábra JOBBRA LÉPTETÉS (**BSR**)

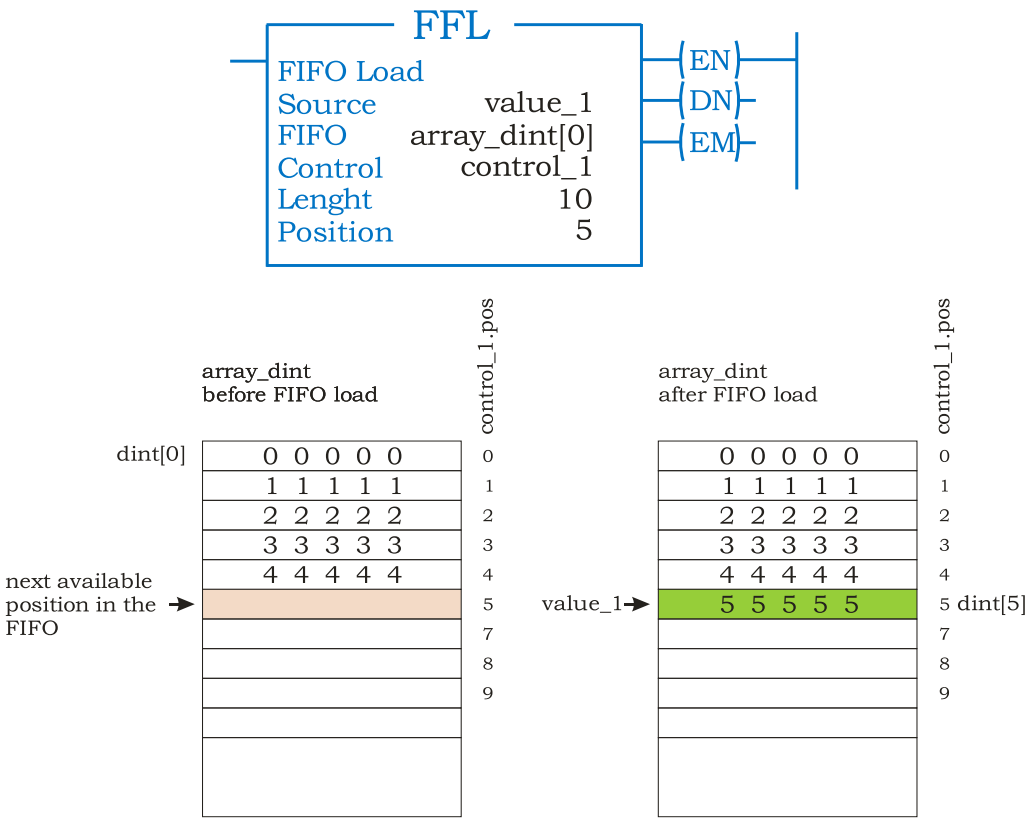


111. ábra A **JOBBRA LÉPTETÉS** (**BSR**) működése

- **FFL** - **FIFO BETÖLTÉS** (112. ábra) az **FFL** betölti a **Source** értékét a **FIFO**-ba. A 113. ábrán szemléltetjük ennek a működését. Az **FFL** utasításnak 5 jellemzője van:
  - **EN** - Jelzi, hogy a **FFL** utasítás aktív.
  - **DN** - Jelzi, hogy a **FIFO** betelt. Ez a bit megakadályozza további értékek betöltését.
  - **EM** - Jelzi, hogy a **FIFO** üres.
  - **LEN** - Meghatározza a maximális elemszámot.
  - **POS** - Meghatározza a következő hely pozícióját.

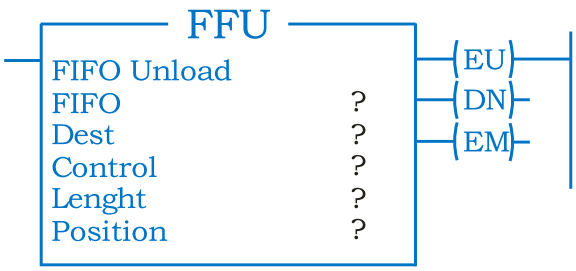


112. ábra FIFO BETÖLTÉS (**FFL**)

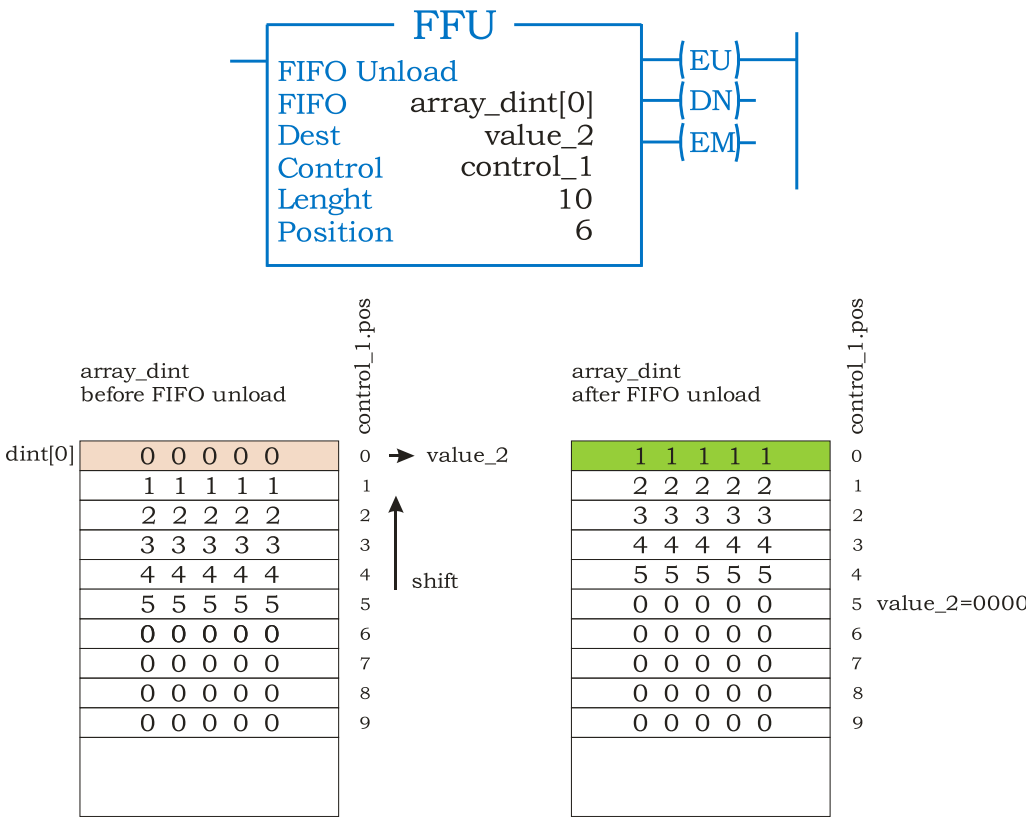


113. ábra Példa a **FIFO BETÖLTÉS** (**FFL**) utasításra

- **FFU** - **FIFO ÜRÍTÉS** (114. ábra) az **FFU** kiveszi a **0** pozíció (első pozíció) értékét és eltárolja a **Dest**-ben. A **FIFO** maradék adatai egy pozícióval eltolódnak. A 115. ábrán szemléltetjük ennek a működését. Az **FFU** utasításnak 5 jellemzője van:
  - **EN** - Jelzi, hogy a **FFU** utasítás aktív.
  - **DN** - Jelzi, hogy a **FIFO** betelt. Ez a bit megakadályozza további értékek betöltését.
  - **EM** - Jelzi, hogy a **FIFO** üres.
  - **LEN** - Meghatározza a maximális elemszámot.
  - **POS** - Meghatározza a következő hely pozícióját.

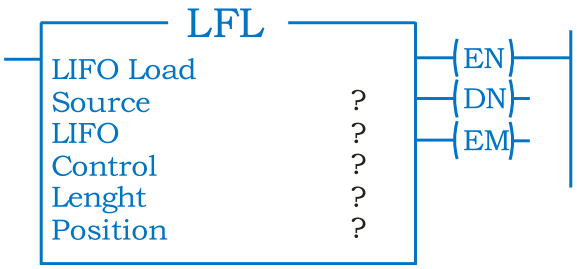


114. ábra FIFO ÜRÍTÉS (**FFU**)

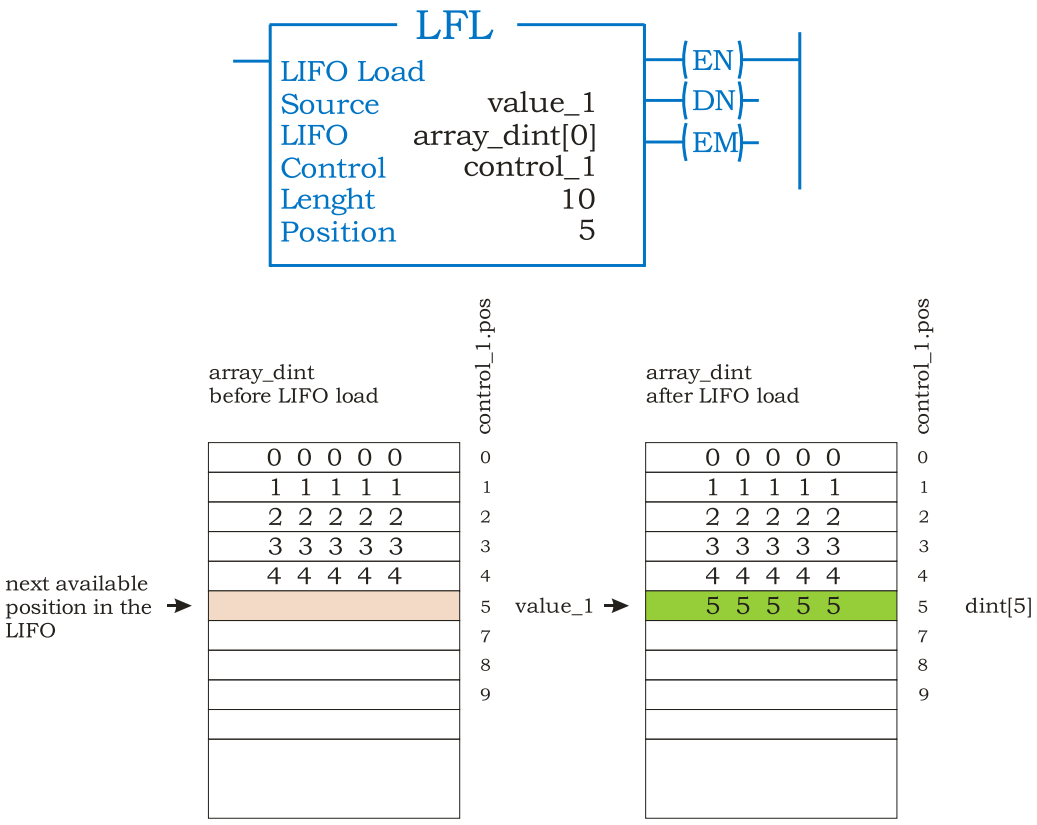


115. ábra Példa a **FIFO ÜRÍTÉS** (**FFU**) utasításra

- **LFL** - **LIFO BETÖLTÉS** (116. ábra) az **LFL** betölti a **Source** értékét a **LIFO**-ba. A 117. ábrán szemléltetjük ennek a működését. Az **LFL** utasításnak 5 jellemzője van:
  - **EN** - Jelzi, hogy a **LFL** utasítás aktív.
  - **DN** - Jelzi, hogy a **LIFO** betelt. Ez a bit megakadályozza további értékek betöltését.
  - **EM** - Jelzi, hogy a **LIFO** üres.
  - **LEN** - Meghatározza a maximális elemszámot.
  - **POS** - Meghatározza a következő hely pozícióját.

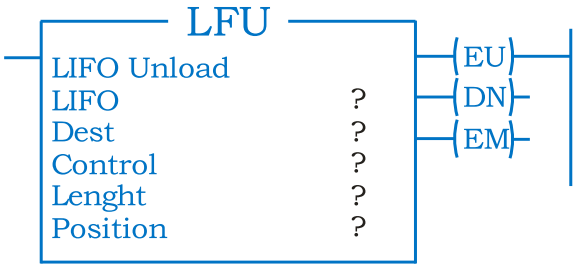


116. ábra LIFO BETÖLTÉS (**LFL**)

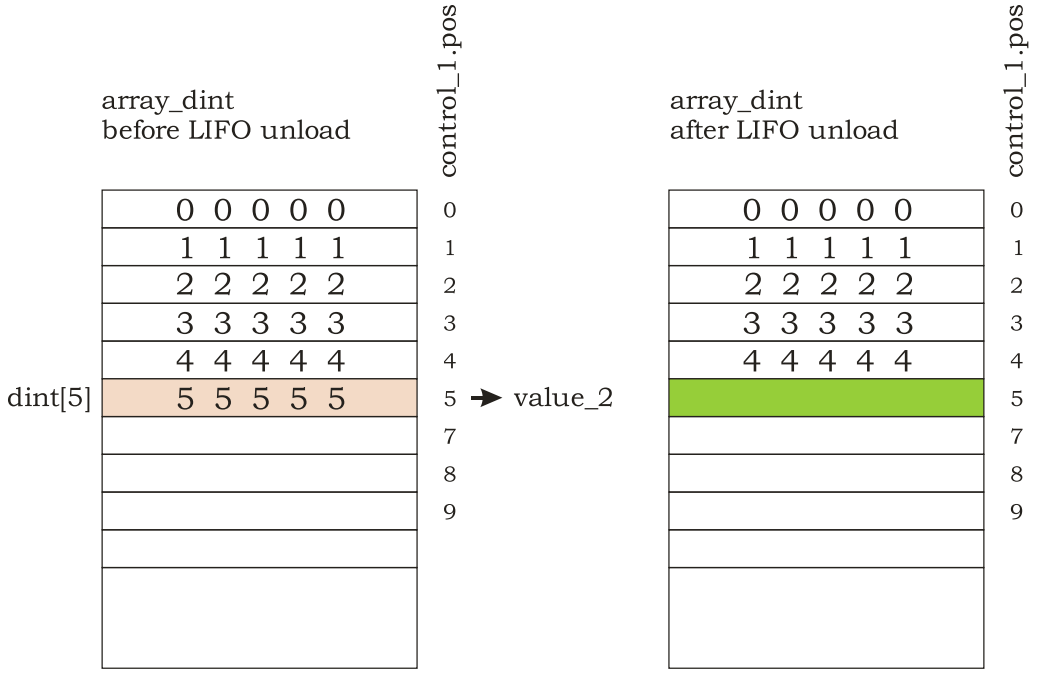


117. ábra Példa a **LIFO BETÖLTÉS** (**LFL**) utasításra

- **LFU** - **LIFO ÜRÍTÉS** (118. ábra) az **LFU** kiveszi a **POS** pozíció értékét és eltárolja a **Dest**-ben. **POS** pozíció új értéke **0** lesz. A 119. ábrán szemléltetjük ennek a működését. Az **LFU** utasításnak 5 jellemzője van:
  - **EN** - Jelzi, hogy a **LFU** utasítás aktív.
  - **DN** - Jelzi, hogy a **LIFO** betelt. Ez a bit megakadályozza további értékek betöltését.
  - **EM** - Jelzi, hogy a **LIFO** üres.
  - **LEN** - Meghatározza a maximális elemszámot.
  - **POS** - Meghatározza a következő hely pozícióját.

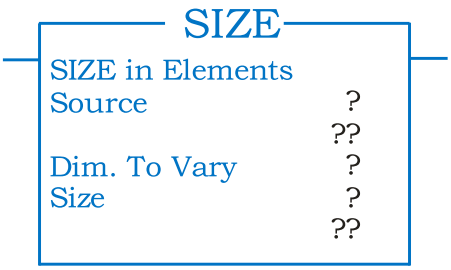


118. ábra LIFO ÜRÍTÉS (**LFU**)



119. ábra Példa a **LIFO ÜRÍTÉS** (**LFU**) utasításra

- **SIZE** - **MÉRET** (120. ábra) megmondja egy tömb méretét egy kiválasztott dimenziója mentén. **Dim**. **To Vary** értéke (**0**, **1**, **2**) lehet. A **Size** tag tartalmazza az eredményt.

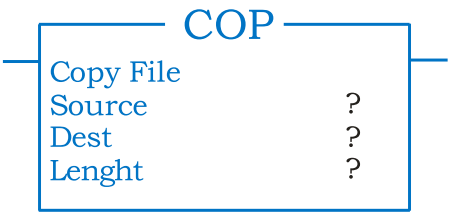


120. ábra MÉRET (**SIZE**)

- **FLL** - **FILE FELTÖLTÉS** (121. ábra) feltölti a tömböt a **Source** értékével. A **Source** értéke változatlan marad.

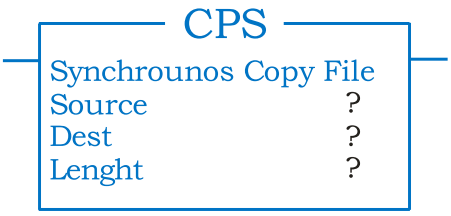
121. ábra FILE FELTÖLTÉS (**FLL**)

- **COP** - **FILE MÁSOLÁS** (122. ábra) a **Source** értékeit bemásolja a tömbbe. A **Source** értéke változatlan marad.



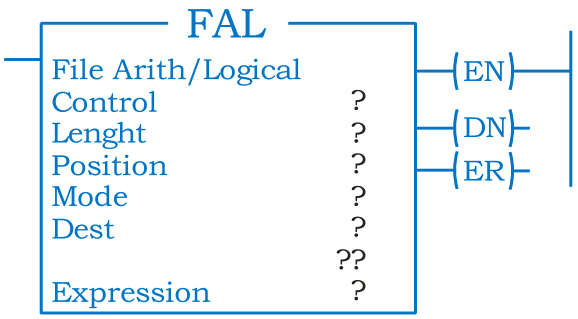
122. ábra FILE MÁSOLÁS (**COP**)

- **CPS** - **SZINKRON FILE MÁSOLÁS** (123. ábra) a **Source** értékeit bemásolja a tömbbe. Másolás közben nem lehet frissíteni az adatokat. A **Source** értéke változatlan marad.



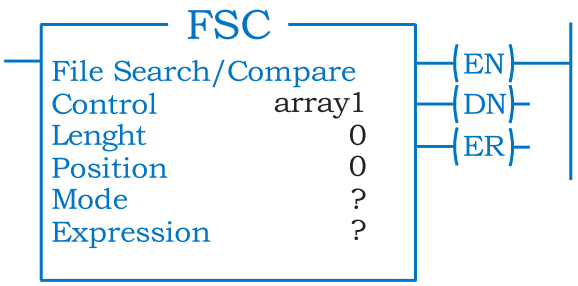
123. ábra SZINKRON FILE MÁSOLÁS (**CPS**)

- **FAL** - **FILE ARITMETIKA ÉS LOGIKA** (124. ábra) másolás, aritmetikai és logikai műveleteket hajt végre a tömbben tárol elemeken. Ez az utasítás a **CPT** tömb változata. A **FAL** utasításnak 5 jellemzője van:
  - **EN** - Jelzi, hogy a **FAL** utasítás aktív.
  - **DN** - Jelzi, hogy elértük az utolsó elemet.
  - **ER** - Jelzi, hogy túlcsordulás történt **(S:V)**. Az utasítás leáll mindaddig még az **ER** bit aktív.
  - **LEN** - Meghatározza hány elemen hajtsunk végre műveletet.
  - **POS** - Meghatározza az aktuális elem pozícióját.



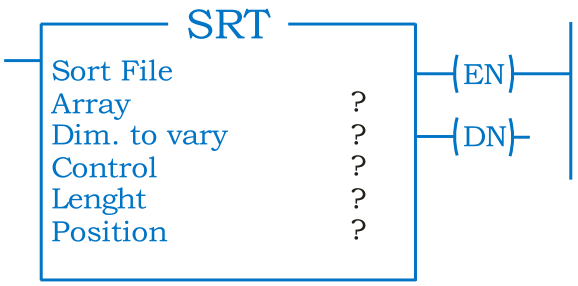
124. ábra FILE ARITMETIKA ÉS LOGIKA (**FAL**)

- **FSC** - **FILE-BAN KERESÉS ÉS ÖSSZEHASONLÍTÁS** (125. ábra) elemenként összehasonlítja egy tömb értékeit a megadott kifejezés szerint. Ez az utasítás a **CMP** tömb változata. Az **FSC** utasításnak 7 jellemzője van:
  - **EN** - Jelzi, hogy a **FSC** utasítás aktív.
  - **DN** - Jelzi, hogy elértük az utolsó elemet.
  - **ER** - Az error bit nincs alkalmazva.
  - **IN** - Az **FSC** utasítás egyezést talált. Amíg aktív, a művelet leáll.
  - **FD** - Az **FSC** utasítás egyezést talált.
  - **LEN** - Meghatározza hány elemen hajtsunk végre műveletet.
  - **POS** - Meghatározza az aktuális elem pozícióját.



125. ábra FILE-BAN KERESÉS ÉS ÖSSZEHASONLÍTÁS (FSC)

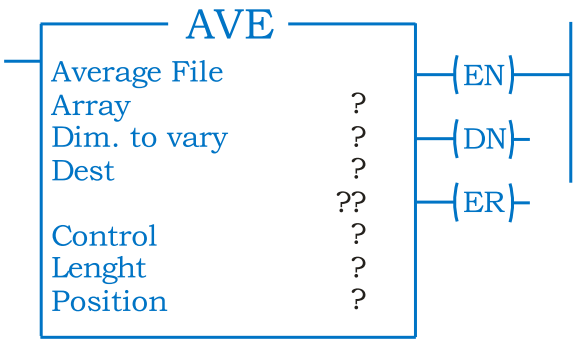
- **SRT** - **FILE ELRENDEZÉS** (126. ábra) növekvő sorrendbe helyezi a tömb elemeit egy dimenzió mentén. Az **SRT** utasításnak 5 jellemzője van:
  - **EN** - Jelzi, hogy a **SRT** utasítás aktív.
  - **DN** - Jelzi, hogy elértük az utolsó elemet.
  - **ER** - Jelzi, hogy **LEN<0** vagy **POS<0**. Bármely ezek közül egy major hibát generál.
  - **LEN** - Meghatározza hány elemen hajtsunk végre műveletet.
  - **POS** - Meghatározza az aktuális elem pozícióját.



126. ábra FILE ELRENDEZÉS (SRT)

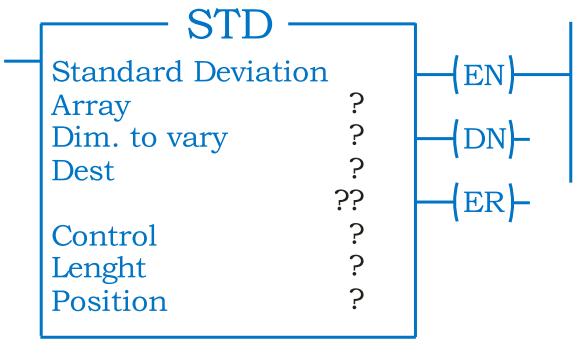
- **AVE** - **FILE ÁTLAG** (127. ábra) kiszámolja egy tömb átlag értékét. Az **AVE** utasításnak 5 jellemzője van:
  - **EN** - Jelzi, hogy a **AVE** utasítás aktív.
  - **DN** - Jelzi, hogy elértük az utolsó elemet.

- **ER** - Jelzi, hogy túlcsordulás történt (**S:V**). Az utasítás leáll mindaddig még az **ER** bit aktív. A **POS** tárolja a túlcsordulást okozó elem pozícióját.
- **LEN** - Meghatározza hány elemen hajtsunk végre műveletet.
- **POS** - Meghatározza az aktuális elem pozícióját.



127. ábra **FILE ÁTLAG** (**AVE**)

- **STD** - **FILE SZÓRÁS** (128. ábra) kiszámolja egy tömb szórását egy dimenzió mentén. Az **STD** utasításnak 5 jellemzője van:
  - **EN** - Jelzi, hogy a **STD** utasítás aktív.
  - **DN** - Jelzi, hogy elértük az utolsó elemet.
  - **ER** - Jelzi, hogy túlcsordulás történt (**S:V**). Az utasítás leáll, mindaddig míg az **ER** bit aktív. A **POS** tárolja a túlcsordulást okozó elem pozícióját.
  - **LEN** - Meghatározza hány elemen hajtsunk végre műveletet.
  - **POS** - Meghatározza az aktuális elem pozícióját.



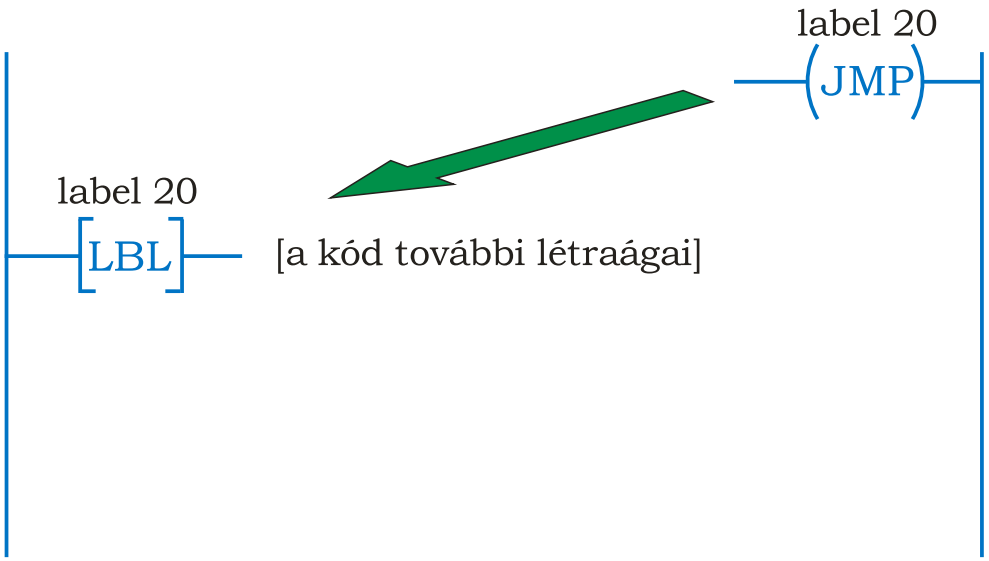
128. ábra FILE SZÓRÁS (**STD**)

2.5.4. Program strukturálása

Ahogy a feladatok egyre komplikáltabbak lesznek, úgy a programunk is egyre nagyobb és egyre nehezebben kezelhető lesz. Egy határon túl a programunk túl nagy lesz ahhoz, hogy átlássuk. A program egyes részeit csak egyszer, míg másokat többször kell végrehajtani. Ez így van minden programozási nyelvben. A programozási nyelv meghatározza, hogy a felsorolt problémákat hogyan kezeljük.

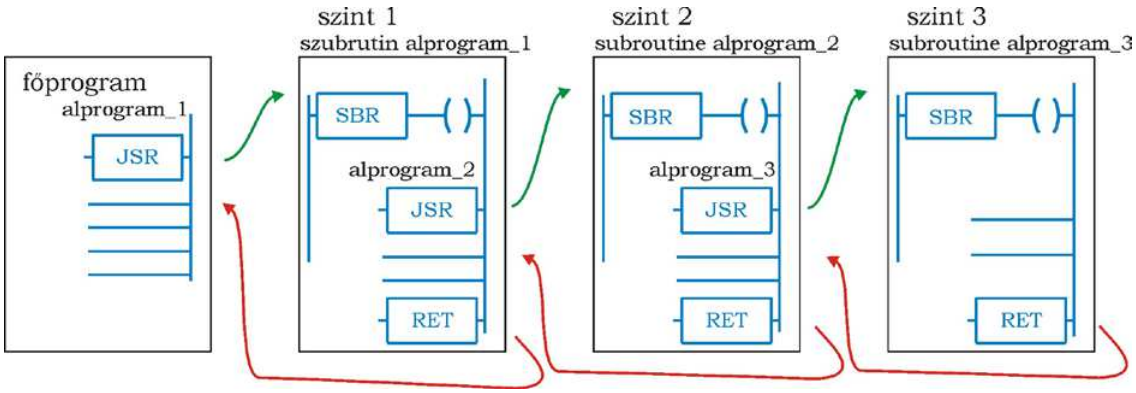
Fejlettebb nyelvek függvényeket alkalmaznak, hardware közeli nyelvek címkéket és szubrutinokat.

A címke (**Label**) egy azonosító a programban, amire egy ugró utasítás hivatkozhat. Általában olyan kódrészlethez tesszük, amit többször kell végrehajtani. Amint a program a címkéhez ugrott, a futása onnan folytatódik. Így a program bizonyos részei átugorhatóak (129. ábra).

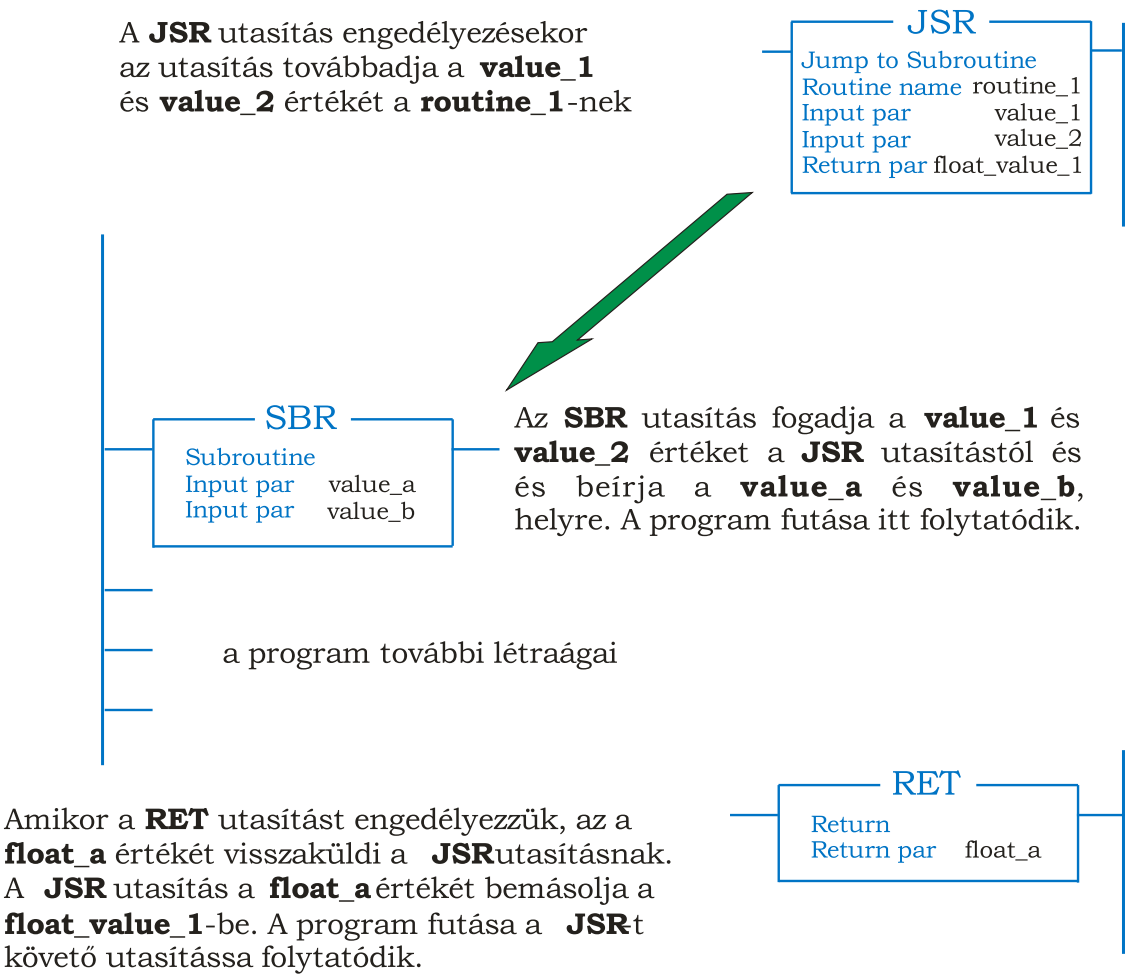


129. ábra Ugrás címkére

A szubrutinok egy másik programban elhelyezett kódrészletek, amiket mi hívunk meg. A fő különbség címke és szubrutin között, hogy a szubrutin végrehajtása után, a program visszatér a szubrutin meghívását követő sorra. Egy szubrutinba beágyazhatunk további szubrutinokat is. Mindegyik a saját meghívó sorához tér majd vissza (130. ábra). Régebbi típusú PLC-knél nem volt lehetőség a szubrutinoknál a paraméterek átadására. Az **RSLogix 5000**-nél már erre is van lehetőség (131. ábra).



130. ábra Szubrutin alkalmazása



131. ábra Szubrutin alkalmazása

132 ábra Programvezérlés (**Program Control**)

A programvezérlés utasításokat az eszköztár **Program Control** fülénél találjuk meg (132 ábra). Az utasítások a következők:

- **LBL** - **CÍMKE** (133 ábra) azonosítót rendel egy programsorhoz, amihez ugrani lehet.

133 ábra CÍMKE (**LBL**)

- **JMP** - **UGRÁS** (134 ábra) egy címkéhez ugrik. Ugorhatunk előre és visszafelé is. Ha az utasítás feltétele **0** a program végrehajtásán nem változtat.

134 ábra **UGRÁS** (**JMP**)

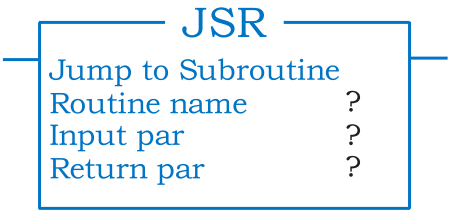
- **SBR** - **SZUBRUTIN** (135 ábra) egy szubrutin kezdetét jelöli és adatot ad át. Az **Input par** tartalmazza a tag-et amit átadtunk a szubrutinnak. Több paraméter is átadható a szubrutinnak egyszerre.

135 ábra SZUBRUTIN (**SBR**)

- **RET** - **VISSZATÉRÉS** (136 ábra) egy szubrutin végét jelöli és visszaad egy értéket.

136 ábra VISSZATÉRÉS (**RET**)

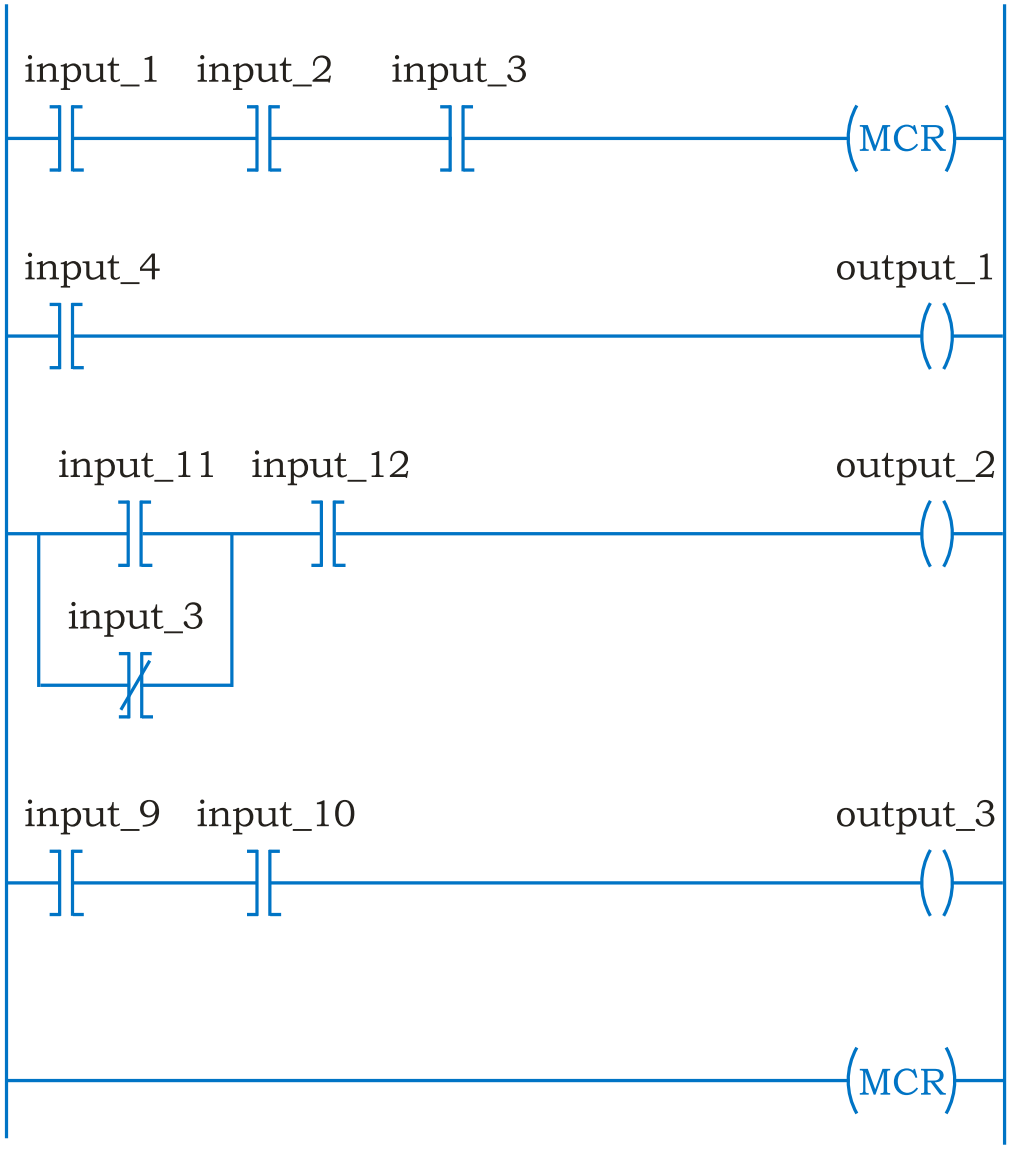
- **JSR** - **SZUBRUTIN MEGHÍVÁSA** (137 ábra) meghív egy szubrutint és paramétereket ad át neki.



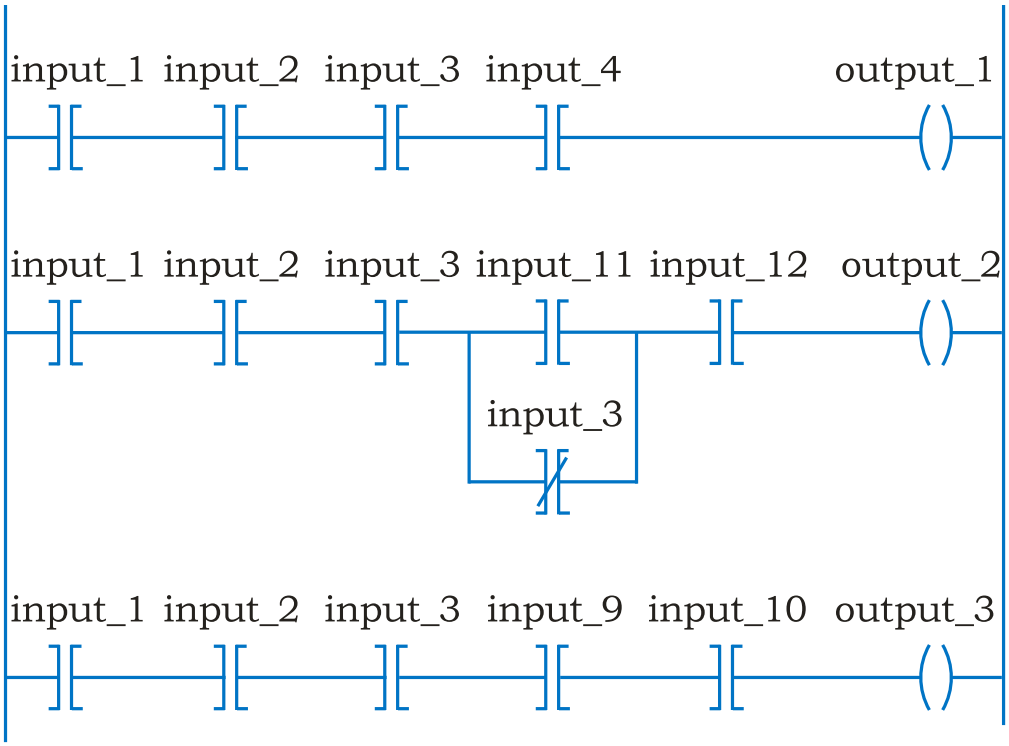
137 ábra SZUBRUTIN MEGHÍVÁSA (**JSR**)

- **MCR** - **MESTER RELÉ** (138 ábra) a mester relé utasítást párokban alkalmazzuk. Egy zónát alkot, amit feltételhez kötötten kihagyhatunk a végrehajtásból (138, 139 ábra).

138 ábra MESTER RELÉ (MCR)



139. bra Példa a **MESTER RELÉ** (**MCR**) alkalmazására

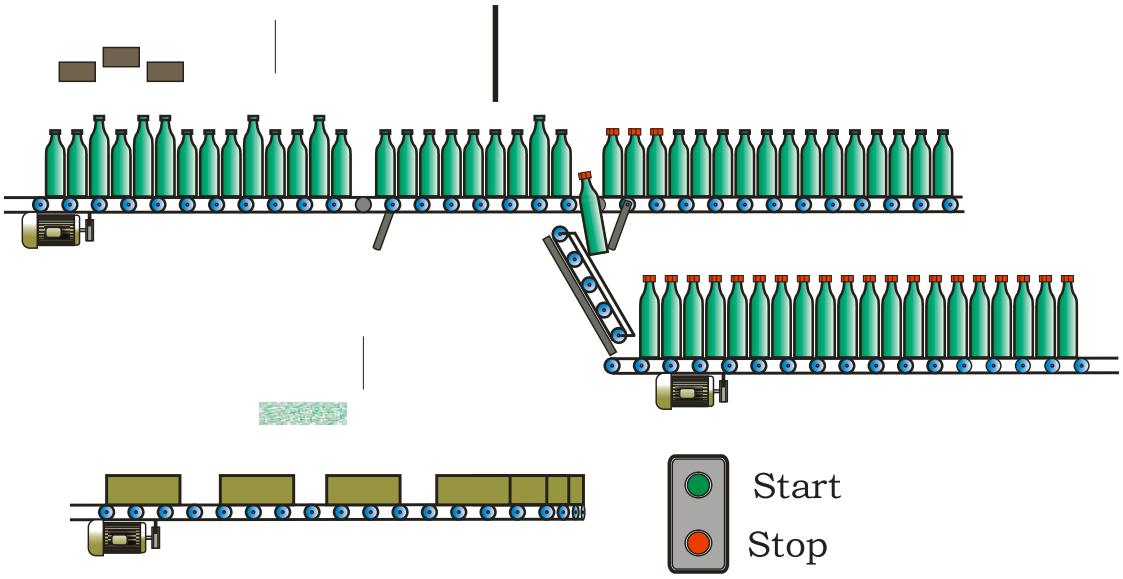


140. ábra Ugyanaz a példa a **MESTER RELÉ** (**MCR**) alkalmazása nélkül

Ezek a leggyakrabban alkalmazott programvezérlő utasítások. Lássunk egy példát az alkalmazásukra.

2.5.5. VI. feladat: Palackozó gépsor

Egy palackozó gépsort kell működtetnünk. A rendszert a 141. ábrán láthatjuk.



141. ábra Palackozó gépsor

A bemenetek a 16. táblában láthatók.

16. Tábla Bemeneti változók

Bemenetek (kapcsolók)	Név	Típus	Azonosító
Nyomógomb	Start	NO	Local:1:I.Data.0
Nyomógomb	Stop	NC	Local:1:I.Data.1
Palack érzékelő	LS1	NO	Local:1:I.Data.2
Nagy palack	LS2	NO	Local:1:I.Data.3
Törött palack	LS3	NO	Local:1:I.Data.4
Töltő alaphelyzetben	LS4	NO	Local:1:I.Data.5
Nagy adag alaphelyz.	LS5	NO	Local:1:I.Data.6
Kis adag alaphelyzet	LS6	NO	Local:1:I.Data.7
Kupakoló alaphelyzet	LS7	NO	Local:1:I.Data.8
Daráló kapu nyitva	LS8	NO	Local:1:I.Data.9
Terelő kapu nyitva	LS9	NO	Local:1:I.Data.10
Doboz érzékelő	LS10	NO	Local:1:I.Data.11

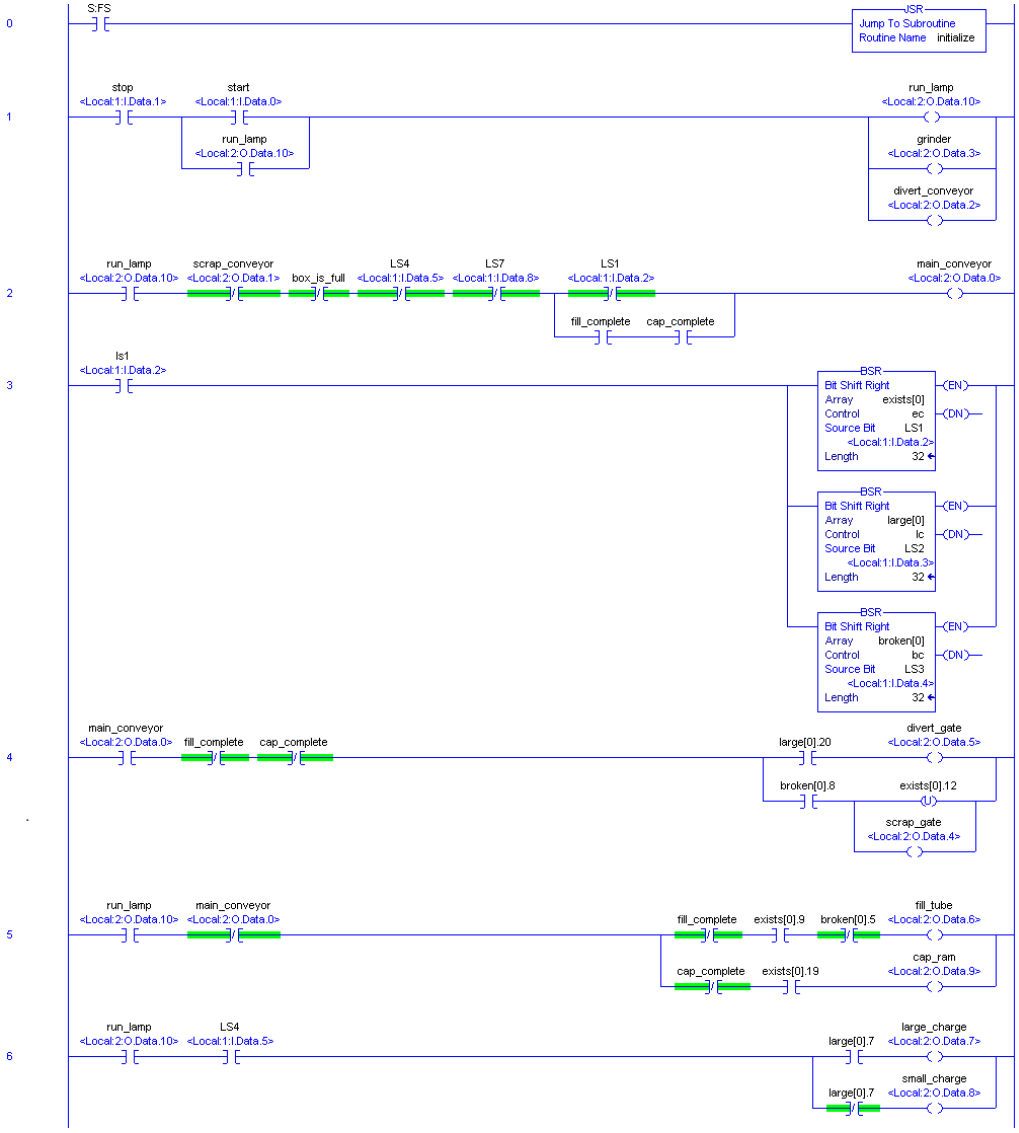
A kimenetek a 17. táblában láthatók.

17. Tábla Kimeneti változók

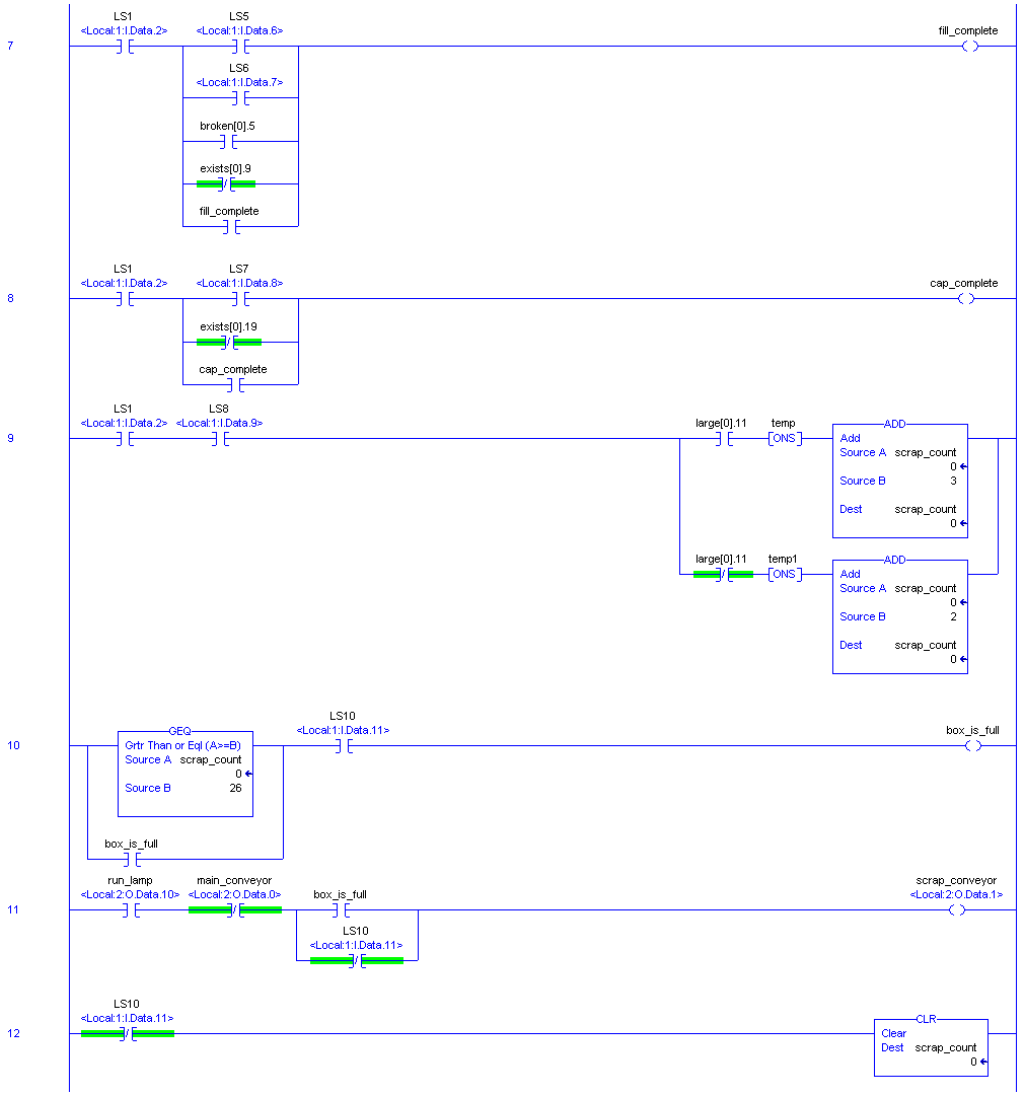
Kimeneti eszközök	Név	Azonosító
Mágneskapcsoló	Main Conveyor	Local:2:O.Data.0
Mágneskapcsoló	Scrap Conveyor	Local:2:O.Data.1
Mágneskapcsoló	Divert Conveyor	Local:2:O.Data.2
Mágneskapcsoló	Grinder	Local:2:O.Data.3
Útváltó szelep	Scrap Gate	Local:2:O.Data.4
Útváltó szelep	Divert Gate	Local:2:O.Data.5
Útváltó szelep	Fill Tube	Local:2:O.Data.6
Útváltó szelep	Large Charge	Local:2:O.Data.7
Útváltó szelep	Small Charge	Local:2:O.Data.8
Útváltó szelep	Cap ram	Local:2:O.Data.9
Jelzőlámpa	Run Lamp	Local:2:O.Data.10

A feladat a következő:

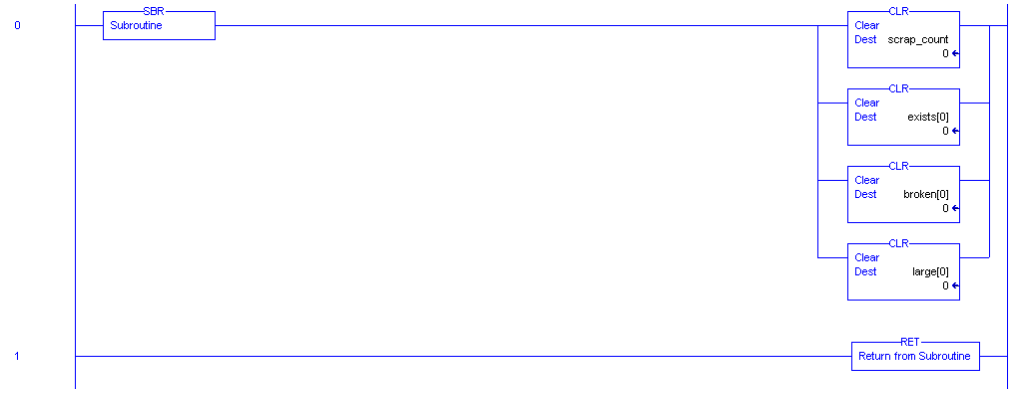
- Nagy palackokat nagy adagnyi folyadékkal töltsünk fel, kis palackokat kis adagnyival. A sérült palackokat daráljuk le. Zárjuk le a már feltöltött palackokat.
- A kis palackokra elhasznált üveg mennyisége 2/3 része a nagy palackokénak. Ügyeljünk erre, amikor a sérült palackokat daráljuk, mivel a dobozok 9 nagy palack után megtelnek.
- Az **LS1** érzékelőnél kezdjük a palackokat számozni (**0**) ez fontos lesz, amikor azt határozzuk meg, hogy melyik palackot kell tölteni, elterelni vagy lezárni.
- **Start** gombra indul a folyamat.
- **Stop** gombra leáll a folyamat.
- A **Run Lamp** jelű lámpa jelzi a gép állapotát.
- A kis palackokat a fő-szállítószalagon tartjuk, míg a nagyokat áthelyezzük a terelő szállítószalagra.



142. ábra A feladat megoldása



143. ábra A feladat megoldása



144. ábra A feladat megoldása

A 142-144. ábrák vázolják a feladat megoldását. Elemezzük a programot.

A 0. sor az **initialize** szubrutint hívja meg a program első ciklusában. Az első ciklus után nem lesz végrehajtva.

Az 1. sor egy öntartó áramkör, ami a **Start** jelű nyomógomb megnyomásának hatására működteti a **Run Lamp** jelű lámpát, a darálót és a terelő futószalagot.

A 2. sor vezérli a fő-futószalagot. A futószalag nem működhet, ha a töltőcső vagy a kupakozó karjai nincsenek belső végállásukban, ellenkező esetben palacktörés történhet. Szintén le kell állítani a fő-futószalagot, amikor a megtelt dobozt elszállítjuk, különben az üvegtörmelék a padlóra kerülne.

A 3. sor felügyeli a palackokat. 3 **BSL** utasítást alkalmazunk arra, hogy az érzékelők értékeit eltárolják. Minden újonnan megérkezett palack, amit **LS1** érzékel, eggyel lépteti a tárolókat. A **BSL**-ben egy 1 dimenziós tömböt alkalmaztunk, aminek 1 **DINT** eleme van. Több elemre nincs szükségünk, mivel a **DINT** 32 bitje 32 palackot tud számon tartani, nekünk viszont 24 is elég. Abban az esetben, ha 32-nél több palackot kell számon tartani, egy 1 dimenziós tömböt alkalmaznánk több mint 1 **DINT** elemmel.

A 4. sor a terelő kapukat vezérli, ugyanakkor törli a **BSL** regiszterből azokat a palackokat, melyeket ledaráltunk. A feltétel a **large[0].20**-hoz van kötve mivel a kapu **LS2**-től számított 21. pozíción van. Hasonlóképpen a **broken[0].8**-at akkor ellenőrizzük, amikor a darálóhoz tartozó terelőt vezéreljük, mivel az 9 pozícióra van **LS3**-tól.

Az 5. sor felel a kupakozóért és a töltőcsőért, melyeknek akkor kell aktiválódniuk, amikor palack van alattuk.

A 6. sor felel a töltésért. Amikor a töltőcső külső véghelyzetben van, nagy adaggal tölti meg a nagy palackokat és kicsivel a kis palackokat.

A 7. és 8. sorok két bitet kezelnek, melyek jelzik, hogy a töltés és kupakozás befejeződött.

A 9. sor a ledarált palackokat számolja, a változóhoz 3-at ad hozzá minden ledarált nagy palackonként és 2-t minden ledarált kis palackonként.

A 10. sor egy öntartó kapcsolás, ami azt jelzi, hogy a doboz megtelt.

A 11. sor szintén egy öntartó kapcsolás, ami a selejt futószalagért felelős.

A 12. sor kinullázza a ledarált palackok számlálóját amint egy üres doboz érkezik.

A szubrutinnak két sora van, de csak az elsőnek van fontos szerepe a program futásában, mivel ez az a sor, ami kinullázza a tömböket amint a program megkezdi futását.

Irodalom

1. W. Bolton: Programmable logic controllers 4th edition 2006. ELSEVIER
   ISBN-13: 978-0-7506-8112-4
   ISBN-10: 0-7506-8112-8
2. Dr. Hugh Jack: Automating manufacturing systems with PLCs Version 5.1 2008.
3. Allen-Bradley company: Understanding and applying micro programmable controllers