HADMÉRNÖK

V. Évfolyam 1. szám - 2010. március

Deák Ferenc
deak@nct.hu

LÉTRADIAGRAM FORDÍTÓK ELMÉLETE PLC VEZÉRLÉSEK SZÁMÁRA II.

Absztrakt

A létradiagram egyszerű, programozási képzettséggel nem rendelkező szakemberek számára is érthető nyelv, ennek köszönhető, hogy az elmúlt 20-25 évben az egyik legnépszerűbb ipari programozási formává vált. Bár létradiagramos PLC (Programmable Logic Controller) rengeteg létezik a piacon, az ezeket feldolgozó fordítóprogramok irodalma meglehetősen szegényes. Az alábbi cikk az NCT új vezérléscsaládja számára kidolgozott PLC fordító elméleti hátterét mutatja be. Ez az algoritmus szakít az irodalom által ismertetett megközelítéssel, és gráfelméleti oldalról közelíti meg a kérdést, ami egy jól áttekinthető, szemléletes többlépcsős megoldást eredményez.

Relay ladder logic has become one of the most popular discrete control programming systems in the last 20-25 years. Programmable Logic Controllers (PLCs) usually can be programmed by wiring up relay contacts and coils on screen. This virtual circuit is transformed into a list of instructions in sequence. In this paper, the translation theory of relay ladder logic for new generation NCT controllers is examined. In contrast to solutions accessible in the literature this algorithm is multiphase, expressive and based on graph theory.

Kulcsszavak: *PLC, létradiagramm, fordító ~ PLC, Programmable logic controllers, relay ladder logic, compiler*

Bevezetés

A cikksorozat első részében a létradiagrammot önállóan lefordítható részekre, létrafokokra bontottuk. Ebben a részben magát a fordítási algoritmust ismertetjük.

Színezés

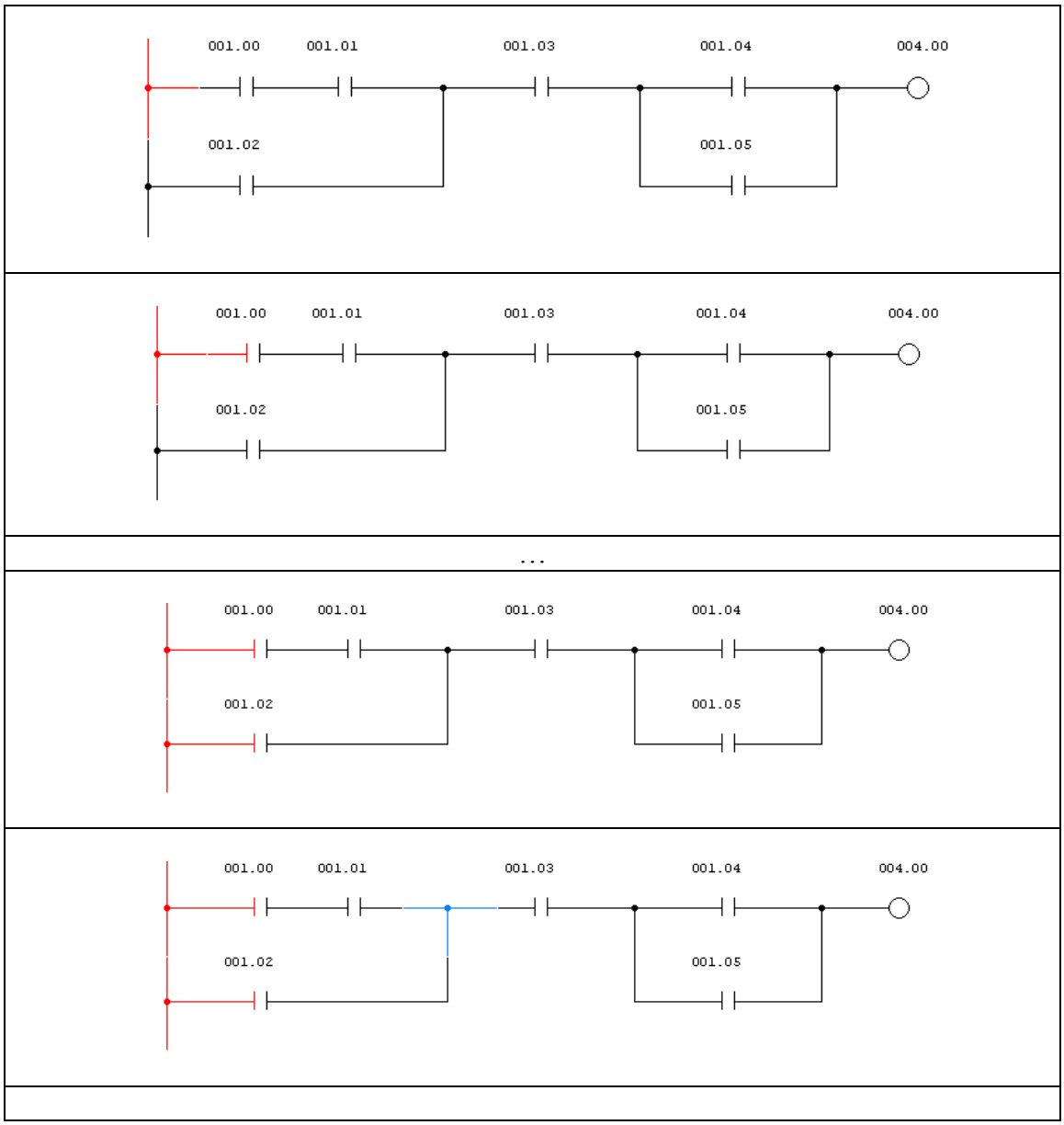
A létradiagramot egy gráffá fogjuk alakítani a következő lépésben. A gráf pontjai annak az áramkörnek a csomópontjai, amit a létradiagram szimulál, élei pedig az áramkör relei és tekercsei (kimenetei). A gráfban már nincs fent és lent, nincs jobbra és balra, és eltűnnek a logikailag felesleges csomópontok, a fordításhoz szükséges információk maradnak csak meg. A gráf leendő pontjainak felismeréséhez a létradiagramot beszínezzük. A színek és a gráf pontjai között kölcsönösen egyértelmű a megfeleltetés.

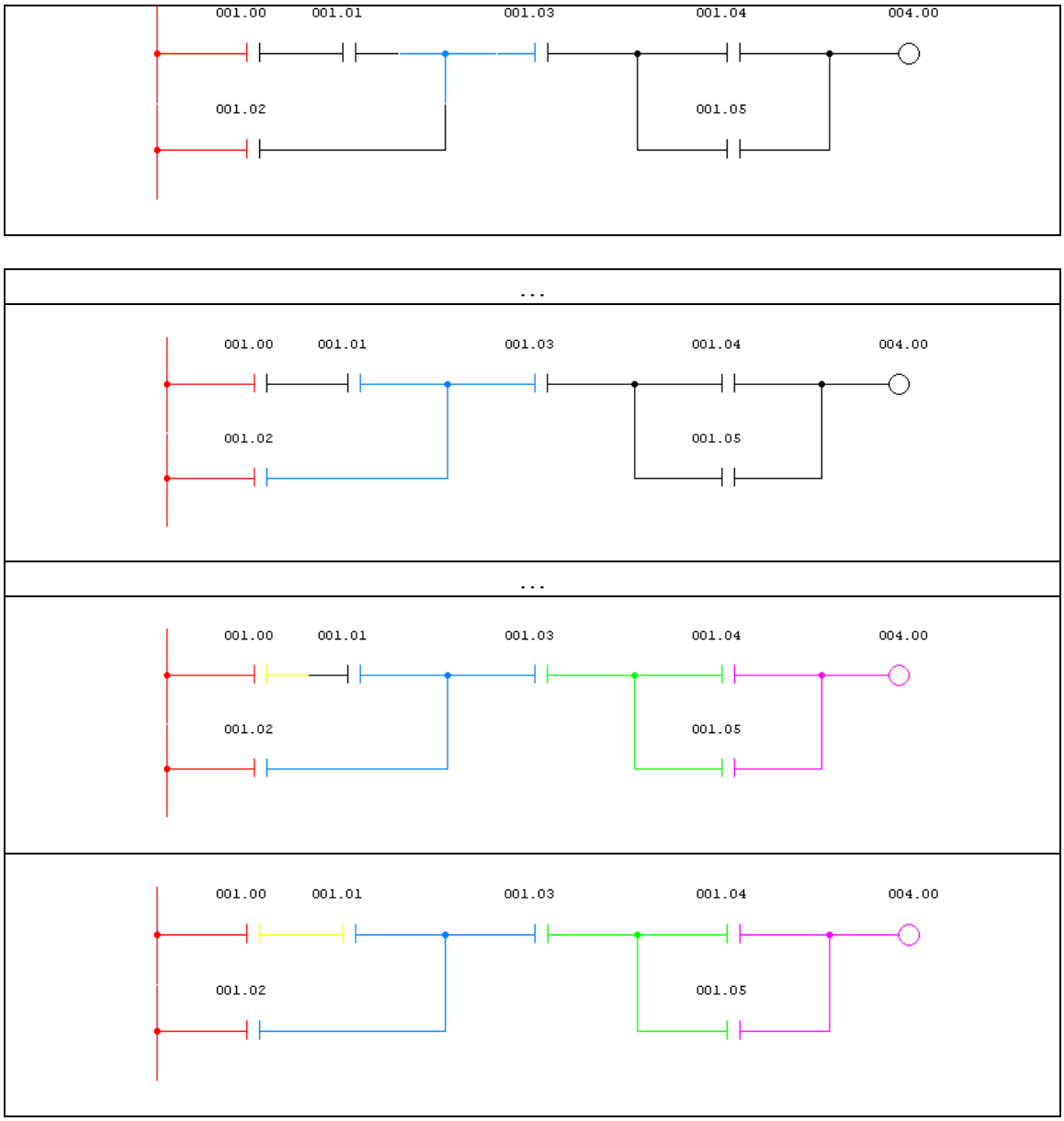
A színezésnél balról jobbra illetve fentről lefelé haladva minden vezetékdarabra elvégezzük a *Színezés* nevű eljárást rekurzív módon. A szomszédok bejárása itt is az óramutatónak megfelelő irányban mélységi kereséssel történik.

```
minden ( V vezeték darabra)
{
        Keresünk egy még nem használt C színt
        Színez( V, C )
}

Az érintkezők jobboldalára is elvégezzük a fentihez hasonló színezési eljárást

Színez ( Létradiagram-elem V, Szín C )
{
        ha ( V vezeték darab és még nincs beszínezve) akkor
        {
                Beszínezzük V-t C színre

                minden (engedélyezett S szomszédra)
                        Színezés( S, C )
        }

        ha ( V érintkező ) akkor
        {
                ha ( V baloldala még nincs beszínezve és a" színezés balról érkezik" )
                akkor
                        Beszínezzük V baloldalát t C színre

                ha ( V jobboldala még nincs beszínezve és a" színezés jobbról érkezik" )
                akkor
                        Beszínezzük V jobboldalát t C színre

        }
```

```
        ha ( V kimenet darab és még nincs beszínezve ) akkor
        {
                Beszínezzük V-t C színre
        }
}
```

1. Pszeudó kód: A színezés





1. táblázat: Színezés bemutatása egy példán

Gráffá és mnemonic kóddá alakítás

A létradiagramot reprezentáló gráf *él*ekből, és *pont*okból áll. Az *él*ek *kimenet*eket és *érintkező*ket modelleznek. Minden *él* két *pont*ot köt össze. Egyiket *baloldali pont*nak hívjuk, és az eredeti *érintkező* vagy *kimenet* baloldalának felel meg, a másikat pedig *jobboldali pont*nak nevezzük, és az eredeti érintkező jobboldalának felel meg. (A *kimenet*nek nincs jobboldala). A pontokhoz az előbbiek szerint élek csatlakoznak. Minden pont két listát tart nyilván a *kimenő élek listáj*át, és a *bemenő élek listá*ját. Minden pontban nyilvántartunk egy *utasításlistá*t, ahol a lista minden eleme

egy mnemonickódos PLC programsornak fel meg. Érintkező esetében ennek formája „LD *cím*”, kimenet esetén ennek formája „OUT *cím*”.

DEF: A gráf egy adott pontjához tartozó **E** élet a pontban *bemenő él*ként tartjuk nyilván, ha az **E** él egy *érintkező,* és a pont a **E** jobboldalához csatlakozik.

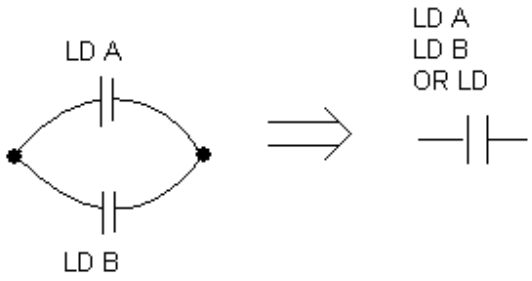
DEF: A gráf egy adott pontjához tartozó élet a pontban *kimenő él*ként tartjuk nyilván, ha
a) az **E** él egy *érintkező*, és a pont **E** baloldalához csatlakozik, vagy
b) **E** él egy kimenet

DEF: $\mathbf{E_1}$ és $\mathbf{E_2}$ élek párhuzamos kapcsolatban vannak, ha
a) $\mathbf{E_1} \neq \mathbf{E_2}$, és
b) $\mathbf{E_1}$ és $\mathbf{E_2}$ jobboldalán lévő pont közös, és
c) $\mathbf{E_1}$ és $\mathbf{E_2}$ baloldalán lévő pont szintén közös

DEF: $\mathbf{E_1}$ és $\mathbf{E_2}$ élek soros kapcsolatban vannak ($\mathbf{E_1}$ jobboldali szomszédja $\mathbf{E_2}$), ha
a) $\mathbf{E_1} \neq \mathbf{E_2}$, és
b) van olyan **P** pont, amelyre igaz, hogy
- **P** pont $\mathbf{E_1}$ él jobboldalán helyezkedik el, és
- **P** pont $\mathbf{E_2}$ él baloldalán helyezkedik el, és
- **P** pontnak csak egy bemenő pontja van (ez $\mathbf{E_1}$), és
- **P** pontnak csak egy kimenő pontja van (ez $\mathbf{E_2}$)

```
vonjuk_össze_párhuzamosan ( E1 élet és E2 élet )
{
        E1 parancslistájához hozzáfűzzük E2 parancslistáját
        Az így létrejövő lista végéhez hozzáírunk egy ” OR LD” sort
        Töröljük E1-et a gráfból
}
```

2. pszeudó-kód: Párhuzamos összevonás



1. ábra: Párhuzamos összevonás

```
Keressünk_párhuzamos_éleket (Él E1 )
{
        Legyen E1 baloldali pontja B

        minden (E2 élre, mely B kimeneti listájában szerepel)
        {
                ha (E1 és E2 párhuzamos kapcsolatban van) akkor
                        vonjuk_össze_párhuzamosan ( E1 élet és E2 élet )
        }
}
```

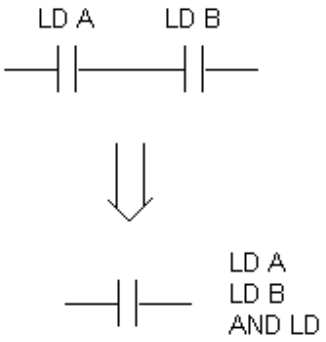
3. pszeudó kód: Párhuzamos élek keresése

```
vonjuk_össze_sorosan ( E1 élet és E2 élet )
{
        ha (E2 kimenet ) akkor
                nem csinálunk semmit
        különben
        {
                E1 jobboldali pontját, ami egyúttal E2 baloldali pontja, nevezzük P1-nek
                E2 jobboldali pontját nevezzük P2-nek

                E1 parancslistájához hozzáfűzzük E2 parancslistáját
                Az így létrejövő lista végéhez hozzáírunk egy ” AND LD” sort

                P2 összes kimenő élét átkötjük P1 kimenő élei közé
                Töröljük E2-őt és P2-őt
        }
}
```

4. pszeudó kód: Soros összevonás

Egy él átkötéséhez

- törölni kell az élt a baloldali pontjának kimenő élei közül
- hozzá kell adni az új baloldali pont kimenő éleihez
- az új baloldali pontot be kell jegyezni a pontban baloldali pontként
- törölni kell az élt a jobboldali pontjának bemenő élei közül
- hozzá kell adni az új jobboldali pont bemenő éleihez
- az új jobboldali pontot be kell jegyezni a pontban jobboldali pontként



2. ábra: Soros összevonás

```
Keressünk_soros_éleket ( Él E1 )
{
        ha (E1-nek létezik jobboldali szomszédja, és soros kapcsolatban vannak) akkor
        {
                ezt a szomszédot nevezzük E2-nek
                vonjuk_össze_sorosan(E1-et  és E2-őt )
        }

        ha ( történt összevonás ) akkor
                megismételjük az eljárást
}
```

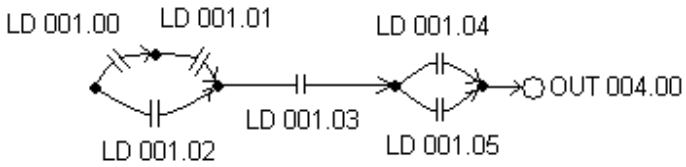
5. pszeudó kód: Soros élek keresése

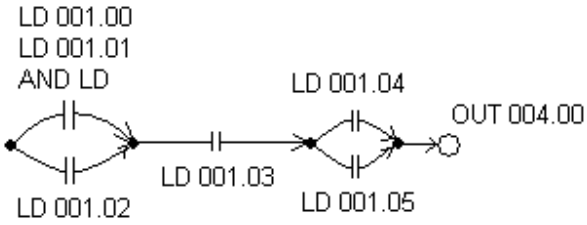
```
minden (E1 élre )
{
        Keressünk_soros_éleket(E1)
        Keressünk_párhuzamos_éleket(E1)
}

ha ( volt összevonás vagy a gráf éleinek száma > 1 ) akkor
        megismételjük az eljárást
```

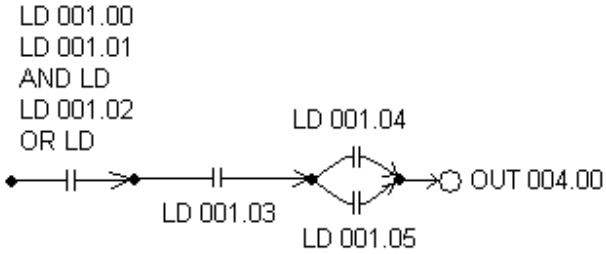
6. pszeudó kód: Gráf reprezentáció átalakítása pszeudó kóddá

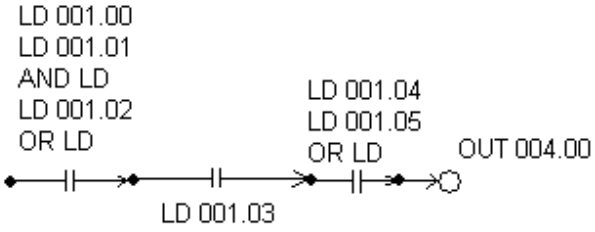
A gráf minden élén végigmegyünk, és keresünk vele soros vagy párhuzamos kapcsolatban levő éleket. Ha találunk ilyet, akkor összevonjuk őket. Az eljárást addig ismételjük, amíg sikerül éleket összevonni, illetve a gráf éleinek száma nagyobb mint egy. Végül két él marad, egy tartalmazza az összes érintkezőre vonatkozó mnemonickódot, egy további él pedig a kimenetet. Utolsó lépésként ezt a két élt vonjuk össze a két él utasításlistájának összefűzésével. Az összefűzött lista lesz a létrafok mnemonickódja. A teljes PLC program mnemonickódját, úgy

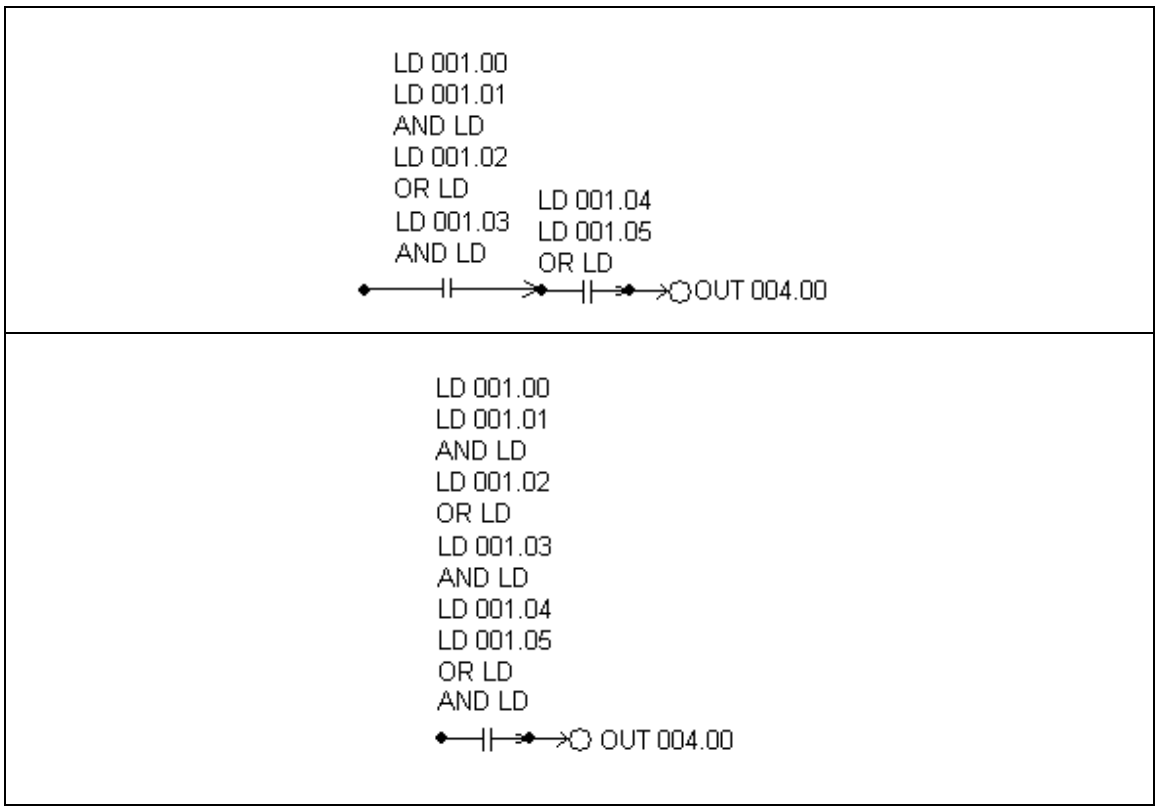
kapjuk, hogy a létrafokokra kapott utasításlistákat egymás a létrafokok sorrendjében egymás után fűzzük.











2. táblázat: Mnemonic kóddá alakítás szemléltetése egy példán

Az 2. táblázatban bemutatott átalakítások utolsó lépése után a 2. mnemonic kód mezőben látható eredmény keletkezik.

```
LD 001.00
LD 001.01
AND LD
LD 001.02
OR LD
LD 001.03
AND LD
LD 001.04
LD 001.05
OR LD
AND LD
OUT 004.00
```

1. mnemonic kód: Az összevonások eredménye fésülés előtt

A következő lépés a kód fésülése. Ennek során a következő a 3. táblázatban látható cseréket hajtjuk végre.

Mit cserélünk	Mire
LD *cím* AND LD	AND *cím*
LD *cím* OR LD	OR *cím*

3. táblázat: A fésülésnél történő cserék

A fésülés műveletét addig ismételjük, amíg történik legalább egy csere. A példában szereplő kódunk a fésülést követően a 3. mnemonic kód mezőben látható.

```
LD 001.00
AND 001.01
OR 001.02
AND 001.03
LD 001.04
OR 001.05
AND LD
OUT 004.00
```

2. mnemonic kód: Fésülés után

Összefoglalás

A cikksorozat második része magát a fordítási algoritmust ismertette. A következő egyben befejező részben néhány olyan speciális esetre térünk ki, melyet most nem tárgyaltunk. A speciális esetek jelentik az algoritmus igazi értékét, mivel ezeknek a segítségével olyan programozási technikák is alkalmazhatóak, melyek a kereskedelemben kapható PLC-esetében nem.

Felhasznált irodalom

[1] IEC 61131-3 Programmable controllers - Part 3: Programming languages International Standard, International Electrotechnical Commission, 2003

[2] John T. Welch: Translating unrestricted relay ladder logic into Boolean form. Computers in Industry, 20 (1992) 45-61

[3] NCT szerszámgép vezérlések PLC programozási leírása
http://www.nct.hu/pdf/NC_Documents/Magyar/Telepites/magplc.pdf

[4] H. A. Barker, J. Song, P. Townsend: A rule based procedure for generating programmable logic controller code from graphical input in the form of ladder diagrams, Eng. Appli. of AI, 1989, Vol. 2, December

[5] R. Wareham, Ladder diagram and sequential function chart languages in programmable controllers, Can. Mach. Metalwork., December 1988, pp. 25-26.

[6] R. Devanathan, Computer aided design of relay ladder diagram from functional specification" Int. Conf. on Industrial Electronics, Control and Instrumentation – IECON, 1990, pp. 527-531.

[7] D. Harel, Statecharts: a visual formalism for complex systems, Sci. Comput. Programming Vol. 8, 1987, pp. 231-274.

[8] M. Courvoisier, R. Valette, J. Bigou and P. Esteban, A programmable controller based on a high level specification tool, IECON 1983, pp. 174-179.

[9] T. Murata, N. Komoda, K. Matsumaoto and K. Haruna, A Petri-net based controller for flexible and maintainable sequence control and its applications in factory automation, IEEE Trans. Ind. Electron, Vol. IE-33, No. 1, February 1986, pp. 1-8.

[10] IEC PAS 62407 Real-time Ethernet control automation technology (ETHERCATTM), International Electrotechnical Commission, 2005