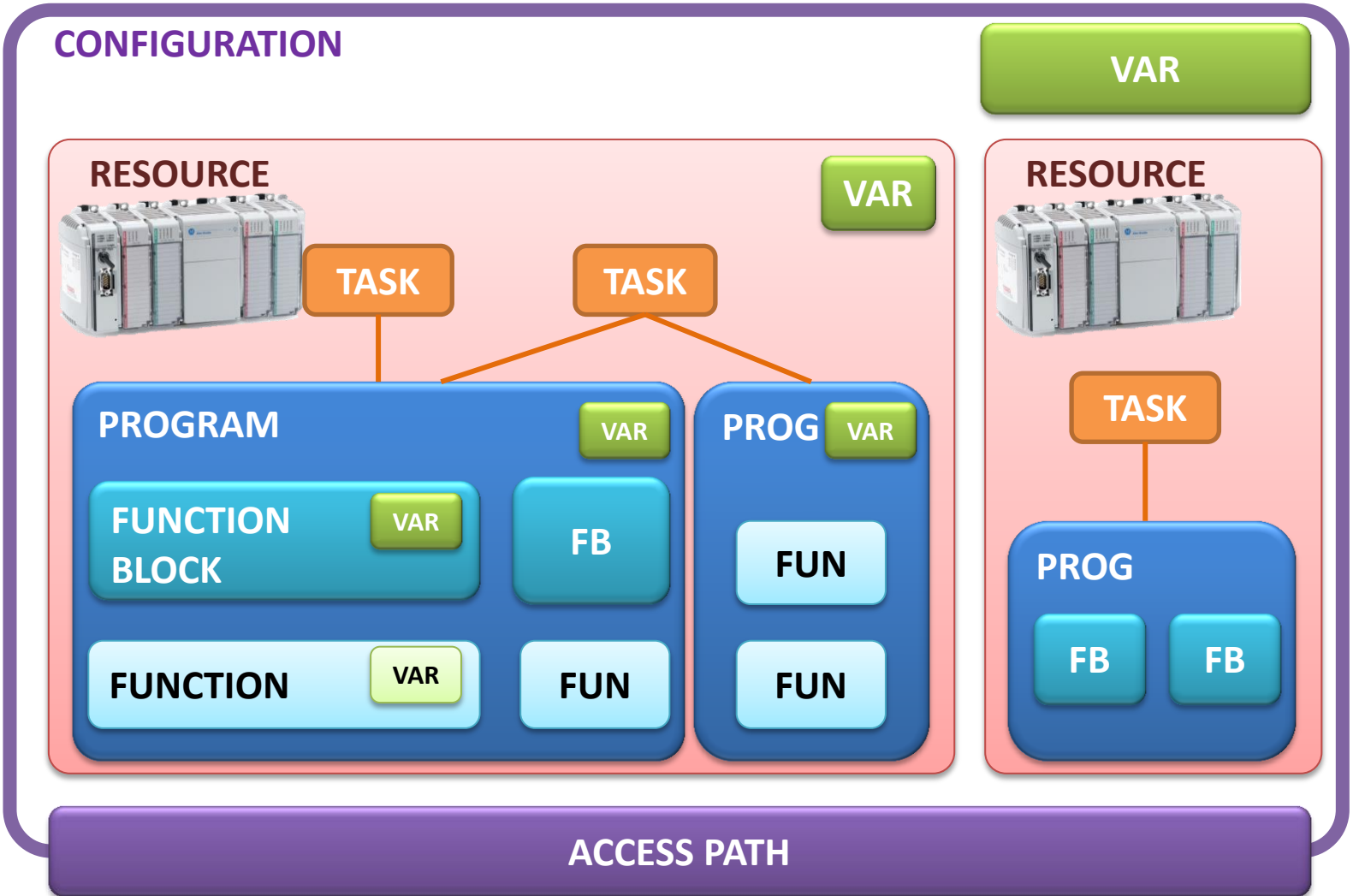


# Az IEC 61131-3 szabvány programozási nyelvei

Folyamatszabályozás

KOVÁCS Gábor  
gkovacs@iit.bme.hu

# Áttekintés



# Programszervezési egységek

## POU típus és név

### Deklarációs rész:

- Interfész változók
- Lokális változók
- Globális változók

### POU törzs: programkód

- Ladder Diagram (LD)
- Instruction List (IL)
- Function Block Diagram (FBD)
- Structured Text (ST)
- Sequential Function Chart (SFC)

```
PROGRAM prog_name
    PROGRAM ConveyorControl
FUNCTION_BLOCK fb_name
    FUNCTION_BLOCK Pusher
FUNCTION fun_name : DataType
    FUNCTION IsReady : BOOL
```

# Létradiagram

- Eredete: huzalozott relés logika
  - kapcsolók  $\approx$  kontaktusok
  - relék  $\approx$  tekercsek
- A mai napig az egyik legnépszerűbb nyelv
- Minden fejlesztői környezetben megtalálható

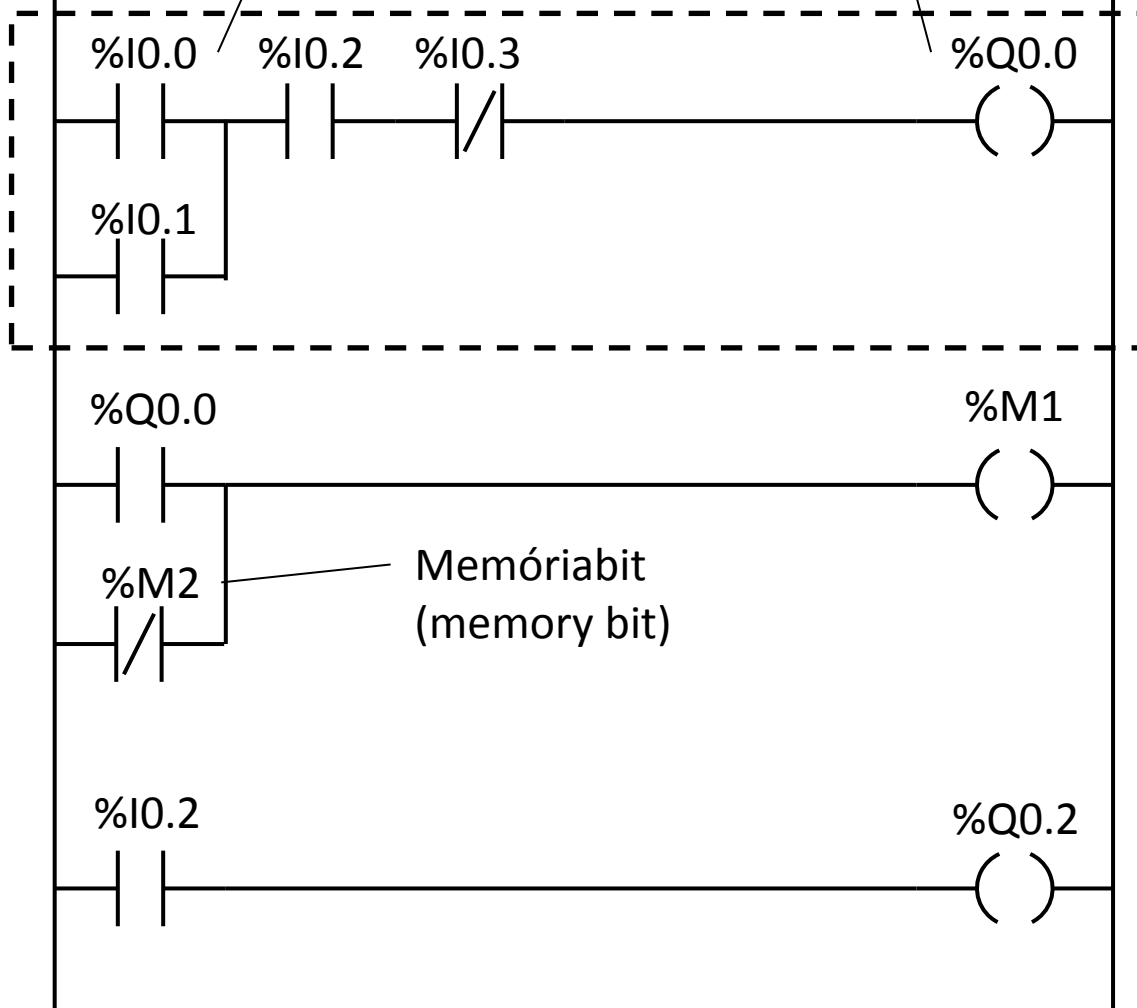
Pozitív sín  
(positive rail)

Föld sín  
(neutral rail)

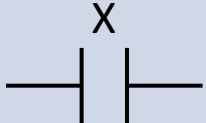
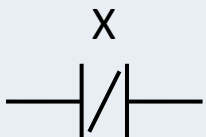
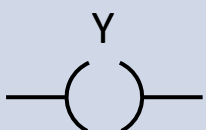
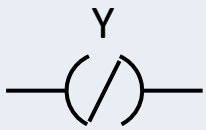
Bemeneti bit  
(input bit)

Kimeneti bit  
(output bit)

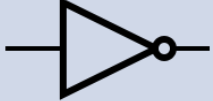
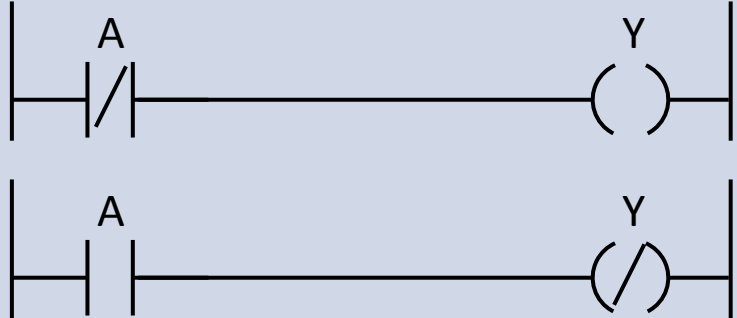




Létraszor  
(rung)




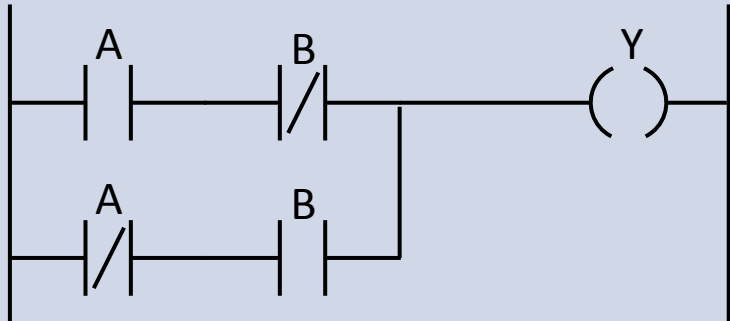

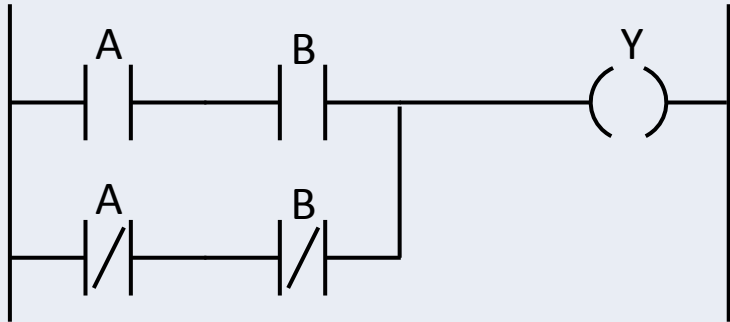
# Kontaktusok és tekercsek

Szimbólum	Megnevezés	Működés	Analógia
	Alaphelyzetben nyitott kontaktus [kontaktus] ( <i>NO contact, contact</i> )	„vezet” ha $X=1$	Alaphelyzetben nyitott nyomógomb
	Alaphelyzetben zárt kontaktus [negált kontaktus] ( <i>NC contact</i> )	„vezet” ha $X=0$	Alaphelyzetben zárt nyomógomb
	Alaphelyzetben nyitott tekercs [tekercs] ( <i>NO coil, coil</i> )	Y-t 1-be állítja ha „táplált”	Alaphelyzetben nyitott relé
	Alaphelyzetben zárt tekercs [negált tekercs] ( <i>NC coil</i> )	Y-t 0-ba állítja ha „táplált”	Alaphelyzetben zárt relé

# Logikai műveletek

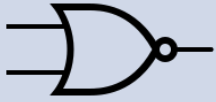
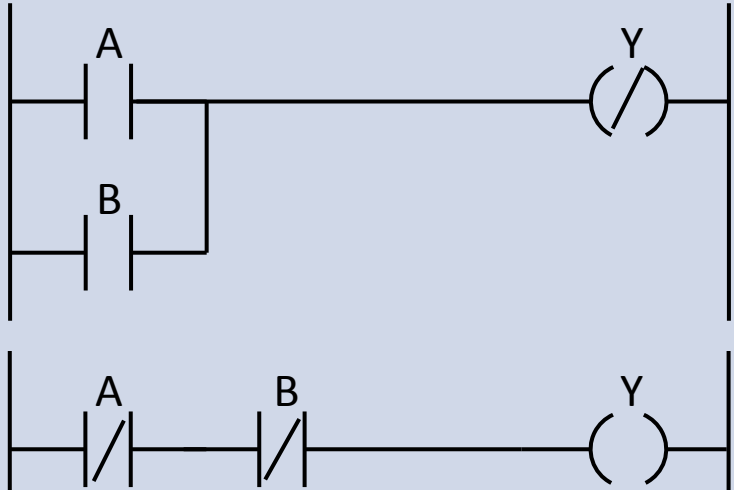

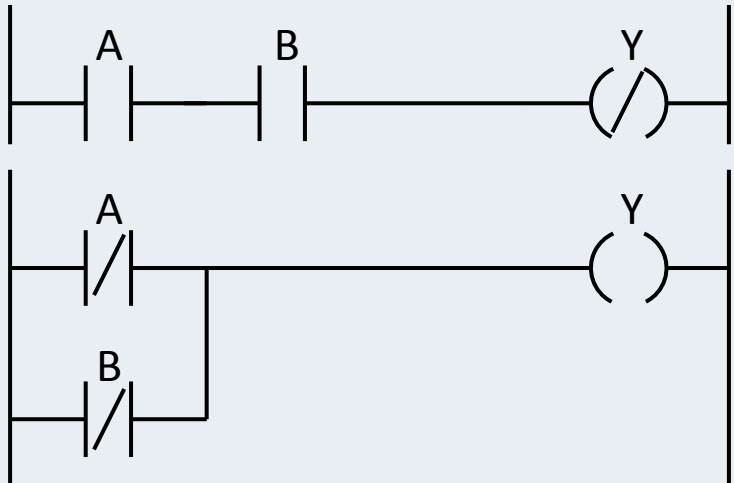
Művelet	Szimbólum	Algebrai jelölés	Létradiagrammos megvalósítás
Negálás		$Y = \bar{A}$	
ÉS		$Y = A \cdot B$ $Y = A \& B$	
VAGY		$Y = A + B$	

# Logikai műveletek

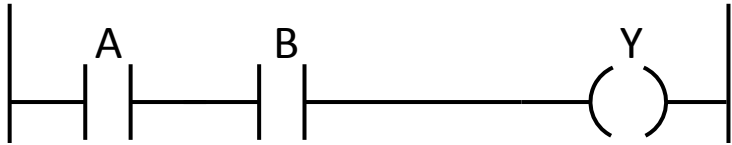
Művelet	Szimbólum	Algebrai jelölés	Létradiagramos implementáció
Antivalencia (XOR)		$Y = A \oplus B$	
Ekvivalencia (NXOR, EOR)		$Y = \overline{A \oplus B}$ $Y = A \odot B$	



# Logikai műveletek

Művelet	Szimbólum	Algebrai jelölés	Létradiagramos megvalósítás
NOR		$Y = \overline{A + B}$ $Y = \bar{A} \cdot \bar{B}$	
NAND		$Y = \overline{A \cdot B}$ $Y = \bar{A} + \bar{B}$	

# Létrásor = Logikai függvény



```
IF (A=1) AND (B=1)
THEN Y=1
```

✘

```
IF (A=1) AND (B=1)
THEN Y=1
ELSE Y=0
```

✔

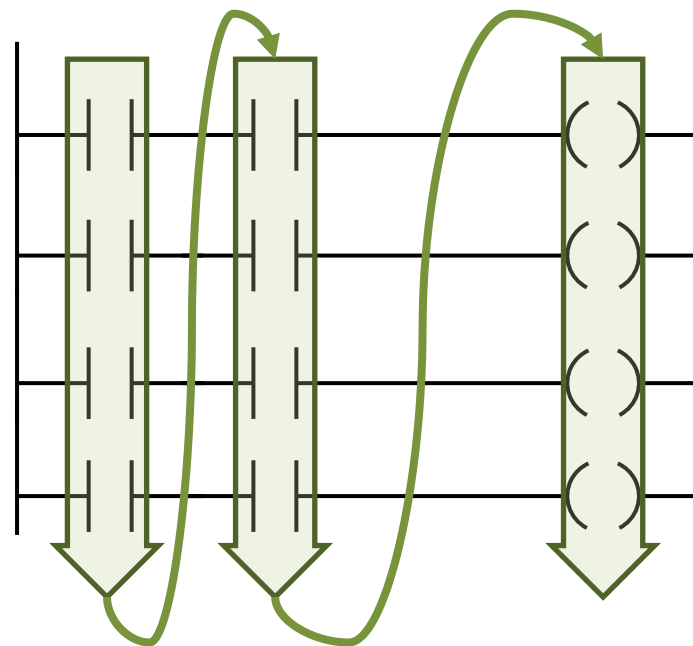
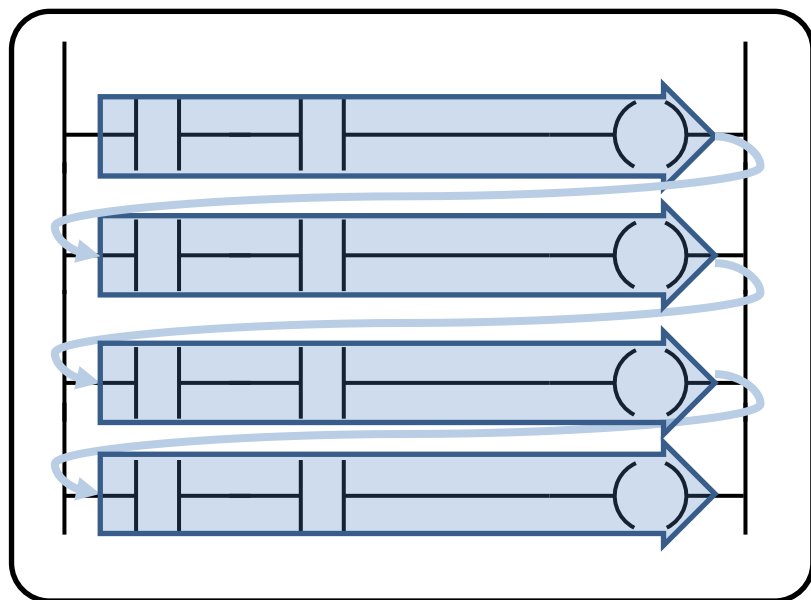
# A létradiagram kiértékelése

- A PLC ciklikus működésű
- A programvégrehajtás fázisában a teljes kód feldolgozásra kerül
- Minden egyes ciklusban a teljes létradiagram kiértékelésre kerül

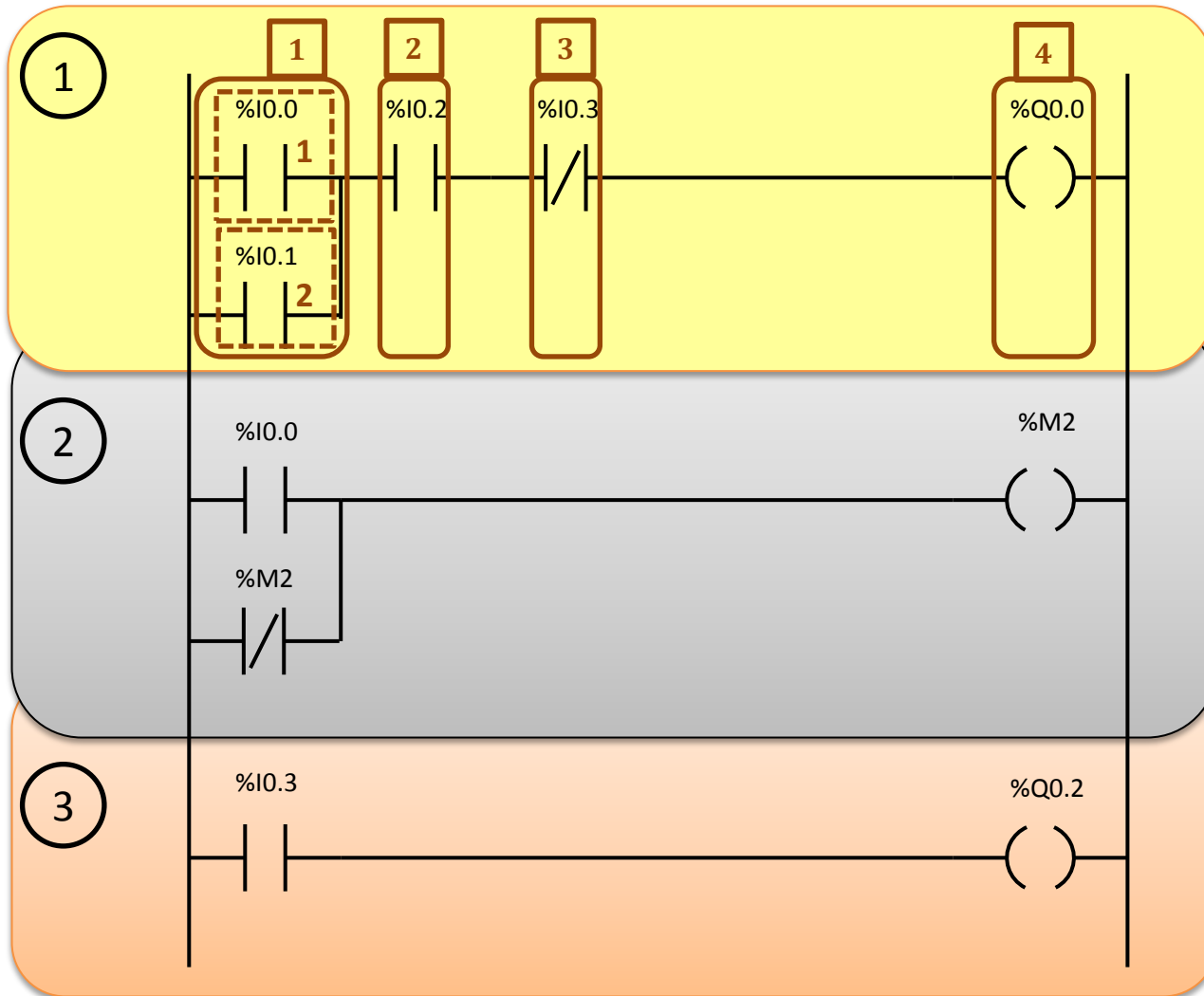


# A létradiagram kiértékelése

- Néhány nanoPLC-típus: létrasorok párhuzamos kiértékelése (ritka)
- Általános: **soros végrehajtás**
  - Soronként
  - Oszloponként (nagyon ritka)





# A létradiagram kiértékelése





# Reteszelés

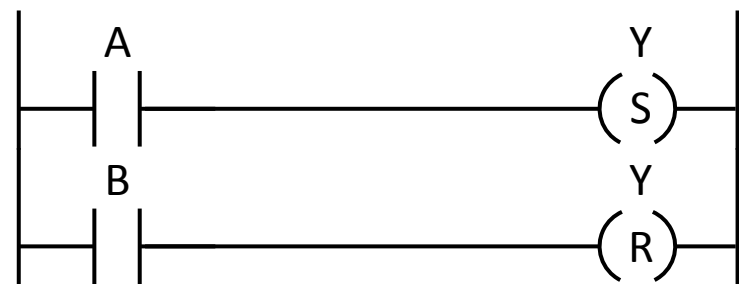
Szabványos jelölés

Symbol	Name
	Set tekercs
	Reset tekercs

RSLogix jelölés

Symbol	Name
	Output Latch (OTL)
	Output Unlatch (OTU)

# A reteszelés értelmezése



**IF (A=1) THEN Y=1**

**IF (B=1) THEN Y=0**

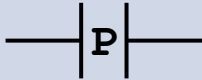
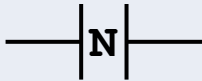


**IF (B=1) THEN Y=0**

**ELSE**

**IF (A=1) THEN Y=1**

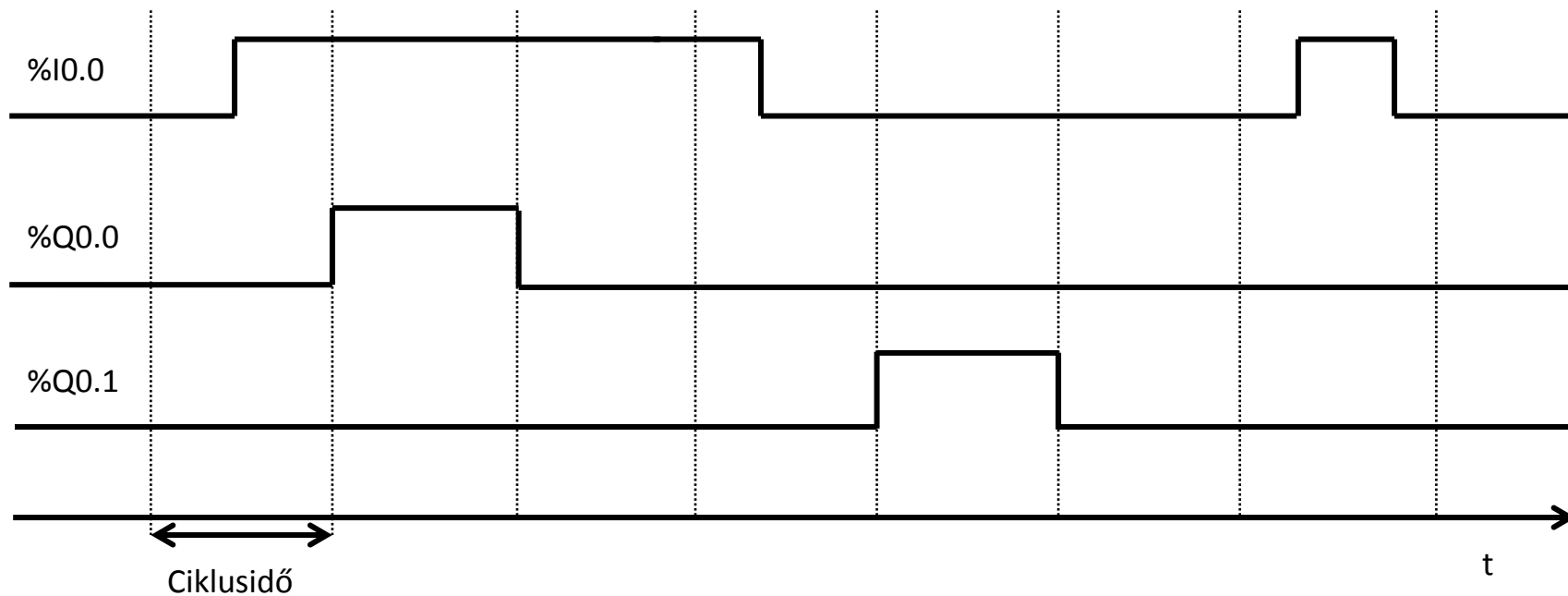
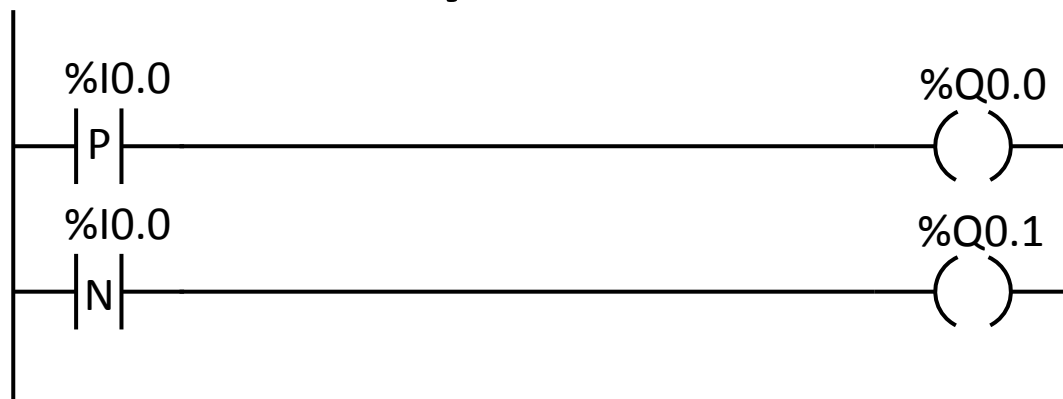
# Élérzékeny kontaktusok

Szimbólum	Megnevezés
	Felfutó él-érzékeny kontaktus <i>(rising edge contact , one shot rising)</i>
	Lefutó él-érzékeny kontaktus <i>(falling edge contact, One Shot Falling)</i>

Megjegyzés: egyes fejlesztőkörnyezetekben az éldekektálás csak az R\_TRIG és F\_TRIG funkcióblokkok használatával lehetséges.

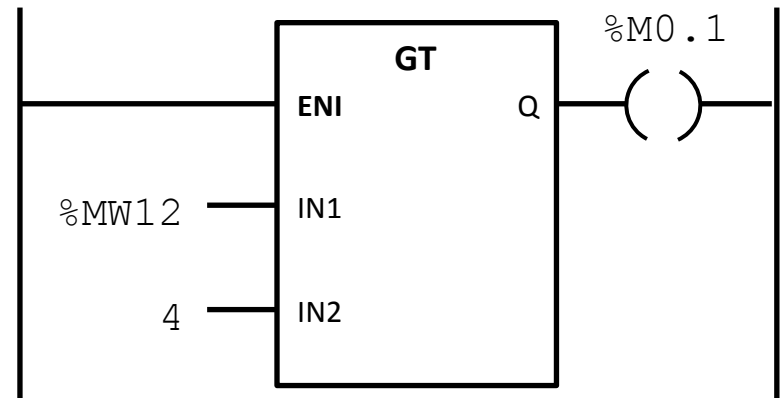
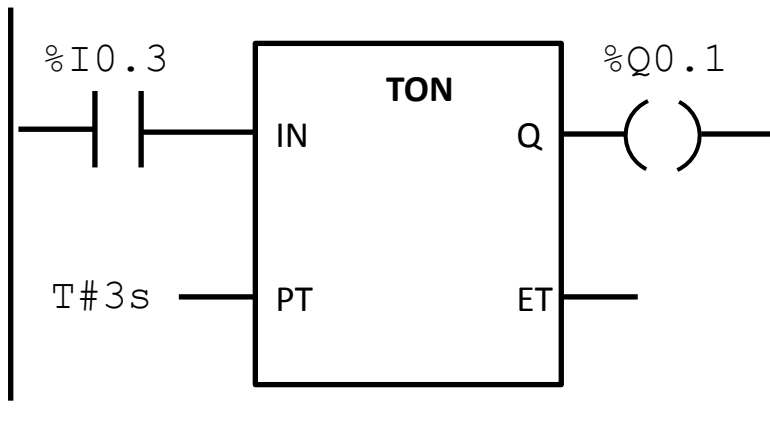


# Érzékeny kontaktusok



# Függvény- és funkcióblokk-hívások

- Hívás grafikusán, a létratorban megjelenő blokk formájában
  - Csak formális hívás
  - A létratorokon csak logikai értékek lehetnek: ENI/ENO használata hiányzó logikai be/kimeneti paraméterek esetén



# Utasításlista (Instruction list, IL)

- Szöveges programozási nyelv
- Alacsony szintű, gépi kódhoz közeli
  - Alapszintű műveletek
  - A programvezérlési lehetőségek korlátozottak
  - A program futása teljes egészében kézben tartható
- Bármely más IEC 61131-3 kompatibilis nyelven írt program leírható IL-lel



**A teljes szöveges kód végrehajtódik ciklusonként!**

# Utasítások felépítése

**Operátor****Módosító** **Operandus**

**LDN** **%I0.1**

**S** **MyBool**

**NOT**

- **Operátor (operator)**
  - Az operátor mnemonikja
- **Módosító (modifier)**
  - **N**: operandus negálása
  - **C**: feltételes végrehajtás
  - **(**: egymásba ágyazás
  - A módosítók használata az operátortól függ
- **Operandus (operand)**
  - Egy operandus vagy egy sem
  - Literális vagy változó (bemenet, kimenet, memória, közvetlen)
  - Változó adattípusú

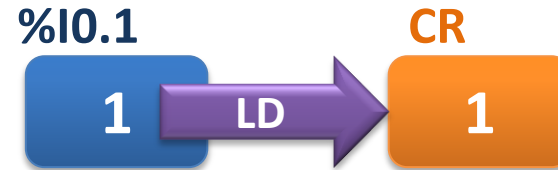
# Az akkumulátor

CR – Current Result

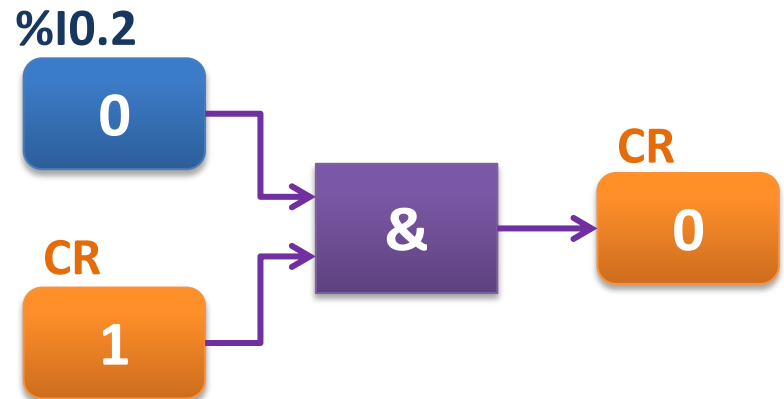
- Általános regiszter
- A műveletek első operandusa mindig az akkumulátor (kivétel: érték betöltése)
- A művelet eredménye az akkumulátorba kerül
- Az akkumulátor adattípusa a művelettől és az operandusok adattípusától függ

# Az akkumulátor

**LD %I0.1**



**AND %I0.2**



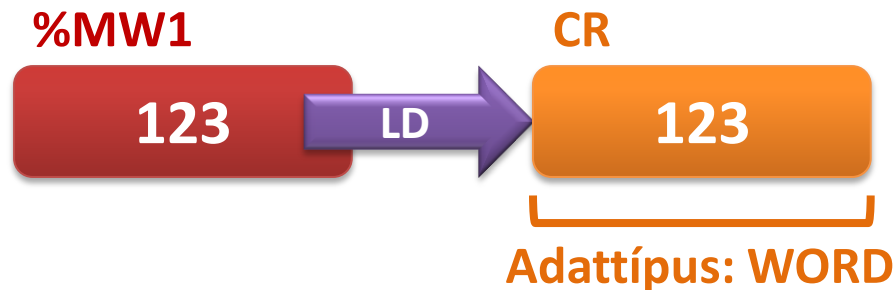
**ST %Q0.1**



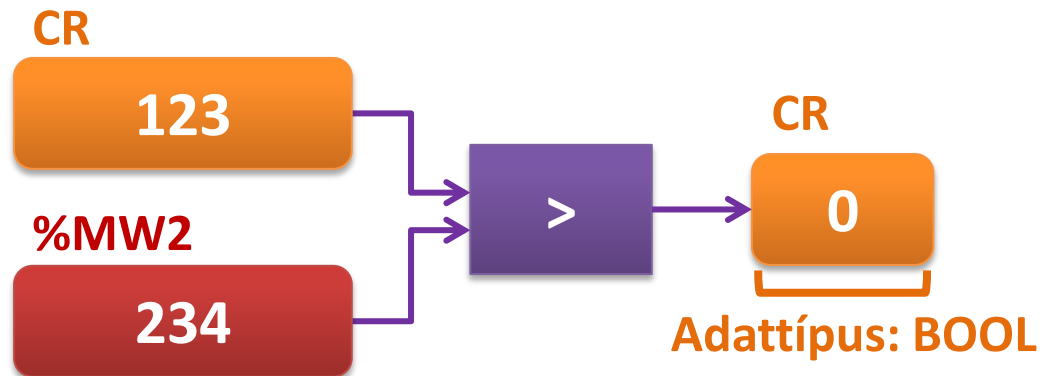
# Az akkumulátor

Az akkumulátor adattípusa a **művelettől** és az **operandustól** függően változik

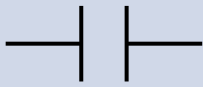



**LD** %MW1



**GT** %MW2



# Bitműveletek

Utasítás	Leírás	Létra-szimbólum	Példa
<b>LD</b>	Bit betöltése az akkumulátorba		LD %I0.1
<b>LDN</b>	Bit negálása és betöltése az akkumulátorba		LDN %M11
<b>ST</b>	Akkumulátor-érték tárolása egy biten		ST %M21
<b>STN</b>	Akkumulátor-érték negáltjának tárolása egy biten		STN %Q0.2



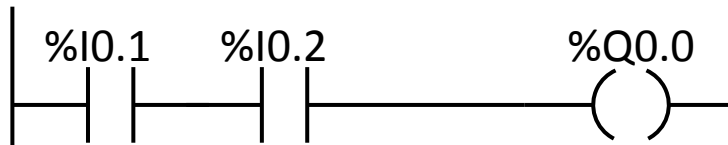
# Retentív hozzárendelések

Utasítás	Leírás	Létra-szimbólum	Példa
<b>S</b>	Bit 1-be állítása ha az akkumulátor-érték 1	$\text{---}(S)\text{---}$	S %Q0.1
<b>R</b>	Bit 0-ba állítása ha az akkumulátor-érték 1	$\text{---}(R)\text{---}$	R %M12

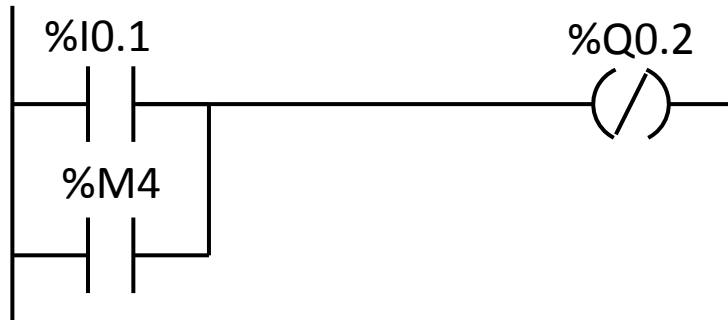
# Bit-logikai műveletek

Utasítás	Leírás	Példa
<b>AND / ANDN</b>	AND / NAND művelet az akkumulátor-értéken és az operanduson, az eredmény az akkumulátorba kerül	AND %I0.1
<b>OR / ORN</b>	OR / NOR művelet az akkumulátor-értéken és az operanduson, az eredmény az akkumulátorba kerül	ORN %M11
<b>XOR / XORN</b>	XOR / XORN (ekvivalencia) művelet az akkumulátor-értéken és az operanduson, az eredmény az akkumulátorba kerül	XOR %M21
<b>NOT</b>	Az akkumulátor-érték negálása	NOT

# Bit-logikai műveletek



```
LD %I0.1  
AND %I0.2  
ST %Q0.0
```



```
LD %I0.1  
OR %IM4  
STN %Q0.2
```

# Aritmetikai műveletek

Utasítás	Leírás	Példa
<b>ADD</b>	Operandus hozzáadása az akkumulátor-értékhez	ADD %MW2
<b>SUB</b>	Operandus kivonása az akkumulátor-értékből	SUB %MW11
<b>MUL</b>	Akkumulátor-érték szorzása	MUL 3
<b>DIV</b>	Akkumulátor-érték elosztása az operandussal	DIV 2
<b>MOD</b>	Maradékképzés (modulo) az akkumulátor-értéken az operandussal	MOD 7

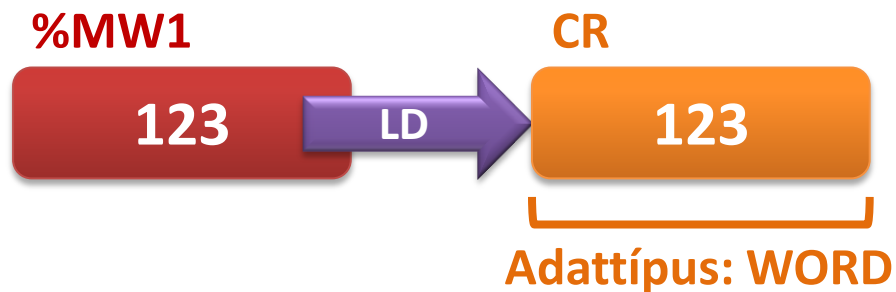
# Összehasonlító operátorok

- Az akkumulátor és az operandus értékét hasonlítják össze: CR ?? OP
- Az eredmény egy logikai érték, ami az akkumulátorba kerül

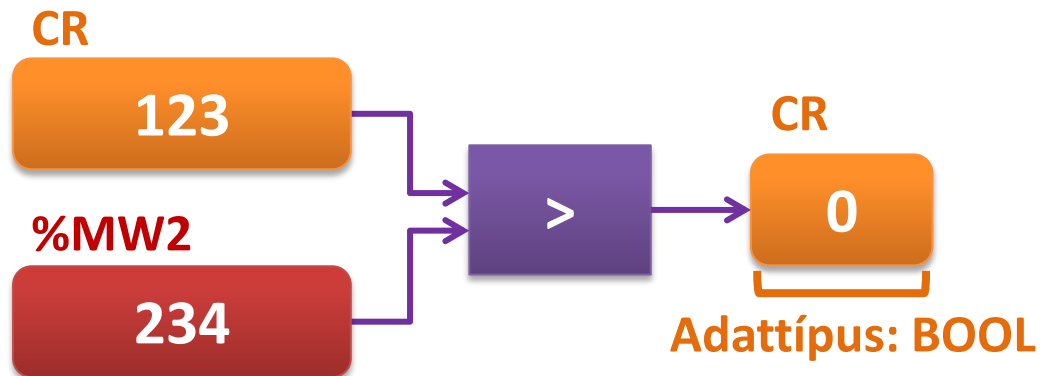
Mnemonik	Eredmény
<b>EQ</b>	Operátor és akkumulátor-érték egyenlő (=)
<b>NE</b>	Az operátor és az akkumulátor-érték nem egyezik meg (<>)
<b>GT</b>	Az akkumulátor-érték nagyobb mint az operandus
<b>GE</b>	Az akkumulátor-érték nagyobb vagy egyenlő mint az operandus
<b>LT</b>	Az akkumulátor-érték kisebb mint az operandus
<b>LE</b>	Az akkumulátor-érték kisebb vagy egyenlő mint az operandus

# Összehasonlítás - Példa


**LD** %MW1

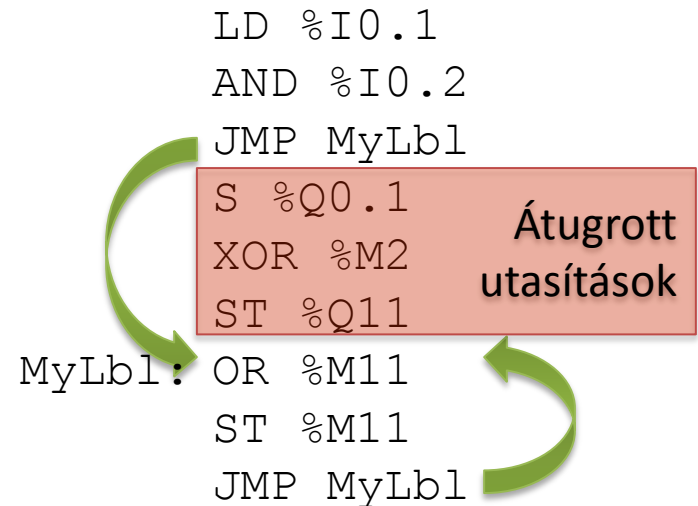


**GT** %MW2



# Ugró és hívó utasítások

- Minden utasítás (sor) címkézhető
- A program tetszőleges címkére ugorhat
  - Előre ugrás
  - Hátra ugrás – veszélyes! 
- Az ugrás során az akkumulátor-érték nem változik



Utasítás	Leírás
<b>JMP</b>	Címkére ugrás
<b>CAL</b>	FB-példány hívása
<b>RET</b>	Visszatérés a hívó POU-ba

# Módosítók

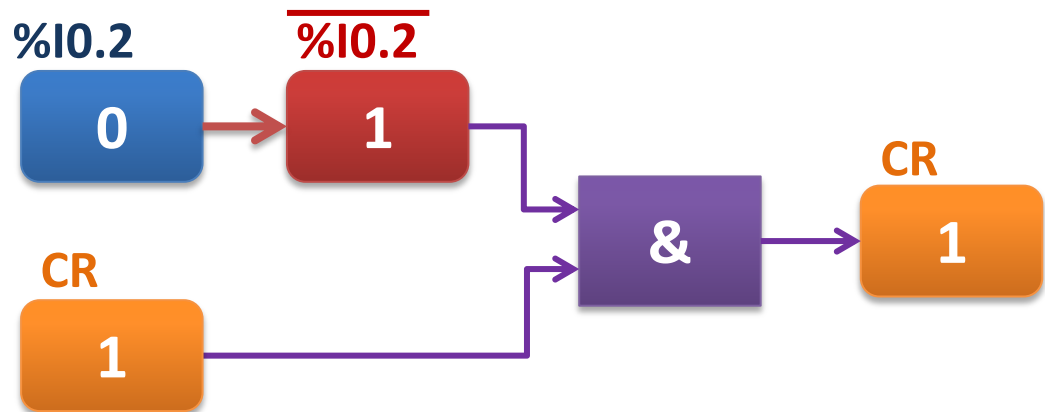
- Egyes utasítások működése módosító-postfixekkel befolyásolható
  - Feltételes végrehajtás – C
  - Operandus negálása – N
  - Egymásba ágyazás – (
- A módosítók kombinálhatók



# Operandus negálása

- Az operandust a művelet elvégzése előtt negáljuk
- Használható a következő utasításokkal
  - Betöltés és tárolás: LDN, STN
  - Bit-logikai műveletek: ANDN, ORN, XORN

```
LD 1  
ANDN %I0.2
```



# Feltételes végrehajtás

- Az utasítás csak akkor hajtódik végre, ha az akkumulátor értéke TRUE
- Feltételes ugrások, FB-hívások és visszatérés
- A következő utasításokkal használható:
  - JMP: JMPC, JMPCN
  - CAL: CALC, CALCN
  - RET: RETC, RETCN

# Feltételes végrehajtás - példa

- Elvárt működés:

IF (%I0.0 AND NOT %I0.1)

THEN %MW1:=%MW1+1

ELSE %MW1:=%MW2;

```
LD %I0.0
```

```
ANDN %I0.1
```

```
JMPC L_THEN
```

```
LD %MW2
```

```
ST %MW1
```

```
JMP L_END
```

```
(* vagy RET *)
```

```
L_THEN:
```

```
LD %MW1
```

```
ADD 1
```

```
ST %MW1
```

```
L_END:
```

```
(* további műveletek *)
```

# Feltételes végrehajtás - példa

- Elvárt működés:

IF (%I0.0 AND NOT %I0.1)

THEN %MW1:=%MW1+1

ELSE %MW1:=%MW2;

```
LD %I0.0
ANDN %I0.1
JMPCN L_ELSE
LD %MW1
ADD 1
ST %MW1
JMP L_END
L_ELSE:
LD %MW2
ST %MW1
L_END:
(* további műveletek *)
```

# Művelet-precedencia

- $Y = A \& (B + C)$

```
LD A
AND B   Y = A&B + C
OR C
ST Y
```

```
LD B
OR C
AND A
ST Y
```

# Művelet-precedencia

- $Y = (A + B) \& (C + D)$

```
LD A
OR B
AND C
OR D
ST Y    Y = (A + B) & C + D
```

```
LD A
OR B
ST X
LD C
OR D
AND X
ST Y
```

```
LD A
OR B
AND (
    LD C
    OR D
)
ST Y
```

# Egymásba ágyazás

$\%Q0.0 = \%I0.1 \& (\%I0.2 + (\%I0.3 \oplus \%M11))$

LD %I0.1

AND (

    %I0.2

    OR (

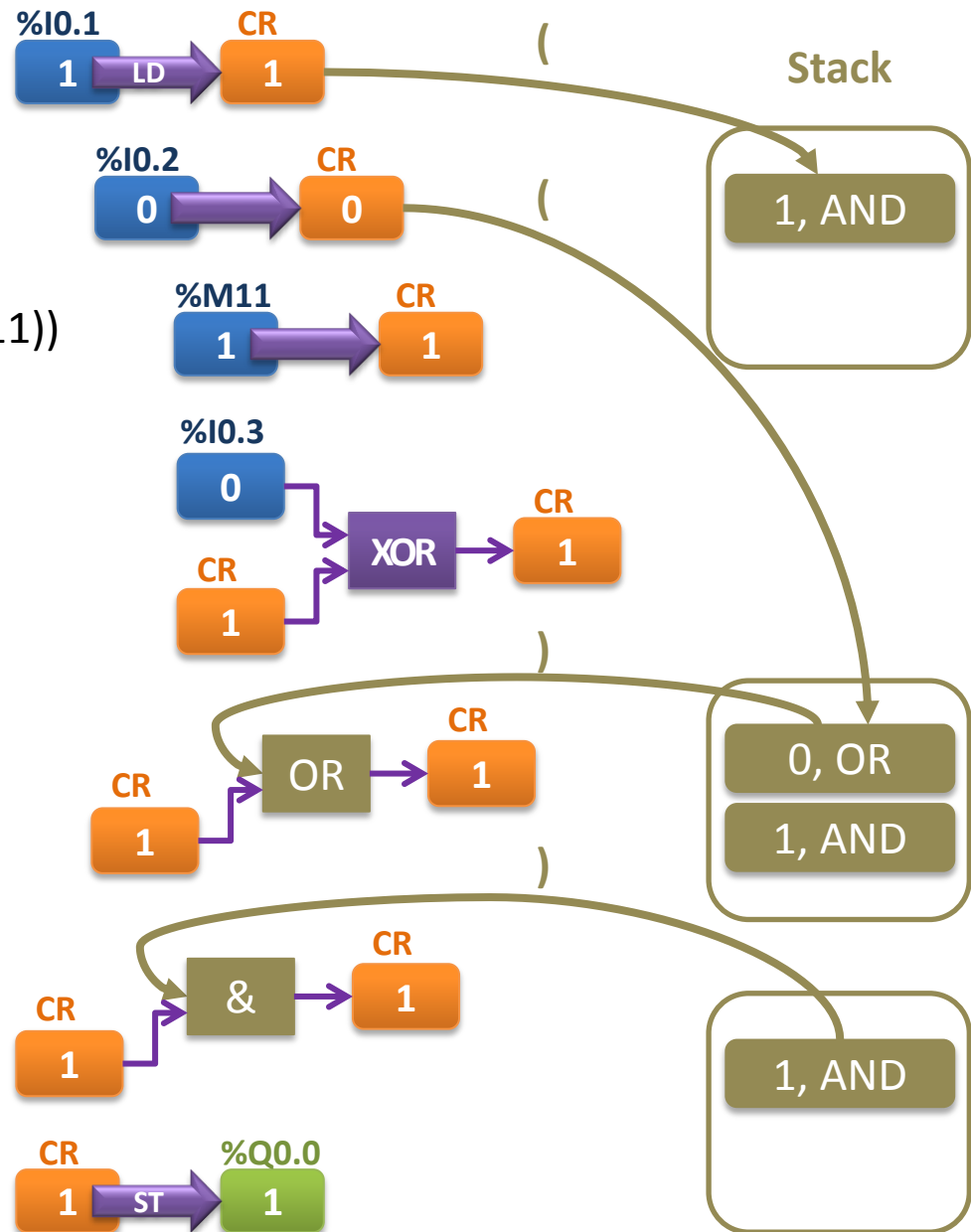
        %M11

        XOR %I0.3

    )

)

ST %Q0.0



# Függvényhívások

- IL-operátorként is használható függvények: hívás közvetlen értékekkel (pl. AND, ADD stb.)
- Más függvények (standard és felhasználói): formális paraméterek vagy közvetlen értékek
- A függvény visszatérési értéke az akkumulátorba kerül



# Függvényhívás közvetlen értékekkel

- Az első paraméter mindig az akkumulátor
- A többi paraméter a függvényhívásban
- A visszatérési érték az akkumulátorba kerül

```
LD 12  
ADD 3
```

```
LD 0  
LIMIT 17, 10
```

# Függvényhívás formális paraméterekkel

- Formális paraméterek megadása
  - Soronként, zárójelben
  - Bemenő paraméterek:  
paraméter := érték
  - Kimenő paraméterek:  
paraméter => cél-változó )
  - A paraméterek sorrendje tetszőleges,  
el is hagyhatók
- A visszatérési érték az  
akkumulátorba kerül

```
LIMIT (  
    EN := TRUE ,  
    MN := 0 ,  
    IN := MyInt ,  
    MX := 10 ,  
    ENO => MyBool  
)
```

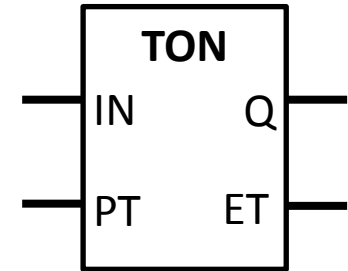
# FB-hívások

- Funkcióblokk-példányok a CAL (CALC, CALCN) utasítással hívhatók
  - Hívás formális paraméterekkel
  - Hívás paraméter-hozzárendeléssel
  - Implicit hívás
- A CAL és RET utasítások az akkumulátor nem definiált értékre állítják
  - A hívott FB nem használhatja az akkumulátort érték betöltése előtt
  - A hívó POU nem használhatja az akkumulátort érték betöltése előtt

# FB-hívás formális paraméterekkel

```
VAR
    TimerIn, TimerOut:    BOOL;
    TimePassed:          TIME;
    Timer1:              TON;
END_VAR

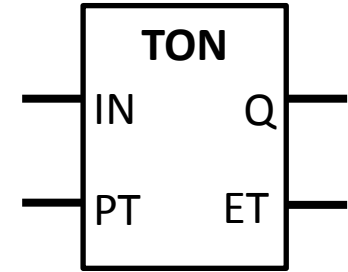
...
CAL Timer1 (
    IN:=TimerIn,
    PT:=T#5s,
    Q=>TimerOut,
    ET=>TimePassed
)
...
```



# FB-hívás paraméter-hozzárendeléssel

```
VAR
    TimerIn, TimerOut:    BOOL;
    TimePassed:           TIME;
    Timer1:               TON;
END_VAR
```

```
...
LD T#5s
ST Timer1.PT
LD TimerIn
ST Timer1.IN
CAL Timer1
LD Timer1.Q
ST TimerOut
LD Timer1.ET
ST TimePassed
...
```

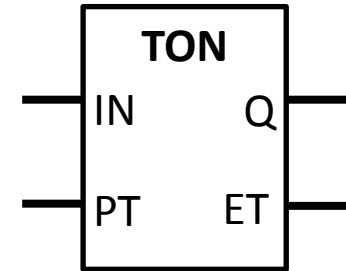


# Implicit FB-hívás

```
VAR
    TimerIn, TimerOut:    BOOL;
    TimePassed:          TIME;
    Timer1:              TON;
END_VAR
```

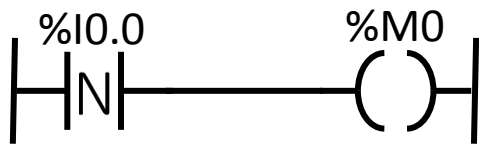
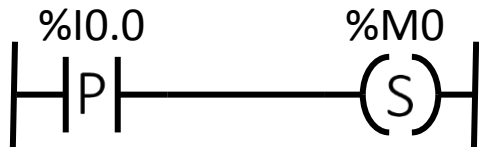
```
...
LD T#5s
PT Timer1
LD TimerIn
IN Timer1

LD Timer1.Q
ST TimerOut
LD Timer1.ET
ST TimePassed
...
```



# Példa: éldetektálás

- Az IEC61131-3 szabvány szerint az éldetektálás az **R\_TRIG** and **F\_TRIG** funkcióblokkokkal lehetséges

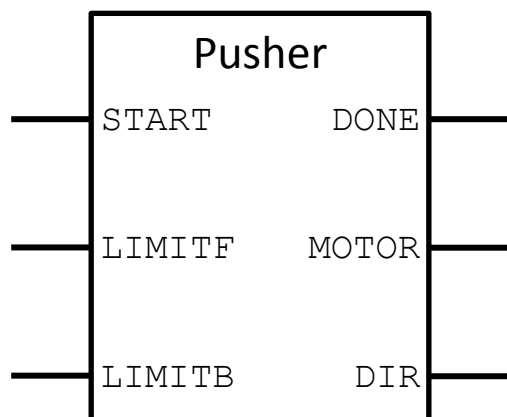
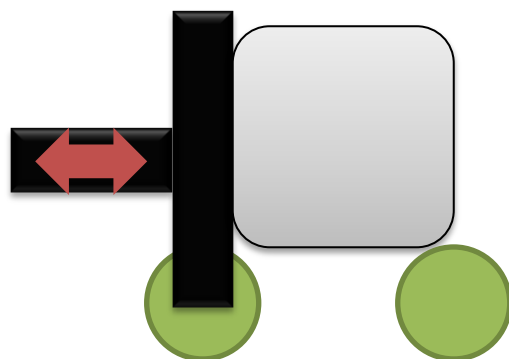


```
VAR
    RisingEdge:      R_TRIG;
    FallingEdge:     F_TRIG;
    AT %I0.0:        BOOL;
    AT %M0:          BOOL;
END_VAR

...
LD %I0.0
ST RisingEdge.CLK
CAL RisingEdge
LD RisingEdge.Q
S %M0

...
CAL FallingEdge(
    CLK:=%I0.0,
    Q=>%M0;
)
```

# Példa: Tologató



- A tologató hátsó véghelyzetéből a START bemenet felfutó élére indul
- Addig mozog előre, amíg az első végálláskapcsoló nem jelez
- Ekkor irányt vált és a hátsó végálláskapcsoló jelzéséig mozog hátra
- Amikor visszatért a kiindulási helyzetbe, a DONE kimenetet egyetlen ciklus idejére igazra állítja



# Tologató – Deklarációs rész

```
FUNCTION_BLOCK Pusher
VAR_INPUT
    Start:          BOOL;
    LimitF:         BOOL;
    LimitB:         BOOL;
END_VAR
VAR_OUTPUT RETAIN
    Motor:          BOOL:=FALSE;
    Dir:            BOOL:=FALSE;
    Done:           BOOL:=FALSE;
END_VAR
VAR
    R_Back:         R_TRIG;
    R_Start:        R_TRIG;
END_VAR
```

# Tologató – Programkód

```
CAL R_Back(CLK:=LimitB,  
        Q=>Done)
```

```
LD Start
```

```
ST R_Start.CLK
```

```
CAL R_Start
```

```
LD R_Start.Q
```

```
AND LimitB
```

```
JMPC MoveFwd
```

```
LD LimitF
```

```
JMPC MoveBwd
```

```
LD R_Back.Q
```

```
R Motor
```

```
RET
```

```
MoveFwd: LD 1
```

```
S Motor
```

```
S Dir
```

```
RET
```

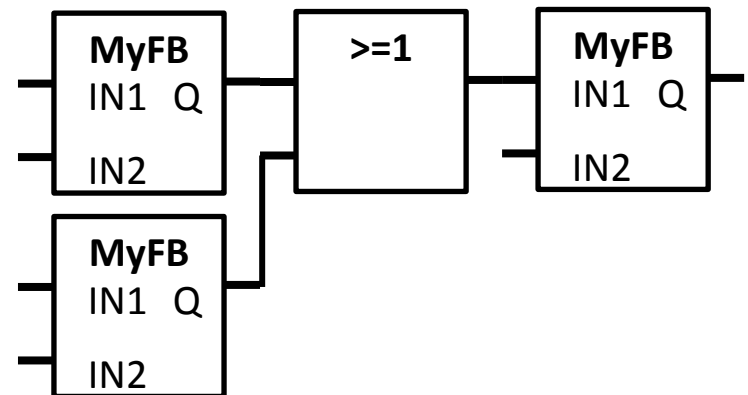
```
MoveBwd: LD 1
```

```
R Dir
```

```
RET
```

# Funkcióblokk-diagram (Function Block Diagram, FBD)

- Magas szintű grafikus nyelv
- Nem logikai műveletek végzésére
- Függvények és FB-k közötti adatfolyam leírása
- Eredet: adatfolyam-diagram
  - Blokkok: függvények és FB-példányok
  - Összeköttetések: változók



# FBD-hálózat (network)

- FBD-programkódok szervezőeleme
- A hálózatok felülről lefelé hajtódnak végre
- A hálózatokhoz címkét kapcsolhatunk
- A címkékre ugorhatunk
- A hálózatokat összekötőkkel (connector) kapcsolhatjuk össze

# FBD-hálózatok elemei

- Függvény- és FB-hívások (blokkok)
- Vezetékek
- Futásvezérlési elemek
- Összekötők

# Függvényhívás

```
FUNCTION MyFun : INT  
VAR_INPUT  
    In1 : BOOL;  
    In2 : BOOL;  
END_VAR  
VAR_OUTPUT  
    Out1 : BOOL;  
END_VAR  
VAR_IN_OUT  
    InOut : INT;  
END_VAR
```

Függvény neve

MyFun

Visszatérési érték  
(informális)

IN1

IN2

InOut

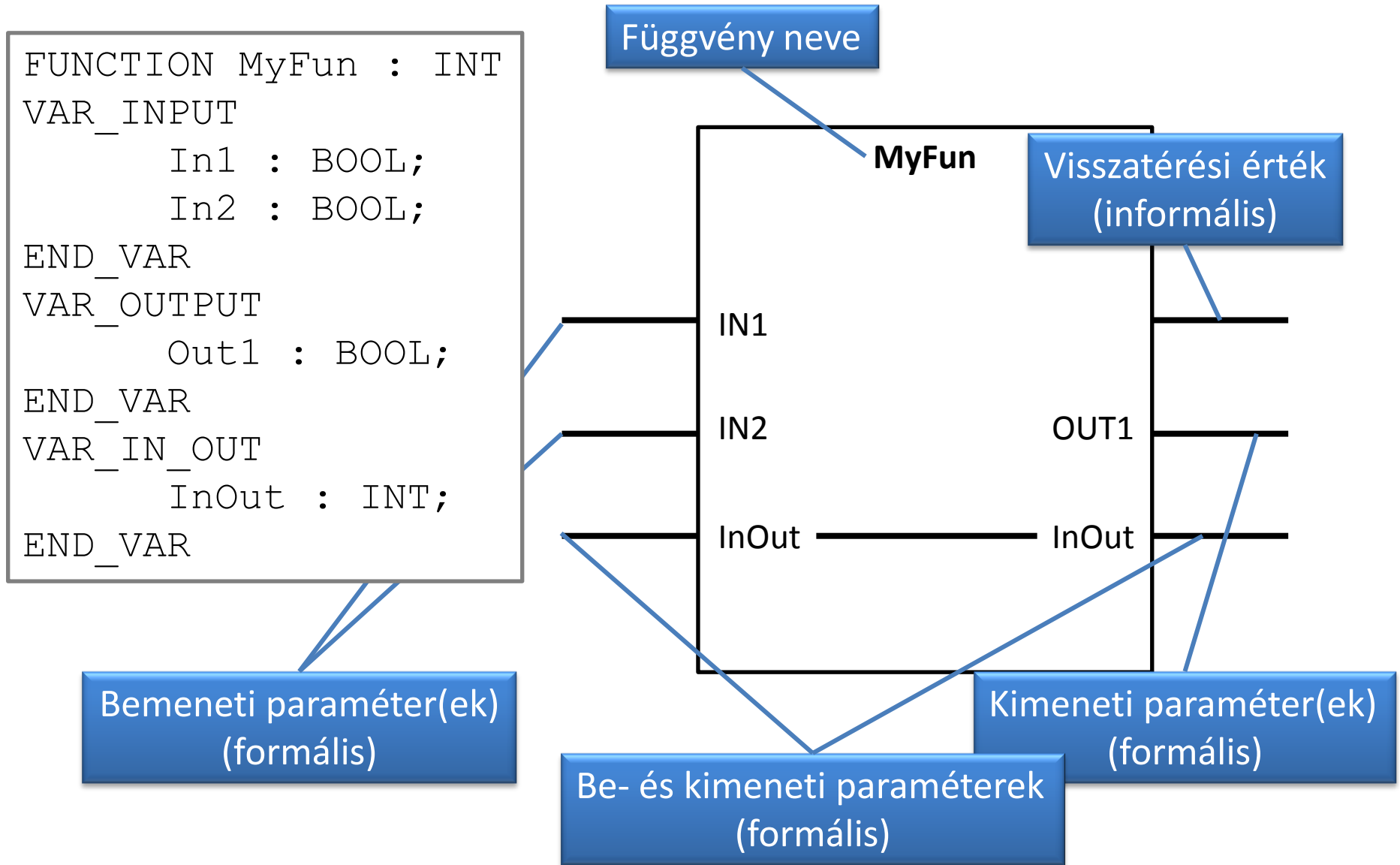
OUT1

InOut

Bemeneti paraméter(ek)  
(formális)

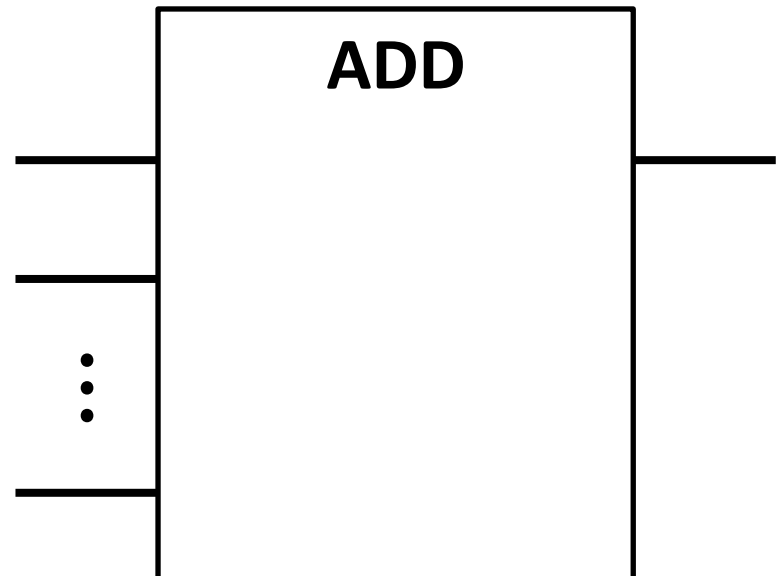
Be- és kimeneti paraméterek  
(formális)

Kimeneti paraméter(ek)  
(formális)



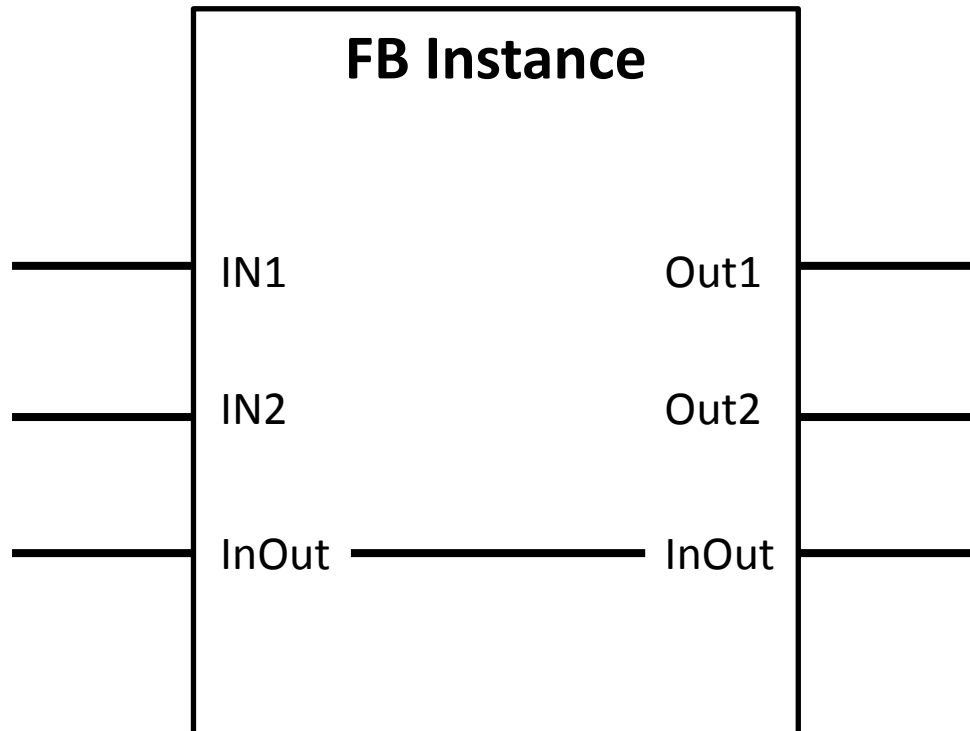
# Függvényhívás

- Overloaded és kiterjeszthető függvények formális paraméterek nélkül hívhatók
  - ADD (+), MUL (\*)
  - AND (&), OR ( $\geq 1$ ), XOR ( $= 2k+1$ )



# FB-hívás

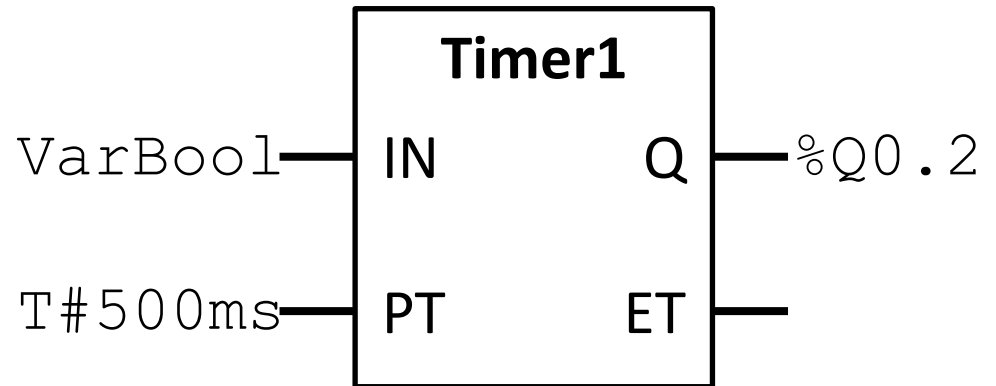
Minden paraméter formális





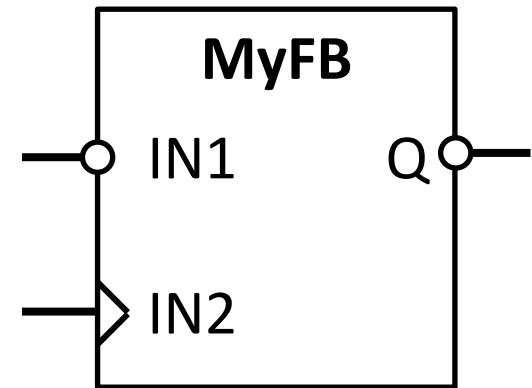
# Blokkok be- és kimenetei

- Változók
- Literálisok
- Vezetékek
- Összekötők
- Nem bekötött
  - Bemenet: adattípus alapértelmezett értéke
  - Kimenet: nem történik hozzárendelés



# Különleges be- és kimenetek

- Negált bemenet/kimenet : o
- Érzékeny bemenetek:
  - Felfutó él: >
  - Lefutó él: <

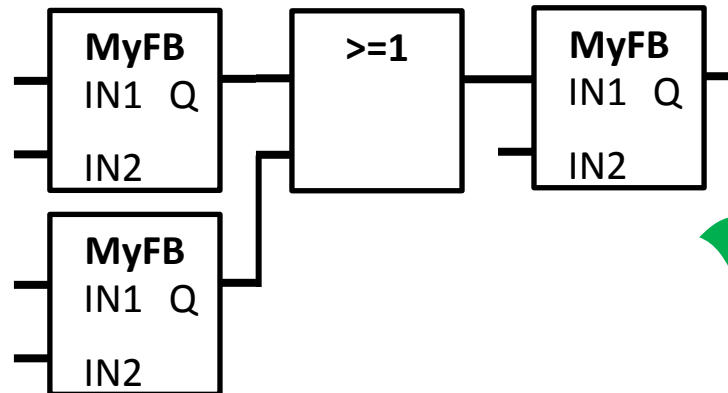
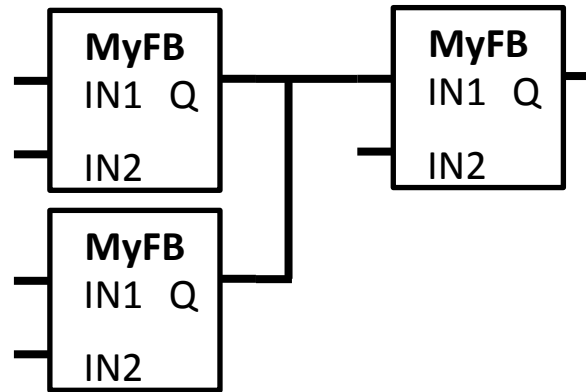
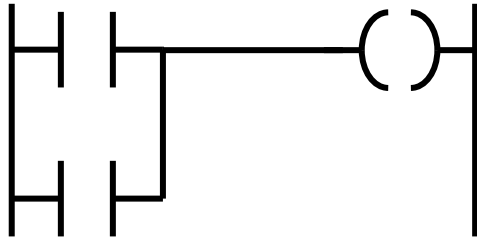


# Vezetékek

- Blokkokat összekötő vízszintes vagy függőleges vezetékek
- Tetszőleges adattípusúak lehetnek
- Csak azonos típusú blokk be- és kimenetek köthetők össze
- Egy blokk-kimenet tetszőleges számú blokk-bemenethez köthető
- Egy blokk-bemenethez csak egyetlen jel köthető

# Összekötések

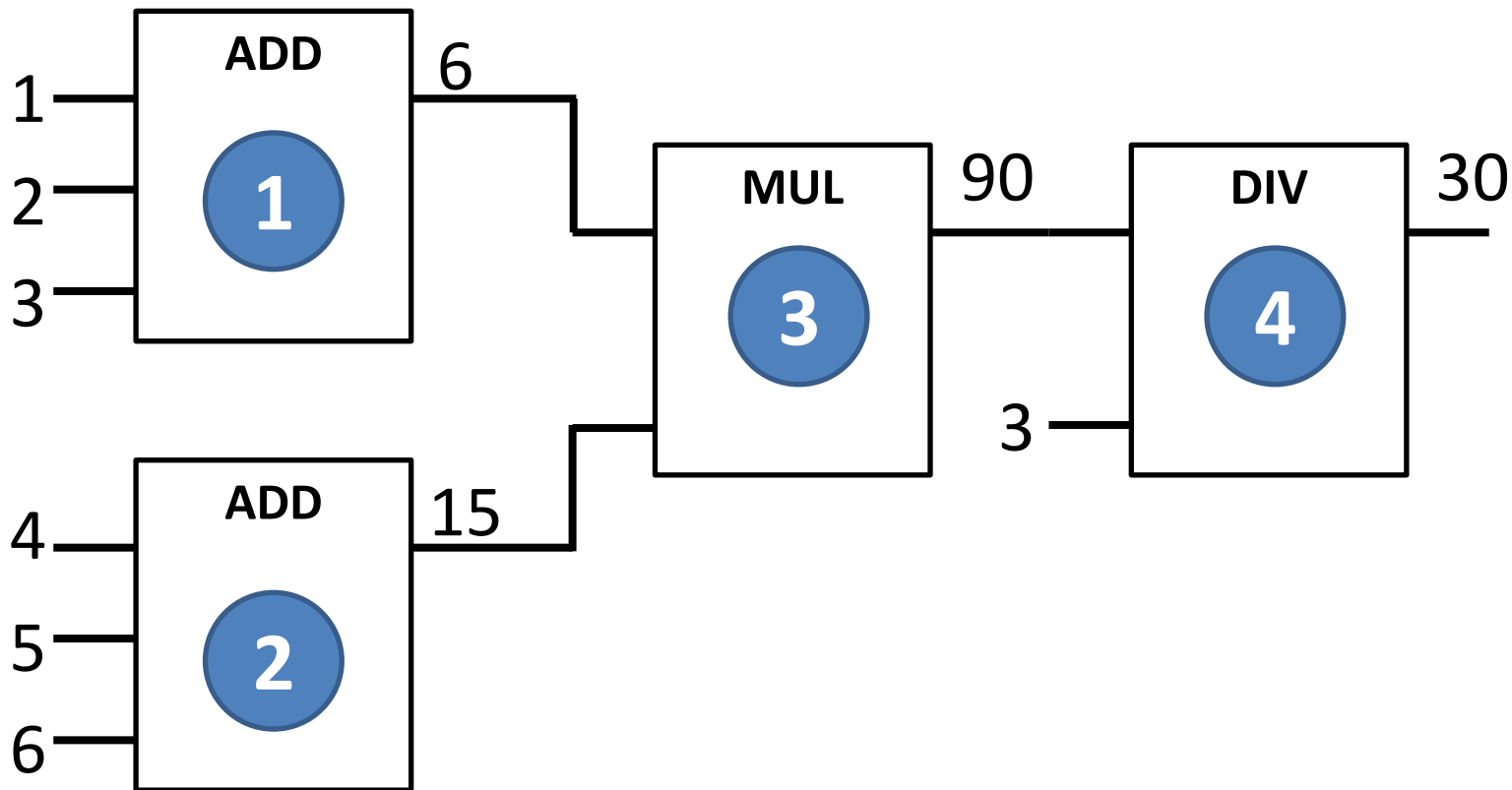
A huzalozott VAGY kapcsolat FBD-ben nem megengedett!



# Hálózat kiértékelése

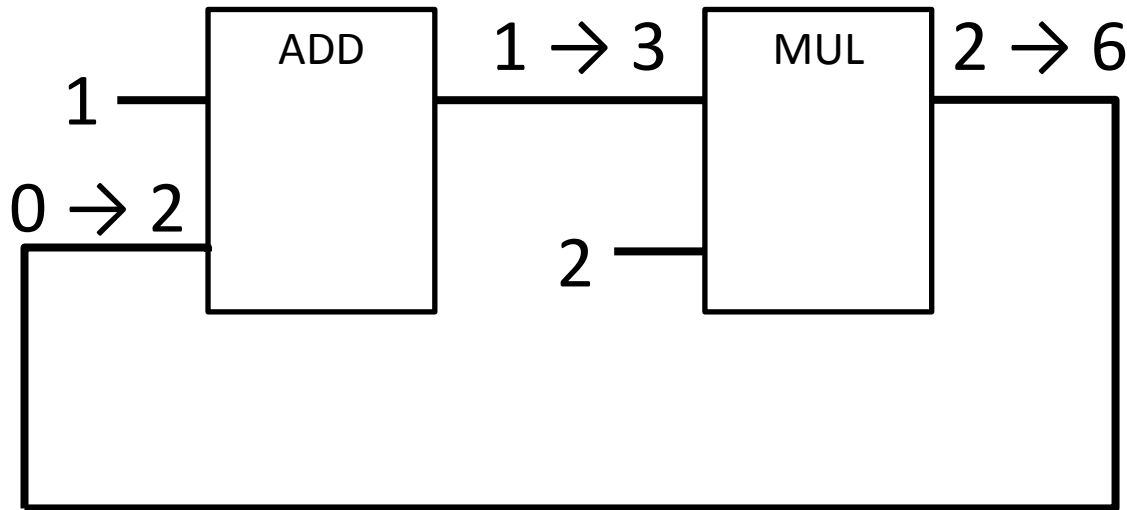
- Egy elem csak akkor értékelhető ki, ha minden bemenetén érvényes érték áll rendelkezésre
- Egy elem kiértékelése mindaddig nem fejeződhet be, amíg mindegyik kimenete nem kerül kiértékelésre
- Egy hálózat kiértékelése mindaddig nem fejeződhet be, amíg mindegyik eleme nem kerül kiértékelésre

# Hálózat kiértékelése - példa



# Visszacsatolás

- Visszacsatolási hurok különböző blokkok között
- Az utolsó ciklusban kapott kimeneti értéket csatolja vissza
- Első futáskor az adattípus kezdeti értékét kapja



# Futásvezérlés

- Ugrás

- Csak feltételes

————>>> LABEL

- A megadott címkére ugrik, ha a bemenete igazra értékelődik ki

- Visszatérés

- Csak feltételes

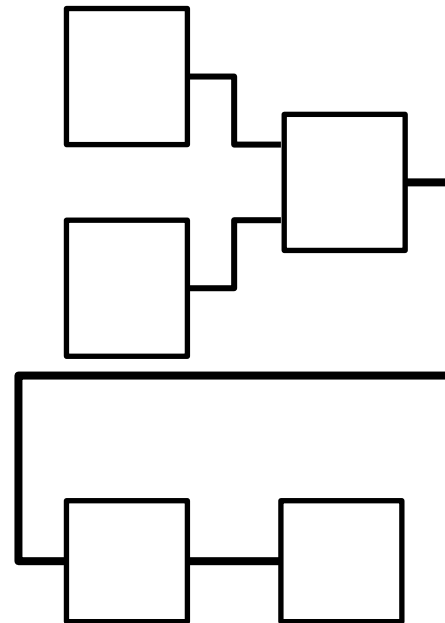
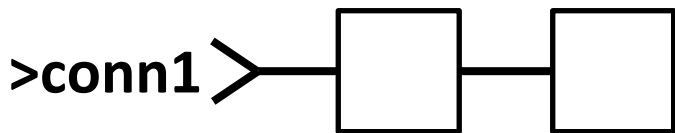
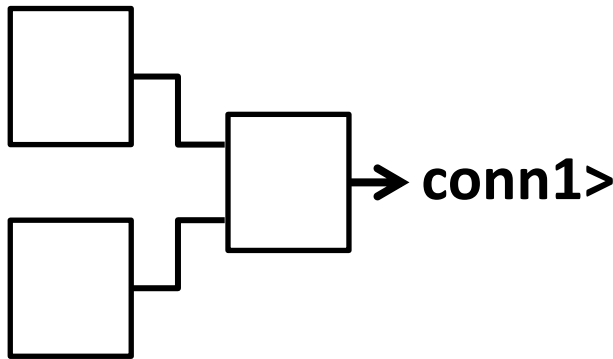
————<(RETURN)

- Visszatér a hívó POU-ba, ha a bemenete igazra értékelődik ki

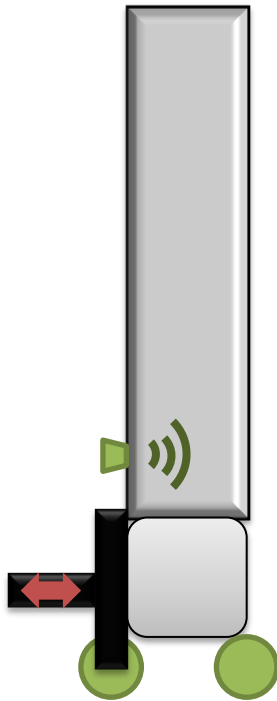


# Összekötők

- Hálózaton belüli adattovábbításra szolgál
- „Új sor karakter” vagy hosszú nyíl
- Hasznos, ha a diagram szélessége korlátozott



# Példa - futószalag



- Indításkor a futószalagnak működni kell
- Ha a futószalag közelítésérzékelője jelez, akkor további 3 másodpercig járassuk a futószalagot, majd állítsuk le és indítsuk el a tologatót
- Ha a tologató végzett, indítsuk újra a futószalagot

# Futószalag – Deklarációs rész

Program PusherConveyor

VAR\_INPUT

S1: BOOL AT %I0.0;

LF: BOOL AT %I0.1;

LB: BOOL AT %I0.2;

END\_VAR

VAR\_OUTPUT

C1: BOOL :=1 AT %Q0.0;

P1Mot: BOOL AT %Q0.1;

P1Dir: BOOL AT %Q0.2;

END\_VAR

VAR

T\_F: TP;

F\_PS: F\_TRIG;

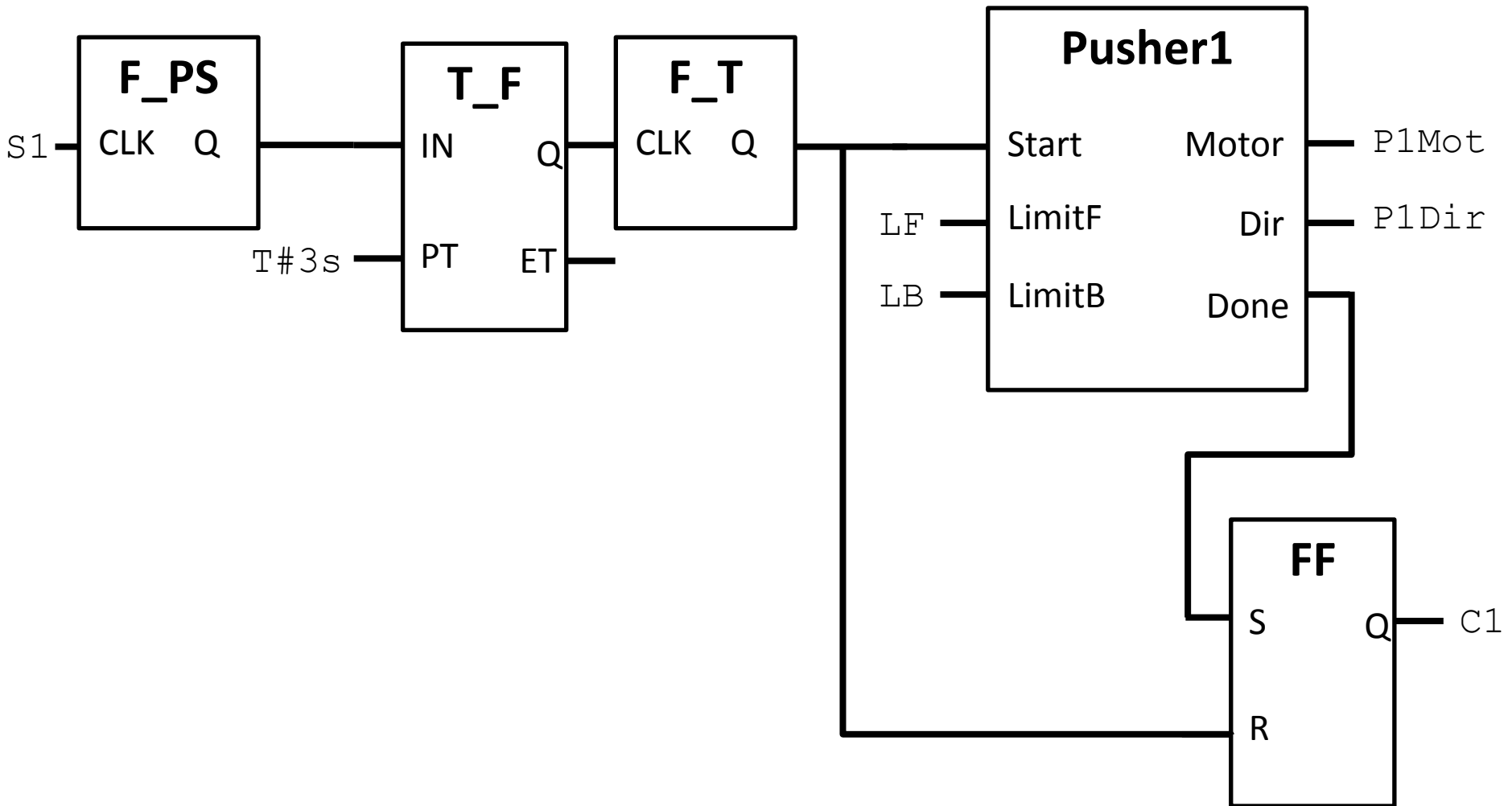
F\_T: F\_TRIG;

Pusher1: Pusher;

FF: SR;

END\_VAR

# Futószalag - Programkód



# Strukturált szöveg (Structured Text, ST)

- Magas szintű szöveges nyelv
- Világos felépítés
- Hatékony programszervezési módok
- A gépi kódra fordítás nem tartható kézben közvetlenül
- A magas absztrakciós szint szuboptimális implementációhoz vezethet



**A teljes szöveges kód végrehajtódik ciklusonként**

# Kifejezések (expression)

- A műveletek a kifejezések eredményét dolgozzák fel
- Egy kifejezés elemei
  - Operandusok – akár más kifejezések
  - Operátorok

# Operandusok

- Literálisok
  - 17, 'my string', T#3s
- Változók (elemiek vagy származtatottak)
  - Var1, VarArray[12]
- Függvények visszatérési értékei
  - Add(2,3), sin(1.76)
- Más kifejezések
  - 10+20 (=Add(10,20))

Operátor	Leírás	Példa → Eredmény	Prioritás
( )	Zárójel: végrehajtási sorrendre hat	$(3+2)*(4+1) \rightarrow 25$	
<fcn name>	Függvényhívás	CONCAT('PL','C') → 'PLC'	
-	Ellentett (aritmetikai)	$-10 \rightarrow -10$ ( $-1 \times 10$ )	
NOT	Komplement (logikai negálás)	NOT TRUE → FALSE	
**	Hatványozás	$2^{**}7 \rightarrow 128$	
*	Szorzás	$2*7 \rightarrow 14$	
/	Osztás	$30/6 \rightarrow 5$	
MOD	Maradékképzés (modulo)	$32 \text{ MOD } 6 \rightarrow 2$	
+	Összeadás	$32+6 \rightarrow 38$	
-	Kivonás	$32-6 \rightarrow 26$	
<, <=, >, >=	Összehasonlítás	$32 < 6 \rightarrow \text{FALSE}$	
=	Egyenlőség	$T\#24h = T\#1d \rightarrow \text{TRUE}$	
<>	Egyenlőtlenség	$2 <> 5 \rightarrow \text{TRUE}$	
&, AND	Logikai ÉS	TRUE AND FALSE → FALSE	
XOR	Logikai kizáró vagy (XOR)	TRUE XOR FALSE → TRUE	
OR	Logikai VAGY	TRUE OR FALSE → TRUE	



# Függvényhívások

- Kifejezésekben: a kifejezés a függvény visszatérési értékét használja fel
- Formális hívás
  - Zárójelek között, paraméter-azonosítókkal
  - A paraméterek sorrendje tetszőleges
  - Elhagyott paraméter esetén a függvény annak kezdeti értékét használja
  - `LIMIT (MN:=0, MX:=10, IN:=7, Q=>VarOut)`
- Informális hívás
  - Közvetlen értékek meghatározott sorrendben
  - `LIMIT (0, 7, 10)`

# Műveletek (statement)

Kulcsszó	Művelettípus
<code>:=</code>	Értékadás
<code>&lt;FB name&gt; (parameters)</code>	FB hívás
<code>RETURN</code>	Visszatérés a hívó POU-ba
<code>IF</code>	Kiválasztás
<code>CASE</code>	Kiválasztás
<code>FOR</code>	Iteráció
<code>WHILE</code>	Iteráció
<code>REPEAT</code>	Iteráció
<code>EXIT</code>	Iteráció befejezése

# Értékadás

```
VAR
    d: INT;
    e: ARRAY [0..9] OF INT;
END_VAR
```

- := operátor

- Értékadás

- Egyelemű változónak
- Tömb elemének

```
d:=4;
```

```
e[3]:=d**2;
```

- Adattípusok

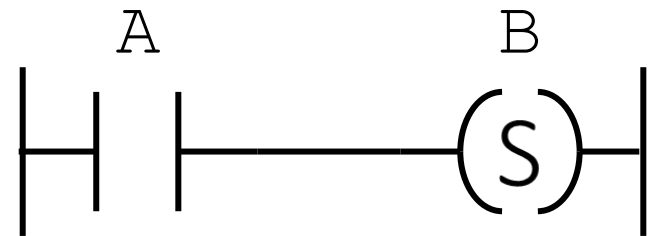
- A bal és jobb oldal adattípusa kompatibilis
- A típuskonverziós függvények kifejezésként használhatók

```
d:=REAL_TO_INT(SIN(2))
```

# Bináris értékadás

- Nem a létradiagramban megszokott logikai függvény (kifejezés), hanem művelet
- Ha az értékadás nem hajtódik végre (pl. IF), akkor a változó megőrzi addigi értékét, nem íródik felül

```
IF A  
THEN B:=1;  
END_IF;
```



# FB hívás

- Az FB-hívás művelet, kifejezésben nem megengedett
- Formális hívás
  - Paraméter-azonosítókkal
  - A paraméterek sorrendje tetszőleges
  - Kihagyott paraméterek helyettesítése
    - Előző híváskori értékükkel
    - Első hívás esetén kezdeti értékükkel
- Informális hívás
  - Közvetlen értékek megfelelő sorrendben

# FB hívás - példa

```
PROGRAM MyProg
VAR
    MyTimer:    TON;
    A:          BOOL;
    MyInt :     INT;
END_VAR
(*...*)
MyTimer (PT:=T#1s, IN:=(MyInt=7), Q=>A);
MyTimer ((MyInt=7), T#1s);
A:=MyTimer.Q;
(*...*)
END_PROGRAM;
```

# Kiválasztás

- Kiválasztás logikai (Boolean) értékű kifejezés alapján
- Minden ágban tetszőleges számú műveletből álló blokk állhat
- ELSIF és ELSE ágak elhagyhatók
- END\_IF; használata kötelező

```
IF <BOOL expression> THEN
    <statement block>
ELSIF <BOOL expression> THEN
    <statement block>
ELSIF <BOOL expression> THEN
    <statement block>
ELSE
    <statement block>
END_IF;
```

# Kiválasztás - példa

```
IF (Pusher_Move=1) THEN
    MotorOut:=1;
    DirOut:=1;
ELSIF (Pusher_Move=2) THEN
    MotorOut:=1;
    DirOut:=0;
ELSE
    MotorOut:=0;
    DirOut:=0;
END_IF;
```



# Eset-kiválasztás

- Kiválasztás egész-értékű kifejezés alapján
- Az esetekhez több érték is megadható
- Alapértelmezett eset: ELSE (elhagyható)
- END\_CASE; nem hagyható el

```
CASE <INT expression> OF
<value1>:          <statement block>
<value2>,<value3>: <statement block>
ELSE <statement block>
END_CASE;
```

# Eset-kiválasztás - példa

```
CASE Pusher_Move OF
  1:  MotorOut:=1;
      DirOut:=1;
  2:  MotorOut:=1;
      DirOut:=0;
ELSE
      MotorOut:=0;
      DirOut:=0;
END_CASE;
```

# Iteráció

- Az iteráció egyetlen PLC-cikluson belül hajtódik végre
- Ha óvatlanul használjuk, akkor rontja a determinizmust és watchdog-hibát is okozhat
- Ne használjuk eseményre való várakozásra!
- Használhatjuk
  - Tömb vagy adatmező elemeinek vizsgálata
  - Egy művelet megadott számú ismétlésére

# While hurok

- A feltételes kifejezést a műveletek végrehajtása előtt vizsgálja
- Akkor hajtja végre a műveleteket, ha a feltételes kifejezés értéke TRUE

```
WHILE <BOOL expression> DO  
    <statement block>  
END_WHILE;
```

# While hurok - példa

```
VAR
```

```
    MyArray:    1..10 OF INT;
```

```
    i:          INT;
```

```
    MaxVal:    INT:=0;
```

```
END_VAR
```

```
(* ... *)
```

```
i:=1;
```

```
WHILE (i<=10) DO
```

```
    IF (MyArray[i]>MaxVal)
```

```
        THEN MaxVal:=MyArray[i];
```

```
    END_IF;
```

```
    i:=i+1;
```

```
END_WHILE;
```

```
(* ... *)
```

# Repeat – Until hurok

- A feltételes kifejezést a műveletek végrehajtása után vizsgálja (a műveleti blokk legalább egyszer végrehajtottodik)
- Az iterációt a feltételes kifejezés TRUE értéke esetén fejezi be

```
REPEAT
    <statement block>
UNTIL <BOOL expression>
END_REPEAT;
```

# Repeat hurok - példa

```
VAR
```

```
    MyArray:    1..10 OF INT;
```

```
    i:         INT;
```

```
    MaxVal:    INT:=0;
```

```
END_VAR
```

```
(* ... *)
```

```
i:=0;
```

```
REPEAT
```

```
    i:=i+1;
```

```
    IF (MyArray[i]>MaxVal)
```

```
        THEN MaxVal:=MyArray[i];
```

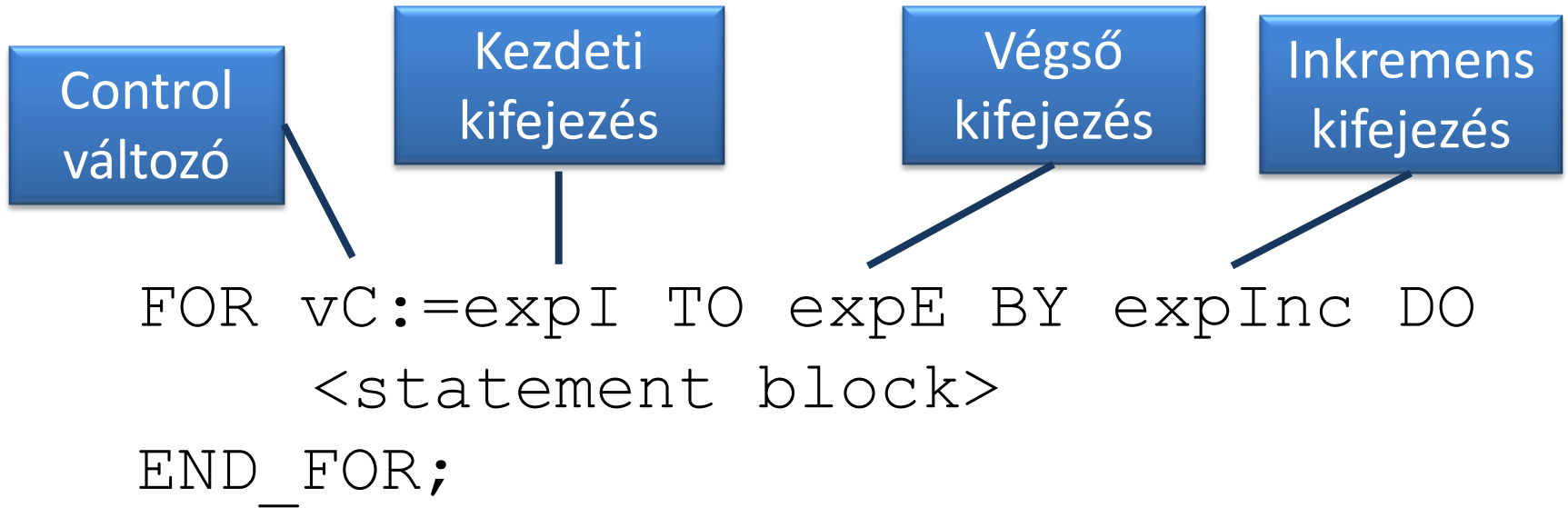
```
    END_IF;
```

```
    UNTIL (i=10)
```

```
END_REPEAT;
```

```
(*...*)
```

# For hurok



- A négy változó/kifejezés azonos adattípusú kell, hogy legyen (SINT, INT, DINT)
- A Control változónak, valamint a kezdeti és végkifejezésben szereplő változóknak nem adható érték a hurkon belül
- Az inkremens kifejezésben szereplő változónak a hurkon belül is adható érték (bár nem ajánlott)



# For hurok - példa

```
VAR
    MyArray:    1..10 OF INT;
    i:          INT;
    MaxVal:     INT:=0;
END_VAR
(* ... *)
FOR i:=10 TO 1 BY -1 DO
    IF (MyArray[i]>MaxVal)
        THEN MaxVal:=MyArray[i];
    END_IF;
END_FOR;
(* ... *)
```

# Kilépés hurkokból

- A hurkokból az `EXIT` művelettel lehet kilépni
- Csak a legbelső szinten hat

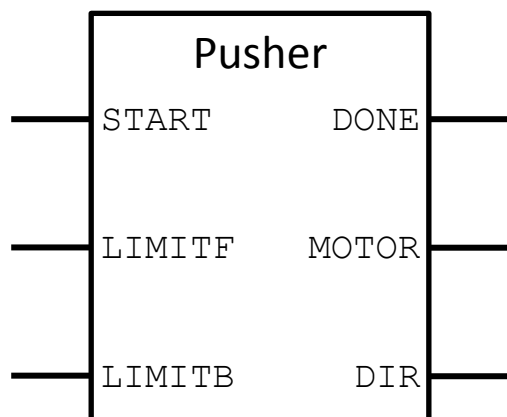
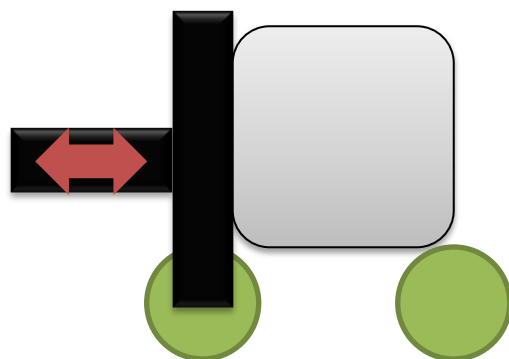
```
j:=0;
WHILE (j<10) DO
  i:=0;
  WHILE (i<10) DO
    IF (i=j) THEN EXIT;
    ELSE i:=i+1;
    END_IF;
  END_WHILE;
  j:=j+1;
END_WHILE
```

i	j
0	0
1	0
1	1
2	0
2	1
2	2
3	0
...	

# Visszatérés a hívó POU-ba

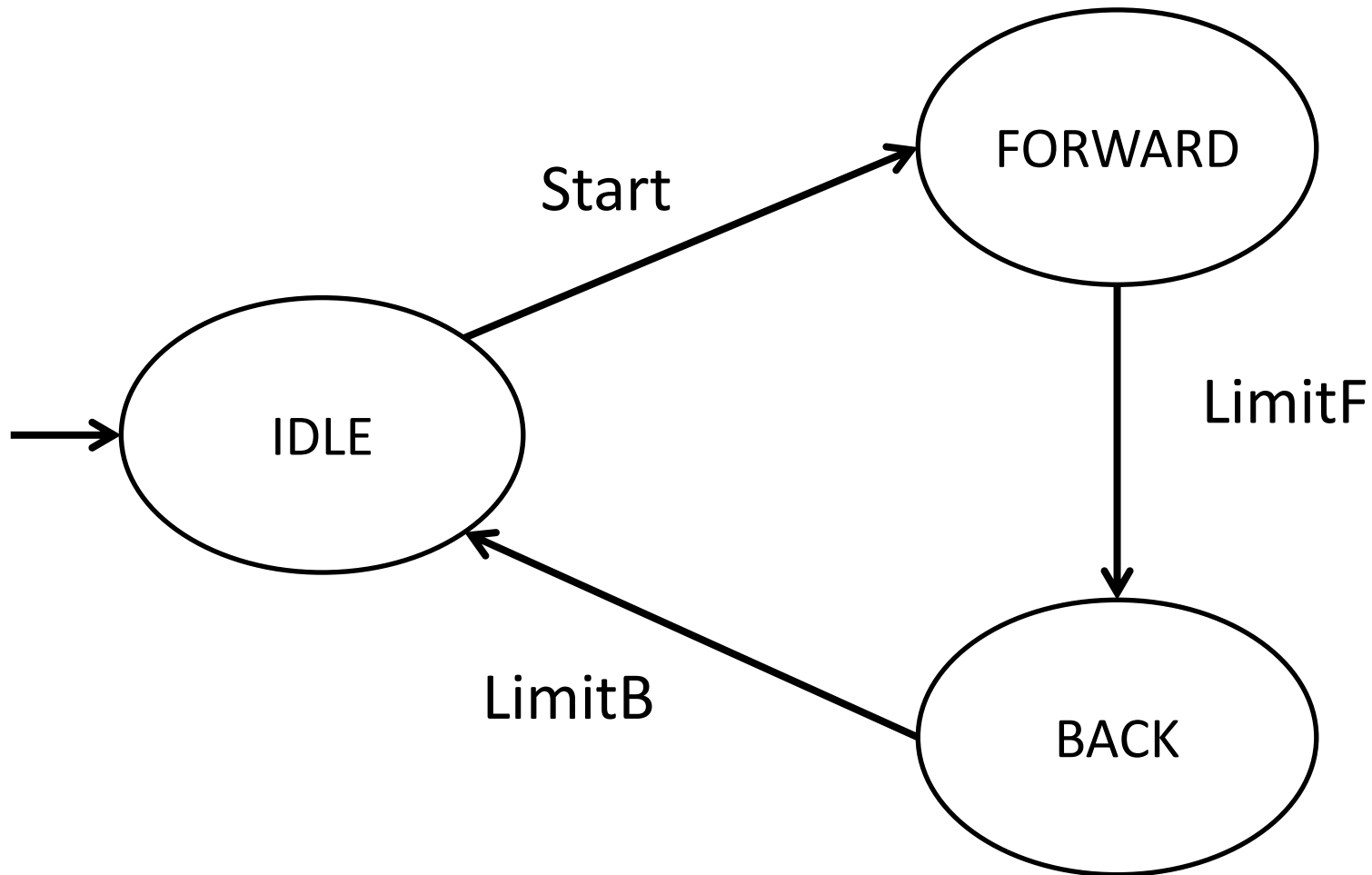
- RETURN kulcsszó
- Kiválasztás művelet tetszőleges ágában használható feltételes visszatérésre
- Függvények esetén a visszatérési értéket előbb be kell állítani (függvénynévvel egyező nevű változó)
- Ha hiányzik, az utolsó sor végrehajtása után történik meg a visszatérés

# Példa: Tologató



- A tologató hátsó véghelyzetéből a START bemenet felfutó élére indul
- Addig mozog előre, amíg az első végálláskapcsoló nem jelez
- Ekkor irányt vált és a hátsó végálláskapcsoló jelzéséig mozog hátra
- Amikor visszatért a kiindulási helyzetbe, a DONE kimenetet egyetlen ciklus idejére igazra állítja

# A tologató állapotgépe



# Pusher - deklarációs rész

```
FUNCTION_BLOCK FBPusher
VAR RETAIN
    StateEnum: (Idle, Fwd, Bwd);
    R_Start:   R_EDGE;
END_VAR
VAR_OUTPUT
    Motor: BOOL :=0;
    Dir:   BOOL;
    Done:  BOOL;
END_VAR
VAR_INPUT
    Start : BOOL;
    LimitF: BOOL;
    LimitB : BOOL;
END_VAR
```

# Pusher - programkód

```
R_Start (CLK:=Start);
```

```
CASE StateEnum OF
```

```
  Idle:      IF (R_Start.Q) THEN StateEnum:=Fwd; END_IF;
```

```
             Done:=FALSE;
```

```
  Fwd:      IF (LimitF) THEN StateEnum:=Bwd; END_IF;
```

```
  Bwd:      IF (LimitB) THEN
```

```
             StateEnum:=Idle;
```

```
             Done := TRUE;
```

```
          END_IF;
```

```
END_CASE;
```

```
Motor:=NOT (StateEnum=Idle);
```

```
Dir:=(StateEnum=Fwd);
```

```
END_FUNCTION_BLOCK;
```

# Sorrendi folyamatábra

## *(Sequential Function Chart, SFC)*

- Az SFC a programfolyamot írja le
- Tárolnia kell az aktuális állapotot: függvény nem valósítható meg SFC-ben
- A kimeneti változók érvényessége sokszor nehezen értelmezhető: egyes fejlesztői környezetek FB megvalósítását sem engedik SFC-ben



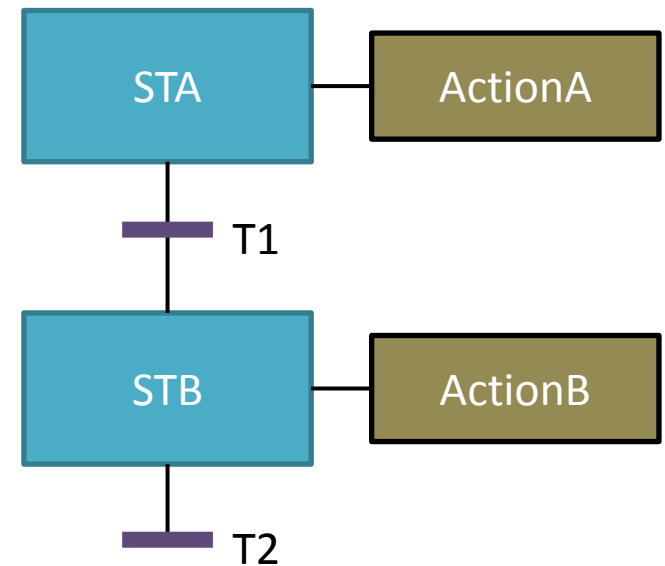
# Sorrendi folyamatábra

*(Sequential Function Chart, SFC)*

- Cél: komplex programok kisebb részekre bontása és az azok közötti programfolyam leírása
- Eredet
  - Folyamatábra
  - Petri-hálók
  - Grafcet

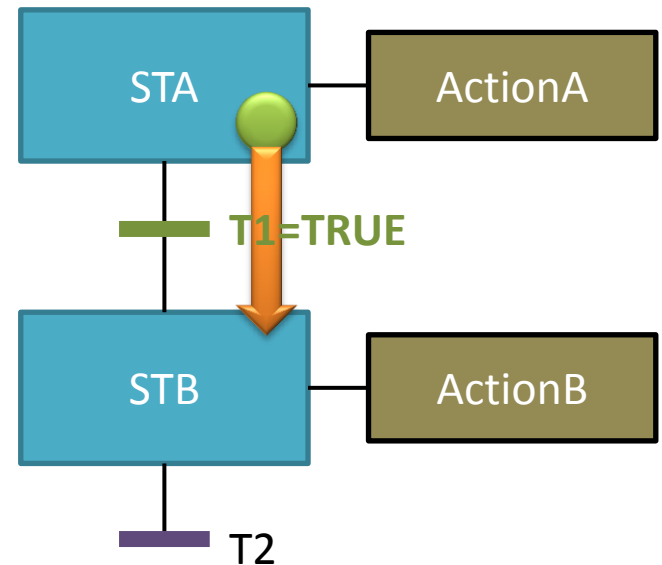
# SFC elemek

- SFC: páros gráf
  - Csomópontok:
    - Lépések (*step*)  $\approx$  állapotok
    - Átmenetek (*transition*) :  
logikai értékre kiértékelődő kifejezések
  - Élek
- Lépésekhez rendelt akciók (*action*)



# Zseton-játék

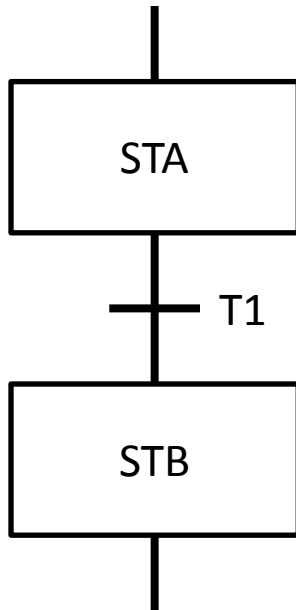
- Az aktív lépést egy zseton (token) jelzi
- A zseton akkor kerül át egy másik lépéshez, amikor egy oda vezető átmenet feltétele igazra értékelődik ki
- Az aktív lépéshez tartozó művelet alapértelmezésben ciklikusan végrehajtódik



# Lépés – Átmenet szekvenciák

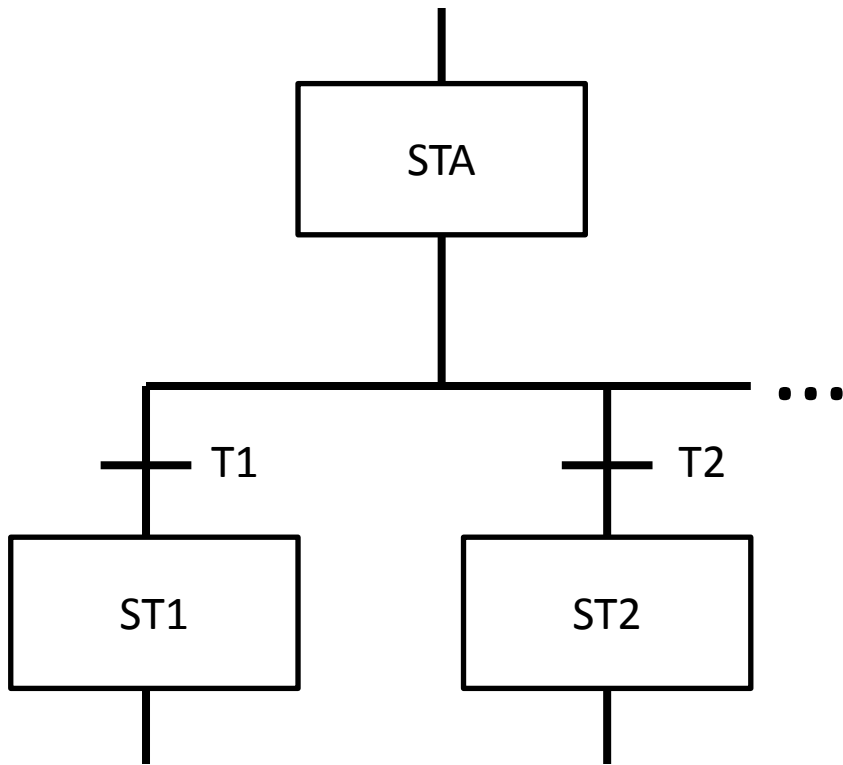
- Egyszerű szekvencia
- Divergens utak (elágazás)
  - Szekvencia-hurok
  - Szekvencia átugrása
- Párhuzamos végrehajtás

# Egyszerű szekvencia



- Amikor T1 igazra értékelődik ki
  - STA deaktiválódik
  - STB aktiválódik

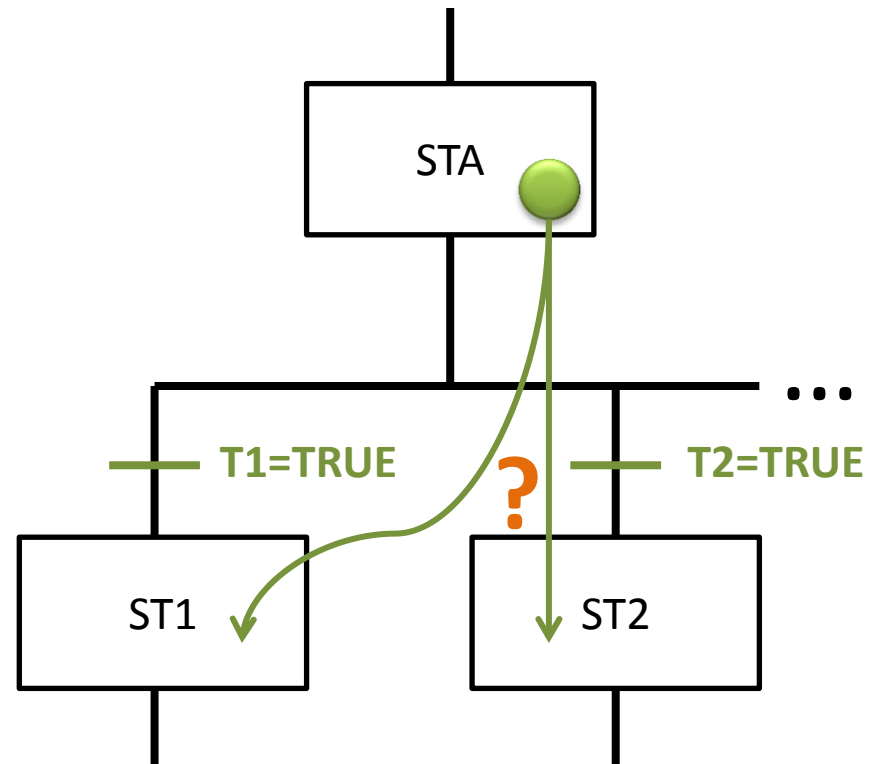
# Divergens utak



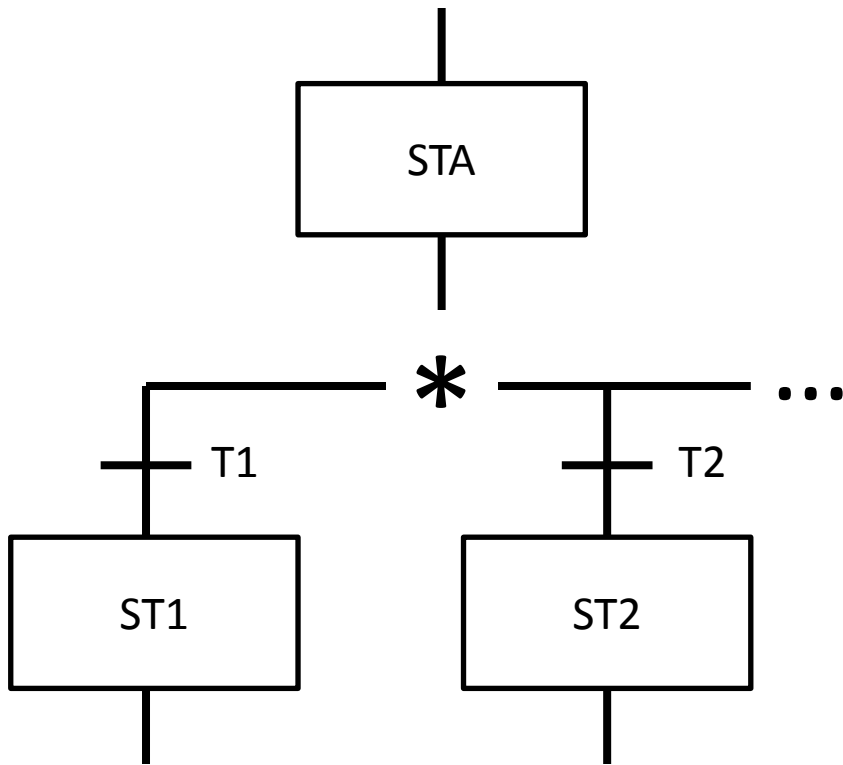
- Egy lépés után több átmenet is következik
- Az első igazra kiértékelődő  $T_i$  átmenet deaktiválja STA-t és aktiválja  $ST_i$ -t
- Az ágak közül csak egy lesz aktív

# Divergens utak átmeneteinek kiértékelése

- Szabvány szerint
  - Kiértékelés balról jobbra
  - Kiértékelés explicit prioritással
  - Kiértékelés felhasználói irányítással
- Általános gyakorlati megvalósítás: kiértékelés balról jobbra



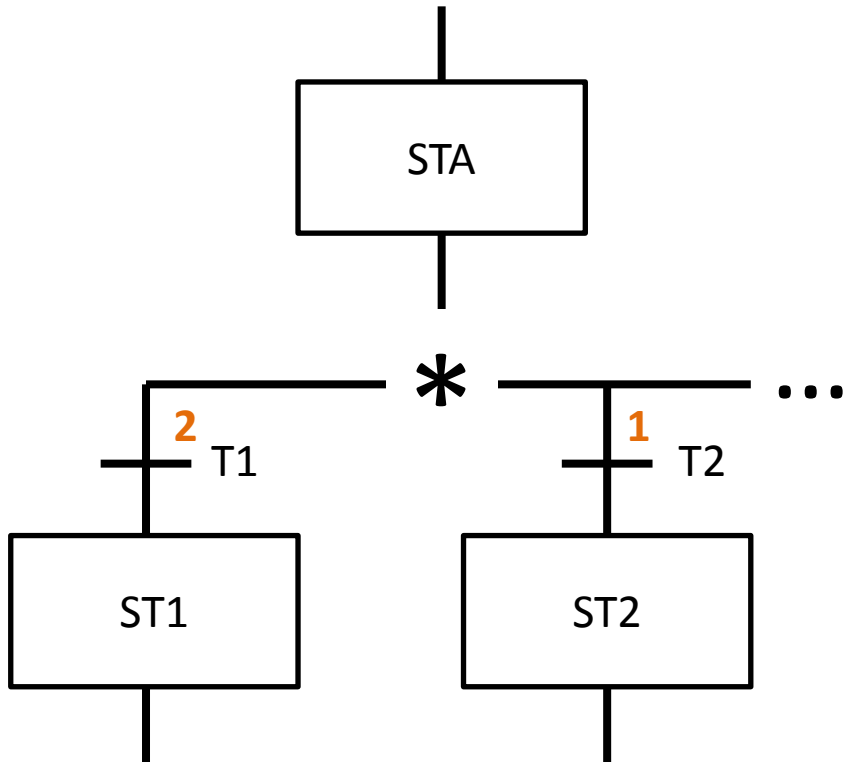
# Kiértékelés balról jobbra



- Amíg STA aktív, addig az átmeneteket folyamatosan kiértékeljük balról jobbra haladva
- Az első igazra kiértékelődő  $T_i$  átmenet deaktiválja STA-t és aktiválja  $ST_i$ -t

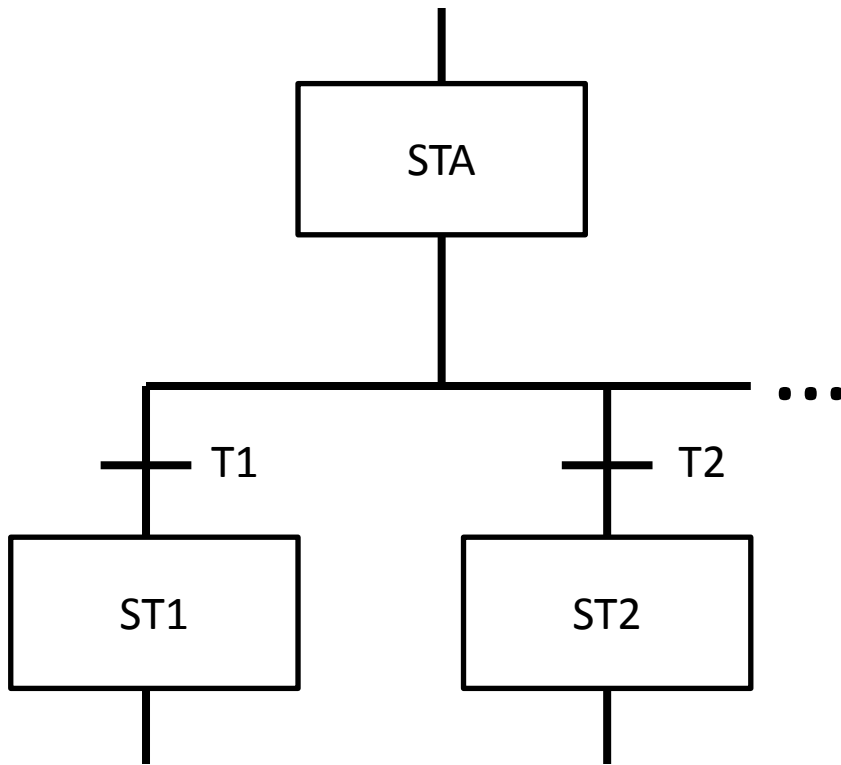


# Kiértékelés felhasználó által megadott prioritással



- Az átmeneteket a megadott sorrendben értékeljük ki
- Az alacsonyabb érték jelzi a magasabb prioritást

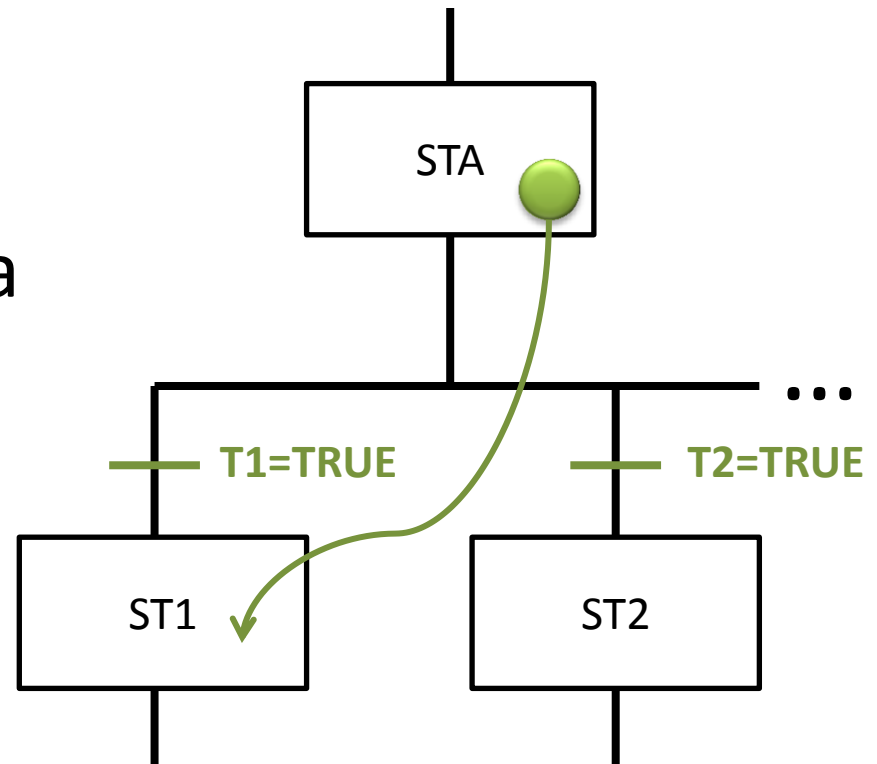
# Kiértékelés felhasználói irányítással



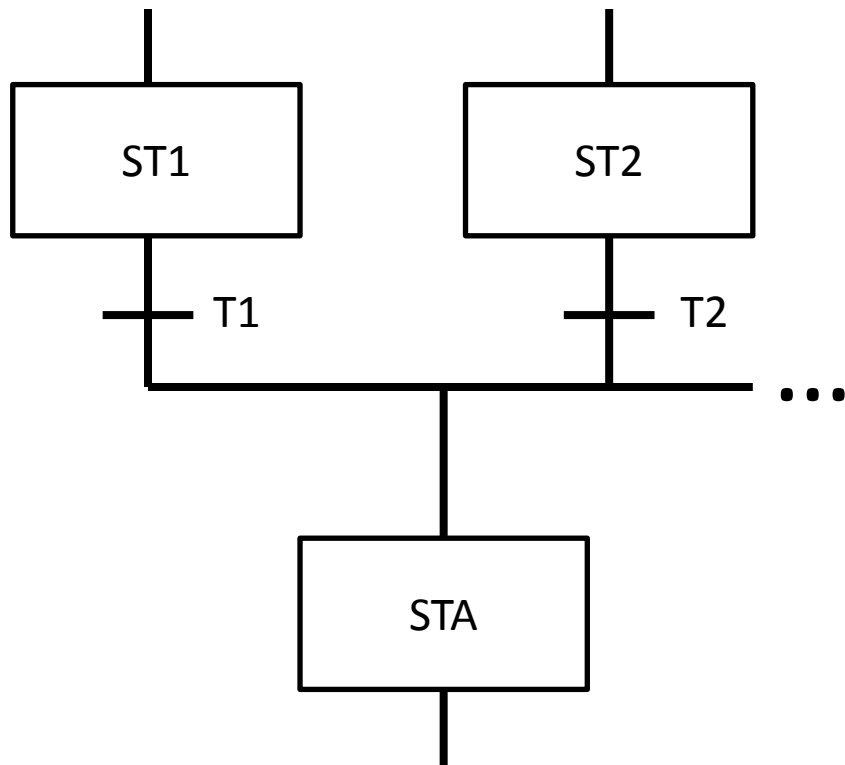
- Az átmenetek kiértékelésének sorrendje nem meghatározott
- A felhasználónak kell biztosítania, hogy a feltételek kölcsönösen kizáróak legyenek

# Divergens utak - megjegyzés

- A legtöbb fejlesztői környezet csak a standard (balról jobbra haladó) kiértékelést támogatja
- Ebben az esetben a jelölésből kimarad a \*

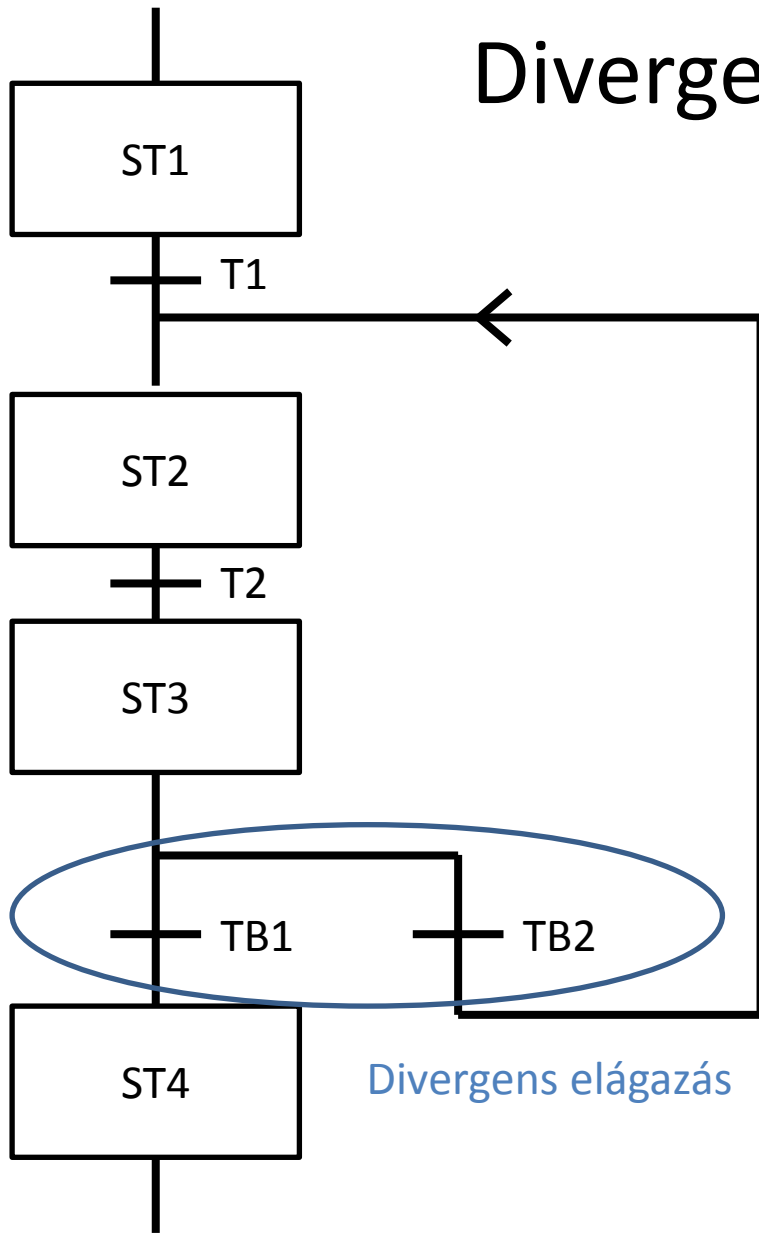


# Divergens utak találkozása



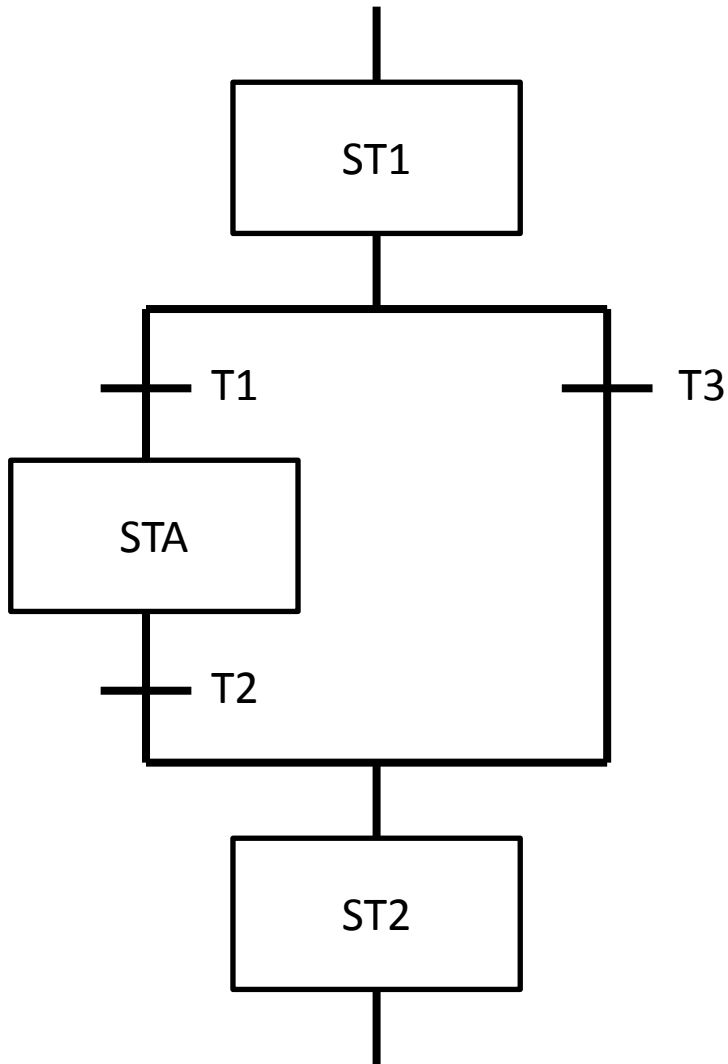
- STA-t akkor aktiváljuk, ha  $ST_i$  aktív és  $T_i$  igazra értékelődik ki
- Ugyanekkor  $ST_i$ -t deaktiváljuk

# Divergens utak: szekvencia-hurok



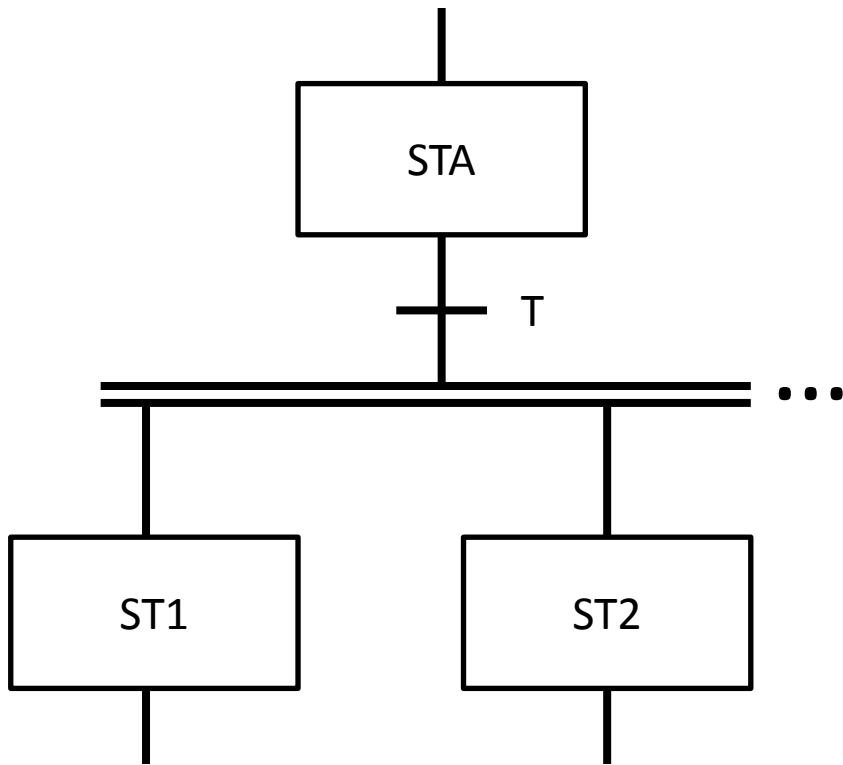
- Előd-lépéshez visszakanyarodó divergens út
- Tetszőleges prioritás-modell használható

# Divergens utak – szekvencia átugrása



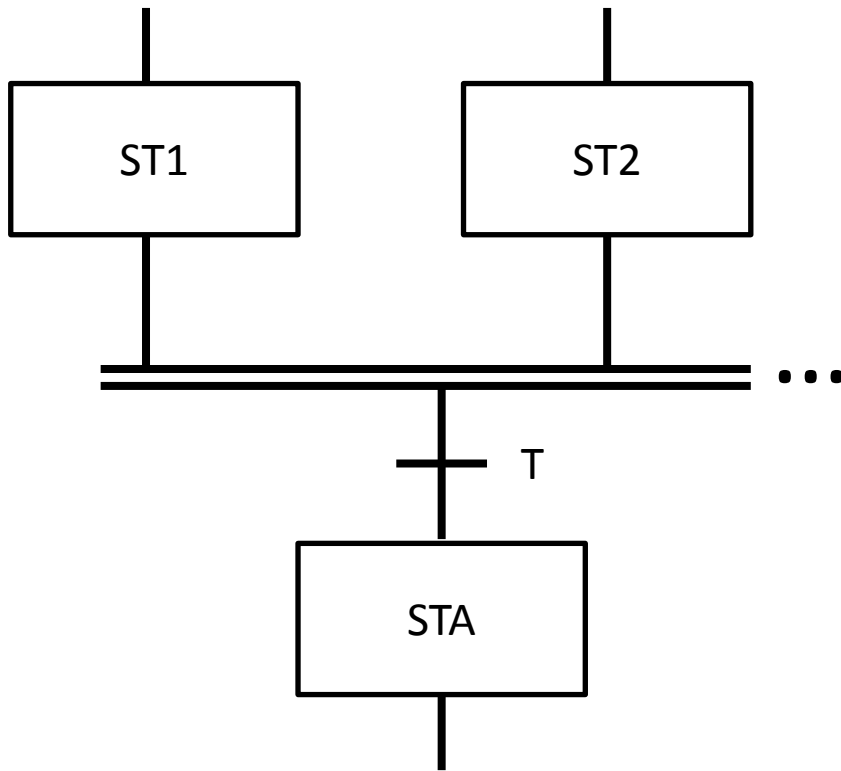
- Az **STA** lépést átugorjuk, ha **T3** igazra értékelődik ki (miközben **T1** hamis)

# Párhuzamos végrehajtás



- Ha STA aktív és T igazra értékelődik ki, akkor STA-t deaktiváljuk és az összes  $ST_i$ -t aktiváljuk
- A token „osztódik”

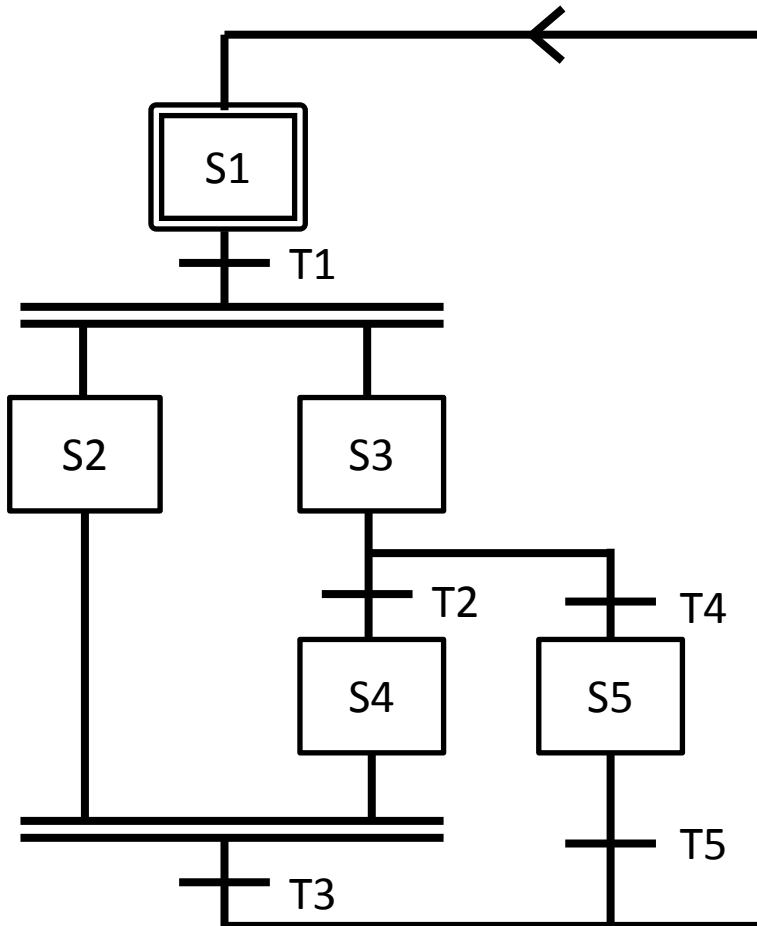
# Párhuzamos szekvenciák találkozása



- STA-t akkor aktiváljuk, ha
  - T igazra értékelődik ki
- **ÉS**
  - Az összes  $ST1...STn$  lépés aktív
- Ekkor az „osztódott” tokenek egyesülnek

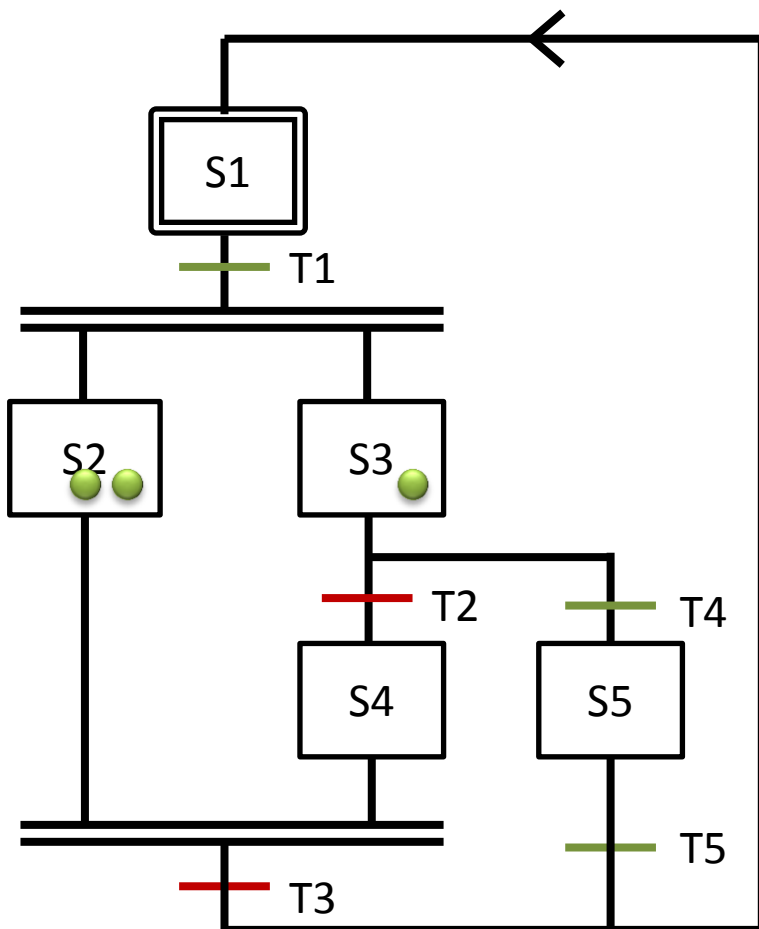


# Nem biztonságos hálózatok



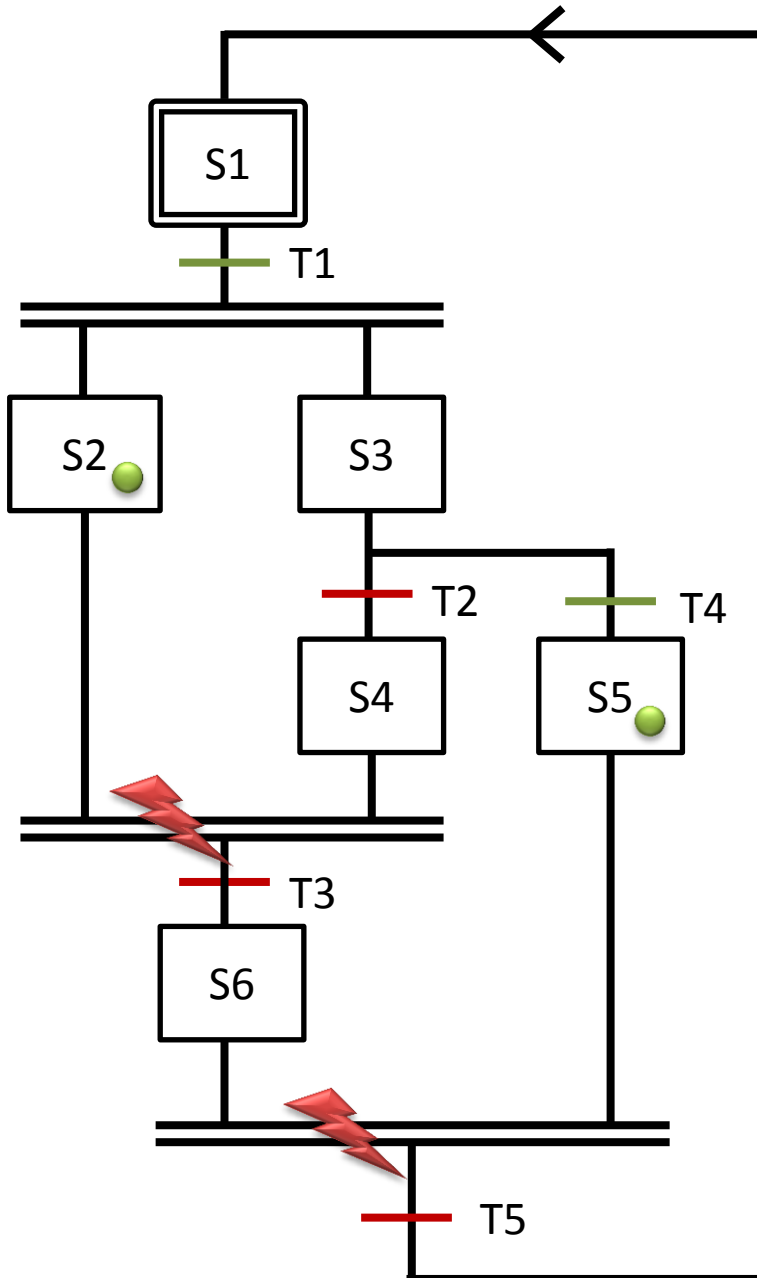
- Ha egy hálózatban az állapotok aktiválása nem kontrollált módon történik (lehetséges több token jelenléte párhuzamos szekvenciákon kívül), akkor a hálózat nem biztonságos (*unsafe*)
- A nem biztonságos hálózatok fordításkor hibát okoznak

# Nem biztonságos hálózatok



- Ha egy hálózatban az állapotok aktiválása nem kontrollált módon történik (lehetséges több token jelenléte párhuzamos szekvenciákon kívül), akkor a hálózat nem biztonságos (unsafe)
- A nem biztonságos hálózatok fordításkor hibát okoznak

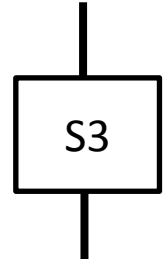
# Nem elérhető hálózatok



- A holtpontot tartalmazó hálózatok nem elérhetők (unreachable)
- A nem elérhető hálózatok hibát okoznak fordítás során

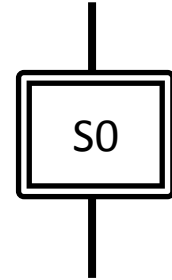
# Lépések

- Téglalap és azonosító: `StepName`
- Lépés flag: `StepName.X`
  - Logikai változó, értéke igaz, ha az adott lépés aktív
- Lépésidő: `StepName.T`
  - Időtartam-változó, értéke a lépés aktiválása óta eltelt idő
  - Értéke a lépés deaktiválásakor befagyasztódik, aktiválásakor  $t_{\#0s}$ -ről indul újra
- A lépés-flag és a lépésidő csak olvashatók



# Kezdeti lépés

- Jelölés: kettős körvonal
- Tetszőleges azonosító
- A hozzá tartozó lépés-flag kezdeti értéke TRUE
- Minden hálózat egy és csakis egy kezdeti lépést tartalmaz



# Lépések szöveges megadása

- **Lépés**

```
STEP StepName  
    (* step body *)  
END_STEP
```

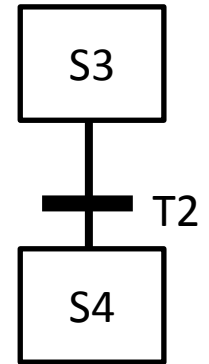
- **Kezdeti lépés**

```
INITIAL_STEP StepName  
    (* step body *)  
END_STEP
```

- A szöveges megadás lehetősége a szabványban szerepel, de a fejlesztői környezetek általában nem támogatják

# Átmenetek

- Vízszintes vonal a lépéseket összekötő függőleges élen
- Minden átmenethez egy és csakis egy feltétel tartozik
  - Logikai értékű kifejezés
  - Az átmenet akkor tüzel, ha igazra értékelődik ki
  - Feltétel nélküli átmenetek konstans TRUE feltétellel valósíthatók meg



# Átmenetek szöveges megadása

```
TRANSITION TranName FROM Step1 TO  
Step2  
    (*body *)  
END_TRANSITION
```

- A szabványban szerepel, de a fejlesztői környezetek általában nem támogatják



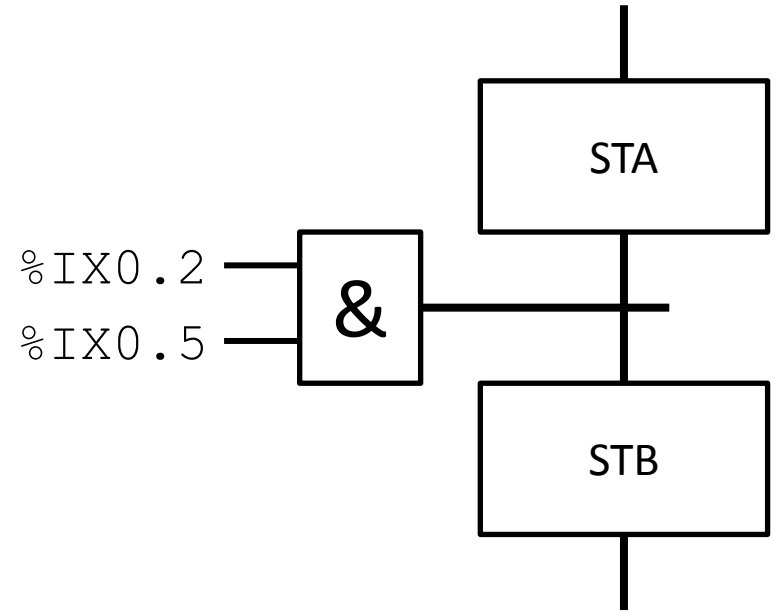
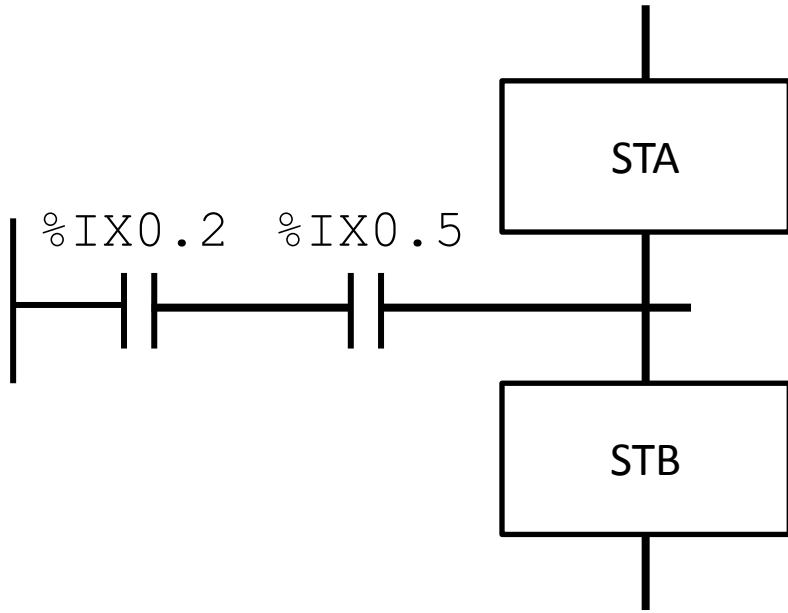
# Átmenet-feltételek megadása

- Közvetlen megadás ST nyelven

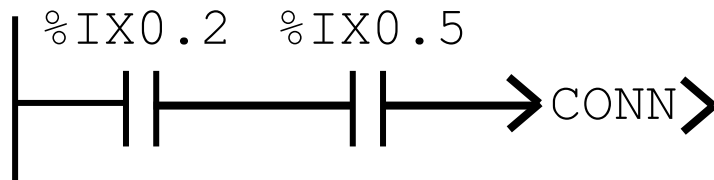


# Átmenet-feltételek megadása

- Közvetlen megadás LD vagy FBD nyelven

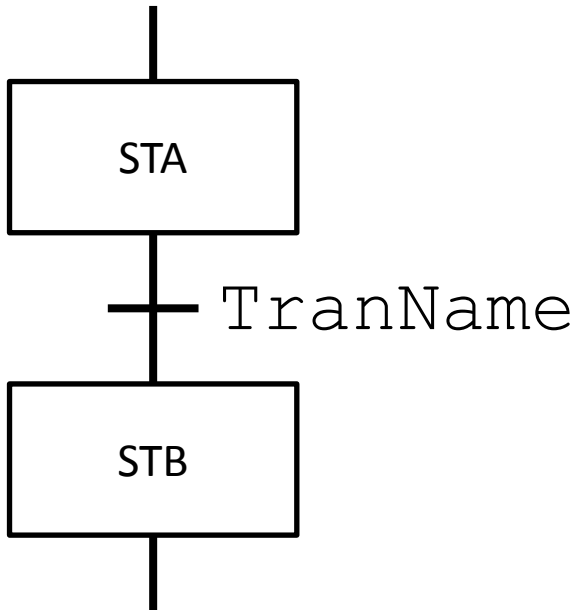


- Az összekötők használata megengedett



# Átmenet-feltételek megadása

- Közvetett módon az átmenet nevével



```
TRANSITION TranName FROM STA TO STB:  
(* LD, IL, FB, ST *)  
END_TRANSITION
```

# Átmenet-feltétel törzse

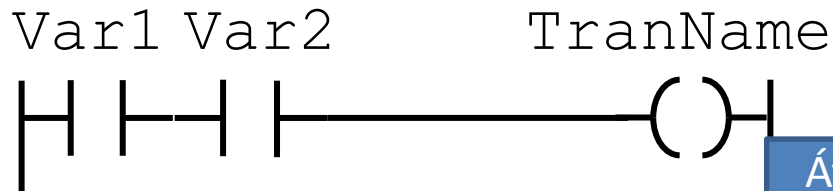
- ST: hozzárendelés egy kifejezéshez (bal oldalon hiányzik a változó)

```
:=Var1 & Var2;
```

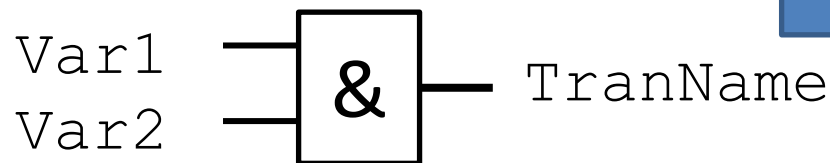
- IL: A feltétel értéke az akkumulátor értéke az utolsó művelet után

```
LD Var1  
AND Var2
```

- LD



- FBD



Átmenet névvel  
megegyező  
kimenet

# Akciók (actions)

- Minden lépéshez nulla vagy több akció rendelhető
- Logikai akció: logikai változó, amit az akció állít be
- Nem logikai akció:
  - IL utasítások
  - ST műveletek
  - LD hálózatok
  - FBD hálózatok
  - Egy másik SFC

# Nem-logikai akciók deklarációja

- LD, FBD, SFC: grafikus deklaráció  
(implementációfüggő, általában a POU-kkal  
megegyező módon)
- ST, IL: ACTION kulcsszó

```
ACTION MyAction:  
    %Q0.1 := %IX0.0 & Step8.X;  
END_ACTION
```

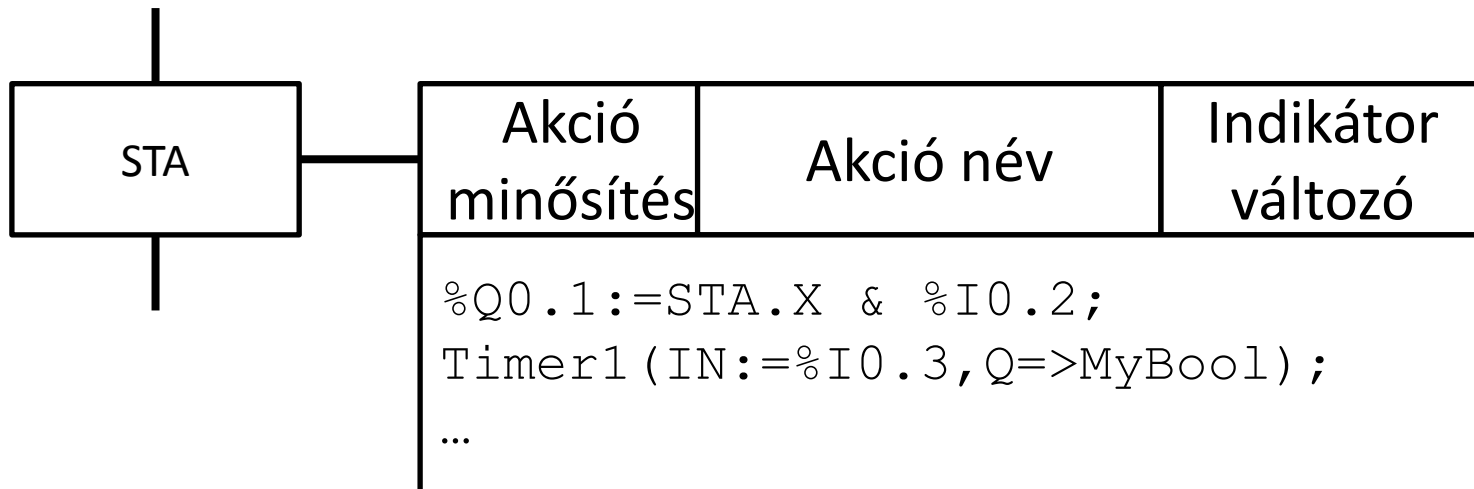
# Lépések és akciók hozzárendelése

- Grafikusan: a lépéshez kapcsolt akcióblokk(ok)al



# Közvetlen akciódefiníció

- Logikai akció: ha létezik az akcióéval megegyező nevű VAR vagy VAR\_OUT típusú változó, akkor az lesz a logikai akció
- Műveletek vagy hálózatok: az akció törzse az akcióblokkon belül is megadható (ekkor az akciónév más akcióblokkokban nem használható)





# Lépések és akciók hozzárendelése

- Szövegesen: a STEP blokkokban

```
STEP StepName  
    ActionName (Qual, QualParameter, IndVar) ;  
END_STEP
```

- A fejlesztői környezettől függő módon
  - Általában akcióblokk
  - Az akció törzse külön ablakban szerkeszthető

# Az akcióblokk szerkezete



Megadja, hogyan hajtódik végre az adott akció a lépés aktiválása után (*action qualifier*)

Azonosítja az akciót (logikai VAR vagy VAR\_OUT változó, Action azonosító)

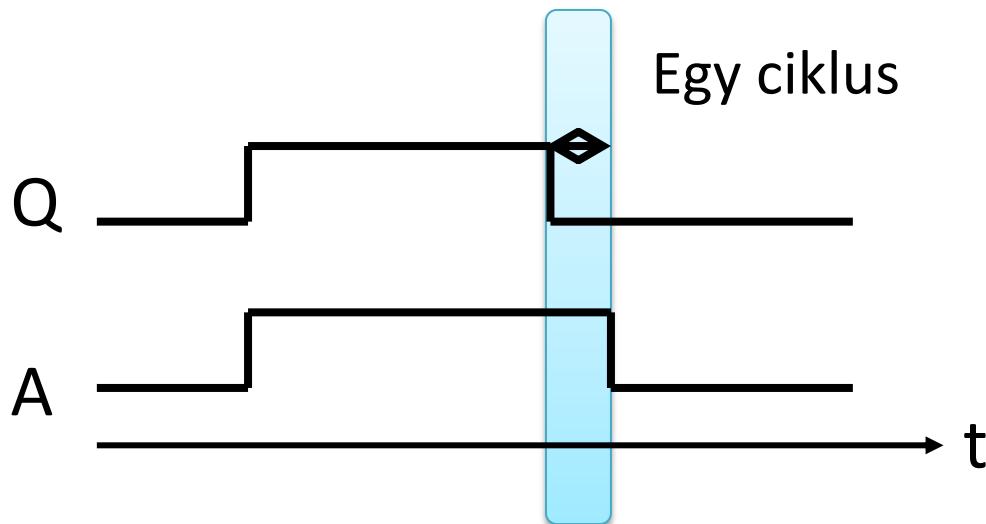
Opcionális logikai változó, amit az akció állít be, hogy sikeres befejezést, hibát stb. jelezen (a gyakorlatban nem használt)

# Akcióvezérlés

- A felhasználó előtt rejtett blokk
- Az akcióminősítéstől függően állítja a flageket:
  - Akció-flag (Q) – nem logikai akciók esetén  
`ActionName.Q` néven érhető el
  - Aktivitás-flag (A) – csak nem logikai akciók esetén
- Logikai akció: a változót az akció-flag értékére állítjuk
- Nem logikai akció: ciklikusan fut, amíg az aktivitás-flag TRUE értékű

# Aktivitás-flag

- Az akció-flaggel együtt állítódik be, annak lefutó éle után még egy ciklus idejéig aktív
- Az akció adott végrehajtási ciklusa az utolsó, ha  $\text{Action.Q} = \text{FALSE}) \& (\text{Action.A} = \text{TRUE})$



# Aktivitás-flag

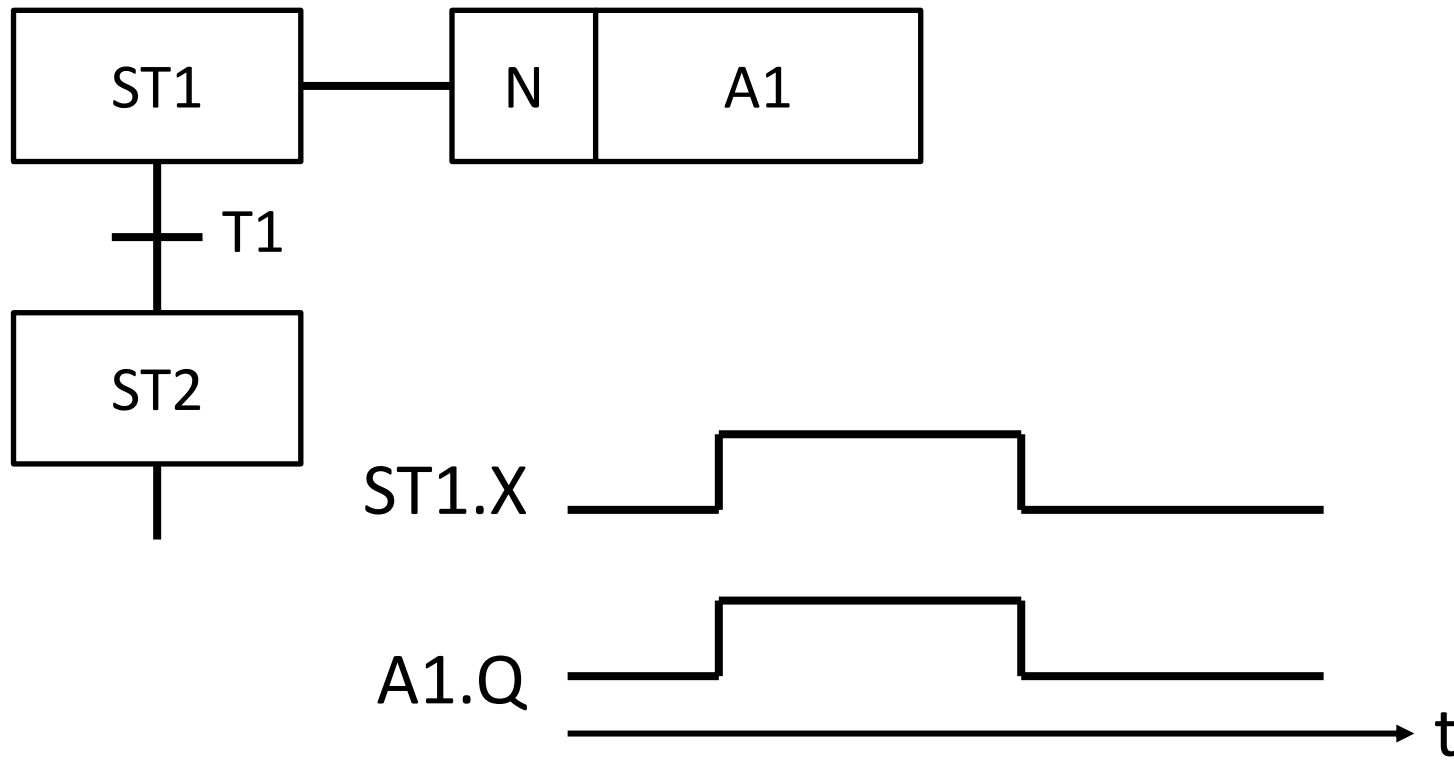
- A fejlesztői környezetek többségében nem érhető el közvetlenül
- Hasonló konstrukciók
  - Exit akció – egyszer, a lépés deaktiválásakor fut le
  - Az akcióból elérhető Last Scan bit – az utolsó végrehajtáskor aktív

# Akcióminősítések

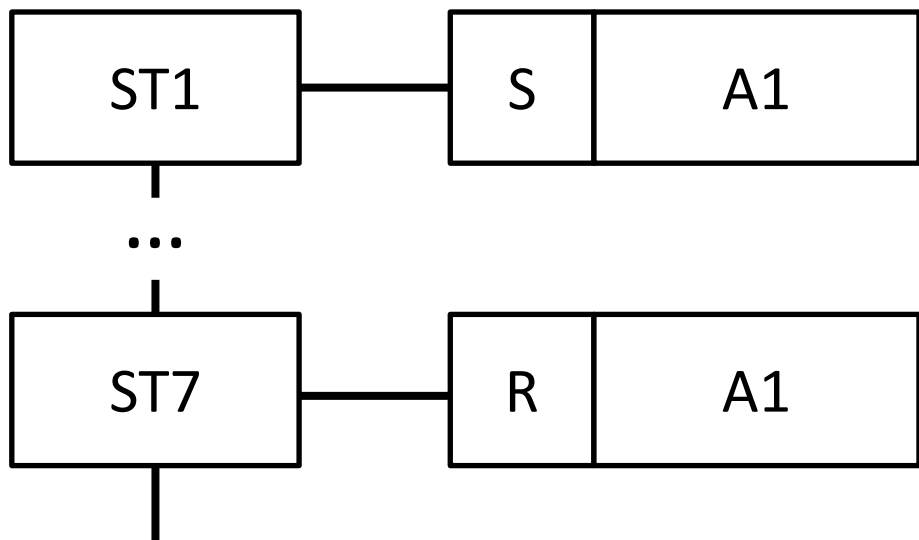
Minősítés	Értelmezés
None / N	Nem tárolt ( <b>N</b> on-stored, null qualifier)
R	Reset (Overriding <b>R</b> eset, tárolt akcióra)
S	Tárolt ( <b>S</b> et, <b>S</b> tored)
L	Időben korlátozott (Time <b>L</b> imited)
D	Időben késleltetett (Time <b>D</b> elayed)
P	Impulzus ( <b>P</b> ulse)
SD	Tárolt és késleltetett ( <b>S</b> tored and <b>D</b> elayed)
DS	Késleltetett és tárolt ( <b>D</b> elayed and <b>S</b> tored)
SL	Tárolt és korlátozott ( <b>S</b> tored and <b>L</b> imited)
<i>P1</i>	<i>Felfutó él érzékeny (<b>P</b>ulse (rising edge))</i>
<i>P0</i>	<i>Lefutó él érzékeny (<b>P</b>ulse (falling edge))</i>

# Nem tárolt akció

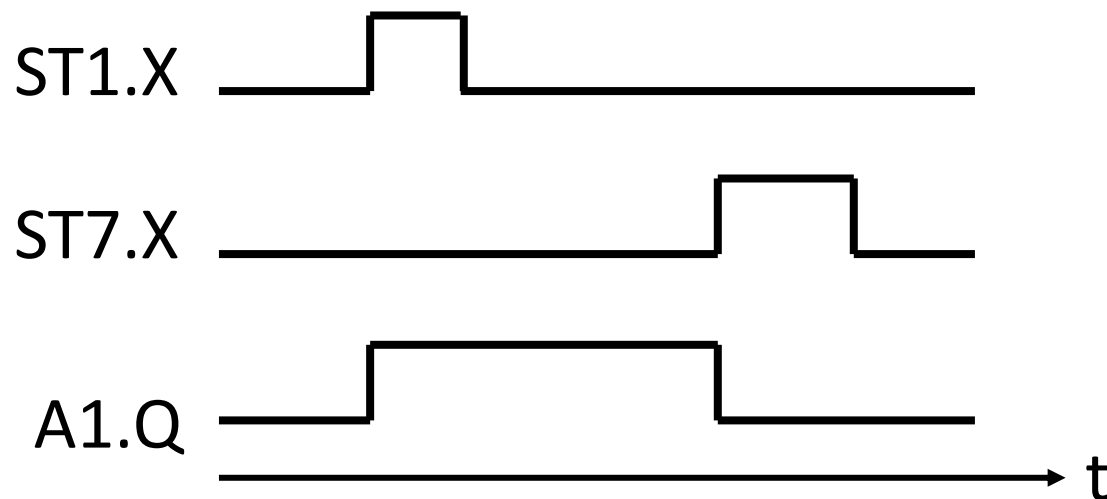
- A Q akció-flag a lépés-flag másolata



# Tárolt akció



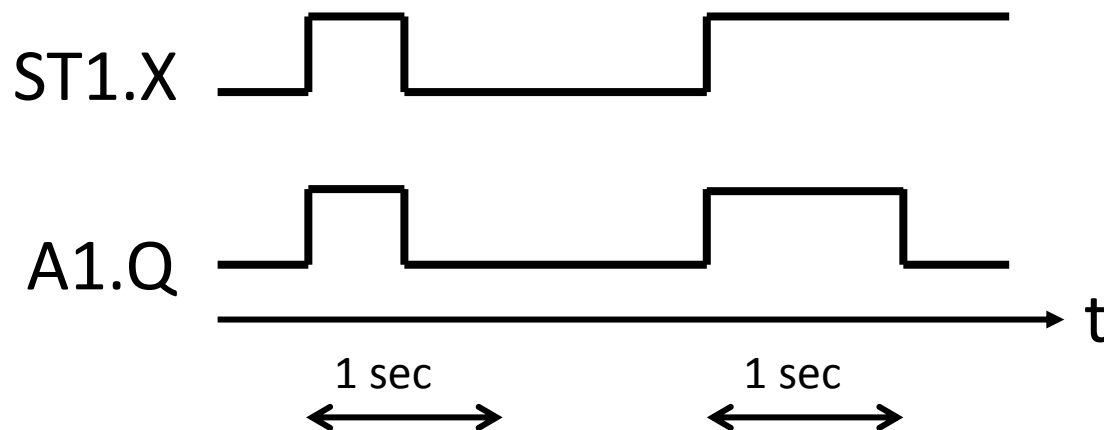
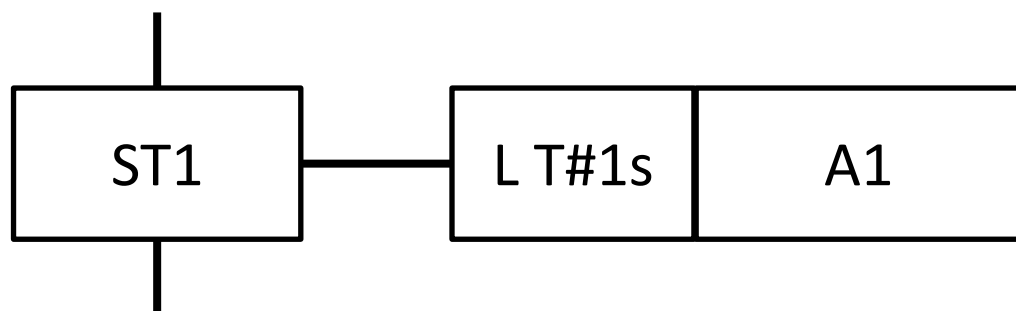
- A Reset művelet magasabb prioritású





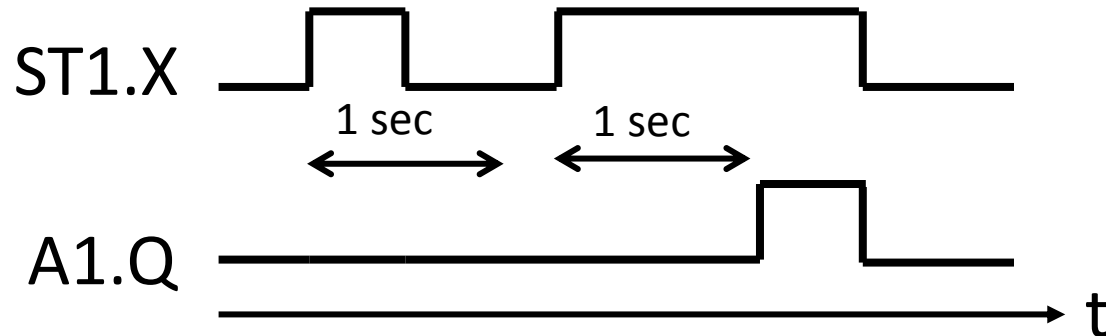
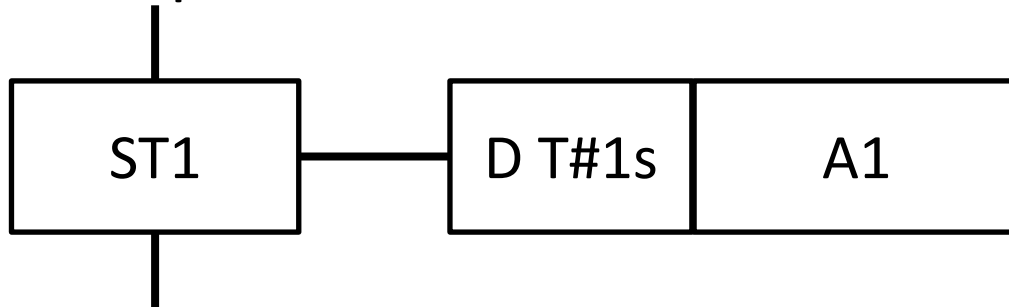
# Időben korlátozott akció

- A Q akció-flag a lépés aktiválásával állítódik be
- Annak deaktiválásáig, de legfeljebb a megadott ideig aktív



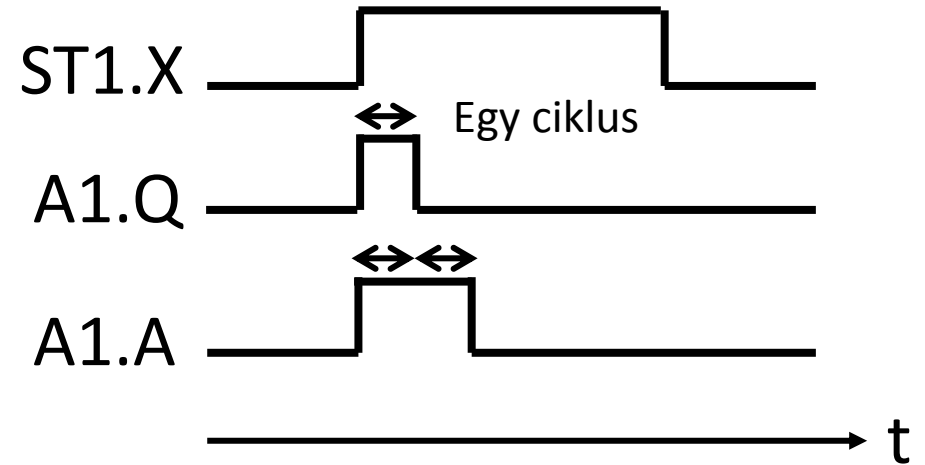
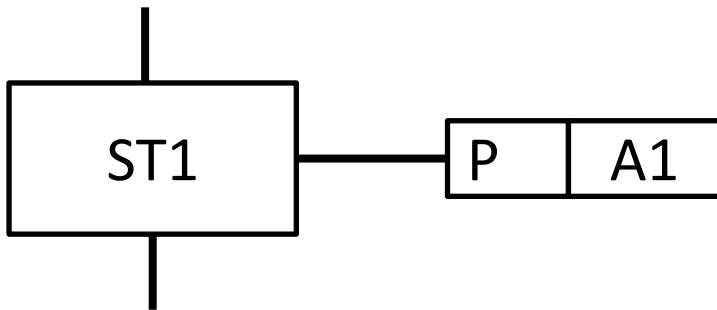
# Időben késleltetett akció

- A Q akció-flag a lépés aktiválása után megadott idővel állítódik be, ha a lépés akkor még aktív
- A lépés deaktiválásakor törlődik



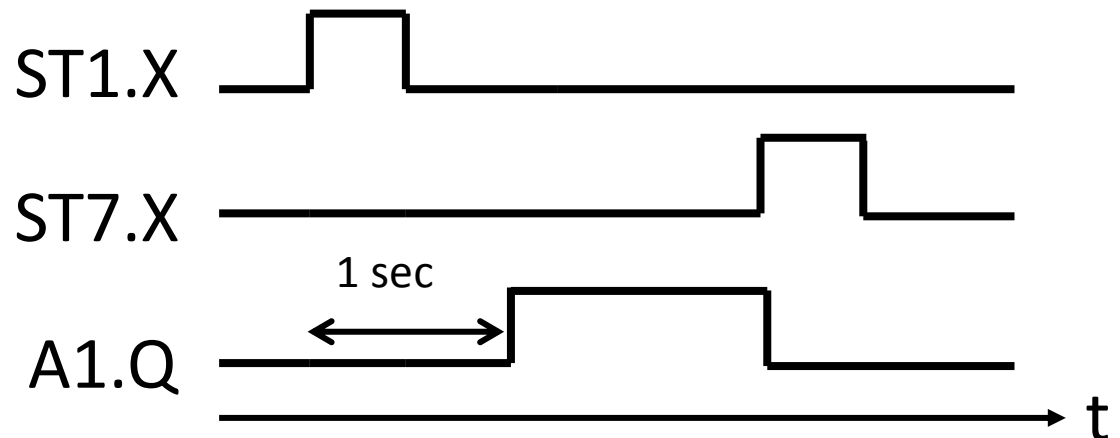
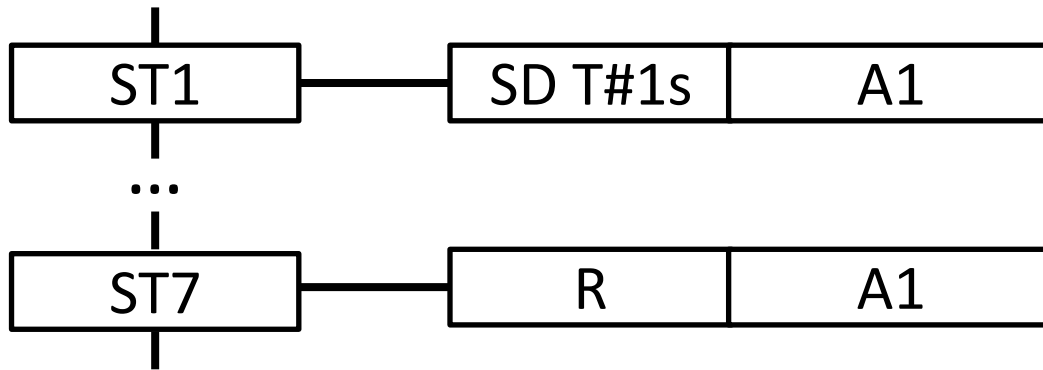
# Impulzus-akció

- Az akció-flag a lépés aktiválása után egyetlen ciklus idejéig aktív
- Egyes fejlesztői környezetekben külön Entry Action definiálható



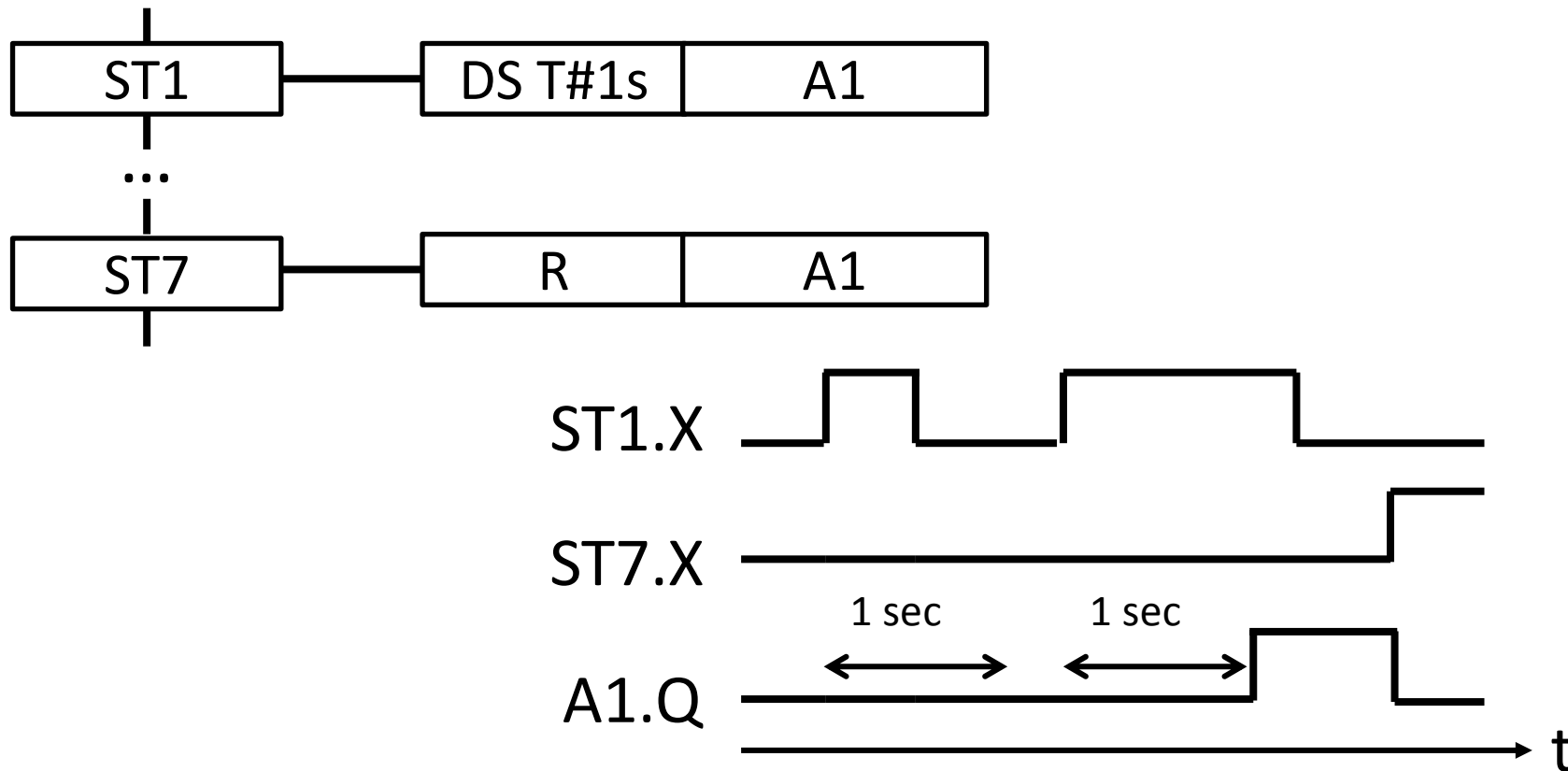
# Tárolt és késleltetett akció

- A Q akció-flag a lépés aktiválása után megadott idővel állítódik be, akkor is, ha a lépés már nem aktív
- Csak Reset-akció törli



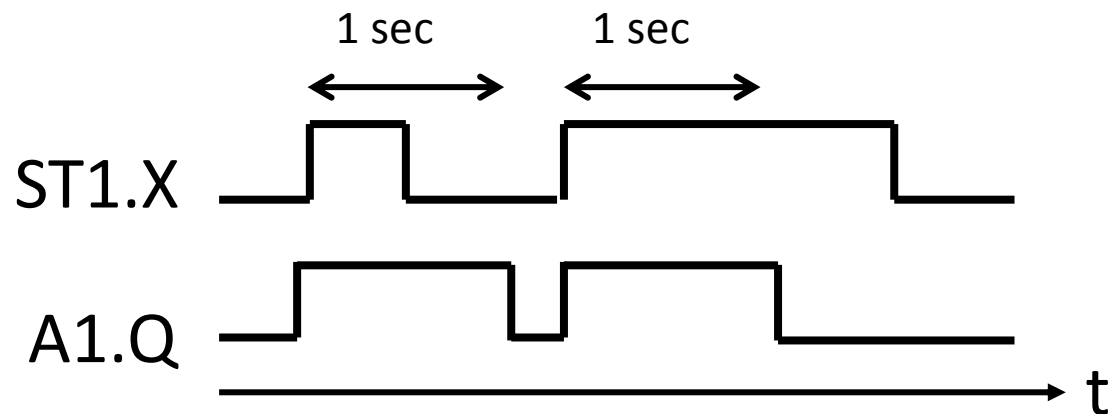
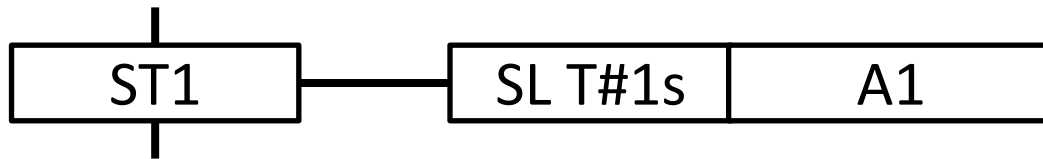
# Késleltetett és tárolt akció

- A Q akció-flag a lépés aktiválása után megadott idővel állítódik be, amennyiben a lépés még aktív
- Csak Reset-akció törli, az állapot deaktiválása nem



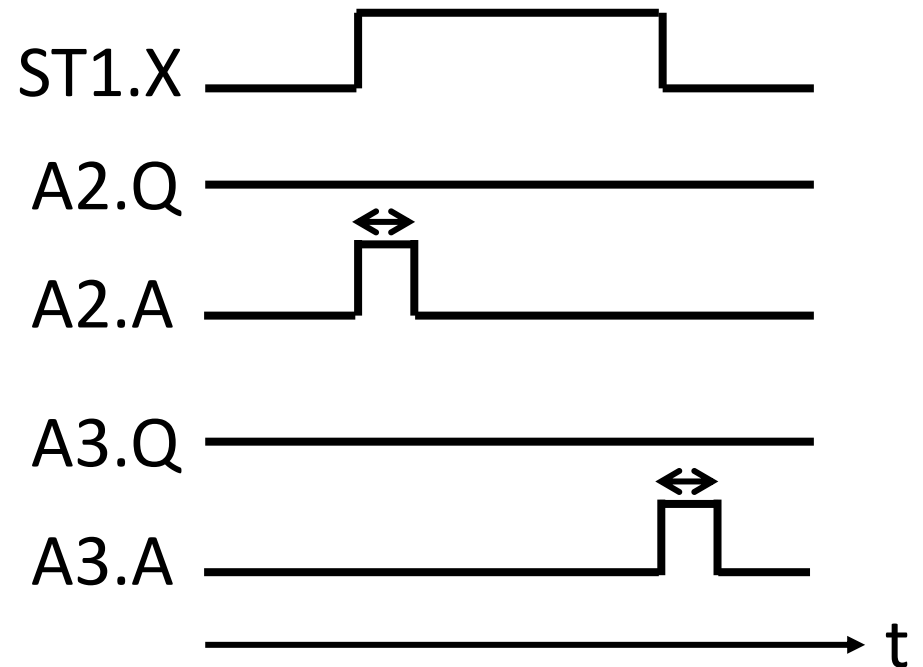
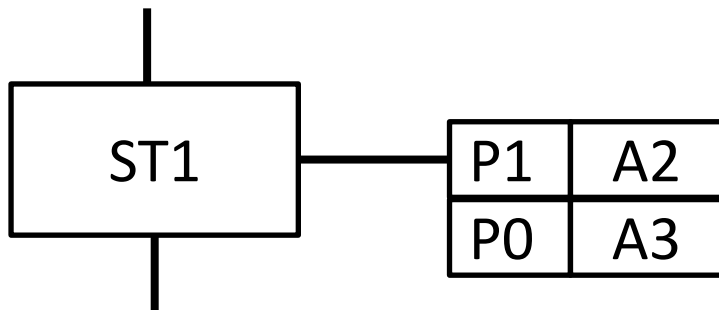
# Tárolt és korlátozott akció

- A Q akció-flag a lépés aktiválásakor állítódik be
- A megadott idő után törlődik a lépés aktivitásától függetlenül



# P1 és P0 impulzus-akciók

- A szabványban definiáltak, de a fejlesztői környezetek általában nem implementálják (helyettük más megoldások)
- A P1 és P0 akcióminősítések logikai akciókra nem értelmezettek (hatástalanok)
- Csak az aktivitás flaget állítják, az akció flaget nem



# Időzített minősítések

- Egy akció csak egy időzített minősítéshez kapcsolható

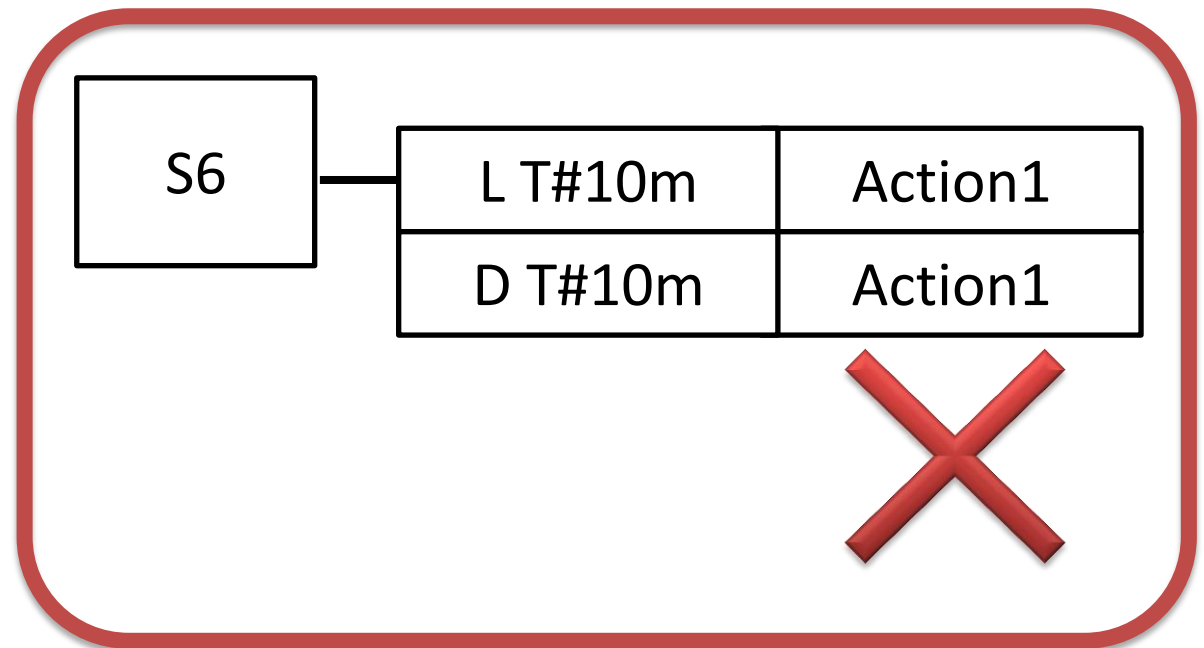
– L

– D

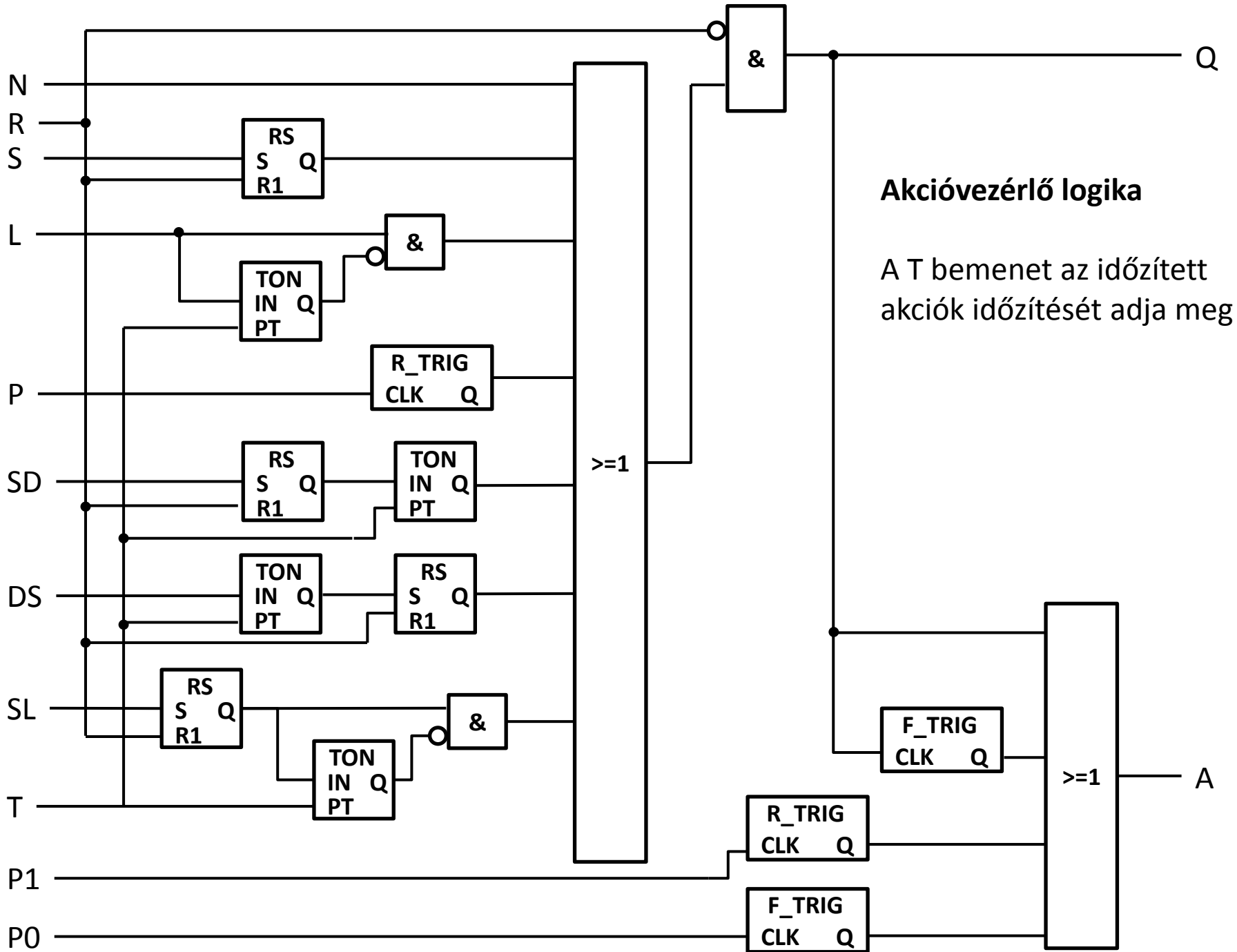
– SD

– DS

– SL

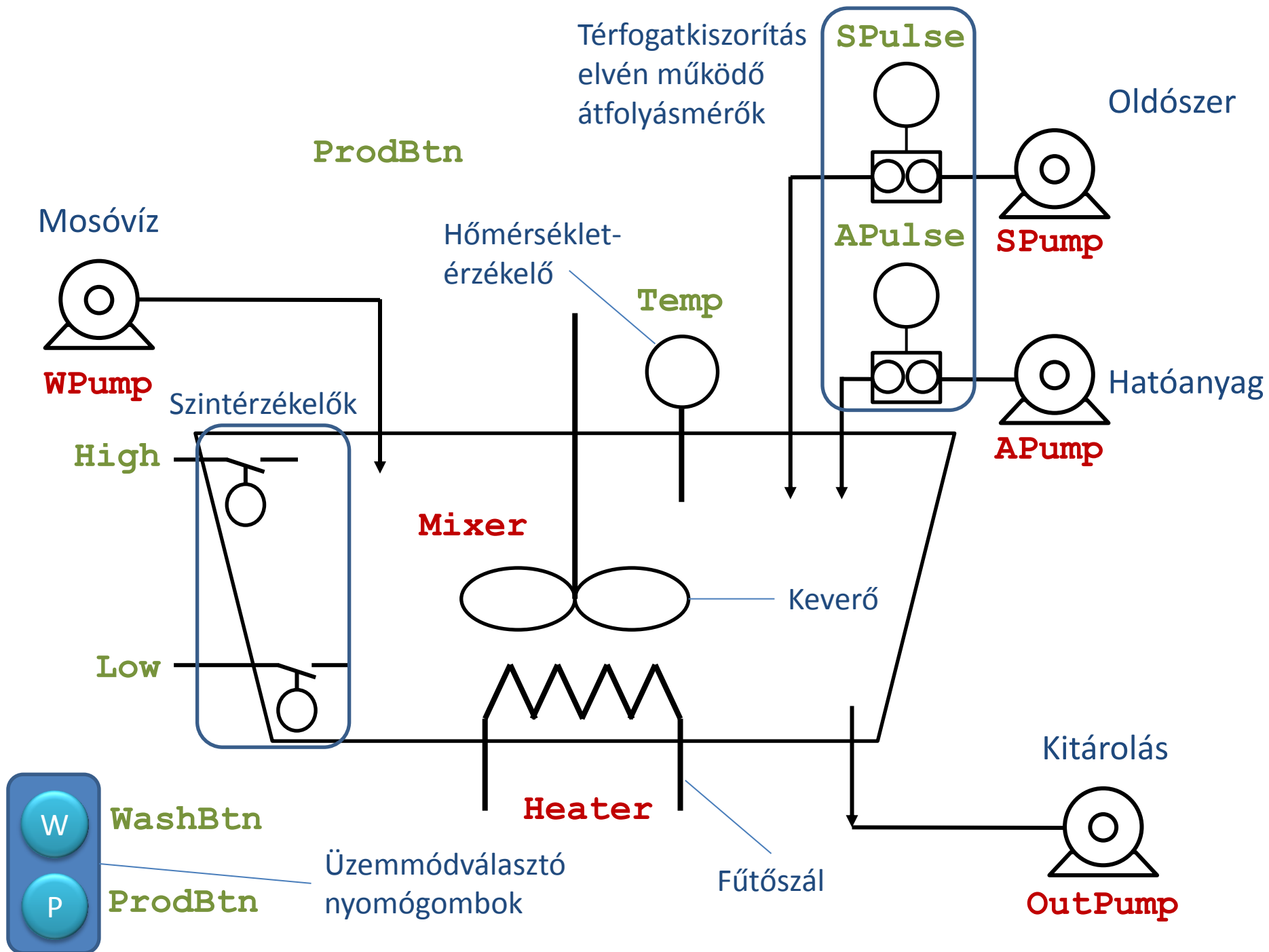






# Példa

- Egy keverővel is ellátott autokláv a következő üzemmódokban működik:
  - Mosás: a tartályt megtöltjük vízzel, majd 10 percen át fűtjük. Ezután bekapcsoljuk a keverőt, és addig működtetjük, amíg a víz hőmérséklete 30 fok alá nem süllyed, majd kiszivattyúzzuk a vizet.
  - Termelés: a tartályba a megadott mennyiségű hatóanyagot (7 egység) és oldószert (30 egység) szivattyúzzuk, majd az elegyet 10 percig fűtjük. Ezután bekapcsoljuk a keverőt, és addig működtetjük, amíg az oldat hőmérséklete 30 fok alá nem süllyed, majd kiszivattyúzzuk azt.



## Bemenetek

Változó	Típus	Értelmezés
APulse	BOOL	Hatóanyag-átfolyásmérő: felfutó éle egy egység betárolását jelzi
SPulse	BOOL	Oldószer-átfolyásmérő: felfutó éle egy egység betárolását jelzi
High	BOOL	Szintérzékelő – teli szint (0: szint alatta, 1: szint felette)
Low	BOOL	Szintérzékelő – üres szint (0: szint alatta, 1: szint felette)
WashBtn	BOOL	Mosás üzemmódválasztó nyomógomb
ProdBtn	BOOL	Termelés üzemmódválasztó nyomógomb
Temp	USINT	Folyadék hőmérséklete [°C]

## Kimenetek

Változó	Típus	Értelmezés
WPump	BOOL	Mosóvíz szivattyú (0: ki, 1: be)
SPump	BOOL	Oldószer szivattyú (0: ki, 1: be)
APump	BOOL	Hatóanyag szivattyú (0: ki, 1: be)
OutPump	BOOL	Kitároló szivattyú – ürítés (0: ki, 1: be)
Mixer	BOOL	Keverő (0: ki, 1: be)
Heater	BOOL	Fűtés (0: ki, 1: be)

# Betárolt mennyiség számlálása

- APulse és SPulse felfutó éleit számoljuk
- A betárolt mennyiséget az AIn és SIn változókba töltjük
- A számlálókat kitároláskor nullázzuk

```
ACTION CountAPulse:
```

```
    CntA(CU:=APulse, R:=NOT LOW, CV=>AIn);
```

```
END_ACTION
```

```
ACTION CountSPulse:
```

```
    CntS(CU:=SPulse, R:=NOT LOW, CV=>SIn);
```

```
END_ACTION
```

PROGRAM MAIN

VAR\_INPUT

APulse AT %IX0.0: BOOL;  
Spulse AT %IX0.1: BOOL;  
Low AT %IX0.2: BOOL;  
High AT %IX0.3: BOOL;  
WashBtn AT %IX0.4: BOOL;  
ProdBtn AT %IX0.5: BOOL;  
Temp AT %IB0.0: USINT;

END\_VAR

VAR\_OUTPUT

APump AT %QX0.0: BOOL;  
SPump AT %QX0.1: BOOL;  
WPump AT %QX0.2: BOOL;  
OutPump AT %QX0.3: BOOL;  
Heater AT %QX0.4: BOOL;  
Mixer AT %QX0.5: BOOL;

END\_VAR

VAR

Ain, Sin: INT;  
CntS, CntR: CTU;

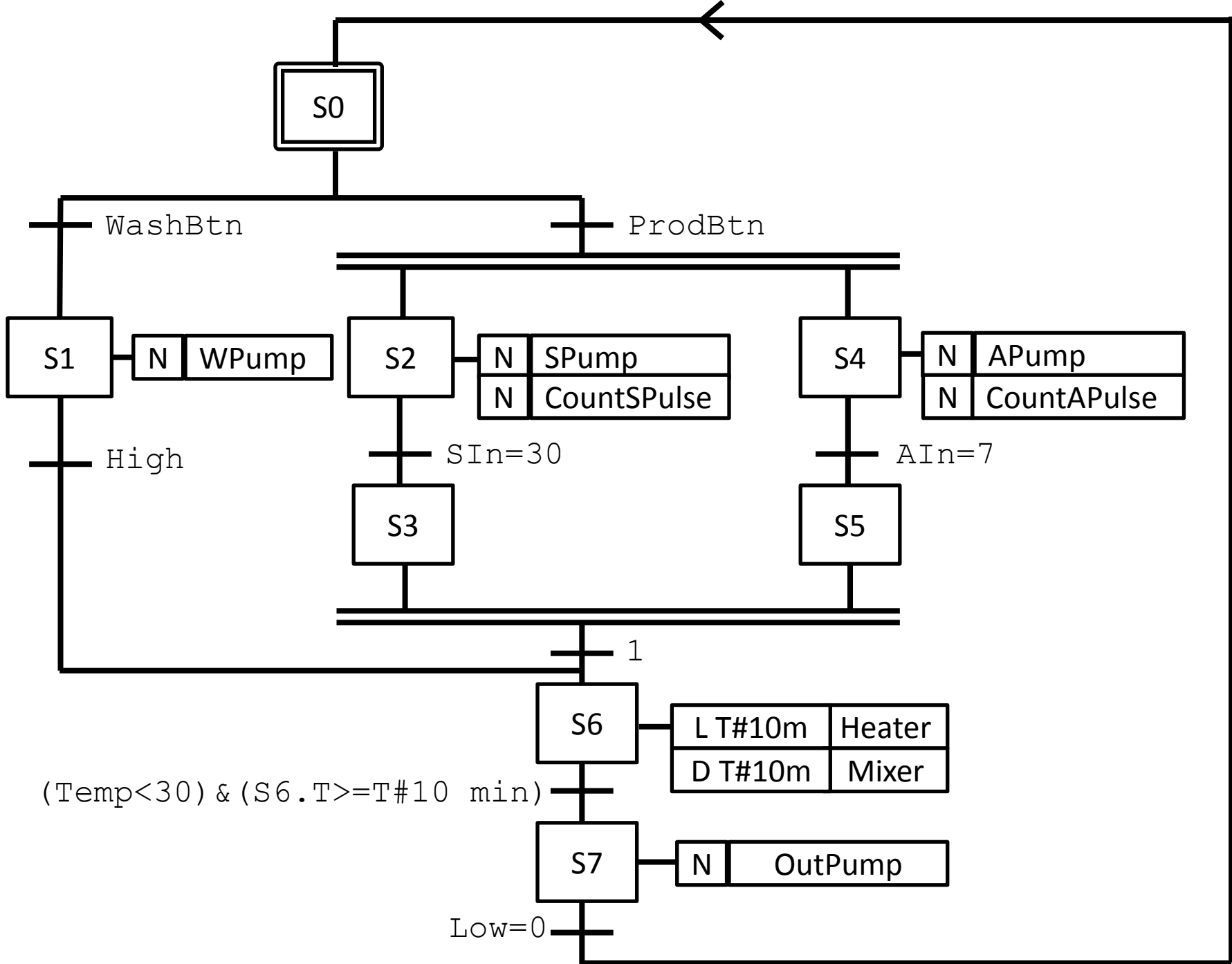
END\_VAR

ACTION CountAPulse:  
CntA( CU:=APulse,  
R:=NOT LOW,  
CV=>Ain  
);

END\_ACTION

ACTION CountSPulse:  
CntS( CU:=SPulse,  
R:=NOT LOW,  
CV=>Sin  
);

END\_ACTION



# Példa- Akciók

Akció	Magyarázat
Wpump	A mosóvíz szivattyút működtető bit (logikai akció)
Spump	Az oldószer szivattyút működtető bit (logikai akció)
Apump	A hatóanyag-szivattyút működtető bit (logikai akció)
CountSPulse	Betárolt oldószer-mennyiség (SIn) mérése – átfolyásmérő impulzusait számláló akció (nem logikai)
CountAPulse	Betárolt hatóanyag-mennyiség (AIn) mérése – átfolyásmérő impulzusait számláló akció (nem logikai)
Heater	A fűtést működtető bit (logikai akció)
Mixer	A keverőt működtető bit (logikai akció)
OutPump	A kitároló szivattyút működtető bit (logikai akció)