Programirljivi logični krmilnik: programiranje - 2. del

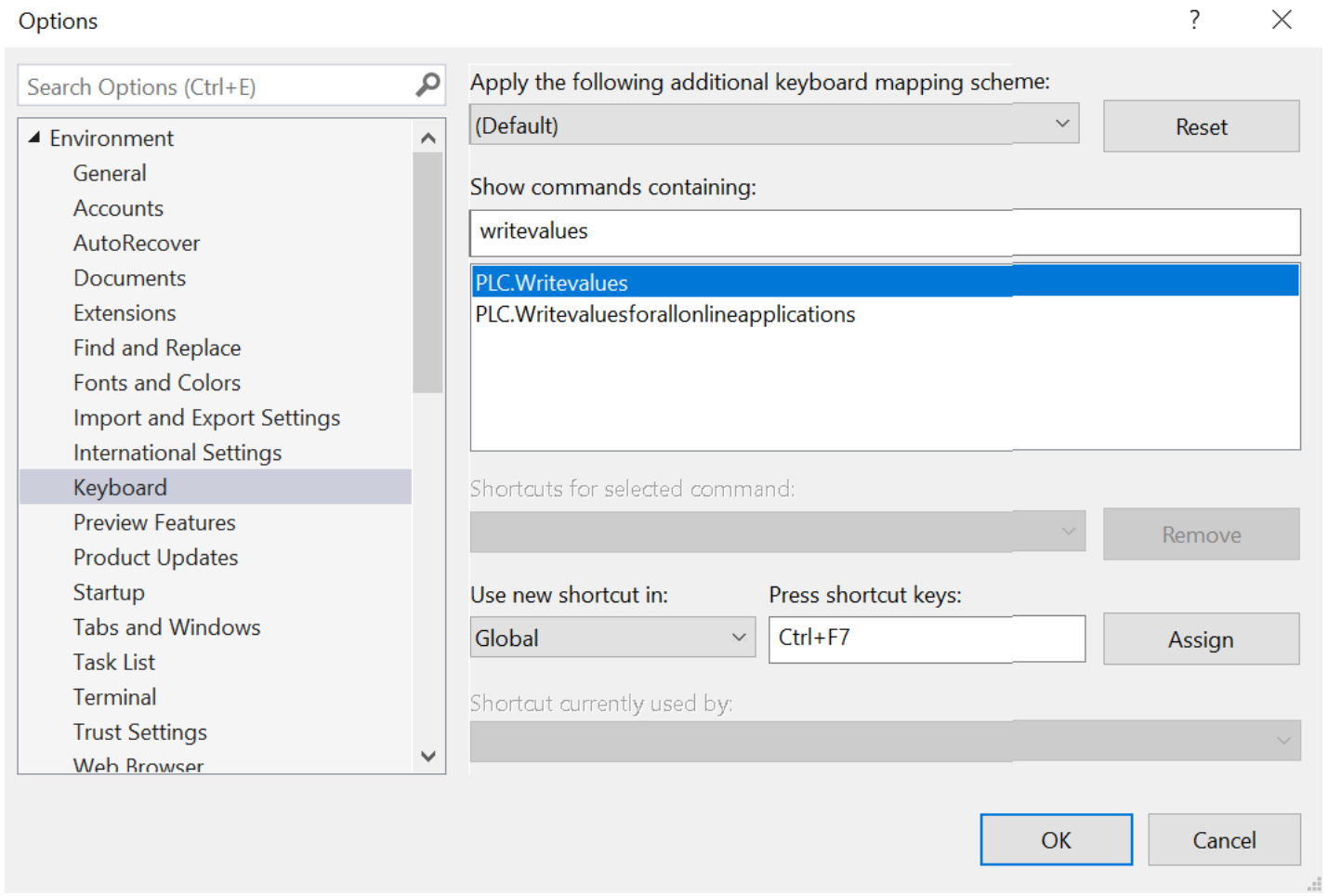
Procesna avtomatika

Uroš Lotrič, Nejc Ilc

TwinCAT – bližnjice za lažje delo

Infosys

Tools → Customize → Commands → Keyboard...



Funkcijski načrt: standard

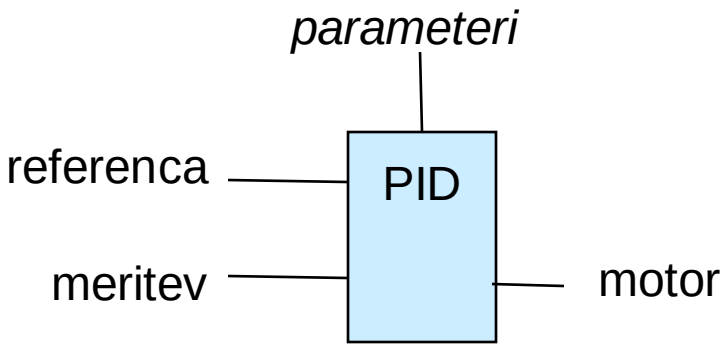
- Funkcijski načrt je eden od standardnih grafičnih jezikov
- Nazorno prikazuje medsebojno povezanost funkcij in funkcijskih blokov
- Podobni so električnim in blokovnim shemam iz analogne in digitalne tehnike
  - Vsak blok ima vhode in izhode
  - Povezave nakazujejo tok električnega toka v pravih vezjih
- Bloki običajno predstavljajo kombinatorične funkcije (odločitvena vezja), lahko pa imajo tudi spomin (sekvenčna vezja).
- Standard dovoljuje tudi povratne povezave
  - Predpisuje definiranje vrstnega reda izvajanja povratno povezanih blokov, ne pa načina

Funkcijski načrt: standard

Funkcijski blok je definiran z

- vmesnikom
  - število in tipi vhodov in izhodov
- funkcionalnostjo črne škatle (ang. black box)
  - delovanje bloka je opisano grafično, tabelarično, s formulo, opisno

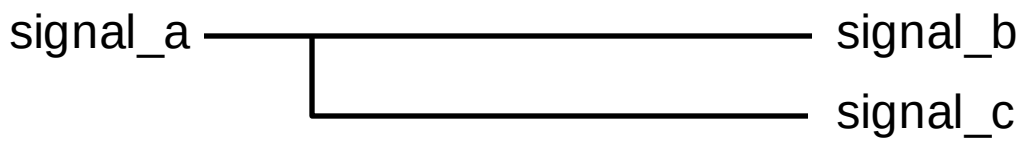
Primer:



Pravila povezovanja

- Vsak signal je povezan z natanko enim virom
- Vir, ponor in povezava morajo biti istega podatkovnega tipa

Primer:



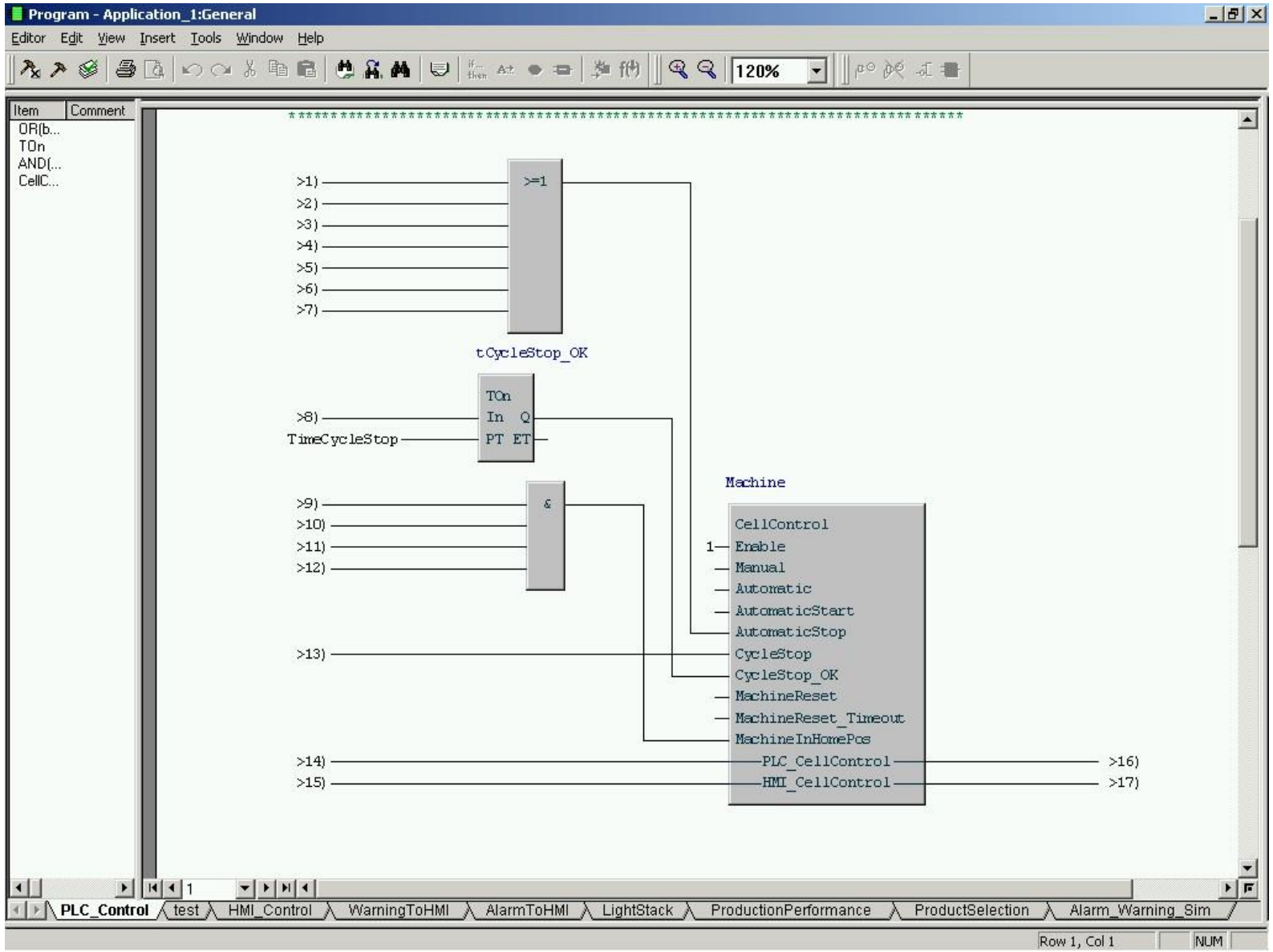
Funkcijski načrt: program

Izvajanje programa

- od zgoraj navzdol
- od leve proti desni
- izjema so povratne povezave

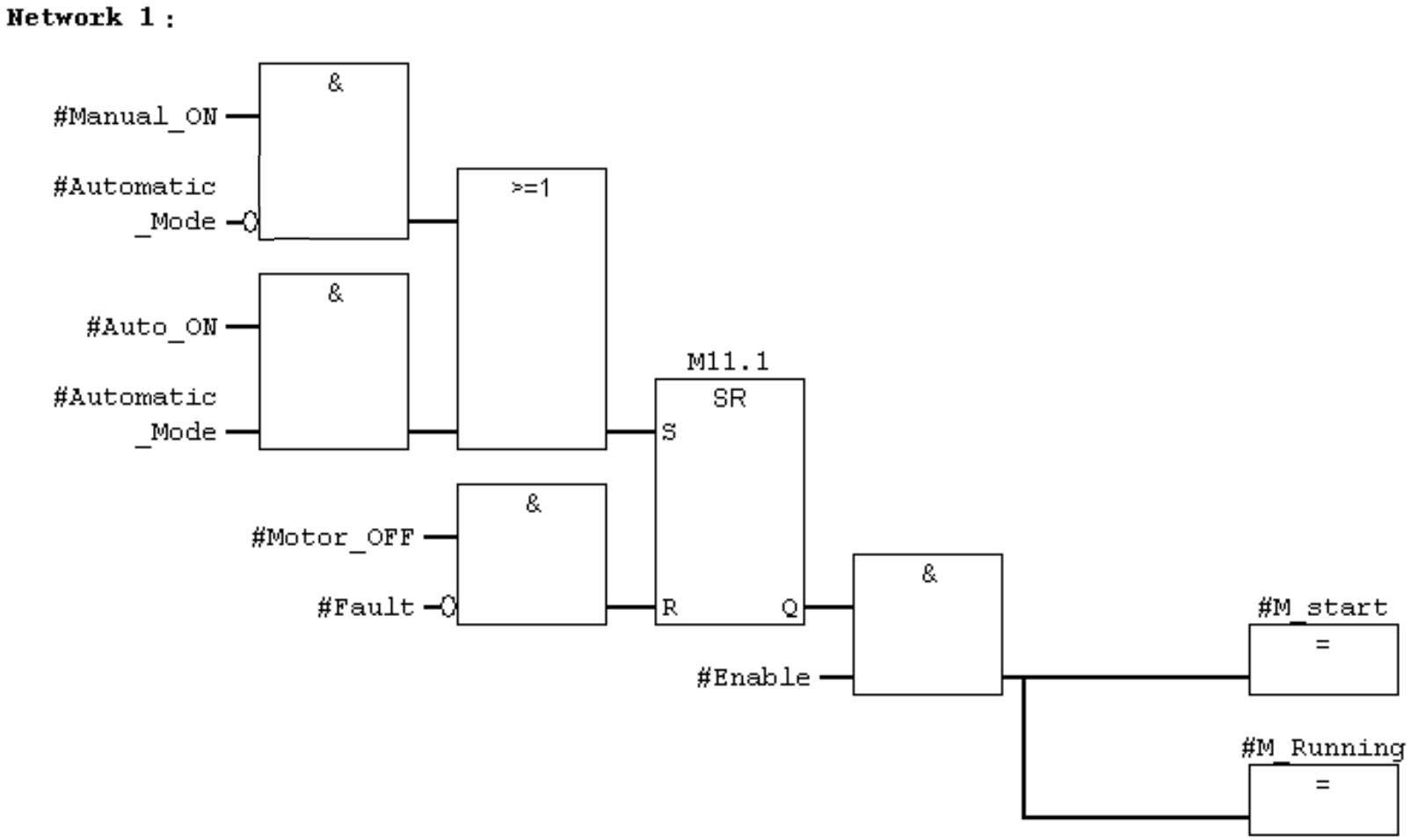
Primer

- Razvojno okolje podjetja ABB



Funkcijski načrt: program

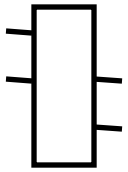
Primer:
Vodenje motorja v Siemens TIA Portal



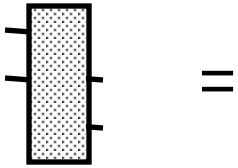
Funkcijski načrt: program

Dekompozicija problema

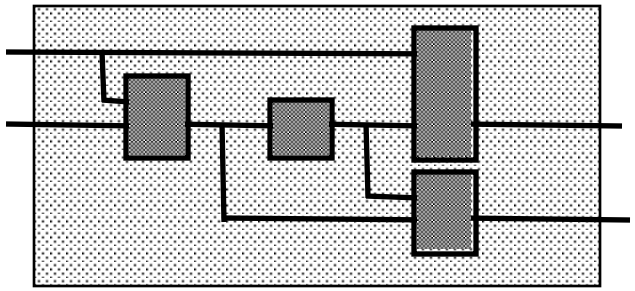
- Elementarni blok
  - Mikrokoda, zbirnik, ST, …



- Sestavljeni blok
  - Povezuje več elementarnih blokov v celoto
  - Funkcijski načrt



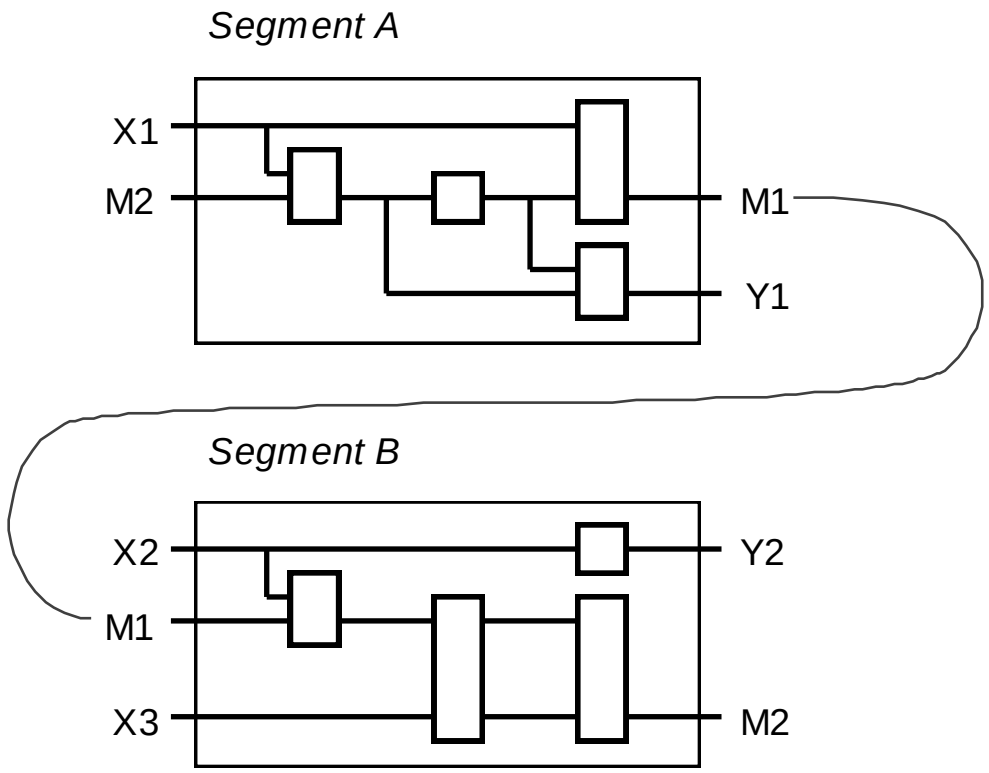
=



Funkcijski načrt: program

Segmentacija

- Zaradi večje preglednosti funkcijski načrt razdelimo v več segmentov
- Znotraj segmenta so povezave predstavljene grafično
- Segmente med seboj povezujejo spremenljivke (začasne)



Funkcijski načrt: TwinCAT

Koncept klinov (*network*) prevzet iz lestvičnih diagramov

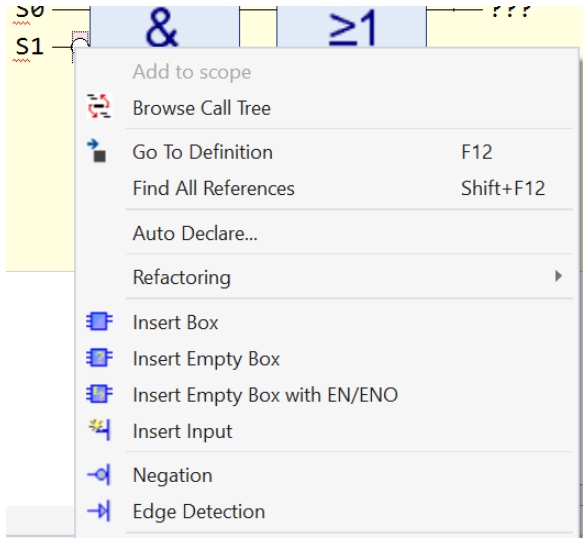
- program se izvaja sekvenčno, klin za klinom, od leve proti desni

Dobra praksa:

- Vsi vhodi naj bodo definirani, da ne pride do neželenega obnašanja.
- Vsak izhod (%Q) naj se v programu pojavi samo na enem mestu.

Primer: osnovne operacije

- negacijo dobimo preko kontekstnega menija (desni klik)



Preklop LD → FBD

- *Extensions → FBD/LD/IL→ View*

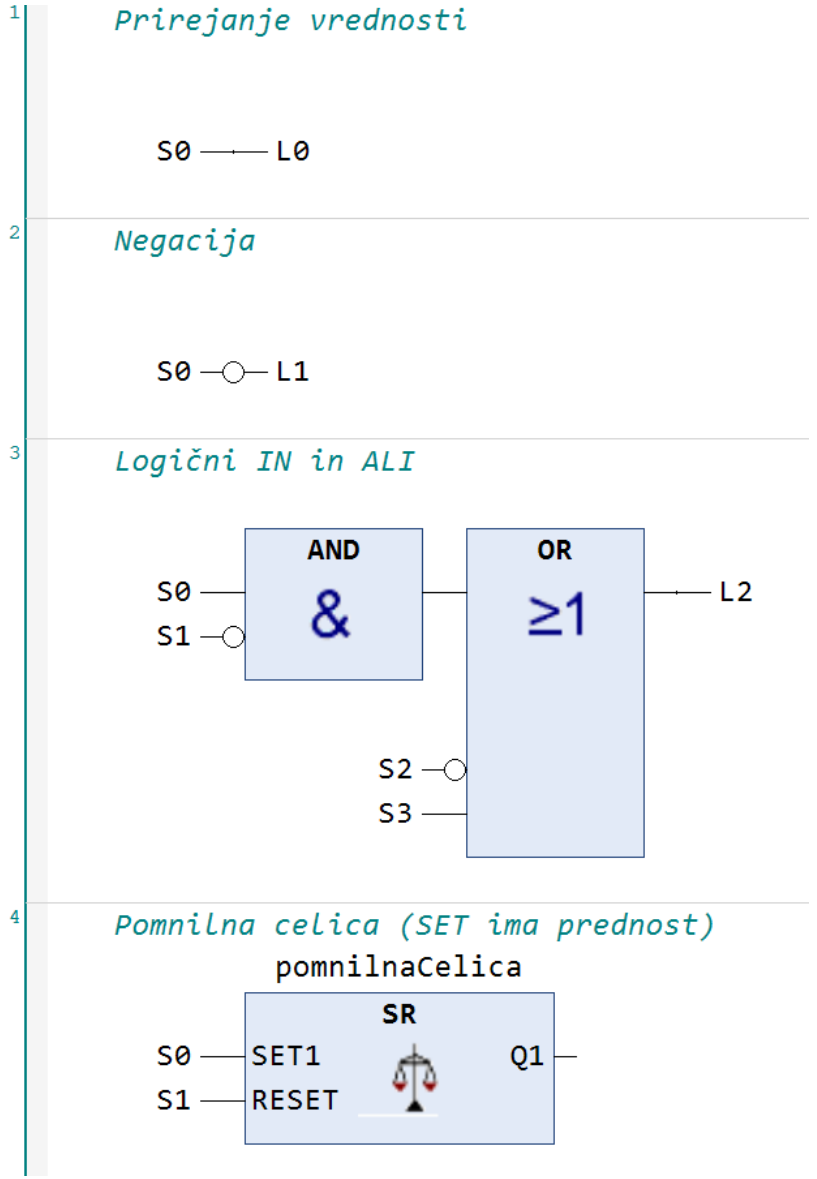


Diagram poteka: standard

SFC (*Sequential Function Chart*)

- Opisuje zaporedje operacij in interakcij med vzporednimi procesi
- Matematični temelji so v Petrijevih mrežah
- Stanja so povezana s prehodi

Žeton

- Stanje je aktivno ob prisotnosti žetona
- Žeton zapusti stanje, ko je izpolnjen pogoj za prehod
- Samo en prehod se lahko zgodi naenkrat
- Žeton je ob začetku programa v začetnih stanjih

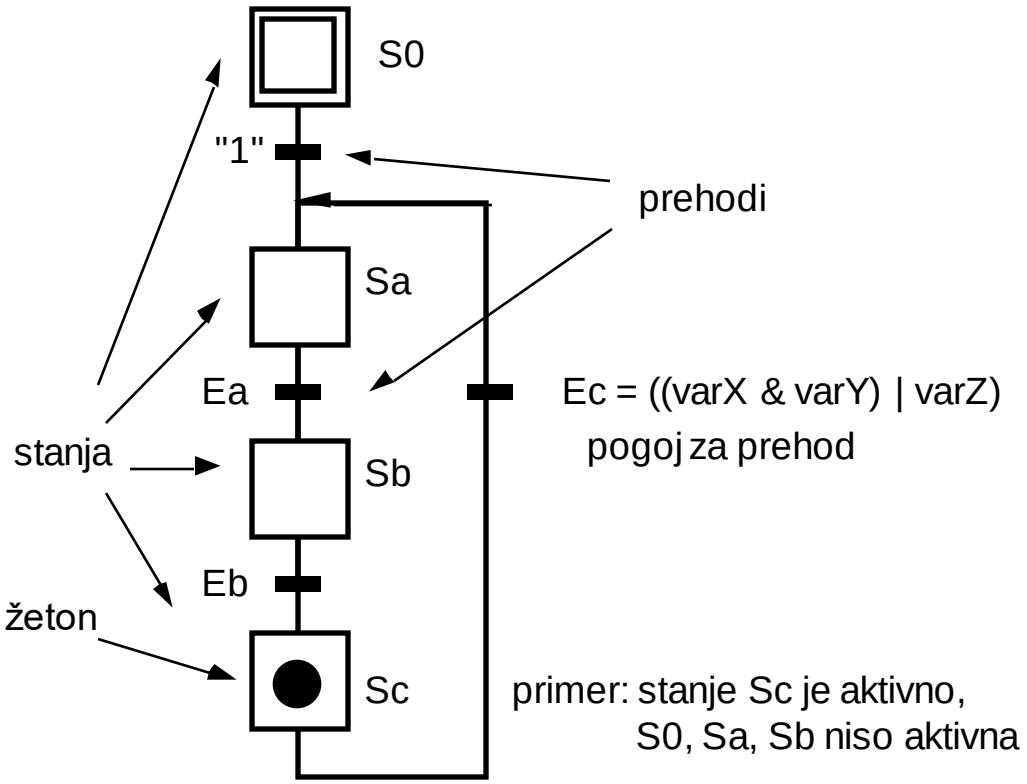


Diagram poteka: standard

Izvajanje programa

- Žeton prečka prvi aktivni prehod
  - V primeru, da sta Ea in Eb oba izpolnjena se upošteva nastavljene prioritete ali naključje
- Ko je pogoj Ee izpolnjen, se žeton razdeli na obe povezani stanji
- Ko so prisotni vsi razdeljeni žetoni in je pogoj Ef izpolnjen, pot nadaljuje en sam žeton

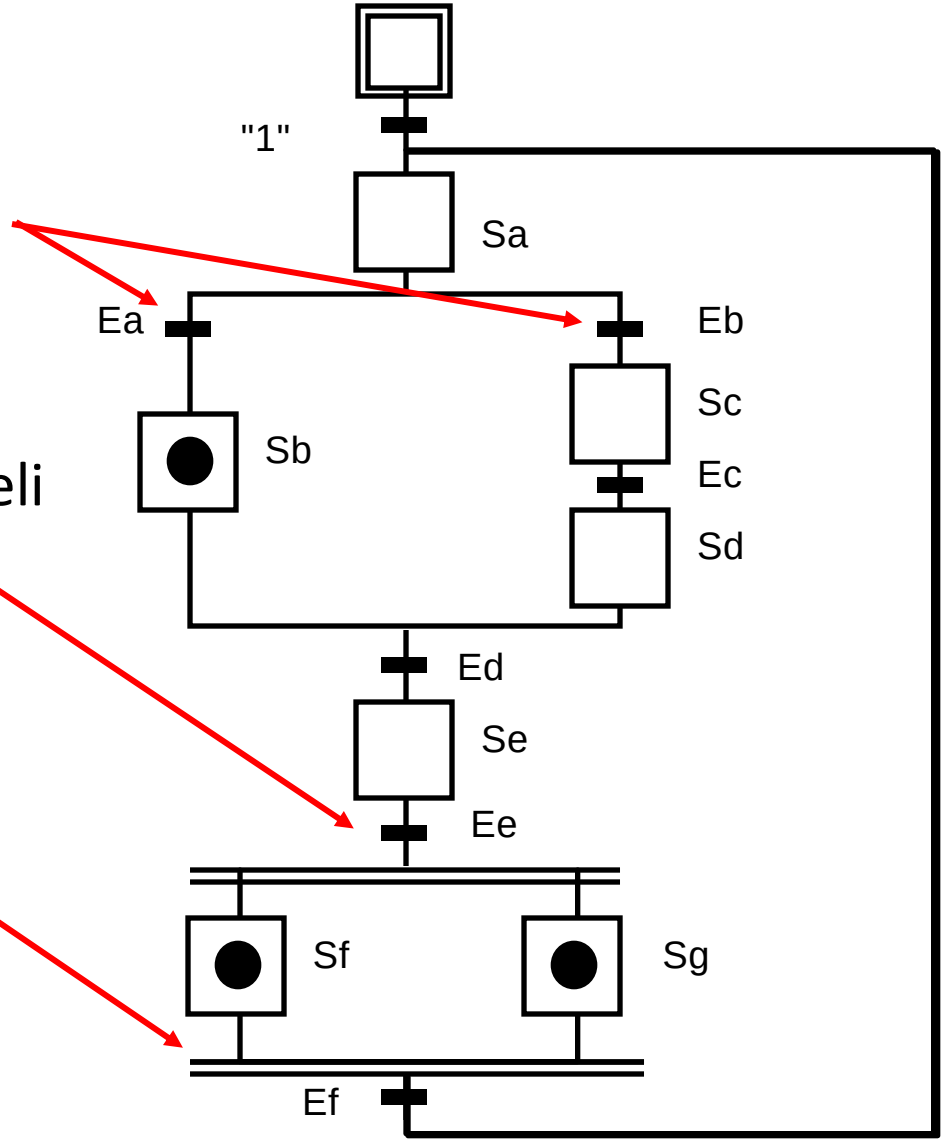


Diagram poteka: program

Nevarnosti zapletenih diagramov

- Smrtni objem (*deadlock*)
- Nekontrolirano delo z žetoni

Rešitev

- Omejitve urejevalnika
- Funkcije urejevalnika

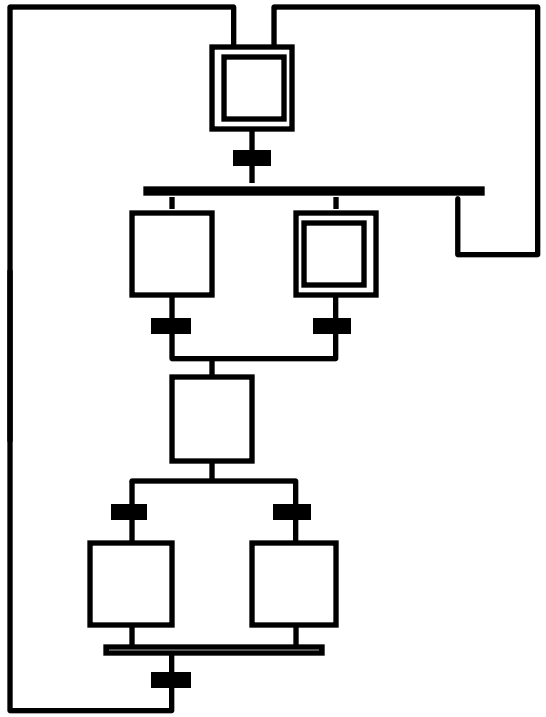
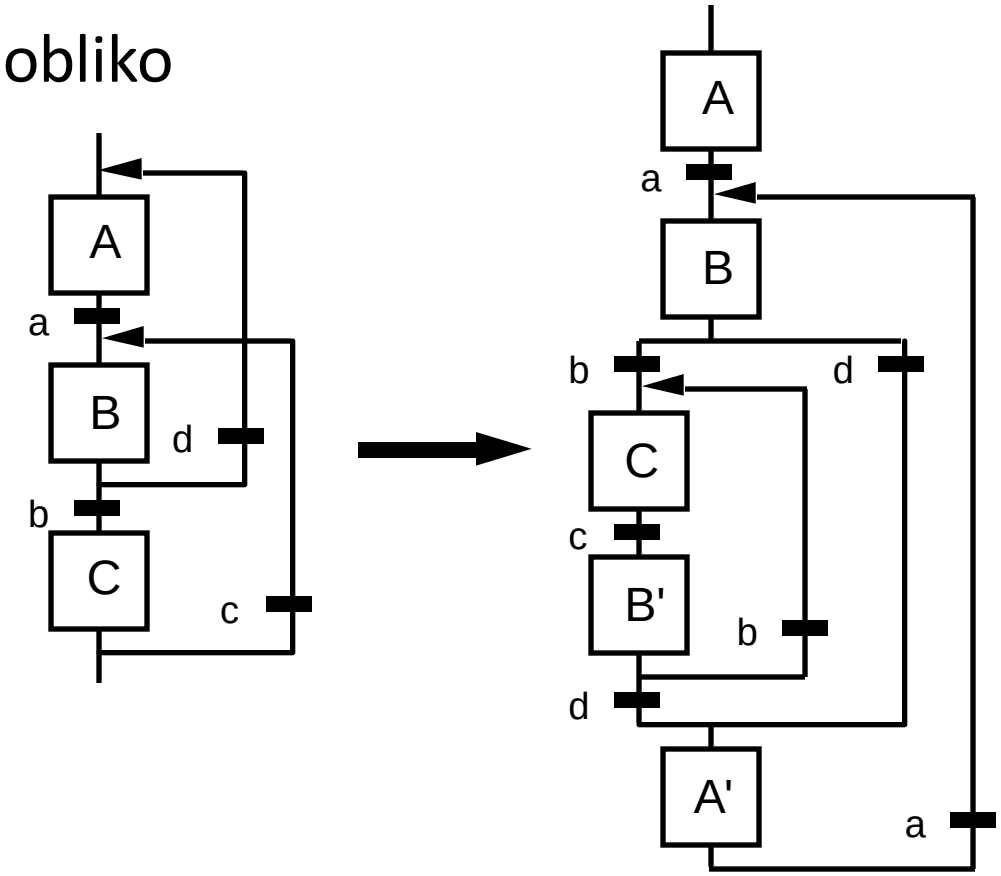


Diagram poteka: program

Zapis diagrama v preglednejšo, strukturirano obliko

- Uporabimo podvajanje stanj



Delo z izjemnimi dogodki

- Zaklepanje stanja (*interlock*)
  - Ob podanem pogoju se akcije v stanju prekinejo
  - Prehod v novo stanje je mogoč
- Nadzorne napake
  - Ko pride do napake, prehod v naslednje stanje ni mogoč
  - Avtomat se ustavi

Diagram poteka: primerjava

Funkcijski načrt

- Zvezno vodenje, regulacija

Diagram poteka

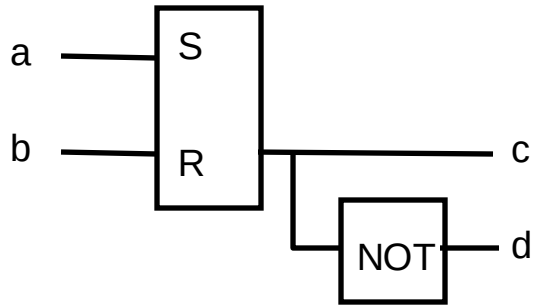
- Koračno/sekvenčno vodenje, krmiljenje

Velikokrat je najboljša izbira kombinacija obeh, zato mora biti komunikacija med njimi mogoča: združevanje na nivoju funkcijskih blokov.

Diagram poteka: primerjava

Primer

- Funkcijski načrt



- Diagram poteka

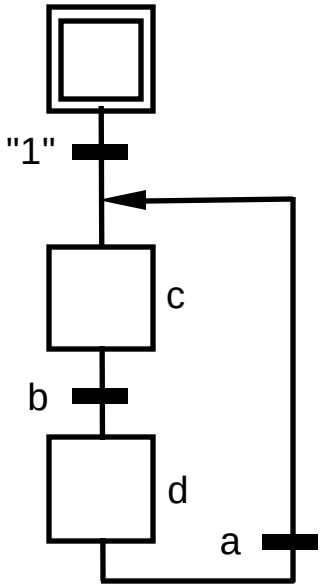
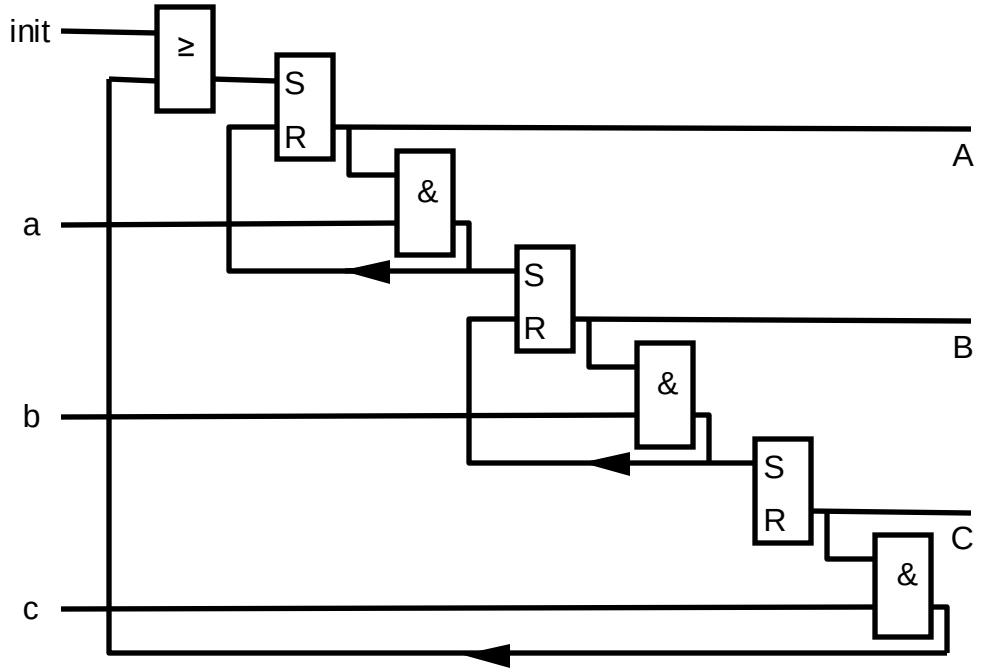


Diagram poteka: primerjava

Primer

- Funkcijski načrt



- Diagram poteka

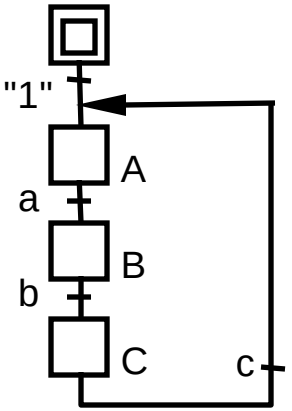


Diagram poteka: primer Siemens Graph

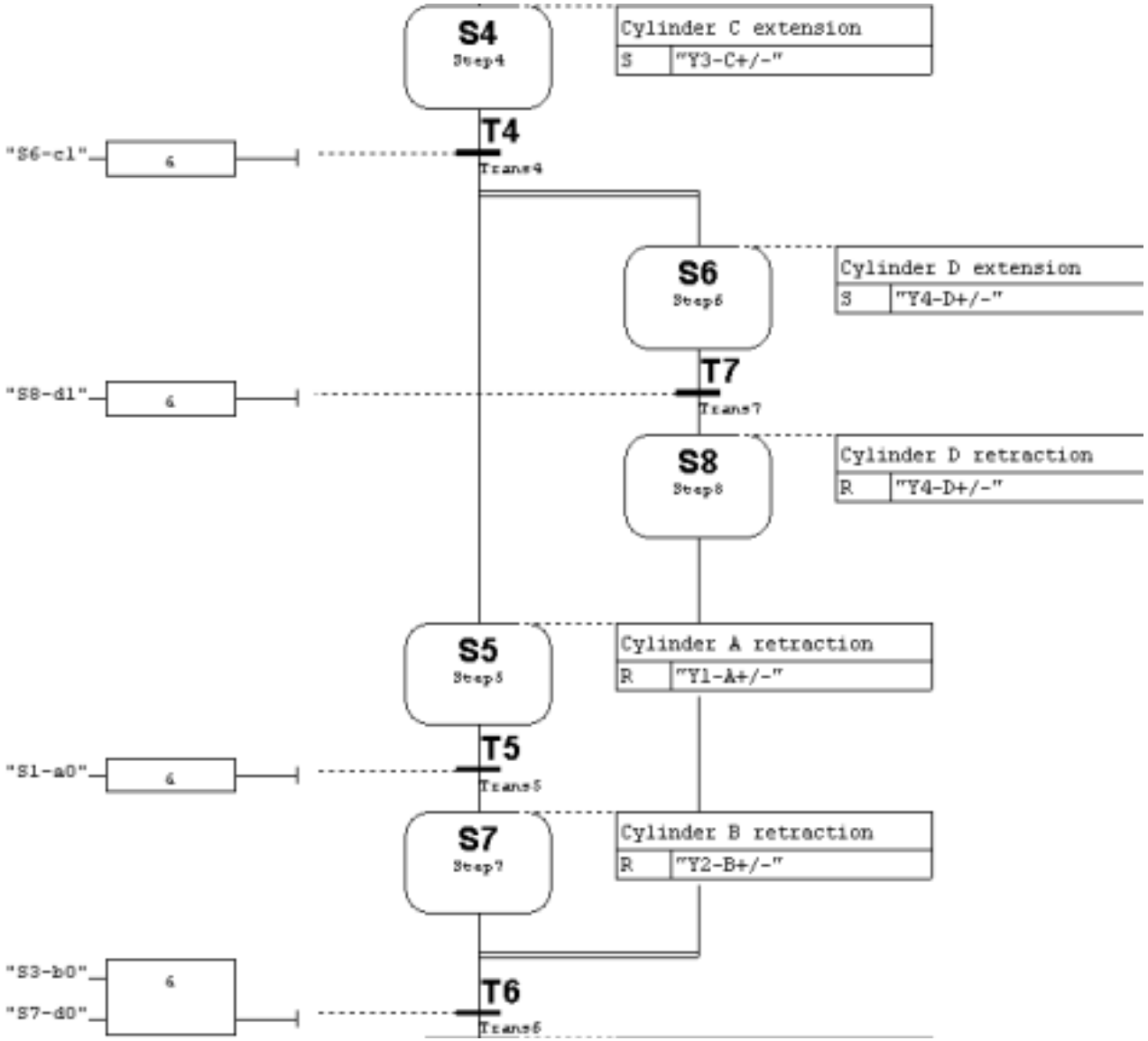


Diagram poteka: TwinCAT

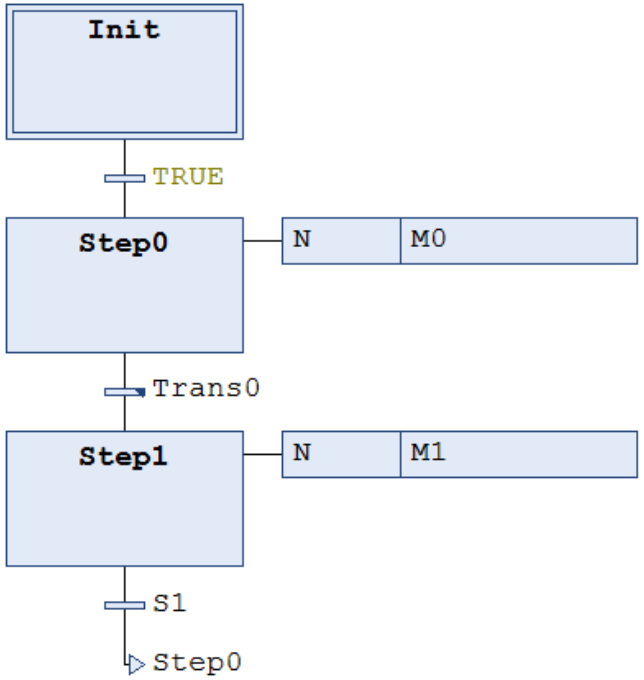
Infosys

SFC lahko uporabimo le v programu ali funkcijskemu bloku, v funkciji ne.

Pogoji za prehode (*transition*) so lahko napisani v poljubnem jeziku IEC in vključeni kot klici ali pa so zapisani neposredno (*inline)* v jeziku ST.

Stanja

- Izvajajo akcije
  - ko je stanje aktivno
    - po standardu IEC – *action association*
    - razširitev standarda – *step main action*
  - ob vstopu (razširitev – *step entry action*)
  - ob izstopu (razširitev – *step exit action*)



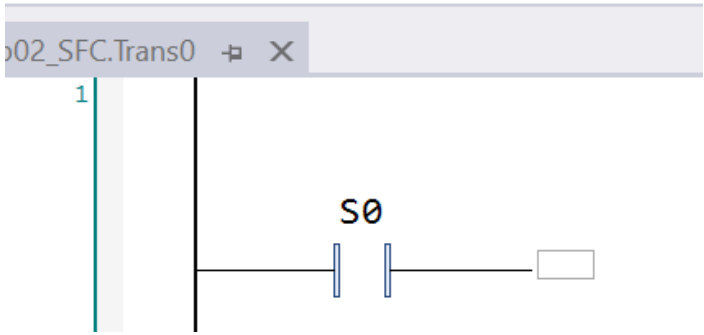


Diagram poteka: Kvalifikatorji akcije v stanju (IEC)

Infosys

Kvalifikator	Ang.	Pomen
N	non-stored	Akcija je aktivna, dokler je stanje aktivno.
R0	overriding reset	Akcija je resetirana, torej deaktivirana.
S0	set / stored	Akcija je izvedena takoj, ko stanje postane aktivno in nadaljuje z izvajanjem tudi, ko stanje ni več aktivno – dokler akcija ni resetirana.
L	time limited	Akcija se začne izvajati takoj, ko je stanje aktivno in se izvaja ves čas aktivnosti stanja oziroma do poteka časovnika.
D	time delayed	Akcija se začne izvajati, ko preteče časovnik ob vstopu v stanje in je stanje še vedno aktivno. Izvaja se dokler je stanje aktivno.
P	pulse	Akcija je izvedena natanko dvakrat: ko stanje postane aktivno in v sledečem programskem ciklu.
SD	stored and time delayed	Akcija se začne izvajati, ko preteče časovnik ob vstopu v stanje in se izvaja, dokler akcije ne resetiramo.
DS	delayed and stored	Akcija se začne izvajati, ko preteče časovnik ob vstopu v stanje in je stanje še vedno aktivno. Izvaja se, dokler akcije ne resetiramo.
SL	stored and time limited	Akcija se začne izvajati takoj, ko je stanje aktivno in se izvaja do poteka časovnika ali do reseta.

Lista ukazov: standard

Nizkonivojski programski jezik

- Podoben zbirniku
- Neprijazen do uporabnika
  - Koda ni strukturirana
  - Šibka semantika
  - Odvisen od programirljivega krmilnika
- Leta 2012 je v 3. izdaji standarda IEC 61131-3 postal zastarel (argument komisije: zbirni jezik ni več primeren za razvoj v modernih razvojnih okoljih).

Večinsko mnenje

- Osnovni jezik, ki naj bi ga podpirali programirljivi logični krmilniki, ki so združljivi z IEC
- Standard osnovnega jezika ne definira

Namenjen izkušenim programerjem za

- Izdelavo časovno in prostorsko učinkovite programske kode
- V ta jezik naj bi se prevedli vsi višjenivojski programski jeziki (tega standard ne zahteva)

Lista ukazov: standard

Vsak ukaz se začne v novi vrstici, ki vsebuje:

- **Oznako** na začetku vrstice, ki se zaključi se z dvopičjem
- **Operacijsko kodo** (operator / mnemonik)
- **Operande**, ločene z vejicami
- **Komentar** na koncu vrstice

Dovoljene so prazne vrstice ali vrstice samo z oklepaji.

Lista ukazov: standard

Osnovni ukazi

- 21 ukazov
- Rezultati operacij se shranjujejo v register RLO (Result of Logic Operation)
- modifikator
  - N – negacija rezultata
  - C – pogojna izvedba
  - (– zakasnitev rezultata

Nr.	Operator	Modifier	Operand	Definition
1	LD	N	Note 1	Sets the actual result of the operand
2	ST	N	Note 1	Stores the actual result in the operand address
3	S	Note 2	**BOOL**	Set Boolean operator to 1
	R	Note 2	BOOL	Reset Boolean operator to 0
4	AND	N, (	BOOL	Boolean AND
5	&	N, (	BOOL	Boolean AND
6	OR	N, (	BOOL	Boolean OR
7	XOR	N, (	BOOL	Boolean Exclusive-OR
8	ADD	(	Note 1	Addition
9	SUB	(	Note 1	Subtraction
10	MUL	(	Note. 1	Multiplication
11	DIV	(	Note 1	Division
12	GT	(	Note 1	Comparison: >
13	GE	(	Note 1	Comparison: >=
14	EQ	(	Note 1	Comparison: =
15	NE	(	Note 1	Comparison: <>
16	LE	(	Note 1	Comparison: <=
17	LT	(	Note 1	Comparison: <
18	JMP	C,N	**MARK**	Jump to the Mark
19	CAL	C,N	NAME	Call function block (Note 3)
20	RET	C,N		Return to a function or a function block
21	)			Processing reset operations

Note 1: The operations must be either loaded or given with a type.
The actual result and the operand must have the same type.
Note 2: The operations are only executed when the value of the actual result is a Boolean 1.
Note 3: A list of arguments in parenthesis follow the name of the function block

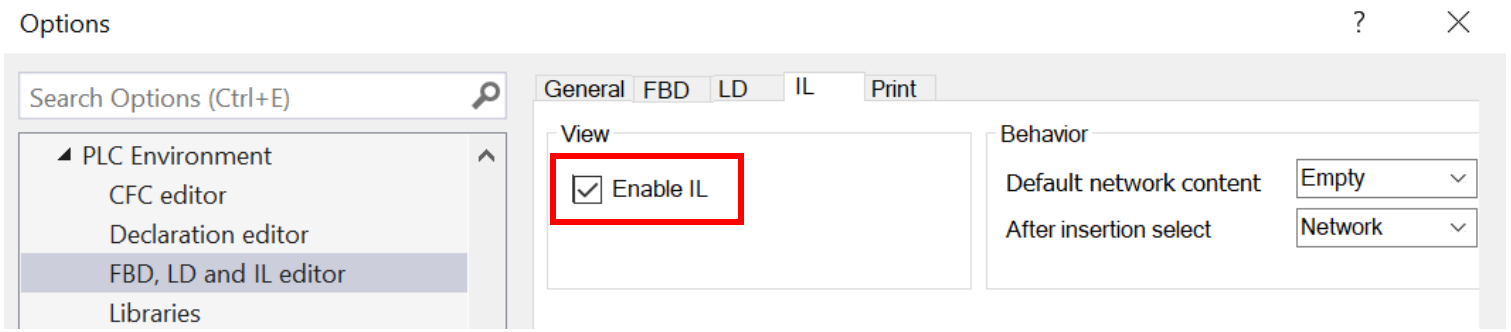
Lista ukazov: standard

Primeri

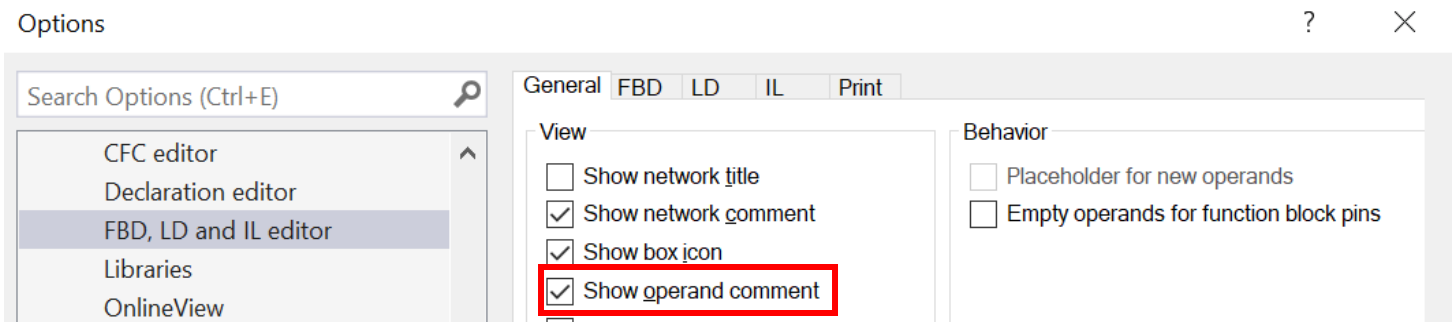
```
AND %IX1  (* Rezultat := Rezultat AND %IX1 *)

AND( %IX1
ORN %IX2
)       (* Rezultat := Rezultat AND (%IX1 OR NOT %IX2) *)

LD   15
ST   C10.PV
LD   %IX10
ST   C10.CU
CAL  C10  (* Klic funkcije: CAL  C10(CU:=%IX10, PV:=15) *)
```

Lista ukazov: TwinCAT

Programski jezik IL moramo omogočiti v nastavitvah:
Tools → Options → TwinCAT → PLC Environment → FBD, LD and IL → IL



Če želimo prikazati komentarje vsake vrstice liste ukazov, to storimo v zavihku *General → Show operand comment*



Lista ukazov: TwinCAT

Koncept klinov (*network*) prevzet iz lestvičnih diagramov.

Primer: funkcijski blok, ki opravlja osnovne logične operacije. FB nato kličemo iz programa.

	Scope	Name	Address	Data type	Initialization	Comment	Attributes
1	VAR_INPUT	**x1**		BOOL			
2	VAR_INPUT	**x2**		BOOL			
3	VAR_OUTPUT	**yNot**		BOOL			
4	VAR_OUTPUT	**yAnd**		BOOL			
5	VAR_OUTPUT	**yOr**		BOOL			

```
1   (* Osnovne logične operacije v listi ukazov: negacija *)
    LDN     x1      (* Load negated x1. RLO = NOT x1 *)
    ST      yNot    (* Store RLO into yNot. *)

2   (* Logični IN *)
    LD      x1      (* RLO = x1 *)
    AND     x2      (* RLO = RLO AND x2 *)
    ST      yAnd    (* yAnd = RLO *)

3   (* Logični ALI *)
    LD      x1      (* RLO = x1 *)
    OR      x2      (* RLO = RLO OR x2 *)
    ST      yAnd    (* yAnd = RLO *)
```

	Scope	Name	Address	Data type	Initialization	Comment	Attributes
1	VAR	**S0**		BOOL			
2	VAR	**S1**		BOOL			
3	VAR	**S2**		BOOL			
4	VAR	**L0**		BOOL			
5	VAR	**L1**		BOOL			
6	VAR	**L2**		BOOL			
7	VAR	**logicneOperacije**		p03_IL_lo...			

```
1   (* Pogojni klic FB - če je aktiven S2. Sicer klic preskočimo. *)
    LD      S2
    JMPC    SKOK
    CAL     logicneOperacije(
              x1:= S0,
              x2:= S1,
            yNot=> L0,
            yAnd=> L1,
             yOr=> L2)

2   (* Skok se zgodi na oznako (label), ki jo damo posameznemu klinu. *)
    SKOK:
```

Strukturirano besedilo: standard

Infosys

- Jezik podoben Pascalu
- Primeren za kompleksne obdelave podatkov
- Z izrazi določamo vrednosti na podlagi vrednosti spremenljivk in konstant
- Nujna je uporaba zahtevanih podatkovnih tipov
  - Pretvarjanje med tipi s funkcijami
  - Primer: REAL_TO_INT(..)
- Izraz je sestavljen iz operatorjev in operandov
  - Izračunavanje po prioriteti
  - V primeru enake prioritete od leve proti desni
  - Primer: X := (A+B-C)*ABS(D);

	Operacija	Oznaka	Prioriteta
1	Oklepaj	`(izraz)`	Visoka
2	Klic funkcije	`ImeF(argumenti)` `LN(A), MAX(X,Y)`	
3	Potenciranje	`EXPT`	
4	Predznak	`-`	
5	Negacija	`NOT`	
6	Množenje	`*`	
7	Deljenje	`/`	
8	Modulo	`MOD`	
9	Seštevanje	`+`	
10	Odštevanje	`-`	
11	Primerjava	`<, >, <=, >=`	
12	Enakost	`=`	
13	Neenakost	`<>`	
14	IN	`AND, AND_THEN`	
16	Ekskluzivni ALI	`XOR`	
17	ALI	`OR, OR_THEN`	Nizka

Strukturirano besedilo: standard – stavki

Infosys

Stavek	Primer
Prireditev	`CV := CV + 1;` `C  := SIN(X);`

Klic in uporaba funkcijskega bloka	`CMD_TIMER(IN := %IX5,`
	`PT := T#300ms);`
	`A := CMD_TIMER.Q;`

Izhod iz FB/FUN	`RETURN;`

IF	`IF D < 0.0 THEN`
	`  NROOTS := 0;`
	`ELSIF D = 0.0 THEN`
	`  NROOTS := 1;`
	`ELSE`
	`  NROOTS := 2;`
	`END_IF;`

CASE	`TW := BCD_TO_INT(THUMBWHEEL);`
	`CASE TW OF`
	`  1:  DISPLAY := OVEN_TEMP;`
	`  2,3: DISPLAY := MOTOR_SPEED;`
	`ELSE`
	`  DISPLAY := 0;`
	`END_CASE;`

Stavek	Primer

Zanka FOR	`J := 101;`
	`FOR I := 1 TO 100 BY 2 DO`
	`  IF WORDS[I] = 'KEY' THEN`
	`  J := I;`
	`  EXIT;`
	`  END_IF;`
	`END_FOR;`

Zanka WHILE	`J := 1;`
	`WHILE J <= 100 DO`
	`  J := J + 2;`
	`END_WHILE;`

Zanka REPEAT	`J := -1;`
	`REPEAT`
	`  J := J + 2;`
	`UNTIL J = 101`
	`END_REPEAT;`

EXIT	`EXIT;`
Prazen stavek	`;`

Strukturirano besedilo: primeri

```
// Stavek IF
IF preklopnik = TRUE THEN
    lucka := TRUE;
END_IF

// Če pogoj ni resničen, lucka ne spremeni stanja, torej se
// stanje ohranja/pomni. Eksplicitno bi to zapisali:
IF preklopnik = TRUE THEN
    lucka := TRUE;
ELSE
    lucka := lucka;
END_IF

// Smo to želeli? Ali bi radi ob izklopu preklopnika
// ugasnili tudi lučko?
IF preklopnik THEN
    lucka := TRUE;
ELSE
    lucka := FALSE;
END_IF

// Ali za to sploh potrebujemo pogojni stavek? Ne ...
lucka := preklopnik;
```

```
// Če se le da, pogojne stavke nadomestimo z logičnimi izrazi
IF preklopnik THEN
    IF tipka1 THEN
        lucka := TRUE;
    ELSE
        lucka := FALSE;
    END_IF
ELSE
    lucka := tipka2;
END_IF

// Enako kot:
lucka := (preklopnik AND tipka1) OR (NOT preklopnik AND tipka2);
```

```
// Detekcija pozitivne fronte na tipki
IF tipka AND NOT tipkaStaro THEN
    // Ob pozitivni fronti obrnemo stanje lucke
    lucka := NOT lucka;
    // Izpustimo ELSE, da povzročimo pomnjenje
END_IF

// Posodobimo staro stanje tipke
tipkaStaro := tipka;
```

Strukturirano besedilo: primeri

Funkcijski blok za krmiljenje motorja tipa premik-smer

- Blokada prehitre menjave smeri vrtenja
- Varčevanje z releji (specifika učnih modelov)

	Scope	Name	Address	Data type	Initialization	Comment	Attributes
1	VAR_INPUT	**naprej**		BOOL			
2	VAR_INPUT	**nazaj**		BOOL			
3	VAR_INPUT	**casBlokade**		TIME			
4	VAR_OUTPUT	**premik**		BOOL			
5	VAR_OUTPUT	**smer**		BOOL			
6	VAR	**casovnikNaprej**		TOF			
7	VAR	**casovnikNazaj**		TOF			
8	VAR	**vrtiNaprej**		BOOL			
9	VAR	**vrtiNazaj**		BOOL			
10	VAR	**blokadaNazaj**		BOOL			
11	VAR	**blokadaNaprej**		BOOL			

```
1
2  // Klic FB časovnikov
3  casovnikNaprej(IN:=naprej, PT:=casBlokade, Q => blokadaNazaj);
4  casovnikNazaj(IN:=nazaj, PT:=casBlokade, Q => blokadaNaprej);
5
6  // Izračun, ali lahko motor vrtimo naprej oziroma nazaj
7  vrtiNaprej := naprej AND NOT blokadaNaprej;
8  vrtiNazaj := nazaj AND NOT blokadaNazaj;
9
10 // Izhod premik
11 premik := vrtiNaprej OR vrtiNazaj;
12
13 // Izhod smer - želimo varčevati rele za smer,
14 // torej bomo pomnili stanje smeri.
15 (*
16 IF vrtiNazaj THEN
17     smer := TRUE;
18 ELSIF vrtiNaprej THEN
19     smer := FALSE;
20 ELSE                // To lahko
21     smer := smer;  // izpustimo.
22 END_IF
23 *)
24
25 // Krajše in slajše:
26 IF premik THEN
27     smer := vrtiNazaj;
28 END_IF
29
```

Tehnike programiranja: uvedba stanj

Razlogi

- Določeni deli programa se lahko izvajajo samo ob določenih pogojih
- Potreba po zaklepanju klinov oz. delov programske kode

Razdelitev programa na logična stanja

- Stanja in prehodi med njimi morajo biti jasno določeni tako v ročnem kot avtomatskem načinu
  - Stanja določena glede na akcije izvršnih sistemov in vrednosti merilnih sistemov
- Lažje programiranje kompleksnih sistemov
- Lažji obratni inženiring
  - Koda za vsako stanje enostavnejša
  - Pogoji za prehajanje med stanji so veliko bolj očitni
  - Vsak programer piše na svoj način

Prednosti

- Skrajšanje zagona sistema zaradi napak v programu za 85 %
  - Predvsem na račun enostavnejših pogojev za zaklepanje klinov
  - V tipičnem lestvičnem diagram je velik del kode namenjenih zaklepanju klinov
    - 35 % pri procesni kontroli (zvezni procesi, regulacija)
    - 60 % pri sekvenčnem procesu

Tehnike programiranja: uvedba stanj

Programiranje

- Posnemanje konceptov jezika diagram poteka (SFC)
- Ob izpolnjenem pogoju za prehod v novo stanje se:
  - aktivira ustrezna spremenljivka (žeton) za novo stanje (nastavimo, *set*)
  - deaktivira spremenljivka za trenutno stanje (brišemo, *reset*)
    - Če je lahko hkrati aktivnih več stanj, je potrebno paziti, da se pri prehodu v novo skupno stanje deaktivirajo vsa trenutna stanja.
- Označevanje stanj
    - Z biti: en bit ustreza enemu stanju (ang. *one hot encoded*)
    - Številčno: uporaba celoštevilčne spremenljivke (in primerjalnika)
- Ob zagonu sistema je potrebna logika, ki zna
  - ugotoviti, v katerem stanju se je sistem ustavil
    - preskakovanje stanj in ne njihovo zaporedno izvajanje, ki je lahko zelo nevarno!
  - postaviti sistem v začetno stanje ali
  - preprečiti njegovo delovanje, če ni v pravem stanju, in to ustrezno alarmirati (najlažje)

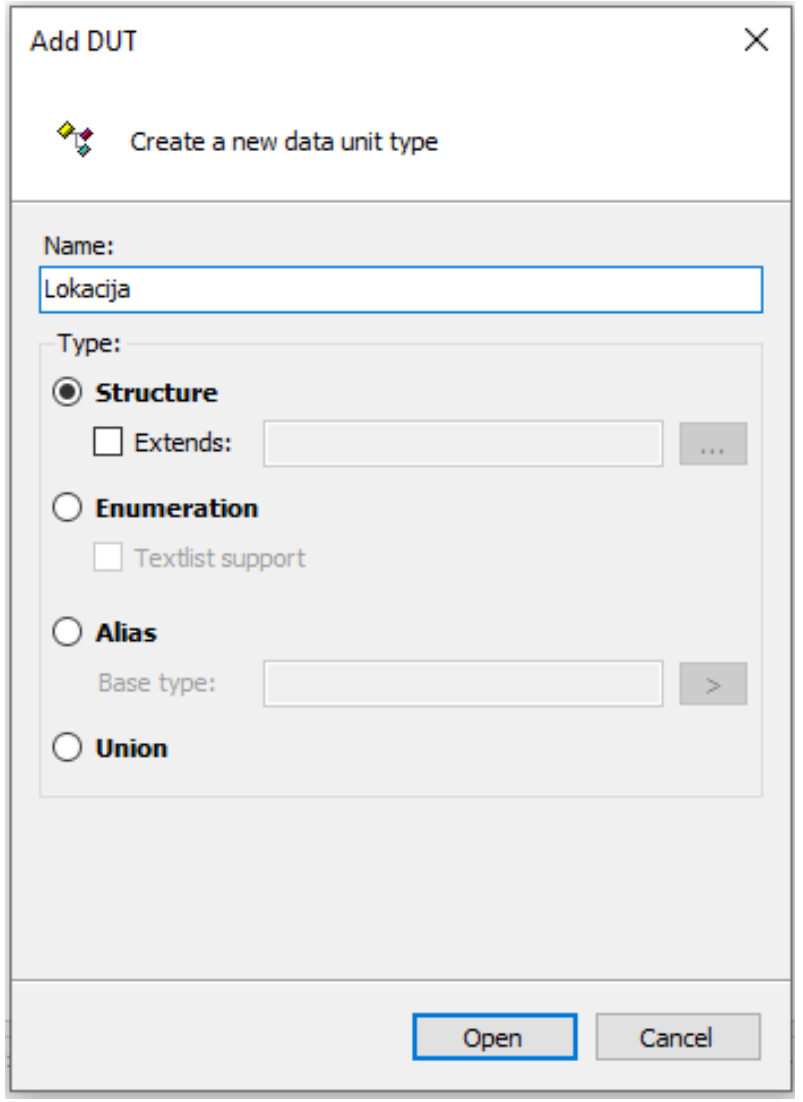
Tehnike programiranja: spremljanje materiala

Izdelava sestavljenega podatkovnega tipa – strukture (DUT, *data unit type*), ki predstavlja logično sliko enega obdelovanca / materiala

- Podatki o materialu: črtna koda (ID), fizične mere, neustreznost, …
- Ciljna lokacija
- Navodila za obdelavo (recept)
- Funkcije na trenutni lokaciji: zasedenost , premikanje, …

TwinCAT

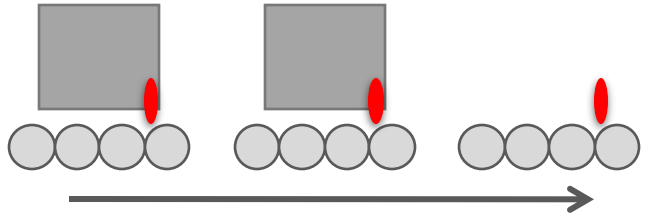
- PLC → <Project> → DUTs → Add → DUT...
- Ustvarimo novo strukturo
- Pomagamo si lahko tudi s tabelami (ARRAY)



Tehnike programiranja: spremljanje materiala

Sekvenčni proces

- Vsaka fizična enota (mesto) ima tudi svojo logično sliko
- Vsaka enota na enkrat lahko izpolnjuje le eno nalogo
  - Primer: trije tekoči trakovi
  - Enostavna komunikacija:
    zahteva, dovoljenje, akcija, potrditev, (alarm)



- Prepisovanje struktur
  - Ob izpolnjenih pogojih se celotna slika prenese iz enega mesta na drugega
    - Material je fizično na novem mestu
    - Senzorska slika ustreza bodoči logični sliki (dvojna kontrola)
  - Prepis naj NE bo vezan na fronto fotocelice
- Alarmiranje v primeru, da se logična in fizična shema po določenem času ne ujemata

Tehnike programiranja: organizacija programa

1. Branje senzorjev v strukture
   Umerjanje senzorjev, skaliranje analognih vrednosti, pretvorbe (NC → NO)
2. Proženje alarmov
3. Upravljanje alarmov
   Potrjevanje in ničenje alarmov
4. Priprava podatkov za vmesnike človek-stroj
   Pretvorbe (NC → NO), izračuni, ločene podatkovne strukture (svoj FB) zaradi boljše preglednosti
5. Glavni program
   Avtomat prehajanja stanj
6. Sledenje materiala
   Glede na fizično sliko
7. Varnostne funkcije
   - Varovanje človeka in opreme
   - Zaradi varnosti neodvisne od glavnega programa
   - Blokada prehitre menjave smeri vrtenja
8. Aktivacija izvršnih sistemov

Tehnike programiranja: učni modeli

Linija z dvema napravama in Pnevmatski sistem:

- Definicija lokacij, kjer je tipalo (fotocelica ali končno stikalo); "navidezne" lokacije (potiskač, vrtljiva miza, vhod na trak)
- Avtomat za vsako lokacijo ali izvršni člen
- Odvisnosti (predhodno, naslednje mesto)
- Vrtljiva miza (vodenje položaja, materiala na mizah)
- Spremljanje materiala
  - črtna koda (ID)
  - naloga/recept
  - vnos vrednosti na prvi lokaciji preko IDE

Robot

- Hierarhija avtomatov:
  - Avtomat za vsako os (vrtenje, dvig, izteg, prijem)
  - Avtomat, ki povezuje vse štiri osi
    - Pojdi na lokacijo
    - Pojdi na lokacijo in poberi
    - Pojdi na lokacijo in odloži
  - Avtomat, ki izvaja "program" premikov
- Spremljanje materiala
  - črtna koda (ID)
  - lokacije/pozicije objektov