UNIVERSITATEA POLITEHNICA BUCUREŞTI
FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE

AUTOMATE PROGRAMABILE

THEODOR BORANGIU **ANDREI NICK IVĂNESCU**

SORIN BROTAC

CUPRINS

CAP 1. CARACTERISTICI ŞI PERFORMANȚE ALE AUTOMATELOR PROGRAMABILE

Un AP (Automat Programabil) este un dispozitiv apărut pentru a înlocui releele şi schemele secvențiale necesare pentru controlul sistemelor automate [Borangiu, 1986]. Principiul de bază al unui AP este următorul: verifică starea intrărilor şi, în funcție de acestea, activează sau dezactivează ieşirile. Utilizatorul introduce un program, care face ca automatul să dea rezultatele dorite.

1.1 Structura automatelor programabile

Un AP este compus în principal din: unitate centrală (UC), zonă de memorie şi circuite pentru recepționarea datelor de intrare/ieşire (fig 1. 1). Putem considera AP-ul ca o cutie plină de relee individuale, numărătoare, ceasuri şi locații de memorare a datelor. În general componentele unui automat programabil sunt:

- Unitate centrală
- Module de intrare / ieşire
- Regiştrii de intrare / ieşire
- Memorie de date
- Regiştrii interni
- Circuite de temporizare
- Circuite de numărare

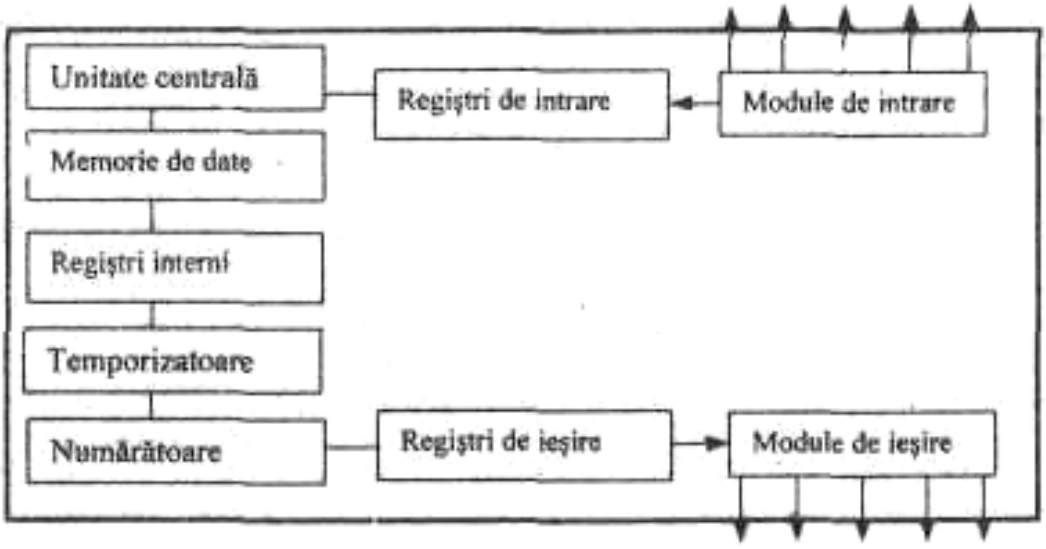


Fig. 1.1 Structura unui automat programabil

Funcționarea lor este următoarea:

- Unitatea centrală : conține un procesor, o unitate de calcul aritmetic şi diferite tipuri de memorie. Gama de procesoare folosite de către producătorii de automate programbile este foarte diversificată, câteva exemple ar putea fi procesoarele Motorola, Intel, Siemens. Procesoarele folosesc o memorie de lucru de tip RAM pentru execuția instrucțiunilor, programul de executat fiind , de obicei memorat într-o memone de tip Flash. Dimensiunile memoriilor diferă de la un tip de automat la altul, influențând performanțele şi costul automatului programabil.
- Module de intrare : conțin unul sau mai multe circuite de intrare. Acestea există fizic, sunt conectate la lumea exterioară şi recepționează semnale de la comutatoare, senzori

etc. Semnalele de citite pot fi de două tipuri: digitale sau analogice. De regulă circuitele de intrare sunt implementate cu relee sau tranzistori.

- Module de ieşire : conţin unul sau mai multe circuile de ieşire. Acestea există fizic, sunt conectate la lumea exterioară şi transmit semnale digitale sau analogice către diferite elemente de execuţie. Ca variantă constructivă pot fi aleşi tranzistorii, releele sau triacele.
- Regiştri de intrare : sunt regişiri asociaţi intrărilor fizice. Valoarea semnalelor de intrare este convertită în formă binara de către circuitele de intrare şi memorată în aceşti regiştri. Pentru intrările de tip digital, un singur bit dintr-un registru poate memora starea activă/inactivă a intrării, dar în cazul unei intrări analogice sunt necesari mai mult biţi pentru memorarea valorii în format numeric.
- Regiştri de ieşire : sunt regiştri asociaţi ieşirilor fizice. Valoarea semnalelor de ieşire este scrisă aici în formă binară, coversia într-un semnal electric de o anumită valoare fiind făcută de către circuitele de ieşire (convertoare digital-analogice). Pentru ieşirile de tip digital, un singur bit dintr-un registru poate memora starea activă/inactivă a ieşirii, dar în cazul unei ieşiri analogice sunt necesari mai mult biţi pentru scrierea valorii în format binar.
- Regiştri interni : nu recepţionează semnale din mediul extern, nici nu există fizic. Conţin relee simulate prin biţii din regiştri şi au fost introduse pentru a elimina releele interne fizice. Există şi regişiri speciali care sunt dedicaţi unui anumit scop. Unele relee sunt intotdeauna deschise, altele sunt întotdeauna închise. Unele sunt deschise numai o dată la pornire şi sunt folosite pentru iniţializarea dalelor memorate.
- Numărătoare : nici acestea nu există fizic. Sunt numărătoare simulate şi pot fi programate să contorizeze impulsuri. De obicei, aceste numărătoare pot număra crescător, descrescător şi în ambele sensuri. Anumiţi producători includ şi numărătoare de mare viteză implementate hard, pe care am putea să le considerăm ca existente fizic. Şi acestea pot număra crescător, descrescător sau în ambele sensuri.
- Circuite de temporizare : nici circuitele de temporizare nu există fizic. Sunt simulate software şi contorizează perioadele timp. Pot fi găsite în diverse variante în ceea ce priveşte parametrii. Pasul de incrementare variază de la 1ms până la 1s.
- Memoria de date : în mod normal este vorba de simpli regiştri care memorează date. De obicei sunt folosiţi pentru aplicaţii matematice sau pentru manipularea datelor. Pot fi folosiţi şi pentru memorarea datelor cât timp AP-ului i se întrerupe alimentarea. La pornire, vor avea acelaşi conţinut ca înainte de oprire.

1.2 Module de intrare-ieşire

Există o mare varietate de module de intrare-ieşire ce pot intra în componenţa unui AP. Există două categorii mari de module:

1. *Module I/O analogice*: semnalul pe care îl transmit sau îl recepţionează are o valoare analogică
2. *Module l/O digital*: semnalul este de tip digital, putând avea doar două valori.

În funcţie de mărimile fizice citite sau de semnalele transmise, iată câteva exemple de module ce pot fi ataşate unui AP:

- intrări/ieşiri de curent continuu
- intrări/ieşiri de curent alternativ
- intrări/ieşiri de tensiuni continui sau alternative
- module de citit temperaturi
- module de reglare PID
- module de control Fuzzy

- **Intrări de curent continuu**

Sunt disponibile circuite care funcţionează la 5, 12, 24 şi 48 de volţi. Modulele de intrare permit conectarea unor dispozitive de tip tranzistor PNP sau NPN. În cazul unui comutator nu se pune problema conectării de tip NPN sau PNP. Fotocuploarele sunt folosite pentru a izola circuitele interne ale AP-ului de intrări. Acest lucru este necesar pentru a elimina zgomotele şi se realizează prin conversia semnalului de intrare din formă electrică in formă luminoasă şi apoi invers.

- **Intrări de curent alternativ**

Modulele de intrare obişnuite funcţionează la 24, 48, 110, 220 volţi. Dispozitivele legate la intrarea de curent alternativ sunt sesizate mai greu de către AP. De cele mai multe ori acest lucru nu are importanţă pentru cel care programează AP-ul, dar este bine de ştiut că se pierde timp din cauză că dispozitivele pe bază de curent alternativ sunt de obicei dispozitive mecanice, care sunt mai lente, în plus apare şi o întârziere de cel puţin 25 ms datorită filtrării care are loc la intrarea în AP. De reţinut că AP-urile funcţionează la cel mult 5 V_{CC}.

- **Ieşiri cu relee**

Ieşirile cele mai obişnuite sunt cele de tip releu. Releele pot fi folosite atât cu sarcină de curent alternativ, cât şi continuu. Metoda standard de conectare a sarcinilor la ieşirile AP-ului presupune folosirea unei surse de curent alternativ, însă poate fi folosit şi curentul continuu.

Releele se află în interiorul automatului programabil. Atunci când programul care rulează în automat indică ieşirii să devină activă (adevărată), AP-ul va aplica o tensiune bobinei releului, care va închide contactul corespunzător. La închiderea contactului începe să circule curent prin circuitul extern. Atunci când programul indică dezactivarea ieşirii, AP-ul va întrerupe tensiunea aplicată bobinei releului, circuitul extern va fi deschis, deci inactiv.

1.3 Funcţionarea automatelor programabile

Automatele programabile funcţionează scanând (executând) continuu un program. Putem spune că un ciclu de scanare are trei paşi importanţi (fig. 1. 2).

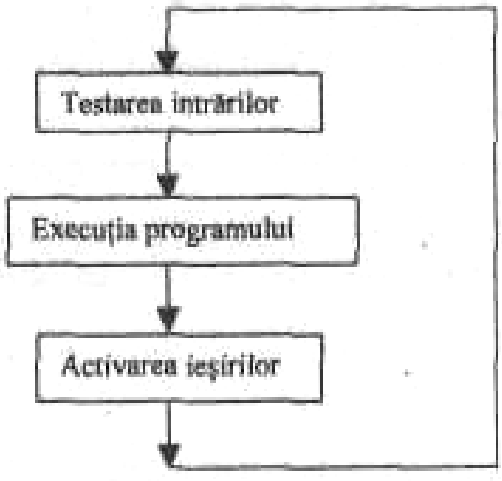


Fig. 1.2 Ciclul de funcţionare al unui automat programabil

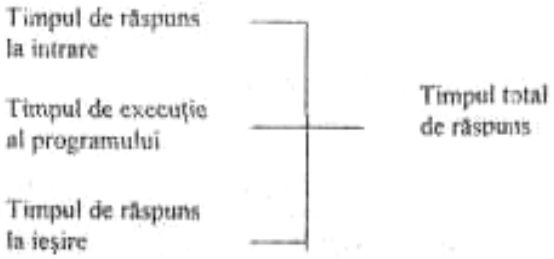
- Pasul 1-TESTAREA INTRĂRILOR-AP-ul cercetează starea intrărilor (activ/inactiv în cazul intrărilor numerice sau valoarea unei intrări analogice) şi o copiază, în formă binară, în anumiţi regiştri asociaţi intrărilor.
- Pasul 2-EXECUTAREA PROGRAMULUI-AP-ul execută programul instrucţiune cu instrucţiune. În funcţie de starea intrărilor si de logica programului, schimbă configuraţia registrilor de ieşire, în formă binară.

- Pasul 3-ACTUALIZAREA IEŞIRILOR-în final AP-u actualizează şi starea ieşirilor fizice pe baza stării ieşirilor rezultate din pasul anterior. După cel de-al treilea pas, AP-ul revine la pasul 1 şi reia ciclul. Un ciclu de scanare este definit ca timpul în care se execută cei trei paşi de mai sus.

De fapt sunt mai mult de trei paşi. Mai au loc verificarea sistemului şi actualizarea valorilor curente ale ceasului şi numărătorului intern.

Timpul de răspuns

Timpul total de răspuns este un parametru important al unui automat programabil. Unui automat programabil îi trebuie un anumit timp pentru a reacționa la schimbările valorilor de intrare. În unele aplicații viteza nu este importantă, în altele da.



AP-ul poate vedea dacă o intrare este activiă sau inactivă doar dacă o citeşte. Cu alte cuvinte, el testează intrările doar in porțiunea corespunzătoare a ciclului de scanare (fig. 1. 3).

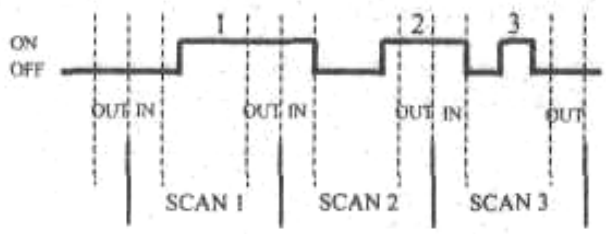


Fig. 1.3 Situații posibile de variație a intrărilor

În diagrama de mai sus, intrarea I nu este văzută înainte de ciclul 2. Acest lucru se întâmplă deoarece atunci când s-a activat intrarea I, ciclul 1 terminase deja de testat intrările.

Intrarea 2 nu este văzută înainte de ciclul 3. Acest lucru are loc deoarece ciclul 2 deja terminase de testat intrările.

Intrarea 3 nu este văzută niciodată, deoarece când ciclul 3 testa intrările, semnalul 3 nu era încă activ, iar la începutul ciclului 4 era deja inactiv.

Pentru a evita acest lucru trebuie ca intrarea să fie activă pentru cel puțin un timp de răspuns la intrare + un timp de scanare.

Dacă nu este posibil ca intrările să fie active atât de mult timp, AP-ul nu mai poate citi intrările active nemaiputând implementa corect o aplicație de control. Există o cale de a rezolva acest neajuns (mai precis două):

Funcția de extindere a impulsului. Această funcție extinde lungimea (durata) unui semnal de intrare până la momentul la care AP-ul testează intrările, în ciclul următor.

Funcția de întrerupere. Această funcție întrerupe ciclul de scanare pentru a rula o rutină specială scrisă de utilizator. Când se activează o intrare, indiferent de starea ciclului de scanare, AP-ul opreşte execuția programului principal şi execută rutina de întrerupere (fig. 4).

O rutină poate fi privită ca un mini-program în afara programului principal. După ce a terminat execuția rutinei de întrerupere, se întoarce în punctul în care se oprise şi continuă norma procesul de scanare.

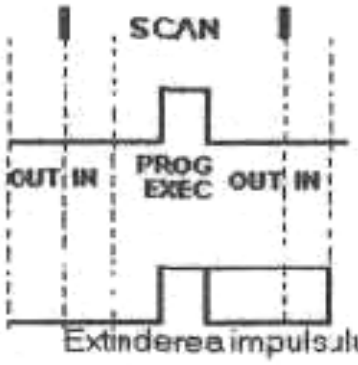


Fig. 1.4 Extinderea impulsului prin întrerupere

CAP. 2 LIMBAJE DE PROGRAMARE A AUTOMATELOR PROGRAMABILE

Datorită diversității firmelor producătoare şi tipurilor de automate programabile, există mai multe variante de programare a unui automat programabil, dintre care pot fi amintite:

- Programarea cu ajutorul limbajelor grafice, de tip (Grafcet sau Isagraf)
- Programarea prin diagrama Ladder
- Programarea in limbaje de nivel înalt (C, Pascal)

2.1 Limbajul Grafcet

Limbajul Grafcet reprezintă o modalitate de descriere grafică a unui sistem logic secvențial, fiind util datorită generalității şi facilităților de care dispune, spre exemplu posibilitatea descrierii secvențelor paralele [Arzen, 1994].

Elementele constitutive ale unei diagrame Grafcet sunt următoarele:

- Etape: corespund unei stări stabite a sistemului automat şi sunt identificate printr-un număr unic

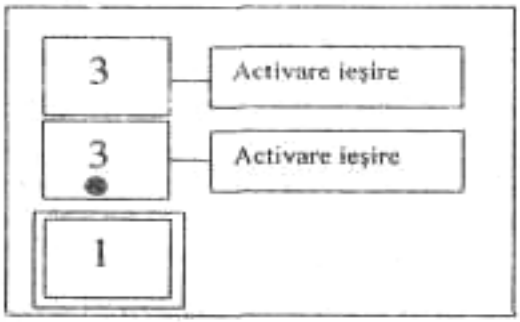


Fig. 2.1 Etape Grafcet

  Etapele pot fi:
  - inițiale: aceste etape sunt active inițial (etapa I din fig. 2. 1)
  - normale: sunt celelalte etape care nu sunt active inițiale (prima etapă 3 din fig. 2. 1). Ele se pot activa la un moment dat (etapa 3 cu un punct ataşat din fig. 2. 1).
- *Tranziții*: o tranziție indică posibilitatea evoluției dintr-o stare activă într-o nouă stare. Fiecărei tranztții îi este asociată o condiție logică, în funcție de valorile logice ale unor variabile de intrare sau de starea activă sau inactivă a altor etape. Condiția de tranziție nu este validată decât dacă etapele imediat precedente sunt active.

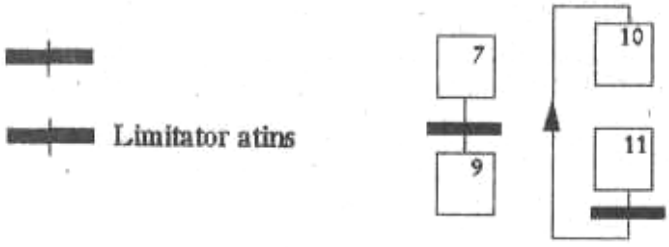


Fig 2.2 Diferite reprezentări de tranziții

- *Acțiuni asociate etapelor*: deoarece o etapa poate fi activă sau inactivă la un moment dat, există acțiuni asociate etapelor, care se execută numai la activarea acestora. Pot exista şi acțiuni condiționate, ce se execută dacă, suplimentar, mai este adevărată altă condiție logică.
  Acțiunile pot fi diverse:

- deschiderea unei vane
- oprirea sau pornirea unui motor (fig. 2. 3)
- incrementarea unui contor
- pornirea unei temporizări

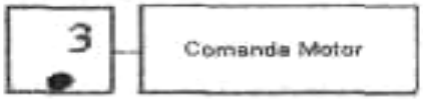


Fig. 2.3 Comanda unui motor

- Ramificații (divergențe): au loc între mai multe secvențe posibile, atunci când condițiile de tranziție se exclud între ele. Convergența diferitelor ramuri are loc atunci când sunt îndeplinite condițiile de tranziție pe fiecare ramură. Se pot stabili priorități dacă nu se exclud condițiile de tranziție.

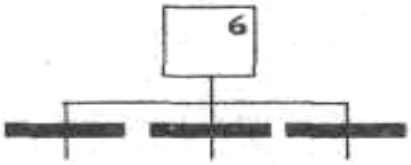


Fig. 2.4 Ramificație (divergență)

- Secvențe simultane (paralelism): mai multe secvențe pot fi activate simultan, plecând de la o condiție de tranziție. Evoluția, pe flecare ramură, se efectuează simultan. Joncțiunea nu se poate efectua decât atunci când toate secvențele sunt terminate.

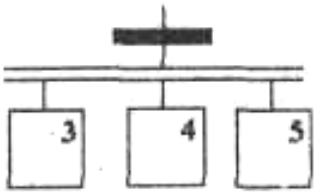


Fig. 2.5 Secvențe simultane (paralelism)

- Macro-etape: macro-etapa este o reprezentare unică a unui ansamblu de etape şi de tranziții. Sunt caracterizate de o etapă inițială şi una finala. Etapa inițială se supune regulilor cunoscute. Etapa finală nu poate avea acțiuni asociate. Cât timp macro-etapa este activă, evoluția Grafcet-ului respectă regulile obişnuite de evoluție. Macro-etapa devine activă când etapa anterioară etapei inițiale este activă şi condiția de tranziție a sa devine adevărată. Ea este dezactivată când etapa finală este activă şi condiția de tranziție asociată este adevărată. Stările unei macro-etape pot fi:
  - de repaus: nici o etapă componentă nu este activă
  - activă: cel puțin o etapă componentă este activă, în afară de etapa finală
  - de sfârşit: etapa finală este activă.

  O macro-etapă poate conține una sau mai multe etape inițiale. Aceste etape pot fi activate la punerea sub tensiune sau prin program.

Dinamica unei diagrame Grafcet este următoarea: dacă o etapă este activă şi una din condițiile de tranziție ataşate este adevărată, etapa actuală se dezactivează şi se activează etapa următoare (sau etapele următoare, în cazul unui paralelism).

Un exemplu de diagramă Grafcet este prezentat în fig. 2. 6

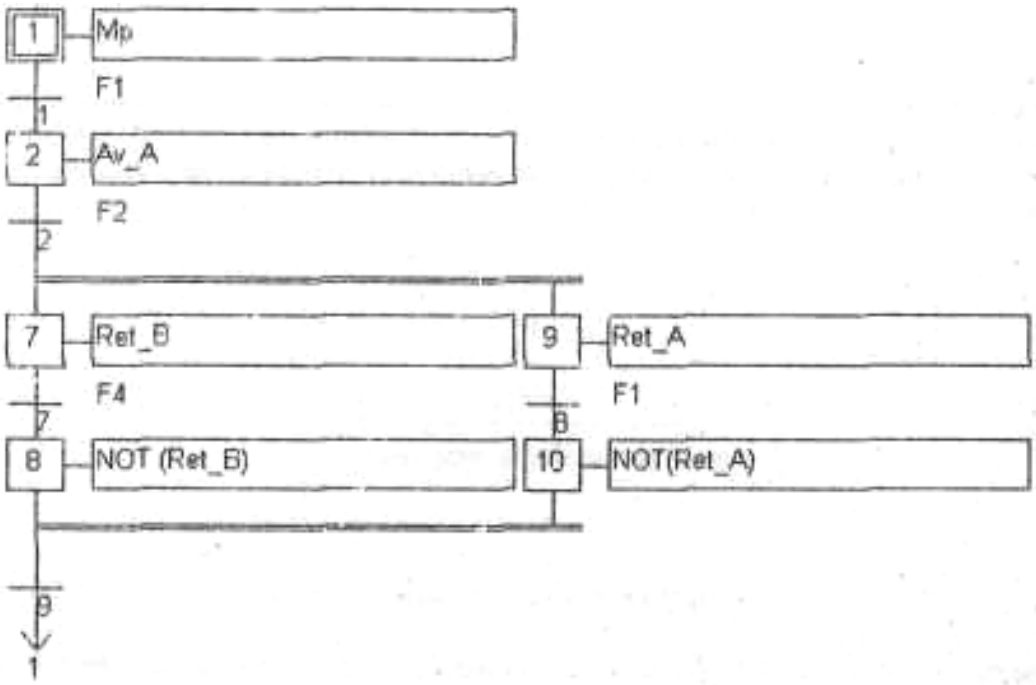


Fig. 2.6 Exemplu de diagramă Grafcet

2.2 Ladder Diagram

Cea mai răspândita modalitate de construire a unui program ce va fi executat de către un automat programabil este varianta construiri unei diagrame în „trepte", numită „Ladder diagram".

O astfel de diagramă este formată din ramuri, pe fiecare ramură existând instrucțiuni specifice automatului respectiv (fig. 2. 7). Întrucât instrucțiunile pot fi împărțite în două mari categorii, de intrare şi de ieşire, pe orice ramură trebuie să existe cel puțin o instrucțiune de ieşire. Fiecare instrucțiune are asociat un simbol grafic.

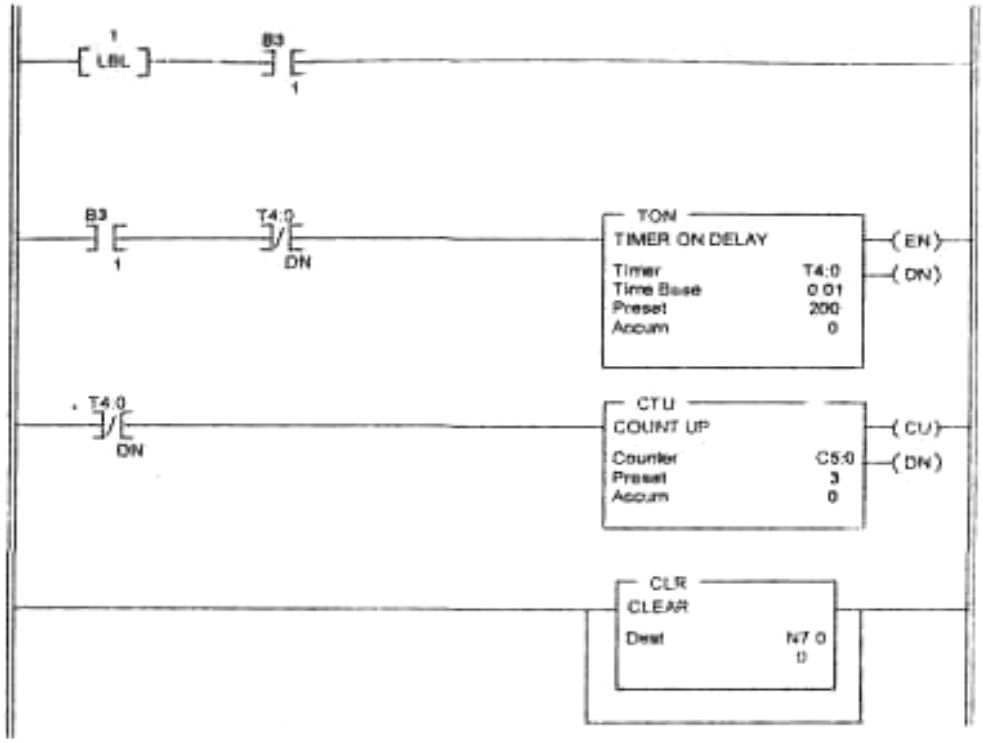


Fig. 2.7 Exemplu de diagramă ladder

Prima ramură din diagrama de mai sus nu este corectă, întrucât conține două instrucțiuni de intrare şi nici una de ieşire. Celelalte ramuri sunt corect editate, cu observația că ultima ramură are doar o instrucțiune de ieşire, situație întâlnită doar atunci când acea instrucțiune trebuie executata la fiecare ciclu automat.

De obicei programul este editat pe un calculator personal, sau cu ajutorul unui panou special (numit „Teach sau Handheld Pendant") ataşat automatului, apoi programul este transformat in instrucțiuni specifice procesorului de AMP şi descărcat în memoria automatului. În momentul execuției, automatul scanează programul ramură cu ramură, iar pe fiecare ramură execută instrucțiunile găsite de la stânga la dreapta. Programul este executat ciclic, deci se trece la scanarea primei ramuri după scanarea ultimei ramuri.

Există şi varianta rulării programului din memoria calculatorului, dar timpul pierdut cu transferul datelor între calculator şi controller poate afecta performanțele programului. Întrucât tipurile de instrucțiuni folosite într-o diagramă Ladder sunt practic aceleaşi indiferent de producătorul automatului programabil, ele vor fi descrise în capitolul dedicat automatelor de tip Allen Bradley.

2.3 Programarea în limbaje de nivel înalt

În ultima vreme a existat o puternică preocupare pentru integrarea performanțelor de conducere şi control a automatelor programabile cu posibilitățile de calcul şi de memorie ale calculatoarelor personale. De aceea majoritatea automatelor programabile posedă un port de comunicație cu un PC, de tip RS-232 sau RS-485, putându-se stabili o legătură cu PC-ul chiar în timpul execuției unui program de către AP.

De asemenea, pentru a profita de larga răspindire a limbajelor de nivel înalt, cum ar fi C sau Pascal, unii producători au echipat AP-urile cu procesoare compatibile Intel, acestea putând executa setul de instrucțiuni al procesoarelor Intel, extins cu instrucțiuni specifice automatelor programabile. În plus mulți producători au dezvoltat compilatoare C pentru procesoarele pe care le folosesc în automatele programabile.

Exista două variante de construire a unui program în limbaj de nivel înalt.

Prima variantă constă în crearea unui program în limbajul de programare C (Borland sau Microsoft), integrând funcțiile specifice ale automat programabil şi funcțiile de comunicație între calculator şi automat. În acest caz programul rulează în calculator, acesta dând permanent comenzi controllerului şi primind răspunsuri în legătură cu executarea acestora. Avantajul unui aslfel de program ar fi capacitatea mare de memorie şi puterea de calcul a unui calculator personal şi deci posibilitatea creării unor module software complexe folosind numeroase variabile.

Atunci când un număr mare de automate sunt legale împreună la un singur calculator, timpul necesar computerului să primească şi să proceseze toate informațiile de I/O poate produce scăderea semnificativă a răspunsului sistemului, făcând astfel dificil controlul procesului. Pentru a evita acest neajuns, programele trebuiesc implementate în fiecare automat şi rulate independent de computer.

Această capabilitate de partajare a acțiunilor permite utilizatorului să determine ce decizii de control vor fi luate de computer şi care de către automat. Această facilitate poate elibera calculatorul pentru a efectua operații de supervizare cum ar fi împrospătarea ecranului, generarea de rapoarte sau monitorizarea performantelor sistemului.

Din acest motiv reiese şi utililatea celei de-a doua variante, de altfel şi cea mai des uzitată, care constă în crearea unor programe de control ce pot rula direct în memoria automatului. Programele se construiesc şi se compilează în C pe un calculator personal, apoi programul executabil este convertit în formatul de fişier Intel HEX cu ajutorul unui program utilitar, după care este descărcat în memoria de tip FLASH EPROM a automatului, de unde poate rula de sine stătător. Un oarecare dezavantaj îl constituie limitarea dimensiunii codului şi a datelor programului.

Se obişnuieşte construcţia a două module software ce rulează simultan, unul în calculator şi celalalt în controller. Schimbul de informaţii între cele două dispozitive se poate face prin intermediul unor structuri de date definite în ambele module, acestea putând fi citite/scrise cu funcţii specifice.

Setul de instrucţiuni C specifice automatului programabil de tip Microdac

Pentru a putea înţelege cum se poate construi un pachet software de control în limbajul C, va fi prezentată pe scurt o parte a colecţiei de funcţii specifice lucrului cu modulele de intrare/ieşire pentru un automat programbil de tip Microdac, construit de firma Grayhill. Aceste funcţii sunt implementate cu întreruperi disponibile din Microdac [Grayhill, 1995].

- Uint md_read_dio_status(Uchar group) citeşte starea de ON/OFF a unui grup de module digitale
- char md_read_dio_point(Uchar point) citeşte starea de ON/OFF a unui singur modul digital
- void md_write_dout(Uint value 0, Uint value l) înscrie toate ieşirile digitale (max. 32) printr-un singur apel
- char md_activate_dout_point (Uchar point) activează o singură ieşire digitală
- char md_deactivate_dout_point (Ucaher point) deactivează o ieşire digitală
- Uint md_read_ain (Uchar chanel) citeşte valoarea unei intrări analogice
- void md_write_aout(Uchar chatmel, Uint value) setează ieşirea analogică channel cu valoarea value
- void md_delay (Uint msec) opreşte execuţia programului pentru un număr dat de milisecunde
- Ulong md_get_counter (void) întoarce valoarea timer-ului
- void md_set_counter (Ulong counter) setează timer-ul cu valoarea count
- Uchar md_get_ctr_res (void) întoarce rezoluţia curentă a timer-ului care poate fi 1ms sau 5Oμs

Aceste funcţii sunt definite într-o bibliotecă specifică Mdac.lib şi nu pot fi apelate decât din automat. Există funcţii analogice, definite în fişierul header „proware.h" ce pot fi apelate din programul calculatorului, dar viteza de execuţie este evident mult mai scăzută. Schimbul de date cu calculatorul este realizat prin apelarea unei funcţii speciale.

proware(&err, &addr, &cmd, posn, mod, pointer)

CAP. 3 AUTOMATELE PROGRAMABILE ALLEN-BRADLEY

În capitolul ce urmează vor fi prezentate pricipalele caracteristici ale automatelor programabile produse de firma Allen Bradley, întrucât aceste tipuri de automate sunt recunoscute pentru performanţele şi fiabilitatea lor, ocupând un important segment de piaţă în domeniul sistemelor de control industrial.

Structura şi programarea acestor automate va fi prezentată în mod particular pentru modelul SLC500, de uz didactic, principalele caracteristici fiind însă comune tuturor automatelor produse de firma Allen Bradley [Allen-Bradley, 1992].

Automatul SLC500 prezintă o structură standard de automat programabil, cu menţiunea că modulele de intrare-ieşire au în componenţă, în general 16 circuite de intrare sau de ieşire, mai multe module putând fi montate pe rack-uri având un număr standard de sloturi, 4, 6 sau 10 module putând fi ataşate unităţii centrale.

Sloturile sunt numerotate, slotul 0 fiind întotdeauna rezervat unităţii centrale, celelalte fiind atribuite modulelor de intrare / ieşire. Numărul slotului este important pentru adresarea corectă a intrărilor şi ieşirilor.

Automatul dispune de o interfaţă serială de tip RS-232 prin care poate fi cuplat la un calculator personal şi o interfaţă de tip RS-485 prin care pot legate mai multe automate într-o reţea de automate.

Editarea programelor se face cu ajutorul calculatorului personal, prin intermediul unui mediu de programare integrat care permite şi transferul programului în memoria automatului, de unde va putea rula. Programarea se face prin metoda Ladder Diagram, realizarea unui program necesitând cunoaşterea organizării memoriei acestui tip de automat.

3.1 Organizarea memoriei şi moduri de adresare

La baza adresării si organizării memoriei stă noţiunea de file (fişier). Memoria este împărţită în fişiere, care sunt de 2 tipuri:

- fişiere program
- fişiere de date

Fiecare program creat pentru acest automat are asociat atât fişiere program, cât şi fişiere de date.

Fişierele program sunt numerotate si pot fi maxim 256. Implicit sunt create doar 3 fişiere program şi anume:

- fişierul 0 ce cuprinde informaţii referitoare la configuraţia hardware a automatului
- fişierul 1 este rezervat
- fişierul 2 conţine diagrama Ladder realizată de către utilizator

Celelate posibile fişiere program sunt create numai în situaţia în care utilizatorul foloseşte subrutine.

Fişierele de date conţin informaţii asociate cu intrările şi ieşirile (externe) precum şi cu toate celelalte instrucţiuni care compun diagram Ladder. În plus, ele mai conţin informaţii cu privire la operaţiile procesorului. Fişierele de date sunt numerotate şi pot fi maxim 256. Implicit sunt create primele 10 fişiere.

Structura acestor fişiere de date este prezentată mai jos:

0: Imagine ieşiri :	conţine valorile intrărilor în format binar
1: Imagine intrări :	conţine valorile ieşirilor în format binar
2: Stare :	conţine informaţii de stare
3: Bit :	utilizat pentru variabile booleene
4: Timer :	utilizat în instrucţiuni de timer

5: Counter : utilizat în instrucțiuni de counter
6: Control: utilizat în instrucțiuni de stivă, shiftare, secvențiere etc.
7: Întregi : folosit pentru memorarea operanzilor de tip Integer
8: Rezervat
9: Folosit în rețea
10 - 255 : pot fi create de utilizator atunci când fişierele create implicit nu sunt suficiente şi pot fi de tipul Bit, Timer, Counter, Control sau Integer.

În scopul adresării, diferența între fişierele menționate mai sus se face folosid un indicator de fişier (o literă) şi un număr ai fişierului. Modul de alocare al identificatorului pentru fiecare tip de fişier este următorul:

Tabel 3 1 Adresarea filierelor de date

Tip fişier	Identificator	Număr fişier
a-pentru fişierele predefinite		
Output	0	0
Input	I	1
Stare	S	2
Bit	B	3
Timer	T	4
Counter	C	5
Control	R	6
Integer	N	7
b-pentru fişierele definite de ulitlizator		
Bit	B	9-255 fişiere adiționale
Timer	T	
Counter	C	
Control	R	
Integer	N	

Fişierele de date conțin elemente. Elementele pot fi de 1 cuvânt sau de 3 cuvinte. Un cuvânt are 16 biți. În consecința este necesară adresarea la nivel de :

I) element
II) cuvânt
III) bit

In cele ce urmează va fi exemplificat fiecare din cele trei tipuri de adresare :

- Adresarea la nivel de element: N7:15
  unde : N - tip fişier, 7 - număr fişier,: - delimitator de element, 15 - număr element
- Adresarea la nivel de cuvânt: T4:7 .ACC
  unde: T7:4 au aceeaşi semnificație ca în cazul I), - delimitator de cuvânt, ACC - cuvânt adresat.
- Adresarea la nivel de bit: B3:64/15
  unde : B3:64 au aceeaşi semnificații ca în cazul II), /-delimitaor de bit, 15-bitul adresat.

3.2 Structura internă a fişierelor de date implicite

a) Fişierele de imagine a intrărilor şi ieşirilor

Biţii din aceste fişiere corespund ieşirilor şi intrărilor fizice ale automatului.

Formatul general de adresare pentru aceste fişiere este:
O:s/b , respectiv **I:s/b**

Unde:
O - ieşire
I - intrare
s - număr slot pe care se găseşte modulul de I/O
/ - delimitator de bit
b - numărul intrării adresate

Ex:exmplu:
- **O:4/6** : ieşirea 6 de pt slotul 4
- **1:1/7** : intrarea 7 de pt slotul 1

b) Fişierele de tip Bit

Formatul general de adresare pentru acesle fişiere este:
Bf:e/b sau **Bf/b** unde

B - identifica tipul fişierului
f - numărul fişierului
e - cuvântul din cadrul fişierului
b - bitul adresat

Exemplu:
B3:1/5 - bitul 5 din cuvântul 1 al fişierului de bit B3
B3/18 - bitul 18 al fişierului B3

c) Fişiere de tip Timer

Aceste fişiere conţin elemente de **3 cuvinte**, având maxim 256 de elemente. Un element tipic are structura din tabelul 3. 2:

Tabel 3. 2 Element al fişierului de Timer

15	14	13	12 | 11 10 | 9 8 | 7-0	Cuvânt
EN	TT	DN	Uz intern	0
PRE				1
ACC				2

Semnificaţia elementelor va fi prezentată în cadrul instrucţiunilor de Timer. Formatul de adresare este : Tf:e.s sau Tf:e/b, unde:
T - identifică tipul fişierului (Timer)
f - numărul fişierului
e - elementul din cadrul fişierului
s - cuvântul din cadrul elementului
b - bitul adresat

Exemple:

- T4:0.ACC - cuvântul ACC din elementul 0 al fişierului de Timer 4
- T4:1/DN - bitul DN din elementul 1 al fişierului de Timer 4

d) Fişierele de tip Counter

Aceste fişiere conţin elemente de 3 cuvinte, având maxim 256 de elemente. Un element tipic are structura din tabelul 3. 3:

Tabel 3.3 Element al fisierului de Counter

15	14	13	12	11	10	9 | 8 |7-0	Cuvânt
CU	CD	DN	OV	UN	UA	/Uz intern	0
PRE							1
ACC							2

cu semnificaţia:

- CU = Count Up
- CD = Count Down
- DN = Done
- OV = Overflow
- UN = Underflow
- UA = Update Accumulated Value

Formatul general de adesare este: Cf:e.s sau Cf:e/b. semnificaţiile fiind cele de la fişierele de tip Timer.

Observaţie: Pentru fişierele care au o anumită notaţie pentru biţi sau cuvinte constituente, adresarea se poate face folosind notaţia respectiva. Astfel, de exemplu, C5:0.PRE <=> C5:0.1.

e) Fişierele de tip Control

Aceste fişiere conţin elemente de 3 cuvinte, având maxim 256 de elemente. Un element tipic are structura din tabelul 3. 4:

Tabel 3. 4 Element al fişierului de Control

15	14	13	12	11	10	9	8	7-0	Cuvânt	Semnificaţie
EN	EU	DN	EM	ER	UL	IN	FD	Uz intern	0	cuvânt de stare
Lungimea ariei de biţi sau a fişierului (LEN)									1	cuvânt ce indică lungimea datelor stocate
Pointer de bit sau poziţie (POS)									2	

cu semnificaţia:

- EN = Enable
- EU = Unload Enable
- DN = Done
- EM = Stack Empty
- ER = Error
- UL = Unload
- IN = Inhibit
- FD = Found

Formatul general de adresare este Rf:e.s sau Rf:e/b cu semnificaţiile deja cunoscute

f) Fişierele de tip Integer

Acest tip de fişier conţine elemente de 1 cuvânt, maxim 255 de elemente.
Formatul de adresare este **Nf:e,** unde e = 255 reprezintă numărul elementului

3.3 Programarea automatului Allen Bradley prin metoda Ladder Diagram

Programarea automatului Allen-Bradley SLC 500 se face folosind metoda Ladder Diagram, explicată în capitolul anterior. Programul încărcat in memoria automatului conţine instrucţiuni; o parte din aceste instrucţiuni reprezintă lucrul cu dispozitive de intrare/ieşire externe.

Pe măsură ce programul este scanat de automat (controller), modificările on/off ale stărilor intrărilor externe se aplică programului, dezactivând ieşiri externe, conform logicii Ladder programate. Pentru a prezenta bazele programării Ladder am ales instrucţiuni pe bit (cu logică tip releu). Cele trei instrucţiuni utilizate în continuare sunt:

Tabel 3.5 Instrucţiuni pe bit

SIMBOL Ladder	SEMNIFICAŢIE ŞI MNEMONICA
---] [---	XIC (Examine if Closed) -procesorului i se cere sa examineze dacă (contactul este) deschis (contactul normal închis)
---]/[---	XIO (Examine if Open) -procesorului i se cere să examineze daca (contactul este) închis (contactul normal deschis)
---( )---	OTE (Output Energize) -procesorul setează aceasta instrucţiune la valoarea adevărat atunci când există o secvenţă de instrucţiuni XIC si XIO adevărate în câmpul de intrări din ramura Ladder respectivă.

Constricţia programelor folosind metoda Ladder-diagram

O diagramă Ladder este formată dinrr-un anumit număr de ramuri, fiecare conţinând cel puţin o insiucţiune de ieşire şi una sau mai multe instrucţiuni de intrare. Fiecare instrucţiune are ataşată o valoare logică, TRUE sau FALSE.

Ramura Ladder prezentată în exemplul următor conţine două instrucţiuni de intrare şi o instrucţiune de ieşire. O instrucţiune de ieşre este situată în dreapta ramurii, lângă bara verticală dublă dreapta. Instrucţiunile de intrare sunt întotdeauna situate la stânga instrucţiunilor de ieşire.

Ex: Instrucţiuni de intrare Instrucţiuni de ieşire

XIC = Examine if Closed - examinează bitul de la adresa B3/10
XIO = Examine if Opened - examinează bitul de la adresa B3/11
OTE = Output Eenergize - aetiveză bitul de la adresa B3/12

Fiecare instrucțiune din ramura precedentă are adresa fişierului unde sunt stocate stările ON(1) / OFF(0) ale biților. Adresele instrucțiunilor de mai sus indică fişierul Bit 3 (B3), biții 10, 11, 12.

Parcurgerea unei astfel de diagrame se face de sus in jos (ramură cu ramură) şi de la stânga la dreapta (în cadrul unei ramuri). Execuția instrucțiunilor se face prin testarea condițiilor logice ale ramurilor.

Starea logică a unei instrucțiuni poate fi adevărat (TRUE T) sau fals (FALSE F). Ansamblul stărilor logice ale instrucțiunilor de intrare determină continuitatea logică a ramurii. În funcție de stările logice ale datelor stocate în biții adresați, stările logice ale instrucțiunilor pot fi:

Tabel 3. 6 Stările logice ale instrucțiunilor din ramura

Stare logica bit	Stare logica a instructiunilor		
	XIC	XIO	OTE
0 logic	Fals (F)	Adevărat (T)	Fals (F)
1 logic	Adevărat (T)	Fals (F)	Adevărat (T)

În exemplul precedent, pentru a executa instrucțiuitea de ieşire OTE (care va avea ca efect setarea bitului B3/12), trebuie să nu se întrerupă continuitatea logică pe ramură, adică trebuie ca starea XIC = TRUE ^ starea XTO = TRUE. Concret, conform tabelului precedent:

a) Dacă B3/10 = 1 şi B3/11 = 0, deci dacă este găsită o cale continuă de instrucțiuni de intrare adevărate, atunci instrucțiunea de ieşire devine sau rămâne adevărată.
În acest caz condițiile ramurii sunt adevărate.

b) Când NU este găsită această continuitate logică a instrucțiunilor de intrare, atunci instrucțiunea de ieşire OTE devine sau rămâne falsă.
În acest caz condițiile ramurii sunt false.

Modul de variație al stării bitului B3/12 în funcție de stările biților B3/10 si B3/11 este:

Tabel 3. 6 Variația valorii bitului B3/12

B3/10	B3/11	B3/12
0	0	0
1	0	1
1	1	0
1	1	0

O posibilă variație în timp a stărilor logice ale instrucțiunilor XIC, XIO şi OTE este:

Tabel 3. 7 Variație marilor Ionice ale Instrucțiunilor

Timp	Instr. de intrare		Instr. de ieşire
tl(initial)	F	T	F
t2	T	T	devine T
t3	T	F	devine F
t4	F	F	rămâne F

Având în vedere cele prezentate se poate trage concluzia că o ramură din diagrama Ladder este echivalentă cu o instrucțiune de tipul:

If condiție **then** ieşire.

Câmpul condiție din model poate însemna:
1) testarea unei intrări sau
2) testarea unei anumite combinații logice între variabilele de intrare.

În exemplul precedent s-a prezentat un bloc AND (şi logic), între valoarea bitului B3/10 şi valoarea negată a bitului B3/11. În cazul general un bloc AND reprezintă un bloc de tipul

Dacă A şi B sunt adevărate atunci C devine sau rămâne adevărată.

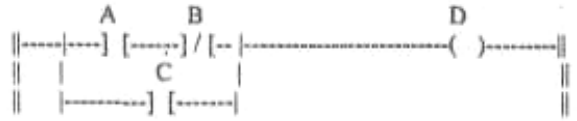
SAU (OR) logic poate fi format, de exemplu, prin ramificarea instrucțiunilor.

Tabela de adevăr pentru acest tip de bloc este prezentată în tabelul 3. 8:

Tabel 3. 8 OR logic inlre 2 instrucțiunide intrare

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

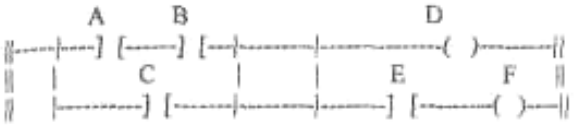
Pentru a mări flexibilitatea realizării programelor, există posibilitatea de a introduce expresii logice complexe pe o ramură, mărind astfel complexitatea blocului logic generat:



D = (A AND B) OR C

În mod asemănător poate avea loc şi o ramificare a instrucțiunilor de ieşire. De asemenea se poate impune şi o condiție suplimentară pentru emisia unei anumite ieşiri.

Ex:



F = ((A AND B)) OR C) AND E
D = (A AND B) OR C

În exemplul precedent, dacă (A AND B) AND E sau C AND E sunt adevărate , atunci F este adevărată.

3.4 Tipuri de instrucțiuni

Pentru procesoarele folosite de către automatul SLC500, instrucțiunile folosite în diagramele Ladder se împart în:

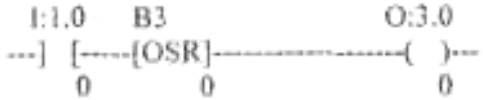
a) instrucțiuni pe bit
b) instrucțiuni de timer/counter
c) instrucțiuni de I/O şi întreruperi
d) instrucțiuni de comparare
e) instrucțiuni matematice
f) instrucțiuni logice şi de mutare
g) instrucțiuni pentru lucrul cu fişiere
h) instrucțiuni de shiftare
i) instrucțiuni de secvențiere
j) instrucțiuni de control
k) blocuri funcționale

În continuare sunt prezentate cele mai utilizate tipuri de instrucțiuni într-o diagramă Ladder

- Instrucțiuni pe bit
  - XIC --] [-- (Examine If Dosed). Instrucțiune de intrare. Este TRUE când bitul testat este 1 (on)
  - XIO --] / [-- (Examine if Open). Instrucțiune de intrare. Este TRUE când bitul testat este 0 (off)
  - OTE --( )-- (Output Energize). Instrucțiune de ieşire. Este TRUE (setează bitul de ieşire) când toate condițiile precedente din ramură sunt TRUE. Resetează bitul altfel.
  - OTL --(L)-- (Output Latch). Instrucțiune de ieşire. Bitul adresat devine TRUE (1) când condițiile precedente de pe ramură sunt TRUE. Când condițiile devin FALSE, OTL rămâne TRUE până când o ramură ce conține o instrucțiune OTU cu aceeaşi adresă devine TRUE.
  - OTU --(U)-- (Output Unlatch). Instrucțiune de ieşire. Bitul adresat devine FALSE (0) când condițiile precedente de pe ramură sunt TRUE. Rămâne FALSE până când o altă ramură ce conține o insirucțiune OTU cu aceeaşi adresă devine TRUE.
  - OSR --[OSR]-- (One-Shot Rising). Instrucțiune de intrare. Trece ramura în TRUE pentru o scanare, la fiecare tranziție din FALSE-TRUE a condițiilor precedente din ramură.

Toate instrucțiunile pe bit necesită ca parametru adresa unui bit dintr-un fişier de date. OSR este o instrucțiune de intrare utilă activării unei instrucțiuni de ieşire o singură data şi anume la prima scanare a programului de către automat. Este des folosită pentru activarea unei etape inițiale dintr-o diagramă Grafcet (vezi problemele rezolvate). Necesită ca parametru adresa unui bit care va fi folosit intern de către instrucțiune.

Exemplu :



Când instrucțiunea de intrare trece din FALSE (o scanare) în TRUE (următoarea scanare), instrucțiunea OSR condiționează ramura astfel încât ieşirea trece în TRUE pentru o scanare a programului, apoi ieşirea devine sau rămâne FALSE pentru următoarele scanări, până când intrarea face o noua tranziție din FALSE în TRUE.

- *Instrucţiuni de Timer/Counter*
  - TON (Timer On-Delay). Instrucţiune de ieşire. Numără intervalele de timp cât timp condiţiile precedente din ramură sunt TRUE. Necesită ca parametri adresa unui element dintr-un fişier de tip Timer, cu valoare presetată (număr de intervale de timp) şi o valoare acumulată iniţial (de obicei 0). Valoarea acumulată este incrementată la fiecare scanare a programului dacă condiţiile din ramură sunt TRUE. Bitul DN devine 1 când valoarea acumulată egalează valoarea presetată, (ACC) = (PRE). Când condiţiile devin FALSE, acumulatorul este resetat: (ACC) = 0. Simbolul grafic este prezentat în figura 3. 1.

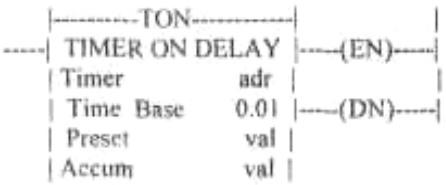


Fig. 3.1 Simbolul grafic al instrucţiunii TON

  - TOF (Timer OfT-Delay). Instrucţiune de ieşire. Analog cu TON, dar incrementarea valorii din acumulator are loc atâta timp cât în ramură condiţiile sunt FALSE. Când condiţiile devin TRUE, (ACC) = 0.
  - RTO (Retentive Timer). Instrucţiune de ieşire. Analog cu TON, dar dacă condiţiile de pe ramura trec din TRUE în FALSE înainte ca (ACC) = (PRE), atunci (ACC) rămâne la ultima valoare.
  - CTU (Count Up). Instrucţiune de ieşire. Numără tranziţiile din FALSE în TRUE ale ramurii, incrementând valoarea acumulată. Numărătoarea se opreşte când condiţiile devin FALSE. Counter-ul trebuie resetat cu o instrucţiune RES, cu aceeaşi adresă. Când (ACC) > (PRE) bitul DN devine 1. Necesită ca parametri adresa unui element dintr-un fişier de Counter precum şi o valoare presetată şi una iniţială pentru acumulator.

Fig. 3.2 Simbolul grafic al instrucţiunii CUP

  - CTD (Count Down). Instrucţiune de ieşire. Numără tranziţiile din TRUE în FALSE ale ramurii, incrementând valoarea acumulata. Counter-ul trebuie resetat cu o instrucţiune RES, cu aceeaşi adresă. Când (ACC) > (PRE) bitul DN devine 1. Necesită ca parametri adresa unui element dintr-un fişier de Counter precum şi o valoare presetată şi una iniţială pentru acumulator.
  - RES (Reset). Instrucţiune de ieşire. Resetează un anumit Timer sau Counter.
    Timer : ACC ← 0. DN ← 0, TT ← 0, EN ← 0.
    Counter : ACC ← 0, CU ← 0, CD ← 0, OV ← 0, UN ← 0, DN ←0.
  - HSC (High Speed Counter). Numărătorul de mare viteză HSC este un numărător hardware şi operează asincron faţă de scanarea programului ladder. Spre deosebire de celelalte numărătoare USC nu contorizeasă tranziţiile FALSE-TRUE ale ramurii ci tranziţiile intrării i:0/0. Frecvenţa maximă acceptată este de 8 kHz

Această instrucţiune este valabilă numai pentru controlere I/O fixe cu 24 intrări VDC. O singură instrucţiune HSC este permisă pe un singur controler.

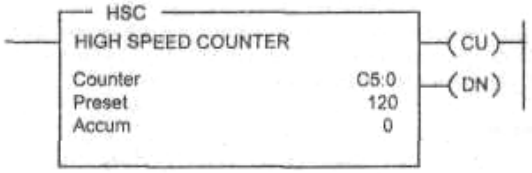


Fig. 3.3 Simbolul grafic al instrucțiunii HSC

Această instrucțiune are asociat un fișier de 3 cuvinte (octeți). Cuvântul 0 este cuvântul de control şi conține biții de stare:

- bitul 10 (UA) actualizează acumulatorul pentru a reflecta starea imediată a numărătorului când HSC devine activă;
- bitul 12 (OV) indică o depăşire a numărătorului;
- bitul 13 (DN) indică egalarea presetului de către acumulator;
- bitul 15 (CU) arată starea de activ/inactiv a instrucțiunii HSC.

Cuvântul 1 conține valoarea de preset în gama 1-32.767, iar cuvântul 2 valoarea acumulată.

Funcționare:
- fiecare tranziție a intrării 1:0/0 face ca acumulatorul să se incrementeze;
- când acumulatorul egalează valoarea presetului bitul Done (C5:0/DN) este setat şi acumulatorul pus pe zero;

Pentru a afla starea HSC-ului, programul ladder trebuie să interogheze bitul Done (C5:0/DN). Imediat cum acesta a fost detectat activ trebuie dezactivat (cu instrucțiunea OTU) înainte ca acumulatorul (C5:0. ACC) să egaleze valoarea presetului (C5:0. PRE) pentru ca altfel bitul Overflow (C5:0/OV) este setat.

Valoarea acumulatorului (C5:0. ACC) de obicei este actualizată de fiecare dată când ramura cu HSC este evaluată, actualizarea însemnând transferarea valorii acumulatorului hardware în acumulatorul HSC software. După actualizare instrucțiunea HSC resetează bitul C5:0/UA.

Dacă se doreşte testarea C5:0. ACC în programul ladder după rung-ul care conține instrucțiunea HSC, trebuie mai intâi să se seteze bitul C5:0/UA. Acest lucru este necesar înlrucât intrarea 1:0/0 poate să tranziteze de câteva ori de la ultima evaluare a lui HSC.

o *Instrucțiuni de comparare*

Sunt instrucțiuni de intrare.

- EQU (Equsil), Compara (A) cu (B), unde A si B sunt adrese. Dacă (A) = (B), atunci instrucțiunea devine TRUE.
- NEQ (Not Eqnal). Devine TRUE dacă (A) diferit de (B).
- LES (LessThan) Devine TRUE daca (A) < (B).
- LEQ (Less Than or Eqnal) Devine TRUE dacă (A) =< (B),
- GRT (Greater Than) Devine TRUE dacă (A) > (B).
- GEQ (Greater than or Equal) Devine TRUE daca (A) => (B).
- MEQ (Masked Comparison for Equal). Permite testarea egalității după aplicarea unei măşti (poate fi o valoare hexa).

Parametrii necesari acestor instrucțiuni sunt:

- Source = adresa valoare
- Compare = o constantă întreagă sau adresa unei referințe.
- LIM (Limit Test) Permite testarea încadrării unei valori între două limite. Low Limit = val. /adr limită inferioara Test = vaLadr. val. lest. High Limit = val./adr. limită superioară. Este TRUE când Low Lini =< High Lim.

o *Instrucțiuni matematice*

O instructiue matematică acceptă doi operanzi, dintre care maxim unu poate fi o constantă. Sunt instrucțiuni de ieşire

- ADD (ADD) Dest = A + B
- SUB (Subtract) Dest = A-B
- MUL (Multiply) Dest = A * B
- DIV (Divide) Dest = A/B
- NEG (Negate) Dest = (complement față de 2) Sursa
- CLR (Clear) Dest ← 0
- SQR (Square Root) Dest = $\sqrt{Sursa}$
- SCL (Scale Data) Dest ←(Sursa): Rate.

o *Instructiuni logice şi de mutare*

Sunt instrucțiuni de ieşire. Instrucțiunile logice se aplică bit ci bit datelor stocate Ia adresele sursă. Când condițiile de pe ramură sunt TRUE, atunci :

- MOV (Move) (Dest) ← (Sursa): .
- MVM (Masked Move) Analog cu MOV, dar după ce sursa a fost trecută prin Mask (mască, mişca poate fi o valoare hexa (o constantă).
- AND (And) (Dest) = A AND B
- OR (Or) (Dest) = A OR B
- XOR (Exclusive Or) (Dest) = A XOR B

o *Instrucțiuni de lucru cu stiva*

Instrucțiunile FFL (FIFO Load) şi FFU (FIFO Untoad) se folosesc împreună. Instrucțiunea FFL încarcă octeți într-un fişier creat de utilizator numit stivă FIFO. Instrucțiunea FFU descarcă octeți din stiva FIFO în aceeaşi ordine în care au fost introduşi.

Pentru folosirea acestor instrucțiuni sunt necesari următorii parametrii:

- **Source** reprezintă o adresă de cuvânt sau o constantă (-32, 768 până la 32. 767) în care se află valoarea de introdus în stiva FIFO.
- **Destination** reprezintă o adresă de cuvânt care memorează valoarea extrasă din stiva FIFO.
- **FIFO** reprezintă adresa stivei. Trebuie să fie o adresă indexată de cuvânt dintr-unul din fişierele de intrare, ieşire, stare, bit, sau întregi. Pentru ambele instrucțiuni FFL şi FFU trebuie să avem aceeaşi stivă.
- **Length** reprezintă numărul de elemente din stivă (maxim 128). Ambele instrucțiuni FFL şi FFU trebuie să aibă acelaşi număr de elemente.
- **Position** reprezintă următoarea locație disponibilă din stiva unde se poat încărca date. Această valoare se schimbă după fiecare operație de încărcare sau descărcare. Acelaşi număr este folosit pentru ambele instrucțiuni FFL şi FFU.
- **Control** reprezintă adresa fişierului de control, fişier în care sunt stocate date despre biții de stare, lungimea stivei şi valoarea poziției. Această adresă nu trebuie folosită pentru nici o altă instrucțiune.

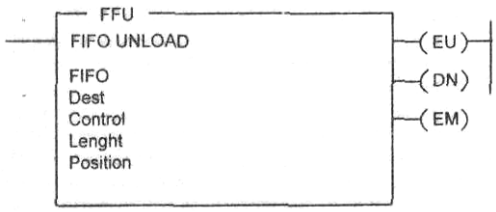


Fig. 3.4 Instrucțiunea de extragere din stivă

Biții de stare din elementul de control sunt folosiți astfel:

- Bitul Empty EM (bitl2) este setat de instrucțiunea FFU indicând că stiva e goală
- Bitul Done DN (bit 13) este setat de instrucțiunea FFL pentru a indica faptul că stiva e plină. Aceasta inhibă încărcarea stivei
- Bitul FFU Enable (bit 14) este setat la tranziția din fals în adevărat a FFU şi resetat la tranziția inversă
- Bitul FFL Enable (bit 15) este setat la tranziția din FALSE în TRUE a ramurii FFL şi resetat la tranziția inversă.

o Blocul funcțional PID

Un bloc funcțional înseamnă un set de instrucțiuni (invizibile utilizatorului) care, având cunoscut un anumit număr de parametri de intrare, sunt capabile să obțină nişte valori de ieşire, calculate după un anumit algoritm, care nu este transparent utilizatorului. Utilizatorul poate doar să configureze blocul funcțional introducând valorile parametrilor de intrare.

PID este un bloc funcțional apelat ca o instrucțiune de ieşire cu ajutorul căreia se pot conduce procese după temperatură, presiune, nivel sau debit folosind bucle de reglare. Instrucțiunile PID controlează în mod normal o buclă închisă folosind intrările de la un modul analogic şi furnizând nişte ieşiri pentru un modul analogic de ieşire.

Aceste instrucțiuni pot opera în modul eşantionat sau în modul STI (Selectable Timed Intettupts - Întreruperi de Timp Reglabile). În modul eşantionat, instrucțiunile actualizează periodic ieşirile la o rată ce poate fi aleasă. În modul ST, instrucțiunilie trebuie să fie plasate în subrutine de întrerupere STI; ieşirile vor fi actualizate în momentele de scanare ale STI. Intervalui STI şi rata de actualizare a buclei trebuie să fie la fel pentru a executa corect ecuația. Blocu funcțional PID (flg. 3. 5) foloseşte următoarea formulă:

Ieşire = $K_C[(E)+1/T_1\int(E)dt+T_D D(PV)/dt]+deviatie$

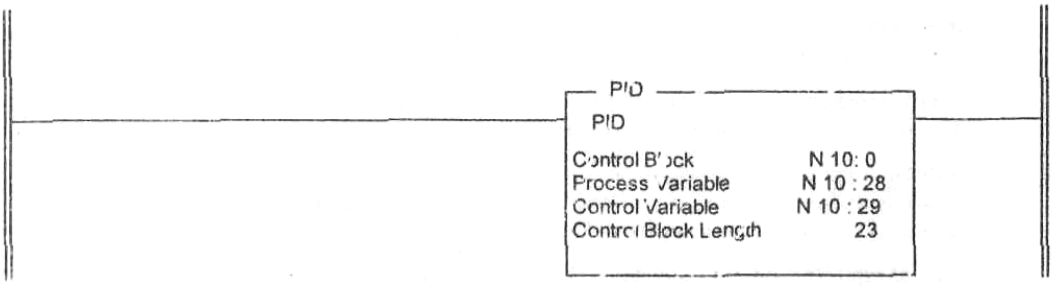


Fig. 3.5 Blocul funcțional PID

Instrucțiunile PID se pun pe o ramură fără conditionari logice. Dacă ramura nu este parcursă ieşirile rămân la valorile anterioare, iar termenul integral este resetat. În timpul programării se introduc adresele Blocului de Control, a Variabilelor de Proces, a Variabilelor de Control după ce instrucțiunile PID au fost puse pe ramură:

- **Blocul de Control** este un fişier, care memorează datele necesare instrucțiunior pentru operare. Mărimea fişierului este fixata la 23 de cuvinte şi poate fi introdusa ca o adresă de fişier pentru întregi. De exemplu: N10:0 va aloca elementele de la N10:0 până ia N 10:20.

- **Variabila de Proces PV** este adresa unui element care stochează valoarea intrării din proces. Aceasta adresă poate fi un cuvânt rezervat unei intrări analogice unde va fi stocată valoarea respectivei intrări. Această valoare poate fi un întreg dacă se alege scala 0-16383.
- **Variabila de Control CV** este adresa unui element care stochează valoarea ieşirii instrucţiunii PID. Domeniul de ieşire este 0-16383 (16383 corespunde unui nivel de 100 din ieşire)

Tabel 3. 8 Structura blocului de control

15 14 13 12 11 10 09	08 07 06 05 04	03 02 01 00	
EN DN FV SP LL UL	DB DA TF SC RG	OL CM AM TM	0
* PID Sub Error Code (MSbyte)			1
* Setpoint SP			2
* Gain Kc			3
* Reset Ti			4
* Rate Td			5
* Feed Torward Bias			6
* Setpoint Max (Smax)			7
* Setpoint Hin (Smin)			8
* Deadband			9
INTERNAL USE DO NOT CHAMGE			10
* Output Max			11
* Output Min -			12
* Loop Update			13
*Scaled Process Variable			14
*Scaled Error SE			15
*Output CVS (0-100)			16
*LSH Integral Sum	5/03 MSW Integral Sum		17
*MSW Integral Sum	5/03 LSW Integral Sum		18
INTERNAI. USE DO NOT CHANGE			19
			20
			21
			22

După ce au fost introduse adresele pentru CB, PV şi CV, va apare următorul ecran cu ajutorul căruia se vor introduce datele necesare configurării şi acordării modulului.

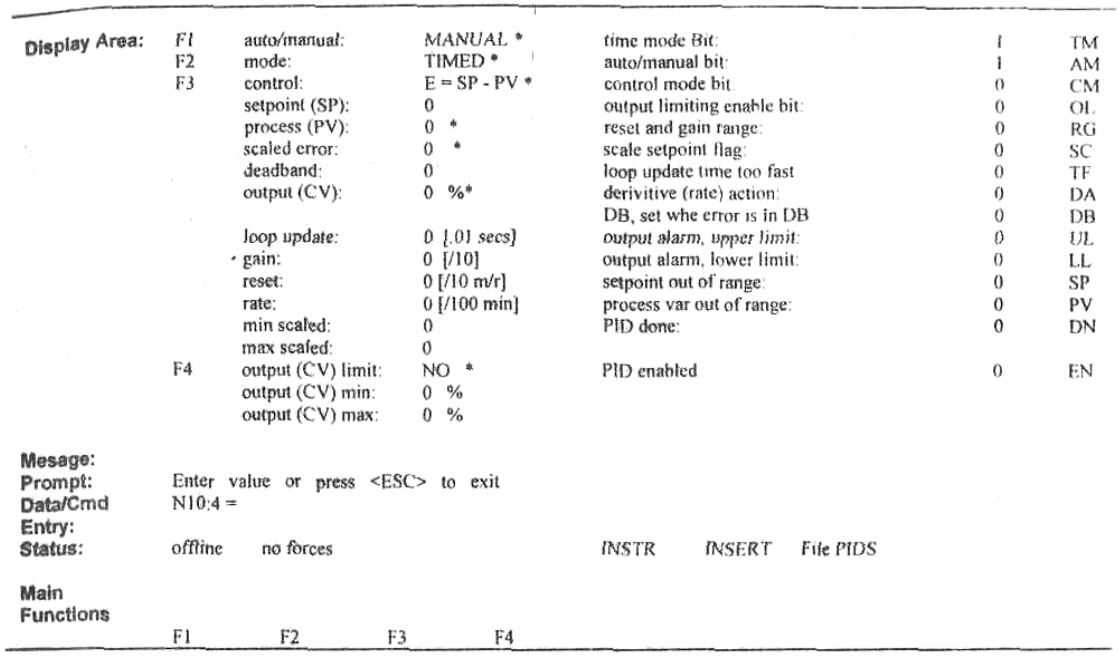


Fig. 3.6 Ecranul de introducere a datelor pentru instrucțiunea PID

In partea stângă sunt parametri instrucțiunii PID care trebuie introduşi:

- **[FI] Auto/Manual AM** (word 0 bit 1) se face alegerea între Auto şi Manual, Auto arată că PID controlează ieşirile automat Manual arată câ utilizatorul controlează ieşirile (setează);
- **[F2] Mode TM** (word 0, bit 0) se face alegerea între modurile eşantionat şi STI. ;
- **[F3] Control CM** (word 0, bit 2) se face alegerea între E = SP-PV şi E = PV-SP. Varianta E = PV-SP presupune incrementarea ieşirii CV când intrarea PV > SP, iar varianta E = SP-PV presupune incrementarea CV când intrarea PV < SP.
  - **Setpoint SP** (word 2) este punctul de control dorit pentru proces. Se introduce valoarea dorită şi se tastează ENTER. Valoarea se poate schimba în programul ladder prin instrucțiuni;
  - **Gain K_c** (word 3) este factorul de amplificare. Proporțional, în gama 0.1-25.5. O regulă este setarea lui la o valoare egală cu 1.5 din valoarea necesară pentru a determina la ieşire oscilații când reset = 0 şi rate terms = 0;
  - **Reset T_i** (word 4) este factorul de timp al integratorului, având valori în gama 0. 1-25.5 min. O regulă este setarea lui la perioada naturală măsurată înaintea calibrării;
  - **Rate T_d** (word 5) este termenul derivativ, putând lua valori în gama 0.01-2.55 min. O regulă este setarea lui la l/8 din T_i; .
  - **Maximum Scaled Smax** (word 7) dacă setpoint este citit în mărimi inginereşti, acest parametru corespunde valorii setpoint când intrarea atinge 16383;
  - **Minimum Scaled Smin** (word 8) dacă setpoint este citit în mărimi inginereşti, acest parametru corespunde valorii setpoint când intrarea atinge valoarea 0;
  - **Deadband DB** (word 9) este o valoare pozitivă. DB este centrată în jurul valorii SP, având lățimea introdusă de utilizator, DB este introdusă când PV şi SP trec prin zero. DB are efect numai după ce PV intră în DB şi trece de SP;
  - **Loop Update** (word 13) este un intervalul de timp între calculele PID. O regulă este setarea lui la o valoare de 5-10 ori mai mică decât perioada naturală de încărcare;
  - **Scaled Process PV** (word 14) este pentru afişare
  - **Scaled Error** (word 15) este pentru afişare

- **Output CV %** (word 16) afişează ieşirea CV în procente (pentru intervalul 0-16383). Dacă este în mod Auto, această setare este numai pentru afişare. Dacă este în mod Manual valoarea CV se poate schimba, iar noua valoare ajunge la ieşire;

- **[F4] Output (CV) Limit OL** (word 0, bit 3) se poate atege între YES (se limitează ieşirea între valorile min şi max) şi NO (nu se limitează ieşirea);

În coloana din dreapta sunt indicatorii (flags) asociaţi instrucţiunii PID:

- **Time Moile Bit TM** (word 0, bit 0) specifică modul PID; este setat de către modul TIMED şi este resetat de către modul STI; poate fi setat/resetat prin instrucţiuni în programele ladder;
- **Auto / Manual Bit AM** (word 0, bit 01) arată că operaţiile sunt automate când este resetat şi manuale când este setat;poate fi setat/resetat prin instructiuni în programele ladder;
- **Control Mode Bit CM** (word 0, bit 02) este resetat dacă controlul este E=PV-SP; poate fi setat/resetat prin instrucţiuni în programele ladder;
- **Output Limiting Enabled Bit OL** (word 0, bit 03) este setat când a fost selectată o limită pentru CV prin [F4]; poate fi setat/resetat prin instrucţiuni în programele ladder;
- **Reset aud Gain Range Enhancement Bit RG** (word 0, bit 04) dacă este setat face ca valoarea Reset Minute/Repeat şi factorul de amplificare să fie multiplicat de 10 ori;
- **Scale Setpoint Flag SC** (word 0, bit 05) este resetat când se specifica o scală pentru setpoint;
- **Loop Update Time Too Fast TF** (word 0 bit 06) este setat de către algoritmul PID dacă timpul de actualizare al buclei, care a fost dat, nu poate fi atins de către program (din cauza limitărilor temporale de scanare). Dacă acest bit este setat se poate încerca corectarea problemei prin actualizarea buclei la o rată mai mică sau mutarea instrucţiunii într-o subrutină STI;
- **Derivitive (Rate) Action Bit DA** (word 0, bit 07) dacă este setat face ca Derivitive Rate să fie evaluată după eroare în loc de PV; dacă este resetat, Derivitive Rate funcţionează ca procesorul 5/02;
- **DB, Set When Error is în DB** (word 0, bit 08) este setat când PV este în DB în 0;
- **Output Alarm, Upper Limft UL** (word 0, bit 09) este setat când ieşirea CV depăşeşte limita superioară;
- **Output Alarm, Lower Urnit LL** (word 0, bit 10) este setat când ieşirea CV depăşeşte limita inferioara;
- **Setpoin Out of Range SP** (word 0, bit 11) este setat când setpoint nu se încadrează în scala dată;
- **Process Var Out of Range PV** (word 0, bit 12) este setat când PV nu se încadrează în intervalul 0-16383;
- **PID Donie DN** (word 0, bit 13) este setat când algoritmul PID a terminat calculele;
- **PID Enable EN** (word 0, bit 15) este setat când ramura instrucţiunii PID este adevărata;

3.5 Dezvoltarea unei diagrame Ladder pornind de la o diagramă Grafcet

Pentru ca un program de control conceput sub forma unei diagrame Ladder să funcţioneze corect şi să fie dezvoltat uşor, o soluţie foarte eficienţă este ridicarea iniţială a unei diagrame de tip Grafcet pornind de la specificaţiile procesului.

Elementele principale ale unei diagrame Grafcet sunt etapele, tranziţiile, acţiunile asociate etapelor, divergenţe, paralelisem etc, şi au fost descrise în capitolul 2. Etapele de creare ale unei diagrame Ladder pentru un automat Allen Bradley SLC 500 sunt următoarele:

- Se construieşte diagrama Grafcet de control logic al procesului
- Se identifică intrările, ieşirile necesare automatului, precum şi numărul de temporizări, contorizări necesare.
- Se asociază intrările şi ieşirile cu biţi din fişierele de imagine a intrărilor şi ieşirilor.
- Se asociază fiecare etapă a diagramei Grafcet cu câte un bit dintr-un fişier de tip BIT al automatului Allen Bradlţy (spre exemplu din fişierul B3).
- Se alocă pentru fiecare temporizare sau contorizare câte un element dintr-un fişier de tip Timer sau Counter.
- Pentru fiecare etapă se scriu, în principiu două ramuri de tipul:

1. Dacă (etapa e activă).....atunci (acţiuni asociate etapei)
2. Dacă (etapa e activă) şi (condiţia de tranziţie) atunci (dezactivare etapă curentă)
(oprire acţiuni asociate etapei)
(activare etape următoare)

Etapele initiale simt activate cu ajutorul instrucţiunii OSR (vezi problemele rezolvate). Mici modificări apar atunci când în diagrama Grafcet există divergenţe şi paralelisme (vezi probleme rezolvate).

În cazul în care dintr-o etapă se poate tranzita în diferite alte etape din cauza existenţei mai multor condiţii de tranziţie, trebuie editată câte o ramură de tip 2) pentru fiecare condiţie de tranziţie. Starea de activare a unei etape se testează cu instructiunea XIC aplicată bitului asociat etapei.

Dacă ulterior activării unei tranziţii urmează un paralelism în care mai multe secvenţe vor rula simultan, o dată cu dezactivarea etapei curente vor fi activate toate etapele iniţiale ale tuturor secvenţelor paralele.

Ieşirile digitale pot fi activate şi dezactivate atât cu instrucţiunea OTE cât şi cu instrucţiunile pereche OTL şi OTU. Instrucţiunea OTE se foloseşte atunci când o acţiune este activă numai în etapa curentă şi va trebui dezactivată o dată cu dezactivarea etapei. Dacă o ieşire trebuie să îşi menţină valoarea şi in etapele următoare, ea trebuie activată obligatoriu cu instrucţiunea OTL.

Exemplu: Să considerăm că avem de controlat un proces a cărui diagramă Grafcet este reprezenată în figura 3. 7:

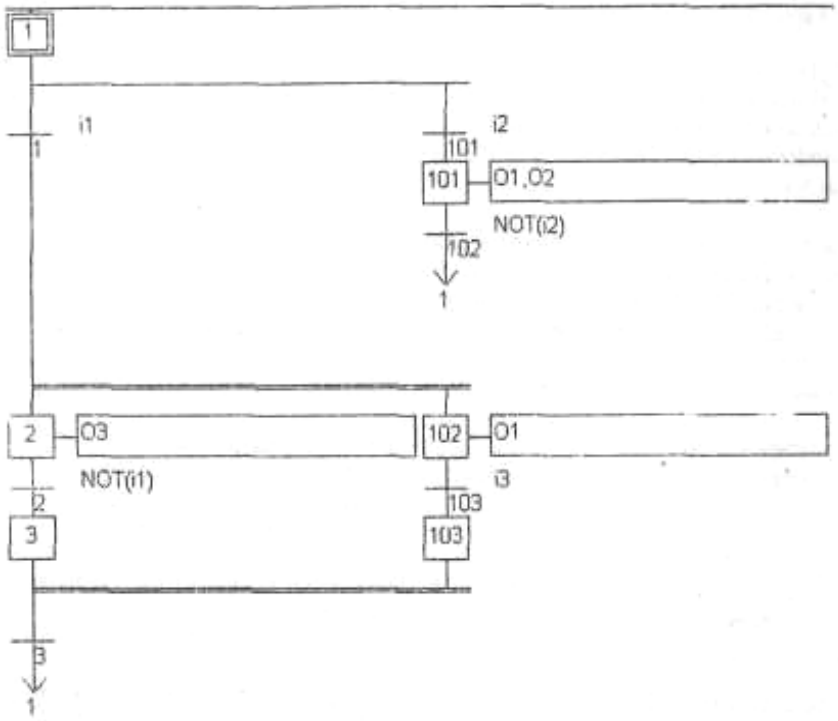


Fig. 3.7 Diagramă Grafcet

Pentru a construi diagram Ladder pornind de la diagrama Grafcet. în primul rând trebuiesc stabilite pozițiile intrărilor şi ieşirilor, precum şi asocierea etapelor din diagrama Grafcet cu biți din fişierul de bit B3.

Presupunând că pe slotul 1 al automatului se găseşte un modul de intrări digitale cu 16 intrări iar pe slotul 3 un modul de ieşiri digitale cu 24 ieşiri, putem conecta intrările din proces şi ieşirile către proces conform tabelului 3. 9:

Tabel 3. 9 Adresele intrărilor şi ieşirilor

Intrare	Adresă intrare	Ieşire	Adresă ieşire
i1	1:1/1	Ol	0:3/7
i2	1:1/2	02	0:3/8
i3	1:1/3	03	0:3/9

Asocierea etapelor Grafcet cu biți din fişierul de Bit B3 se poate face conform tabelului 3. 10:

Tabel 3. 10 Asocierea etapelor cu biți din fişierul B3

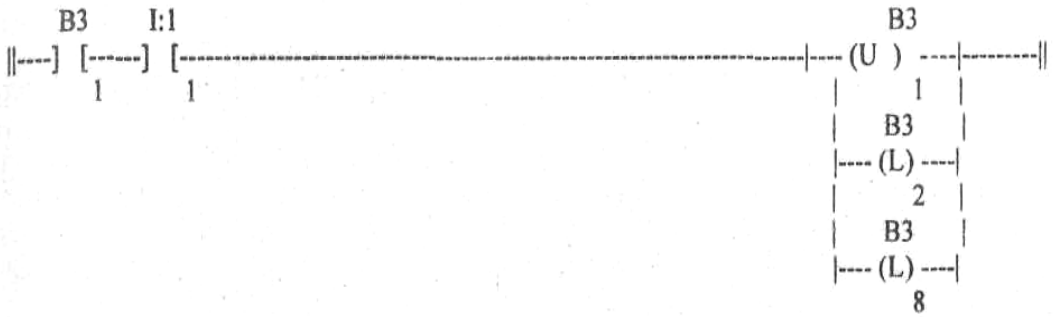
Etapă	Bit asociat	Etapă	Bit asociat
1	B3/1	101	B3/7
2	B3/2	102	B3/8
3	B3/3	103	B3/9

Bitul B3/0 va fi rezervat instrucțiunii OSR ca parametru intern. Se poate trece acum la scrierea ramurilor diagramei Ladder, conform algoritmului descris anterior.

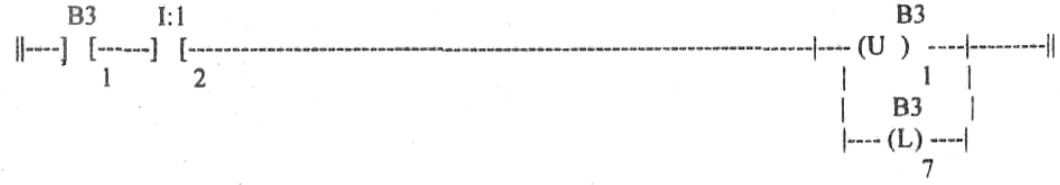
Etapa 1 va fi activată inițial cu ajutorul instrucțiunii OSR astfel:

Deoarepe în etapa 1 nu au loc acțiuni, nu se mai scrie nici o ramură de tip 1) pentru etapă, în schimb, deoarece are loc o divergență pornind din etapa 1, trebuiesc scrise două ramuri de tip 2) pentru cele două condiții de tranziție ce se exclud:

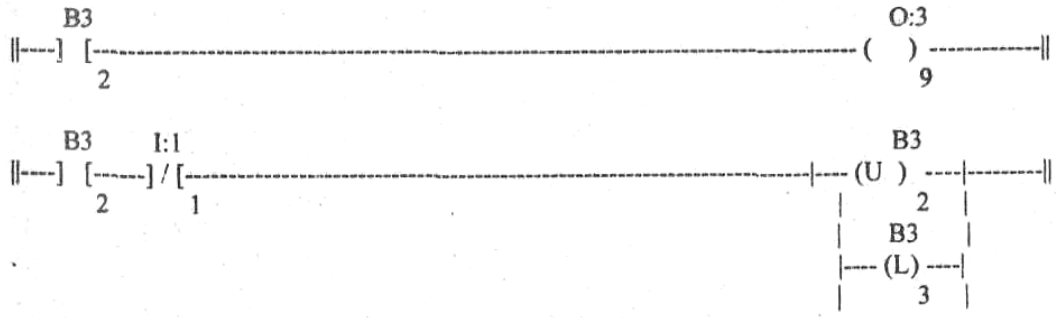
Pentru tranziția cu condiția i1:



Pentru tranziția cu condiția i2:



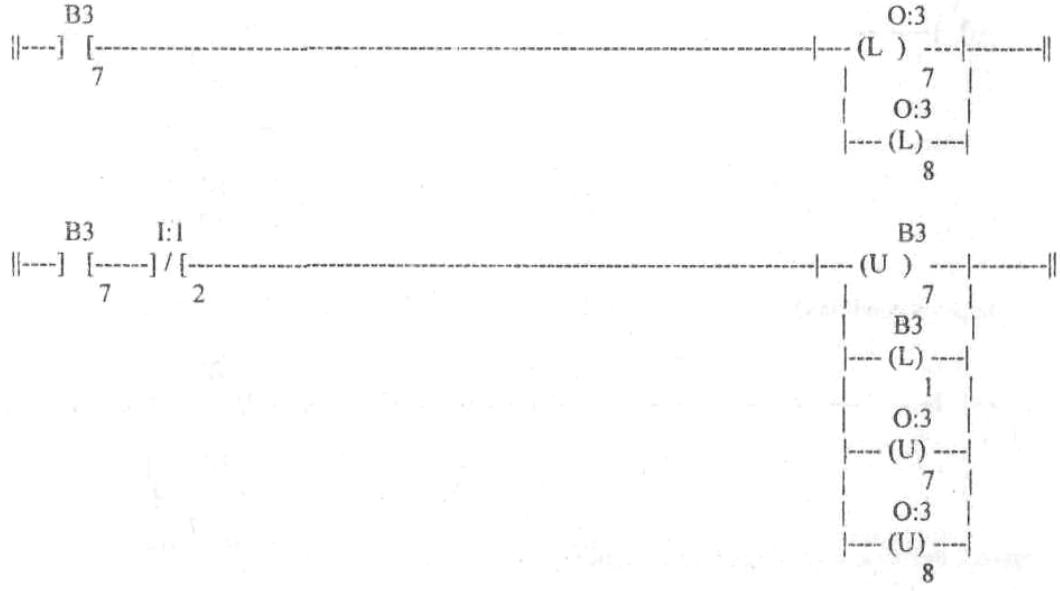
Pentru etapa 2 trebuiesc scrise cele 2 ramuri tipice:



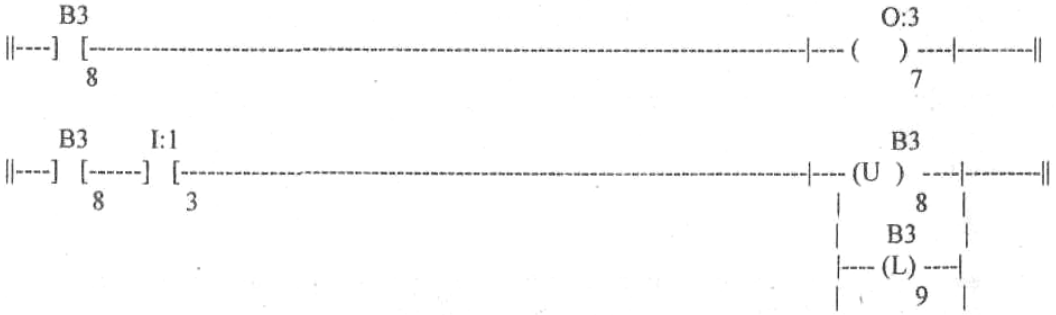
Etapa 3 nu are acţiuni asociate, iar tranziţia din această etapă este condiţionată de activarea etapei 103, astfel încât pentru etapa 3 se scrie o singură ramură de tip 2) si anume:

Această ramură este valabilă ca ramură de tip 2) şi pentru trazîţia din etapa 103, fiind suficientă editarea ei o singura dată.

Pentru etapa 101 se scriu cele două ramuri tipice:



Analog pentru etapa 102:



Programe mai complexe sunt prezentate în cadrul capitolului de probleme rezolvate.

CAP 4. MEDIUL DE PROGRAMARE LOGICA ISAGRAF

ISAGRAF este un mediu de programare dezvoltat de CJ Intenationai destinat aplicațiilor pentru automate programabile. Acest mediu permite programarea, simularea, testarea aplicațiilor folosind limbajele standardului IEC 1131-3 [CJ International, 1998].
Caracteristici:

- Mediu de dezvoltare complet
- Independență fată de configurația hardware a unui AP
- Limbaje de programare specificate în standardul JEC 1131-3
- Facilități de simulare
- ON si OFF line debugger
- Dezvoltarea de biblioteci software

4.1 Structura unei proiect Isagraf

O aplicație (proiect) ISAGRAF este compusă din mai multe unități numite programe. Un program este considerat a fi o unitate logica de programare, care descrie operațiile dintre variabilele procesului. Programele aplicației sunt conectate într-o structură arborescentă şi pot fi implementate folosind limbaje grafice sau literale, programatorul având la dispoziție următoarele opțiuni:

- **Sequnntial Function Chart** (SFC) pentru programare la nivel înalt
- **Flow Chart** (FC) pentru programare la nivel înalt
- **Function Block Diagram** (FBD) pentru operații ciclice complexe
- **Structured Text** (ST) pentru operații booleene
- **Instruction List** (IL) pentru operații la nivel jos

Limbajul utilizat pentru scrierea unui program se alege la crearea acestuia şi nu poate fi schimbat mai târziu. Exceptie se poate face în cazul în care se combină FBD şi LD.

o *Operațiuni ciclice şi secvențiale*

Programele descriu operații ciclice sau secvențiale. Programele ciclice sunt executate la fiecare ciclu al sistemului țintă. Execuția programelor secvențiale urmează regulile dinamice ale limbajului SFC, fie ale limbajului FC.

Programele sunt legate între ele într-o structură arborescentă. Programele aflate în vârful ierarhiei sunt activate de către sistem. Subprogramele (nivele mai jos în ierarhie) sunt activate de către programul părinte.

Structura ierarhică a unui program se descompune în patru secțiuni sau grupuri:

Begin	programe executate la inceputul fiecărui ciclu al sistemului țintă
Sequential	programe ce urmează regulile SFC sau FC
End	programe executate la sfârşitul fiecărui ciclu al sistemului ținta
Functions	set de subprograme nededicate

Programele din secțiunile Begin şi End descriu operații ciclice şi nu sunt dependente de timp. Programele principale sunt executate la începutul, respectiv sfârşitul, fiecărui ciclu de execuție.

Programele secțiunii Sequential descriu operații secvențiale, unde variabila timp sincronizează operațiunile primare. Execuția programelor principale ale acestei secțiuni se face conform regulilor SFC sau FC.

Programele secțiunii Functions sunt subprograme ce pot fi apelate de către oricare alt program al aplicației. Programele principale şi programele fiu (child programs) trebuie să fie descrise în limbajele SFC sau FC, Programele secțiunilor ciclice, (begin şi end), nu pot fi descrise în SFC sau FC. Orice program al oricărei secțiuni poate avea unul sau mai multe programe fiu, în conformitate cu limbajul fiecăruia. Subprogramele nu pot fi deserse în SFC sau FC.

Programele secțiunii Begin sunt de obicei folosite pentru descrierea operațiunilor preliminare în scopul citirii corecte a unor valori de la dispozitivele de intrare. Variabile de intrare citite sunt utilizate de către programele secțiunii Sequentiat pentru ca după o eventuală prelucrare valorile variabilelor de ieşire să fie transmise dispozitivelor de ieşire. Programele secțiunii End sunt utilizate pentru a descrie operațiuni de protecție înainte de a trimite o variabilă către un dispozitiv de ieşire.

- *Programe SFC / FC child*

Orice program SFC al secțiunii secvențiale poate controla alte programe SFC. Acest tip de programe, situate pe un nivel ierarhic inferior, se numesc program SFC child ele fiind programe paralele ce pot, fi pornite (started), oprite (killed), înghețate (frozen), repornite (restarted) de către programul părinte. Important eiste ca ambele programe, părinte şi fiu, să fie scrise in SFC. Un program SFC fiu poate conține variabile locale şi cuvinte cheie.

Atunci când un program părinte porneşte un program SFC fiu, transmite un jeton SFC, (SFC token), fiecărui pas inițial al programului fiu, în cazul opririi sunt eliminate toate jetoanele existente. Dacă este înghețat, un program fiu SFC îşi păstrează jetoanele pentru o eventuală repornire.

Şi în cazul programelor FC ale secțiunii secvențiale pot exista programe FC fiu, dar un părinte FC este blocat în timpul execuției unui subprogram FC. Astfel nu sunt posibile operațiuni simultane în programul FC părinte şi subprogramul FC.

- *Funcții şi subprograme*

Execuția unui subprogram sau a unei funcții este condusă de către programul părinte. Execuția programului părinte este suspendată până la terminarea funcției sau a subprogramului. Un subprogram poate fi controlat numai de către părintele său. El poate conține variabile locale. Pentru descrierea unui subprogram fiu pot fi utilizate limbajele SFC şi FC.

Programele secțiunii Functions sunt subprograme ce pot fi apelate de orice program al aplicației. Spre deosebire de alte subprograme acestea nu sunt dedicate programului părinte şi pot apela programe ale aceleiaşi secțiuni. Pentru o mai mare flexibilitate pot fi grupate în librării de fiincții.

- *Reguli de execuție*

ISAGRAF este un sistem sincron. Toate operațiunile sunt declanşate de către ceas. Intervalul dedicat-execuției se numeşte durata ciclului (the cycle timing):

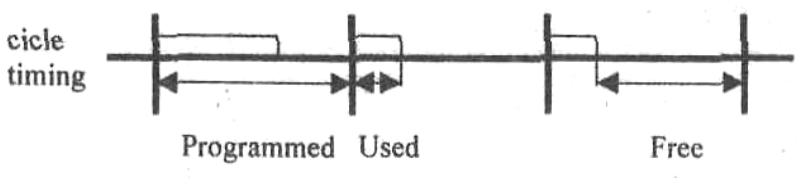


Fig.4.1 Ciclul Isagraf

Operațiile de bază ce se execută pe durata unui ciclu sunt:

- citirea variabilelor de intrare;
- tratarea programelor secțiunii Begin;
- tratarea programelor secțiunii Sequential în conformitate cu regulile dinamicii SFC/FC;
- tratarea programelor secțiunii End;
- actualizarea variabilelor de ieşire;

Acest sistem face posibile:

- garantarea unor valori constante pe durata unui ciclu pentru variabile;
- stabilitatea ieşirilor, acestea nefiind modificate decât o singură dată într-un ciclu de execuție;
- accesul sigur la aceleaşi variabile globale al mai multor programe;
- estimarea şi controlul timpului de răspuns al întregii aplicații;

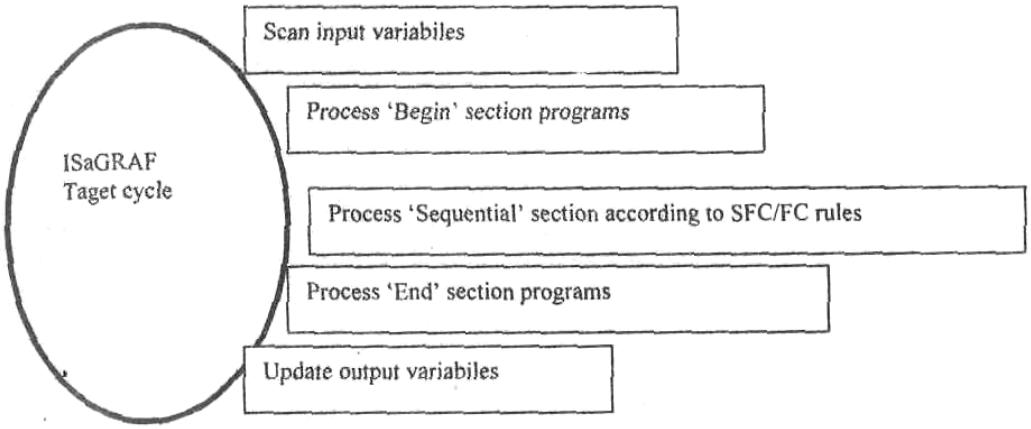


Fig. 4.2 Operațiile de bază ale unui ciclu de execuție.

4.2 Descrierea limbajul SFC (Sequential Function Chart)

Sequentia Function Chart (SFC) este un limbaj grafic utilizat pentru a descrie operații secvențiale. Procesul este reprezentat de o succesiune de paşi de finiți expliciți, legați prin tranziții. Fiecărei tranziții îi este ataşată o condiție de tip boolean. Acțiunile ce au loc la fiecare pas sunt descrise folosind celelalte limbaje (ST, IL, LD sau FDB).

Un program SFC se reprezintă ca un graf orientat cu noduri de tip etape (steps) şi tranziții (transitions). Legăturile multiple sunt utilizate pentru a reprezenta convergențe şi divergențe. Unele părți ale programului pot fi descrise separat şi reprezentate în program printr-un simbol. Aceste părți sunt cunoscute sub denumirea de macro steps (eng).

Regulile de bază ale SFC:

- Un nod de tip etapă nu poate fi urmat de un nod de acelaşi tip.
- Un nod de tip tranziție nu poate fi urmat de un nod de acelaşi tip.

Componentele de baza (simbolurile grafice) ale limbajului SFC sunt:

- **Etapele** şi etapele inițiale (steps and initial steps) :

O etapă este reprezentată printr-un pătrat. Fiecare etapă este referită de un număr înscris în interiorul pătratului. Descrierea etapei se face în interiorul dreptunghiului legat de pătrat. Aceasta este cunoscuta ca nivelul I al etapei (level 1 of the step).

Nivelul 2 conține descrierea acțiunilor ce au loc în etapa respectivă şi este referit într-o fereastră, separată. În momentul rulării un jeton arată că etapa este activă

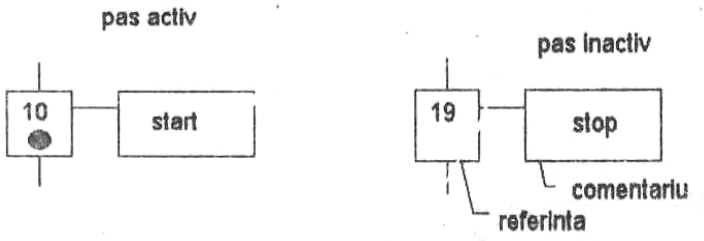


Fig. 4.3 a) Etapă activă b) Etapă inactivă

Starea inițială a unui program SFC se expune cu ajutorul etapelor inițiale, reprezentate grafic printr-un pătrat cu margine dublă. Când programul este pornit automat se plasează câte un jeton în interiorul fiecărei etape inițiale.

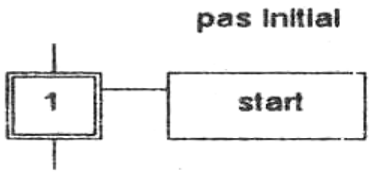


Fig.4.4 Pas inițial

Un program SFC trebuie să conțină cel puțin o etapă inițială. Fiecare etapă are două atribute ce pot fi utilizate de celelalte limbaje:

- GSnnn.x starea pasului: activ/inactiv (variabila booleana)
- GSnnn.t cât timp este activ pasul (timp) (nnn este numărul de referință al pasului).

o **Tranziții** (transitions):

Tranzițiile sunt reprezentate printr-o bară orizontală ce taie linia de legătură dintre doi paşi. Similar etapelor, tranzițiile sunt referite printr-un număr şi au aceeaşi organizare pe nivele:

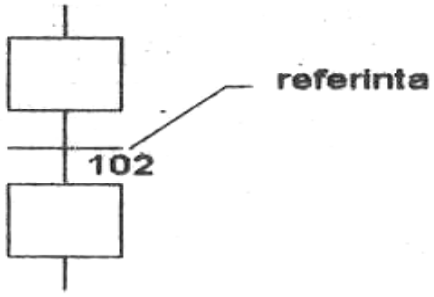


Fig.4.5 Tranziție

o **Legături orientate** (Oriented links):

Liniile simple sunt utilizate pentru legăturile dintre paşi şi tranziții. Atunci când nu este specificata explicit orientarea se consideră a fi de sus în jos:

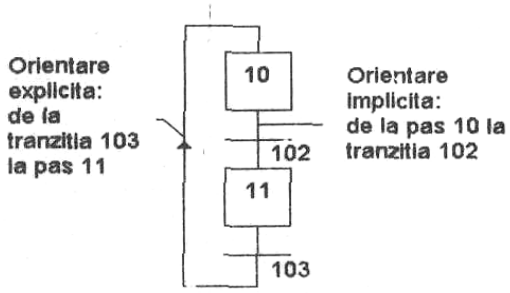


Fig.4.6 Legături orientate

- **Salt către un pas** (Jump to a step):

Simbolul de salt poate fi utilizat pentru a indica o legătura dintre o tranziție şi un pas, fără a fi nevoie să se deseneze linia de conexiune. Saltul trebuie referit cu numărul pasului destinație. Saltul nu poate ft utilizat pentru a lega un pas şi o tranziție.

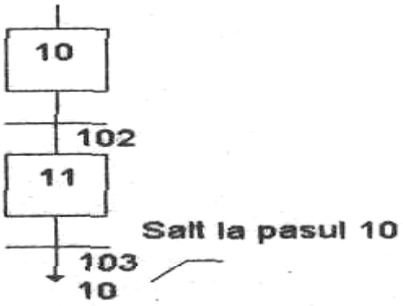


Fig. 4.7 Salt către o etapă

- **Divergențe şi convergențe**

Divergențele sunt conexiuni multiple de la un simbol SFC (pas sau tranziție) către mai multe simboluri SFC. Convergențele sunt conexiuni multiple de la mai multe simboluri SFC către un singur alt simbol. Structurile de acest tip pot fi simple sau duble, după cum urmează;

- Divergențe simple: Legături multiple de la un pas la mai multe tranziții. Aceasta permite trecerea jetonului activ către mai multe ramuri.
- Convergențe simple: Legături multiple de la mai multe tranziții către acelaşi pas. Sunt folosite în general pentru a grupa ramuri ce s-au desnărțit de la aceeaşi traziție.

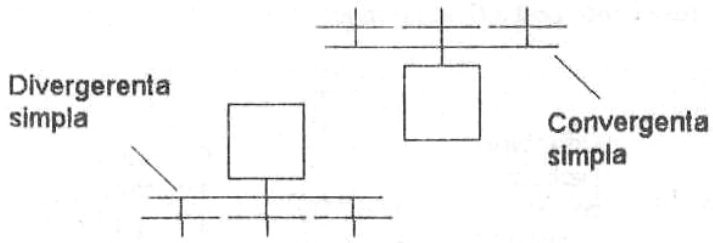


Fig. 4.8 Divergență şi convergență simplă

- Divergențe duble: Legături multiple de la o tranziție către mai mulți paşi. Au corespondențe în execuția în paralel a operațiunilor într-un proces.
- Convergențe duble: Legături multiple de la mai mulți paşi către aceeaşi tranziție. Folosite in general pentru a grupa ramuri ce s-au despărțit prin aceeaşi dublă divergență.

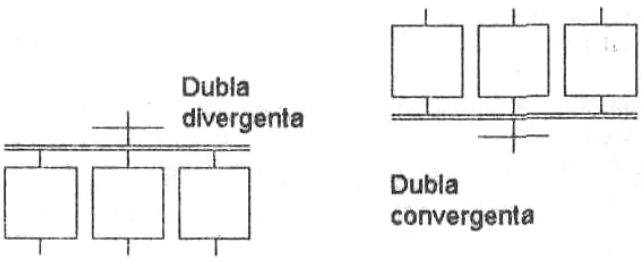


Fig. 4.9 Divergență și convergență dublă

- **Paşi Macro (Macro steps)**

Macro-urile sunt un mod de reprezentare a unei succesiuni de paşi şi tranziții. Corpul unui macro este descris separat în programul SFC, el apărând ca un singur simbol în reprezentarea SFC principală. Numărul de referință ce apare în simbolul macro-ului este referința pentru primul pas din corpul unui macro. Acest prim pas trebuie să fie de tipul pas de început (beginning step). Corespondentul său la terminarea reprezentării este pasul de sfârşit (ending step).

Reprezentarea unui macro trebuie să fie independentă, neexistând legături în nici o direcție, Simbolul unui macro poate fî indus în reprezentarea altui macro. Deoarece un macro este reprezentat printr-un set unic de paşi şi tranziții, nu poate fi folosit mai mult decât o dată în programul SFC.

- **Operații în cadrul etapelor**

Nivelul 2 al unei etape SFC conține descrierea detaliată a operațiilor ce se execută în timpul activării. Aceasta se face utilizând facilitățile literale ale SFC şi celelalte limbaje cum ar fi Structured Text (ST). Tipurile de operații ce sunt accesibile la acest nivel sunt:

- Operații asupra variabilelor booleene:

  Atribuirea de valori unor Variabile booleene în timpul activării unui pas. Aceste variabile pot fi de ieşire sau interne. Valorile se stabilesc la activarea sau dezactivarea pasului:

  <var_bool>(s); variabila primeşte valoarea TRUE când pasul devine activ.

  Sintaxa acestor operații:

  <var_bool>(N); asociază starea pasului activ/inactiv cu TRUE/FALSE (1/0 logice) valoarea variabilei
  <var_bool>; acelaşi efect atributul a fiind opțional
  /<var_bool>; asociază negatul stării pasului cu valoarea variabilei
  <var_bool>(s); variabila primeşte valoarea TRUE când pasul devine activ
  <var_bool>(r); variabila primeşte valoarea FALSE când pasul devine activ

- **Operații de tipul pulse actions:**

Succesiunea de instrucțiuni ST sau IL se execută numai o singură dată la activarea pasului indiferent de numărul de cicluri de execuție ce se efectuează până la dezactivarea pasului.
Sintaxa operațiilor:

```
ACTION(P):
    (*instrucțiuni ST*)
END_ACTION;
```

- **Operații de tipul non-stored actions:**

Succesiunea de instrucțiuni ST sau IL se execută la fiecare ciclu de execuție pe toată perioada în care pasul este activ.
Sintaxa operațiilor:

```
ACTION(N):
        (*instrucțiuni ST*)
END_ACTON;
```

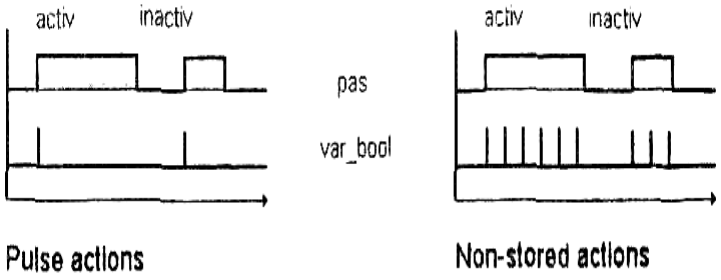


Fig. 4.10 Operații de tip P și N

- Operații SFC

O operație SFC este o secvență SFC fiu pornită sau oprită după cum se schimbă starea pasului. Poate avea unul din atributele: N (non stored), S (set), R (reset).
Sintaxa operațiilor:

<prog_fiu>(N);	prog_fiu este pornit când pasul devine activ şî oprit când pasul devine inactiv
<prog_fiu>;	acelaşi efect, atributul N fiind opțional
<prog_fiu>(S)	prog_fiu este pornit cînd pasul devine activ şi îşi păstrează starea când pasul devine inactiv
<prog_fiu>(R);	prog_fiu este oprit când pasul devine activ şî îşi păstrează starea când pasul devine inactiv

Secvența SFC fiu prog_fiu trebuie să fie un program SFC fiu al programului curent :

O serie de operații (fie că sunt sau nu de acelaşi tip) pot fi descrise în acelaşi pas. Opțiunile sunt următoarele:

- *Apelul funcțiilor şi al funcțiilor bloc*: Subprograme, funcții sau funcții bloc (scrise înST, IL, LD sau FDB) sau funcții C şi funcții bloc C pot fi direct apelate din interiorul unei operații după cum urmează:
  -Pentru sub programe, funcții şi funcții C:

```
ACTION(P):
        result:=sub_prog();
END_ACTION;

sau

ACTION(N):
        result:=sub_prog();
END_ACTION;
```

Pentru funcţii bloc C, ST, IL, LD, FDB:

```
ACTION(P):
        Fb(inl, in2);
         result1:=Fb.out1;
         result2:=Fb. out2;
END_ACTION;

sau

ACTION(N):
        Fb(inl, in2);
        resut1:=Fb.outl;
        result2:=Fb.out2;
END_ACTON;
```

- *Convenţia IL:*

Codul IL poale fi introdus după cum urmează:

```
ACTION(P):          (*sau N*)
#info=lL
        <instrucţiune>
        <instrucţiune>
        .....
END_ACTION;
```

- **Condiţii ataşate tranziţiilor**
  Fiecărei tranziţii îi este ataşată o expresie booleana ce condiţionează trecerea la pasul următor. Aceasta condiţie este de obicei implementată în limbajele ST sau LD la nivelul 2 al tranziţiei. Pot fi utilizate şi alte modalităţi de implementare:

- Convenţia ST:
  Limbajul ST poate descrie condiţia ataşată tranziţiei. Expresia finală a acesteia trebuie să fie de tip boolean şi să se termine cu punct şi virgulă. Expresia poate conţine constante TRUE sau FALSE, o singură variabilă de intrare sau o variabilă internă booleana, sau o combinaţie de variabile ce conduce la o valoare booleană.

- Conventia IL:
  Condiţia este descrisă conform următoarei sintaxe:

```
    #info=IL
        <iustructiune>
        <instructiune>
        .....
    #endinfo;
```

Valoarea conţinută în registrul IL (current result) la sfârşitul secvenţei condiţionând tranziţia:

```
current result=0        conditia este FALSE
current result=0        condiţia este TRUE
```

- Apelul unei funcţii
  Orice subprogram sau funcţie (scrisă în FBD, LD, ST sau IL), sau funcţie C poate fi apelată pentru a evalua condiţia ataşata tranziţiei conform sintaxei: <subprogram>();

Valoarea returnată trebuie să fie de tip boolean şi să returneze un rezultat ce va fi evaluat astfel:
val_ret=0 condiția este FALSE
val_ret<>0 condiția este TRUE
Dacă nu este ataşată nici o expresie tranziției, se consideră implicit valoarea TRUE

SFC - Regulile dinamicii limbajului

Cele cinci reguli ale dinamicii limbajului SFC sunt:

- Starea inițială: Caracterizată prin paşi inițiali (initial steps) care prin definiție, sunt în stare activă la începutul operațiilor. Fiecare program SFC trebuie să conțină ce puțin un pas inițial.
- Depăşirea unei tranziții: tranziția poate fi validată când toți paşii care o preced direct sunt activi, în caz contrar fiind invalidată. Tranziția nu poate fi depăşită decât dacă:
  - este validată şi
  - condiția asociată este TRUE
- Schimbarea stării paşilor: depăşirea unei tranziții conduce simultan la activarea paşilor care o succed şi la dezactivarea imediată a paşilor precedenți
- Depăşirea simultană a tranzițiilor: Tranzițiile care trebuiesc depăşite simultan pot fi indicate prin două linii. Dacă aceste tranziții sunt descrise separat, indicatorul de stare al paşilor precedenți (GSnnn.x) poate fi utilizat pentru descrierea condițiilor de trecere.
- Activarea şi dezactivarea simultana a unui pas: Dacă în timpul operațiilor, un pas este simultan activat şi dezactivat, se dă prioritate activării.

o **SFC-Ierarhizarea programelor**

Sistemul ISAGRAF permite descrierea pe verticală a structurii programelor SFC. Acestea sunt organizate într-o ierarhie arborescentă. Fiecare program SFC poate controla (porni, opri) alte programe SFC. Acestea din urmă se numesc fii ai programului care le controlează.
Regulile pe care le implică structura ierarhizată:

- Programele SFC care nu au un pătinte se numesc programe SFC principale.
- Programele SFC principale sunt activate de sistem la începutul aplicației.
- Un program poale avea mai multe programe fiu.
- Un program fiu nu poate avea mai mult decât un părinte.
- Un program fiu nu poate fi controlat dctât de părintele său.
- Un program nu poate controla fiii unui program fiu al său.

Operațiile pe care le poate efectua un părinte pentru controlul fiilor săi sunt:

- Start (GSTART) Porneşte programul fiu: activează fiecare pas inițial al său. Fii progamului fiu nu sunt porniți automat.
- Kill (GKILL) Opreşte programul fiu, dezactivând fiecare pas activ. Toți fiii programului fiu sunt de asemenea opriți.
- Freeze (GFREEZE) Suspendă execuția programului (dezactivând fiecare pas activ şi suspendând verificarea condițitor de depăşire a tranzițiilor), şi salvează starea paşilor programului astfel încât programul să poală fi repornit. Toți fii programului fiu sunt suspendați din execuție.
- Restart (GRST) Reporneşte un progam SFC suspendat reactivându-i toți paşii dezactivați la suspendarea din execuție. Programele fiu nu sunt automat repornite.
- Get status (GSATUS) Obține starea curenta (activ, inacliv, suspendat) a unui program fiu.

4.3 Descreierca limbajului ST (Structured Text)

ST (Struciured text) este un limbaj structurat de nivel înalt proiectat pentru procesele de automatizare. Acest limbaj este utilizat in special pentru a implementa proceduri complexe ce nu pot fi uşor exprimate într-un limbaj grafic, ST este limbajul implicit pentru descrierea operațiilor din interiorul paşilor şi a condițiilor atașate tranzițiilor din limbajul SFC.

o *ST-Elemcnte de sintaxă*

Un program ST este o listă de declarații ST. Fiecare declarație se termină cu separatorul punct şi virgula (;). Cuvintele utilizate în codul sursă (identificatori de variabile, constante, cuvinte cheie) sunt despărțite de separatori, inactivi (space, eol, tab) sau de separatori activi ce au o semnificație bine definită (> scmmfică mai mare, operatie de comparație). Comentariile pot fi incluse oriunde în text dar trebuiesc încadrate între ”(*”ş”i*)”. Pentru o mai bună lizibilitate a codului se recomandă câteva reguli: să nu se scrie mai mult de o declarație pe un rând, să se folosească indentarea declarațiilor complexe, inserarea de comentarii.

Tipurile de baza ale declarațiilor ST:

- Atribuire de valori (variabila:=expresie)
- Apelare de subprograme şi fimcții
- Apelare de funcții bloc
- Instrucțiuni de selecție(IF, THEN, ELSE, CASE...)
- Instrucțiuni de iterație (FOR, WHILE, REPEAT...)
- Instrucțiuni de control (RETURN, EXIT...)
- Instrucțiuni dedicate legăturilor cu alte limbaje, cum ar fi SFC

o *Expresii şi paranteze*

Expresiile ST combină operatori ST cu operanzi variabile sau constante. Pentru fiecare expresie tipul operanzilor trebuie să fie acelaşi. Parantezele sunt folosite pentru a separa părți ale expresiei şi pentru a stabili explicit prioritățile operațiilor. Când o expresie complexă nu are paranteze, succesiunea operațiilor este dată de prioritățile implicite între operatorii ST. O expresie poate conține maxim opt nivele de paranteze.

o *ST-operatori standard, funcții, proceduri*

Operatori standard:

- Transferul datelor:
  -atribuirea <var_o>:=<var_l>
  -negația analogică <var_o>:=-<var_l>

- Operații booleene:
  -AND ŞI <var_o>:=<var_l>AND<var_2>
  -OR SAU <var_o>:=<var_l>OR<var_2>
  -XOR SAU EXCLUSIV <var_o>:=<var_l>XOR<var_2>

- Operații aritmetice:
  - +, -, *, / adunare, scădere, înmulțire, împărțire
- Comparații:
  - <, <=, >, >=, =, <> mai mic, mai sau egal, mai mare, mai mare sau egal, egal, diferit

- Operații logice:
  -AND_MASK ŞI bit cu bit
  -OR_MASK SAU bit cu bit
  -XORMASK SAU EXCLUSIV bit cu bit
  -NOTMASK Negație bit cu bit

- Conversia datelor:
  -BOO conversie în valoare de tip boolean
  -ANA conversie în valoare de tip întreg
  -REAL conversie în valoare de tip real
  -TMR conversie în valoare de tip timer
  -MSG conversie în valoare de tip mesaj

- Altele:
  -CAT concatenarea mesajelor
  -SYSTEM acces la resursele sistemului
  -OPERATE operații cu canalele I/O

Proceduri standard suportate de sistemul ISAGRAF. Aceste proceduri sunt predefinite şi nu trebuie declarate într-o librărie.

- Proceduri pentru variabile booleene:
  -SR setare bistabil
  -RS resetare bistabil
  -R_TRIG detectare front crescător
  -F_TRIG detectare front descrescător
  -SEMA semafor

- Proceduri pentru numărătoare:
  -CTU numărător crescător
  -CTD numărător descrescător
  -CTUD numărător reversibil

- Proceduri pentru numărătoare:
  -TON incrementează un timer intern până la o anumită valoare de la detectarea frontului crescător al semnalului declanşator până la detectarea frontului descrescător
  -TO incrementează un timer intern până la o anumită valoare de la detectarea frontului descrescător al semnalului declanşator până la detectarea frontului crescător
  -TP incrementează un timer intern până la o anumită valoare de la detectarea frontului crescător al semnalului declanşator un interval de timp determinat .

- Proceduri pentru întregi:
  -CMP comparație
  -STACKINT gestionează o stivă de întregi

- Proceduri pentru valori întregi:
  -AVERAGE media în eşantioane
  -INTEGRAL integrare în timp
  -DERIVATE derivare în timp

- Proceduri pentru generare de semnale:
  -BLINK generează un semnal boolean în pulsuri
  –SIG_GEN generează o varietate de semnale (boolean, pulsatoriu, incremental, sinusoidal)

Funcţii standard suportate de sistemul ISAGRAF. Aceste funcţii sunt predefinite şi nu trebuiesc declarate între librărie.

- Funcţii matematice:
  -ABS valoare absolută
  -EXPT funcţia exponenţială
  -LOG funcţia logaritm
  -POW funcţia putere
  -SQRT rădăcină pătrată
  -TRUNC trunchierea părţii zecimale

- Funcţii trigonometrice:
  -ACOS
  -ASIN
  -ATAN
  -COS
  -SIN
  -TAN

- Funcţii de control a şirurilor de biţi:
  -ROL rotire la stânga
  -ROR rotire la dreapta
  -SHL deplasare la stânga
  -SHR deplasare la dreapta

- Funcţii de control a datelor:
  -MIN minimul a două valori întregi
  -MAX maximul a două valori întregi
  -LIMIT limitează valoarea unei variabile într-un interval precizat
  -MOD modulul unei valori întregi
  -MUX4 multiplexează 4 întregi
  -MUX8 multiplcxează 8 întregi
  -ODD testează paritatea unui întreg
  -RAND generează o valoare aleatoare într-un interval determinat
  -SEL selector binar: selectează o valoare dintre 2 valori întregi

- Funcţii de conversie a datelor:
  -ASCII returnează codul ASCII al unui caracter
  -CHAR înscrie într-jug şţring caracterul cu codul ASCII dat ca parametru funcţiei

- Funcţii pentru şiruri de caractere:
  -DELETE şergere subşir de caractere
  -INSERT inserare şir de caractere
  -FIND găsire subşir de caractere
  -MLEN obţinere lungime şir de caractere
  -LEFT extrage un număr specificat de caractere din partea stângă a unui şir de

caractere
-MID extrage un număr specificat de caractere dintr-un şir pornind de la o poziţie specificată
-REPLACE înlocuire subşir de caractere
-RlGTH extrage un număr specificat de caractere din partea dreaptă a unui şir de caractere
-DAY_TIME întoarce data/timpul ca un şir de caractere

- Funcţii pentru vectori:
  -ARCREATE crează un vector
  -ARREAD citeşte un elemet al unui vector
  -ARWR1TE scrie un element într-un vector

o *Extensii ale limbajului ST:*

- Funcţii pentru controlul variabilelor de tipTimer
  -TSTART (timer) porneşte incrementarea unei variabile de tip Timer, fără a-i reseta valoarea
  -TSTOP (timer) opreşte incrementarea unei variabilei de tip Timer

4.4 Variabile şi constante

Variabilele sunt grupate după domeniu şi tip. Numai variabilele de acelaşi tip şi din acelaşi domeniu fac parte din aceeaşi grila de intrare.

Domeniile variabilelor:

- GLOBAL: variabilele din acest domeniu pot fi utilizate de orice program al aplicaţiei curente;
- LOCAL : variabilele din acest domeniu pot fi utilizate numai de un singur program;

Tipuri de variabile:

-BOOLEAN valori binare de tipul TRUE/FALSE;
. -ANALOG valori întregi (INTEGER) sau reale (REAL);.
-TIMER valori de tip timer;
-MESSAGE şiruri de caractere.

O variabilă este identificată prin nume, comentariu atribut, adresă şi alte câmpuri specifice.
Tipurile de atribute ale variabilelor:

-INTERNAL variabilă din memorie (internă);
-INPUT variabilă conectată la un dispozitiv de intrare;
-OUTPUT variabilă conectată la un dispozitiv de ieşire;
-CONSTANT variabilă din memorie cu atributul de read-only si valoare iniţială.

Variabilele de tip Timer sunt întotdeauna variabile interne Variabile de Input şi Output fac parte din domeniul Global.

o *CONSTANTE*

Nu există decât două constante de tip Boolean şi anume:

-TRUE: reprezintă echivalentul numărului întreg 1
-FALSE : reprezintă echivalentul numărului întreg 0

O constantă de tip INTEGER este un număr întreg, cuprins între valorile -2^{32} şi $+2^{32}$
Constantele de tip INTEGER pot fi exprimate şi în alte baze de numeraţie şi trebuie să înceapă cu un prefix care să identifice baza folosită.

Bază	Prefix	Exemplu
ZECIMAL	(none)	-908
HEXAZECIMAL	16#	16#1A2B3C4D
OCTAZECIMAL	8#	8#1756402
BINAR	2#	2#1101_0001_0101_110

Constantele analogice reale pot fi scrise atât în reprezentare zecimală cât şi în reprezentare ştiinţifică. Punctul zecimal este folosit pentru a diferenţia o constantă reală de una de tip Integer. Reprezentarea ştiinţifică foloseşte E sau F pentru a separa mantisa de exponent. Dedesubt sunt prezentate câteva exemple de constante reale :

3. 14159
1. 0F-15
-789. 56

Numărul 123 nu reprezintă o constantă reală, reprezentarea ei corectă este 123, 0.

Constantele de tip Timer sunt vaiori cuprinse între 0 şi 23h59m59s999ms. Cea mai mică unitate permisă este milisecunda. Unităţile de timp standard folosite în expresii de tip Timer sunt:

- Hour - litera h trebuie să fie urmată de oră
- Minute - litera m trebuie să fie urmată de numărul de minute
- Second - litera s trebuie să fie urnată de numărul de secunde
- Milliseconds - literele ms trebuie să fie urmate de numărul de milisecunde

O constantă de tip Timer trebuie să înceapă cu prefixul T# sau TIME#. Unele unităţi pot să nu apară.
Exemple:

T#1H45OMS 1 oră, 450 milisecunde
time#5s 5 secunde

4. 5 Descrierea automatelor programabile volante de tip Smart PLC

În scopul proiectării unei platforme performante, hardware şi software, pentru controlul sistemelor discrete, o alegere foarte bună poate fi automatul de lip Smart PLC, fabricat de firma PEP Modular Computere [Pep, 1998]. Este un bun exemplu de automat programabil evoluat, cu o configuraţie hardware şi software extrem de performantă. Pricipalele elemente hardware din componenţa sa sunt:

- procesor standard Motorola 68***
- procesor comunicaţie
- memorie pentru firmware (EPROM)
- memorie pentru software de aplicaţie (RAM)
- memorie pentru reţinere date la pierderea alimentării de la reţea (SRAM)
- port pentru comunicaţii sub protocol PROFIBUS (RS485)
- port pentru comunicaţii RS212 pentru consola operator locală

- port pentru comunicații RS232 pentru terminal OS9 şi ISAGRAF Debugger
- intrări analogice
- intrări digitale
- ieşiri analogice
- ieşiri digitale

De asemenea dispune de module software rezidente în memoria nevolatilă :

- Sistem de operare OS-9: este un sistem de operare performant, independent de hardware, multitasking, multiuser, cu funcționare în întreruperi, planificare taskuri, control I/O în timp real etc
- Sistem integrat de interpretare şi execuție locală a programelor generate prin intermediul pachetului software de inginerie asistată ISAGRAF, precum şi de comunicație cu acesta
- Sistem de comunicații în rețele industriale deschise, conform specificațiilor de standard PROFIBUS
- Profibus Layer_2 şi Layer_7 client şi server ;
- Pachet PROFIBUS pentru administrare si monitorizare
- Sistem de comunicații seriale în standard RS232
- Manager de comunicații seriale
- Driver de comunicații seriale pentru consola OS9 şi ISaGRAF Debugger
- Driver de comunicații seriale pentru consola operator locală
- Sistem de drivere pentru modulele de I/O

Un program Isagraf este numai unul din taskurile care rulează la un moment dat pe automatul Smart PLC. Alte câteva task-uri sunt încărcate la punerea sub alimentare a aparatului. Unul dintre task-uri permite o comunicație permanentă între programul Isagraf ce rulează pe automat şi mediul de dezvoltare, Isagraf Workbench, de pe calculatorul personal. Astfel pot fi urmărite on-line, pe ecranul computerului, dinamica diagramelor SFC şi valorile tuturor variabilelor declarate.

Dacă se doreşte crearea unui program Isagraf care să comunice anumite date pe portul serial către o altă aplicație din computerul personal (nu Isagraf Workbench) acest task de debug trebuie oprit prin comenzi specifice sistemului de operare OS9 al automatului Smart PLC [Ivanescu, 1999].

Acest tip de automat programabil a fost ales în exemplele din capitolul 2.

CAP 5. PROBLEME REZOLVATE

Problema 1: Controlul unei macarale

1. Descrierea procesului:

Aplicația constă în controlul unei macarale care trebuie să realizeze cele 2 cicluri de mişcare reprezentate în figura 5.1. Inițial macaraua se găseşte în poziția de repaus 1. La apăsarea butonului de pornire, macaraua porneşte şi se realizează ciclul 1, până se ajunge în poziția de repaus 2, unde rămâne pentru un anumit timp cunoscut (3 s), înainte de a porni ciclul 2; când ajunge în poziția de repaus 1, macaraua se va opri. Un nou ciclu va pomi după reapăsarea butonului de pornire.

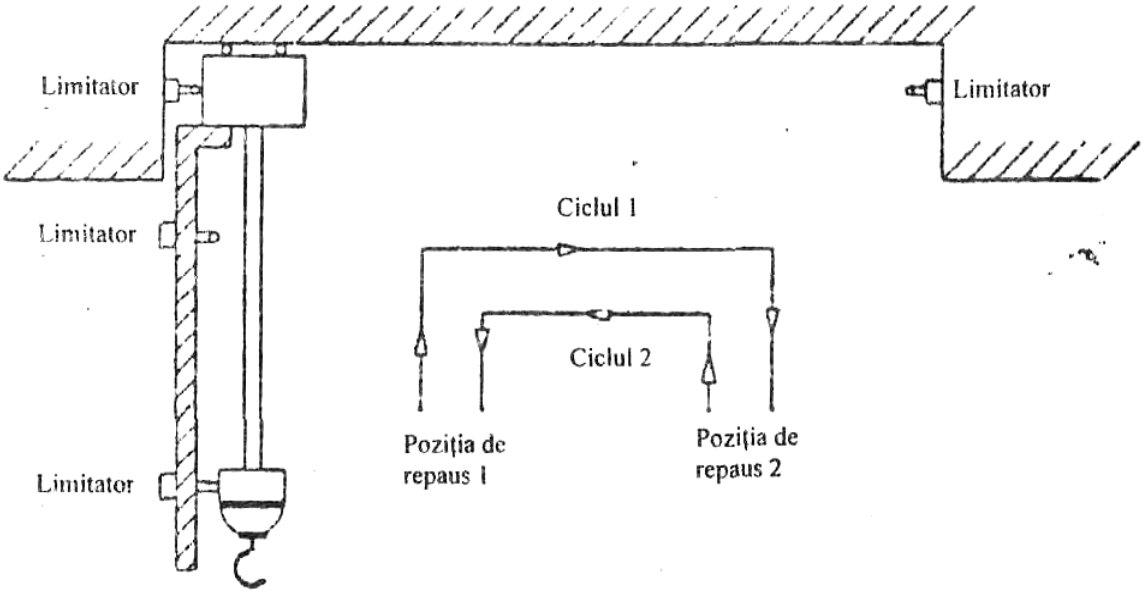


Fig. 5.1 Controlul unei macarale

Elemente de execuție:

- 2 motoare cu 2 sensuri de rotație, unul pentru mişcarea orizontală şi unul pentru cea verticală

Elemente de măsură:

- 4 timifatoare de cursă
- 1 buton de pornire

2. Soluția de automatizare

o Varianta 1: implementarea în mediul ISAGraf

Prima soluție pentru controlul acestei aplicații o reprezintă un automat programabil de tip PEP Smart pentru care s-a dezvoltat un proiect Isagraf ce cuprinde un program principal secvențial.

Dicționarul de variabile globale:

Variabile de intrare booleene:

- Pornire : buton de pornit ciclu
- l1: limitator jos
- 12: limitator sus
- 13: limitator stânga
- 14: limitator dreapta

Variabile de ieşire booleene:

M1S: acționare motor 1 stânga
M1D: acționare motor 1 dreapta
M2S: acționare motor 2 sus
M2J: acționare motor 2 jos

Programul principal este prezentat în figura 5.2

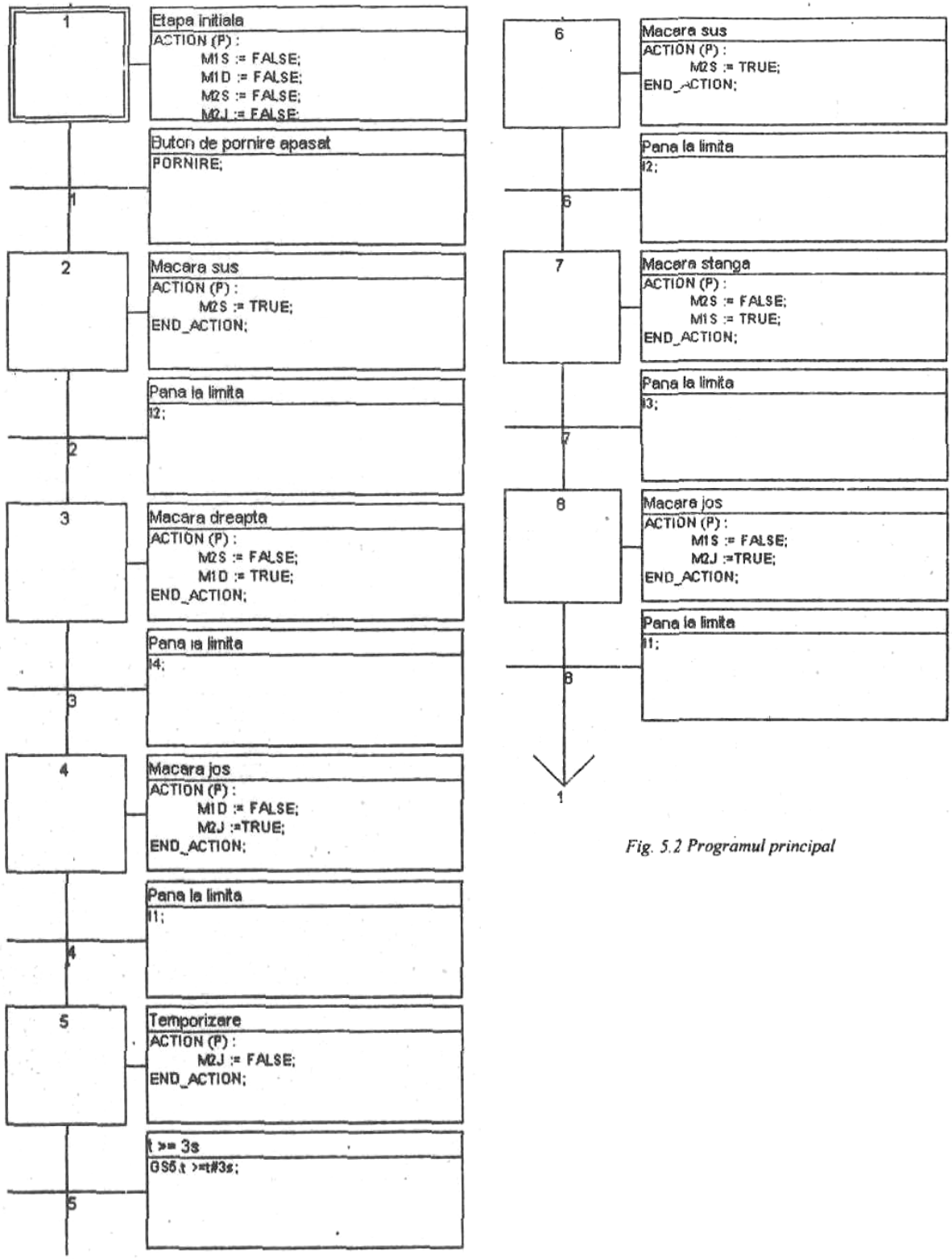


Fig. 5.2 Programul principal

- Varianta 2: implementarea în limbajul Ladder Diagram

Pentru controlul acestei aplicații s-a ales un automat programabil de tip Allen Bradley pentru care s-a dezvoltat o diagramă Grafcet (fig 5.3) şi un program de tip Ladder Diagram (fig. 5.4)

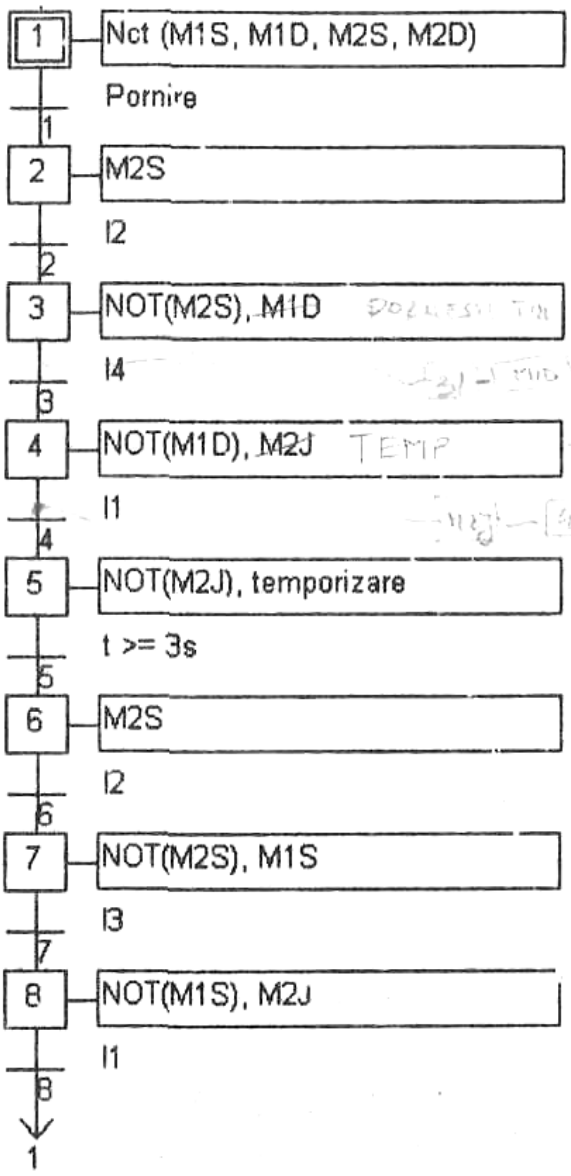


Fig.5.3 Diagrama Grafcet

Asocierea intrărilor şi ieşirilor fizice cu biți din registru de intrare/ieşire este prezentată în tabelul 5.1:

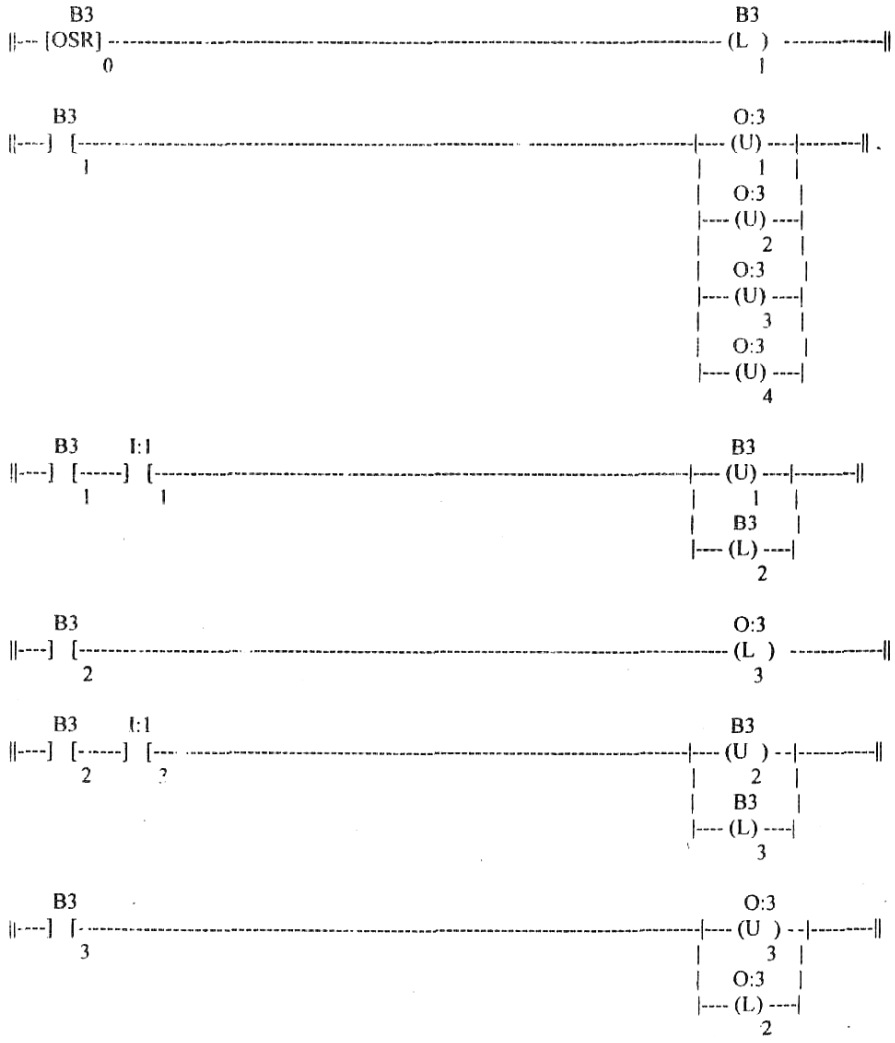
Tabelul 5.1

Intrare fizică	**Adresă internă**	**Ieşire fizică**	**Adresă internă**
Pornire	I : 1/1	M1S	0 : 3/1
I1	I : 1/2	M1D	0 : 3/2
I2	I : 1/3	M2S	0 : 3/3
I3	I : 1/4	M2J	0 : 3/4
I4	I : 1/5		

Asocierea etapelor cu biți din fișierul de bit B3 şi alegerea fişierului de timer este pezentată în **Tabelul 5. 2**

Etapa	**Adresa bit**	**Temporizare**	**Fişier de timer**
1	B3/1	Temporizare 1	T4:0
2	B3/2		
3	B3/3		
4	B3/4		
5	B3/5		
6	B3/6		
7	B3/7		
8	B3/8		

Diagrama Ladder este prezentată în continuare:



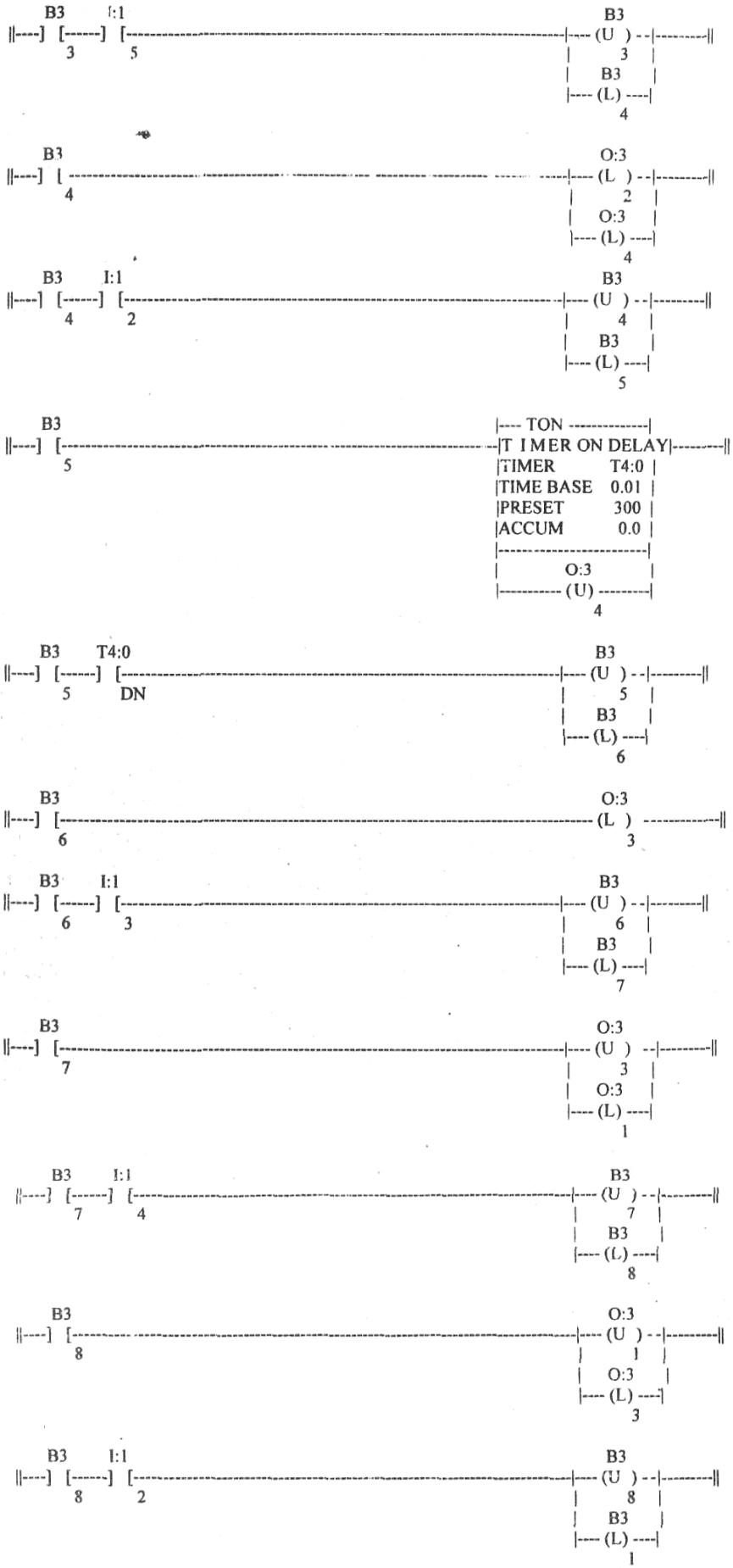


Fig.5.4 Diagrama Ladder

Problema 2: Comanda mişcării oscilatorie a unui mobil

1. Descrierea procesului:

Un mobil alunecă pe un şurub mişcat de un motor acționat de 2 contactoare (Cd - dreapta şi Cs -stânga). Mobilul trebuie să realizeze o mişcare oscilatorie continuă din momentul în care se primeşte comanda (impuls) de la butonul M. Un impuls de la butonul P trebuie să oprească motorul, dar nu imediat, ci la finalul mişcării începute. Un impuls de la butonul E produce o retragere imediată a mobilului în poziția de origine, iar sistemul se mai poate pune în mişcare doar apăsând butonul R.

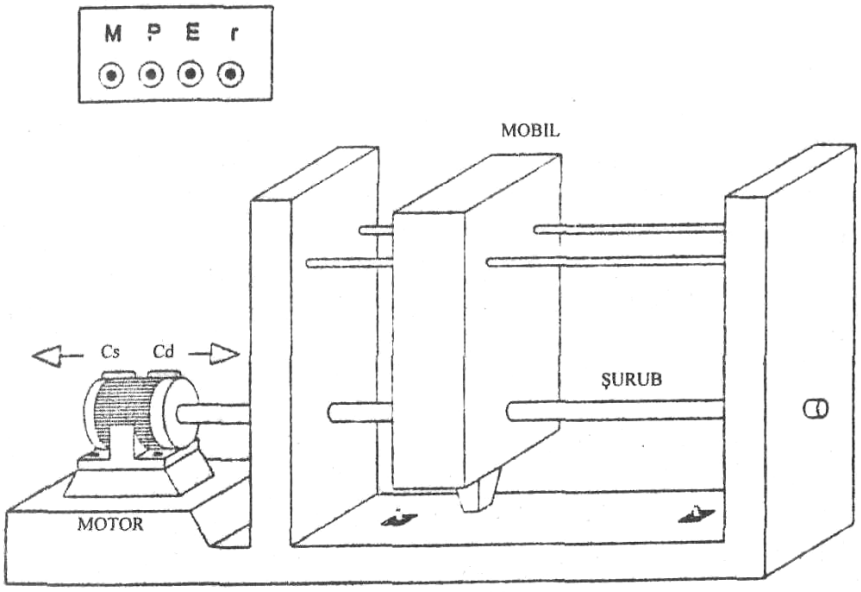


Fig. 5.5 Mişcarea oscilatorie a unui mobil

Element de execuție:

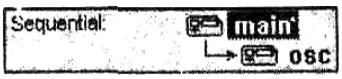
- 1 motor cu 2 sensuri de rotație

Elemente de măsură:

- 2 limitatoare de cursă

2. Soluția de automatizare

Pentru controlul acestei aplicații se alege un automat programabil de tip PEP Smart PLC pentru care se dezvoltă un proiect Isagraf. Proiectul conține 2 programe, un program principal şi un program copil, numit „Osc". Programul „Osc" este pornit şi oprit de către programul principal şi este responsabil cu realizarea mişcării oscilatorii şi sesizarea apăsării butoanelor conform specificațiilor aplicației.



Dicționarul de variabile globale:

Variabile de intrare digitale:

- M : buton de pornire mişcare oscilatorie
- P : buton de oprire motor
- E : buton de retragere în poziția de origine
- RST : buton de repornire
- L0 : limitator stânga
- L1 : limitator dreapta

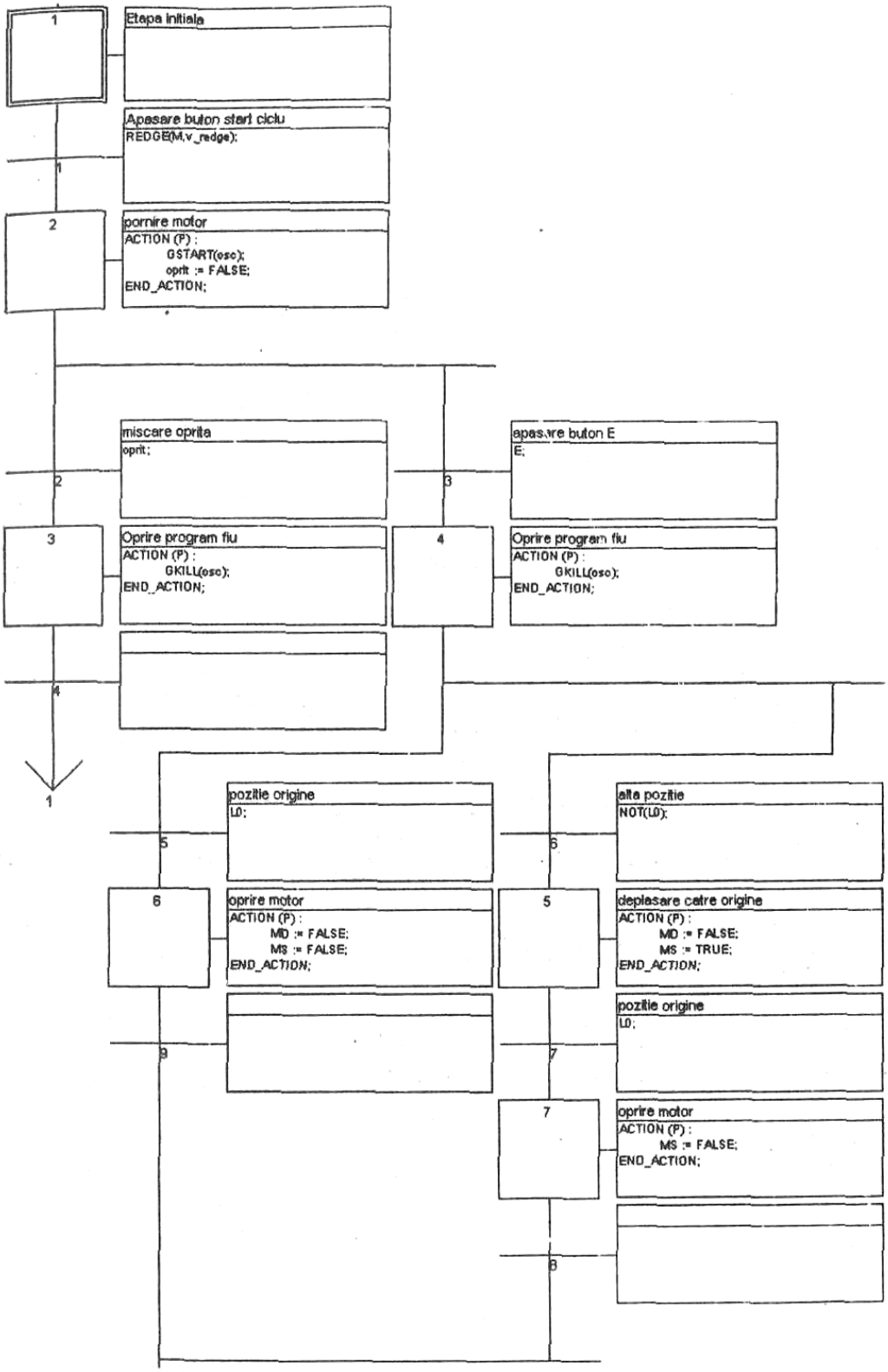
Variabile de ieşire digitale:

- MS : comandă motor stânga
- MD : comandă motor dreapta

Variabile interne de tip Boolean:

- oprit : are valoarea TRUE când mşcarea oscilatorie este oprită
- v_redge : necesar funcţiei REDGE

Programul „Main” este prezentat în figura 5.6.



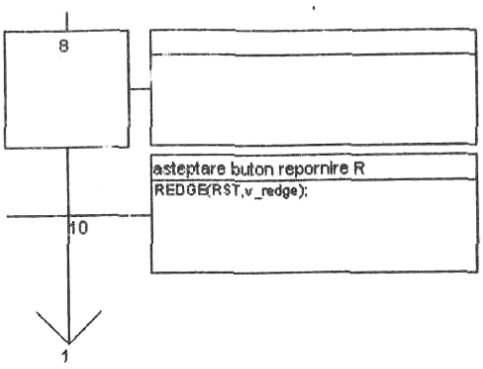
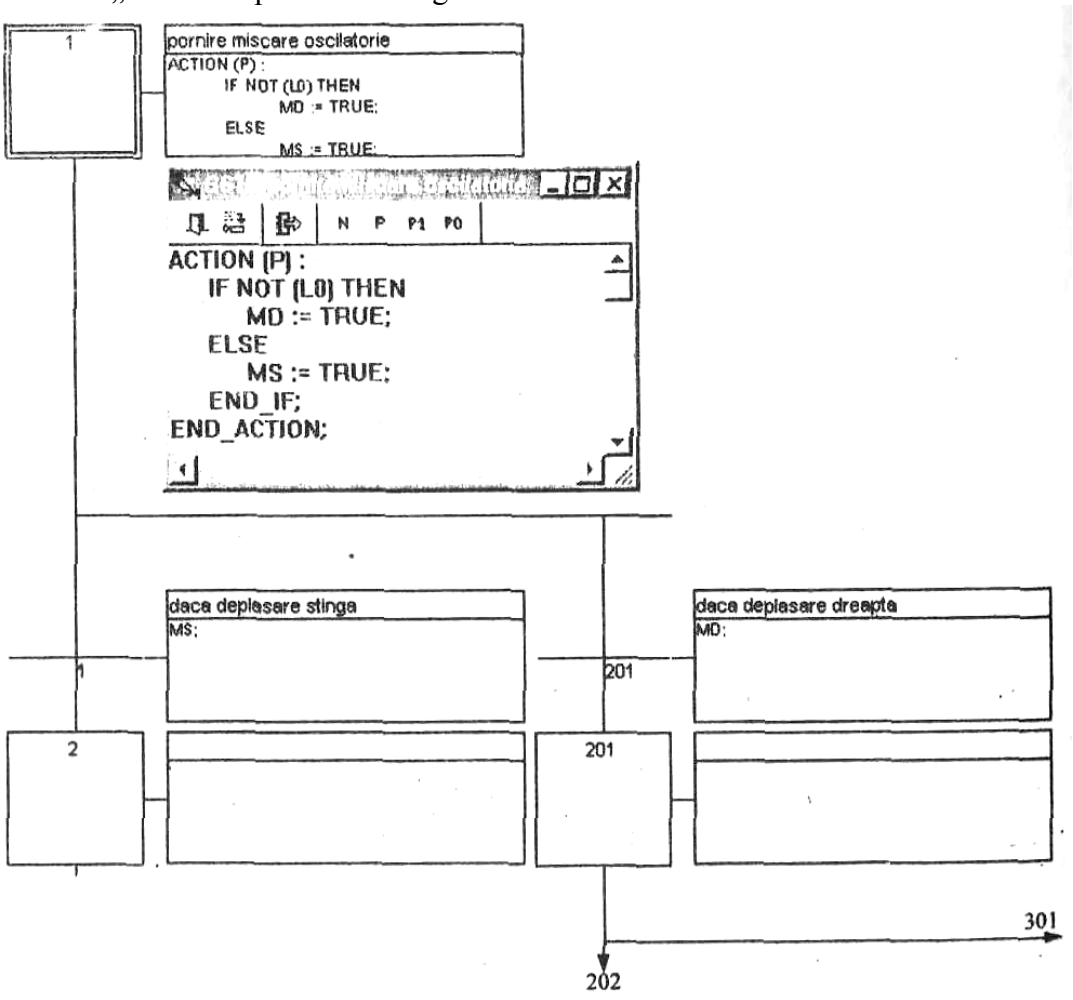


Fig 5.6 Programul „Main"

Programul fiu „Osc" este prezentat în figura 5.7.



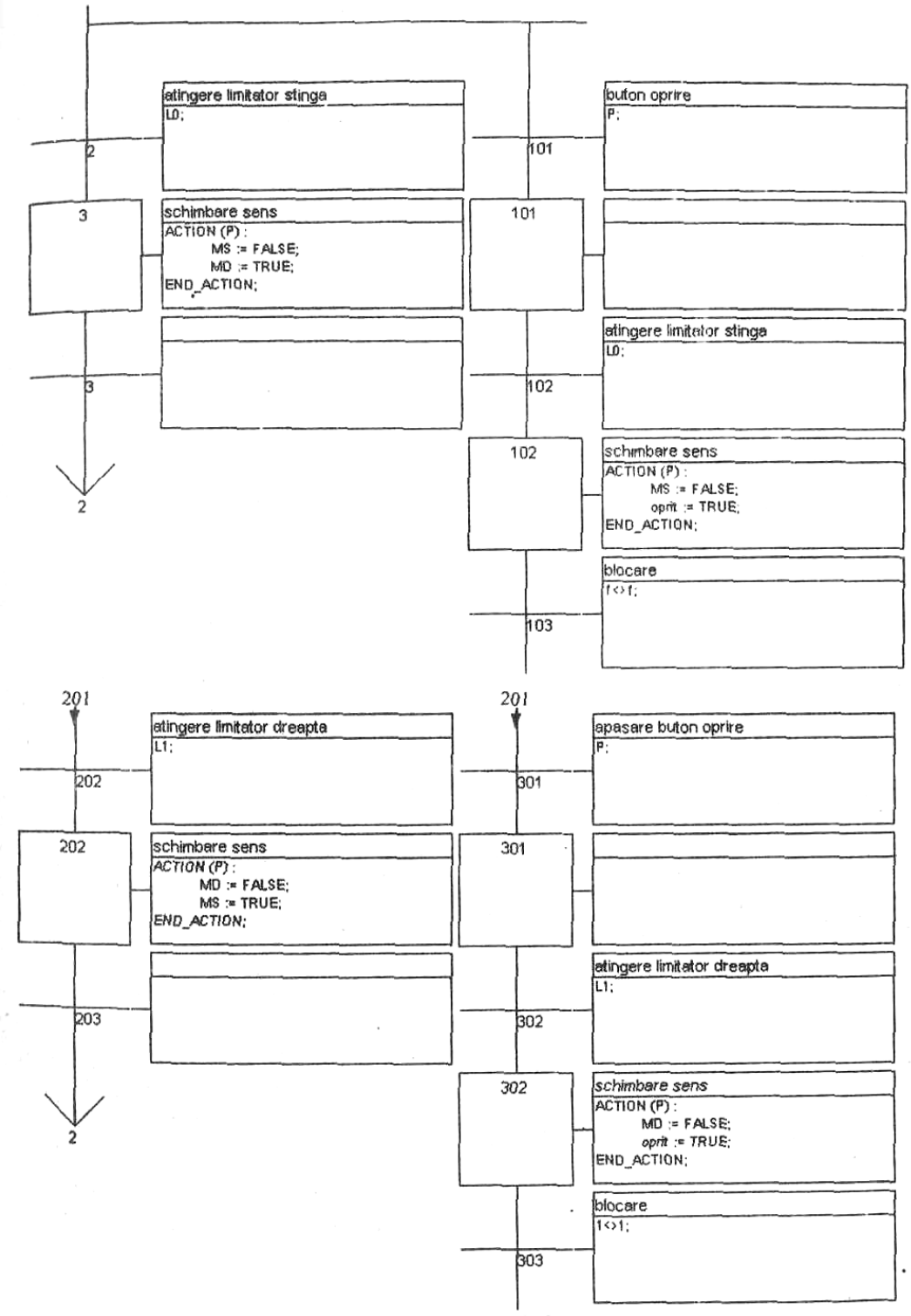


Fig. 5.7 Programul copil „Osc"

Comentarii:

- Funția REDGE este folosită pentru detectarea impulsurilor produse prin apăsarea butoanelor
- Comunicația între programe este realizată prin intermediul variabilei „oprit"
- Programul principal oprește execuția programului copil „Osc" atunci când variabila „oprit are valoarea „TRUE" sau mișcarea este întreruptă de apăsarea butonului E

Propunere:

- Să se modifice programul în condițiile în care la apăsarea butonului de oprire, ciclul să se încheie totdeauna când mobilul ajunge prima dată in partea dreaptă.

Problema 3: Detecția şi expulzarea automată a sticlelor fără dop

3. Descrierea procesului:

Una din fazele de producție într-o linie de îmbuteliere constă în aşezarea unui dop, ca urmare a încheierii secvenței de umplere. Sticlele se deplasează pe banda 1, separate de aceeaşi distanță şi cu viteză constantă. Scopul aplicației este detectarea şi extragerea sticlelor care ies din faza de închidere fără dopul corespunzător; pe lângă aceasta, dacă într-o perioadă determinată de timp (în acest caz 7 sticle), sunt rejectate mai mult de 3 sticle consecutive, trebuie activată o alarmă. Repornirea ciclului se face prin apăsarea butonului Pc. Pentru detecția sticlei defecte se conjugă acțiunile unui senzor inductiv, care detectează prezența dopului şi un echipament foto electric care semnalează prezența unei sticle.

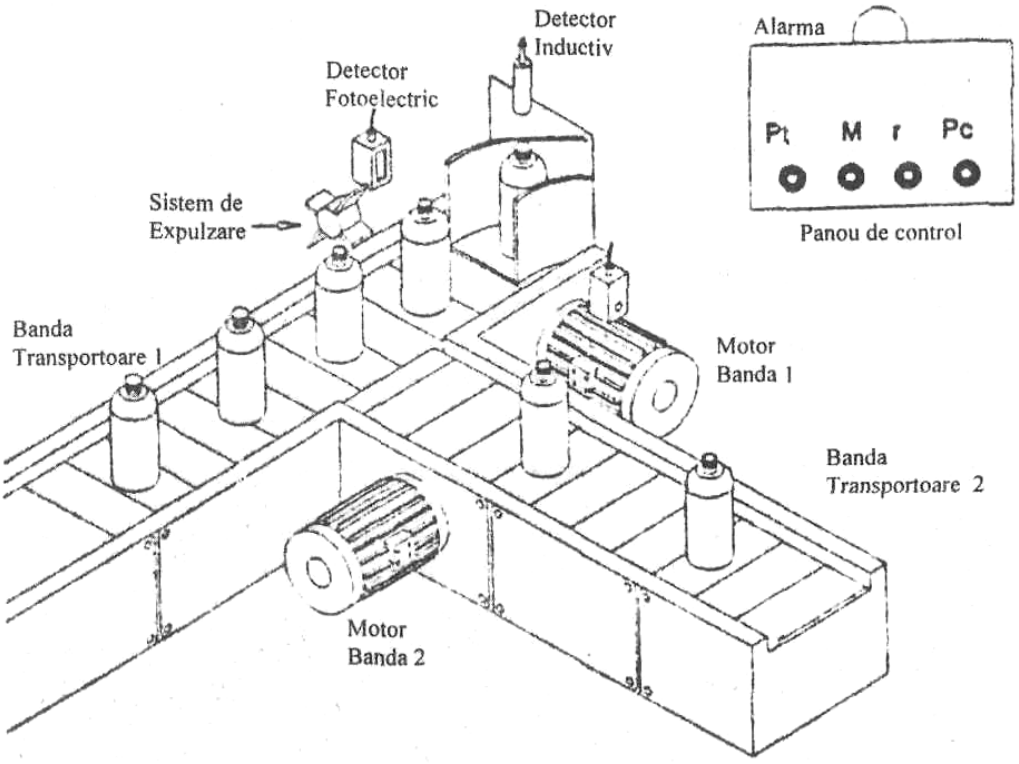


Fig. 5.8 Detecția şi expulzarea sticlelor fără dop

Procesul este inițiat prin apăsarea butonului M, care produce pornirea benzii transportoare 1. Atunci cînd se detectează o sticlă fără dop, este oprită banda 1 şi este pornită banda transportoare 2 (dacă era oprită). În momentul în care sticla fără dop se găseşte în zona de expulzare, este activat mecanismul de expulzare. Banda 1 va fi repornită în momentul în care sticla fără dop nu se mai găseşte în zona de expulzare (practic semnalul transmis de detectorul fotoelectric are valoarea logică fals). Banda 2 va fi oprită după 5 secunde de la începerea expulzării ultimei sticle.

Elemente de execuție:

- 2 motoare care acționează 2 benzi transportoare
- 1 dispozitiv de expulzare a sticlelor fără dop

Elemente de măsură:

- 1 detector inductiv pentru dopuri.
- 1 detector fotoelectric pentru sticle

2. Soluția de automatizare:

Pentru controlul acestei aplicații se alege un automat programabil de tip PEP Smart pentru care se dezvoltă un proiect Isagraf, cu un singur program principal, dar cu 2 secțiuni programate şi anume secțiunea secvențială şi secțiunea de sfârşit, end section ca în figura 5.9.

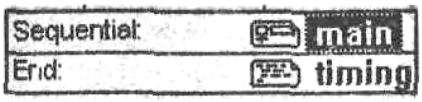


Fig. 5.9 Secțiunile proiectului

Dicționarul de variabile globale:

- *Variabile de intrare booleene:*
  - M : buton de pornire
  - R : buton de rearmare
  - I : detector inductiv
  - F : detector fotoelectric
  - Pc : buton de punere la zero şi oprire alarmă

- *Variabile de ieşire booleene:*
  - Banda_1 : comandă pornire/oprire bandă
  - Banda_2 : comandă pornire/oprire bandă 2
  - Alarmă : comandă alarmă
  - Expulzare : comandă dispozitiv de evacuare a sticlelor fără dop

- *Variabile interne de tip Integer:*
  - Nr_sticle : contorizează numărul de sticle fără dop expulzate consecutiv

- *Variabile interne de tip timer:*
  - Timer : folosit la contorizarea celor 5 secunde de activare a conveiorului 2

Programul „timing" din secțiunea de end este prezentat în figura 5.11

```
IF (timer >= t#5s) THEN
    Banda_2 : = FALSE;
    TSTOP (timer);
    timer := t#0s;
ELSE
ENDIF;
```

Fig. 5.11 Programul principal, secțiunea de „end"

Observații:

- secțiunea de end este necesară pentru ca banda 2 să poate fi oprită în orice moment dacă perioada ei de activare a expirat
- deoarece secțiunea de end se execută la fiecaie ciclu automat, testarea timerului se va face la fiecare ciclu automat
- contorizarea timpului este făcută cu ajutorul funcțiilor TSTART şi TSTOP
- dacă o sticlă trebuie să fie expulzată în timp ce o altă sticlă se găseşte pe conveiorul 2, timer-ul este resetat şi este reactivată incrementarea timer-ului

Propuneri:

- Să se construiască o diagramă Ladder pentru un automat de tip Allen Bradley, care să controleze acest proces

Programul principal „main” este prezentat în figura 5.10

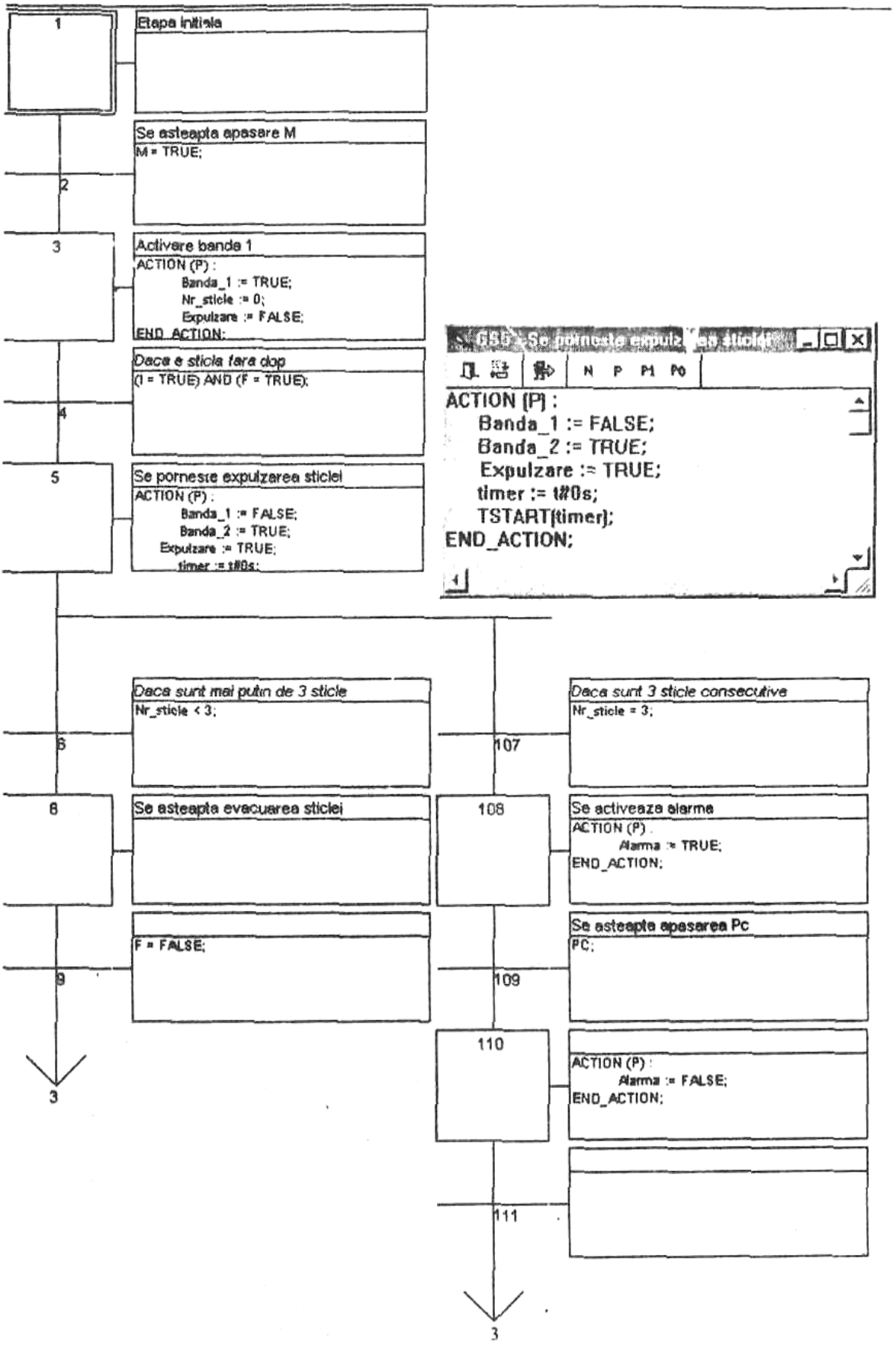


Fig. 5.10 Programul principal, secțiunea secvențială

Problema 4: Stație automată de spălat autovehicule

4. Descrierea procesului:

Scopul proiectării acestui sistem de control îl reprezintă automatizarea unei stații de spălat autovehicule. Vehiculele vor trebui să treacă succesiv prin 4 posturi de lucru, înmuiere, spălare cu detergent, clătire şi uscare. Procesul este inițiat de apăsarea unui buton de pornire, care determină activarea benzii transportoare iar vehiculele vor trece succesiv prin cele 4 posturi. Bariera, în condiții normale, trebuie să stea ridicată şi semaforul dezactivat. Când în stație sunt detectate 4 vehicule, câte unul în fiecare post, bariera trebuie coborâtă şi semaforul activat, indicând faptul că nu se mai poate trece. Atât bariera cât şi semaforul rămân în această stare până când cele 4 vehicule au părăsit stația, moment în care bariera trebuie ridicată şi semaforul dezactivat.

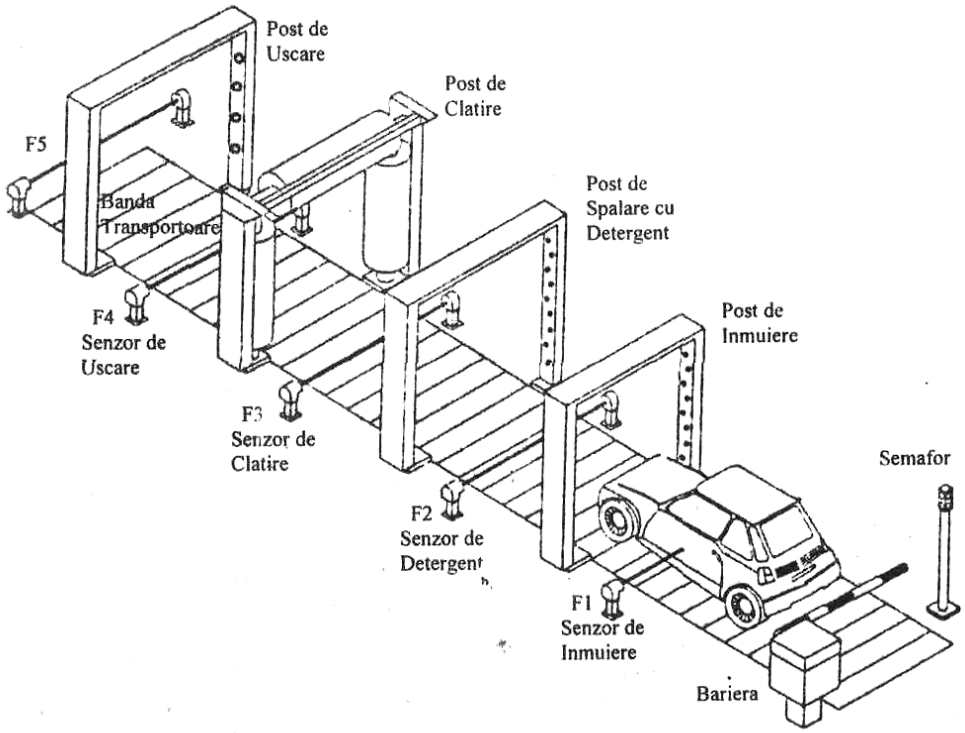


Fig. 5.12 Stație automată de spălat autovehicule

În momentul în care se activează celula fotoelectrică 1, se va activa postul 1. Când se activează fotocelula 2 iar fotocelula 1 nu mai este activată, se dezactivează postul 1. In momentul activării fotocelulei 2 se activează postul 2. Acesta va fi dezactivat când fotocelula 2 nu este activă dar fotocelula 3 este activă. Analog pentru postul 3. Postul 4 se dezactivează când fotocelula 4 se dezactivează iar fotocelula 5 se activează

Elemente de execuție:

- motorul benzii transportoare
- motorul barierei cu 2 sensuri de rotație
- 1 semafor
- 4 posturi de lucru

Elemente de măsură:

- 5 celule fotoelectrice
- 2 limitatoare de cursă ale barierei

2. Soluția de automatizare:

Pentru controlul acestei aplicații se alege un automat programabil de tip PEP Smart pentru care se dezvoltă un proiect Isagraf, proiect ce constă din 6 programe SFC ce rulează în paralel. Structura proiectului Isagraf este prezentată în figura 5.13.

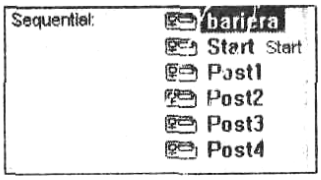


Fig. 5.13 Programele componente ale proiectului Isagraf

Dicționarul de variabile globale:

- *Variabile de intrare booleene:*
  - Start : contact de pornire
  - Limita_SUS : limitator de cursă sus pentru barieră
  - Limita_JOS : limitator de cursă jos pentru barieră
  - Foto_1 : fotocelula postului 1
  - Foto_2 : fotocelula postului 2
  - Foto_3 : fotocelula postului 3
  - Foto_4 : fotocelula postului 4
  - Foto_5 : fotocelula postului 5

- *Variabile de ieşire booleene:*
  - Banda : comandă pornire/oprire bandă
  - Bariera_sus : comandă ridicare barieră
  - Bariera_jos : comandă coborâre barieră
  - Semafor : comandă activare/dezactivare semafor
  - Înmuiere : comandă activare dezactivare post înmuiere
  - Detergent : comandă activare dezactivare post detergent
  - Clătire : comandă activare dezactivare post clătire
  - Uscare : comandă activare dezactivare post uscare

Comentarii:

- Toate programele dezvoltate sunt independente şi rulează în paralel, fapt ce uşurează foarte mult înțelegerea programului

Propunere:

- Să se modifice programul în cazul în care bariera se va ridica atunci când se eliberează primul post de lucru.
- Să se modifice proiectul astfel încât să se dezvolte im singur program care să automatizeze acest proces

Programul „Bariera” este prezentat în figura 5.14
Programul „Start” este prezentat în figura 5.15
Programul „Post1” este prezentat în figura 5.16
Programul „Post2” este prezentat în figura 5.17
Programul „Post3” este prezentat în figura 5.18
Programul „Post4” este prezentat în figura 5.19

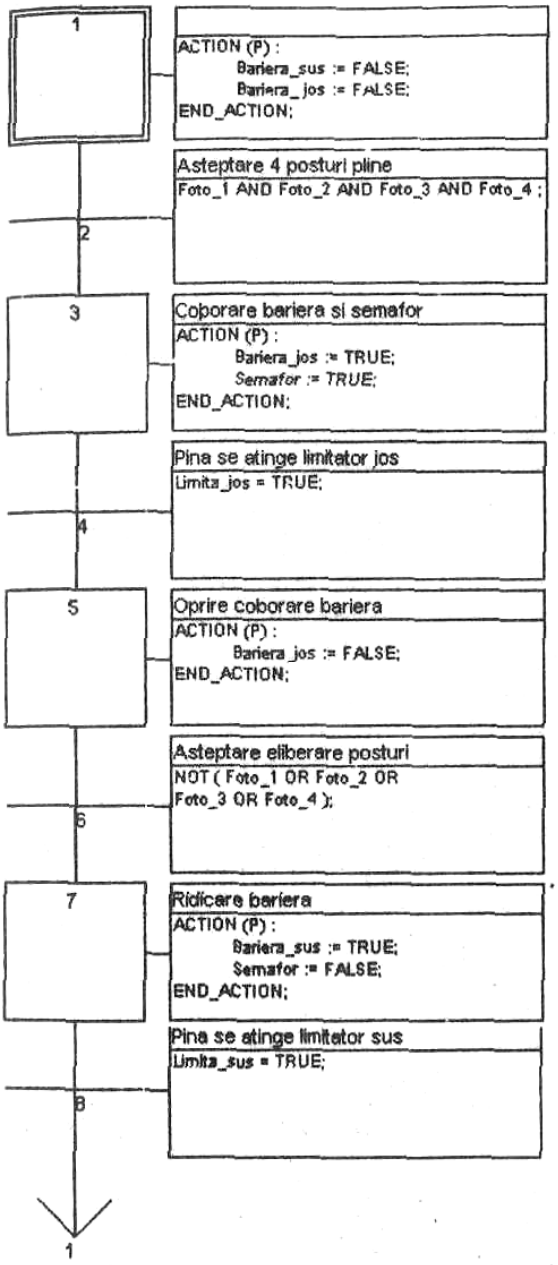


Fig. 5.14 Programul „Bariera"

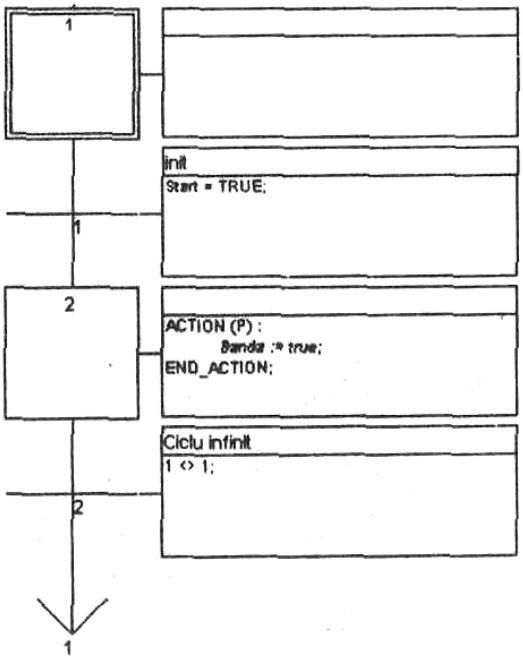


Fig.5.15 Programul „Start"

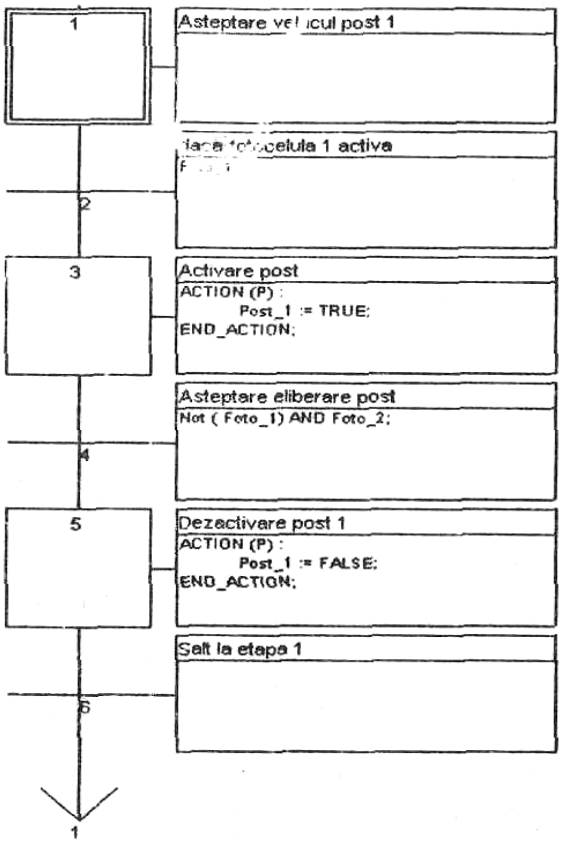


Fig.5.16 Programul „Post1"

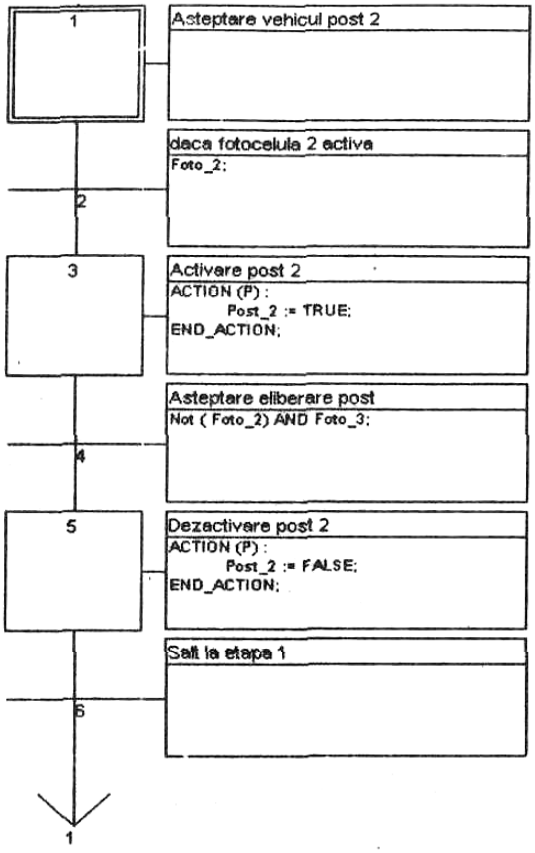


Fig.5.17 Programul „Post2"

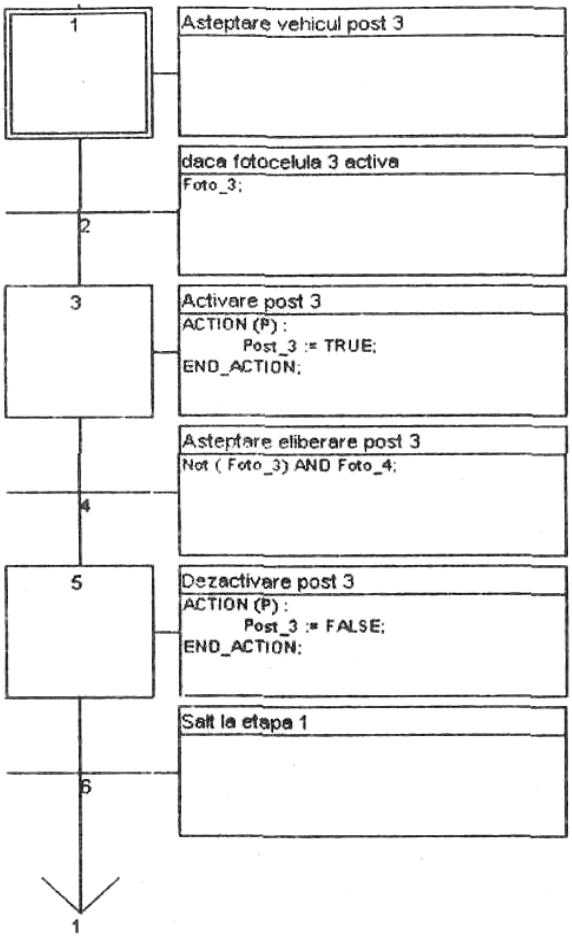


Fig.5.18 Programul „Post3"

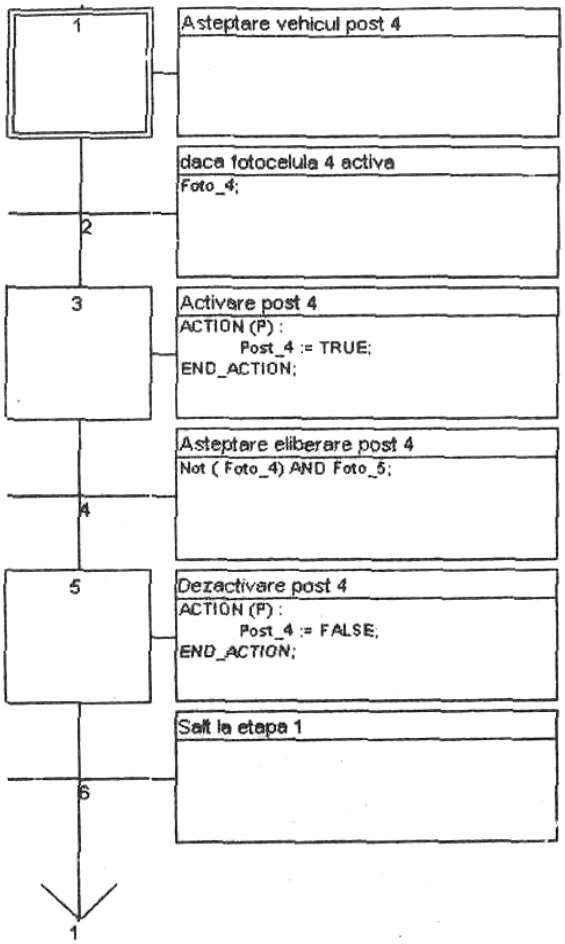


Fig.5.19 Programul „Post4"

Problema 5: Elevator clasificator de pachete

1. Descrierea procesului

Pe o bandă transportoare vin 2 tipuri de pachete (mic şi mare). Tipul pachetului este determinat de un cântar, ulterior pachetele fiind transportate în direcţii diferite în funcţie de tipul pachetului. Procesul porneşte cu transportul unui pachet către cântar; aici pachetul este cântărit fiind astfel identificat în funcţie de greutatea citită. În continuare pachetul este transportat pe banda 1 până la planul elevator. Cilindrul C ridică pachetele. Apoi pachetele sunt transportate diferit; pachetele mici sunt plasate pe banda 2 de cilindrul A, iar pachetele mari sunt aşezate pe banda 3 de cilindrul B. Cilindrul elevator C se retrage doar când cilindrii A şi B au atins poziţia finală.

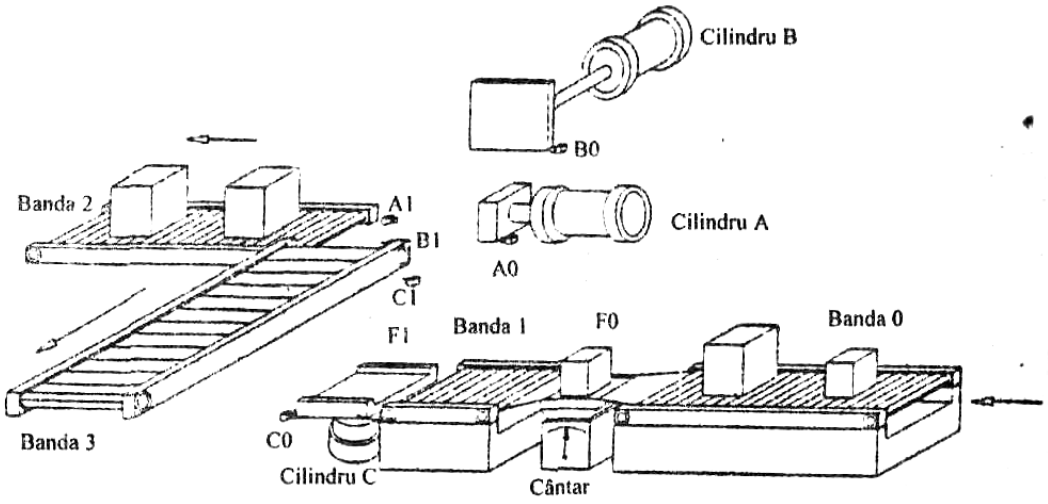


Fig. 5.20 Elevator clasificator pentru pachete

Elemente de execuţie:

- 3 cilindri cu dublu efect (A, B, C)
- basculă însărcinată cu clasificarea pachetelor
- 4 benzi transportoare

Elemente de măsură

- 6 limitatoare decursă
- 2 detectoare de poziţie

2. Soluţia de automatizare

Pentru controlul acestei aplicaţii s-a ales un automat programabil de tip PEP Smart pentru care s-a dezvoltat un proiect Isagraf ce cuprinde 5 programe secvenţiale ce rulează în paralel şi un program în secţiunea de „Begin", program ce se execută la începutul fiecărui ciclu automat. Structura proiectului Isagraf este prezentată în figura 5.21

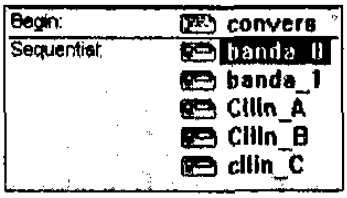


Fig. 5.21 Structura proiectului Isagraf

Dicționarul de variabile globale:

- *Variabile de intrare booleene:*
  - Foto_0 : fotocelula 0
  - Foto_1 : fotocelula 1
  - A0 : Limită retragere cilindru A
  - Al : Limită avans cilindru A
  - B0 : Limita retragere cilindru B
  - Bl : Limită avans cilindru B
  - C0 : Limita coborâre cilindru C
  - C1 : Limită ridicare cilindru C

- *Variabile de ieşire booleene:*
  - Banda_0 : activare / dezactivare banda 0
  - Banda_1 : activare / dezactivare banda 1
  - Banda_2 : activare /dezactivare banda2
  - Banda_3 : activare / dezactivare banda 3
  - A_avansat : avans cilindru A
  - A_retras : retragere cilindru A
  - B_avansat : avans cilindru B
  - B_retras : retragere cilindru B
  - C_ridicare : ridicare cilindru C
  - C_coborâre : coborâre cilindru C

- *Variabile interne booleene:*
  - eroare : cod de eroare la cântărirea pachetelor

- *Variabile globale analogice*
  - Cântar : variabilă internă, reprezintă valoarea reală a greutății de pe cântar (integer)
  - traductor_cantar : variabliă de intrare, valoarea primită de la traductorul cântarului (integer, între 0-4096)
  - pachet_actual : variabilă internă, în care se memorează tipul pachetului actual ce urmează a fi transportat
  - pachet_viitor : variabilă internă, în care se memorează tipul pachetului de pe cântar (următorul ce va fi transportat)
  - lim_inf : constantă, greutatea minimă a pachetului mic
  - lim_sup_mic : greutatea maximă a pachetului mic
  - mic : constantă cu valoarea 1
  - mare : constantă cu valoarea 2

Programul „Banda_0” este prezentat în figura 5.22
Programul „Banda_1” este prezentat în figura 5.23
Programul „Cilin_A” este prezentat în figura 5.24
Programul „Cilin_B” este prezentat în figura 5.25
Programul „Cilin_C” este prezentat în figura 5.26

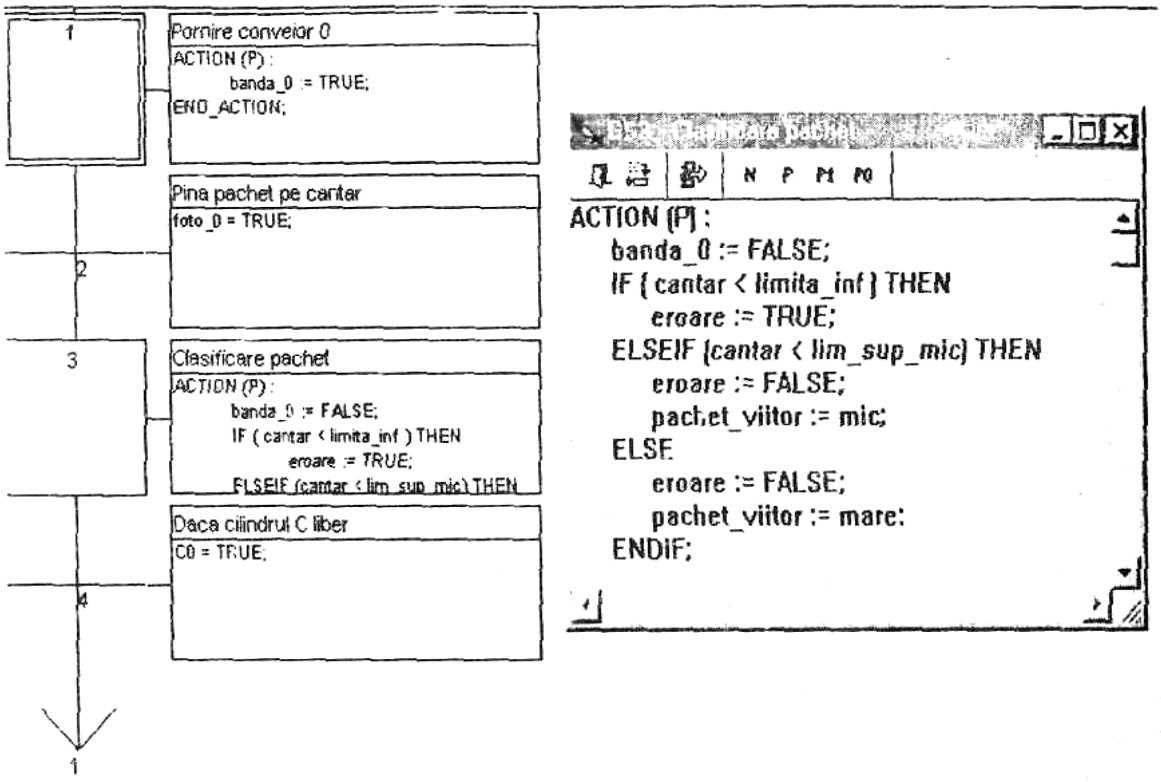


Fig 5 22 Programul Banda_0

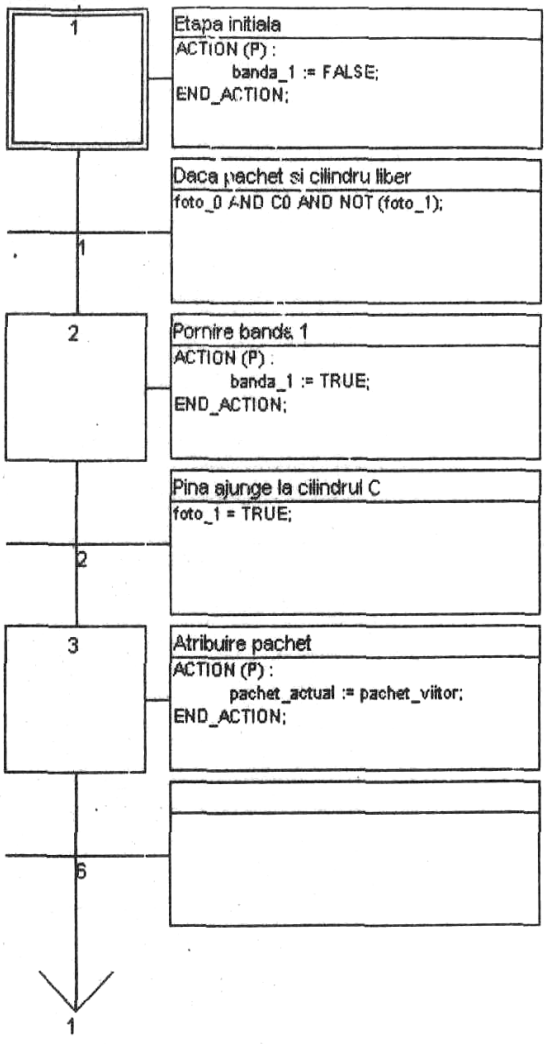


Fig.5.23 Programul „Banda_1"

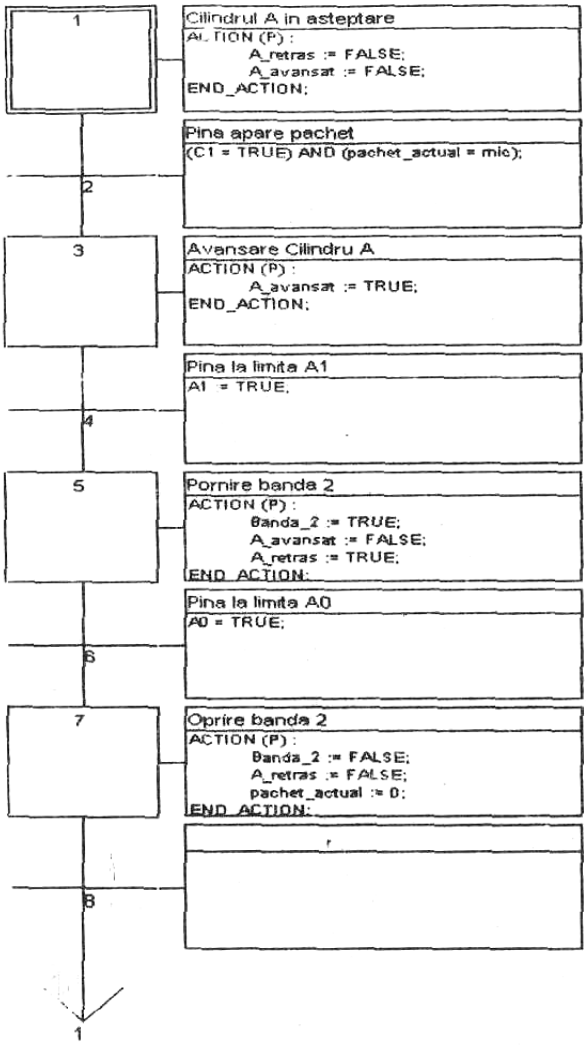


Fig. 5. 24 Programul Cilin_A

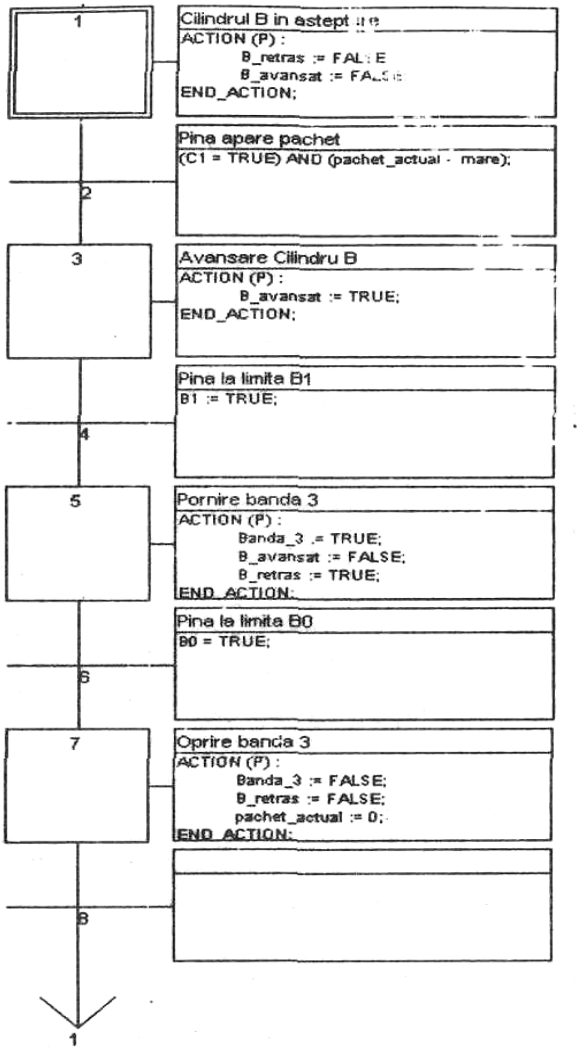


Fig.5.25 Programul Cilin_B

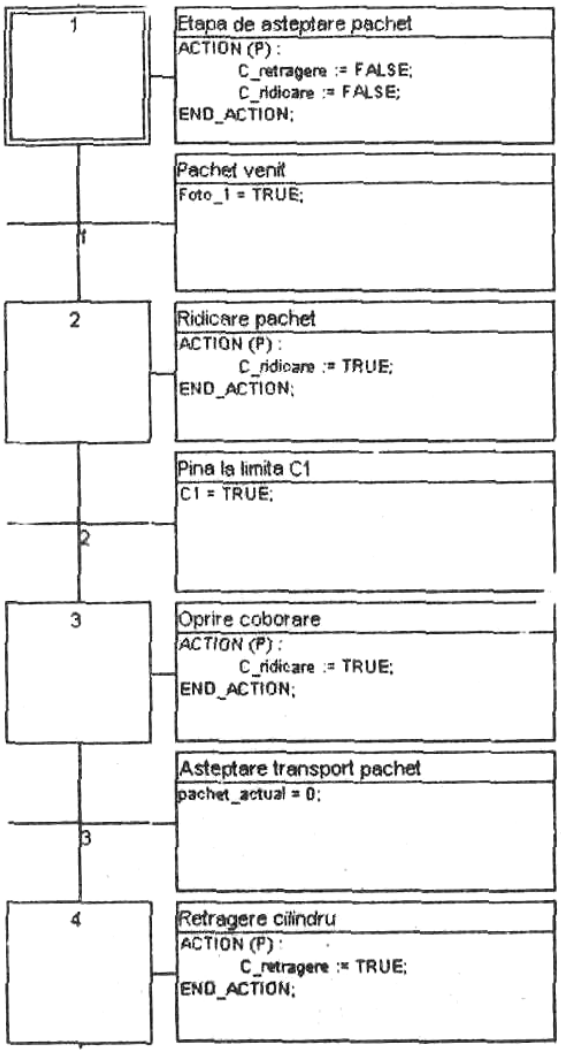


Fig.5.26 Programul "Cilin_C"

Programul „convers" din secțiunea Begin

Programul „convers" realizează conversia din unități CAN (Convertor Analog Numeric) în valori exprimate în unități de măsură inginereşti. Intrarea analogică a modulului de intrare lucrează pe 12 biți şi măsoară un curent de 0 - 20 mA, dar traductorul de la cântar generează un curent de 4 – 20mA. În această situație trebuie făcută o translație de scală. Se observă că la valoarea minimă a domeniului de măsură traductorul generează 4 mA, corespunzătoare valorii 819 citită de automat, valoare pentru care automatul trebuie să indice vloarea minimă a mărimii măsurate. Astfel formula de conversie este:

Val _ing = (Val_cit-819)*(Ds-Di)/(4095-819)

Unde:

- Val_ing : valoarea în unități inginereşti
- Val_cit : valoarea citită în unități CAN
- Di : domeniul inferior de măsură
- Ds : domeniul superior de măsură

În cazul nostru Ds=l00, Di=0, astfel încât instrucțiunea executată în programul „convers" este

cântar := INT(((REAL(traductor_cantar) - 819)*100.0) / (3276);

Propunere:

- Să se modifice programul de conversie în cazul în care traductorul de temperatură are ca domeniu -15 +150 grade şi generează un curent în gama 2-10 mA

Problema 6: Controlul temperaturii unui lichid

1. Descrierea procesului:

Problema constă în menţinerea temperaturii unui lichid între 2 valori determinate (60 şi 65 grade), în timp ce nivelul în cele 2 rezervoare păstrează o capacitate determinată. Dacă temperatura se găseşte între limitele fixate, valva 1 se va deschide iar valva 2 se va deschide până când rezervorul 2 ajunge la capacitatea fixată; în acel moment, valva 2 se va închide şi va rămâne aşa până când lichidul din rezervorul 2 se va găsi sub limita fixată. Când temperatura atinge marginile de temperatură fixată, valvele de intrare şi de ieşire se vor închide (indiferent dacă rezervorul 2 şi-a recuperat nivelul) şi vor rămâne închise până când temperatura va fi reglată. Totdeauna va fi prioritară variația temperaturii față de variația nivelului de lichid.

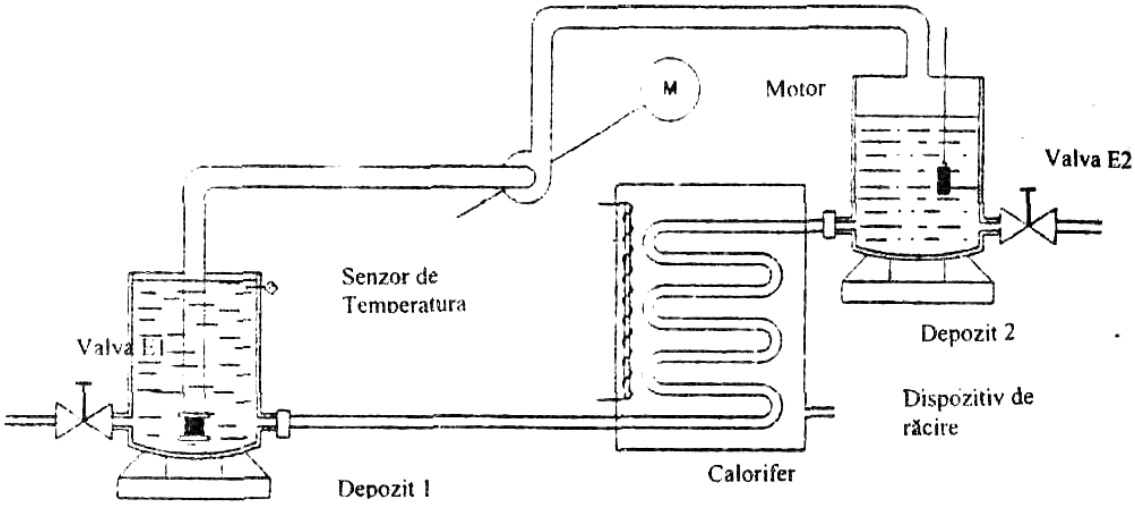


Fig. 5.27 Controlul temperaturii unui lichid

Elemente de execuţie:

- 1 pompă cu motorul său
- 1 motor al echipamentului de pompat aerul
- 2 electrovalve
- 1 rezistenţă

Elemente de măsură:

- 2 senzori de nivel
- 1 traductor de temperatură

2. Soluția de automatizare

Pentru controlul acestei aplicații s-a ales un automat programabil de tip PEP Smart pentru care s-a dezvoltat un proiect Isagraf ce cuprinde un program principal, „Main” şi programul secțiunii de „Begin”, numit „Convers”.

Dicționarul de variabile globale:

- Variabile de intrare booleene:
  - senzor_plin : are valoarea TRUE când depozitul 2 este plin
  - senzor_gol : are valoarea TRUE când depozitul 2 este gol

- Variabile de ieşire booleene:
  - valva_1 : comanda electrovalvei 1, atunci când are valoarea TRUE electrovalva 1 se închide
  - valva_2 : comanda electrovalvei 2, atunci când are valoarea TRUE electrovalva 2 se închide
  - rezistenta : comandă cuplarea / decuplarea rezistenței
  - răcire : comandă dispozitivul de pompat aer
  - pompa : comandă motorul pompei

- Variabile analogice:
  - temp : variabilă internă de tip Real, reprezintă temperatura apei în grade Celsius
  - temp_citita : variabilă de intrare de tip Real ce reprezintă valoarea analogică (între 0 şi 4095) primită de la traductorul de temperatură

Programul „Convers" al secțiunii de Begin

Programul „convers" realizează conversia din unități CAN (Convertor Analog Numeric) în valori exprimate în unități de măsură inginereşti. Intrarea analogică a modulului de intrare lucrează pe 12 biți şi măsoară un curent de 0-20 mA, dar traductorul de la cântar generează un curent de 4-20 mA. În această situație trebuie făcută o translație de scală. Se observă că la valoarea minimă a domeniului de măsură traductorul generează 4 mA, corespunzătoare valorii 819 citită de automat, valoare pentru care automatul trebuie să indice valoarea minimă a mărimii măsurate. Astfel formula de conversie este:

Val_ing = (Val_cit-819)*(Ds-Di) / (4095-819)

Unde:
- Val_ing : valoarea în unități inginereşti
- Val_cit : valoarea citită în unități CAN
- Di : domeniul inferior de măsură
- Ds : domeniul superior de măsură

În cazul nostru Ds=100, Di=0, astfel încât instrucțiunea executată în programul „convers" este

temp := INT(((REAL(temp_citita)-819)*100.0) / (3276);

Comentarii:
- Programul foloseşte variabile analogice deoarece evenimentele ce produc modifcări în sistem depind de o mărime cu variație continuă (temperatura)

Programul „Main" este prezentat în figura 5.28

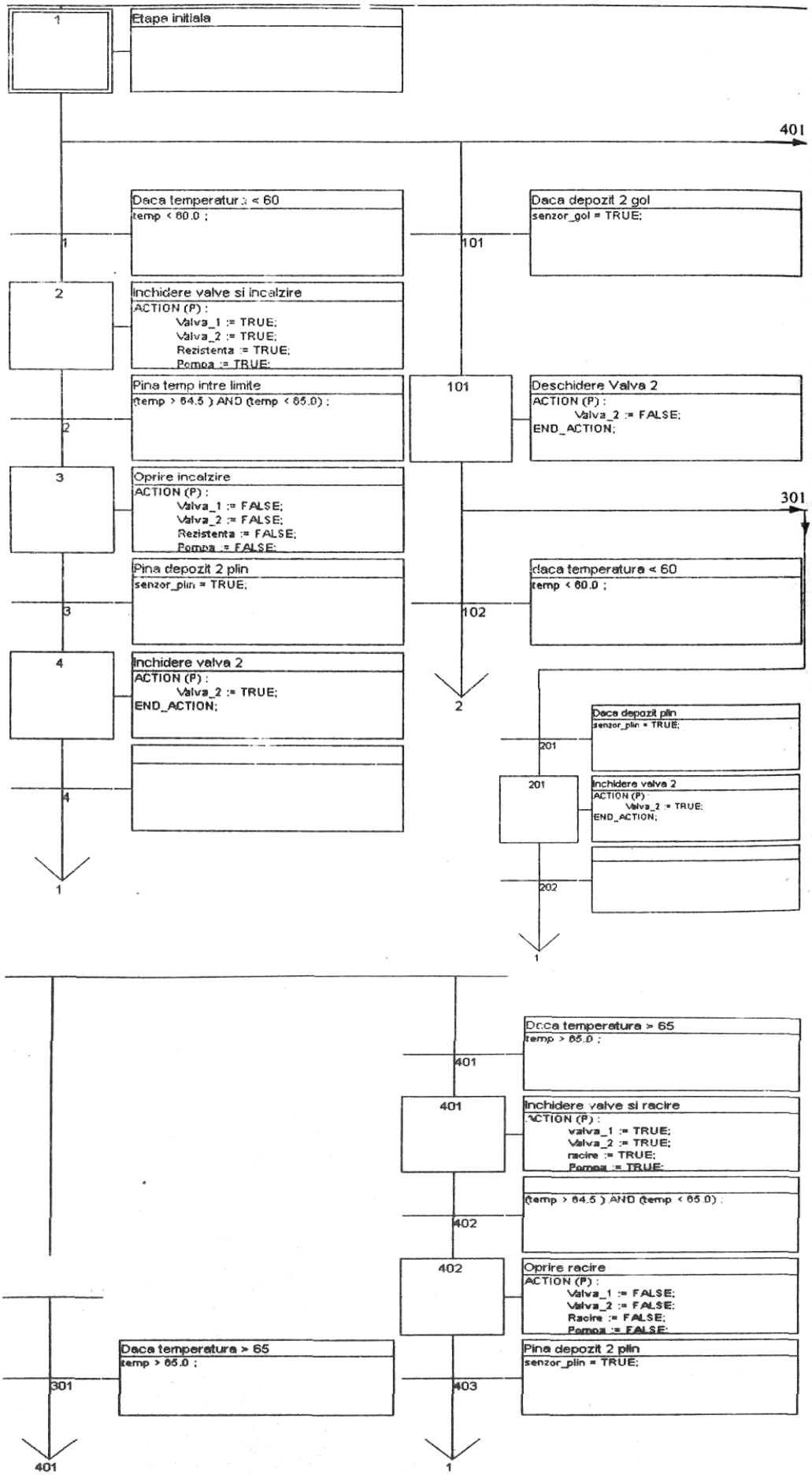


Fig.5.28 Programul „Main"

Problema 7: Dozare şi malaxare automată

1. Descrierea procesului:

Un malaxor pivotant primeşte produsele A şi B cântărite de bascula C şi brichete solubile aduse una câte una pe bandă transportoare. Automatizarea permite realizarea unei amestecări a celor 3 produse. Ciclul de realizat este următorul: la acționarea butonului de alimentare se porneşte cântărirea şi alimentarea produselor în următorul mod:

- cântărirea produsului A prin deschiderea valvei Va, până la referința A
- cântărirea produsului B prin deschiderea valvei Vb, până la referința B
- apoi, golirea basculei în malaxor prin deschiderea valvei Vc până la referința zero
- simultan cu precedentele operații are loc alimentarea malaxorului cu 2 brichete solubile

Ciclul se termină cu rotația malaxorului un anumit timp t şi apoi pivotarea lui, menținându-se rotația în timpul golirii

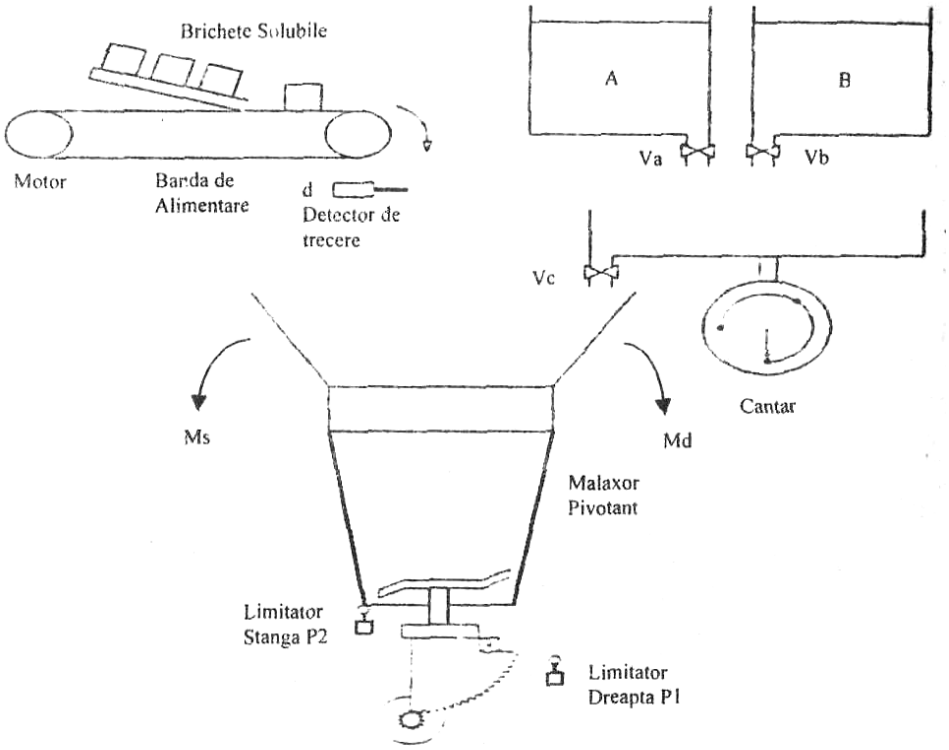


Fig 5.29 Proces de dozare şi malaxare

Elemente de execuție:

- 3 electrovalve (Va, Vb, Vc)
- Motorul benzii transportoare cu un singur sens de rotație
- Motorul de rotație al malaxorului
- Motorul de pivotare al malaxorului, cu două sensuri de rotație

Elemente de măsură:

- 3 senzori de greutate, pentru referințele A, B şi zero
- 2 limitatoare de cursă
- 1 detector de trecere

2. Soluții de automatizare:

2.1 Varianta 1 : implementarea în mediul Isagraf

Această variantă presupune utilizarea unui automat programabil de tip PEP Smart, pentru care s-a dezvoltat un proiect Isagraf ce cuprinde 1 program principal.

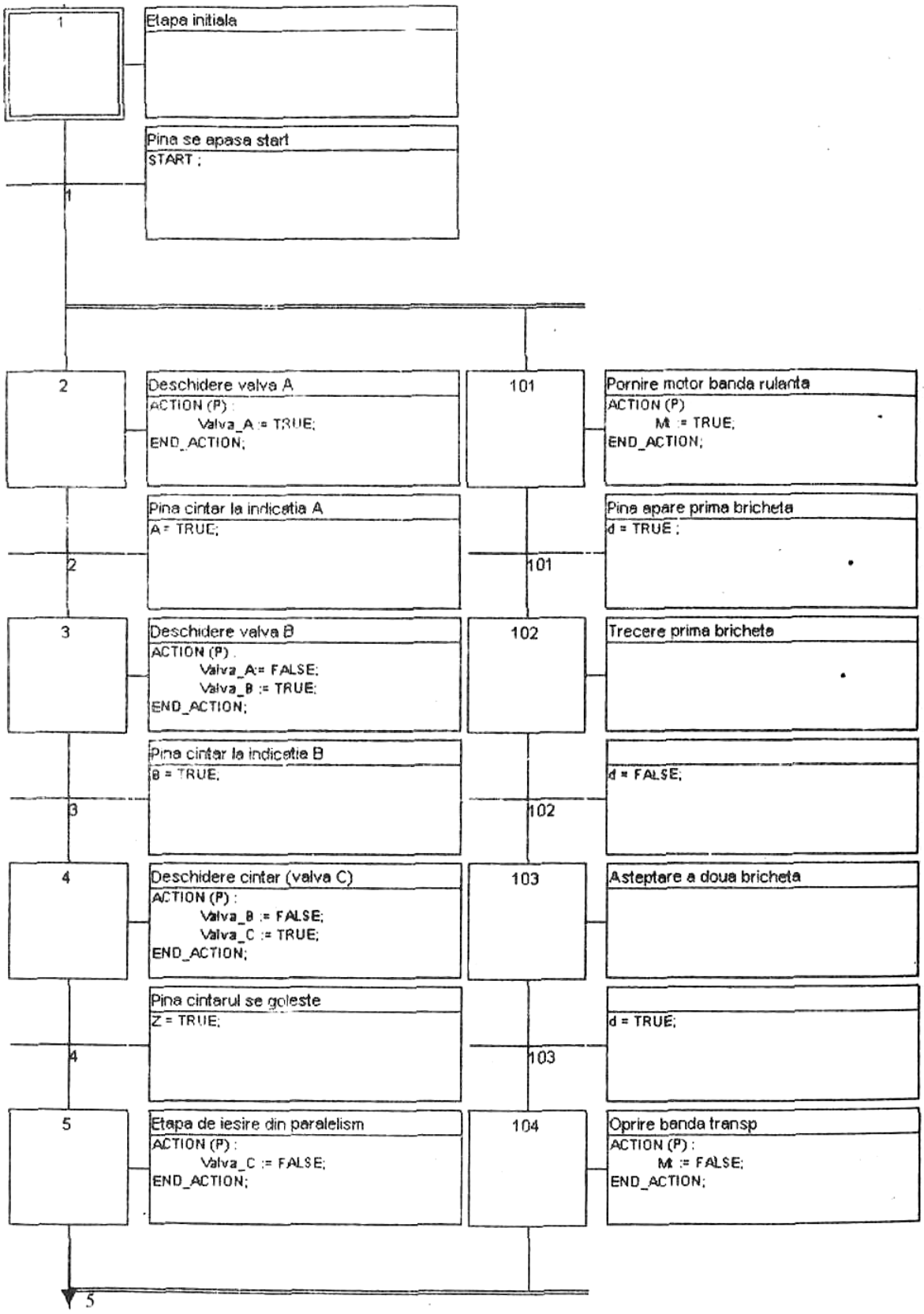
Dicționarul de variabile globale:

- *Variabile de intrare booleene:*
  - Start : buton de pornire
  - d : detector de trecere brichete solubile
  - A : detectorul greutății produsului A
  - B : detectorul greutății produselor A + B
  - Lim_stanga : limitator stânga malaxor
  - Lim_dreapta : limitator dreapta malaxor

- *Variabile de ieşire booleene*
  - Valva_A : comandă deschiderea / închiderea valvei A
  - Valva_B : comandă deschiderea / închiderea valvei B
  - Valva_C : comandă deschiderea / închiderea valvei C
  - Mt : comandă motorul benzii transportoare
  - Rotire : comandă rotația malaxorului
  - Piv_dreapta : comandă pivotarea către dreapta a malaxorului
  - Piv_ stânga : comandă pivotarea către stânga a malaxorului

- *Variabile globale de tip timer:*
  - Timer : temporizare folosită la rotația malaxorului

Programul principal „Main” este prezentat în figura 5.30



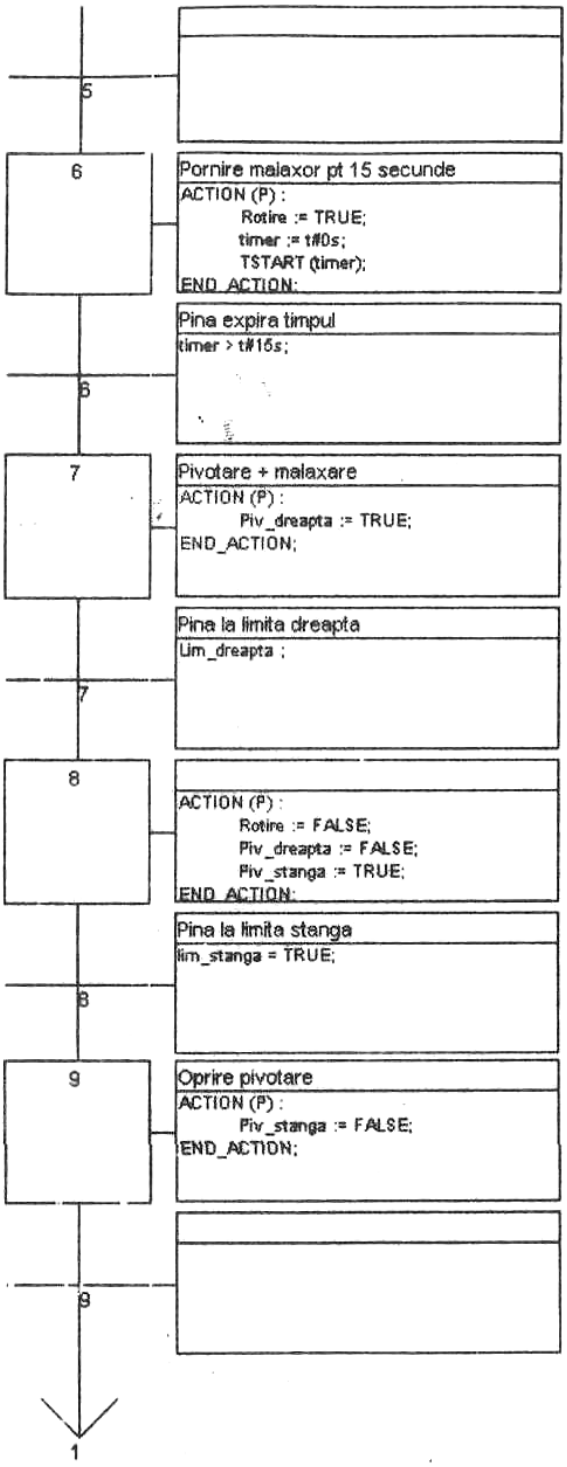


Fig. 5.30 Programul principal

2.2 Varianta 2 : implementarea în limbajul Ladder Diagram

Această variantă presupune folosirea unui automat programabil Allen Bradley, tip SLC500, pentru care se dezvoltă o diagramă de tip Grafcet şi o diagramă de tip Ladder

Diagrama Grafcet este prezentată in figura 5.31:

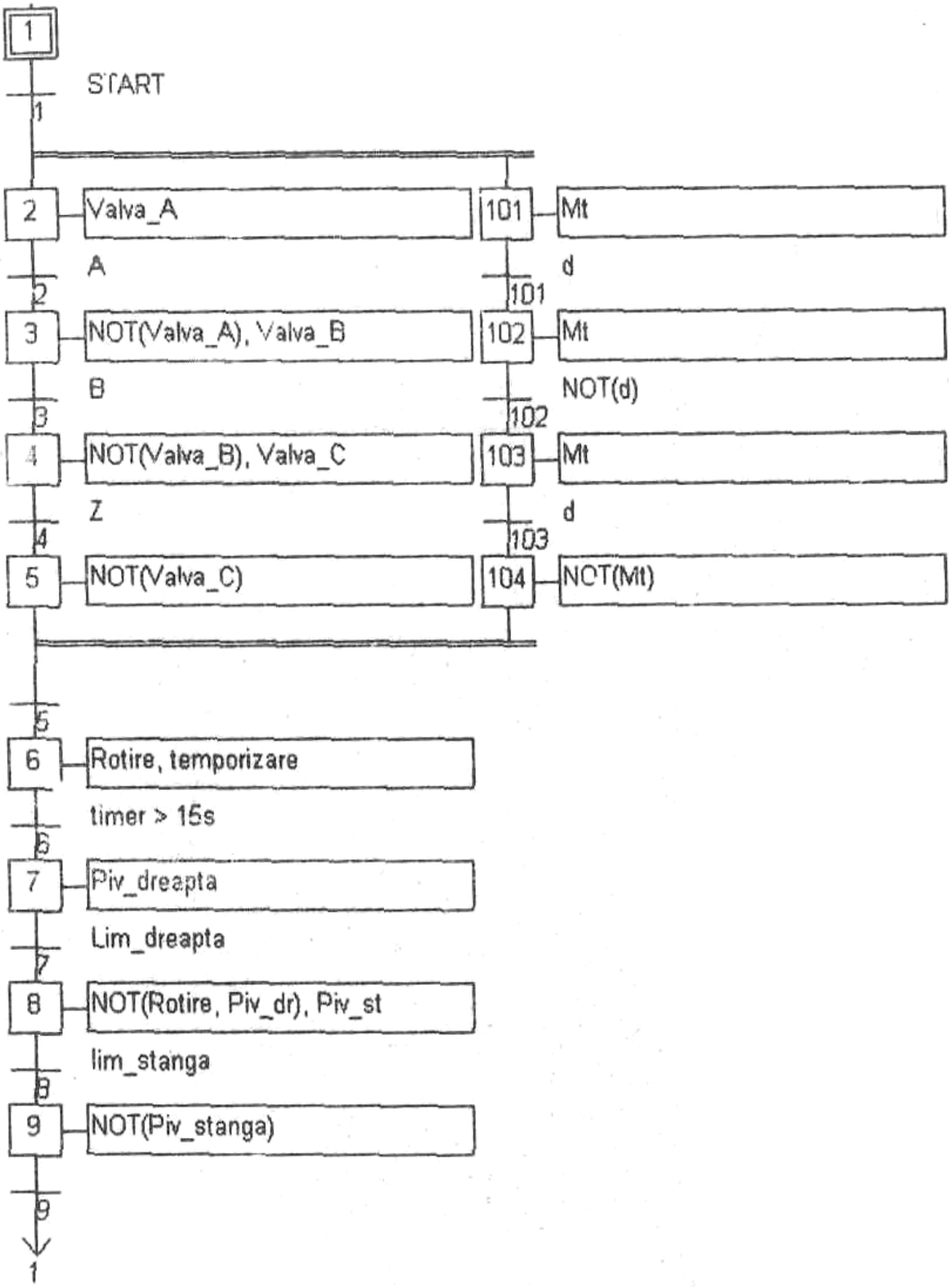


Fig. 5.31 Diagrama Grafcet

Asocierea etapelor cu biți din fişierul de bit B3 şi alegerea fişierului de timer este prezentată în tabelul 5.4.

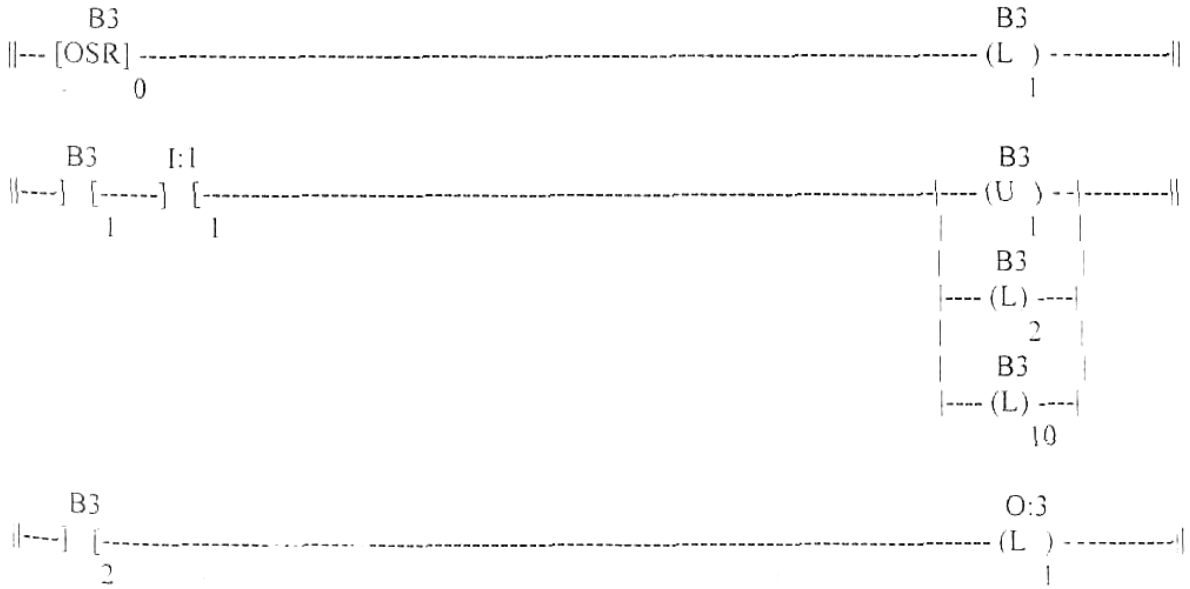
Tabelul 5.3

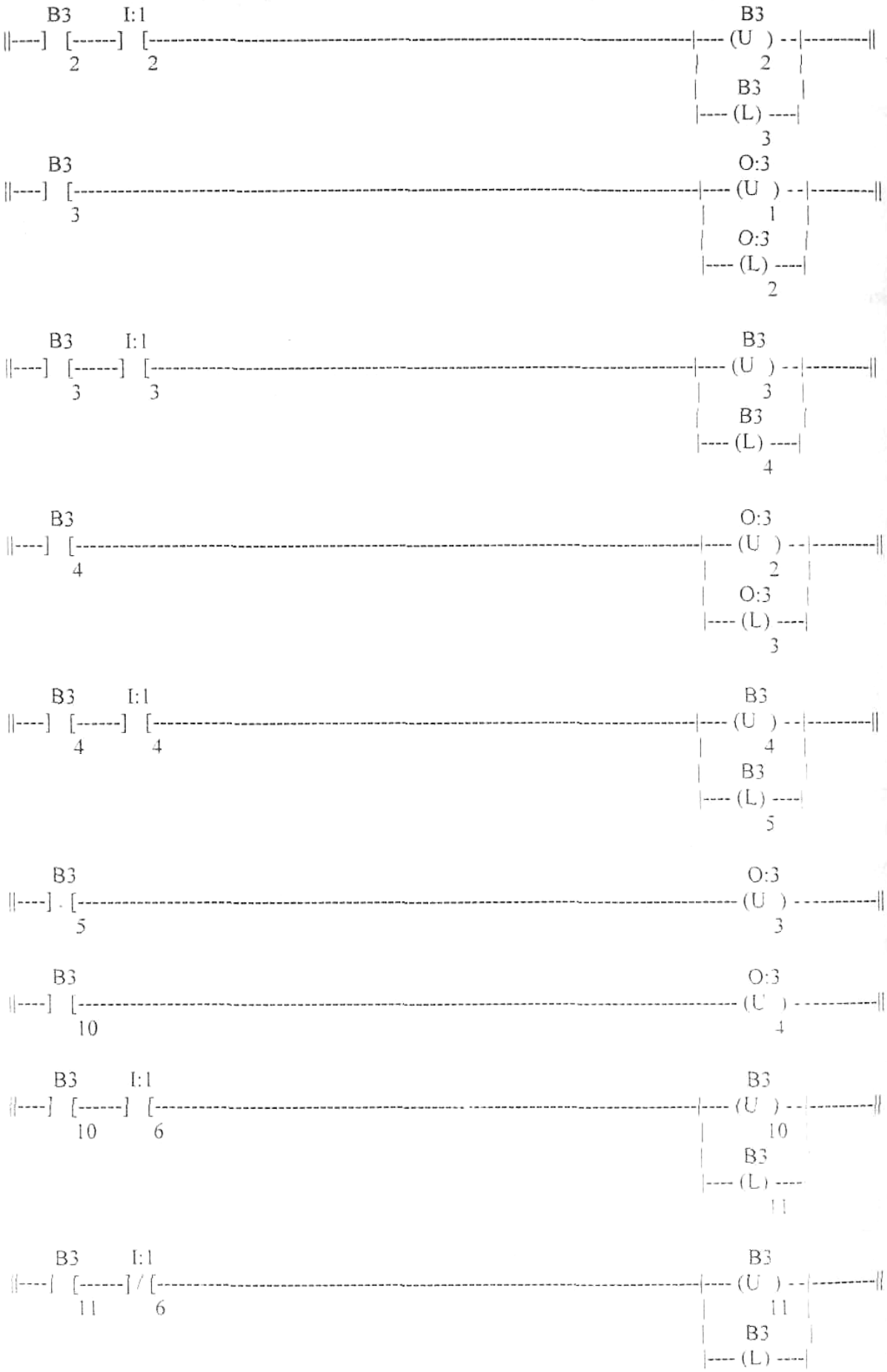
Intrare fizică	Adresă internă	Ieşire fizică	Adresa internă
START	1:1/1	Valva A	0:3/1
A	1:1/2	Valva B	0:3/2
B	1:1/3	Valva C	0:3/3
Z	1:1/4	Mt	0:3/4
lim_dreapta	1:1/5	Rotire	0:3/5
d	1:1/6	Piv_dreapta	0:3/6
lim stânga	1:1/7	Piv_stanga	0:3/7

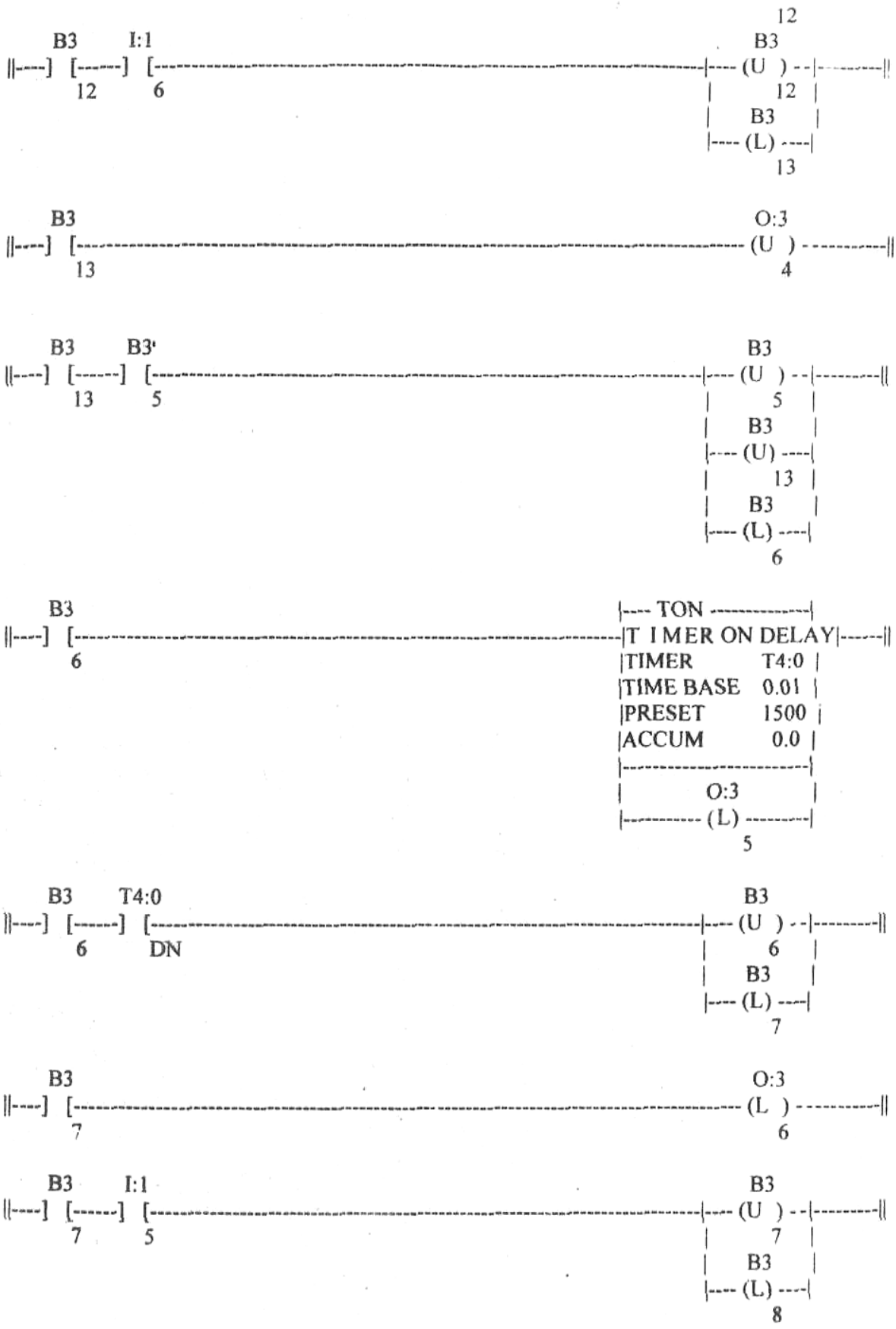
Tabelul 5.4

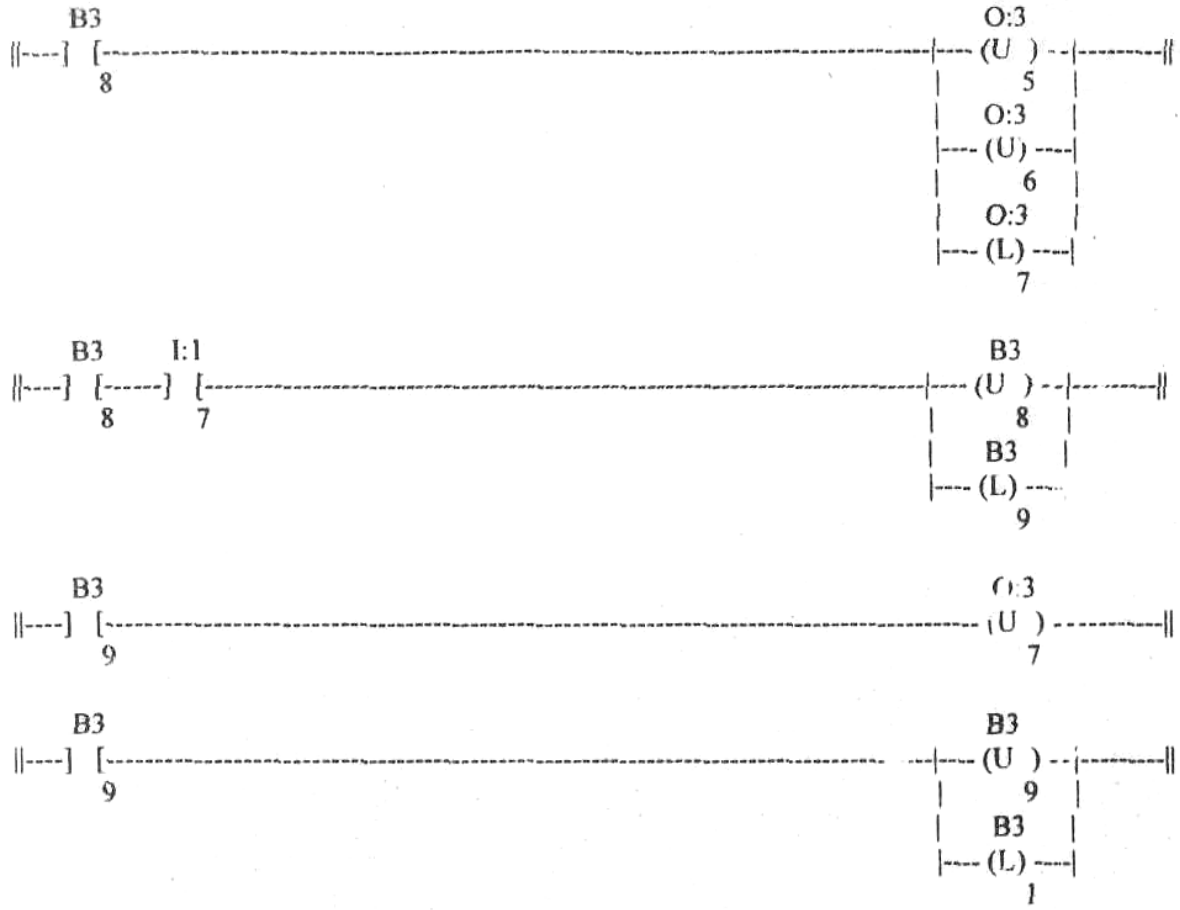
Etapa	Adresă bit	Temporizare	Fişier de timer
l	B3/1	Temporizare 1	T4:0
2	B3/2		
3	B3/3		
4	B3/4		
5	B3/5		
6	B3/6		
7	B3/7		
8	B3/8		
9	B3/9		
101	B3/10		
102	B3/11		
103	B3/12		
104	| B3/13		

Diagrama Ladder este prezentată în continuare:




B3 I:1
2 2
B3
(U) 2
B3
(L) 3
B3
3
O:3
(U) 1
O:3
(L) 2
B3 I:1
3 3
B3
(U) 3
B3
(L) 4
B3
4
O:3
(U) 2
O:3
(L) 3
B3 I:1
4 4
B3
(U) 4
B3
(L) 5
B3
5
O:3
(U) 3
B3
10
O:3
(U) 4
B3 I:1
10 6
B3
(U) 10
B3
(L) 11
B3 I:1
11 6
B3
(U) 11
B3
(L)


12
B3
B3
I:1
(U)
12
6
12
B3
(L)
13
B3
O:3
(U)
13
4
B3
B3'
B3
(U)
13
5
5
B3
(U)
13
B3
(L)
6
B3
TON
TIMER ON DELAY
TIMER T4:0
TIME BASE 0.01
PRESET 1500
ACCUM 0.0
6
O:3
(L)
5
B3
T4:0
B3
(U)
6
DN
6
B3
(L)
7
B3
O:3
(L)
7
6
B3
I:1
B3
(U)
7
5
7
B3
(L)
8



Comentarii:

- În diagrama SFC se pot observa cele 2 secvențe paralele, aducerea lichidelor şi aducerea brichetelor în malaxor. Ieşirea din paralelism se face atunci când ambele secvențe s-au încheiat
- Temporizarea a fost realizată cu instrucțiunile TSTART şi TSTOP

Propunere:

- Să se modifice diagrama Ladder folosind pentru activarea ieşirilor instrucțiunea OTE (acolo unde este posibil)

Problema 8: Umplerea şi astuparea automată a sticlelor

1. Descrierea procesului:

Scopul acestei aplicaţii îl constituie controlul unui sistem de umplere şi astupare a unor sticle. La conectarea sistemului, se porneşte motorul benzii transportoare. Acesta se va opri când există sticle în situaţia de a fi umplute şi în situaţia de a fi astupate. Se cere ca simultan cu umplerea unei sticle alta deja umplută să fie astupată, acţiune posibilă datorită configuraţiei sistemului. Cilindrul A este responsabil cu umplerea sticlelor, deplasându-se între limitatoarele F1 şi F2. Cilindrul C aduce dopurile din stiva de dopuri, pentru a fi împinse de către cilindrul B pentru astuparea unei sticle. Astuparea sticlei se face de către un dispozitiv de rotire, a cărui mişcare este limitată de un limitator. Sticlele de umplut şi cele de astupat sunt detectate de câte două fotocelule.

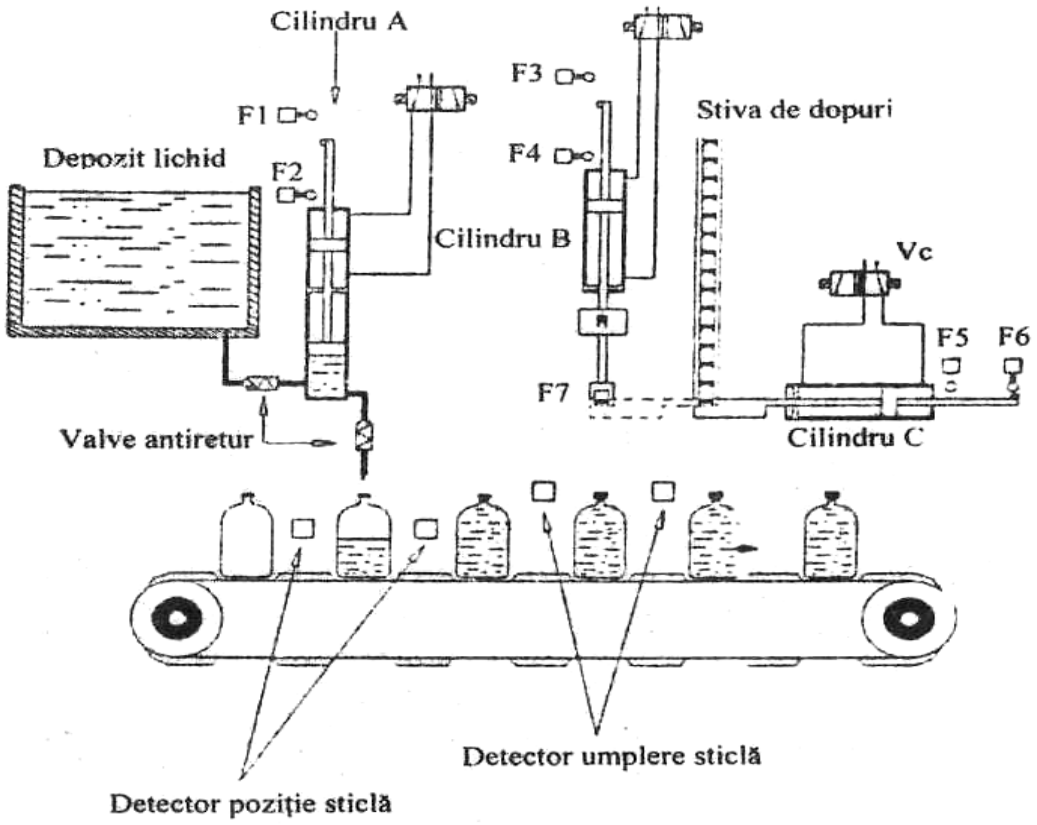


Fig. 5.32 Umplerea şi astuparea automată a sticlelor

Elemente de execuţie:

- 1 cilindru (A) ce reglează dozatorul volumetric
- 1 cilindru de avans (B) cu 3 poziţii
- 1 cilindru (C) ce reprezintă mecanismul de transfer al capacelor
- motorul bezii transportoare
- un mecanism de înşurubare dopuri

Elemente de măsură:

- 6 lmitatoare de cursă
- 1 detector de poziţie
- 1 fotocelulă pentru detecţie sticlă de umplut
- 1 fotocelulă pentru detecţie sticlă plină

2. Soluția de automatizare

Pentru controlul acestei aplicații s-a ales un utomat programabil de tip PEP Smart pentru care s-a dezvoltat un proiect Isagraf ce cuprinde un program principal, numit „Main”.

Dicționarul de variabile globale:

- *Variabile de intrare booleene:*
  - F1 : Limitator sus cilindru A
  - F2 : Limitator jos cilindru A
  - F3 : Limitator sus cilindru B
  - F4 : Limitator jos cilindru B
  - F5 : Limitator stânga cilindru C
  - F6 : Limitator dreapta cilindru C
  - F7 : Detector poziție de preluat dop
  - senzor_rotire : Limitator rotire dispozitiv de înşurubare
  - st_de_reumplut : Senzor detecție sticlă de umplut
  - st_plina : Senzor detecție sticlă plină

- *Variabile de ieşire booleene:*
  - Banda : comandă pornire / oprire bandă
  - A_avans : comandă avans cilindru A
  - A_retragere : comandă retragere cilindru A
  - B_avans : comandă avans cilindru B
  - B_retragere : comandă retragere cilindru B
  - C_avans: comandă avans cilindru C
  - C_retragere : comandă retragere cilindru C
  - Insurubare : comandă de înşurubare a dopului

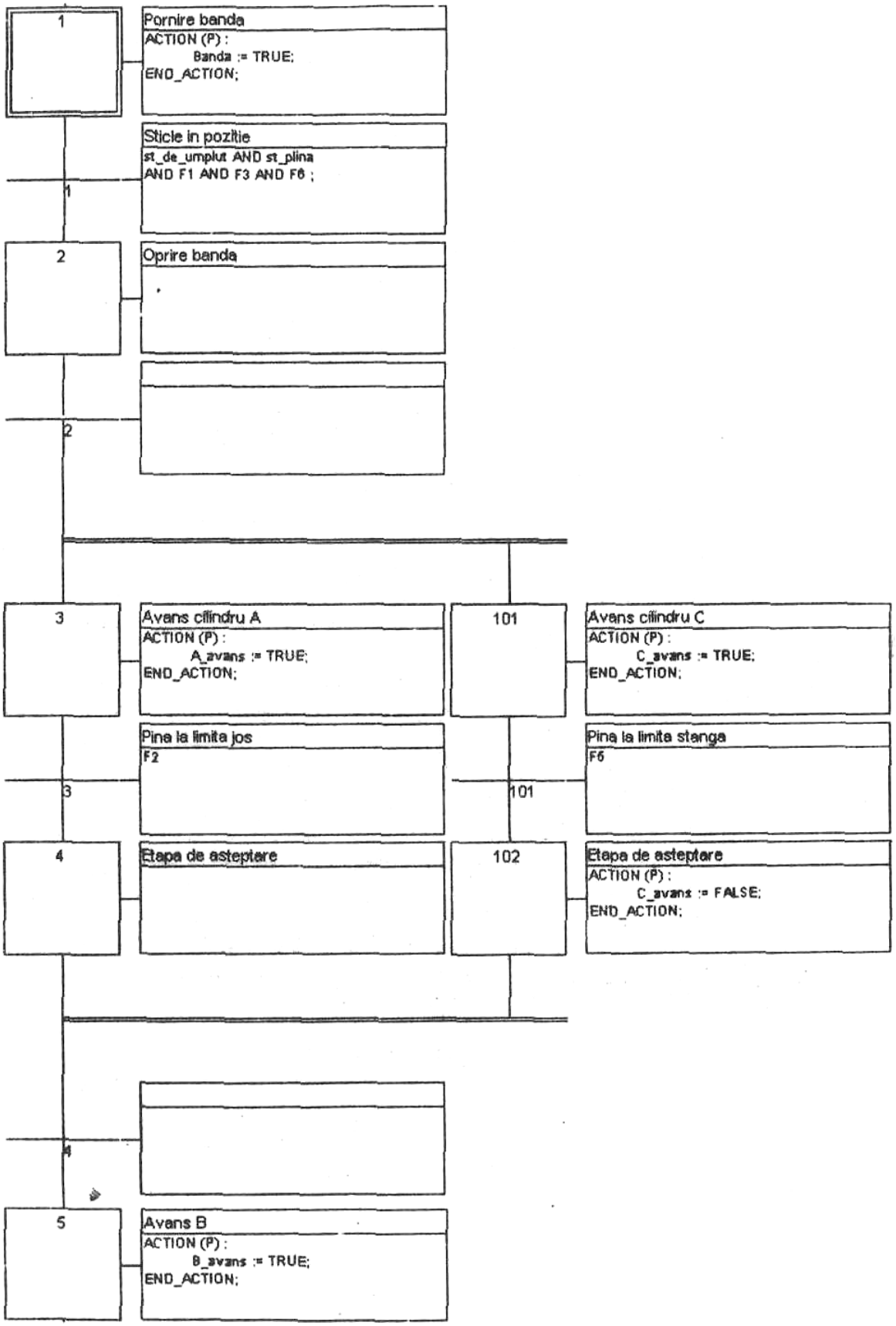
Comentarii:

- În program au fost folosite 2 paralelisme, fiecare având câte 2 secvențe ce se execută simultan
- A doua secvență paralelă implementează condiția ca o dată cu umplerea unei sticle, alta să fie astupată

Propunere:

- Să se modifice programul în situația în care sticlele vin aleator pe banda transportoare

Programul principal este prezentat în figura 5.33



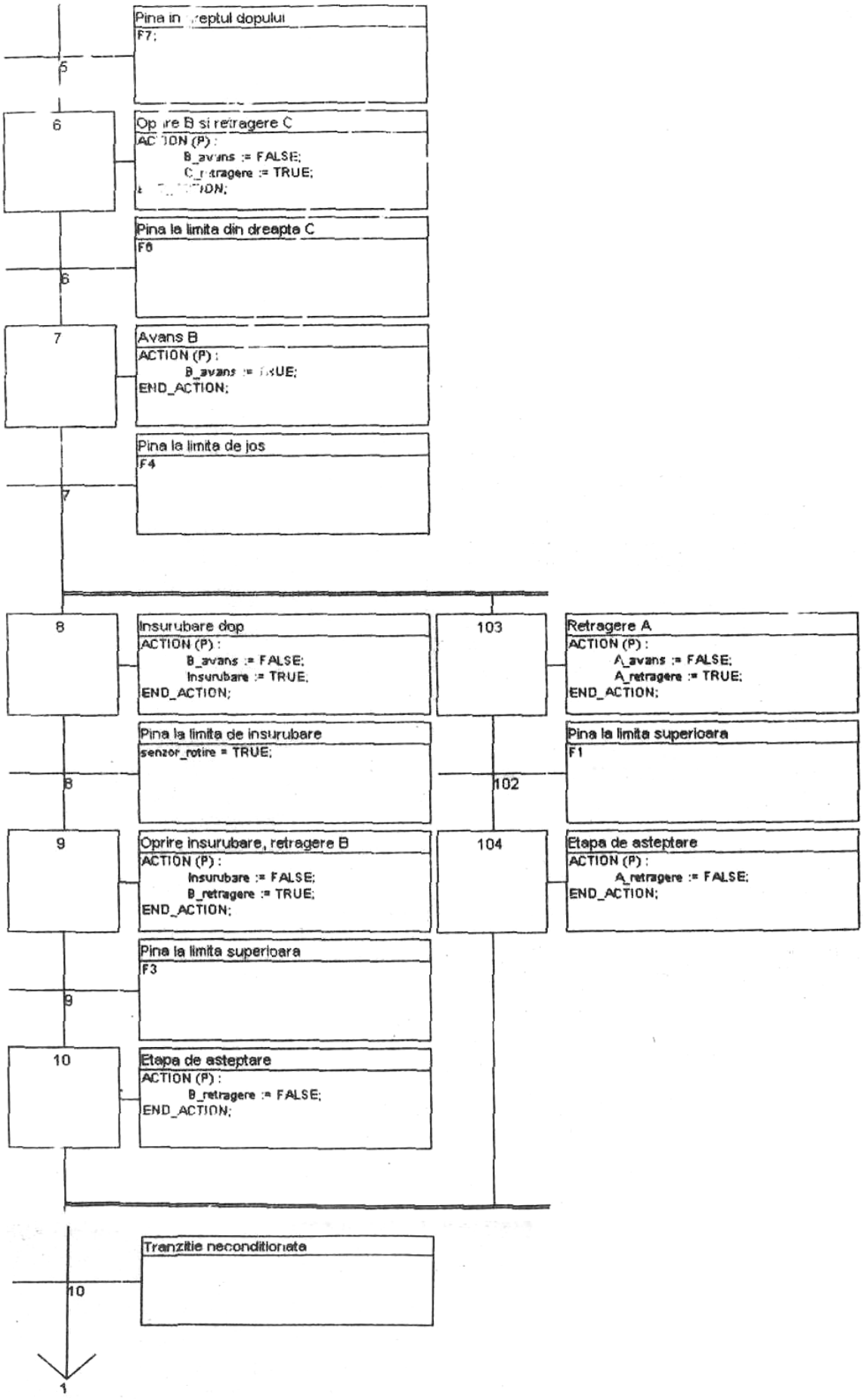


Fig. 5.33 Programul principal

Problema 9: Umplerea automată a unor containere

1. Descrierea procesului:

Aplicația constă în umplerea cu lichid a 3 containere (A, B, C) şi evacuarea lor pe o bandă transportoare. Umplerea containerelor trebuie făcută în următoarea manieră:

- containerul A : 5 secunde cu lichid de tip A
- containerul B : 7 secunde cu lichid de tip A şi 7 secunde cu lichid de tip B
- containerul C : 3 secunde cu lichid de tip C, 5 secunde cu lichid de tip B şi 8 secunde cu lichid de tip A

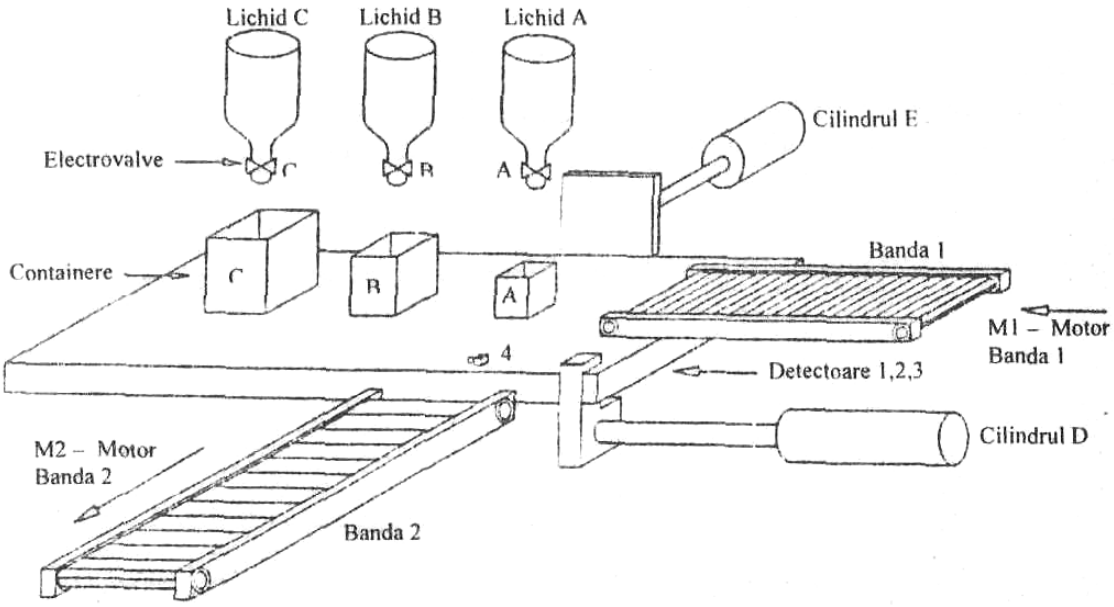


Fig. 5.34 Umplerea automată a unor containere

În cadrul sistemului există o bandă transportoare pe care vin, unul după altul, cele trei containere A, B, C. Primul dintre ele care ajunge la platformă este containerul C apoi B şi ultimul cel de tip A. Cilindrul E este responsabil cu evacuarea recipienților cu ajutorul celei de-a doua benzi transportoare. Inițierea procesului se face prin pornirea benzii transportoare 1 pe care sunt aduse containerele. în momentul în care un container de tip C se găseşte pe platformă, banda 1 va fi oprită iar cilindrul D va avansa o poziție. Când containerul C activează detectorul 2, banda 1 va fi din nou activată iar cilindrul D va fi oprit; banda 1 se va opri din nou când containerul B ajunge la platformă şi în consecință cilindrul D va avansa din nou până când containerul C activează detectorul 3 iar containerul B activează detectorul 2. În acest moment banda 1 este repornită până când containerul A atinge platforma, moment în care banda este oprită. În acel moment cele trei valve vor fi deschise simultan, fiecare fiind menținută deschisă un anumit timp, astfel încât contaienrul A se va umple cu lichid A timp de 5 secunde, containerul B cu lichid de tip B timp de 7 secunde iar containerul C timp de 3 secunde cu lichid de tip C. Când toate aceste temporizări au expirat, valvele vor fi închise, cilindrul E va avansa pentru a evacua containerul A până activează detectorul 4. În acest moment cilindrul E se retrage. După ce a ajuns în poziția de retragere, cilindrul D va fi retras până activează detectoarele 1 şi. 2. Apoi containerele B şi C vor fi umplute cu lichid de tip A, respectiv B, după care urmează evacuarea containerului B, în final containerul C va fi umplut cu lichid de tip A şi va fi evacuat. După evacuare cilindrul D va fi retras şi un nou ciclu poate începe.

Elemente de execuție:

- 3 electrovalve
- 2 cilindri cu dublu efect
- 2 motoare ale benzilor transportoare

Elemente de măsură:

- 4 detectoare de poziție

2. Soluția de automatizare

Pentru controlul acestei aplicații s-a ales un automat programabil de tip PEP Smart pentru care s-a dezvoltat un proiect Isagraf ce cuprinde un program principal.

Dicționarul de variabile globale:

Variabile de intrare booleene:

- Poz_A : detectorul de poziție A
- Poz_B : detectorul de poziție B
- Poz_C : detectorul de poziție C
- Lim_4 : limitatorul de avans al cilindrului E

Variabile de ieşire booleene:

- Banda_l : activare / dezactivare banda l
- Banda_2 : activare / dezactivare banda 2
- D_avans : avans cilindru D
- D_retragere : retragere cilindru D
- E_avans : avans cilindru E
- E_retragere : retragere cilindru E
- Valva_A : comanda de deschidere / închidere valva A
- Valva_B : comanda de deschidere / închidere valva B
- Valva_C : comanda de deschidere / închidere valva C

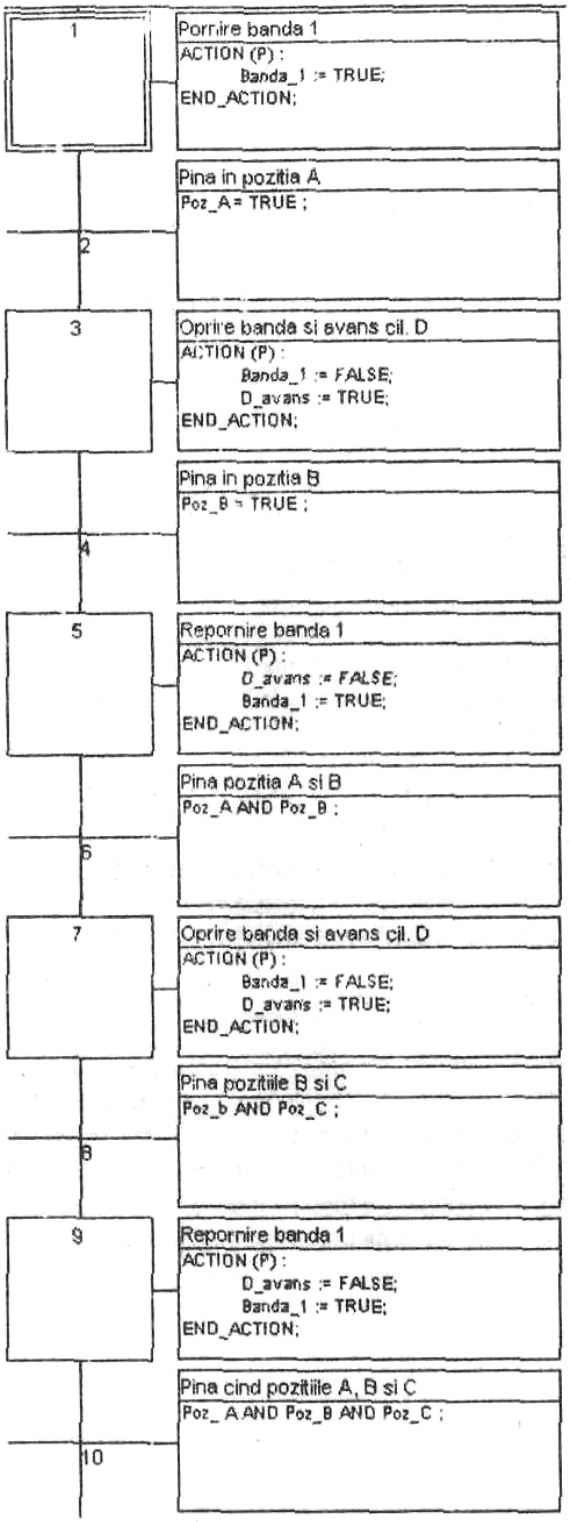
Constante interne de tip INTEGER:

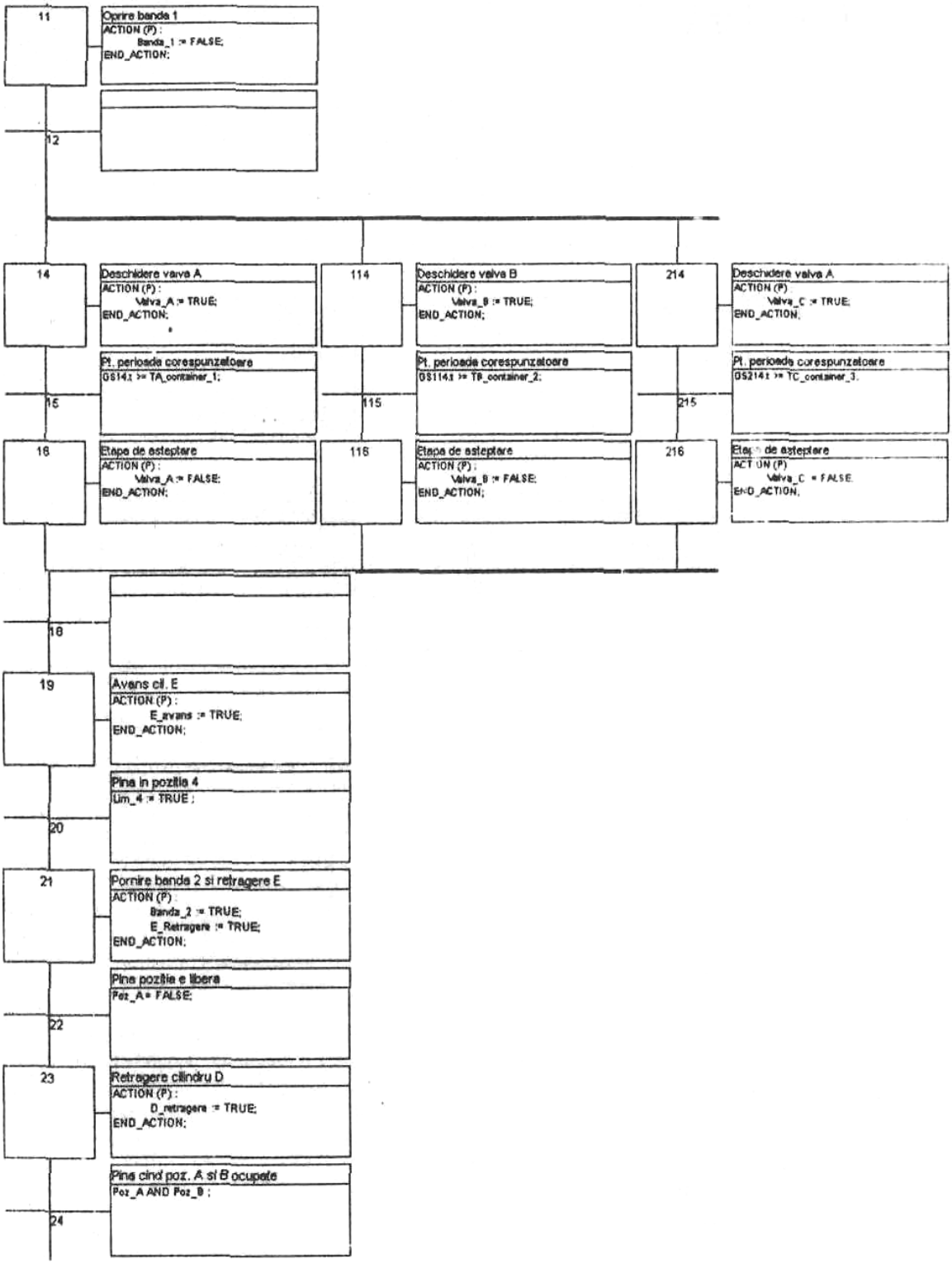
- TA_container_I : perioada de timp pentru umplerea containerului 1 cu lichid de tip A
- TB_container_2 : perioada de timp pentru umplerea containerului 2 cu lichid de tip B
- TC_container_3 : perioada de timp pentru umplerea containerului 3 cu lichid de tip C
- TA_container_2 : perioada de timp pentru umplerea containerului 2 cu lichid de tip A
- TB_container_3 : perioada de timp pentru umplerea containerului 3 cu lichid de tip B
- TA_container_3 : perioada de timp pentru umplerea containerului 3 cu lichid de tip A

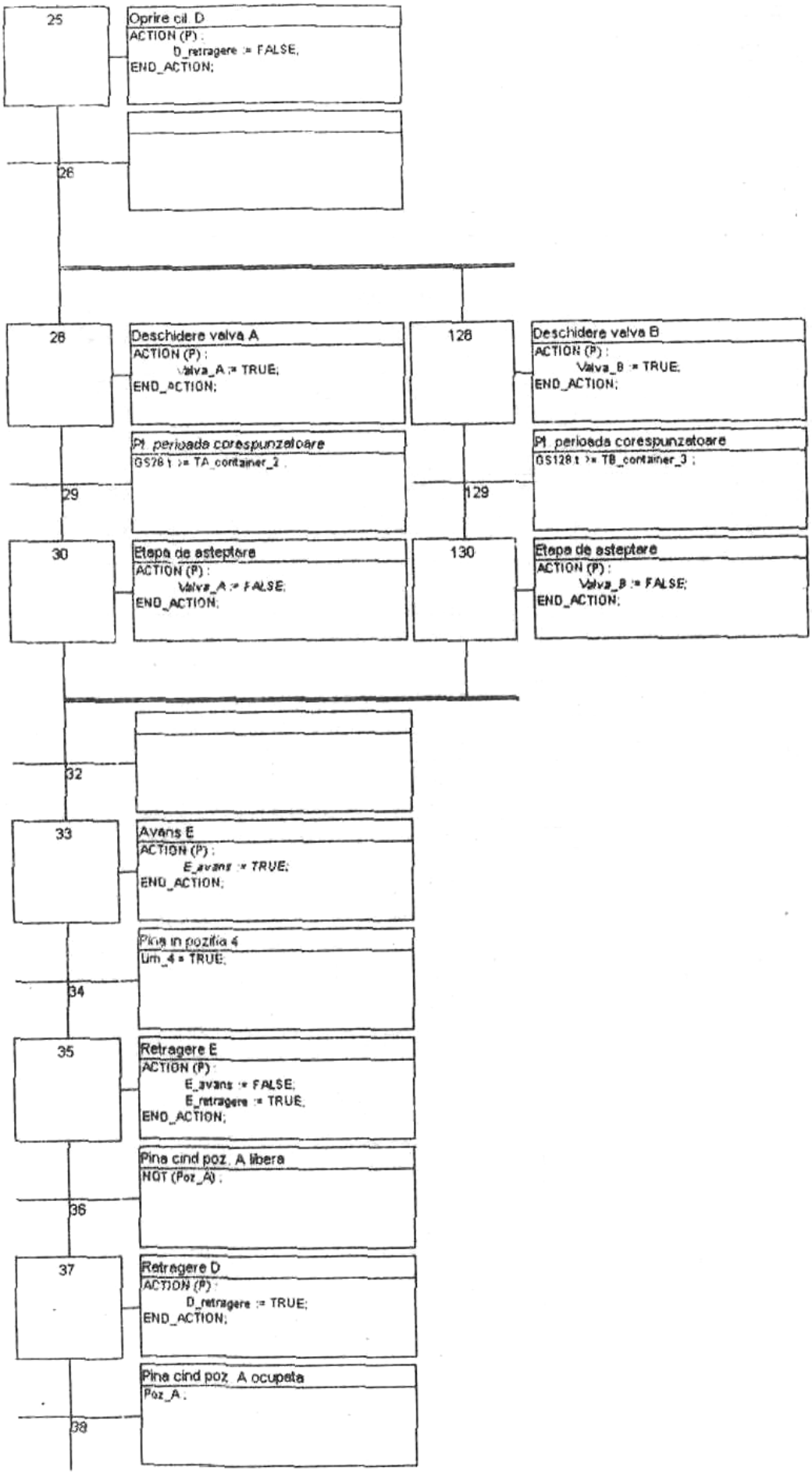
Comentarii:

- În cadrul secvențelor de deschidere a valvelor, temporizările au fost implementate prin tastarea permanentă a parametrului GSxxx.t asociat unei etape. Acest parametru indică timpul de când o etapă este activă. O altă variantă era folosirea unor variabile de tip Timer.
- Umplerea containerelor este făcută în paralel cu ajutorul elementului de paralelism al diagramelor SFC

Programul principal este prezentat în figurile următoare:




11
Oprire banda 1
ACTION (P) :
Banda_1 := FALSE;
END_ACTION;
12
14
Deschidere valva A
ACTION (P) :
Valva_A := TRUE;
END_ACTION;
15
Pt. perioada corespunzatoare
OS14.t >= TA_container_1;
16
Etapa de asteptare
ACTION (P) :
Valva_A := FALSE;
END_ACTION;
114
Deschidere valva B
ACTION (P) :
Valva_B := TRUE;
END_ACTION;
115
Pt. perioada corespunzatoare
OS114.t >= TB_container_2;
116
Etapa de asteptare
ACTION (P) :
Valva_B := FALSE;
END_ACTION;
214
Deschidere valva A
ACTION (P)
Valva_C := TRUE;
END_ACTION;
215
Pt. perioada corespunzatoare
OS214.t >= TC_container_3;
216
Etapa de asteptare
ACTION (P)
Valva_C := FALSE;
END_ACTION;
18
19
Avans cil. E
ACTION (P) :
E_avans := TRUE;
END_ACTION;
Pina in pozitia 4
Lim_4 := TRUE ;
20
21
Pornire banda 2 si retragere E
ACTION (P) :
Banda_2 := TRUE;
E_Retragere := TRUE;
END_ACTION;
Pina pozitia e libera
Poz_A= FALSE;
22
23
Retragere cilindru D
ACTION (P) :
D_retragere := TRUE;
END_ACTION;
Pina cind poz. A si B ocupate
Poz_A AND Poz_B ;
24


25
Oprire cil. D
ACTION (P) :
D_retragere := FALSE;
END_ACTION;
26
26
Deschidere valva A
ACTION (P) :
Valva_A := TRUE;
END_ACTION;
Pt. perioada corespunzatoare
GS26.t >= TA_container_2 ;
29
30
Etapa de asteptare
ACTION (P) :
Valva_A := FALSE;
END_ACTION;
128
Deschidere valva B
ACTION (P) :
Valva_B := TRUE;
END_ACTION;
Pt. perioada corespunzatoare
GS128.t >= TB_container_3 ;
129
130
Etapa de asteptare
ACTION (P) :
Valva_B := FALSE;
END_ACTION;
32
33
Avans E
ACTION (P) :
E_avans := TRUE;
END_ACTION;
Pina in pozitia 4
Um_4 = TRUE;
34
35
Retragere E
ACTION (P) :
E_avans := FALSE;
E_retragere := TRUE;
END_ACTION;
Pina cind poz. A libera
NOT (Poz_A) ;
36
37
Retragere D
ACTION (P) :
D_retragere := TRUE;
END_ACTION;
Pina cind poz. A ocupata
Poz_A ;
38

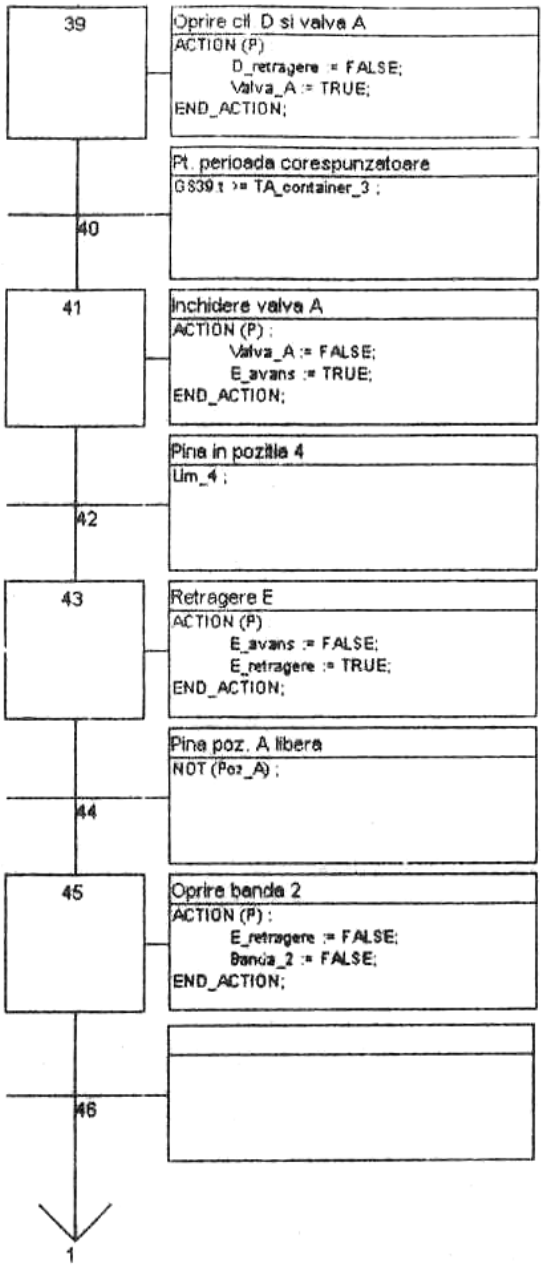


Fig. 5.35 Programul principal

Problema l0: Proiectarea unui regulator PID pentru reglarea unei temperaturi

1. Descrierea procesului:

Problema constă în menținerea unei anumite temperaturi într-o instalație de granulare prin reglarea debitului de abur, dar, ca orice regulator, se poate acorda pentru o mare varietate de procese. Regulatorul PID trebuie să funcționeze în două regimuri de lucru, automat şi manual, cu posibilitatea de a fi conectat în diferite scheme de reglare.

Temperatura este măsurată cu un traductor de temperatură având semnal de ieşire 4-20mA, măsura se filtrează iar valoarea reală se transmite regulatorului în procente. Referința se citeşte de la un panou operator care comunică cu automatul prin interfața serială 232.

Regulatorul face parte dintr-o schemă în care se intenționează menținerea constantă a temperaturii unui agent termic (în cazul acesta abur de joasă presiune 4-6 bar) prin comanda debitului de intrare al gazului metan.

Bucla de reglare în această situație este compusă din:

- termocuplu (pentru măsurarea temperaturii) care generează milivolți după o curbă care nu este liniară (ca alternativă la această soluție se poate utiliza orice alt traductor al cărui semnal este în curent unificat). Soluția cu automate programabile este destul de flexibilă şi din faptul că ele au module de intrare pentru toate tipurile de termocuple, eliminând astfel din buclă convertorul milivolți - curent unificat, dar alegerea între cele două metode se face strict economic
- convertor 4-20 mA (pentru conversia în miliamperi a semnalului de la termocuplu (mV) şi liniarizarea curbei valorilor)
- automat programabil SmartPLC cu module de intrări analogice, respectiv în termocuple şi modul de ieşiri analogice care la rândul lor sunt nişte convertoare analog-numerice şi invers (pentru realizarea algoritmului de reglare)
- electroventil comandat în curent de 420mA, reprezentând elementul de execuție (se mai poate folosi şi un ventil pneumatic împreună cu un convertor electropneumatic dacă este necesar)

În fig. 4.12 este prezentată schematic bucla de reglare pentru varianta cu convertor, cealaltă fiind mai simplă (ieşirea termocuplului intrând direct în automat):

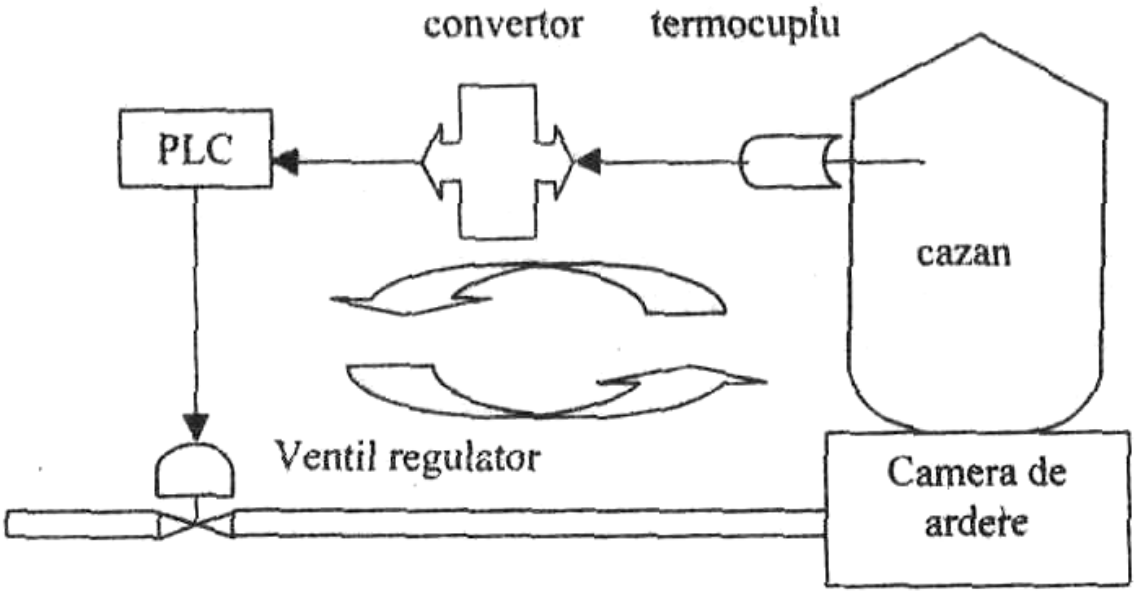


Fig. 5.36 Structura buclei de reglare folosind convertor tensiune - curent

Regulatorul se acordează prin:

- banda de proporționalitate [în procente]
- timp de integrare [în secunde]
- timp de derivare [în secunde]

Valorile de acordare se citesc şi ele de la interfața operator odată cu referința.

În regim manual regulatorul transmite valoarea absolută a poziției elementului de execuție (comanda directă), venită tot de la consola operator, de unde se citeşte şi regimul de lucru.

Trecerea din regim manual în regim automat şi invers trebuie să se facă fără variații bruşte de comandă, pentru a se evita intrarea sistemului în regimuri de lucru improprii. La comutarea regimurilor de lucru, pe cât posibil comanda va trebui să rămână aceeaşi.

Elementul de execuție primeşte la intrare valoarea absolută a poziției. Menținerea acestei valori este o problemă a elementului de execuție. Dacă e necesar se poate insera un regulator de poziție între regulatorul de temperatură şi elementul de execuție.

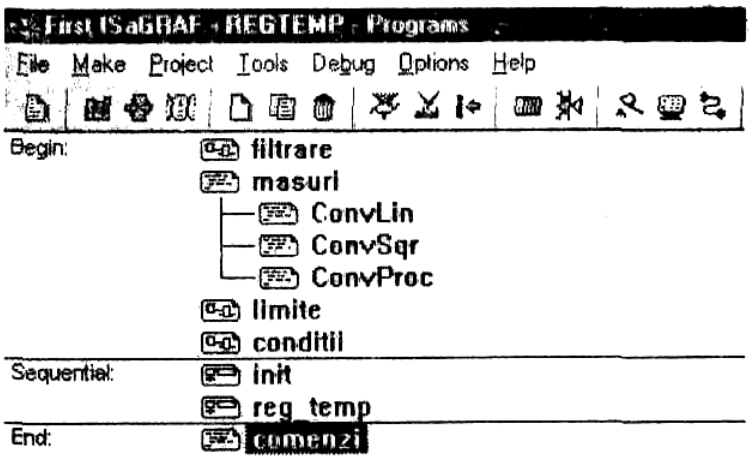


Fig. 5.37 Structura proiectului Isagraf pentru implementarea regulatorului

2. Soluția de automatizare

Pentru implementarea acestei aplicații s-a ales un automat programabil de tip PEP Smart pentru care s-a dezvoltat un proiect Isagraf, a cărui structură ierarhică poate fi vizualizată în figura 5.37 şi ale cărui secțiuni standard sunt descrise în continuare.

Inițializările de constante, măsurile, filtrările, adaptările, evaluarea existenței condițiilor de funcționare, interfața cu utilizatorul, sunt realizate în afara regulatorului de către alte taskuri care se ocupă doar de aceste lucruri. Comanda se adaptează şi se transmite în afara regulatorului.

Numele de variabile s-au ales pentru varianta cu identificatori proprii. Pentru o versiune generică, impersonală, se lucrează cu array-uri (vectori de variabile), iar accesul se face prin indexul la variabila respectivă.

În cadrul secțiunii BEGIN, care se execută la începutul fiecărui ciclu de automat, au loc următoarele acțiuni:

- Măsuri, filtrări, testare de încadrare în limite, evaluare alarme, avarii, contorizări
- Adaptări, pregătire variabile de lucru
- Citirea variabilelor de control de la interfața operator

În cadrul secțiunii SEQUENTIAL, adică acolo unde este implementată logica de funcționare a programului, au loc, în principal, următoarele acțiuni:

- secvența de inițializare, în care sunt inițializate anumite constante şi este pornit timer-ul pentru perioada de eşantionare
- algoritmul de reglare, a cărui descriere detaliata este prezentată în subcapitolul următor

Nu întâmplător secțiunea secvențială are o singură stare pentru fiecare regim de lucru. Această necesitate a apărut datorită faptului că automatul (care poate lucra multitasking) nu trece la o altă activitate decăt după terminarea unui ciclu maşină. Într-un ciclu maşină se execută secțiunea de BEGIN, cea de END şi un anumit număr de stări din secțiunea SEQUENTIAL, cât permite durata unui ciclu maşină. Dacă pentru execuția acțiunilor asociate unei stări, procesorului i-ar lua un timp mai îndelungat decât perioada unui ciclu predefinit, ciclul va fi prelungit până când toate acțiunile asociate stării sunt executate.

Cu această metodă ciclul de reglare devine constant şi cât mai mic posibil. Durata ciclului poate fi fixată şi altfel (software), dar oricum, perioada va avea o valoare mai mare, pierzându-se astfel din timpul de răspuns al regulatorului. Mai mult, in cazul în care ar exista mai multe etape, apare posibilitatea ca comanda ce se dă elementului de execuție să nu fie consecința ultimei citiri, ceea ce face să se piardă chiar noțiunea de cauzalitate.

Secțiunea END, care se execută la sfârşitul fiecărui ciclu automat, după secțiunea secvențială, realizează în principal următoarele acțiuni:

- Adaptări de valori
- Actualizare ieşiri: concret, în cazul algoritmului de reglare ce respectă condițiile de mai sus, ieşirea este comanda către elementul de execuție calculată conform algoritmului la ultimul ciclu maşină şi ca urmare a ultimei valori a intrării
- Transmiterea către interfața utilizator a valorilor calculate, pentru vizualizare

Diagrama logică de funcționare a regulatorului (Sequential Function Chart), în care pot fi identificate etapele în care se poate găsi automatul şi tranzițiile pe care le poate efectua, sunt ilustrate în figura 5. 38.

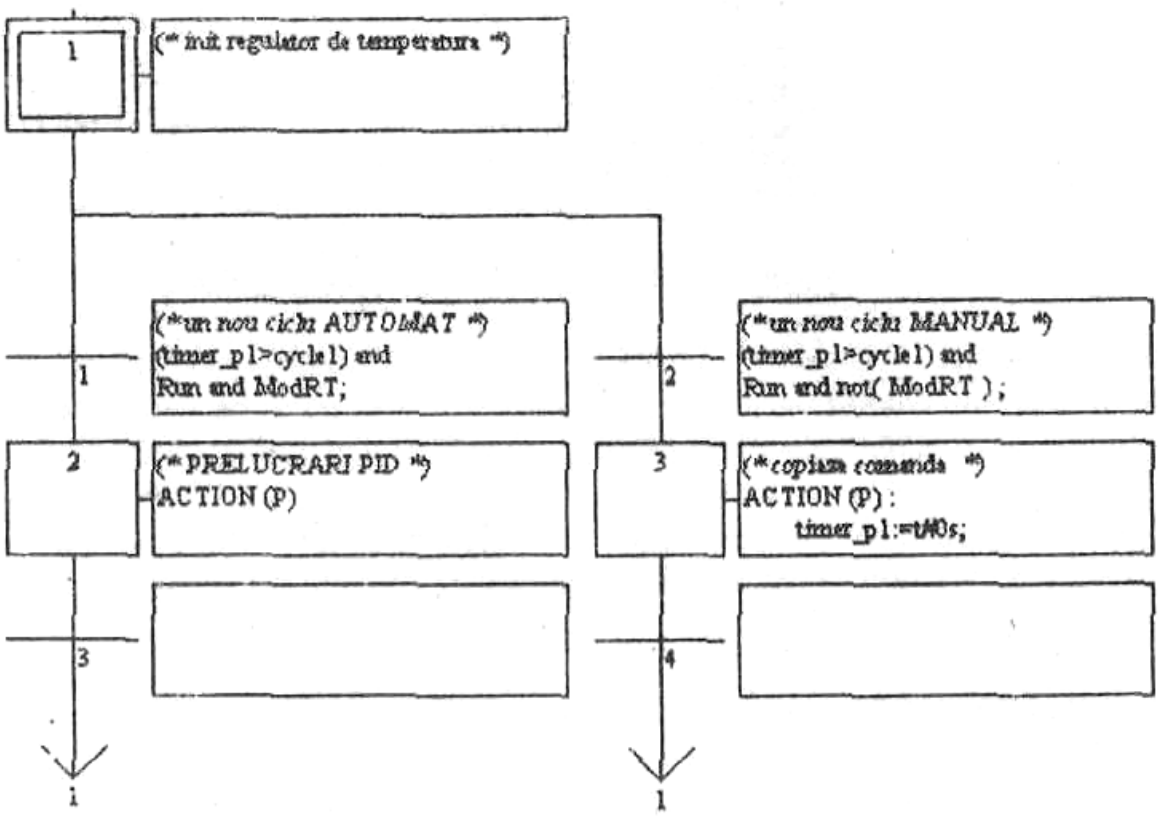


Fig. 5.38 Diagrama logică de funcționare a regulatorului

Se observă că diagrama conține 3 etape şi pot fi efectuate 4 tranziții între aceste etape.

Etape:

- Etapa 1 : inițializare regulator
- Etapa 2 : se execută un pas de reglare în regim automat
- Etapa 3 : se execută un pas de reglare în regim manual

Tranziții:

- Tranziția 1 – S1→S2 : se execută dacă sunt condiții de trecere în starea de calcul PID (regimul automat)
- Tranziția 2 – S1→S3 : se execută dacă sunt condiții de trecere în starea de comandă manuală. (regimul manual)
- Tranziția 3 – S2→S1 : salt necondiționat în etapa inițială
- Tranziția 4 – S3→S1 : salt necondiționat în etapa inițială

Variabilele declarate în cadrul proiectului sunt prezentate în tabelul 5.5:

Tabel 5.5

Variabilă	**Tip**	**Descriere**
Run	Logic	Există condiții generate de funcționare pentru regulatorul PID, evaluate în exterior
	0	OK, adică sunt măsuri bune, elementele de execuție funcționează, resurse energetice OK, etc
	1	not OK, nu se face reglare cu regulatorul PID
ModRT	Logic	Regimul de lucru al reglatorului, evaluat în exterior
	0	regim AUTOMAT (PID)
	1	regim MANUAL (comandă directă)
timer_ pi	Timer	Timer de ciclu de eşantionare
cycle1	Timer	Perioada de eşantionare, constantă [ms]
banda1	Real	Bandă de proporționalitate, inversul amplificării, în %
Td1	Real	Constantă de timp pentru efectul de derivare, valoare de acord
Tint1	Real	Constantă de timp pentru efectul de integrare, valoare de acord
kp1	Real	Amplificarea, valoare calculată, dacă banda este valoarea de acord
deriv1	Real	Componenta derivativă, variabilă de calcul
w	Real	Calcul parțial pentru incrementul de comandă, variabilă de lucru
T_pv	Real	Valoarea curentă a temperaturii reglate, în %
T_pvl	Real	Valoarea curentă, la pasul anterior, a temperaturii reglate, în %
T_pv2	Real	Valoarea curentă, la pasul anteanterior, a temperaturii reglate, în %
T_sp	Real	Valoarea impusă pentru temperatură, în %
Err1	Real	Eroarea (t_sp - t_pv) la pasul anterior
Comandă ventil	Real	Comandă curentă, poziția absolută a elementului de execuție, în %
xo1	Real	Comandă manuală, poziția absolută a elementului de execuție, în %

În etapa 1 (Init) se actualizează (dacă e nevoie) valorile variabilelor de lucru (dacă regulatorul funcționează în alte regimuri de lucru, spre exemplu în cascadă, sau în regim de selecție).

Acțiunile asociate **etapei 2** (Regim automat) sunt descrise în tabelul 5.6:

Tabel 5. 6

inițializează ciclul	timer_pl := t#0s
calcul amplificare	kpl := 100.0/banda1
calcul componentă derivativă	deriv1:=Td1*1000.0*(T_pv-2.0*T_pvl+T_pv2)/real(cycle1)
salvare valori pv anterioare	T_pv2:=T_pv1; T_pv1:=T_pv;
calcul noua comandă (pt timpi, se lucrează în [ms]=[s]/1000)	w:=(T_sp-T_pv)+(T_sp-T_pv)*real(cycle1)/(Tint1*1000.0)-err1-deriv1 Comanda ventil:=Comanda ventil+Kp1*w;

limitări la 0% şi 100%	if(Comanda ventil>100.0) then Comanda ventil:=l00.0;
	end_if;
	if(Comanda ventil<0.0) then Comanda ventil:=0.0;
	end_if;

salvare eroare	Err1:=T_sp-T_pv;
pregătire trecere în regim manual	xo1:=Comanda ventil;

În etapa 3 (Regim manual) este inițializat ciclul şi pregăteşte trecerea în regim automat:

- **Tranziția 1** (intrare în regulator automat S1→S2) are loc dacă s-a epuizat timpul pentru perioada de eşantionare şi sunt condiții de reglare în regim automat:
  timer_pl>cycle1 şi Run şi ModRT
- **Tranziția 1** (intrare în regulator manual S1→S3) are loc dacă s-a epuizat timpul pentru perioada de eşantionare şi sunt condiții de reglare în regim manual:
  timcr_pl>cycle1 şi Run şi notModRT

E important să se efectueze toate calculele necesare într-o singură stare (un sigur ciclu IsaGRAF) şi, pentru uniformitate, e de dorit ca pe oricare ramură (automat sau manual), durata unui ciclu de reglare sa fie aceeaşi, cât mai scurtă, adică două cicluri IsaGRAF (init+automal sau init +manual), pentru ca modul de funcționare a IsaGRAF-ului (la un ciclu maşină execută secțiunea de BEGIN, din secțiunea SEQUENTIAL câte stări îi permite ciclul maşină şi apoi secțiunea END) să nu influențeze (mărească) perioada de eşantionare.

Un ciclu de eşantionare, pentru valorile cu care se lucrează la reglare, se calculează ținând cont că un ciclu activ înseamnă stare - calcul - comandă + stare init (2 isa) iar un ciclu pasiv înseamnă stare init (1 isa) unde se aşteaptă epuizarea timpului ales pentru ciclul de reglare. Daca ciclul isa se fixează, atunci se poate alege ca perioada de eşantionare pentru reglare un multiplu de ciclu isa mai mare sau egal cu 2 şi se introduc în secvență etape lipsite de acțiuni care asigură implicit realizarea perioadei. Dacă sunt necesare regulatoare suplimentare (de exemplu de poziție) va trebuie asigurată o sincronizare.

Oricum, secvența în două stări pe ramură este cea mai convenabilă în cazul în care se adoptă această soluție. Soluția corectă de realizare a unui regulator este aceea în care funcția de reglare este descrisă ca o funcție C ce se va apela chiar în secțiunea de BEGIN şi va fi executată cu siguranță câte un pas incremental la fiecare ciclu maşină.

Diagrama SFC a programului principal de reglare este prezentată în figura 5, 39.

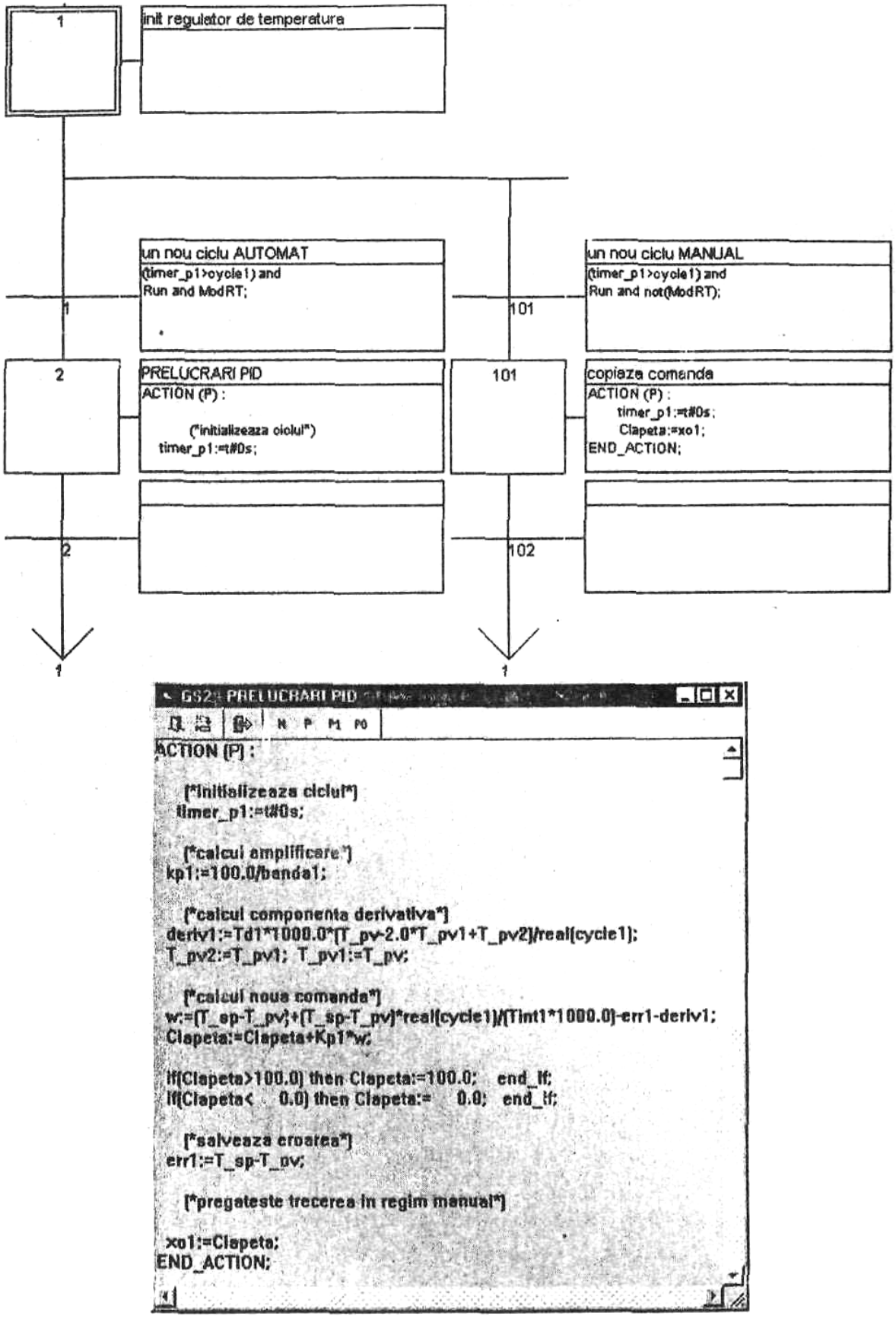


Fig. 5.39 Diagrama SFC a programului „Reg_temp"