SUPORT DE CURS

INDRUMAR DE LABORATOR – Partea I

5

5.ELEMENTE DE PROIECTARE A SISTEMELOR DE AUTOMATIZARE

5.1. Introducere

Sistemele de automatizare proiectate pot fi implementate în două moduri:

1. Prin logica cablată, caz în care:
    a. Funcţia realizată depinde de conexiunile dintre module, deci de cablaj şi
    b. Orice modificare în funcţia de conducere a sistemului necesită modificări hardware.
2. Prin logica programată, ceea ce presupune:
    a. Existenţa unui echipament universal pe care poate rula orice aplicaţie;
    b. Funcţia sistemului de automatizare este realizată de un program aflat într-o memorie. Modificare funcţiei sistemului, în acest caz, nu se face prin modificări hardware ci software, deci prin încărcarea în memorie a unui alt program.

Deşi astăzi, logica programată a câştigat teren în faţa celei cablate, datorită avantajelor şi dezavantajelor ambelor moduri, alegerea între cele două nu este întotdeauna facilă. Dacă sistemul de automatizare are de gestionat un număr mare de parametri şi/ sau algoritmul de conducere este complex atunci se optează pentru implementarea prin logică programată. Dacă în schimb, cerinţele de viteză sunt primordiale, se optează pentru implementarea prin logică cablată. În cazul sistemelor care au cerinţe şi de complexitate şi de viteză, soluţia de implemetare va fi una mixtă. Proiectarea unui sistem de automatizare va începe în concluzie cu faza de *definire a problemei* în care se face şi partajarea sistemului în logică cablată şi în logica programată. În această fază se determină numărul de intrări şi ieşiri din sistem, viteza lor de variaţie, cantitatea şi viteza de prelucrare a datelor, tipul de erori şi modul de tratare al acestora. Pentru partajarea pe tipuri de logică se pot întâlni următoarele situaţii:

- Există un număr mic de semnale de intrare şi ieşire iar logica de prelucrare a lor este simplă astfel încât realizarea unui sistem în logică cablată este mai economică decât implemetarea sistemului în logică programată chiar în varianta minimală;

- Cerinţele de performanţă ale sistemului nu sunt deosebite, astfel încât se poate implementa în logică programată;

- Logica de prelucrare a datelor este complexă, numărul de intrări şi ieşiri este mare deci se poate utiliza logica programată. Există însă şi cerinţe de performanţă ce nu pot fi satisfăcute decât prin logică cablată. Se impune în acest caz o prelucrare parţială a semnalelor în circuite specializate. Se atribuie părţii din sistem realizată în logică

programată cât mai multe funcţii pentru astfel încât logica cablată să fie cât mai simplă.

5.2. Proiectarea logicii cablate

Proiectarea logicii cablate se face pe baza algoritmului din Figura 1.1.

Semnificaţia blocurilor este următoarea:

A. Sistemul se împarte în blocuri folosind criterii funcţionale sau poziţionale. Se urmăreşte pe de o parte ca blocurile proiectate să fie de complexitate cât mai redusă pentru a putea fi uşor de realizat şi testat. Pe de altă parte numărul blocurilor trebuie să fie cât mai mic pentru a nu apărea probleme de interconectare.
B. Se exprimă funcţionarea blocurilor prin funcţii logice.
C. Se implementează funcţiile logice cu circuite logice pe circuite imprimate realizate special sau prin wraping pe plăci universale. Înainte de implantare componentele vor fi testate. Se realizează apoi conexiunile între plăci.
D. Testarea blocurilor componente se face separat, în condiţii cât mai apropiate de cele reale, simulându-se blocurile încă nerealizate.
E. În această etapă pot să apară conexiuni greşite datorate fazei de sinteză. Modificările făcute în această etapă pot afecta funcţionarea întregului sistem. Nu este exclusă reproiectarea întregului sistem.
F. Proiectarea produsului final poate dura foarte mult mai ales dacă prototipul a fost implementat pe plăci universale iar produsul final trebuie realizat pe circuite imprimate. De multe ori ciclul trebuie parcurs de mai multe ori până la realizarea unui produs acceptabil.

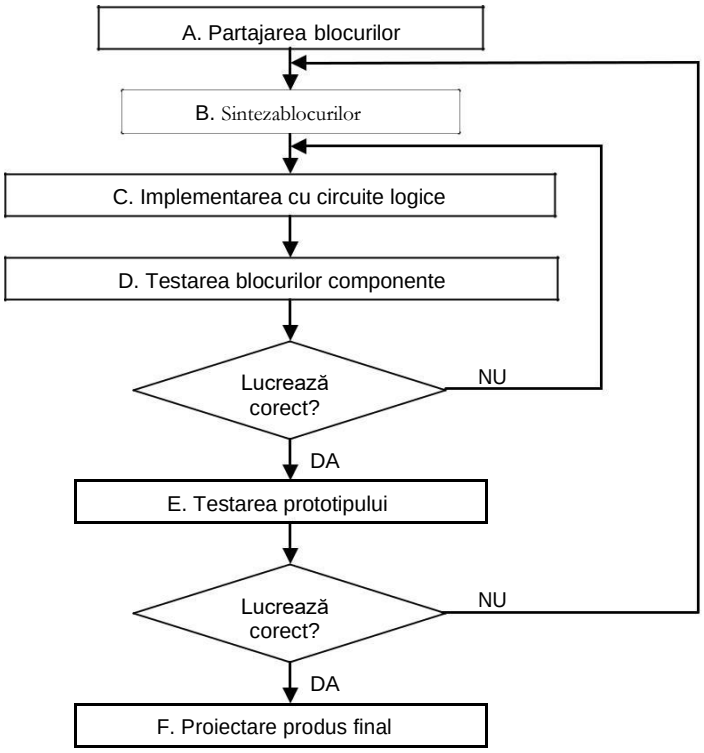


Figura 1.1. Algoritmul de proiectare a logicii cablate.

5.3.Proiectarea logicii programate

Proiectarea logicii programate se face pe baza algoritmului din Figura 1.2.

Semnificaţia blocurilor componente este următoarea:

A. Alegerea configuraţiei microcalculatorului pe baza analizei de sistem. Se determină numărul de intrări, numărul de ieşiri , numărul de dispozitive de

transmitere a informaţiei, capacitatea memoriei, tipul şi numărul de interfeţe cu procesul. Tot în această etapă se face o analiză preliminară a performanţelor sistemului şi a soluţiilor care ar duce la realizarea acestora.

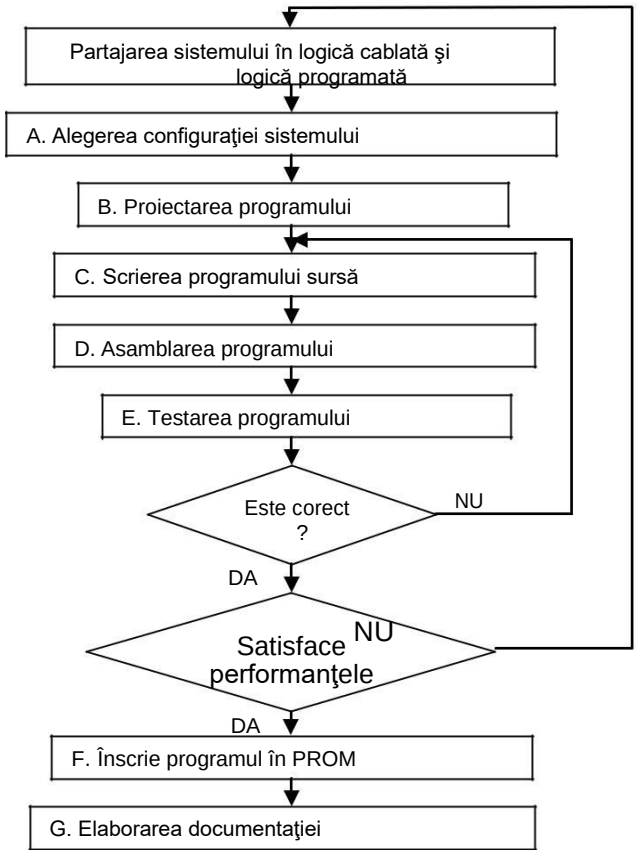


Figura 1.2. Algoritmul de proiectare a logicii programate.

B. Schiţarea operaţiilor ce urmează a fi executate de program.

C. Scrierea programului în limbaj de asamblare sau într-un limbaj de nivel superior.

D. Translatarea în limbaj cod maşină automat în mediul de dezvoltare.

E. Testarea programului urmăreşte verificarea faptului că programul răspunde corect la cerinţele impuse. Fazele de scriere, translatare şi testare se repetă de mai multe ori până când codul maşină funcţionează corect. Se verifică apoi dacă programul satisface performanţele impuse sistemului prin tema de proiectare. Se verifică mai ales viteza de

răspuns. Dacă sistemul nu răspunde se încearcă o optimizare a acestuia. Dacă nici după optimizare sistemul nu răspunde cerinţelor se reia analiza sistemului pentru a vedea ce blocuri pot fi realizate în logica cablată.

F. Dacă verificarea sistemului este încheiată cu succes, programul este înscris în PROM.

G. Se elaborează documentaţia.

Clasificarea automatelor programabile după principiul constructiv

Funcţie de principiul constructiv al automatelelor programabile, acestea se clasifică în:

- *Automate programabile algoritmice* şi
- *Automate programabile vectoriale*.

Automatele programabile algoritmice implementează cu ajutorul memoriilor de tip ROM maşini algoritmice de stare sau se realizează ca structuri microprogramate. La cele din urmă, evoluţia în timp este determinată de o secvenţă coerentă de microinstrucţiuni aflate în memoria internă. Structura lor este asemănătoare cu cea a unităţilor de control ale procesoarelor. Acestea se construiesc, ca sisteme înglobate[1], de către firme ce realizează sisteme de serie mare. Programarea acestor automate este destul de greoaie şi este făcută de personal cu pregătire superioară.

Automatele programabile vectoriale sunt microcalculatoare special concepute pentru tratarea prin program a problemelor de logică combinaţională şi secvenţială. Aceste automate sunt foarte flexibile deoarece simulează structurile logice de comandă printr-o configuraţie elastică, programabilă. Pentru cele mai multe din automatele programabile vectoriale există limbaje de programare care permit programarea similar proiectării logicii cablate sau imprimate.

Clasificarea automatelor programabile funcţie de numărul procesoare din structură

Funcţie de numărul de procesoare, automatele programabile se clasifică în:

- *Automate programabile cu un singur procesor;*
- *Automate programabile multiprocesor.*

Automatele programabile cu un singur procesor folosesc un tampon de memorie, numit *imagine de proces*. Înainte de intrarea în ciclul unui program, se încarcă în memoria imaginii de proces valoarea semnalelor fizice de intrare. Pe parcursul unui ciclu, valorile intrărilor sau ieşirilor folosite în program sunt cele din memoria imaginii de proces, chiar dacă pe parcursul ciclului unele intrări se pot schimba. Imaginea de proces este actualizată cu comenzi de setare sau resetare a ieşirilor. La terminarea ciclului, ieşirile fizice sunt actualizate corespunzător valorilor din imaginea de proces. Memoria cu imaginea de proces se actualizează şi în cazul în care, în program, se fac salturi înapoi. Dacă proiectul implementat în automatul programabil conţine mai multe blocuri distincte, actualizarea memoriei cu imaginea de proces se actualizează la începutul fiecărui bloc.

Avantajele folosirii imaginii de proces sunt:

[1] Embedded system

- Execuţia rapidă a programului;
- Depistarea erorilor de programare prin rutine de tratare a acestora lansate la sfârşitul ciclului program.

Automatele programabile multiprocesor nu utilizează memorie pentru imaginea de proces. Starea intrărilor fizice este citită imediat cum se face referire la acestea. Semnalele de ieşire sunt, de asemenea, actualizate imediat, fiind comutate înainte de terminarea ciclului.

Avantajele metodei de acces direct la intrări/ ieşiri constau în faptul că:

- Este posibilă funcţionarea în paralel a mai multor module procesor;
- Se asigură utilizarea stării curente, practic instantanee, a unei intrări;
- În ciclul programului nu se mai consumă timp cu actualizarea imaginii de proces;
- Se poate utiliza o comutare rapidă a ieşirilor pentru comanda unor periferice electronice (de exemplu: motoare pas cu pas, numărătoare electronice, afişaje electronice, multiplexoare electronice).

Clasificarea automatelor programabile după dimensiunea magistralei de date

După dimensiunea magistralei de date automatele programabile se clasifică astfel:

1. Automate programabile cu prelucrare la nivel de bit, la care dimensiunea magistralei de date este de 1 bit, astfel încât operanzii care se procesează au şi ei dimensiunea de 1 bit.

2. Automate programabile cu prelucrare la nivel de cuvânt de *n* biţi, dimensiunea magistralei şi a operanzilor fiind egală cu lungimea acestui cuvânt, $n \geq 8$.

3. Automate programabile mixte, prevăzute cu două unităţi de calcul aritmetic şi logic, una pentru procesare pe 1 bit şi alta pentru cuvinte de *n* biţi.

5.4 Schema bloc a automatului programabil

5.4.1.Schema bloc a automatului programabil cu prelucrare la nivel de bit (APB)

Automatele programabile cu prelucrarea la nivel de bit sau automatele programabile pe bit, sunt destinate conducerii proceselor de complexitate medie. Având o arhitectură internă simplificată şi un set de instrucţiuni redus, un automatele programabile pe bit realizează prelucrări simple de date, în principal logice, fiind însă capabil să controleze un număr mare de intrări şi ieşiri de un bit asociate procesului controlat, într-o siguranţă funcţională ridicată.

În Figura 3.1. se prezintă schema bloc a unui automat programabil generic cu prelucrare la nivel de bit.

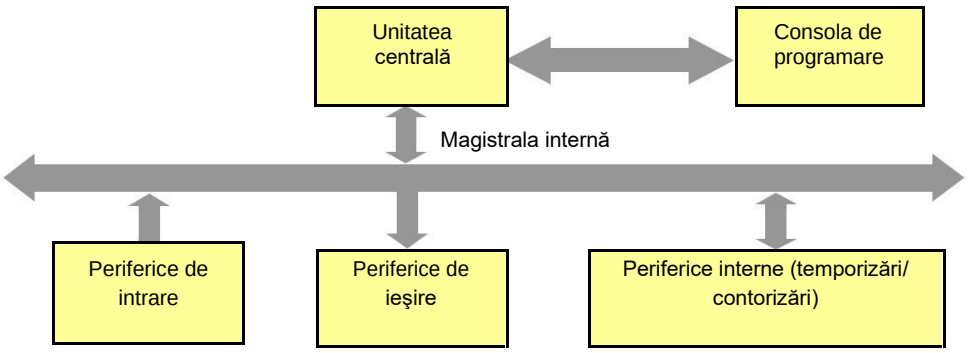


Figura 3.1. Schema bloc a unui automat programabil pe bit.

Blocurile componente ale APB sunt:

- *Unitatea centrală*, este "creierul" APB, ce coordonează activitatea din întregul sistem;
- *Consola de programare*, echipamentul pe care se realizează programul ce va rula pe APB, şi de pe care se încarcă în memoria de programe a APB acest program;
- *Periferice de intrare*[2], subsistemul prin care APB primeşte informaţii din proces (de la întrerupătoare, comutatoare, contactoare, relee, limitatoare);
- *Periferice de ieşire*[3], subsistemul prin care APB trimite comenzi în proces (de exemplu pentru alimentarea unor bobine de relee sau contactoare, sau aprinderea de lămpi de semnalizare);
- *Periferice interne (temporizări/ contorizări),* subsistemul prin care se pot genera intervale de timp şi contorizări de evenimente;

Schimbul de date între modulele componente ale automatului se face prin intermediul *magistralei interne* structurată funcţional în:

- *magistrala de date*, bidirecţională, cu dimensiunea de un bit;
- *magistrala de adrese,* unidirecţională, cu dimensiunea dată de spaţiul de adresare (de exemplu 10 biţi pentru un spaţiu de adresare de 1kbit), pe care *unitatea centrală* depune adresele perifericelor cu care dialoghiază;
- *magistrala de control,* cu semnale de comandă spre periferice.

Toate transferurile de date se fac prin mijlocirea unităţii centrale. Aceasta plasează pe magistrala de adrese adresa modulului cu care doreşte să comunice iar pe magistrala de control

activează semnalul care defineşte sensul informaţiei. Datele de intrare citite sunt prelucrate în unitatea centrală iar rezultatele sunt trimise la ieşiri.

Unitatea centrală

În Figura 3.2. se prezintă schema bloc a unităţii centrale a automatului programabil cu prelucrare pe bit.

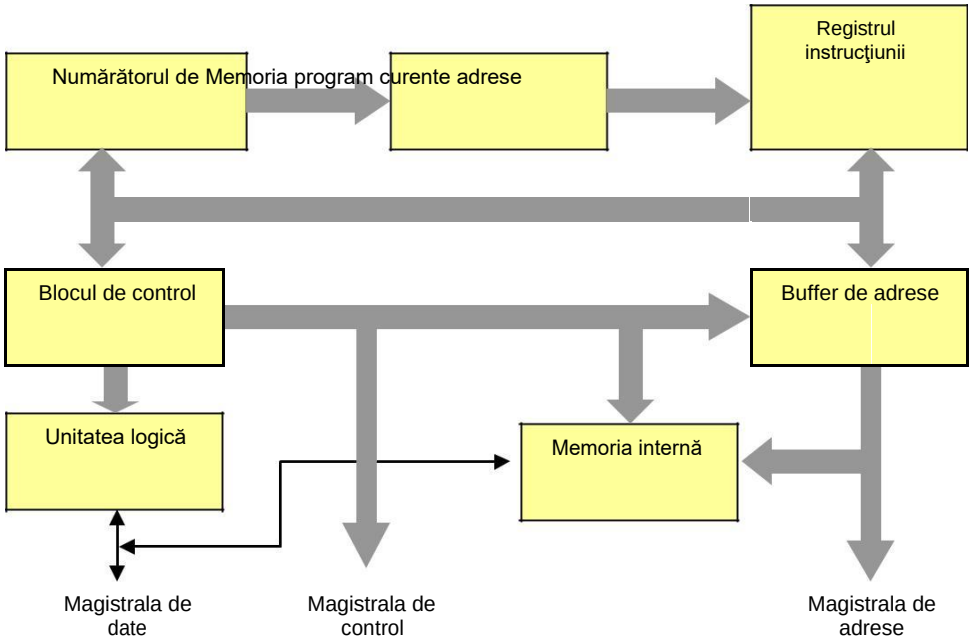


Figura 3.2. Schema bloc a unităţii centrale a unui automat programabil pe bit.

Semnificaţia blocurilor unităţii centrale este următoarea:

Numărătorul de adrese, este un circuit numărător care indică adresa din memorie de unde va fi citită instrucţiunea ce urmează a fi executată;

Memoria program, este un circuit de tip EEPROM în care se află programul APB, încărcat de la consola de programare. La aplicarea la intrarea sa a adresei instrucţiunii ce trebuie executată, la ieşire va trimite codul acestei instrucţiuni, memorat la adresa respectivă

Registrul instrucţiunii curente, este un registru ce se încarcă cu instrucţiunea de executat citită din memoria de program de la adresa indicată de numărătorul de adrese;

 *Blocul de control,* este un circuit ce decodifică codul instrucţiunii aflat în corpul instrucţiunii şi prin semnalele de control rezultate, comandă operaţiile din APB implicate de instrucţiunea curentă;

 *Bufferul de adrese,* este un registru care memorează temporar adresa perifericului sau locaţiei de memorie cu care unitatea centrală face schimb de informaţii;

 *Unitatea logică,* este un circuit ce prelucrează datele achiziţionate de periferice de intrare sau citite din memoria internă şi trimite rezultatul în memoria internă sau la perifericele de ieşire;

 *Memoria internă,* este un circuit de tip RAM destinat memorării variabilelor utilizate în program.

Periferice de intrare[4]

În Figura 3.3. este reprezentată schema bloc a perifericelor de intrare ale automatului programabil pe un bit.

 *Blocul de decodificare a adresei*, este un circuit ce primeşte la intrare codul adresei depusă pe magistrala de adrese de către unitatea centrală, compară această adresă cu adresa proprie implementată hardware şi emite semnal de recunoaştere în caz de coincidenţă a acestora.

 *Blocul de multiplexare a semnalelor de intrare*, este un circuit ce selectează intrarea indicată de decodificatorul adresei şi o depune pe magistrala de date.

 *Blocul de prelucrare a semnalului de intrare*, este un circuit ce adaptează semnalele din proces pentru a deveni compatibile cu cele din automatul programabil. Tot aceste blocuri asigură şi izolarea galvanică între proces şi automatul programabil. Construcţia acestui bloc depinde de tipul semnalelor achiziţionate din proces, respectiv semnal de tensiune sau de curent, semnale continue sau alternative, semnale de nivel mic sau de nivel mare.

[4] În alte abordări se utilizează noţiunea de *interfeţe de intrare* pentru acest bloc funcţional.

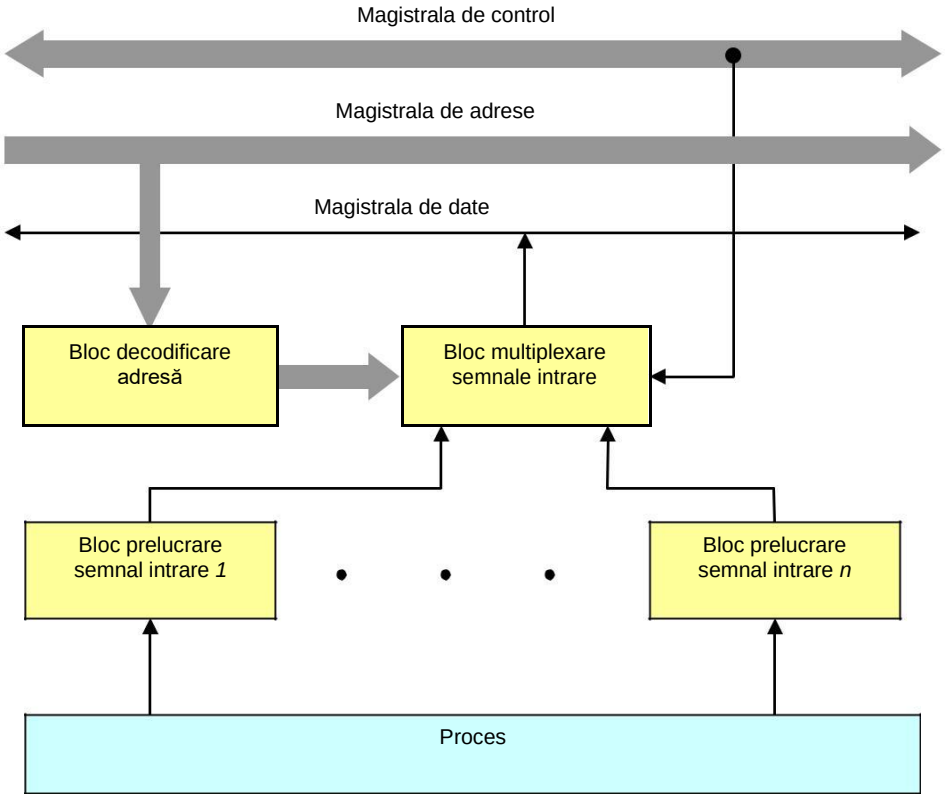


Figura 3.3. Schema bloc a perifericelor de intrare pentru un automat programabil cu prelucrare la nivel bit.

Periferice de ieşire[5]

Schema bloc a perifericelor de ieşire este prezentată în Figura 3.4.

Blocul de decodificare a adresei este un circuit care are aceeaşi semnificaţie şi funcţionalitate ca în cazul perifericelor de intrare.

Blocul de comandă canal este un demultiplexor prin intermediul căruia semnalul de pe magistrala de date este trimis la ieşirea selectată de către decodificatorul adresei. Acest bloc are şi funcţia de memorare a canalului, astfel încât semnalul să fie prezent în permanenţă la intrările blocurilor de ieşire.

Blocul de ieşire este un circuit care realizează adaptarea de nivel a semnalul de ieşire. Ieşirea poate fi prin releu pentru semnale de curent continuu sau alternativ, prin tranzistor pentru semnale în curent continuu de nivel mic sau prin triac pentru semnale alternative de

.

nivel mare. Pentru evitarea perturbaţiilor datorate procesului condus se preferă izolarea galvanică a blocurilor de ieşire de elementele comandate din proces prin: relee intermediare, transformatoare de impuls sau optocuploare.

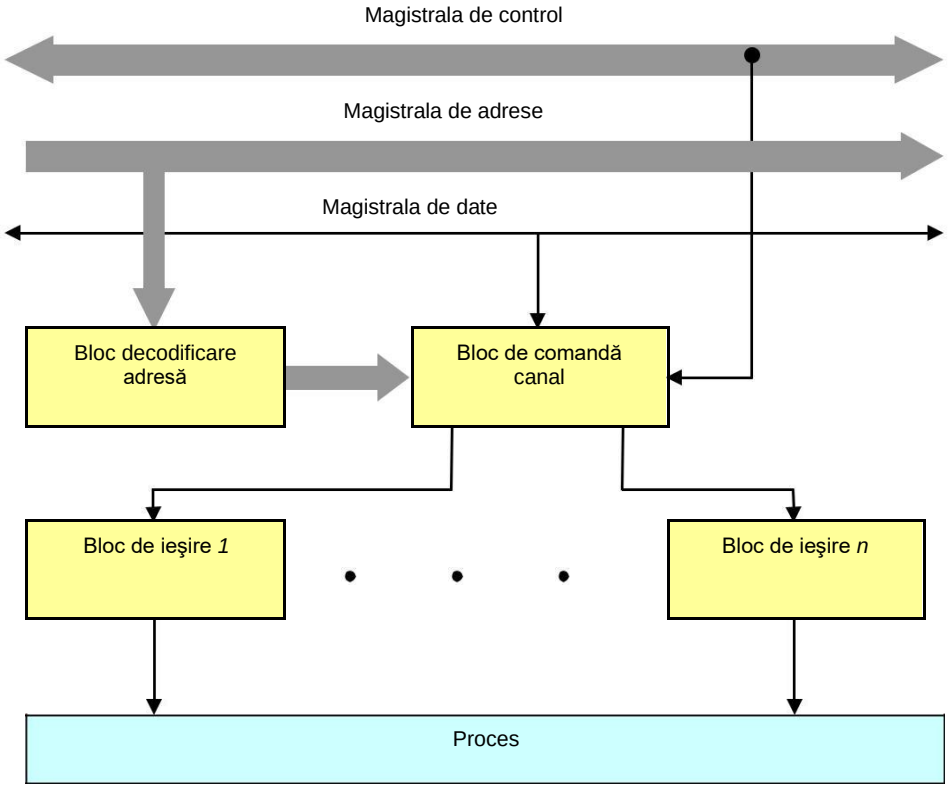


Figura 3.4. Schema bloc a perifericelor de ieşire pentru un automat programabil cu prelucrare la nivel bit.

Periferice interne

În Figura 3.5. este prezentată schema bloc a perifericelor interne. Acestea sunt module de temporizare şi contorizare fiind în acelaşi timp module de intrare şi de ieşire. Ca atare în structura lor intră blocuri prezente în interfeţele de intrare şi ieşire, respectiv: blocul de decodificare a adresei, blocul de multiplexare a semnalelor de intrare şi blocul de comandă canal.

Blocul de decodificare a adresei, este un circuit ce primeşte la intrare codul adresei unui bloc de temporizare din APB, compară această adresă cu adresa proprie implementată hardware şi emite semnal de recunoaştere în caz de coincidenţă a acestor două adrese.

Blocul de multiplexare a semnalelor de intrare, este un circuit ce selectează blocul de temporizare indicat de decodificatorul adresei şi depune informaţia citită pe magistrala de date la momentul indicat de un semnal primit de pe magistrala de control.

Blocul de comandă canal este un circuit demultiplexor prin intermediul căruia semnalul de pe magistrala de date este trimis la blocul de temporizare selectat de către decodificatorul adresei.

Blocul de temporizare este un circuit ce realizează temporizări şi numărări (contorizări). Este "văzut" ca un periferic de ieşire în momentul în care primeşte semnal pentru contorizare sau de iniţiere a temporizării. Este "văzut" ca periferic de intrare în momentul în care temporizarea s-a încheiat sau se citeşte cantitatea contorizată. Poate fi de tip analogic (monostabil) sau numeric (numărător).

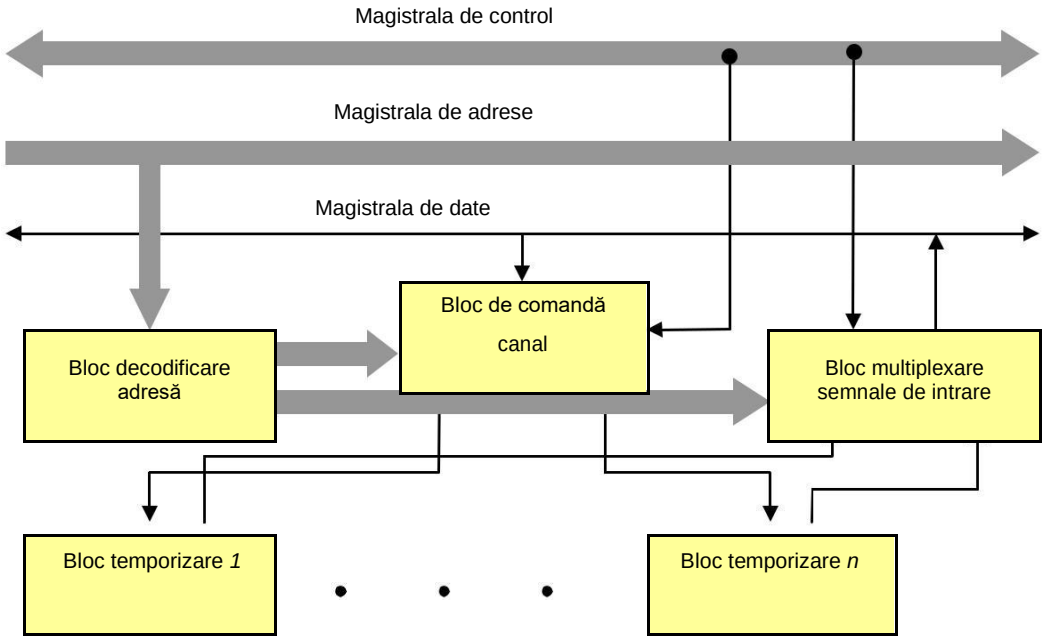


Figura 3.5. Schema bloc a perifericelor interne ale unui automat programabil cu prelucrare la nivel bit.

Operaţii pentru execuţia unei instrucţiuni

Funcţionarea unităţii de control presupune execuţia ciclică a programului, intrucţiune după instrucţiune. Execuţia fiecărei instrucţiuni înseamnă realizarea următoarelor operaţii:

1. Aducerea instrucţiunii de executat din memoria de programe, de la adresa indicată de numărătorul de adrese în registrul instrucţiunii curente.

   Memoria de programe este organizată în cuvinte cu dimensiunea egală cu cea a registrului instrucţiunii curente respectiv cu dimensiunea instrucţiunilor.

2. Incrementarea numărătorului de adrese de către blocul de comandă. Numărătorul de adrese va indica astfel adresa următoarei instrucţiuni ce trebuie executată.

Există două situaţii în care nu este luată în considerare adresa obţinută prin incrementare:
- Dacă următoarea instrucţiune din program nu se află după instrucţiunea curentă, numărătorul de adrese se va încărca cu adresa (numită *de salt*) a acestei instrucţiuni ce se află în corpul instrucţiunii curente, într-un câmp ce-i este destinat special.
- La iniţializarea automatului programabil la punerea sub tensiune sau la apăsarea butonului RESET. În acest caz, numărătorul se încarcă cu adresa de start a programului, adresă stabilită prin comutatoare (switch-uri).

3. Decodificarea codului instrucţiunii în blocul de control şi execuţia operaţiilor implicate, de exemplu: efectuarea operaţiilor logice în unitatea logică, memorarea stărilor în memoria internă de tip RAM, etc.

4. Transmiterea pe magistrala de adrese a adresei perifericului de dialog (după memorarea acesteia în registrul buffer de adrese), şi stabilirea dialogului după recunoaşterea perifericului adresat.

Dacă instrucţiunea curentă este de salt, adresa conţinută în corpul instrucţiunii se va încărca în numărătorul de adrese, nu în bufferul de adrese.

Dacă instrucţiunea curentă nu necesită dialog cu perifericele şi nici salt în program, corpul instrucţiunii poate conţine în loc de adresă chiar operandul. În acest caz biţii din câmpul corespunzător al instrucţiunii sunt trimişi blocului de control.

Execuţia programului este supravegheată de un circuit numărător[6] care este iniţializat în trei situaţii:

- la punerea sub tensiune;
- la apăsarea butonului RESET;
- la sfârşitul fiecărui ciclu de execuţie a programului.

În situţia depăşirii timpului de execuţie alocat pentru program, numărătorul va ajunge la valoarea maximă, şi se blochează unitatea de control iar ieşirile se pun pe 0, pentru a preveni situaţii de avarie în proces.

5.4.2 Programarea automatului programabil cu prelucrare la nivel de bit

Orice automat programabil, pe parcursul funcţionării, execută într-o ordine logică instrucţiuni ce compun programul aflat în memoria de programe. Acest lucru este posibil deoarece instrucţiunile din program aparţin unui set de instrucţiuni concepute special pentru automatul programabil respectiv. Fiecare automat programabil are propriul său set de instrucţiuni ce acoperă toată gama de operaţii necesară achiziţiei datelor din proces, prelucrării lor şi apoi trimiterii comenzilor în proces.

Instrucţiunile sunt de fapt coduri binare, iar programul astfel prezentat se spune că este scris în *cod maşină*, cod executabil de către unitatea de control a automatului programabil.

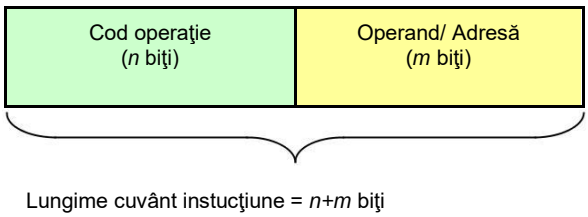
[6] Watchdog

Pentru realizarea programului automatului programabil nu se procedează la scrierea sa de către programator ca atare, în cod maşină, deoarece este nepractic (dificil de scris, dificil de depanat şi cronofag).

Scrierea programului este înlesnită de *limbajul de asamblare*, în cadrul căruia, fiecărei instrucţiuni din cod maşină îi corespunde o *mnemonică* (descriere concisă a instrucţiunii). Conversia programului scris în limbaj de asamblare în limbaj cod maşină este făcută automat de un program numit *compilator*.

Structura instrucţiunii cod maşină

Instrucţiunile automatului programabil conţin două câmpuri cu semnificaţie şi dimensiune diferită, aşa cum se poate vedea în figura de mai jos



Câmpul *Cod operaţie* este specific instrucţiunii şi conţine informaţii referitoare la modul de prelucrare a datelor precum şi semnificaţia celui de-al doilea câmp şi a *modului de adresare*.

Semnificaţia celui de-al doilea câmp poate fi de *operand* sau de *adresă*, caz în care se specifică dacă adresa este de salt sau este adresă de operand.

În automatul programabil cu prelucrare la nivel de bit, există două moduri de adresare:

- Adresare directă, prin care adresa operandului este indicată direct, lungimea de *m* biţi alocată adresei putând defini întreg spaţiul de adresare disponibil al automatului programabil;
- Adresare indexată, prin care adresa operandului este relativă la valoarea dintr-un registru index. Mai precis adresa absolută a operandului se află prin adunarea valorii conţinută în câmpul de adresă al instrucţiunii cu valoarea conţinutul registrului index. Acest de mod adresare permite funcţionarea automatului programabil în regim de multitasking fix[7] sau relocabil[8].

Instrucţiuni de testare a condiţiilor

Aceste instrucţiuni sesizează modificarea stării intrărilor, ieşirilor sau temporizatoarelor interne şi încarcă într-un registru cu semnificaţie specială din unitatea centrală numit *acumulator*, noua stare. Sunt, în esenţă, instrucţiuni de citire din locaţiile adresate. Citirea se face la intervale regulate de timp egale cu durata de execuţie a programului.

[7] Mai multe programe ce rulează (în aparenţă) în acelaşi timp şi ocupă zone de memorie la adrese fixe;

[8] Programele concurente se încarcă la adrese oarecare, gestionate de sistemul de operare.

Generic, mnemonicele acestor instrucţiuni sunt *LD* pentru încărcarea conţinutului locaţiei adresate şi *LDC* pentru încărcarea complementului locaţiei adresate, în registrul acumulator. În limbajul de asamblare, fiecare mnemonică este însoţită de un identificator al adresei locaţiei sursă. De exemplu:

```
LD I1    // încarcă în acumulator conţinutul intrării I1.

LDC T4 // încarcă în acumulator conţinutul complementat
                // al temporizatorului T4.
```

Instrucţiuni de transfer date

Aceste instrucţiuni permit salvarea datelor din acumulator într-o locaţie a memoriei RAM sau într-unul din bistabilele aflate în structura canalelor de ieşire sau temporizare/ contorizare. În esenţă, sunt instrucţiuni de scriere în locaţiile adresate.

Generic, mnemonicele acestor instrucţiuni sunt *STO* pentru încărcarea locaţiei adresate cu conţinutul acumulatorului şi *STOC* încărcarea locaţiei adresate cu conţinutul complementat al acumulatorului. În limbajul de asamblare, fiecare mnemonică este însoţită de un identificator al adresei locaţiei destinaţie. De exemplu:

```
STO O2 // încarcă ieşirea O2, conţinutul acumulatorului.

STOC M1        // încarcă locaţia RAM M1, cu conţinutul
                // complementat al acumulatorului.
```

Observaţie: *Simultan cu scrierea informaţiei în bistabilul canalului de ieşire se face scrierea şi în RAM unde se va păstra o imagine a tuturor canalelor de ieşire. Acest lucru se face din două motive:*

1. *Posibilitatea implementării pe automatele programabile a unor automate finite sevenţiale ce conţin reacţii ieşire-intrare;*
2. *Protecţia comenzilor către proces, comenzi ce pot fi alterate prin modificarea stării bistabilelor din căile de ieşire de perturbaţiile din proces. Acest lucru se realizează prin reîncărcarea stărilor memorate anterior în RAM, în bistabilele canalelor de ieşire, la sfârşitul fiecărui ciclu de execuţie a programului.*

Transferul condiţionat al datelor din acumulator la destinaţie (canal de ieşire, de temporizare sau locaţiei RAM), este realizat de instrucţiunile: *STC*, care setează ieşirea adresată dacă acumulatorul conţine valoarea 1, respectiv *RTC*, care resetează ieşirea adresată dacă acumulatorul conţine valoarea 1.

În limbajul de asamblare, fiecare din aceste mnemonici este însoţită de un identificator al adresei locaţiei destinaţie. De exemplu:

```
STC O1 // setează ieşirea O1, dacă în acumulator se
                află // valoarea 1.
RTC O1 // resetează ieşirea O1, dacă în
                // acumulator se află valoarea 1.
```

Instrucţiuni de prelucrare logică a datelor

Formele normale şi canonice, disjunctive şi conjunctive ale funcţiilor logice pot fi implementate pe automatele programabile deoarece acestea dispun de un set complet de instrucţiuni logice la nivel de bit. Ele prelucrează numai conţinutul acumulatorului în cazul operaţiilor cu un singur operand iar în cazul operaţiilor cu doi operanzi, conţinutul acumulatorului şi al unui canal de intrare, de temporizare sau locaţie RAM. La sfârşitul execuţiei oricărei instrucţiuni logice, rezultatul se află în acumulator.

Instrucţiunile pe doi operanzi sunt:

- *AND,* ce implementează funcţia booleană elementară ŞI dintre acumulator şi un canal de intrare, temporizare sau locaţii RAM;
- *ANDC*, ce implementează funcţia booleană elementară ŞI dintre acumulator şi complemetul unui canal de intrare, temporizare sau locaţii RAM;
- *OR*, ce implementează funcţia booleană elementară SAU dintre acumulator şi un canal de intrare, temporizare sau locaţii RAM;
- *ORC*, ce implementează funcţia booleană elementară SAU dintre acumulator şi complemetul unui canal de intrare, temporizare sau locaţii RAM;
- *XOR*, ce implementează funcţia booleană elementară SAU-EXCLUSIV dintre acumulator şi un canal de intrare, temporizare sau locaţii RAM;

În limbajul de asamblare, fiecare din aceste mnemonici este însoţită de un identificator al adresei locaţiei în care se află cel de-al doilea operand. De exemplu:

```
AND I3 // realizează funcţia ŞI între conţinutul
                // acumulatorului şi conţinutul intrării I3,
                // Rezultatul se încarcă în acumulator.
ORC M5          // realizează funcţia SAU între conţinutul
                // acumulatorului şi complementul conţinutului
                // locaţiei RAM M5. Rezultatul se încarcă
                // în acumulator.
```

Instrucţiunile pe un singur operand sunt:

- *NOT*, care complementează conţinutul acumulatorului;
- *CLR*, care resetează acumulatorul.

Instrucţiuni de salt

Aceste instucţiuni permit întreruperea executării liniare a programului prin încărcarea în numărătorul de adrese a unei adrese diferită de cea obţinută prin incrementare, automat, după aducerea din memorie a instrucţiunii curente.

Saltul necondiţionat se realizează prin instrucţiunea *JMP*, iar cel condiţionat de existenţa valorii 1 în acumulator, de instrucţiunea *JMPC*.

În limbajul de asamblare, mnemonicile sunt însoţite de adresa de salt. De exemplu:

```
JMP   05h   // salt la adresa 05 în hexazecimal, continuă
            // execuţia de la această adresă din memoria de
            // programe.
JMPC  3Ch   // salt la adresa 3C în hexazecimal dacă în
            // acumulator se află 1, continuă execuţia de la
            // această adresă din memoria de programe.
```

Observaţie: *Automatul programabil dispune şi de o instrucţiune numită NOP, ce nu execută nicio operaţie, singurul efect este acela de a consuma un timp egal cu durata unui ciclu de execuţie al unei instrucţiuni. Această instrucţiune se utilizează la generarea temporizărilor sofware.*

Generarea temporizărilor

Circuitele de generare a temporizărilor sunt module specializate din automatul programabil conectate la magistrala internă. Prescrierea temporizării poate fi făcută manual prin intermediul unor microswitch-uri de pe modul, sau prin program funcţie de tipul automatului şi/ sau al modulului. La modulele cu prescriere manuală, circuitul de temporizare poate fi un circuit monostabil sau un circuit numărător.

Temporizarea la anclanşare

Pentru modulul cu temporizare setabilă manual cu monostabil ca circuit de temporizare, se poate utiliza secvenţa de program următoare:

```
        LD I1 // Citeşte starea intrării I1
        // şi o încărca în acumulator.
STO T1  // Scrie conţinutul acumulatorului la
        // intrarea  modulului de temporizare T1
        // Temporizarea începe când data trece de la 0 la
        // 1, momentul t1 Fig.5.1. Temporizarea nu
                // începe dacă data trece de la 0 la 1.
        LD  T1  // Citeşte de la ieşirea temporizatorului T1
            // valoarea şi o transferă în acumulator. La
                // sfârşitul perioadei de temporizare, modulul T1
                // comută, şi ca urmare în acumulator se încarcă
                // 1, momentul t2.
STO  O1 // Conţinutul acumulatorului este
                // transferat ieşirii O1.
                    // În momententul t3 se termină comanda dată prin // I1.
```

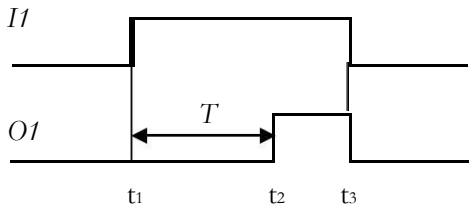


Figura 5.1. Diagrama temporală pentru temporizare la anclanşare.

Temporizarea la declanşare

Pentru modulul cu temporizare setabilă manual cu monostabil ca circuit de temporizare, se poate utiliza secvenţa de program următoare:

```
LDC  I1   // Încărcare acumulator cu conţinutul
```

```
        // complementat al intrării I1.
STO T1    // Se scrie data din acumulator la intrarea
        // în modulul de temporizare T1.
        // La trecerea de 1 la 0, nu se iniţiază
                        // temporizare, momentul t1 în Fig.5.2,
        // iar la trecerea de la 0 la 1 se iniţiază
                        // temporizare, momentul t2.
LD  T1   // Se încarcă acumulatorul cu valoarea de la
        // ieşirea modulului de temporizare T1.
OR  I1    // Se realizează funcţia logică SAU între
        // conţinutul acumulatorului şi intrarea I1.
        // Între momentul comenzii anclanşării ieşirii şi
        // sfârşitul temporizării T, momentul t3,
                        // valoarea din acumulator este 1.
STO O1 // Se transferă conţinutul acumulatorului //
                        la ieşirea O1.
```

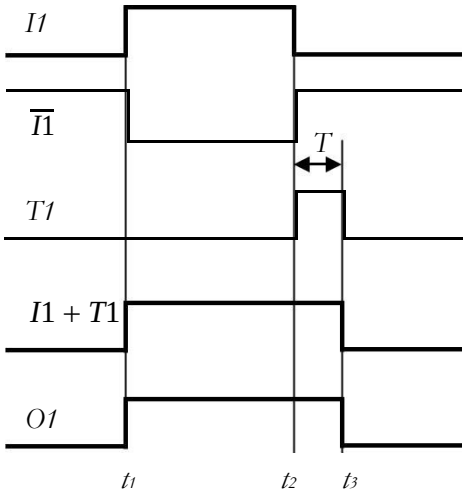


Figura 5.2. Diagrama temporală pentru temporizare la declanşare.

Noţiuni de bază în alegerea soluţiei de automatizare

5.4.3 Noţiuni de bază în alegerea soluţiei de automatizare. Alegerea hardware-ului

Există mai mulţi factori care concură la alegerea tipului de automat programabil.

- Dacă aplicaţia este mai simplă, criteriul de alegere cel mai important este numărul de intrări şi ieşiri precum şi dimensiunea programului utilizator. La aplicaţiile mai complexe, sunt luaţi în considerare şi timpii de răspuns precum şi dimensiunea memoriei care trebuie să înmagazineze un număr mare de date.

- La o maşină unealtă comandată de un automat programabil, numărul de intrări/ ieşiri, dimensiunea memoriei şi timpul de răspuns sunt parametrii definitorii de care se ţine cont la alegerea automatului programabil.

- În cazul proceselor răspândite în mai multe locaţii este mult mai indicată alegerea unor module de intrare/ ieşire distribuite decât a modulelor de intrare/ ieşire dispuse pe automat. Această soluţie duce la reducerea numărului de cabluri de legătură cu procesul, comunicaţia între modulele de intrare/ ieşire şi unitatea centrală a automatului programabil făcându-se prin intermediul magistralei de comunicaţie pe un număr redus de fire. În acest caz şi viteza de răspuns poate fi simţitor mai mare.

- Dacă procesul automatizat poate fi împărţit în procese relativ autonome atunci varianta automatelor programabile dedicate subproceselor, interconectate în reţea, este soluţia cea mai bună. Simplitatea programării fiecărui automat este evidentă. Creştea vitezei de răspuns este evidentă de asemenea comparativ cu varianta reţelei distribuite de module intrare/ ieşire, deoarece achiziţia semnalelor, trimiterea comenzilor şi procesarea datelor se face concentrat, în cadrul fiecărui subproces. Programele care rulează pe fiecare automat programabil sunt mult mai simple, mai scurte şi mai rapide, ele rulând independent unele de altele. Schimbul de date dintre automatele programabile se realizează prin intermediul unei reţele specializate rapide.

De exemplu:

A. *Semaforizarea intersecţiei unui drum principal, cu prioritate, cu un drum secundar;*

B. *Automatizarea unei maşini unelte care trebuie să:*
- *execute mai multe tipuri de prelucrări (strunjire, găurire, frezare, şlefuire, etc.), cu scule de diferite dimensiuni la aceeaşi tip de operaţie;*
- *facă controlul dimensional permanent şi să minimizeze erorilor prin corecţie rapidă;*
- *afişeze rapoarte statistice ale parametrilor de proces;*
- *comunice cu calculatorul de proces de la nivel ierarhic superior.*

Alegerea limbajului de programare

Alegerea limbajului de programare depinde de utilizator şi de complexitatea algoritmului de conducere.

În cazul prelucrării datelor binare este recomandabil să se utilizeze limbajul LAD sau FBD, limbaje care sunt mult mai intuitive.

În cazul manipulării de variabile complexe şi adresări indirecte este indicat limbajul STL care este asemănător limbajelor de programare de nivel înalt şi permite procesarea unui volum mare de date.

Crearea proiectului

Mediile de programare actuale permit optimizarea proiectării, ele oferind facilităţi în organizarea resurselor necesare. Datele culese pentru realizarea proiectului sunt structurate ierarhic.

Scrierea, analiza şi salvarea unui program

Programul care cuprinde instrucţiunile necesare realizării sarcinii impuse prin tema de proiectare este recomandat a fi modular.

Modulele de program pot fi:

- orientate către proces, caz în care, fiecare modul corespunde unei părţi din proces sau maşini;
- orientate funcţional, caz în care modulele corespund funcţiilor din proces, ca

de exemplu: comunicare, mod de operare, etc.

După scriere, programul este testat. Testarea poate fi făcută pe un automat programabil virtual implementat chiar în mediul de progamare, sau în automatul programabil real, după încărcarea programului în memoria de programe a acestuia.

După testarea cu succes a programului acesta este încărcat în memoria EPROM şi apoi este generată documentaţia aferentă.

5.4.4. Limbaje de programare ale automatelor programabile

Principala cerinţă a limbajului de programare pentru un automat programabil este aceea de a fi uşor de înţeles şi de utilizat în aplicaţii de conducere a proceselor.

Cei mai mulţi producători de automate programabile oferă aceleaşi tipuri de instrucţiuni de bază, dar există, în general, diferenţe de formă, operaţii etc., de la un producător la altul.

Comisia Electrotehnică Internaţională (IEC) a dezvoltat standardul IEC 1131-3 care recomandă diferiţilor producători să ofere acelaşi set de instrucţiuni. Setul de instrucţiuni al acestui standard este mai mic decât cel oferit de producători. Normele IEC 1131-3 definesc SFC (Sequential Function Chart) ca fiind un mijloc destinat pentru structurarea şi organizarea unui program. SFC are la bază reprezentarea sub formă de reţea GRAFCET a acţiunilor secvenţiale.

Standardul IEC 1131-3 defineşte două limbaje literale:

- STL[9] (**ST**atement **L**ist) – Listă de instrucţiuni, cu structură asemănătoare limbajelor de asamblare ale microprocesoarelor;

- ST (**S**tructured **T**ext) – Text structurat, care foloseşte instrucţiuni de atribuire, de selecţie şi de control a subprogramelor având o structură apropiată de limbajele de nivel înalt,

şi două limbaje (semi)grafice:

- LD (**L**adder **D**iagram) – Diagramă scară, care permite programarea aplicaţiilor într-o manieră asemănătoare cu proiectarea unui circuit cu contacte şi relee. Limbajul operează numai cu variabile booleene;

- FBD (**F**unction **B**lock **D**iagram) – Diagramă cu blocuri de funcţii, care este o extensie a limbajului LD, conţinând şi blocuri complexe. Acest limbaj permite operarea şi cu variabile de tip real.

Tipurile de date elementare definite de standard sunt:

[9] Sau IL (Instruction List)

- Booleene, notate cu BOOL;
- Întregi, notate cu INT;
- Cuvinte (16 biţi), notate cu WORD;
- Cuvinte duble (32 biţi), notate cu DWORD;
- Reale (32 biţi), notate cu REAL;
- Şiruri de caractere, notate cu STRING;
- Timp şi dată, notate cu TIME repectiv cu DATE.

Este permisă şi utilizarea de date de tip tablou (ARRAY) şi structură (STRUCT), precum şi derivate ale acestora.

Identificarea datelor se face utilizând atât adrese *absolute* (adresare directă) cât şi *simbolice* (adresare indirectă).

Adresarea directă utilizează denumirea zonei de memorie pentru identificarea adresei. Denumirile zonelor de memorie pot cuprinde două prefixe. Primul prefix poate fi:

- %I, pentru intrări;
- %Q, pentru ieşiri;
- %M, pentru variabilele

interne. Al doilea prefix poate fi:

- x, y, pentru variabilele de tip boolean. Valoarea x reprezintă octetul, iar valoarea y reprezintă bitul;
- B, pentru octet (Byte);
- W, pentru cuvânt (Word);
- D, pentru dublu cuvânt (Double word).

De exemplu:

- %Ix.y, reprezintă o variabilă de intrare booleană reprezentând bitul y din octetul x;
- %QBx, reprezintă o variabilă de ieşire booleană reprezentată de octetul x;
- %MWx, reprezintă o variabilă internă booleană reprezentată de cuvântul x;
- %IDx, reprezintă o variabilă de intrare booleană reprezentată de cuvântul dublu x;

Adresarea indirectă utilizează identificatorii, care sunt şiruri de caractere alfanumerice, începând cu o literă, pentru identificarea adresei. În aceste cazuri este nevoie de editarea unei tabele de simboluri pentru a face legătura dintre adresa absolută şi cea indirectă.

Limbajul de programare STL

Este un limbaj de nivel scăzut. Este utilizat pentru realizarea aplicaţiilor mici sau pentru optimizarea codului anumitor părţi ale unor aplicaţii.

Un program STL este o listă de instrucţiuni de diferite tipuri, care calculează, de obicei, termeni ai unor expresii logice, rezultatul fiind de asemenea o valoare logică.

Fiecare instrucţiune începe pe o linie nouă, conţine un operator, completat eventual cu un modificator şi, dacă este nevoie de unul sau mai mulţi operanzi separaţi prin virgulă aşa cum se poate vedea mai jos

Eticheta: *Operaţie Operand1[, Operand2]* *(* Comentariu *)*

(Operator + Modificator)

Operanzii instrucţiunilor sunt variabile interne, intrări sau ieşiri ale automatului programabil, mai precis referinţe la memoria fizică.

La instrucţiunile cu un singur operand, celălalt operand este implicit fiind reprezentat de conţinutul unui registru, de obicei registrul acumulator. Operaţia descrisă de operator se execută între operatorul scris explicit şi conţinutul acumulatorului, iar rezultatul se încarcă tot în acumulator.

Documentarea programelor se face utilizând comentarii. Comentariile se pot face pe aceeaşi linie cu instrucţiunea sau pe linii separate. Identificrea comentariilor se face cu ajutorul grupului de caractere (* la început şi *) la sfârşit, sau cu grupul // numai la început de comentariu.

Operatori STL

A. Operatori pentru date booleene

- Operatori de transfer:
  LD – Transferă datele din memorie în acumulator;
  ST sau = - Transferă datele din acumulator în memorie.

Exemplu:

```
LD %I0.0        //Transferă conţinutul intrării I0.0 în
                //acumulator
ST %Q1.0        //Transferă la ieşirea Q1.0 conţinutul
                //acumulatorului
= %Q1.2         //Transferă la ieşirea Q1.2 conţinutul
                //acumulatorului
```

- Operatori de setare/ resetare a
  operanzilor: S – Set-are operand;
  R – Reset-are operand.

Exemplu:

```
S %I0.0 //Setează bitul I0.0
S %M0.3         //Resetează bitul M0.3
```

- Operatori logici:
  AND – Realizează operaţia logică ŞI între conţinutul acumulatorului şi operand; OR - Realizează operaţia logică SAU între conţinutul acumulatorului şi operand;
  XOR - Realizează operaţia logică SAU-EXCLUSIV între conţinutul acumulatorului şi operand;

Exemplu:

```
AND %M0.0       //Realizează operaţia ŞI între conţinutul
                // acumulatorului şi operandul M0.0. Rezultatul
                // se păstrează în acumulator.
OR %M0.0        //Realizează operaţia SAU între conţinutul
                // acumulatorului şi operandul M0.0. Rezultatul
                // se păstrează în acumulator.
XOR %M0.0       // Realizează operaţia SAU-EXCLUSIV între
                // conţinutul acumulatorului şi operandul M0.0.
                // Rezultatul se păstrează în acumulator.
```

B. Operatori pentru date pe octet, cuvânt sau dublu cuvânt

- Operatorul de transfer MOV. Datele se transferă între o sursă şi o destinaţie. Pentru a specifica tipul datei operatorul se suplimentează cu un: B, pentru transferul unui octet, W, pentru transferul unui cuvânt şi DW, pentru transferul unui cuvânt dublu. Instrucţiunea are 2 operanzi, primul fiind sursa iar cel de-al doilea destinaţia.

Exemplu:

```
MOVB %MB0, %MB1     // Realizează transferul octetului din
                                    // MB0 în MB1
MOVW %MW0, %MW2 // Realizează transferul cuvântului din
                                    // MW0 în MW2
```

- Operatori aritmetici:
  ADD – Adunarea cu un operand a conţinutului acumulatorului;
  SUB – Scăderea cu un operand a conţinutului acumulatorului;
  MUL – Înmulţirea cu un operand a conţinutului acumulatorului;
  DIV – Împărţirea întreagă cu un operand a conţinutului acumulatorului.

Exemplu:

```
// Secvenţa de program următoare face adunarea operanzilor
// a şi b, rezultatul păstrându-se în c. Variabilele a, b
// şi c sunt variabile simbolice de tip întreg
LD a          // Încarcă a în acumulator
ADD   b       // Adună b la conţinutul acumulatorului
ST    c       // Salvează conţinutul acumulatorului în c
```

- Operatori relaţionali:
  GT – Verifică dacă valoarea din acumulator este mai mare decât valoarea unui operand. Dacă DA, setează acumulatorul, dacă NU îl resetează;
  GE - Verifică dacă valoarea din acumulator este mai mare sau egală cu valoarea unui operand. Dacă DA, setează acumulatorul, dacă NU îl resetează;
  EQ - Verifică dacă valoarea din acumulator este egală cu valoarea unui operand. Dacă DA, setează acumulatorul, dacă NU îl resetează;
  NE - Verifică dacă valoarea din acumulator este diferită de valoarea unui operand. Dacă DA, setează acumulatorul, dacă NU îl resetează;
  LE - Verifică dacă valoarea din acumulator este mai mică sau egală cu valoarea unui operand. Dacă DA, setează acumulatorul, dacă NU îl resetează;
  LT - Verifică dacă valoarea din acumulator este mai mică decât valoarea unui operand. Dacă DA, setează acumulatorul, dacă NU îl resetează;

Exemplu:

```
// Secvenţa de program următoare setează variabilele
// booleene X şi y funcţie de rezultatul comparaţiilor a>b,
// respectiv b>c. Variabilele a, b şi c sunt variabile
// simbolice de tip întreg
LD    a       // Încarcă a în acumulator
GT    b       // Compară cu b
ST    x       // Memorează rezultatul în x
LD    b       // Încarcă b în acumulator
GT    c       // Compară cu c
ST    y       // Memorează rezultatul în y
```

- Operatori de salt:
  JMP – Salt necondiţionat la o adresă diferită de adresa din numărătorul de adrese;
  CALL – Salt la o adresă de la care începe o subrutină [10];
  RET – Apare obligatoriu în corpul unei subrutine şi produce salt la adresa următoare adresei instrucţiunii CALL apelante.

Exemplu:

```
        // Secvenţa de program următoare memorează în c rezultatul
        // operaţiei a-b dacă a≥b sau rezultatul operaţiei b-a dacă
        // a<b. Variabilele a, b şi c sunt variabile simbolice de
        // tip întreg.
        LD      a       // Încarcă a în acumulator
        GE      b       // Verifică dacă a≥b
        JMPC    UNU     // Dacă DA, sare la adresa de etichetă UNU
        LD      b       // Dacă NU, atunci a<b şi se va face operaţia b-a
        SUB     a       // Se scade din acumulatorul încărcat cu b, a
        ST      c       // Se memorează rezultatul în c
        JMP     CONT    // Se face salt necondiţionat la adresa CONT
UNU:    LD      a       // Se încarcă a în acumulator
        SUB     b       // Se scade din acumulatorul încărcat cu a, b
        ST      c       // Se memorează rezultatul în c
CONT:
```

În secvenţa de program anterioară s-a utilizat un modificator pentru realizarea unui salt condiţionat, *C* adăugat lui *JMP* rezultând *JMPC*. Modificatorul este un caracter care este ataşat operatorului şi poate realiza:

- Negarea operandului - litera folosită cel mai adesea fiind N;

Exemplu:

```
        ANDN        %I0.1   // Realizează operaţia ŞI între
                            // conţinutul acumulatorului şi
                            // intrarea I0.1 negată
```

- Întârzierea operatorului - caracterul utilizat de regulă fiind (;

Exemplu:

```
        AND ( %I0.2
              OR %I0.3
              )     // Întârzie aplicarea operandului ŞI, realizând
                    // mai întâi operaţia logică SAU între intrările
                    // I0.2 şi I0.3. La întâlnirea parantezei închise
                    // se realizează operaţia logică ŞI între
                    // acumulator şi rezultatul operaţiei SAU
                    // anterioare
```

- Realizarea unei operaţii condiţionate, caracterul utilizat fiind *C*, aşa cum a fost exemplificat în secvenţa de program anterioară.

[10] secvenţă de program cu funcţionalitate distinctă.

Etichetele sunt utilizate pentru specificarea punctelor ţintă ale instrucţiuilor de salt. O instrucţiune poate avea o etichetă urmată opţional de :. O etichetă poate fi scrisă şi pe o linie separată.

Limbajul de programare LAD

Limbajul LAD este un limbaj grafic. El este utilizat la realizarea aplicaţiilor de către programatori care au experienţe anterioare în proiectarea aplicaţiilor cu relee şi contacte.

Limbajul LAD realizează o transpunere grafică a ecuaţiilor booleene, realizând combinaţii între contacte (variabile de intrare) şi bobine (variabile de ieşire).

Simbolurile grafice ale limbajului sunt plasate în diagramă în mod asemănător contactelor şi releelor dintr-o schemă electrică, Figura 7.1. Corespondenţa elementelor este evidentă: I_1 - %I1.0, I_2 – %I1.1, I_3 – %I1.2, k – %Q0.1

Un program în limbajul LAD este alcătuit din reţele ce utilizează simboluri grafice. Reţeaua este conectată în partea stângă şi partea dreaptă la barele de alimentare de la o sursă de putere. Execuţia unui program se face de sus în jos şi de la stânga la dreapta.

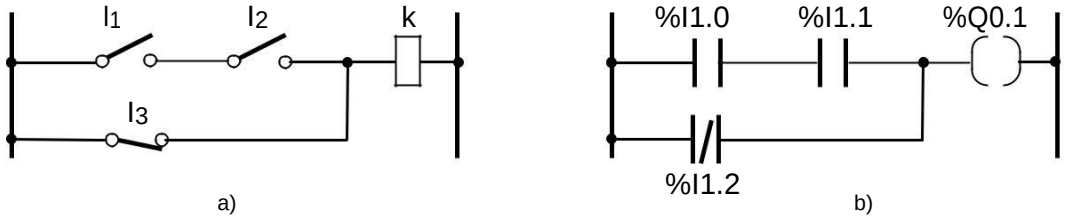


Figura 7.1. Analogia dintre schemele electrice cu contacte şi relee a) şi programele realizate în limbajul LAD din automatele programabile b).

Contactele şi bobinele sunt conectate la barele de alimentare prin linii orizontale şi verticale. Fiecare segment al unei linii poate avea starea TRUE sau FALSE. Starea segmentelor legate împreună este aceeaşi. Orice bară orizontală legată la bara din stânga se află în starea TRUE.

În Figura 7.2. sunt prezentate simbolurile grafice de bază ale limbajului LAD, conform IEC 1131-3.

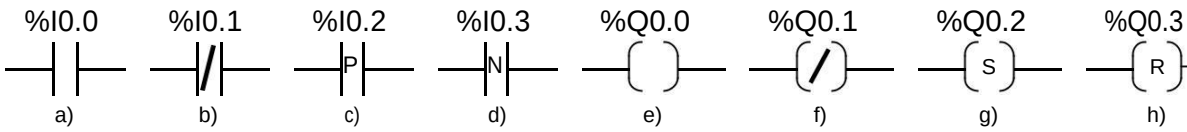
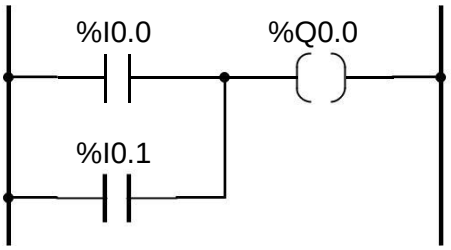


Figura 7.2. Simbolurile grafice de bază ale limbajului LAD.

Contactul direct[11], Figura 7.2.a, realizează operaţia ŞI între starea legăturii stângi şi valoarea variabilei booleene asociate.

Exemplu:

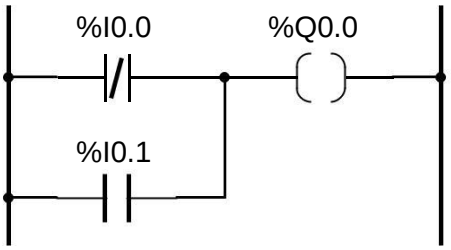


Echivalenţa STL

```
LD      %I0.0
OR      %I0.1
ST      %Q0.0
```

Contactul inversat[12], Figura 7.2.b, realizează operaţia ŞI între starea legăturii stângi şi valoarea variabilei booleene negate asociate.

Exemplu:

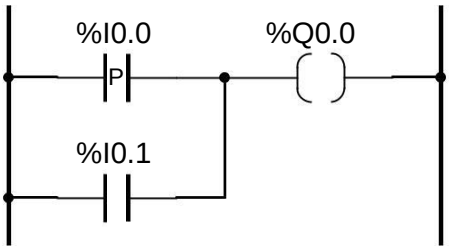


Echivalenţa STL

```
LDN     %I0.0
OR      %I0.1
ST      %Q0.0
```

Contactul de sesizare a frontului crescător[13], Figura 7.2.c, realizează operaţia ŞI între starea legăturii stângi şi frontul crescător al variabilei booleene asociate.

Exemplu:



Contactul de sesizare a frontului descrescător[14], Figura 7.2.d, realizează operaţia ŞI între starea legăturii stângi şi frontul descrescător al variabilei booleene asociate.

---

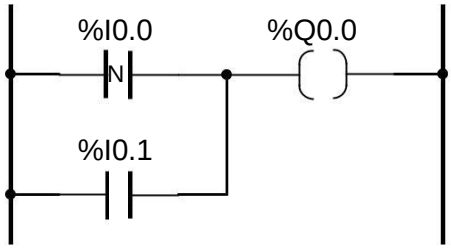
[11] Contactul normal deschis

[12] Contactul normal închis

[13] Frontul pozitiv

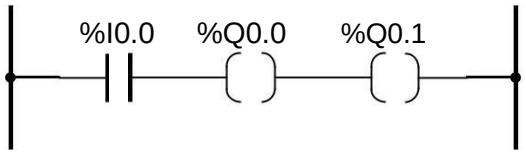
[14] Frontul negativ

Exemplu:



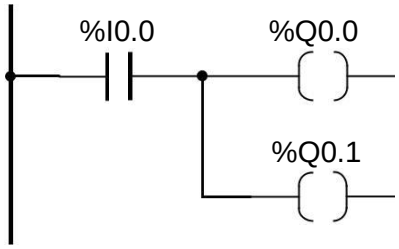
Bobina directă, Figura 7.2.e, realizează o asociere între o variabilă de ieşire şi starea legăturii stângi. La unele implemetări starea legăturii stângi se propagă spre legătura dreaptă putându-se astfel conecta în serie mai multe bobine. La alte implementări, pentru a conecta mai multe bobine, acestea trebuie conectate în paralel. Legătura dreaptă este realizată efectiv sau se consideră legată, aşa cum se poate vedea în exemplele de mai jos.

Exemple:



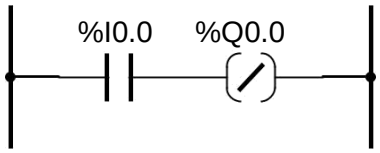
Echivalenţa STL

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



Bobina inversă, Figura 7.2.f, realizează o asociere între o variabilă de ieşire şi starea negată a legăturii stângi. La unele implementări starea legăturii stângi se propagă spre legătura dreaptă putându-se astfel conecta în serie mai multe bobine. La alte implementări nu există acest tip de bobină.

Exemplu:

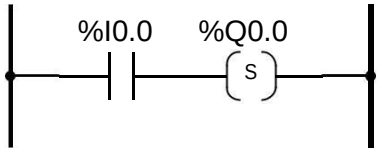


Echivalenţa STL

```
LD      %I0.0
STN     %Q0.0
```

Bobina de setare, Figura 7.2.g, realizează o setare a variabilei de ieşire asociate atunci când starea legăturii stângi devine TRUE. Valoarea variabilei rămâne TRUE până când o instrucţiune inversă, de resetare, se aplică aceleiaşi variabile.

Exemplu:

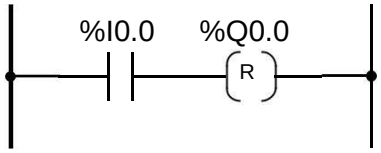


Echivalenţa STL

```
LD      %I0.0
S       %Q0.0
```

Bobina de resetare, Figura 7.2.h, realizează o resetare a variabilei de ieşire asociate atunci când starea legăturii stângi devine TRUE. Valoarea variabilei rămâne FALSE până când o instrucţiune inversă, de setare, se aplică aceleiaşi variabile.

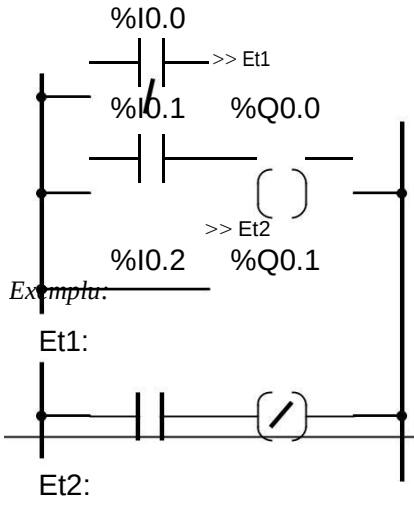
Exemplu:



Echivalenţa STL

```
LD      %I0.0
R       %Q0.0
```

Etichete, salturi necondiţionate şi condiţionate. Într-un program LAD se pot utiliza etichete, salturi necondiţionate şi condiţionate pentru a controla execuţia programului. Eticheta se pune pe bara de alimentare stângă sau într-o reţea separată. Medii de programare diferite utilizează simboluri grafice diferite pentru etichete.



Echivalenţa STL

```
        LDN     %I0.0
        JMPC    Et1
        LD      %I0.1
        ST      %Q0.0
        JMP     Et2
Et1:    LD      %I0.2
        STN     %Q0.1
```

Exemplu:

Majoritatea mediilor de programare au posibilitatea de a converti un program LAD într-un STL şi invers. Această facilitate poartă denumirea de *reversibilitate* şi arată faptul că indiferent cum este scris programul, el va fi memorat sub formă STL.

Limbajul FBD

FBD este un limbaj grafic. El permite programatorului să construiască funcţii complexe utilizând blocurile existente în bibliotecile mediului de programare.

Un program FBD este alcătuit din blocuri de funcţii elementare, conectate între ele prin linii de legătură. Ca şi programul LAD, programul FBD se execută de sus în jos şi de la stânga la dreapta.

Fiecare bloc are un număr de intrări şi ieşiri. Blocul este reprezentat printr-un dreptunghi. Intrările sunt în partea stângă, iar ieşirile în partea dreaptă. Un bloc elementar realizează o singură prelucrare asupra intrărilor. Funcţia realizată de bloc este scrisă în interiorul acestuia. La intrările unui bloc sunt legate variabilele de intrare, iar variabilele de ieşire ale blocurilor pot fi conectate la ieşirile automatului programabil sau la intrările altor blocuri. Tipul variabilelor de intrare trebuie să coincidă cu tipul cerut de intrarea blocului. Ieşirea blocului are acelaşi tip cu intrările.

Conform recomandărilor IEC, Figura 7.3, orice bloc are, pe lângă intrările asupra cărora realizează operaţii *X* respectiv *Y*, o intrare numită *EN* şi o ieşire numită *ENO* pe lângă ieşirea *Z*. Când EN este FALSE nu se operează asupra intrărilor de date iar ieşirea ENO este FALSE. Când EN devine TRUE, blocul devine operaţional iar ieşirea ENO trece în starea TRUE. Dacă în cursul operării apare o eroare, ieşirea ENO trece în starea FALSE.

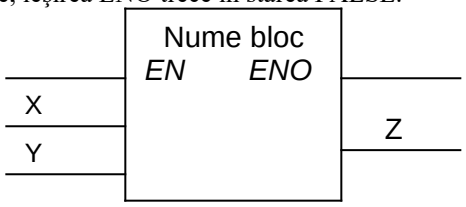


Figura 7.3. Bloc funcţional, conform recomadărilor IEC 1131-3.

Principalele blocuri pot fi împărţite în următoarele categorii:

1. *Blocuri standard*, care corespund operatorilor standard ai limbajului STL;
2. *Blocuri speciale*, implementate prin proceduri complexe.

Blocurile standard sunt: blocuri de manipulare a datelor[15], blocuri pentru operaţii booleene (AND, OR, XOR, etc., Figura 7.4), blocuri aritmetice (pentru efectuarea operaţiilor elementare, adunare, scădere, înmulţire şi împărţire), blocuri de comparaţie.

[15] Se mai numesc şi blocuri de asignare

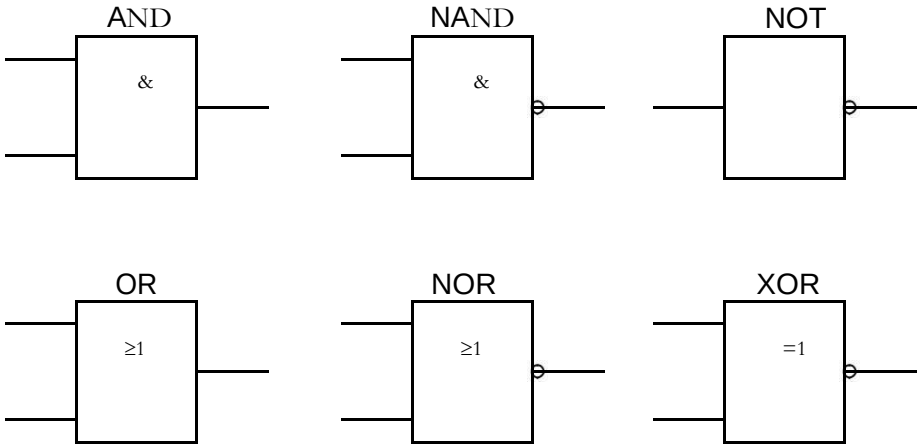


Figura 7.4. Blocuri standard ale limbajului FBD

Blocurile speciale sunt: blocuri de manipulare a datelor (bistabile SR şi RS şi de detecţie a fronturilor crescătoare şi descrescătoare, multiplexoare, generatoare de numere aleatoare), contoare, temporizatoare, blocuri de procesare a semnalelor (histerezis sau trigger Schmitt, regulatoare PID, integratoare, derivatoare, etc.), blocuri generatoare de semnal (generatoare de semnal dreptunghiular, generatoare de semnal modulat în durată PWM), blocuri matematice (de calcul a valorii absolute, a funcţiei exponenţiale, a logaritmului, a rădăcinii pătrate, a funcţiilor trigonometrice, etc.).

În figura 7.5. este prezentată forma generală a unei secvenţe de program realizată în limbajul FBD.

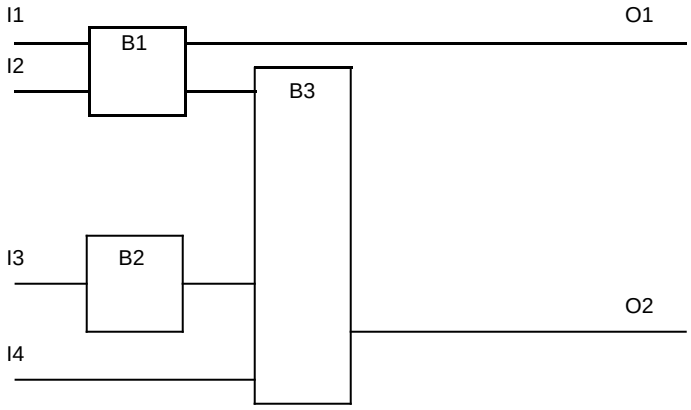


Figura 7.5. Exemplu de program realizat în limbajul FBD.

5.5. Utilizarea automatelor programabile pe bit la implementarea automatelor cu stări finite, definite prin diagrame de stare

La implementarea cu automatelor programabile pe bit a automatelor cu stări finite, definite prin diagrame de stare, programul se poate scrie pornind direct de la diagrama stărilor. Se recomandă alocarea fiecărei stări a automatului o variabilă internă (locaţie din RAM). De asemenea se alocă adrese variabilelor de intrare şi ieşire din diagrama stărilor. Variabilele de intrare, de ieşire, de stare şi de temporizare reprezintă parametrii programului.

Programul începe cu o parte de iniţializare prin care variabilele de stare primesc valorile iniţiale. În partea de lucru a programului se parcurge diagrama stărilor într-o ordine arbitrară. Se testează condiţiile de trecere dintr-o stare în alta acolo unde există aceste condiţii de tranziţie. La sfârşitul programului de lucru se dau ieşirilor valorile varibilelor de stare actualizate, după programul se reia.

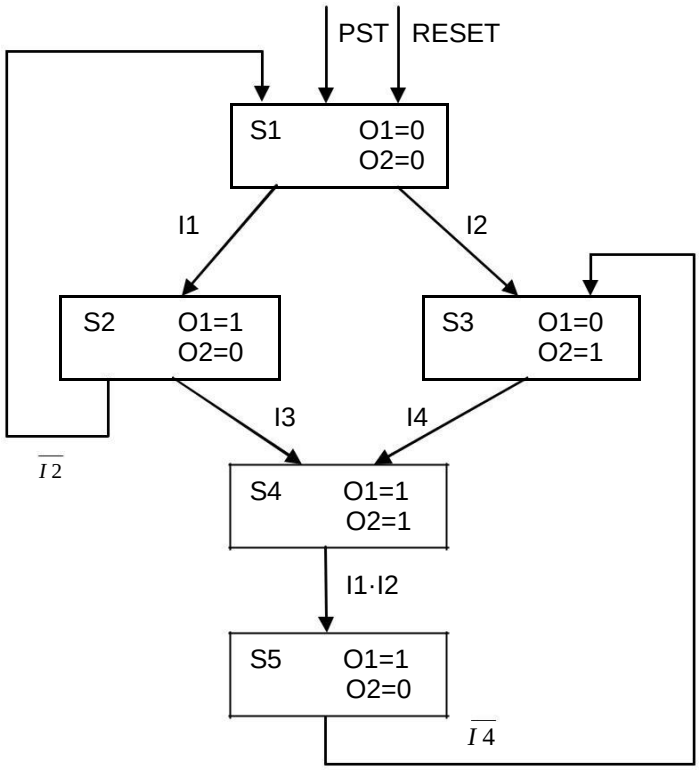


Figura 8.1. Diagrama stărilor unui automat cu stări finite.

În Figura 8.1 se prezintă diagrama de stare a unui automat finit cu 5 stări, 4 intrări şi 2 ieşiri. Iniţializarea hardware a automatului se face la punerea sub tensiune, semnalul PST (Power Start) sau la apăsarea butonului RESET.

Stărilor S1, S2, …, S5 ale automatului li se asociază variabilele M1, M2, …, M5, intrărilor variabilele I1, I2, I3 şi I4, iar ieşirilor O1 şi O2. Programul pe automatul programabil pe bit prezentat anterior, care implementează acest automat cu stări finite este următorul:

```
START:
// Tranziţia S1->S2
        LD      M1      //Se încarcă acumulatorul cu starea S1
        AND     I1      //ŞI între acumulator şi intrarea I1, deci S1·I1
        RTC     M1      //Se resetează M1 dacă acumulatorul este 1
        STC     M2      //Se setează M2 dacă acumulatorul este 1

//-------------------------------------
// Tranziţia S1->S3
        LD      M1      //Se încarcă acumulatorul cu starea S1
        AND     I2      //ŞI între acumulator şi intrarea I2, deci S1·I2
        RTC     M1      //Se resetează M1 dacă acumulatorul este 1
        STC     M3      //Se setează M3 dacă acumulatorul este 1

//-------------------------------------
// Tranziţia S2->S1
        LD      M2      //Se încarcă acumulatorul cu starea S2
        ANDC    I2      //ŞI între acumulator şi intrarea I2 negată, deci
                        //S2·not(I2)
        RTC     M2      //Se resetează M2 dacă acumulatorul este 1
        STC     M1      //Se setează M1 dacă acumulatorul este 1

//-------------------------------------
// Tranziţia S2->S4
        LD      M2      //Se încarcă acumulatorul cu starea S2
        AND     I3      //ŞI între acumulator şi intrarea I3, deci S2·I3
        RTC     M2      //Se resetează M2 dacă acumulatorul este 1
        STC     M4      //Se setează M4 dacă acumulatorul este 1

//-------------------------------------
// Tranziţia S3->S4
        LD      M3      //Se încarcă acumulatorul cu starea S3
        AND     I4      //ŞI între acumulator şi intrarea I4, deci S3·I4
        RTC     M3      //Se resetează M3 dacă acumulatorul este 1
        STC     M4      //Se setează M4 dacă acumulatorul este 1

//-------------------------------------
// Tranziţia S4->S5
        LD      M4      //Se încarcă acumulatorul cu starea S4
        AND     I1      //ŞI între acumulator şi intrarea I1, deci S4·I1
        AND     I2      //ŞI între acumulator şi intrarea I2, deci
                        //S4·I1·I2
        RTC     M4      //Se resetează M4 dacă acumulatorul este 1
        STC     M5      //Se setează M5 dacă acumulatorul este 1

//-------------------------------------
// Tranziţia S5->S1
        LD      M5      //Se încarcă acumulatorul cu starea S5
        ANDC    I4      //ŞI între acumulator şi intrarea I4 negată, deci
                        //S5·not(I4)
        RTC     M5      //Se resetează M5 dacă acumulatorul este 1
        STC     M1      //Se setează M1 dacă acumulatorul este 1

//-------------------------------------
// Actualizare O1
        LD      M2      //Se încarcă acumulatorul cu starea S2
        OR      M4      //SAU între acumulator şi starea M4, deci S2+S4
        OR      M5      //SAU între acumulator şi starea M5, deci
                        //S2+S4+S5
        STO     O1      //Se actualizează O1 cu valoarea din acumulator

//-------------------------------------
```

```
// Actualizare O2
        LD      M3      //Se încarcă acumulatorul cu starea S3
        OR      M4      //SAU între acumulator şi starea M4, deci S3+S4
        STO     O2      //Se actualizează O2 cu valoarea din acumulator
//----------------------------------------
        JMP     START

INIT:
        CLR             //Resetare acumulator
        STOC    M1      //Încarcă starea S1=1
        STO     M2      //Încarcă starea S2=0
        STO     M3      //Încarcă starea S3=0
        STO     M4      //Încarcă starea S4=0
        STO     M5      //Încarcă starea S5=0
        JMP     START
```

5.6 Conectarea unui automat programabil la un proces de automatizare

Conectarea intrărilor fizice ale automatelor programabile

Conectarea unei intrări se face în funcţie de modul de realizare a acesteia de către producător. Există două variante de intrări: în curent continuu şi în curent alternativ. Conectarea intrărilor de curent continuu este prezentată în Figura 9.1.

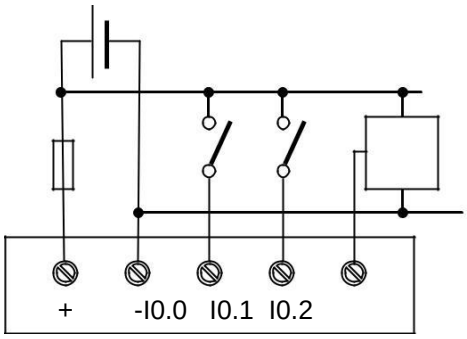


Figura 9.1. Conectarea intrărilor automatului programabil la o sursă de curent continuu.

La schema din Figura 9.1, bornele + şi – se conectează la o sursă exterioară de curent continuu, protejată la supracurenţi printr-o siguranţă montată pe plusul sursei . Intrările I0.0 şi I0.1 se leagă contacte (de la relee, contactoare, limitatoare) alimentate de la plusul sursei de curent continuu. La intrarea I0.2 se conectează ieşirea de semnal a unui traductor alimentat şi el de la aceeaşi sursă de curent continuu exterioară.

Observaţie: *La unele automate programabile nu este necesară o sursă de curent continuu exterioară deoarece această tensiune este obţinută în interiorul automatului programabil. Şi în acest caz automatul este prevăzut cu borne de curent continuu.*

Conectarea intrărilor de curent alternativ este prezentată în Figura 9.2. În această schemă, bornele *L* (Line) şi *N* (Neutral) se conectează la o sursă de curent alternativ. Borna L este protejată la supracurenţi printr-o siguranţă. Sursa de tensiune alternativă alimentează şi contactele din proces conectate la intrările I0.0, I0.1 şi I0.2 ale automatului programabil.

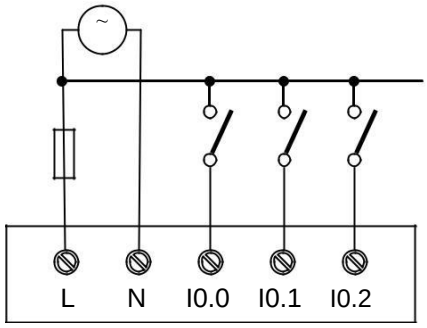


Figura 9.2. Conectarea intrărilor automatului programabil la o sursă de curent alternativ.

Conectarea ieşirilor fizice ale automatelor programabile

Ieşirile automatelor programabile actuale pot fi prin tranzistor sau prin releu, Figura 9.3.

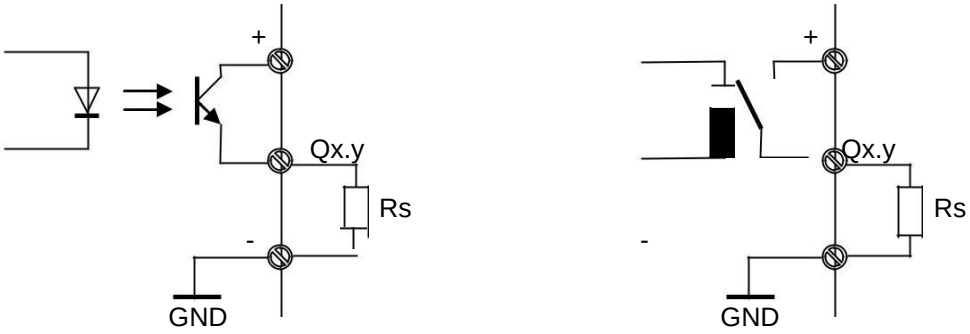


Figura 9.3. Conectarea ieşirilor unui automat programabil; a) ieşire prin tranzistor, b) ieşire prin releu.

La ambele tipuri de ieşiri, sarcina reprezentată de actuator[16] se conectează la borna de ieşire propriu-zisă Qx.y şi la borna – (GND). Conectarea ieşirii, deci a sarcinii (pentru efectuarea comenzii din proces), la borna + se face în interiorul automatului programabil prin deschiderea tranzistorului la polarizarea bazei respectiv la alimentarea bobinei releului. Alimentarea cu curent continuu poate fi făcută şi în acest caz cu o sursă externă sau internă.

[16] Elementul de execuţie

Conectarea intrărilor şi ieşirilor fizice ale automatelor programabile la un proces de automatizare

Pe baza documentaţiei pusă la dispoziţie de firma producătoare şi cea rezultată în urma proiectării, intrările din proces provenite de la contacte de relee, limitatoare, contactoare şi de la senzori (de proximitate, de presiune, etc.) se leagă la intrările corespunzătoare ale automatului programabil. De asemenea ieşirile către proces se conectează la actuatorii corespunzători (bobine de relee, contactoare, becuri de semnalizare, etc.).

Dacă automatul nu dispune de sursă de alimentare în curent continuu, din exterior se leagă o astfel de sursă la bornele corespunzătoare. Conectarea sursei de alimentare, 230Vca, se face la bornele L (faza) şi N (nulul de lucru). În Figura 9.4 se prezintă un exemplu de conectare a unui automat programabil cu 8 intrări şi 8 ieşiri la un proces.

Din cele 8 intrări sunt utilizate doar 6 (3 legate la contacte iar 3 la senzori din proces). De semenea doar 6 ieşiri din 8 sunt utilizate (4 pentru comanda bobinelor de contactoare şi 2 pentru aprinderea lămpilor de semnalizare).

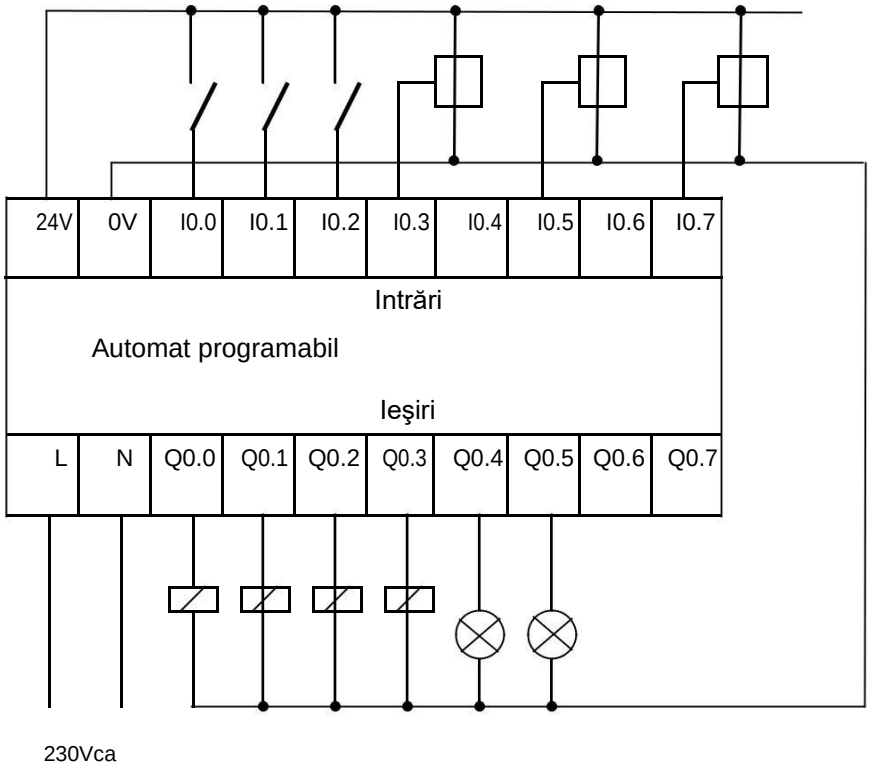


Figura 9.4. Exemplu de conectare a unui automat programabil la un proces.

5.6. **Programarea AP[17] utilizând limbajele STL, LAD şi FBD.**

Aplicaţie:

Pentru circuitul din Figura 10.1 care converteşte codul octal în caractere alfabetice afişate pe 7 segmente, conform tabelului de adevăr 10.1, să se realizeze programul în limbajul:

a) STL;
a) LAD;
b) FBD.

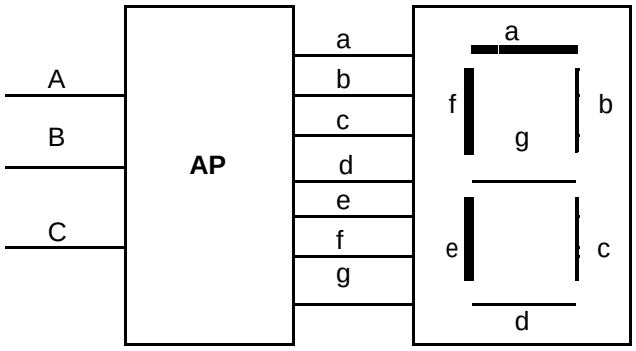


Figura 10.1. Convertor octal – alfabetic

Tabelul 10.1. Tabelul de adevăr al convertorului octal-alfabetic

Cifra octală	Triada binară	Caracter afișat	a	b	c	d	e	f	g
0	000	A	1	1	1	0	1	1	1
1	001	C	1	0	0	1	1	1	0
2	010	E	1	0	0	1	1	1	1
3	011	F	1	0	0	0	1	1	1
4	100	H	0	1	1	0	1	1	1
5	101	J	0	1	1	1	0	0	0
6	110	L	0	0	0	1	1	1	0
7	111	P	1	1	0	0	1	1	1

Funcţiile corespunzătoare celor 7 ieşiri, obţinute prin minimizare (recomandabil cu ajutorul diagramelor Veitch-Karnaugh) sunt:

$a = AB + \overline{C}$

$b = \overline{A} \cdot \overline{B} + AC$

[17] Automate programabile

$$c = \overline{A} \cdot \overline{B} + \overline{B}C$$

$$d = \overline{A}B + A\overline{B} = A \oplus B$$

$$e = f = \overline{A} + B + \overline{C}$$

$$g = \overline{A} \cdot \overline{B} + AB + \overline{B}C = \overline{A \oplus B} + \overline{B}C$$

10.1. Programarea în limbajul STL a AP ce emulează circuitul convertor octal-alfabetic.

În Tabelul 10.2. sunt prezentate asocierile dintre variabilele de intrare ale automatului şi intrările în circuit, respectiv dintre variabilele de ieşire şi ieşirile circuitului.

Tabelul 10.2 Asocierea variabile intrare/ ieşire AP şi intrări/ ieşiri circuit convertor

Intrări convertor	Intrări AP
A	%I0.0
B	%I0.1
C	%I0.2
Ieşiri convertor	Ieşiri AP
a	%Q0.0
b	%Q0.1
c	%Q0.2
d	%Q0.3
e	%Q0.4
f	%Q0.5
g	%Q0.6

Programul STL pentru emulator este următorul:

```
        LDN     %I0.2   //încarcă negatul intrării C în acumulator
        OR      (%I0.0
                AND %I.01
                )               //Execută operaţiile logice AB+C'
        ST      %Q0.0   //Stochează rezultatul în a
//----------------------------------------------------
        LDN     %I0.0   //încarcă negatul intrării A în acumulator
        ANDN    %I0.1   //Execută AND între acumulator şi B'
        OR      (%I0.0
                AND     %I0.2
                )               //Execută operaţiile logice A'B'+AC
        ST      %Q0.1   //Stochează rezultatul în b
//----------------------------------------------------
        LDN     %I0.0   //încarcă negatul intrării A în acumulator
        ANDN    %I0.1   //Execută AND între acumulator şi B'
        OR      (%I0.2
                ANDN    %I0.1
                )               //Execută operaţiile logice A'B'+B'C
        ST      %Q0.2   //Stochează rezultatul în c
//----------------------------------------------------
        LD      %I0.0
        XOR     %I0.1   //Execută operaţia A XOR B
        ST      %Q0.3   //Stochează rezultatul în d
```

```
//---------------------------------------------------
      LDN   %I0.0   //Încarcă A' în acumulator
      OR    %I0.1   //Execută operaţia A'+B
      ORN   %I0.2   //Execută operaţia A'+B+C'
      ST    %Q0.4   //Stochează rezultatul în e si f
      ST    %Q0.5
//---------------------------------------------------
      LDN   %I0.0
      ANDN  %I0.1   //În acumulator se află A'B'
      OR    (%I0.0
            AND %I0.1
            )               //În acumulator se află A'B'+AB
      OR    (%I0.1
            ANDN  %I0.2
            )               //În acumulator se află A'B'+AB+BC'
      ST    %Q0.6   //Stochează rezultatul în g
```

Programarea în limbajul LAD a AP ce emulează circuitul convertor octal-alfabetic.

Cu asocierile din Tabelul 10.2 în Figura 10.2.se prezintă programul în limbajul LAD.

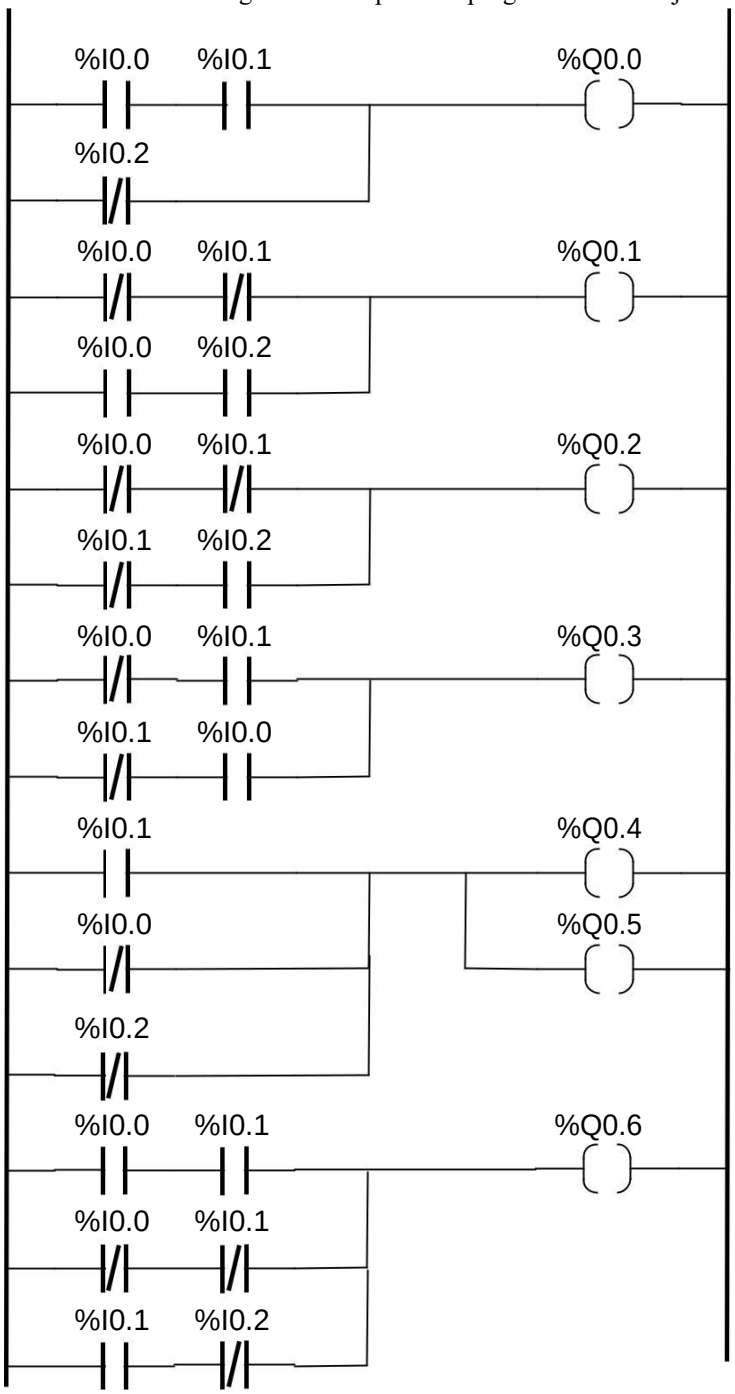


Figura 10.2. Programul LAD pentru AP ce emulează convertorul octal-alfabetic.

Programarea în limbajul FBD a AP ce emulează circuitul convertor octal-alfabetic.

Cu asocierile din Tabelul 10.2 programul pentru emulatorul convertorului octal-alfabetic în limbajul FBD este prezentat în Figura 10.3.a. şi b.

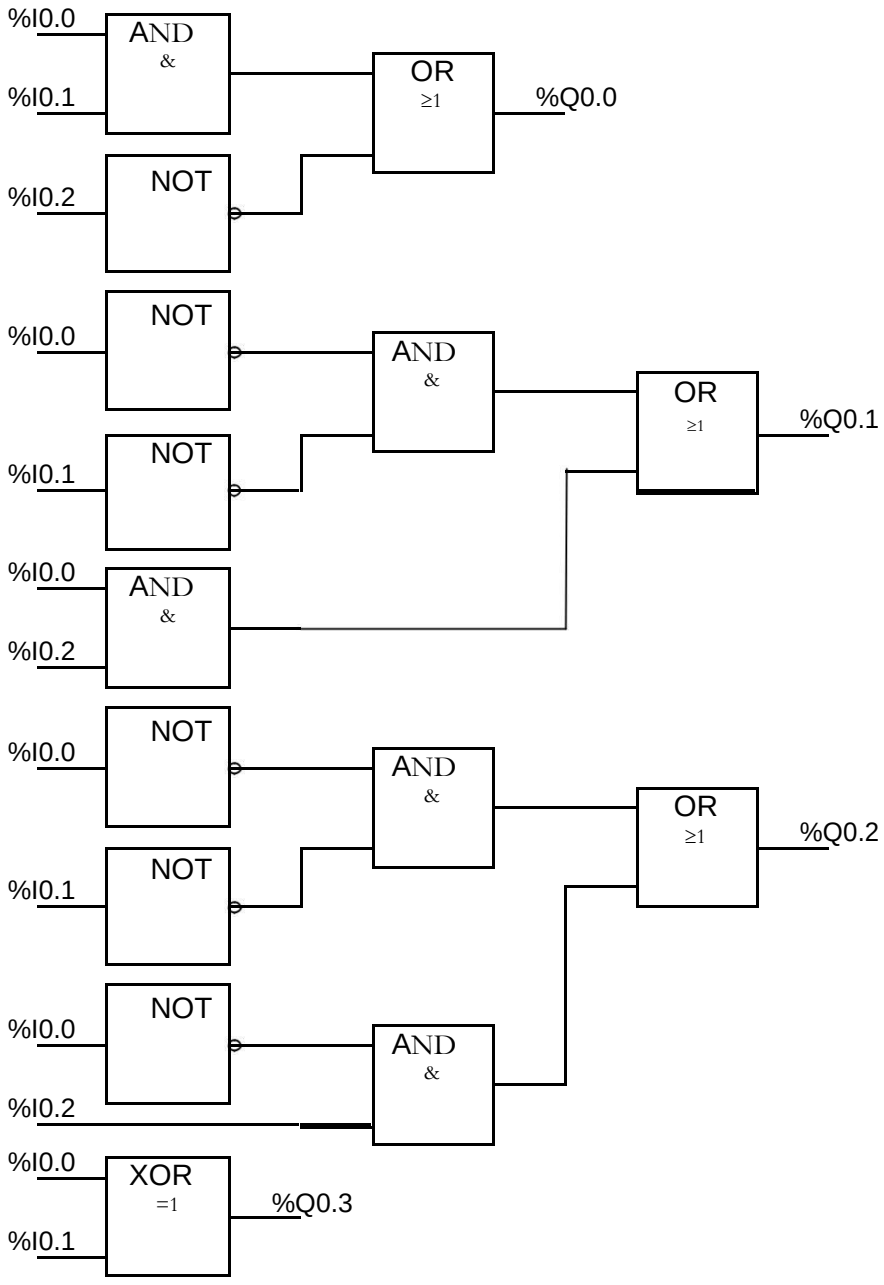


Figura 10.3.a. Programul în limbajului FBD al emulatorului circuitului convertor de cod octal-alfabetic.

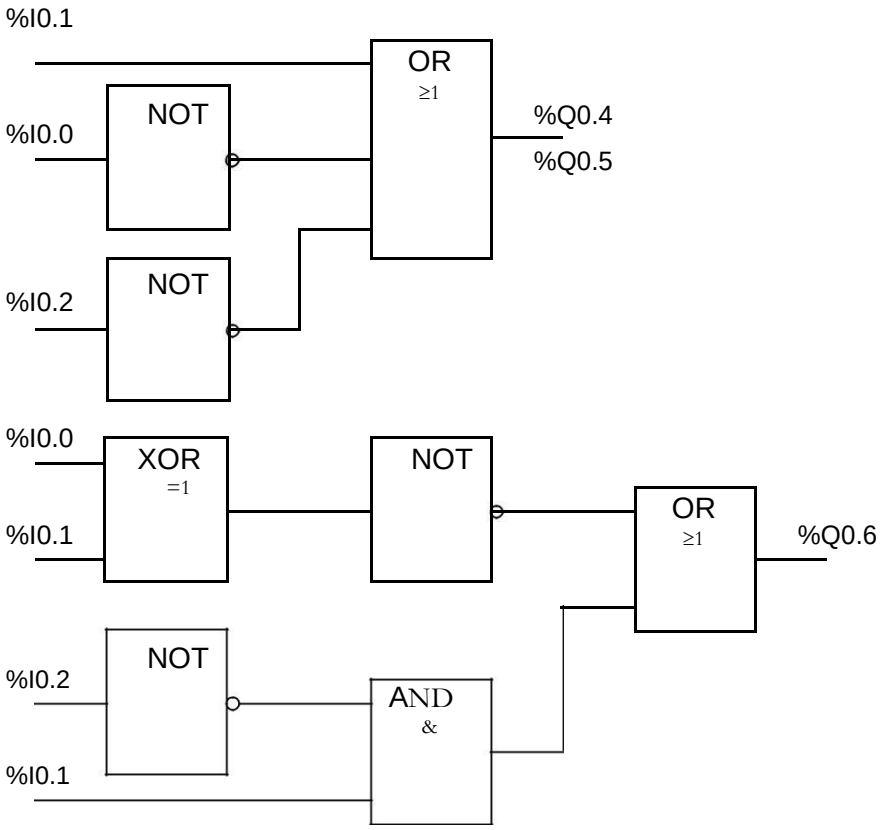


Figura 10.3.b. Programul în limbajului FBD al emulatorului circuitului convertor de cod octal-alfabetic.